DATABASE MANAGEMENT APPLICATIONS

CLASS XII

STUDENT HANDBOOK

CENTRAL BOARD OF SECONDARY EDUCATION

Shiksha Kendra, 2, Community Centre, Preet Vihar, Delhi-110301

•BUILDING SKILLS•BUILDING FUTURE•

DATABASE

MANAGEMENT APPLICATIONS

CLASS
XII

STUDENT
HANDBOOK

CENTRAL BOARD OF SECONDARY EDUCATION
Shiksha Kendra, 2, Community Centre, Preet Vihar, Delhi-110301

Database Management Applications
Student Handbook, Class-XII

Price: ₹

First Edition: February 2016, CBSE

Copies:

Paper Used: 80 gsm CBSE Water Mark White Maplitho

"This book or part thereof may not be reproduced by any person or agency in any manner."

Published By : The Secretary, Central Board of Secondary Education, Shiksha Kendra, 2, Community Centre, Preet Vihar, Delhi-110301

Design & Layout : Multi Graphics, 8A/101, WEA Karol Bagh, New Delhi-110005
Phone: 011-25783846, 47503846

Printed By :

भारत का संविधान

उद्देशिका

हम, भारत के लोग, भारत को एक सम्पूर्ण [1]प्रभुत्व-संपन्न समाजवादी पंथनिरपेक्ष लोकतंत्रात्मक गणराज्य बनाने के लिए, तथा उसके समस्त नागरिकों को:

सामाजिक, आर्थिक और राजनैतिक न्याय,
विचार, अभिव्यक्ति, विश्वास, धर्म
और उपासना की स्वतंत्रता,
प्रतिष्ठा और अवसर की समता
प्राप्त कराने के लिए
तथा उन सब में व्यक्ति की गरिमा
[2]और राष्ट्र की एकता और अखंडता
सुनिश्चित करने वाली बंधुता बढ़ाने के लिए

दृढ़संकल्प होकर अपनी इस संविधान सभा में आज तारीख 26 नवम्बर, 1949 ई॰ को एतद्द्वारा इस संविधान को अंगीकृत, अधिनियमित और आत्मार्पित करते हैं।

---

1. संविधान (बयालीसवां संशोधन) अधिनियम, 1976 की धारा 2 द्वारा (3.1.1977) से "प्रभुत्व-संपन्न लोकतंत्रात्मक गणराज्य" के स्थान पर प्रतिस्थापित।
2. संविधान (बयालीसवां संशोधन) अधिनियम, 1976 की धारा 2 द्वारा (3.1.1977) से "राष्ट्र की एकता" के स्थान पर प्रतिस्थापित।

भाग 4 क

मूल कर्त्तव्य

51 क. मूल कर्त्तव्य – भारत के प्रत्येक नागरिक का यह कर्त्तव्य होगा कि वह –

(क) संविधान का पालन करे और उसके आदर्शों, संस्थाओं, राष्ट्रध्वज और राष्ट्रगान का आदर करे;

(ख) स्वतंत्रता के लिए हमारे राष्ट्रीय आंदोलन को प्रेरित करने वाले उच्च आदर्शों को हृदय में संजोए रखे और उनका पालन करे;

(ग) भारत की प्रभुता, एकता और अखंडता की रक्षा करे और उसे अक्षुण्ण रखे;

(घ) देश की रक्षा करे और आह्वान किए जाने पर राष्ट्र की सेवा करे;

(ङ) भारत के सभी लोगों में समरसता और समान भ्रातृत्व की भावना का निर्माण करे जो धर्म, भाषा और प्रदेश या वर्ग पर आधारित सभी भेदभाव से परे हों, ऐसी प्रथाओं का त्याग करे जो स्त्रियों के सम्मान के विरुद्ध हैं;

(च) हमारी सामासिक संस्कृति की गौरवशाली परंपरा का महत्त्व समझे और उसका परिरक्षण करे;

(छ) प्राकृतिक पर्यावरण की जिसके अंतर्गत वन, झील, नदी, और वन्य जीव हैं, रक्षा करे और उसका संवर्धन करे तथा प्राणी मात्र के प्रति दयाभाव रखे;

(ज) वैज्ञानिक दृष्टिकोण, मानववाद और ज्ञानार्जन तथा सुधार की भावना का विकास करे;

(झ) सार्वजनिक संपत्ति को सुरक्षित रखे और हिंसा से दूर रहे;

(ञ) व्यक्तिगत और सामूहिक गतिविधियों के सभी क्षेत्रों में उत्कर्ष की ओर बढ़ने का सतत प्रयास करे जिससे राष्ट्र निरंतर बढ़ते हुए प्रयत्न और उपलब्धि की नई उंचाइयों को छू ले;

[1](ट) यदि माता-पिता या संरक्षक है, छह वर्ष से चौदह वर्ष तक की आयु वाले अपने, यथास्थिति, बालक या प्रतिपाल्य के लिये शिक्षा के अवसर प्रदान करे।

---

1. संविधान (छयासीवां संशोधन) अधिनियम, 2002 की धारा 4 द्वारा प्रतिस्थापित।

THE CONSTITUTION OF INDIA

PREAMBLE

WE, THE PEOPLE OF INDIA, having solemnly resolved to constitute India into a [1]**SOVEREIGN SOCIALIST SECULAR DEMOCRATIC REPUBLIC** and to secure to all its citizens :

JUSTICE, social, economic and political;

LIBERTY of thought, expression, belief, faith and worship;

EQUALITY of status and of opportunity; and to promote among them all

FRATERNITY assuring the dignity of the individual and the[2] unity and integrity of the Nation;

IN OUR CONSTITUENT ASSEMBLY this twenty-sixth day of November, 1949, do **HEREBY ADOPT, ENACT AND GIVE TO OURSELVES THIS CONSTITUTION.**

---

1. Subs, by the Constitution (Forty-Second Amendment) Act. 1976, sec. 2, for "Sovereign Democratic Republic" (w.e.f. 3.1.1977)
2. Subs, by the Constitution (Forty-Second Amendment) Act. 1976, sec. 2, for "unity of the Nation" (w.e.f. 3.1.1977)

THE CONSTITUTION OF INDIA

Chapter IV A

FUNDAMENTAL DUTIES

ARTICLE 51A

Fundamental Duties - It shall be the duty of every citizen of India-

(a) to abide by the Constitution and respect its ideals and institutions, the National Flag and the National Anthem;

(b) to cherish and follow the noble ideals which inspired our national struggle for freedom;

(c) to uphold and protect the sovereignty, unity and integrity of India;

(d) to defend the country and render national service when called upon to do so;

(e) to promote harmony and the spirit of common brotherhood amongst all the people of India transcending religious, linguistic and regional or sectional diversities; to renounce practices derogatory to the dignity of women;

(f) to value and preserve the rich heritage of our composite culture;

(g) to protect and improve the natural environment including forests, lakes, rivers, wild life and to have compassion for living creatures;

(h) to develop the scientific temper, humanism and the spirit of inquiry and reform;

(i) to safeguard public property and to abjure violence;

(j) to strive towards excellence in all spheres of individual and collective activity so that the nation constantly rises to higher levels of endeavour and achievement;

[1](k) who is a parent or guardian to provide opportunities for education to his/her child or, as the case may be, ward between age of 6 and 14 years.

---

1. Subs. by the Constitution (Eighty - Sixth Amendment) Act, 2002

Preface

In an increasingly globalised world and the changing paradigm of urbanized living the demand for Information Technology (IT) has been increased manifold throughout the world. In this ever expanding sector, it has become essential to provide competency based Vocational Education. It is in this context that CBSE has launched a course in Information Technology under NSQF from level 1 to 4.

The Student Handbook on "Database Management Applications" is a part of qualification package developed for the implementation of National Skill Qualification Framework (NSQF), an initiative of Government of India to set common principles and guidelines for a nationally recognised qualification system covering the schools, vocational education and training institutions, technical education institutions, college and universities. It is envisaged that the NSQF will promote transparency of qualifications, cross-sectoral learning, student qualifications, thus encouraging life-long learning.

It has been a deliberate effort to keep the language used in this Student Handbook as simple as possible for the benefit of the students. Necessary pictorial illustrations and tables have been included to help the students to understand the concepts without any difficulty.

Practicing professionals from the field of Information Technology (IT) comprised the team of authors for this book. I hope this book will help the students to serve as useful resource in this subject.

This Student Handbook attempts to bring about necessary flexibility in offering of courses, necessary for breaking the sharp boundaries between different subject areas. The Handbook attempt to enhance these endeavour by giving higher priority and space to opportunities for contemplation and wondering, discussion in small groups and activities requiring hands-on-experience.

The Board is grateful to the members of the Committee of Course for their advice, guidance and total commitment towards development of this course. We are indeed indebted to these academic advisors who have lent us the benefit of their rich and insightful experience. I would like to appreciate Vocational Education Cell, CBSE for coordinating and successfully completing the work.

Any suggestions, feedback from the readers for improvement in the future editions of the volume shall be heartily welcomed.

Chairman, CBSE

Acknowledgements

ADVISORS

Sh. Y.S.K. Seshu Kumar, Chairman, CBSE

Sh. K.K. Choudhury, Controller of Examinations & Director (V.E.), CBSE

MATERIAL PRODUCTION GROUP

Authors

- **Mr. Naveen Kumar, Associate Professor**
  Department of Computer Science
  Delhi University - Convener
- **Ms. Anita Goel, Associate Professor**
  Computer Science Department
  Dyal Singh College, University of Delhi
- **Ms. Hema Banati, Associate Professor**
  Computer Science Department
  Dyal Singh College, University of Delhi
- **Ms. Rakhi Saxena, Assistant Professor**
  Computer Science Department
  Deshbandhu College, University of Delhi
- **Ms. Sheetal Taneja, Assistant Professor**
  Computer Science Department
  Dyal Singh College, University of Delhi
- **Ms. Shikha Badhani, Assistant Professor**
  Computer Science Department
  Maitreyi College, University of Delhi

EDITING & COORDINATION

Dr. Biswajit Saha, Additional Director, (V.E.), CBSE

Content

Unit - 1: Database Concepts 1-47

1.1 Introduction 1

1.2 Basic Concepts and Definitions 1

1.3 Need for a Database 2

1.4 Database Management System (DBMS) 2

1.4.1 Characteristics of Database Management Systems 3

1.4.2 Types of users of DBMS 4

1.4.3 Advantages of using DBMS Approach 4

1.4.4 Limitations of using DBMS Approach 4

1.5 Relational Database 5

1.5.1 Relational Model Constraints 7

1.6 Structured Query Language (SQL) 9

Unit - 2: Operating Web 48-80

2.1 Operating Web-based Applications 48

2.2 Online Reservation Systems 48

2.2.1 Advantages of Online Reservation System 48

2.2.2 Precautions while Performing Online Transactions 50

2.2.3 Using Online Reservation Systems 51

2.2.4 Case Study: Book Rail Ticket 51

2.3 E-Governance 56

2.3.1 Initiative 56

2.3.2 E-Governance Sites 56

2.4 Online Shopping and Bill Payments 59

2.4.1 Benefits of Online Shopping 59

2.4.2 How it works 60

2.4.3 Bill Payments 61

2.4.4 Case Study: Online Shopping using Online Shopping Website 61

2.5 Online Courses, Tutorials and Tests 65

2.5.1 Online Educational Sites 65

2.6 Project Management - Web Based Application Development 69

2.6.1 Project Essentials and Tips 69

2.6.2 Case Study - Online Game 71
2.6.3 Case Study - Online Quiz 73
2.6.4 Case Study - Online Bill Calculator 76

Unit - 3: Fundamentals of Java Programming 81-143

3.1 Introduction to Java 81
3.2 Data Types and Variables 89
3.2.1 Variables 89
3.2.2 Primitive Data Types 93
3.2.3 String Variables 94
3.3 Operators 94
3.4 Control Flow 96
3.4.1 Selection Structures 97
3.4.2 The if Else Statement 97
3.4.3 The Switch Statement 100
3.4.4 Repetition Structures 101
3.4.5 The While Statement 102
3.4.6 The Do While Statement 103
3.4.7 The for Statement 106
3.5 Arrays 108
3.6 User Defined Methods 111
3.7 Object Oriented Programming 113
3.8 Class Design 114
3.8.1 Constructors 115
3.8.2 Access Modifiers 118
3.8.3 Getter and Setter Methods 119
3.9 Java Libraries 120
3.9.1 Data Input 121
3.9.2 Array Manipulation 122
3.9.3 String Manipulation 124
3.10 Exception Handling 127
3.11 Database Connectivity 129
3.11.1 Connecting to the MySQL Server in NetBeans 129
3.11.2 Adding the MySQL Connector JAR to the NetBeans Libraries 133
3.11.3 Database Connection from Java 134

3.12 Assertions, Threads, and Wrapper Classes 136
3.12.1 Assertions 136
3.12.2 Threads 138
3.12.3 Wrapper Classes 141

Unit - 4: Work Integrated Learning IT - DMA 144-157

4.1 Introduction 144
4.2 Identification of Potential Work Areas 144
4.3 A Shopping Website - A Case Study 146
4.3.1 Entities Involved 147
4.3.2 Functionality 150

Appendix - A 158-164

Installing and Staring NetBeans IDE 158

Unit - 1: Database Concepts

1.1 Introduction

The key to organizational success is effective decision making which requires timely, relevant and accurate information. Hence information plays a critical role in today's competitive environment. Database Management Software (DBMS) simplifies the task of managing the data and extracting useful information out of it. In this chapter, we shall learn about the basic concepts of databases and also learn how to use DBMS for some applications.

1.2 Basic Concepts and Definitions

Data is a collection of raw facts which have not been processed to reveal useful information. Information is produced by processing data as shown in Figure 1.2(a).

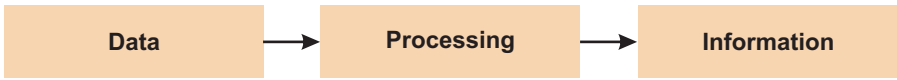


Figure 1.2(a): Data Processing

For example, given the data of the test marks of all the students in a class (data), the average, maximum and minimum marks in the class can be used as indicators of the performance of the class (information). In other words, we can say that we have extracted the information about average, maximum and minimum marks for given student data in Figure 1.2(b).

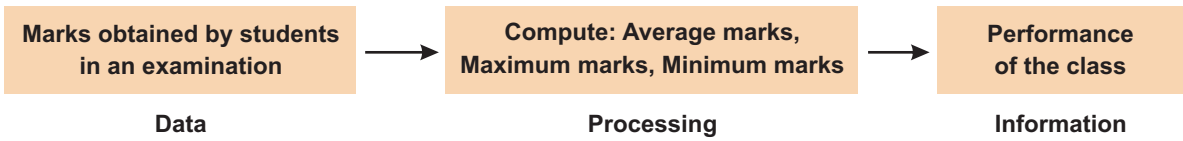


Figure 1.2(b): Example of Data Processing

Databases are being used extensively in our day-to-day life. Be it business, engineering, medicine, education, library, to name a few. For example, consider the name, class, roll number, marks in every subject of every student in a school. To record this information about every student in a school, the school might have maintained a register, or stored it on a hard drive using a computer system and software such as a spreadsheet or DBMS package. Such collection of related data that has been recorded, organized and made available for searching is called a Database.

A database has the following properties:

1) A database is a representation of some aspect of the real world also called miniworld. Whenever there are changes in this miniworld they are also reflected in the database.
2) It is designed, built and populated with data for specific purpose.
3) It can be of any size and complexity.
4) It can be maintained manually or it may be computerized.

1.3 Need for a Database

In traditional file processing, data is stored in the form of files. A number of application programs are written by programmers to insert, delete, modify and retrieve data from these files. New application programs will be added to the system as the need arises. For example, consider the Sales and Payroll departments of a company. One user will maintain information about all the salespersons in the Sales department in some file say File1 and another user will maintain details about the payroll of the salesperson in a separate file say File2 in the Payroll Department as shown in Figure 1.3(a).

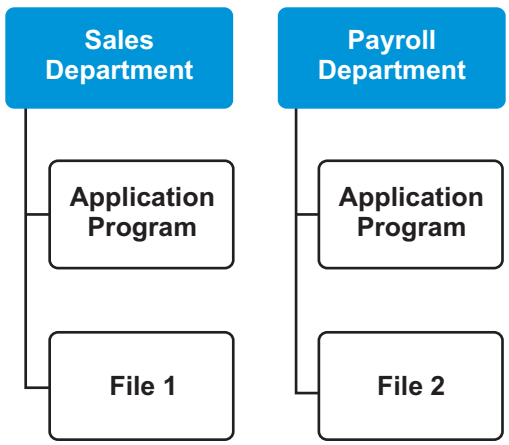


Figure 1.3(a): Traditional File Processing System

Although both the departments need information about the salesperson but they will store information about the salesperson in different files and will use different application programs to access those files. This would result in:

1. **Data Redundancy:** Same information is stored in more than one file. This would result in wastage of space.
2. **Data Inconsistency:** If a file is updated then all the files containing similar information must be updated else it would result in inconsistency of data.
3. **Lack of Data Integration:** As data files are independent, accessing information out of multiple files becomes very difficult.

Database approach overcomes these problems and also adds a lot of advantages as discussed later. In database approach, a single repository of data is maintained which is accessed by different users as per their needs.

1.4 Database Management System (DBMS)

A database management system is a collection of programs that enables users to create, maintain and use a database. It enables creation of a repository of data that is defined once and then accessed by different users as per their requirements. Thus there is a single repository of data which is accessed by all the application programs as shown below Figure 1.4(a).

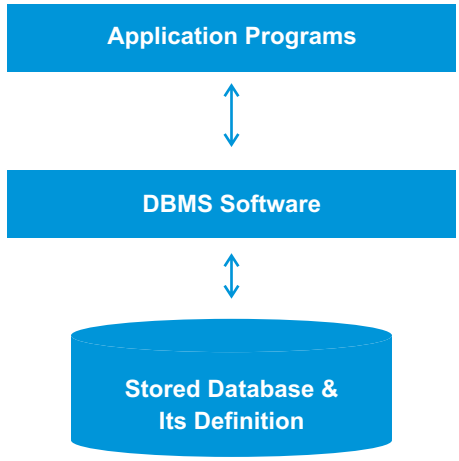


Figure 1.4(a): DBMS Environment

The various operations that need to be performed on a database are as follows:

1. **Defining the Database:** It involves specifying the data type of data that will be stored in the database and also any constraints on that data.
2. **Populating the Database:** It involves storing the data on some storage medium that is controlled by DBMS.
3. **Manipulating the Database:** It involves modifying the database, retrieving data or querying the database, generating reports from the database etc.
4. **Sharing the Database:** Allow multiple users to access the database at the same time.
5. **Protecting the Database:** It enables protection of the database from software/ hardware failures and unauthorized access.
6. **Maintaining the Database:** It is easy to adapt to the changing requirements.

Some examples of DBMS are – MySQL, Oracle, DB2, IMS, IDS etc.

1.4.1 Characteristics of Database Management Systems

The main characteristics of a DBMS are as follows:

1. **Self-describing Nature of a Database System:** DBMS contains not only the database but also the description of the data that it stores. This description of data is called meta-data. Meta-data is stored in a database catalogue or data dictionary. It contains the structure of the data and also the constraints that are imposed on the data.
2. **Insulation Between Programs and Data:** Since the definition of data is stored separately in a DBMS, any change in the structure of data would be done in the catalogue and hence programs which access this data need not be modified. This property is called Program-Data Independence.
3. **Sharing of Data:** A multiuser environment allows multiple users to access the database simultaneously. Thus a DBMS must include concurrency control software to allow simultaneous access of data in the database without any inconsistency problems.

1.4.2 Types of Users of DBMS

DBMS is used by many types of users depending on their requirements and interaction with the DBMS. There are mainly four types of users:

1. **End Users:** Users who use the database for querying, modifying and generating reports as per their needs. They are not concerned about the working and designing of the database. They simply use the DBMS to get their task done.
2. **Database Administrator (DBA):** As the name implies, the DBA administers the database and the DBMS. The DBA is responsible for authoring access, monitoring its use, providing technical support, acquiring software and hardware resources.
3. **Application Programmers:** Application programmes write application programs to interact with the database. These programs are written in high level languages and SQL to interact with the database.
4. **System Analyst:** System analyst determines the requirements of the end users and then develops specifications to meet these requirements. A system analyst plays a major role in the database design and all the technical, economic and feasibility aspects.

1.4.3 Advantages of using DBMS Approach

The need of DBMS itself explains the advantages of using a DBMS. Following are the advantages of using a DBMS:

1. **Reduction in Redundancy:** Data in a DBMS is more concise because of the central repository of data. All the data is stored at one place. There is no repetition of the same data. This also reduces the cost of storing data on hard disks or other memory devices.
2. **Improved Consistency:** The chances of data inconsistencies in a database are also reduced as there is a single copy of data that is accessed or updated by all the users.
3. **Improved Availability:** Same information is made available to different users. This helps sharing of information by various users of the database.
4. **Improved Security:** Though there is improvement in the availability of information to users, it may also be required to restrict the access to confidential information. By making use of passwords and controlling users' database access rights, the DBA can provide security to the database.
5. **User Friendly:** Using a DBMS, it becomes very easy to access, modify and delete data. It reduces the dependency of users on computer specialists to perform various data related operations in a DBMS because of its user friendly interface.

1.4.4 Limitations of using DBMS Approach

The two main disadvantages of using a DBMS:

1. **High Cost:** The cost of implementing a DBMS system is very high. It is also a very time-consuming process which involves analyzing user requirements, designing the database specifications, writing application programs and then also providing training.
2. **Security and Recovery Overheads:** Unauthorized access to a database can lead to

threat to the individual or organization depending on the data stored. Also the data must be regularly backed up to prevent its loss due to fire, earthquakes, etc.

Hence the DBMS approach is usually not preferred when the database is small, well defined, less frequently changed and used by few users.

1.5 Relational Database

Various types of databases have been developed. One of them was relational database developed by E.F Codd at IBM in 1970. It is used to organize collection of data as a collection of relations where each relation corresponds to a table of values. Each row in the table corresponds to a unique instance of data and each column name is used to interpret the meaning of that data in each row. For example, consider `EMPLOYEE` table in Figure 1.5(a). Each row in this table represents facts about a particular employee. The column names – `Name`, `Employee_ID`, `Gender`, `Salary` and `Date_of_Birth` specify how to interpret the data in each row.

EMPLOYEE

Name	Employee_ID	Gender	Salary	Date_of_Birth
Neha Mehta	1121	Female	20000	04-03-1990
Paras Bansal	2134	Male	25000	19-10-1993
Himani Verma	3145	Female	20000	23-11-1992

Figure 1.5(a): Employee Table

In relational model,

- A row is called a **Tuple**.
- A column is called an **Attribute**.
- A table is called as a **Relation**.
- The data type of values in each column is called the **Domain**.
- The number of attributes in a relation is called the **Degree** of a relation.
- The number of rows in a relation is called the **Cardinality** of a relation.
- **Relation Schema** R is denoted by $R(A_1, A_2, A_3, \ldots, A_n)$ where R is the relation name and $A_1, A_2, A_3, \ldots A_n$ is the list of attributes.
- **Relation State** is the set of tuples in the relation at a point in time. A relation state r of relation schema $R(A_1, A_2, \ldots, A_n)$, denoted r(R) is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$, where each **n-tuple** is an ordered list of values $t = <v_1, v_2, \ldots, v_n>$, where v_i is in domain of A_i or is `NULL`. Here n is the degree of the relation and m is the cardinality of the relation.

Hence in Figure 1.5(a),

- `EMPLOYEE` table is a relation.
- There are three tuples in `EMPLOYEE` relation.
- `Name, Employee_ID, Gender, Salary, Date_of_Birth` are attributes.
- The domain is a set of atomic (or indivisible) values. The domain of a database attribute is the set of all the possible values that attribute may contain. In order to specify a domain, we specify the data type of that attribute. Following are the domain of attributes of the `EMPLOYEE` relation:

  (a) Name – Set of character strings representing names of persons.

  (b) Employee_ID–Set of 4-digit numbers

  (c) Gender – male or female

  (d) Salary – Number

  (e) Date_of_Birth – Should have a valid date, month and year. The birth year of the employee must be greater than 1985. Also the format should be dd-mm-yyyy.
- The degree of the EMPLOYEE relation is 5 as there are five attributes in this relation.
- The cardinality of the EMPLOYEE relation is 3 as there are three tuples in this relation.
- Relation Schema – EMPLOYEE (Name, Employee_ID, Gender, Salary, Date_of_Birth)
- Relation State –{<Neha Mehta, 1121,Female,20000,04-03-1990>,

  <Paras Bansal, 2134, Male, 25000, 19-10-1993>,

  <Himani Verma, 3145, Female, 20000, 23-11-1992>}

Some More Characteristics of Relations:

1. Ordering of tuples is not important in a Relation. Thus, the following relation in Figure 1.5(b) is same as the relation in Figure 1.5(a).

EMPLOYEE

Name	Employee_ID	Gender	Salary	Date_of_Birth
Paras Bansal	2134	Male	25000	19-10-1993
Neha Mehta	1121	Female	20000	04-03-1990
Himani Verma	3145	Female	20000	23-11-1992

Figure 1.5(b): Employee Table (reordered tuples)

2. The ordering of attributes is also unimportant.
3. No two tuples of relation should be identical i.e. given any pair of two tuples, value in at least one column must be different.

4. The value in each tuple is an atomic value (indivisible).
5. If the value of an attribute in a tuple is not known or not applicable or not available, a special value called **null** is used to represent them.

For example consider the following cases:

- Unknown value: A person has a date of birth but it is not known at the time of data entry.
- Unavailable value: A person has a home phone but does not want it to be listed.
- Not applicable: College degree attribute would be **NULL** for a person who has no college degrees.

In all the above cases NULL value would be used.

A **Relational DBMS (RDBMS)** is a DBMS which is based on the relational model as discussed above. This is one of the most commonly used databases. Examples of RDBMS are Oracle, MySQL, IBM DB2.

1.5.1 Relational Model Constraints

Constraints, are restrictions on the values, stored in a database based on the requirements. For example, in the relation `EMPLOYEE`, the Employee_ID must be a 4-digit number, the Date_of_Birth must be such that the birth year > 1985.

We describe below various types of constraints in Relational model:

1. **Domain Constraint**: It specifies that the value of every attribute in each tuple must be from the domain of that attribute. For example, the Employee_ID must be a 4-digit number. Hence a value such as "12321" or "A234" violates the domain constraint as the former is not 4-digit long and the latter contains an alphabet.
2. **Key Constraint**: Before we can explain this constraint, we need to describe the terms superkey, key, candidate key and primary key.

   (i) *Superkey* is a set of attributes in a relation, for which no two tuples in a relation state have the same combination of values. Every relation must have at least one superkey which is the combination of all attributes in a relation. Thus for the `EMPLOYEE` relation, following are some of the superkeys:

   (a) {Name, Employee_ID, Gender, Salary, Date_of_birth} - default superkey consisting of all attributes.

   (b) {Name, Employee_ID, Date_of_Birth}

   (c) {Employee_ID, Gender, Salary}

   (d) {Name, Employee_ID, Gender}

   (e) {Employee_ID}

However {Gender, Salary} is not a superkey because both these attributes have identical values for employees Neha and Himani.

(ii) *Key* is the minimal superkey, which means it is that superkey of a relation from which if any attribute is removed then it no longer remains a superkey. For example the superkey {Name, Employee_ID, Gender}is not a key as we can remove Name and Gender from this combination and then what is left {Employee_ID} is still a Superkey. Now {Employee_ID} is a key as it is a superkey as well as no more removals are possible. A relation may have more than one key. Consider the relation PERSON with the following schema: `PERSON` (Aadhar_number, PAN, Voter_ID_cardno, Name, Date_of_birth, Address). This relation has three keys namely : {Aadhar_number}, {PAN}, {Voter_ID_no} as every individual in India has a unique Aadhar card number, PAN as well as Voter ID card number.

(iii) *Candidate key*: A key as described above is called candidate key of the relation. For example, the `PERSON` relation has three candidate keys as discussed above.

(iv) *Primary Key:* One of the candidate keys may be designated as Primary key. Primary key is used to identify tuples in a relation. If a relation has many candidate keys it is preferable to choose that one as primary key which has least number of attributes. Primary key are usually underlined in the schema of the relation. For example in the relation schema: PERSON (Aadhar_number, PAN, Voter_ID_cardno, Name, Date_of_birth, Address), Aadhar_number is the primary key.

The relation between superkey, key, candidate key and primary key can be explained with the help of Figure 1.5(c).

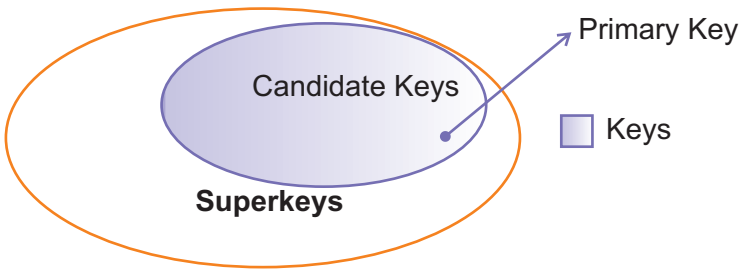


Figure 1.5(c): Superkey, Key, Candidate key and Primary key

3. **Null Value Constraint:** Sometimes it is required that certain attributes cannot have null values. For example, if every `EMPLOYEE` must have a valid name then the Name attribute is constrained to be `NOT NULL`.

4. **Entity Integrity Constraint:** This constraint specifies that primary key of a relation cannot have null value. The reason behind this constraint is that we know primary key contains no duplicates. However if we allow null values for a primary key then there can be multiple tuples for which primary key is having null values. This would imply that we are allowing duplicate values (`NULL`) for a primary key which itself violates the definition of primary key.

5. **Referential Integrity Constraint:** This constraint is specified between two relations. Before defining this constraint let us study the concept of foreign keys. *Foreign key* in a relation R1 is the set of attributes in R1 that refer to primary key in another relation R2 if the domain of foreign key attributes is same as that of primary key attributes and the value of foreign key either occurs as a value of primary key in some tuple of R2 or is NULL.

   R1 is called the referencing relation and R2 is called referenced relation, and a referential integrity constraint holds from R1 to R2.

   The main purpose of this constraint is to check that data entered in one relation is consistent with the data entered in another relation. For example, consider two relation schemas:

   Department (Dept_Name, Dept_ID, No_of_Teachers)

   Teacher (Teacher_Name, Teacher_ID, Dept_ID, Subject)

   Following are the primary keys (Underlined above):

   - Dept_ID is the primary key of Department relation.
   - Teacher_ID is the primary key of Teacher relation.

Now you may notice that Dept_ID– the primary key of relation in Department, is also present in relation Teacher. The reason is that every teacher belongs to a particular department. Now that means Dept_ID of Teacher relation must have a value that exists in Dept_ID attribute of Department relation or it can be NULL in case a teacher has not yet been assigned to a department. We say that Dept_ID of Teacher relation is a foreign key that references primary key of Department relation (Dept_ID).

It is important to emphasize it is not necessary to have same name for foreign key as of the corresponding referenced primary key. The above two schemas can also be written as follows:

Department (Dept_Name, Dept_ID, No_of_Teachers)

Teacher (Teacher_Name, Teacher_ID, Dept_No, Subject)

Where Dept_No is the foreign key that references Dept_ID of Department relation.

A foreign key may also refer to the same relation. For example suppose we have to create a database of all residents in a colony along with their best neighbors. Consider the following relation:

Residents (Name, RID, Block_no, House_no, Floor, Neighbor_RID)

The Primary key of this relation is RID (Resident ID). In order to store information about neighbor we have created a foreign key Neighbor_RID that references RID of Residents. Note that the referencing and referenced relation are same in this case.

1.6 Structured Query Language (SQL)

SQL is a language that is used to manage data stored in a RDBMS. It comprises of a Data Definition Language (DDL) and a Data Manipulation Language (DML) where DDL is a

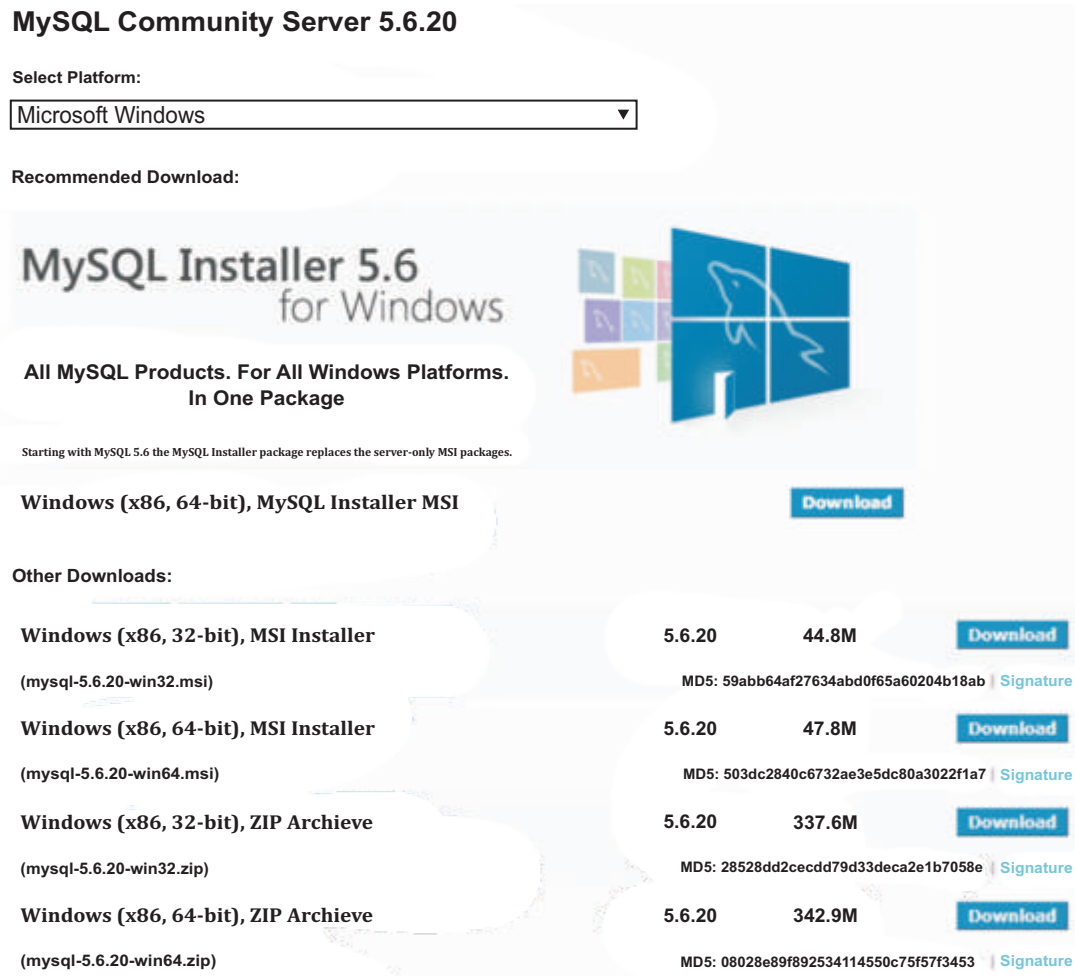
language which is used to define structure and constraints of data and DML is used to insert, modify and delete data in a database.

SQL commands are used to perform all the operations. In order to study SQL commands, a database system needs to be installed on the Computer. There are various softwares available. We will study the MySQL server.

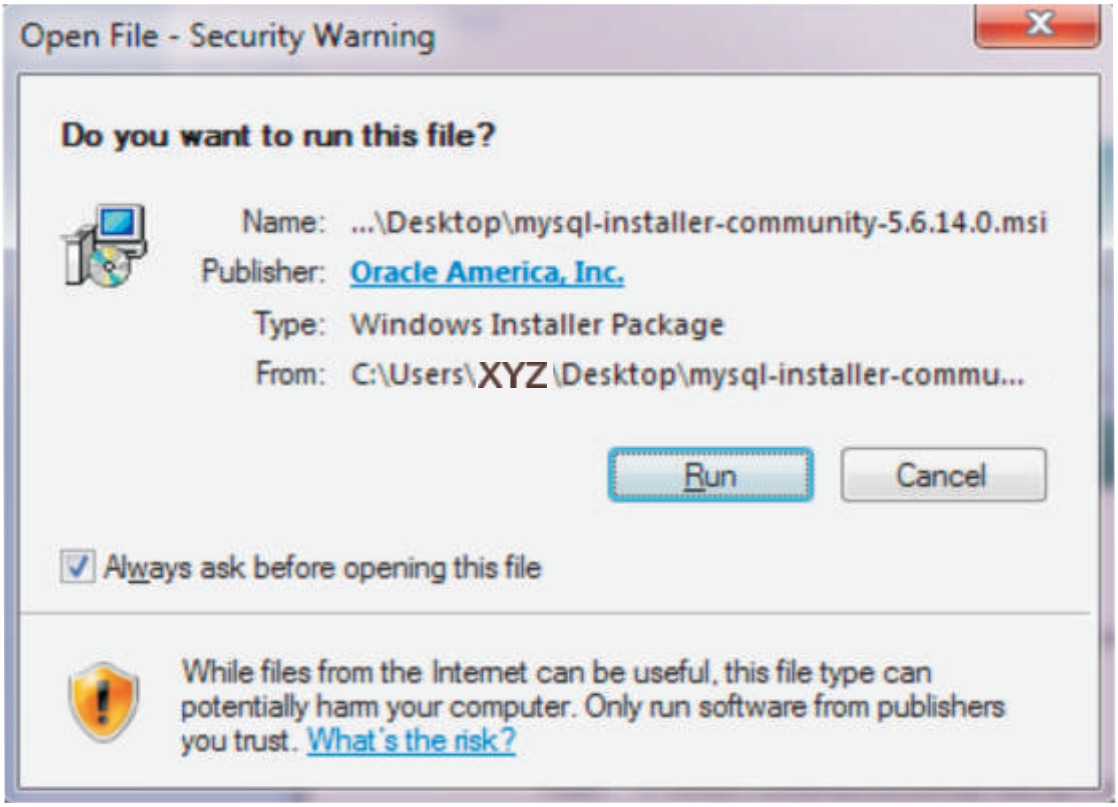
SQL uses the terms table, row and column for the relational model terms relation, tuple and attribute.

For studying SQL we will use MySQL Community Server 5.6.20 which is freely downloadable. The most recent versions can be found on the website: http://dev.mysql.com/downloads/ Following are the steps to install and configure MySQL Community Server 5.6.20 for studying SQL commands.

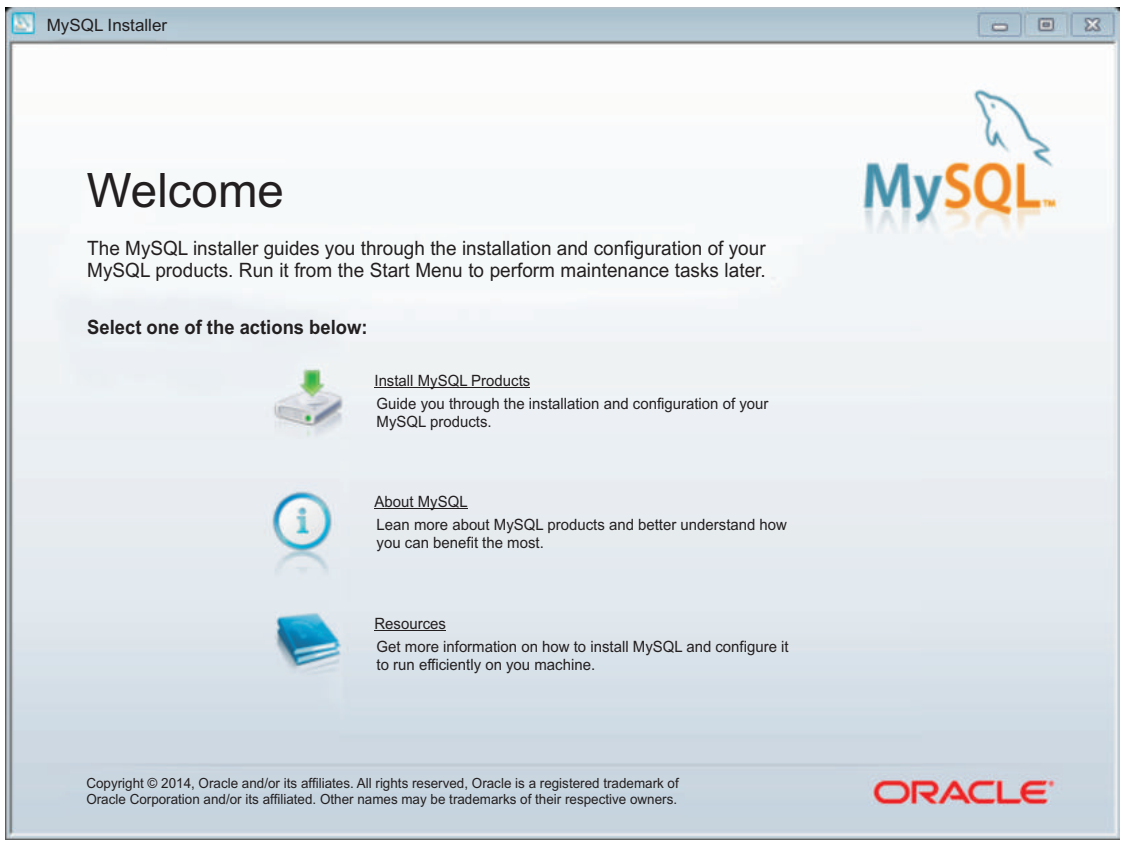
1. **Open the URL:** http://dev.mysql.com/downloads/mysql/#downloads
2. **Download the MySQL** Community Server 5.6.20 available on the above webpage. You can select the platform (Windows/Linux) as shown below.



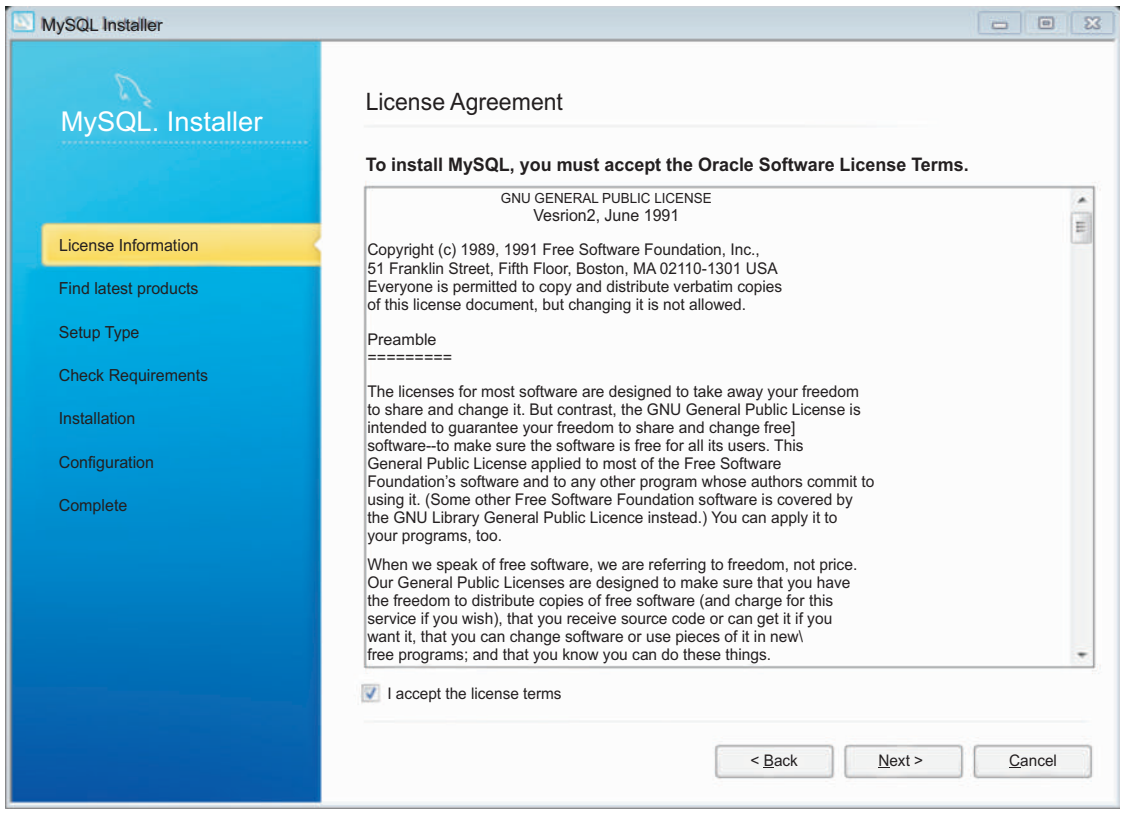
3. Once you have downloaded the file mysql-installer-community-5.6.20.0.msi, double click on the downloaded file and then click on the “Run” button.



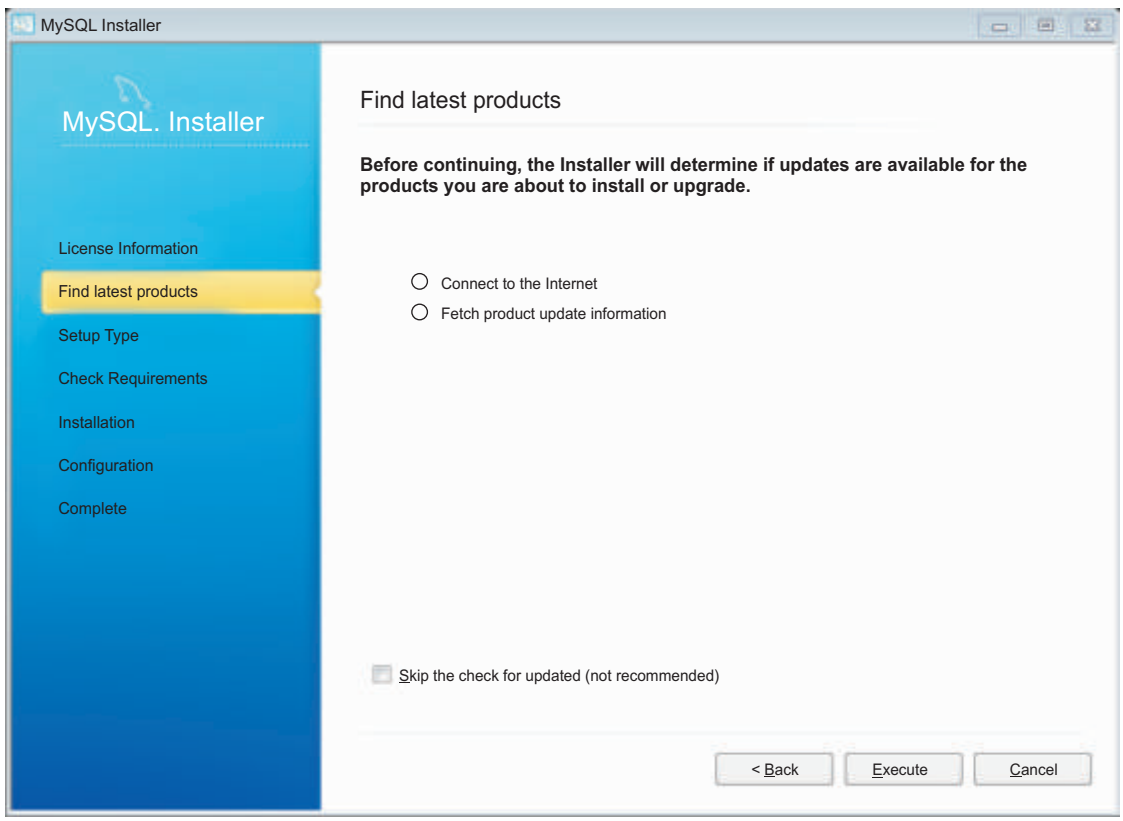
4. MySQL Installer will start installing. Click on the "Install MySQL Products" option.



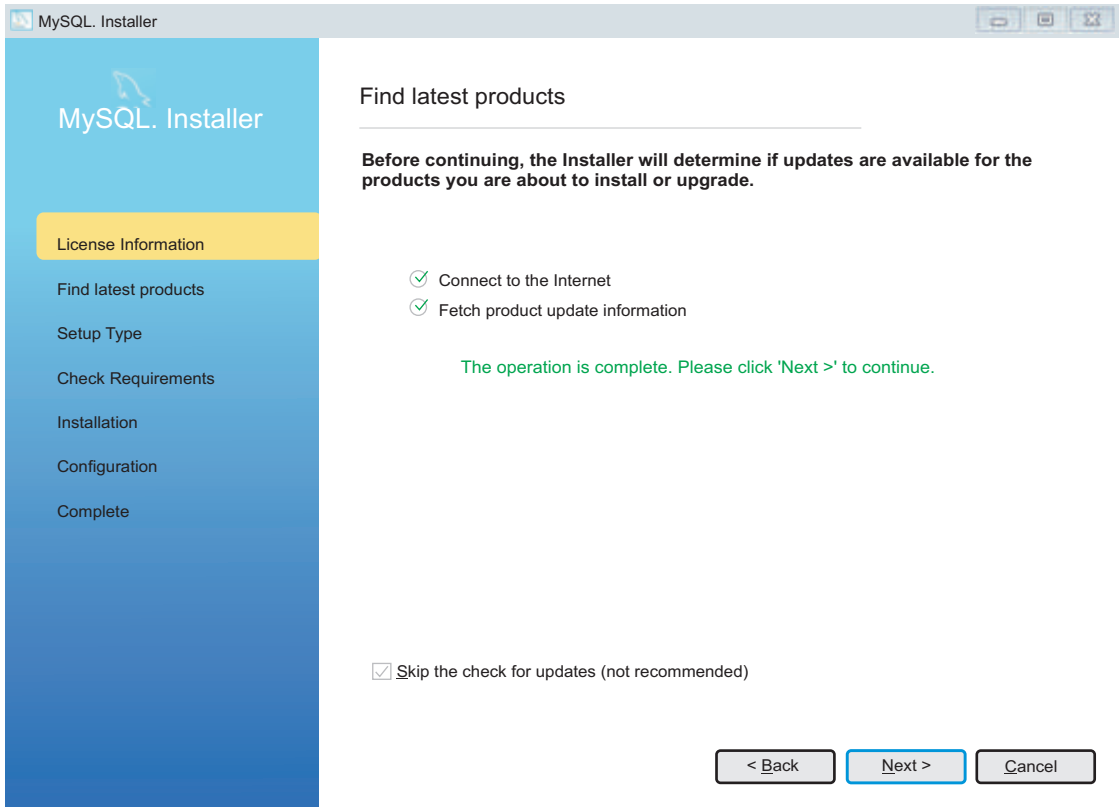
5. Check the option “I accept the license terms” and then Click on “Next” button.



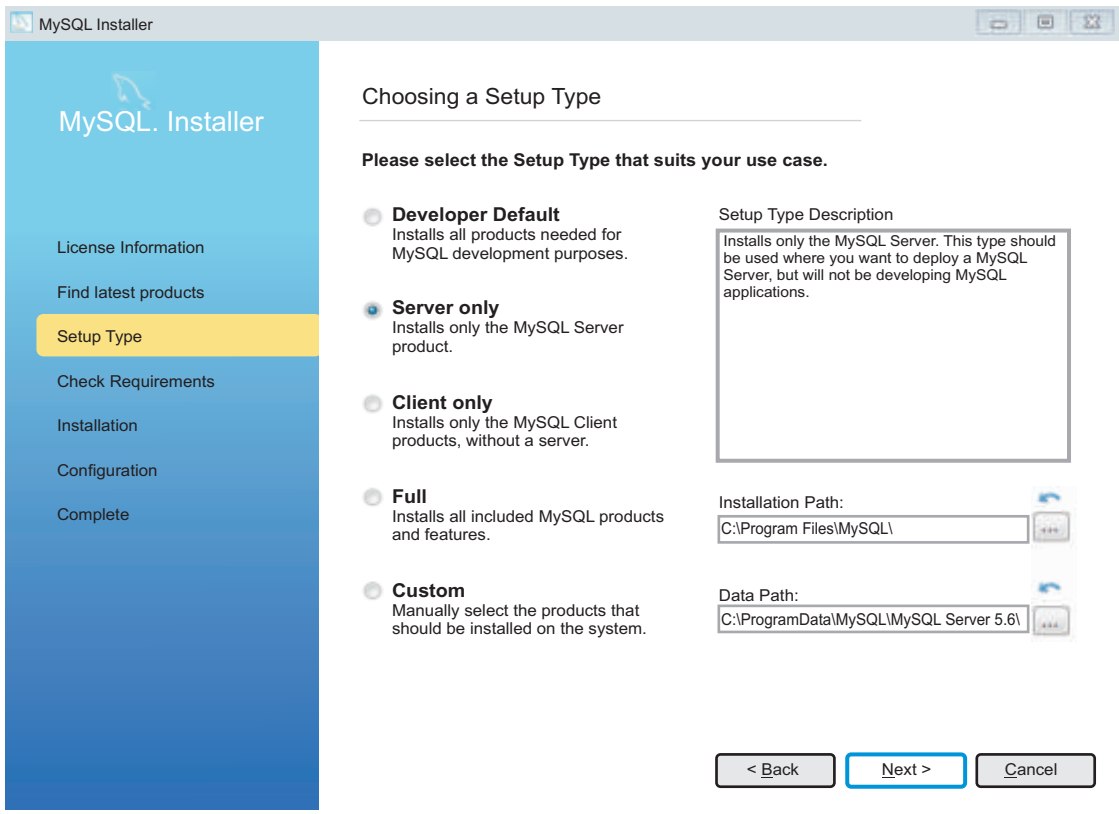
6. Then next click on the “Execute” button.



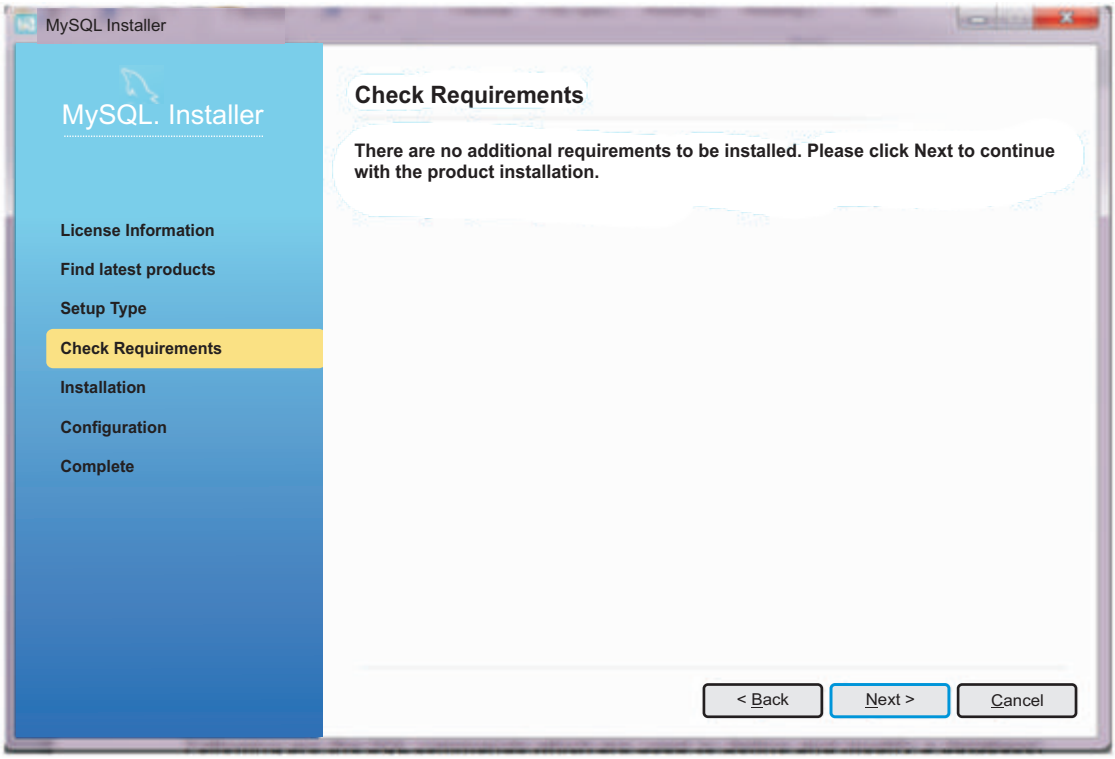
7. On successful execution, click on the “Next” button.



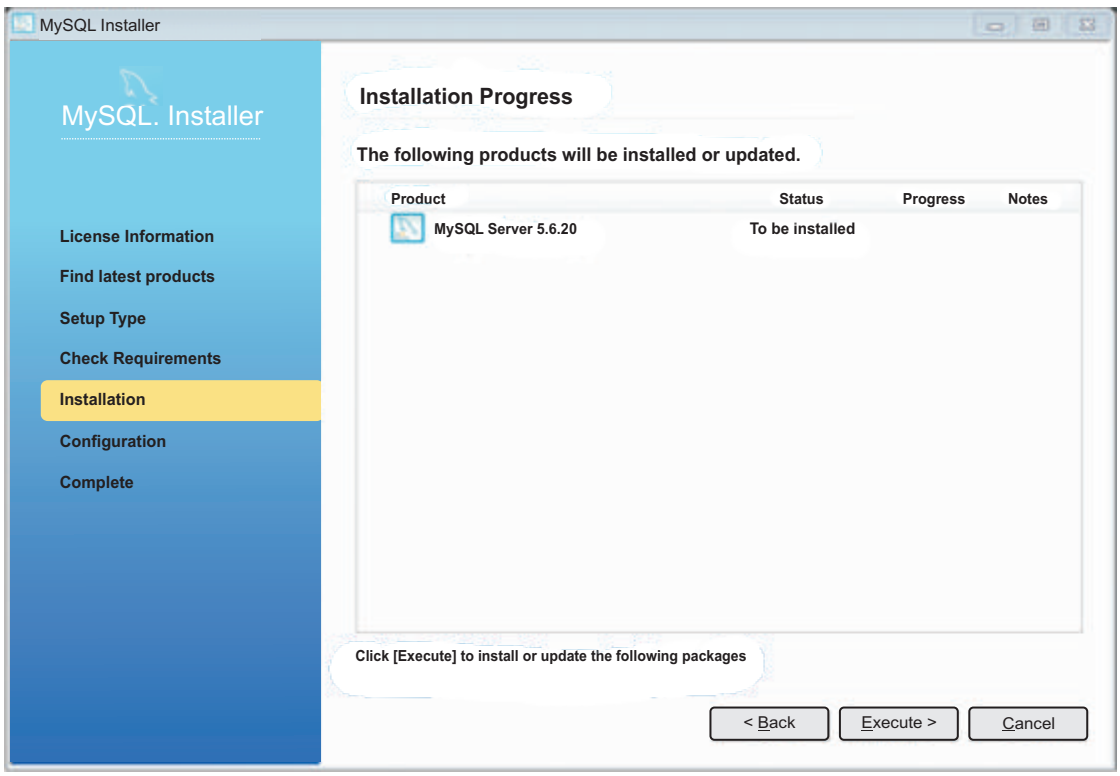
8. Select the “Server only” option. Then click on “Next” button.



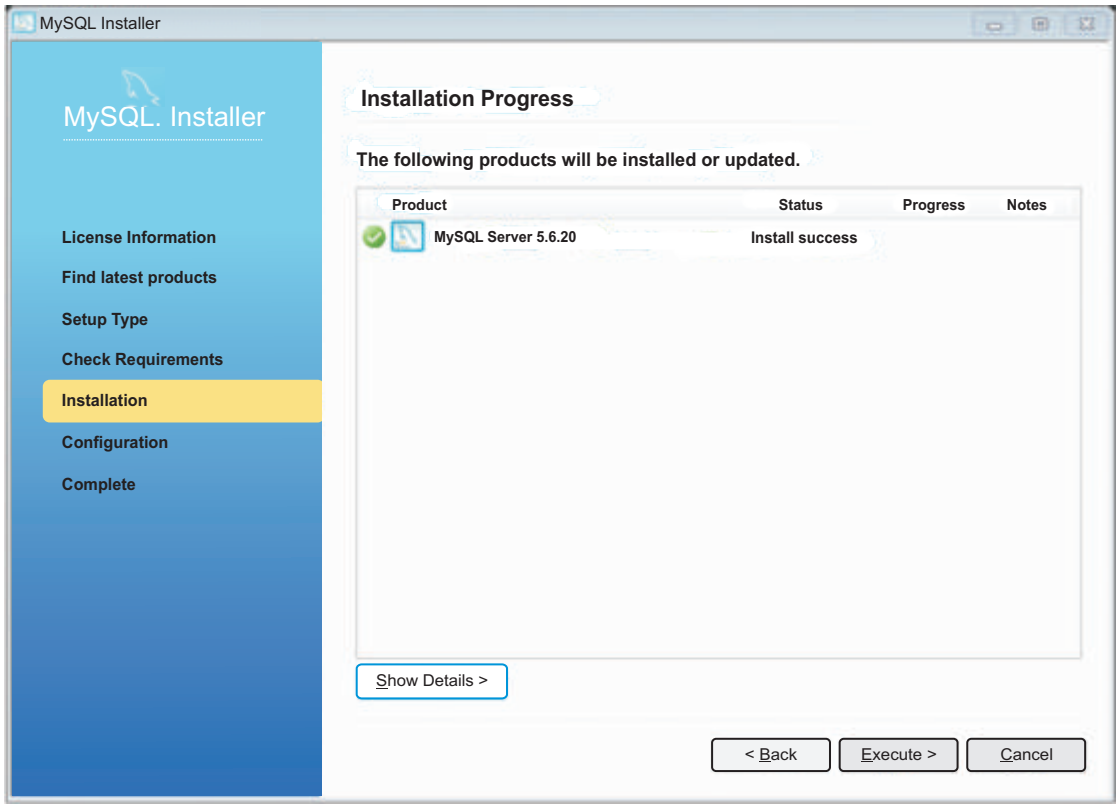
9. Installer will check for the requirements. If any requirements are required, you have to download them first before installing MySQL server. If all the requirements are met, then the following message will be displayed. Click “Next” to continue.



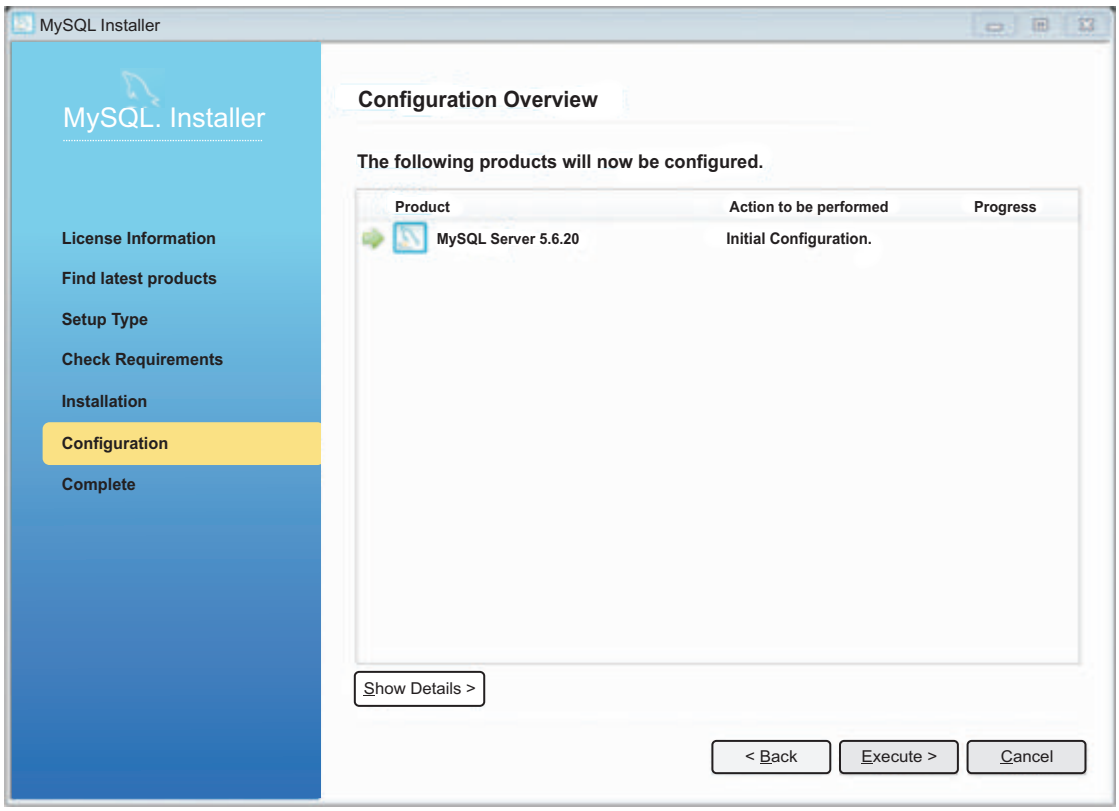
10. Click on “Execute” to install MySQL Server 5.6.20.



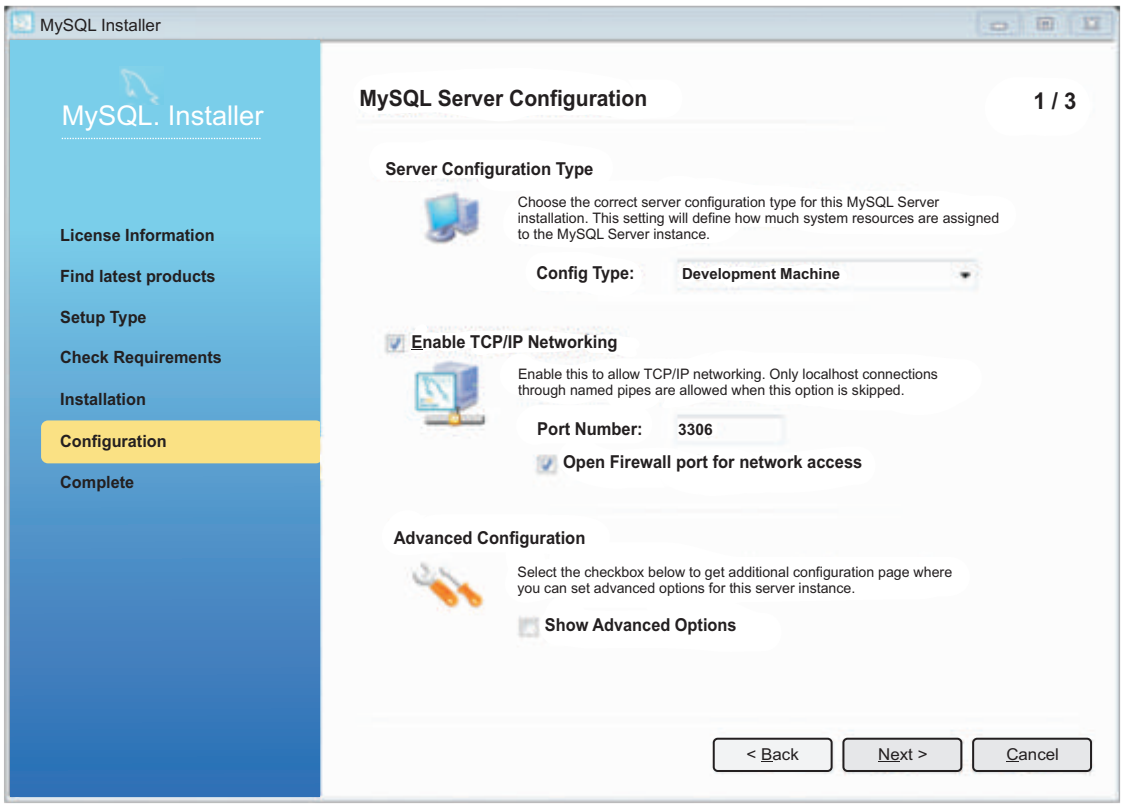
11. On successful installation, Click on “Next”



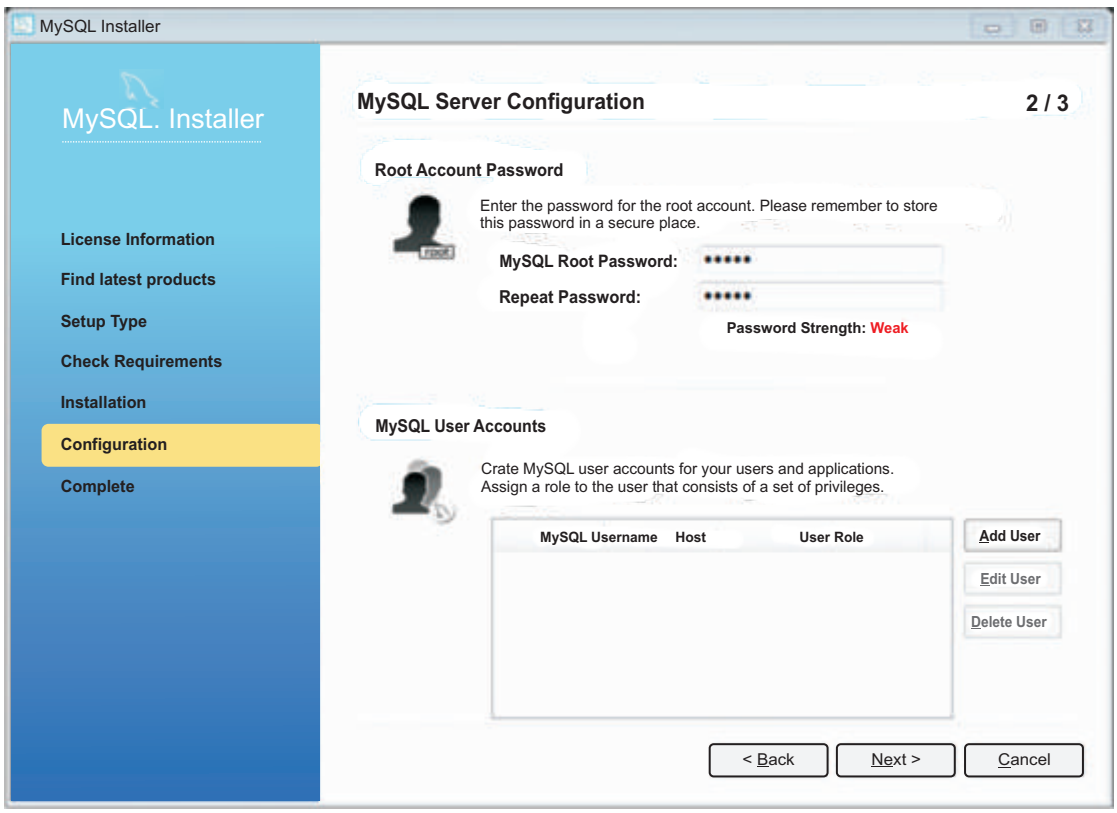
12. Click on “Next” to start initial configuration.



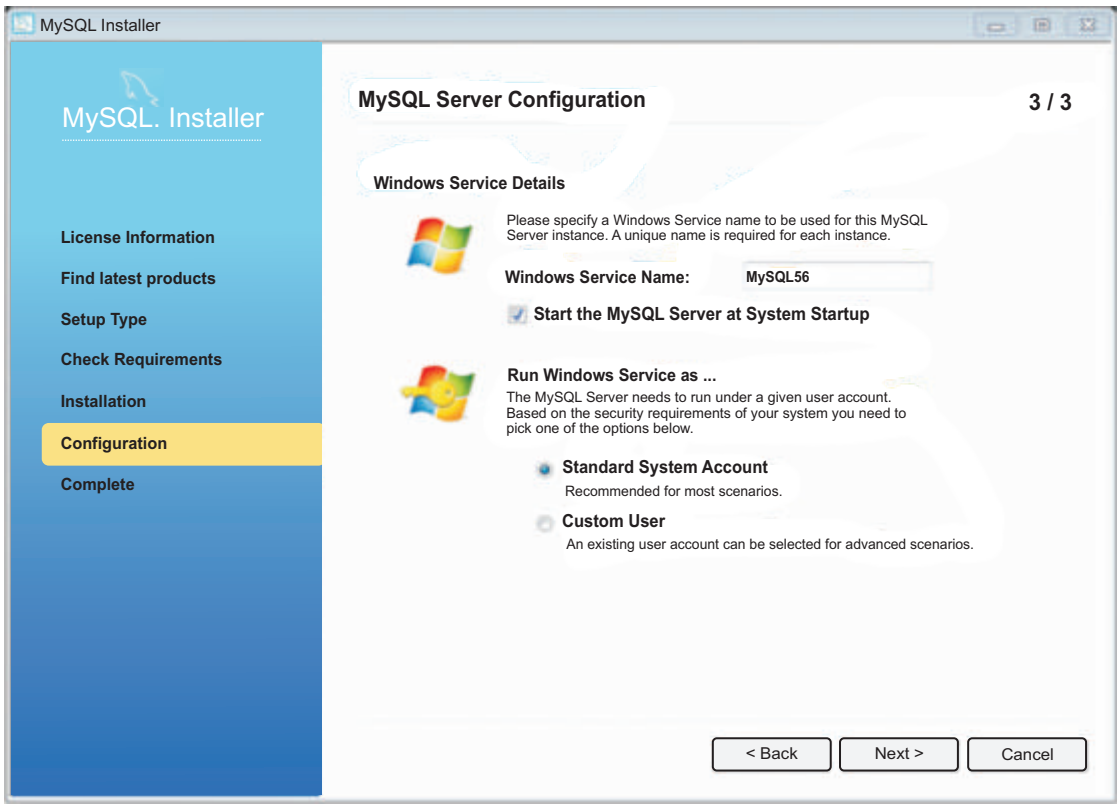
13. Select the following configurations and then Click "Next".



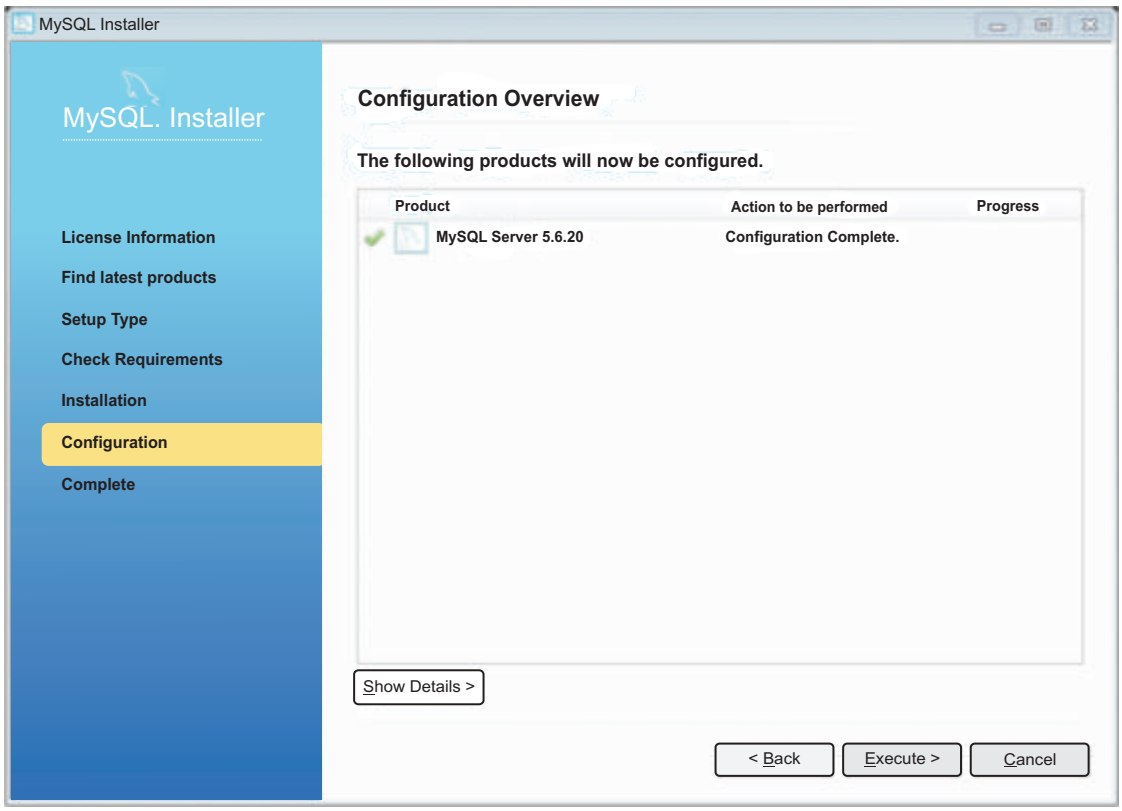
14. Type the MySQL root password (minimum 4 characters long) and then click "Next"



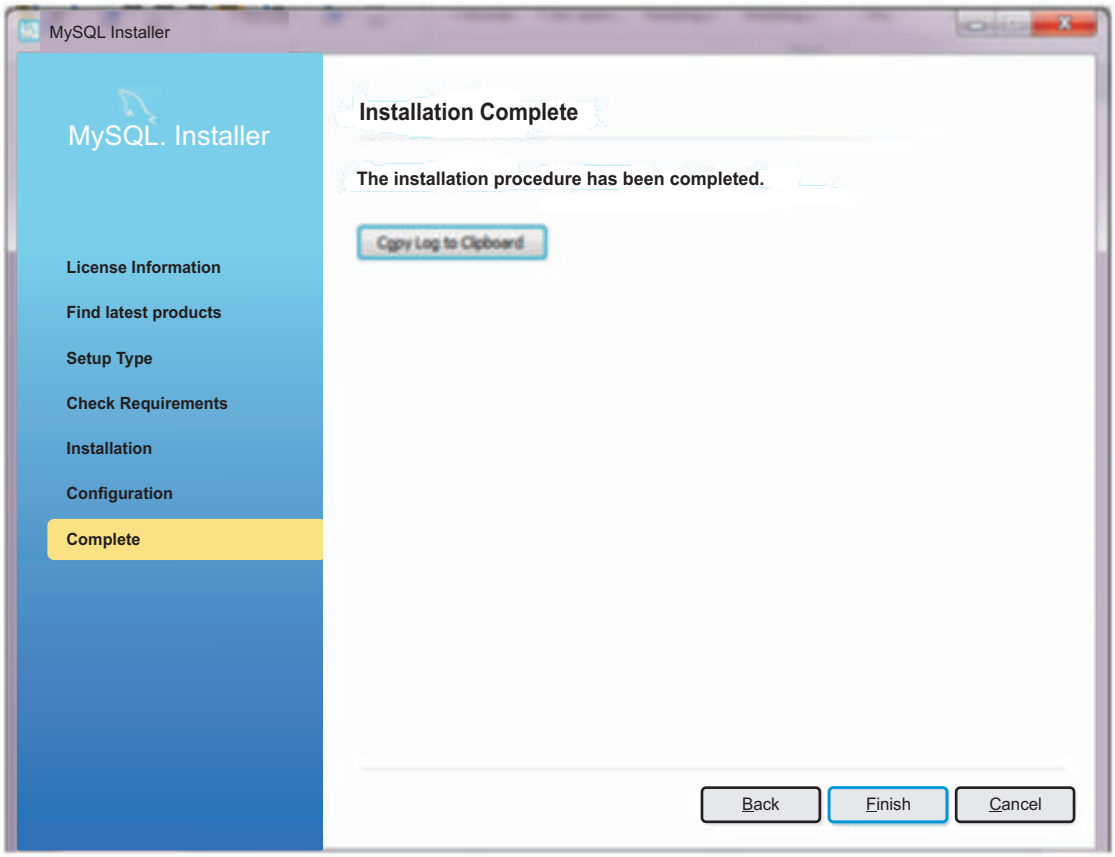
15. Click on “Next” on the following window.



16. Installer will configure the server. On successful configuration, click on “Next”.



17. Then Click on “Finish”. The installation and configuration of MySQL Server 5.6.20 is now complete. You can now start using the server for creating and modifying databases.



We give below SQL commands used to define and modify a database:

1. **Create Table Command:** This command is used to create a new table or relation. The syntax for this command is :

```
CREATE TABLE<table name>
(
<column 1><data type> [constraint] ,
<column 2><data type>[constraint],
<column 3><data type>[constraint]
);
```

where []=optional

The keyword `CREATE TABLE` is followed by the name of the table that you want to create. Then within parenthesis, you write the column definition that consists of column name followed by its data types and optional constraints. There can be as many columns as you require. Each column definition is separated with a comma (,). All SQL statements should end with a semicolon (;).

Table 1 shows the data types commonly used.

Data type	Meaning	Example
CHAR (n)	Fixed length character string. 'n' is the number of characters.	CHAR(5):“Ashok” “Vijay”
VARCHAR(n)	Variable length character string. 'n' is the maximum number of characters in the string.	VARCHAR(15): “Vijay Kumar” “Ashok Sen”
DATE	Date in the form of YYYY-MM-DD	DATE: '2014-03-20'
INTEGER	Integer number	23 56789
DECIMAL (m, d)	Fixed point number m represents the number of significant digits that are stored for values and d represents the number of digits that can be stored following the decimal point. If d is zero or not specified then the value does not contains any decimal part.	DECIMAL(5,2) : 999.99 -567.78 DECIMAL (5) : 23456 99999

Table 1: Commonly used data types

Suppose we wish to create a database of all the teachers working in a school. This database should include the Teacher relation (schema given below):

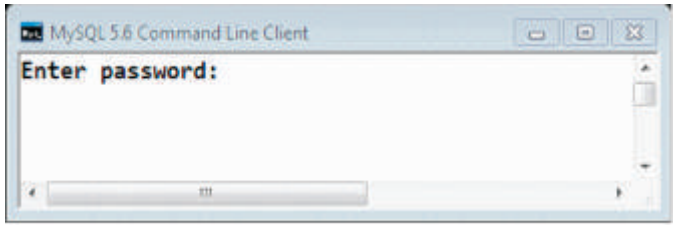
```
Teacher (Teacher_ID, First_Name, Last_Name, Gender, Date_of_Birth, Salary, Dept_No)
```

The above schema stores the information about all the teachers working in the school such as their unique ID, first and last name, gender, salary, date of birth and the department to which the teacher belongs.

To create the above relations in SQL, following CREATE TABLE command is used:

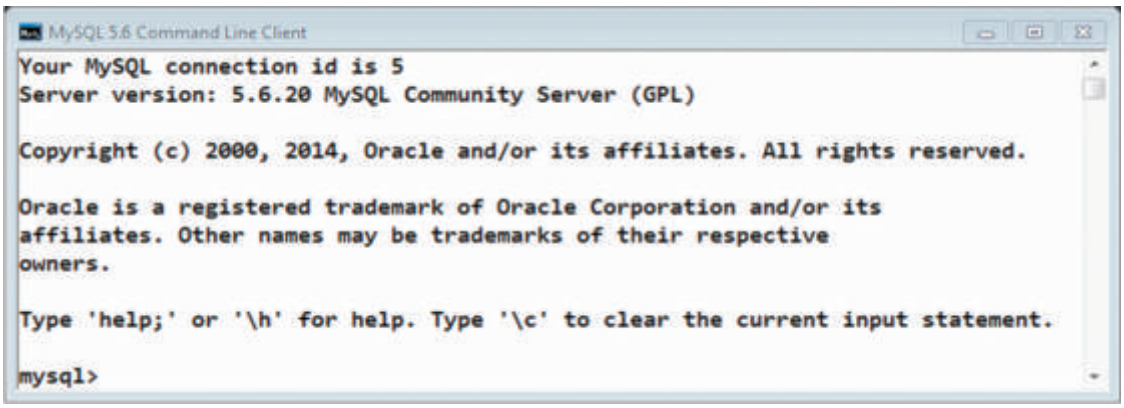
```
CREATE TABLE Teacher
(
Teacher_ID INTEGER,
First_Name VARCHAR(20),
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2),
Date_of_Birth DATE,
Dept_No INTEGER
);
```

To create the above table in MySQL, click on MySQL 5.6 Command Line Client in the Start Menu. Following command line window will open:



Enter the password as you have entered during the installation of MySQL community server 5.6.20.

After entering the password, you can see the MySQL monitor:

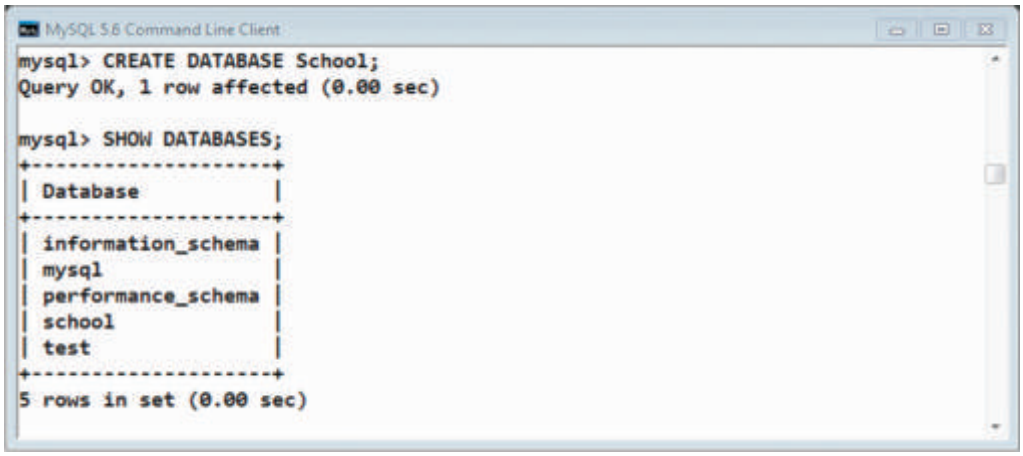


You can now type the SQL commands on the mysql> prompt.

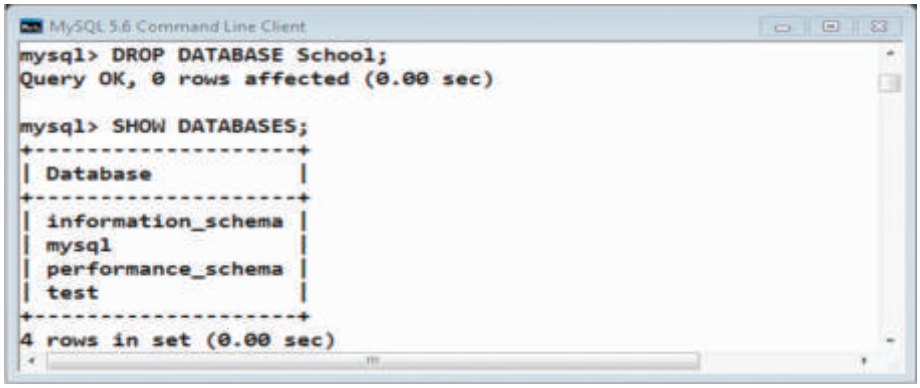
Now the next step is to create a database using the `CREATE DATABASE` command.

```
CREATE DATABASE School;
```

Once the database is created, you can check it in the list of databases that currently exist on the server by using the `SHOW DATABASES` command as shown below:

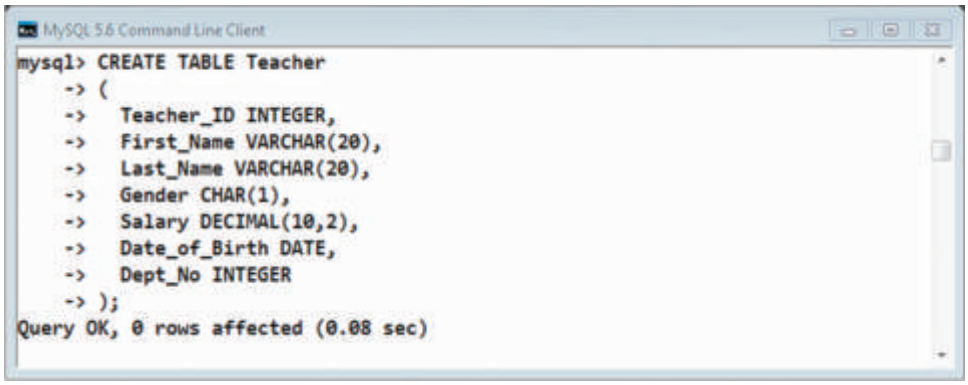


In case you want to remove a database, this can be done by using the `DROP DATABASE` command.

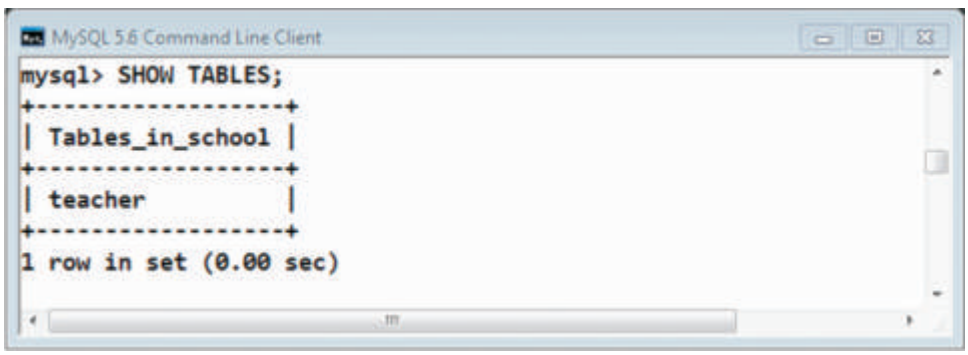


Next step is to tell the server which database we will use for further statements. This can be done by using the USE command.

Now you can create the Teacher table which would be associated with the School database.



To verify you can use SHOW TABLES command which displays all the tables created in the current database.



Database Constraints: DBMS can enforce several constraints for smooth operations on databases. These constraints can be specified while creating the table as shown below:

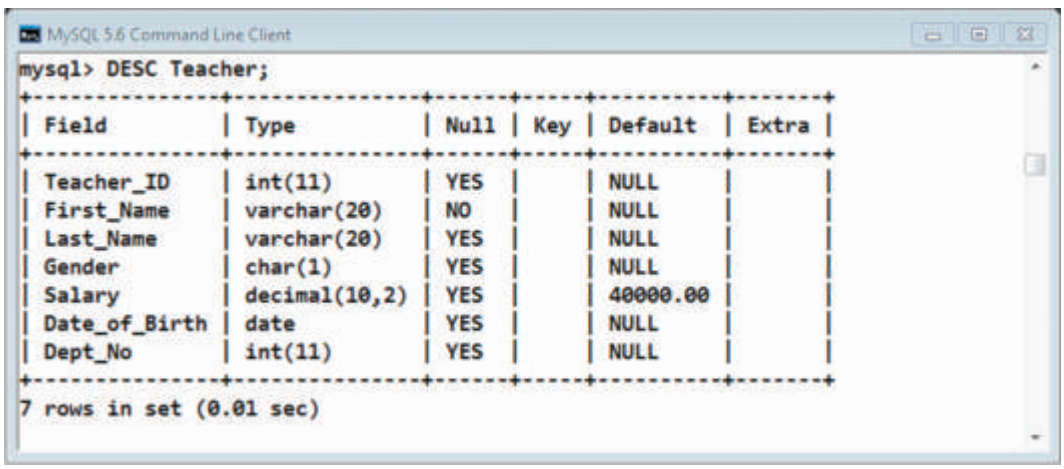
a) **NOT NULL:** An attribute value may not be permitted to be `NULL`. For example, the First name of the Teacher cannot be NULL. Hence `NOT NULL` constraint can be specified in this case.

```
CREATE TABLE TEACHER
(
Teacher_ID INTEGER,
First_NameVARCHAR(20) NOT NULL,
Last_NameVARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2),
Date_of_Birth DATE,
Dept_No INTEGER
);
```

b) **DEFAULT** : If a user has not entered a value for an attribute, then default value specified while creating the table is used. For example, if a teacher's salary has not been entered, then by default the database should store 40000 assuming that the minimum salary given to every teacher is ₹ 40000. This is illustrated as follows:

```
CREATE TABLE TEACHER
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER
);
```

In MySQL, if you want to look at the structure and description of the tables created, DESC command can be used. The description of the table Teacher created above is as follows:



c) **CHECK:** In order to restrict the values of an attribute within a range, `CHECK` constraint may be used. For example `Dept_No` of any teacher must not exceed 110. This can be specified as follows:

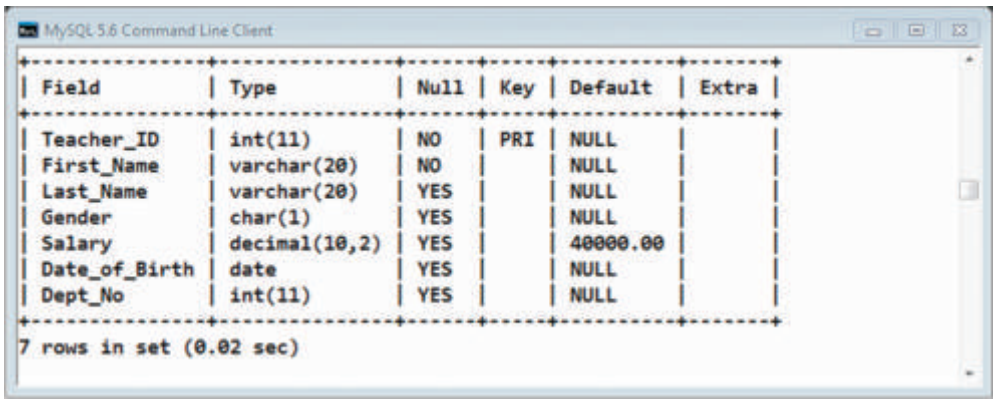
```
CREATE TABLE TEACHER
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER CHECK (Dept_No<=110)
);
```

You can also use the CHECK constraint to compare two attributes within the same relation having same data type.

> MySQL does not support CHECK constraint, although it will not give any error if you have a check constraint in your table.

d) **KEY CONSTRAINT:** Primary Key of a table can be specified in two ways. If the primary key of the table consist of a single attribute, then the corresponding attribute can be declared primary key along with its description. For example, if `Teacher_ID` attribute of the Teacher relation is the `PRIMARY KEY` then it can be specified as follows:

```
CREATE TABLE TEACHER
(
Teacher_ID INTEGER PRIMARY KEY,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER
);
```



As shown above First_Name has value NO in the Null column as it was specified to be NOT NULL in the Create table command. Further Teacher_ID being the primary key (PRI in the Key column) cannot be NULL. Also, note that default value is specified only for SALARY, and shown NULL for other attributes indicating no default value.

However if primary key contains more than one attribute then it must be specified separately as a list of attributes it comprises of, within parenthesis, separated by commas. For example, the primary key of the TEACHER relation comprises of Teacher_ID and Date_of_Birth.

```
CREATE TABLE TEACHER
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER,
PRIMARY KEY (Teacher_ID, Date_of_Birth)
);
```

By default, Primary keys are NOT NULL and there is no need to mention this constraint separately.

e) **REFERENTIAL INTEGRITY CONSTRAINT**- This constraint is specified by using the foreign key clause. This clause contains the foreign key and the primary key referred to by this foreign key along with the name of the relation. For example consider the following tables created in the School Database:

Department (Dept_ID, Dept_Name)

Teacher (Teacher_ID, First_Name, Last_Name, Gender, Salary, Date_of_Birth, Dept_No)

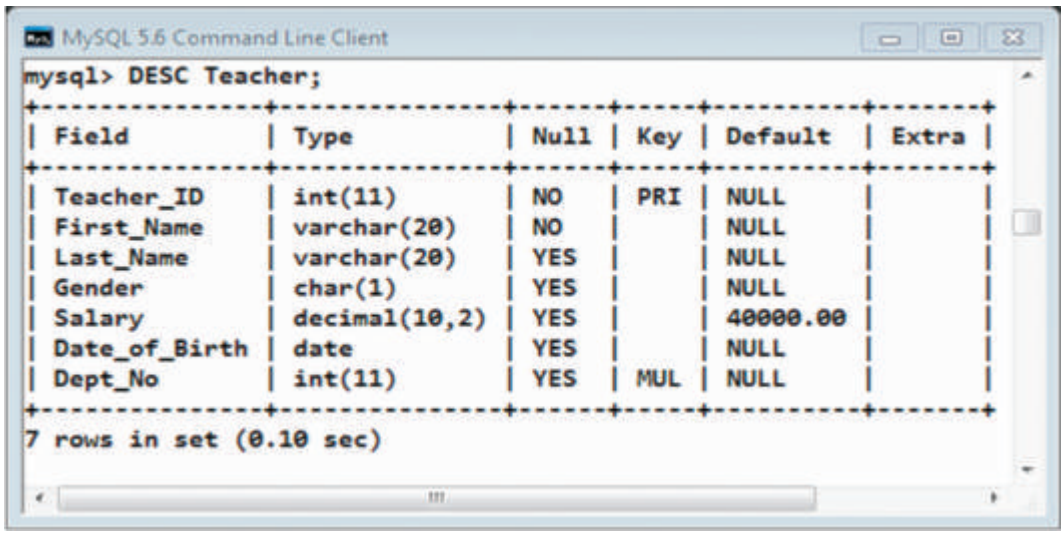
In this example Dept_No is the foreign key that references Dept_ID of Department relation which is a primary key.

The SQL command for creating these tables would be as follows:

```
CREATE TABLE Department
(
Dept_ID INTEGER PRIMARY KEY,
Dept_Name VARCHAR(20) NOT NULL
);
CREATE TABLE Teacher
(
```

```
Teacher_ID INTEGER PRIMARY KEY,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER,
FOREIGN KEY (Dept_No) REFERENCES Department(Dept_ID)
);
```

The foreign key is created separately by using the key words `FOREIGN KEY` followed by the attribute that is the foreign key within parenthesis, then the keyword `REFERENCES` followed by the name of the referred relation and its primary key within parenthesis.



As shown in above Figure, the key column for `Dept_No` contains `MUL` which implies multiple occurrences of a given value can appear within the column. This is also because it is a foreign key.

Note that the referenced table must be created before it is referred in any other table.

You can also specify the action to be taken in case the foreign key attribute or the primary key attribute value are changed as that may result in violation of the referential integrity constraint. This may be done using the following commands:

1. `ON DELETE`
2. `ON UPDATE`

Actions that can be taken are as follows:

1. `SET NULL`
2. `CASCADE`
3. `RESTRICT`

Let us discuss each in detail:

- ```
  CREATE TABLE Teacher
  (
  Teacher_ID INTEGER PRIMARY KEY,
  First_Name VARCHAR(20) NOT NULL,
  Last_Name VARCHAR(20),
  Gender CHAR(1),
  Salary DECIMAL(10,2) DEFAULT 40000,
  Date_of_Birth DATE,
  Dept_No INTEGER,
  FOREIGN KEY (Dept_No) REFERENCES Department (Dept_ID) ON
  DELETE SET NULL ON UPDATE SET NULL
  );
  ```

  Thus if a department with a given value of `Dept_ID` is deleted in Department table, then the corresponding tuples that contains the deleted value for `Dept_No` attribute in Teacher table would be set to `NULL`. Similarly, if `Dept_ID` value is updated then also the corresponding attribute in Teacher table would be set to `NULL`.

- ```
  CREATE TABLE Teacher
  (
  Teacher_ID INTEGER PRIMARY KEY,
  First_Name VARCHAR(20) NOT NULL,
  Last_Name VARCHAR(20),
  Gender CHAR(1),
  Salary DECIMAL(10,2) DEFAULT 40000,
  Date_of_Birth DATE,
  Dept_No INTEGER,
  FOREIGN KEY (Dept_No) REFERENCES Department (Dept_ID) ON
  DELETE CASCADE ON UPDATE CASCADE
  );
  ```

  In the above table, if a department with a given value of the `Dept_ID` attribute in Department table is deleted, then the corresponding rows in the Teacher table would also be deleted. However if `Dept_ID` value is updated in the Department table, the change in corresponding value is also reflected in Teacher Table.

- ```
  CREATE TABLE Teacher
  (
  Teacher_ID INTEGER PRIMARY KEY,
  First_Name VARCHAR(20) NOT NULL,
  ```
  ```

```
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER,
FOREIGN KEY (Dept_No) REFERENCES Department (Dept_ID) ON
DELETE RESTRICT ON UPDATE RESTRICT
);
```

RESTRICT option will reject the delete or update operation for the referenced table if there are one or more related foreign key values in a referencing table, i.e, you cannot delete or update a department if there are teachers who belong to that department.

Self-Referencing Tables: A foreign key constraint can reference columns within the same table. These tables are called as self-referencing tables. For example, consider a table Employee that contains five columns: `Employee_ID`, `Name`, `Age`, `Salary` and `Manager_ID`. Because the manager is also an employee, there is a foreign key relationship between the `Manager_ID and Employee_ID` as shown below:

```
CREATE TABLE Employee
(
Employee_ID INTEGER PRIMARY KEY,
Name VARCHAR(30),
Age INTEGER,
Salary DECIMAL(10,2),
Manager_ID INTEGER,
FOREIGN KEY (Manager_ID) REFERENCES Employee
(Employee_ID)
);
```

1. **Naming of Constraint:** Constraints can be named in the Create Table command. The advantage is that named constraints can be easily deleted or updated using the Alter Table command. A constraint can be named by using the keyword `CONSTRAINT` followed by the name of the constraint and its specification.

   For example consider the following Create Table command:

```
CREATE TABLE Teacher
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
```

```
Date_of_Birth DATE,
Dept_No INTEGER,
CONSTRAINT TEACHER_PK PRIMARY KEY (Teacher_ID),
CONSTRAINT TEACHER_FK FOREIGN KEY (Dept_No) REFERENCES
Department(Dept_ID) ON DELETE SET NULL ON UPDATE SET NULL
);
```

In the above table, the primary key constraint is named as TEACHER_PK and the foreign key constraint is named as TEACHER_FK.

2. **Drop Table Command:** This command is used to delete tables. For example, suppose you want to drop the Teacher table then the command would be:

```
DROP TABLE Teacher CASCADE;
```

Thus Teacher table would be dropped and with the CASCADE option, i.e. all the constraints that refer this table would also be automatically dropped.

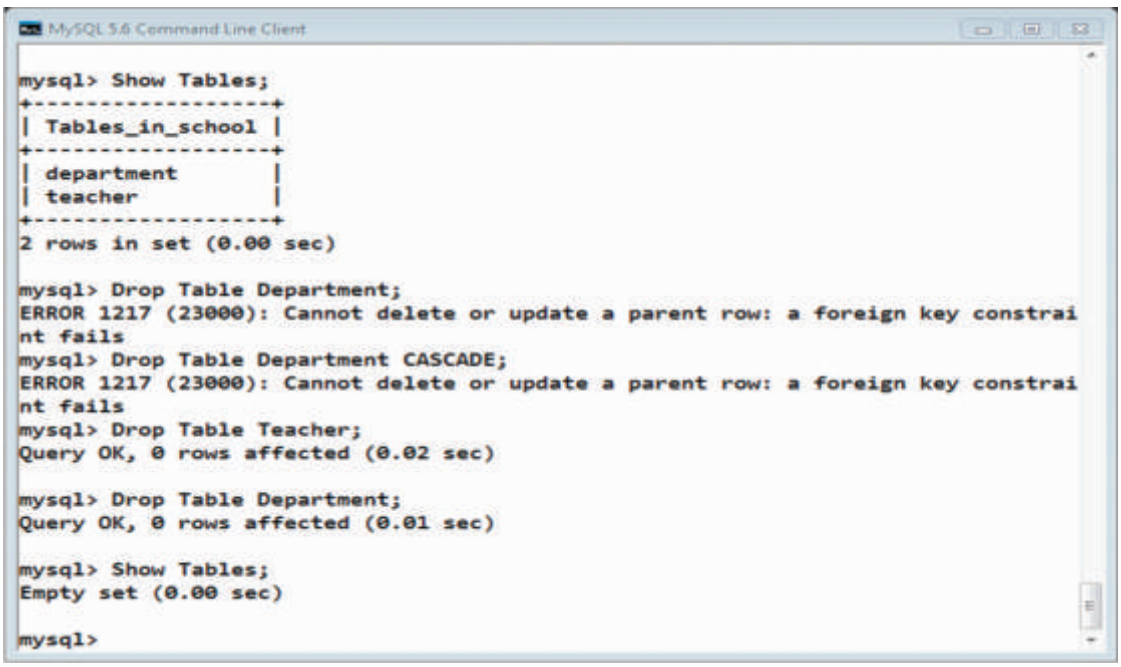
However if the requirement is that the table should not be dropped if it is being referenced in some other table then RESTRICT option can be used as shown below:

```
DROP TABLE Teacher RESTRICT;
```

Drop table command can also be used to drop named constraints. However it is used along with the Alter Table command which is discussed in next section.

Note in MySQL server 5.6.20, `RESTRICT` and `CASCADE` options are not supported though they are permitted to make porting easier.

In MySQL server 5.6.20, you can simply type Drop Table followed by the table name to delete a table from the database. However, it will not allow you to drop a table if the table to be deleted is being referenced in some other table.



As shown in the Figure, there were two tables Department and Teacher. Table Teacher contained a above foreign key that references the primary key of Department table. Hence when we attempt to drop Department table, error is shown as Department table is being referred in Teacher Table. Also note that the CASCADE option is not working as it is not supported in MySQL server 5.6.20. Department table can only be deleted if all the tables that refer Department table are deleted so that there are no references to this table anywhere else. That is why after we have dropped Teacher table, Department table deletion was possible as shown above.

3. **Alter Table Command:** This command is used to modify the base table definition. The modifications that can be done using this command are:

(a) Adding a column: Suppose we want to add a column Age in the Teacher table. Following command is used to add the column:

```
ALTER TABLE Teacher ADD Age INTEGER;
```

(b) Dropping a column: A column can be dropped using this command but one must specify the options (`RESTRICT or CASCADE`) for the drop behavior. As discussed earlier, `RESTRICT` would not let the column be dropped if it is being referenced in other tables and `CASCADE` would drop the constraint associated with this column in this relation as well as all the constraints that refer this column.

```
ALTER TABLE Teacher DROP Dept_No CASCADE;
```

This will drop the `Dept_No` column in the Teacher Table and it would also drop the foreign key constraint `TEACHER_FK` as it uses this column.

As stated earlier, `RESTRICT` and `CASCADE` options are not supported in MySQL server 5.6.20. Hence it will not allow to drop a column if it is referred anywhere else. The MySQL command to drop Age Column from Teacher is:

```
ALTER TABLE Teacher DROP Age;
```

(c) Altering a Column: A column definition can also be altered. For example – dropping the default value or defining a new default value. For example, in the Teacher table the default value of Salary is 40000. If you want to drop this default value or change this value to 30000 then it can be done by using the following commands:

```
ALTER TABLE Teacher ALTER Salary DROP DEFAULT;
ALTER TABLE Teacher ALTER Salary SET DEFAULT 30000;
```

(d) Dropping keys: A foreign key/primary key/key can be dropped by using `ALTER TABLE` command. For example if you want to delete the foreign key `TEACHER_FK` in the Teacher table then following command can be used:

```
ALTER TABLE Teacher DROP FOREIGN KEY TEACHER_FK;
```

`CASCADE` or `RESTRICT` option can be specified for the drop behavior (though not supported in MySQL server 5.6).

Primary key can be dropped by using the command:

```
ALTER TABLE Teacher DROP PRIMARY KEY TEACHER_PK;
```

However, primary key cannot be removed if it is the only primary key of the table. Hence the above command will result in an error.

(e) Adding a Constraint: If you want to add the foreign key constraint `TEACHER_FK` back, then the command would be:

```
ALTER TABLE Teacher ADD CONSTRAINT TEACHER_FK FOREIGN KEY
(Dept_No) REFERENCES Department(Dept_ID) ON DELETE SET NULL ON
UPDATE SET NULL;
```

4. **Insert Command:** This command is used to insert a tuple in a relation. We must specify the name of the relation in which tuple is to be inserted and the values. The values must be in the same order as specified during the Create Table command. For example, consider the following table Teacher:

```
CREATE TABLE Teacher
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER,
CONSTRAINT TEACHER_PK PRIMARY KEY (Teacher_ID),
);
```

To insert a tuple in the Teacher table `INSERT` command can be used as shown below:

```
INSERT INTO Teacher
VALUES (101,"Shanaya", "Batra", 'F', 50000, '1984-08-11', 1);
```

Note that the values listed above are in the same order of the attributes as specified in the Create Table command.

Another form of `INSERT` command is used to insert a tuple in which the ordering of values is done by explicitly specifying the attribute names as shown below:

```
INSERT INTO Teacher (First_Name, Last_Name, Gender,
Teacher_ID, Date_of_Birth, Dept_No, Salary)
VALUES ("Shanaya", "Batra", 'F', 101, '1984-08-11', 1, 50000);
```

Note that the values entered are ordered with respect to the attributes mentioned. If an attribute value is not explicitly specified its `DEFAULT` value is used. If `DEFAULT` value is also not specified then `NULL` value is used. Thus in the above example if salary was not

specified in the INSERT command, then its default value i.e.40000 as given in the table creation command would have been used as shown below.

```
INSERT INTO Teacher (First_Name, Last_Name, Gender,
Teacher_ID, Date_of_Birth, Dept_No, Salary)
VALUES ("Shanaya", "Batra", 'F', 101, '1984-08-11', 1,);
```

5. **Update Command:** This command is used to update the attribute values of one or more tuples in a table. For example in the Teacher table, we want to update the Salary of teacher with Teacher_ID=101 to 55000. This can be done using the following command:

```
UPDATE Teacher
SET Salary=55000
WHERE Teacher_ID=101;
```

We can also have an algebraic expression in the SET clause. Suppose it is required to increase the salary of a Teacher Shanaya by 5000, the command would be:

```
UPDATE Teacher
SET Salary=Salary+5000
WHERE Teacher_Name="Shanaya";
```

6. **Delete Command:** In order to delete one or more tuples, DELETE command is used. If we want to delete the tuple for Teacher with ID=101 the command would be:

```
DELETE FROM Teacher
WHERE Teacher_ID=101;
```

If the WHERE clause is missing then it will delete all the tuples in a table as shown below:

```
DELETE FROM Teacher;
```

7. **Select Command:** The SELECT Command is used to retrieve information from a database. There are various ways in which the SELECT command can be used. Syntax of SELECT Command is as follows:

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>
```

The basic structure of a SELECT command comprises of the keyword SELECT followed by the attribute list (separated by comma (,)) you want to select, followed by FROM clause followed by the table name and lastly an optional WHERE clause which is followed by a condition that can be a Boolean expression or another SELECT command that identifies the tuples to be selected.

Consider the following tables in the School Database for all the queries that follow:

```
CREATE TABLE Department
```

```
(
Dept_ID INTEGER PRIMARY KEY,
Dept_Name VARCHAR(30) NOT NULL
);
CREATE TABLE Teacher
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_No INTEGER,
CONSTRAINT TEACHER_PK PRIMARY KEY (Teacher_ID),
CONSTRAINT TEACHER_FK FOREIGN KEY (Dept_No) REFERENCES
Department (Dept_ID)
);
```

The Teacher table refers Department table to keep track of the department to which a teacher belongs.

Assume that the following data has been inserted in the above tables using the Insert commands:

DEPARTMENT

Dept_ID	Dept_Name
1	Chemistry
2	Computer Science
3	English
4	Hindi
5	Physics
6	Commerce
7	Biology
8	Mathematics
9	Economics

TEACHER

Teacher_ID	First_Name	Last_Name	Gender	Salary	Date_of_Birth	Dept_No
101	Shanaya	Batra	F	50000	1984-08-11	1
102	Alice	Walton	F	48000	1983-02-12	3
103	Surbhi	Bansal	F	34000	1985-06-11	4
104	Megha	Khanna	F	38000	1979-04-06	4
105	Tarannum	Malik	F	54000	1978-04-22	5
106	Tarun	Mehta	M	50000	1980-08-21	2
107	Puneet	NULL	M	52500	1976-09-25	3
108	Namit	Gupta	M	49750	1981-10-19	1
109	Neha	Singh	F	49000	1984-07-30	7
110	Divya	Chaudhary	F	39000	1983-12-11	6
111	Saurabh	Pant	M	40000	1982-01-05	8
112	Sumita	Arora	F	40000	1981-10-10	9
113	Vinita	Ghosh	F	51500	1980-09-09	9
114	Vansh	NULL	M	53500	1982-05-04	2

(a) **Query:** To retrieve all the information about Teacher with ID=101. In this query we have to specify all the attributes in the `SELECT` clause. An easier way to do this is to use asterisk (*), which means all the attributes.

```
SELECT *
FROM Teacher
WHERE Teacher_ID=101;
```

Result:

Teacher_ID	First_Name	Last_Name	Gender	Salary	Date_of_Birth	Dept_No
101	Shanaya	Batra	F	50000	1984-08-11	1

(b) **Query**: To find the names of all teachers earning more than 50000.

```
SELECT First_Name,Last_Name
FROM Teacher
WHERE salary > 50000;
```

Result:

First_Name	Last_Name
Tarannum	Malik
Puneet	NULL
Vinita	Ghosh
Vansh	NULL

Logical comparisons equal to (=), less than (<), greater than (>), less than or equal to (<=), greater than or equal to(>=), not equal to(<>) can be used in the `WHERE` clause.

(c) Query: To display `Teacher_ID,First_Name,Last_Name` and `Dept_No` of teachers who belongs to department number 4 or 7.

```
SELECT Teacher_ID,First_Name,Last_Name, Dept_No
FROM Teacher
WHERE Dept_No = 4 OR Dept_No = 7;
```

Result:

Teacher_ID	First_Name	Last_Name	Dept_No
103	Surbhi	Bansal	4
104	Megha	Khanna	4
109	Neha	Singh	7

Thus Boolean operations `AND,OR` can also be used in the `WHERE` clause.

(d) Query: To retrieve names of all the teachers and the names and numbers of their respective departments.

Note that the above query requires two tables – Teacher and Department. Consider the following query:

```
SELECT First_Name, Last_Name, Dept_ID, Dept_Name
FROM Teacher, Department;
```

This query will result in a set in which the number of rows will be the number of rows in Teacher table (14) multiplied with number of rows in Department table (9), i.e. 126 rows. Since we have not specified any WHERE clause, this query will combine each row in Teacher table with each row of Department table resulting in Cartesian product of two tables. This is called as Cross Join.

Now if we have to combine each teacher with his/her respective department, there should be a connecting column between the two tables which can be used to join them. Thus `Dept_No` (Teacher table) and `Dept_ID` (Department table) can be used to join the two tables. This type of join will result in joining rows from Teacher and Department table

based on the equality between Dept_No and Dept_ID columns. The query is as follows:

```
SELECT First_Name, Last_Name, Dept_ID, Dept_Name
FROM Teacher, Department
WHERE Dept_ID=Dept_No;
```

Here Dept_ID=Dept_No is the join condition. This would produce the following result:

First_Name	Last_Name	Dept_ID	Dept_Name
Shanaya	Batra	1	Chemistry
Namit	Gupta	1	Chemistry
Tarun	Mehta	2	Computer Science
Vansh	NULL	2	Computer Science
Alice	Walton	3	English
Puneet	NULL	3	English
Surbhi	Bansal	4	Hindi
Megha	Khanna	4	Hindi
Tarannum	Malik	5	Physics
Divya	Chaudhary	6	Commerce
Neha	Singh	7	Biology
Saurbh	Pant	8	Mathematics
Sumita	Arora	9	Economics
Vinita	Ghosh	9	Economics

Now suppose we have to retrieve the similar details for the teacher in Chemistry department, the query would be:

```
SELECT First_Name, Last_Name, Dept_ID, Dept_Name
FROM Teacher, Department
WHERE Dept_ID=Dept_No AND Dept_name="Chemistry";
```

First_Name	Last_Name	Dept_ID	Dept_Name
Shanaya	Batra	1	Chemistry
Namit	Gupta	1	Chemistry

(e) Suppose the teacher and department table both had same names for the department number, say Dept_ID as shown below:

```
CREATE TABLE Department
```

```
(
Dept_ID INTEGER PRIMARY KEY,
Dept_Name VARCHAR (30) NOT NULL
);
CREATE TABLE Teacher
(
Teacher_ID INTEGER,
First_Name VARCHAR(20) NOT NULL,
Last_Name VARCHAR(20),
Gender CHAR(1),
Salary DECIMAL(10,2) DEFAULT 40000,
Date_of_Birth DATE,
Dept_ID INTEGER,
CONSTRAINT TEACHER_PK PRIMARY KEY (Teacher_ID),
CONSTRAINT TEACHER_FK FOREIGN KEY (Dept_ID) REFERENCES
Department (Dept_ID)
);
```

In such case, when the join condition is specified, there will be an ambiguity about which `Dept_ID` we are talking about. To resolve this problem, we have to prefix the name of the attribute with the relation name followed by a period as shown in the query below:

Query: To retrieve names of all the teachers who belong to Hindi department.

```
SELECT First_Name, Last_Name
FROM Teacher, Department
WHERE Department. Dept_ID=Teacher. Dept_ID AND
Dept_Name="Hindi";
```

First_Name	Last_Name
Surbhi	Bansal
Megha	Khanna

Another method is to create aliases. Aliases are used to resolve ambiguity of the relations. They are created by using the keyword 'AS'. For example the above query can also be written as:

```
SELECT First_Name, Last_Name
FROM Teacher AS T, Department AS D
WHERE D.Dept_ID = T. Dept_ID AND Dept_Name="Hindi";
```

Here T is an alias for Teacher table and D is an alias for Department table.

We can also create alias to rename a column name as shown below:

```
SELECT First_Name AS Fname, Last_Name AS Lname
FROM Teacher AS T, Department AS D
WHERE D.Dept_ID = T. Dept_ID AND Dept_Name="Hindi";
```

Result:

Fname	Lname
Surbhi	Bansal
Megha	Khanna

Note that the column names as displayed have been changed.

(f) In SQL, duplicate tuples can appear more than once in a table and in the result of a query. However if the requirement is to list distinct values of an attribute then this can be done by using the keyword – `'DISTINCT'`.

For example, following query will list all the Department numbers corresponding to departments having male teachers.

```
SELECT Dept_No
FROM Teacher;
WHERE GENDER ='M';
```

The result of the above query is:

Dept_No
2
3
1
8
2

In the above result, 2 is appearing twice which is not required as the query is to find only the department numbers. Hence we can use DISTINCT keyword in the SELECT clause so that there is no repetition in the result.

```
SELECT DISTINCT Dept_No
FROM Teacher;
WHERE GENDER ='M';
```

The result of the above query is:

Dept_No
1
2
3
8

(g) Sometimes it is required to match part of the string. This is called as string pattern matching. We can use 'LIKE' keyword along with two more reserved characters - % (percent) and _ (underscore) for specifying different number of characters. % replaces zero or more number of random characters and _ replaces a single character.

Some examples:

- **Query:** To retrieve names of all the teachers starting from letter 'S'.

```
SELECT First_Name
FROM Teacher
WHERE First_Name LIKE "S%";
```

Result:

First_Name
Shanaya
Surbhi
Saurabh
Sumita

- **Query:** To retrieve names of all the teachers having 6 characters in the first name and starting with 'S'.

```
SELECT First_Name
FROM Teacher
WHERE First_Name LIKE "S_ _ _ _ _";
```

Result:

First_Name
Surbhi
Sumita

- **Query:** To retrieve names of all the teachers having at least 6 characters in the first name.

```
SELECT First_Name
FROM Teacher
WHERE First_Name LIKE "_ _ _ _ _ _%";
```

Result:

First_Name
Shanaya
Surbhi
Tarannum

Puneet
Saurabh
Sumita
Vinita

(h) Suppose it required to sort the result of a query based on some attributes. This can be achieved by using the clause – `ORDER BY` followed by the attributes which needs to be sorted. (By default the order is ascending) For ascending order the keyword `ASC` and for descending order the keyword `DESC` is used. By default the order is ascending.

- **Query:** To list the names of teachers in alphabetical order.

```
SELECT First_Name, Last_Name
FROM Teacher
ORDER BY First_Name, Last_Name;
```

Result:

First_Name	Last_Name
Alice	Walton
Divya	Chaudhary
Megha	Khanna
Namit	Gupta
Neha	Singh
Puneet	NULL
Saurabh	Pant
Shanaya	Batra
Sumita	Arora
Surbhi	Bansal
Tarannum	Malik
Tarun	Mehta
Vansh	NULL
Vinita	Ghosh

- **Query:** To list the names of all the Departments in the descending order of their names.

```
SELECT Dept_Name
FROM Department
ORDER BY Dept_Name DESC;
```

Result:

Dept_Name
Physics
Mathematics
Hindi
English
Economics
Computer Science
Commerce
Chemistry
Biology

- **Query:** To retrieve the names and department numbers of all the teachers ordered by the Department number and within each department ordered by the names of the teachers in descending order.

```
SELECT First_Name, Last_Name, Dept_No
FROM Teacher
ORDER BY Dept_No ASC, First_Name DESC, Last_Name DESC;
```

First_Name	Last_Name	Dept_No
Shanaya	Batra	1
Namit	Gupta	1
Tarun	Mehta	2
Vansh	NULL	2
Alice	Walton	3
Puneet	NULL	3
Surbhi	Bansal	4
Megha	Khanna	4
Tarannum	Malik	5
Divya	Chaudhary	6
Neha	Singh	7
Saurabh	Pant	8
Sumita	Arora	9
Vinita	Ghosh	9

(i) To test whether a value is unavailable or unknown or not applicable in a column (i.e, `NULL` values), SQL allows us to test this condition using keywords `IS NULL` and `IS NOT`

NULL.

Query: To retrieve all the details of those employees whose last name is not specified.

```
SELECT *
FROM Teacher
WHERE Last_Name IS NULL;
```

Result:

Teacher_ID	First_Name	Last_Name	Gender	Salary	Date_of_Birth	Dept_No
107	Puneet	NULL	M	52500.00	1976-09-25	3
104	Vansh	NULL	M	53500.00	1982-05-04	2

(j) We can have another query in the `WHERE` clause of SQL query if the condition is based on the result of another query as shown below:

Query: To retrieve the names of all the departments having female teachers.

```
SELECT DISTINCT Dept_Name
FROM Department
WHERE Dept_ID IN (Select Dept_No
FROM Teacher
WHERE Gender = 'F');
```

Result:

Dept_Name
Chemistry
English
Hindi
Physics
Commerce
Biology
Economics

The above query is a nested query (query within another query). The outer one is called outer query and the inner one is called as inner query. The inner query will return those department ID's which have female teachers and the outer query will retrieve the names of those departments respectively.

We could have also written the above query by using JOIN condition as shown below:

```
SELECT DISTINCT Dept_Name
```

```
FROM Department , Teacher
WHERE Dept_ID = Dept_No AND Gender='F';
```

(k) Sometimes it is required to apply certain mathematical functions on group of values in a database. Such functions are called **Aggregate Functions**. For example retrieving the total number of teachers in all the Departments. Following are the commonly used built-in aggregate functions:

- **COUNT**- It counts the numbers of tuples in the result of the query.
- **SUM** – It finds the sum of all the values for a selected attribute which has numeric data type.
- **MAX** –It finds the maximum value out of all the values for a selected attribute which has numeric data type.
- **MIN** - It finds the minimum value out of all the values for a selected attribute which has numeric data type.
- **AVG** – It finds the average value of all the values for a selected attribute which has numeric data type.

Examples:

- **Query:** To find total salary of all the teachers .

```
SELECT SUM(Salary) AS Total_Salary
FROM Teacher;
```

Result:

Total_Salary
203250.00

The above query finds the sum of all the values in the Salary column and the column is renamed via an alias as `Total_Salary` which is optional as it only enhances readability.

- **Query:** To find the maximum and minimum salary.

```
SELECT MAX(Salary) AS Max_Salary, MIN(Salary) AS
Min_Salary
FROM Teacher;
```

Result:

Max_Salary	Min_Salary
54000.00	34000.00

- **Query:** To count the number of teachers earning more than Rs 40000.

```
SELECT COUNT(Salary)
FROM Teacher
WHERE Salary > 40000;
```

Result:

COUNT (Salary)
9

The above query can also be written as follows:

```
SELECT COUNT(*)
FROM Teacher
WHERE Salary >40000;
```

The difference is in the use of asterisk (*). Asterisk symbol is used to count the number of rows in the result of the query.

- **Query:** To retrieve the number of teachers in "Computer Science" Department.

```
SELECT COUNT(*) AS No_of_Computer_Science_Teachers
FROM Department, Teacher
WHERE Dept_Name = "Computer Science"AND Dept_No=Dept_ID;
```

Result:

No_of_Computer_Science_Teachers
2

(I) We can also use arithmetic operators in the `SELECT` clause. For example, if we want to display Teacher name, current salary and a 10% increase in the salary for those teachers who belongs to Department number 4, the `SELECT` statement can be written as shown below:

```
SELECT First_Name, Last_Name, Salary, Salary*1.1 AS New_Salary
FROM Teacher
WHERE Dept_No = 4;
```

Result:

First_Name	Last_Name	Salary	New_Salary
Surbhi	Bansal	34000.00	37400.00
Megha	Khana	38000.00	41800.00

(m) Grouping based on an attribute can be done in SQL. For such grouping, `GROUP BY` clause is added in the SQL query.

For example, we have to find the number of teachers teaching in each Department. Thus we have to group the result based on the Departments and for each Department we have to count number of teachers who teach in that Department. This query is written by using `GROUP BY` clause and aggregate function as shown below:

```
SELECT Dept_No, COUNT(*) AS No_of_Teachers
FROM Teacher
GROUP BY Dept_No;
```

Result:

Dept_No	No_of_Teachers
1	2
2	2
3	2
4	2
5	1
6	1
7	1
8	1
9	2

The above result can be enhanced if we display the name of Departments also as shown below:

```
SELECT Dept_No, Dept_Name, COUNT(*) AS No_of_Teachers
FROM Teacher, Department
WHERE Dept_ID = Dept_No
GROUP BY Dept_No;
```

Result:

Dept_No	Dept_Name	No_of_Teachers
1	Chemistry	2
2	Computer Science	2
3	English	2
4	Hindi	2
5	Physics	1

6	Commerce	1
7	Biology	1
8	Mathematics	1
9	Economics	2

Also it is important to note that the attribute used for grouping (`Dept _No` in the above query) **must** also be present in the `SELECT` clause.

It is also possible to apply some condition on the group. This condition will not come under the `WHERE` clause, but in a new clause `HAVING`.

For example, we have to find those departments which have more than one teacher.

```
SELECT Dept_No, Dept_Name, COUNT(*) AS No_of_Teachers
FROM Teacher, Department
WHERE Dept_No=Dept_ID
GROUP BY Dept_No
HAVING COUNT(*) > 1;
```

Result:

Dept_No	Dept_Name	No_of_Teachers
1	Chemistry	2
2	Computer Science	2
3	English	2
4	Hindi	2
9	Economics	2

(n) A Select command can also result in an empty set.

For example, retrieve the name of Teacher with ID=115. Since there is no such teacher in the Teacher table, following query results in an empty set.

```
SELECT *
FROM Teacher
WHERE Teacher_ID = 115;
```

Result:
Empty set

Exercise:

Q1. Consider the following Employee table:

Table Name: Employee

Employee_ID	Employee_Name	Job_Title	Salary	Bonus	Age	Manager_ID
1201	Divya	President	50000	NULL	29	NULL
1205	Amyra	Manager	30000	2500	26	1201
1211	Rahul	Analyst	20000	1500	23	1205
1213	Manish	Salesman	15000	NULL	22	1205
1216	Megha	Analyst	22000	1300	25	1201
1217	Mohit	Salesman	16000	NULL	22	1205

The primary key of this table is Employee_ID and Manager_ID is a foreign key that references Employee_ID.

Write SQL commands for the following:

(a) Create the above table.

(b) Insert values as shown above.

(c) Delete the Employee having Employee_ID 1217.

(d) Update the salary of "Amyra" to 40000.

(e) Alter the table Employee so that NULL values are not allowed for Age column.

(f) Write a query to display names and salaries of those employees whose salary are greater than 20000.

(g) Write a query to display details of employees who are not getting any bonus.

(h) Write a query to display the names of employees whose name contains "a" as the last alphabet.

(i) Write a query to display the name and Job title of those employees whose Manager_ID is 1201.

(j) Write a query to display the name and Job title of those employees whose Manager is "Amyra".

(k) Write a query to display the name and Job title of those employees aged between 26 years and 30 years (both inclusive)

Q2. A Railway company uses machines to sell tickets. The machine details and daily sales information are recorded in two tables:

Table Name: Machine

Field	Data Type
Machine_ID	CHAR (3)
Station	CHAR (30)

Table Name: Sales

Field	Data Type
Machine_ID	CHAR (3)
Date	DATE
Tickets_Sold	INTEGER
Income	DECIMAL (8, 2)

The primary key of the table Machine is Machine_ID. Records in the table Sales are uniquely identified by the fields Machine_ID and Date.

(a) Create the tables Machine and Sales.

(b) Write a query to find the number of ticket machines in each station.

(c) Write a query to find the total ticket income of the station "New Delhi" of each day.

(d) Write a query to find the total number of tickets sold by the machine (Machine_ID = 122) till date.

Unit - 2: Operating Web

2.1 Operating Web-based Applications

An application that can be accessed via a web browser, over a network connection is called a web-based application. A web-based application provides the convenience of availability 24x7 across the world. Nowadays, web-based applications are used for reservation of tickets and bookings, e-governance, online shopping with the provision of making payments online using credit/debit cards, and learning using online tutorials, tests etc. They allow access to the facilities via the Internet from the comfort of home, office, car, etc.

The web-based online applications can be broadly categorized into applications that require financial online transactions and applications that provide information and allow interaction via filling forms, posting query, viewing information, sending email or chatting. The online reservation system, online shopping, and bill payments fall in the first category and e-governance and online tutorial and tests fall in the second category. In this chapter, we discuss both kinds of applications.

2.2 Online Reservation Systems

An online reservation system allows booking /cancellation of tickets through use of Internet. A user can retrieve information and perform transaction related to reservation of ticket. Some examples of online reservation systems are :

- Airline ticket
- Railway ticket
- Bus ticket
- Movie ticket
- Hotel booking
- Car rental
- Tour and travel booking

Such reservation systems store information and allow the users to retrieve the information, for example, the information about all the flights from Delhi to Mumbai like, timings, prices, air flight provider (Air India, Spice jet etc.) and availability of seats. A user who wishes to fly from Delhi to Mumbai can check the information about the flight timings, rates, seat availability, etc., and book a ticket.

2.2.1 Advantages of Online Reservation System

Online reservation systems are advantageous both for the company providing reservation services and for the consumer who uses the online reservation system for booking. We discuss the advantages of the online reservation system :

Advantages for consumers

- **Convenient:** The user/consumer can book tickets anytime anywhere - from home,

office, while travelling, etc. All the user needs is just a computer, Internet access, and a card (credit, debit, etc.) for payment. The user saves time (going physically to a booking office in the defined timings, finding parking for car, etc., standing in a queue in booking office) and energy. Making changes to bookings like, cancellation, upgradation is also easy. It is very convenient for the users with special needs.

- **Price Comparison:** In this competitive world, a service may be provided by several providers - cinema tickets on Wave cinema, PVR, Big Cinema etc.; online booking allows the customer to check the prices, compare them and get the best deal. Several websites which provide the comparisons across similar service providers exist. For example, www.makemytrip.com provides comparative pricing of the air ticket providers from a particular source to a destination Figure 2.2(a). This is in contrast to manual system where a user is made to book his/her ticket for the airline that the travel agent recommends.

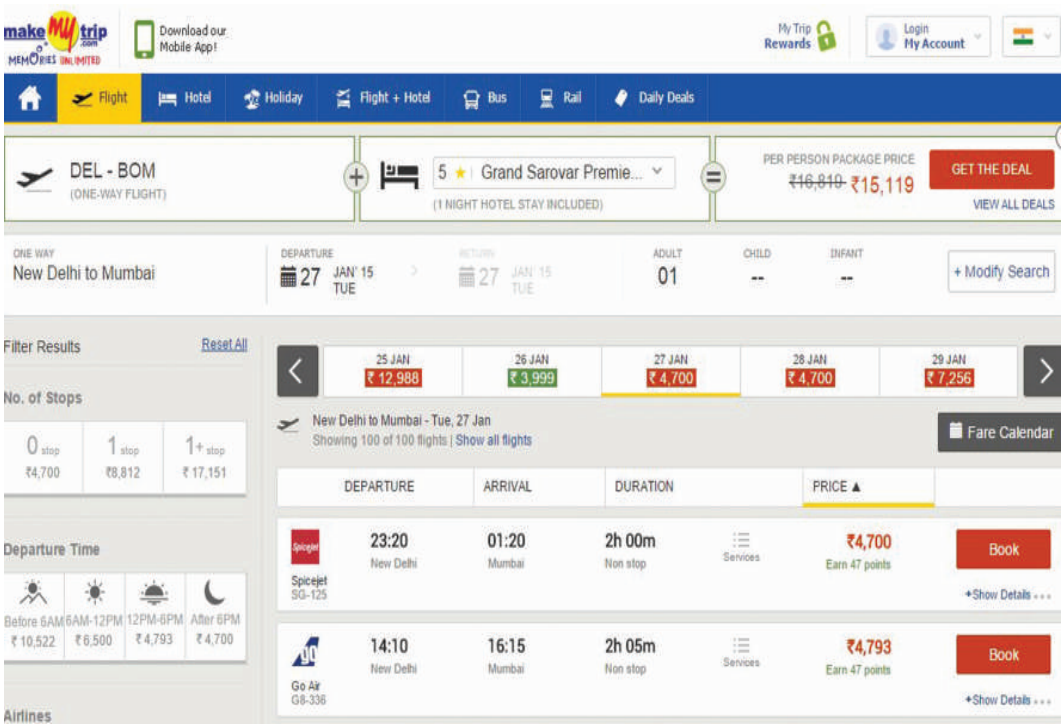


Figure 2.2(a): makemytrip.com showing ticket options for different airlines

- **Security:** Most online reservation sites require the user to create their profile and provide them with a username and password. The information thus remains secure until their password is secure.
- **Confirmation of Reservation:** When a booking is done, the confirmation of the booking just takes a few seconds. It is fast and secure. The confirmed booking of your booked ticket can be viewed on your mobile, iPad, iPhone, laptop, etc., and can also be printed.
- **Making Choices:** In online booking, the user can see the arrangement of seats, select a particular seat, etc. In railway booking, selection of lower berth/upper berth,

type of food required and booking for beddings is possible. You can read reviews posted by people when making choices.

- **Time Saving:** Online reservation consumes very less time in contrast to long queues at the counter.
- **Discount Scheme:** The users can avail the special discount schemes provided by the service provider like, season discount, advance booking discount and frequent user discount.

However, for using the online reservation system, it is important that the user must have the understanding of using the computer for reservation.

Advantages for Providers: The organization or the company providing the online reservation facility to their users also has several advantages :

- **Requires Less Staffing:** Less man-power is required as every task is done through computers.
- **No Spatial Restrictions:** The organization which is providing online reservation may be operating from the basement of their house, a garage, an office space in a small building, an office in a large commercial mall, or an office from their own building. The physical location of the office does not really matter to a customer who is reserving tickets online. So, the organization, big or small, can choose their office space depending on their needs. Online reservation also reduces the bills for office rent, electricity, etc. although initial costs need to be incurred on setting up the website and maintaining it.
- **Global Access:** Online reservation is available to anyone irrespective of their physical location. The scope of access to the online reservation becomes large as anyone who has access to the Internet can search for the reservation site and book the tickets.

2.2.2 Precautions while Performing Online Transactions

Internet transactions are susceptible to frauds. The precautions to be taken are as follows:

The combination of username and password is the most common method of user identification and authentication. There are several security issues with the use of password. An unauthorized access to an account through stealing/guessing a password can get access to the system and a simple password can be easily cracked. Some actions that can be taken to make the passwords safer are as follows:

- Make a password complex, like mix case, use numbers and special characters. This decreases ability of automated attacks by trying different character combinations.
- Be cautious not to leave passwords around and do not share them with friends.
- Never use names of near and dear ones as passwords.

2.2.3 Using Online Reservation Systems

For using any online reservation system, we need to do the following:

- Open the online reservation website: On your machine having an Internet connection, open a web browser like Internet Explorer, Google's chrome to open a website.
  - If you know the address of the website then type the address in the address bar, or
  - If you do not know the address, open a search engine like google.com or bing.com and search for an online reservation website. Get the address of the required site. For example, on the Google Search, you can search for the Air India ticket reservation site Figure 2.2(b)

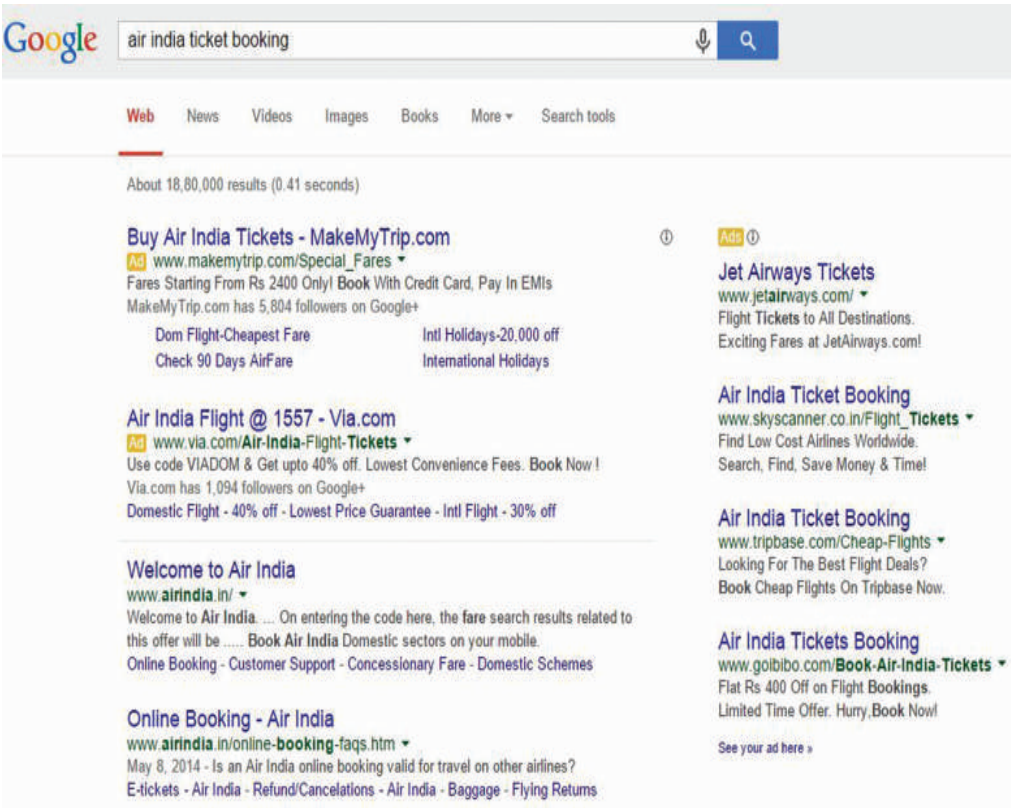


Figure 2.2(b): Google search results for air India ticket booking

- Open the website
- Browse and search for the required information
- Use the website for reservation.

2.2.4 Case Study: Book Rail Ticket

Here, we will learn online booking of train tickets using the Indian Railway Catering and Tourism Corporation (IRCTC) online portal.

- Check you have a web browser and an Internet connection on your computer.
- Open the browser and type the website address of IRCTC i.e. https://www.irctc.co.in/ in the address bar. Press <Enter>.
- A web page opens, as shown in Figure 2.2 (c)

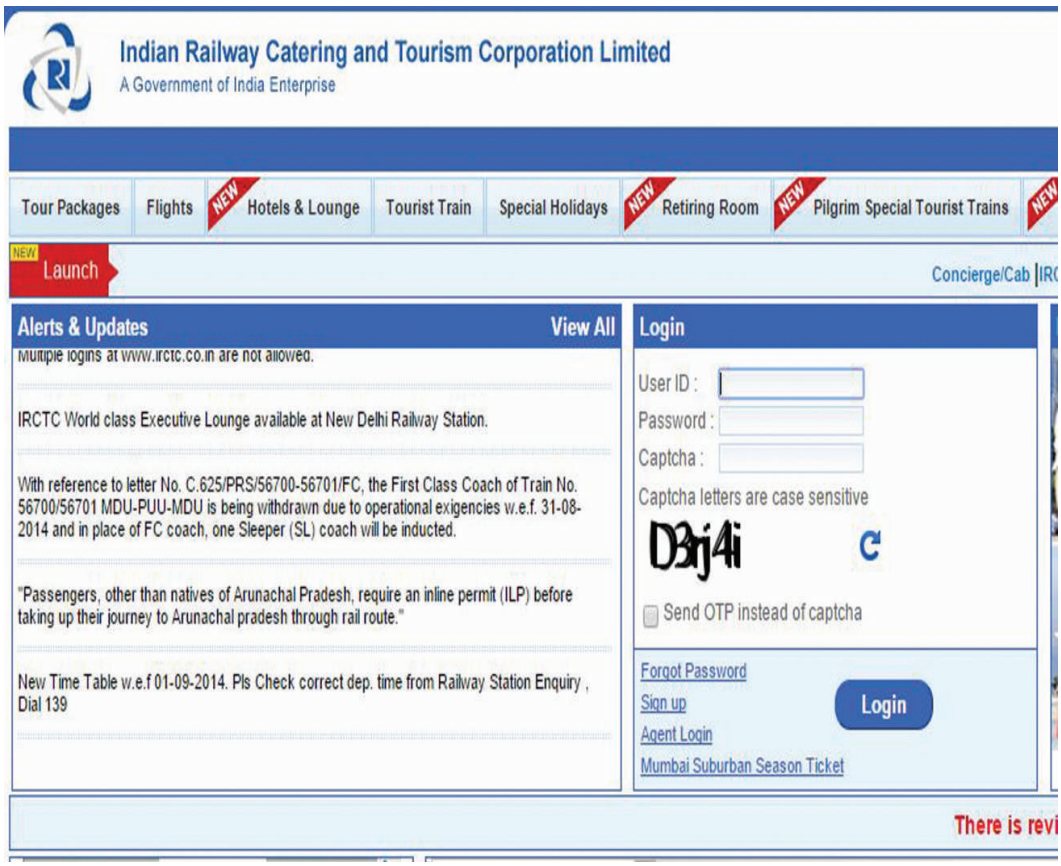


Figure 2.2(c): Home page of IRCTC website

- For booking a ticket, you need to have an account on the IRCTC website. If you already have an account in IRCTC, then type your username and password and login. If you are not an existing user, you need to create your account. You need to sign up on the IRCTC website, to perform any transaction.
- Create New Account
  - Locate and click <Signup> link. An individual registration page opens, as shown in Figure 2.2(d).
  - Read the instructions and fill the registration form. The fields marked with a star (*) are mandatory.
  - After filling the form, click the <Submit> button.
  - Once your account has been created, now you can perform transactions from your account.

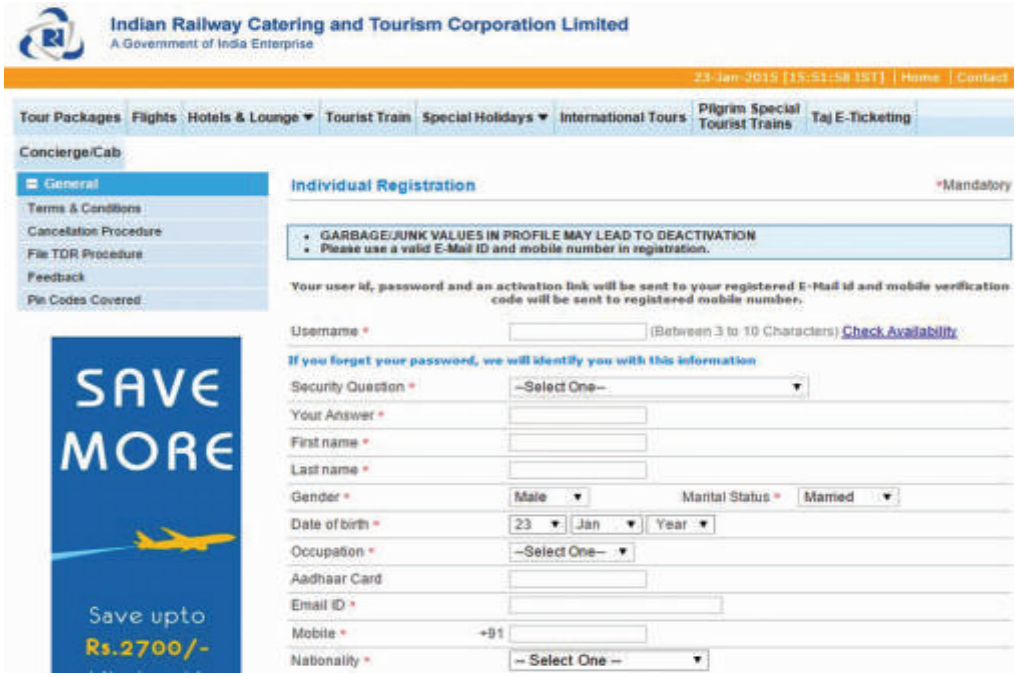


Figure 2.2(d): Individual Registration Form

- Once you have created your account you can login to the site using your username and password.
- After you login, you can see your page with your name displayed on the right side of the web page Figure 2.2(e).
- Once your account has been created, now you can perform transactions from your account.

✦ Book a Ticket

- On the Plan My Travel box on the left side of the page, enter the details, i.e. the start station, end station, date on which you want to travel, ticket type (e-ticket) and your quota. Next, click on <Find Trains>.
- A list of available trains with information about availability of tickets is displayed Figure 2.2(f).

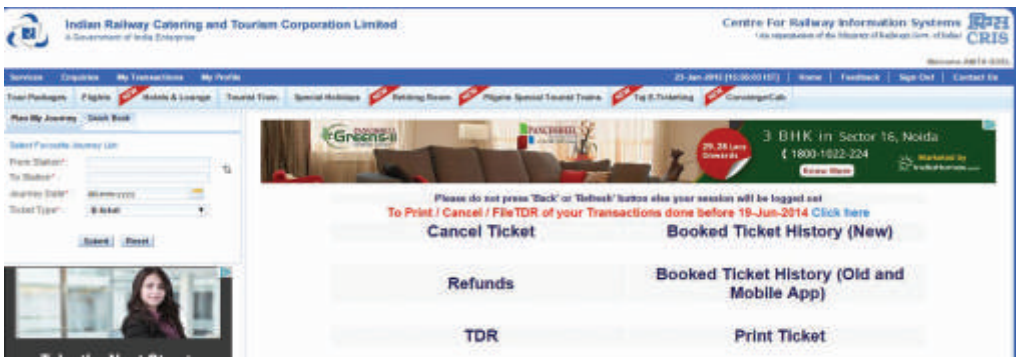


Figure 2.2(e): User's page

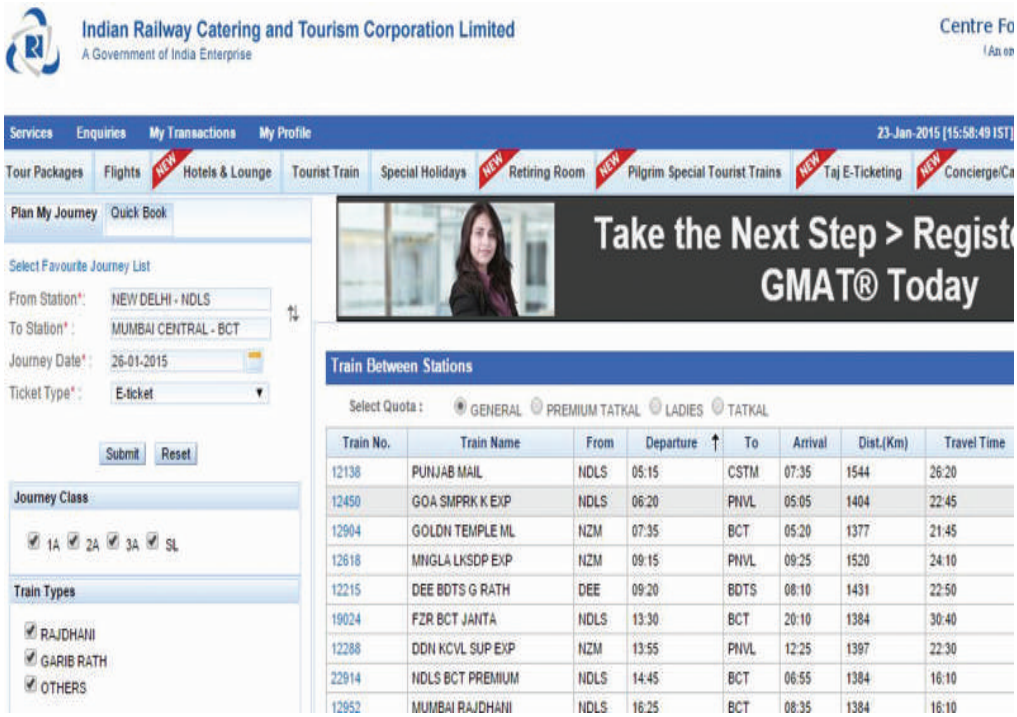


Figure 2.2(f): Trains available from source to destination on selected dates

- You can see the train number, train name, departure time, arrival time at the destination, category of the seats allowed in the train, like, 1A, 2A, 3A, SL (Sleeper) etc. A cross (x) mark depicts non availability of that category in the specific train.
- Select the category of the seat for a train. A screen appears which shows the availability of seats in the selected train and selected category, for the date requested by you and a few days ahead Figure 2.2 (g). It also shows the ticket fare.
- After you have checked the fares and you are ready to book a ticket, click on <book>.
- A page as shown in Figure 2.2(h) a form appears. Enter the detail of the persons for whom you are booking the tickets, like, name, age, sex, berth/seat preference (lower, middle, upper), and check the box, if you are a senior citizen.
- Once you have filled in the details press <enter>, the page showing your booking details is displayed Figure 2.2 (i).
- Now, you are required to make payment for the ticket Figure 2.2(i). You can make payment using credit card, debit card, etc. Once payment is done, you can view your ticket.
- An electronic copy of your paid ticket is shown on the screen. Also, your ticket will be e-mailed to you at the email address you have specified in the registration form.

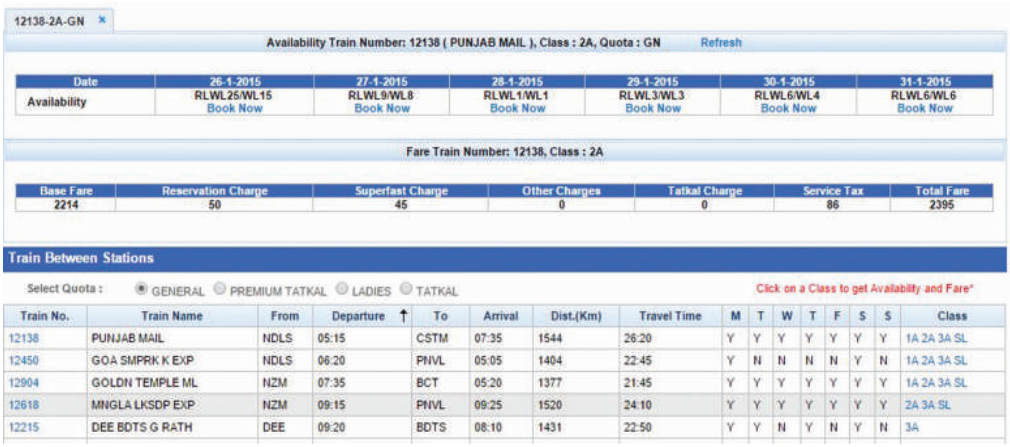


Figure 2.2(g): Ticket availability for selected train and category

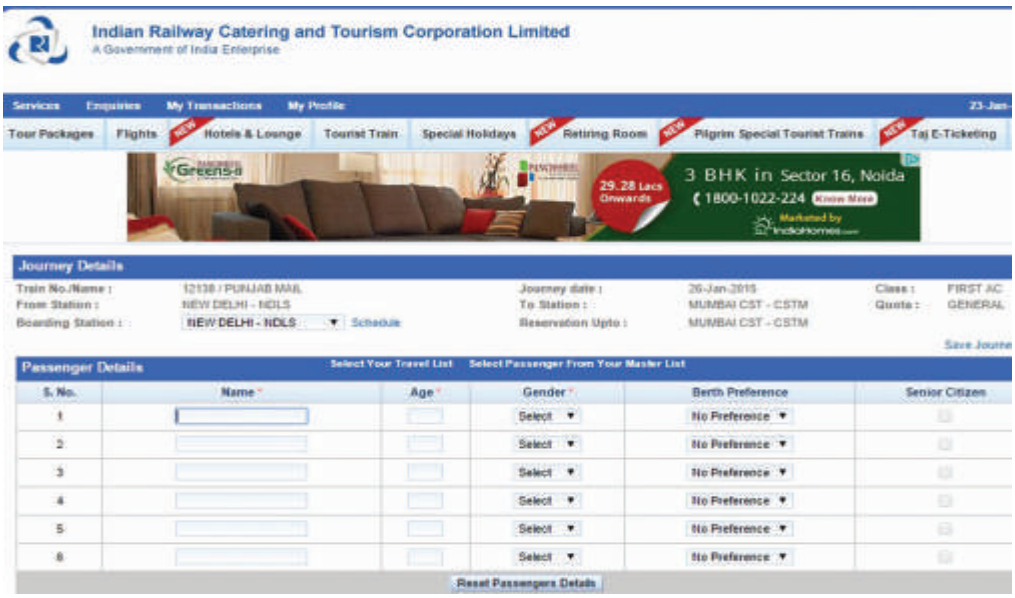


Figure 2.2(h): Form to fill the passenger details

Figure 2.2(i): Ticket details with passenger details

Figure 2.2(j): Make payment for the reserved ticket

2.3 E-Governance

E-Governance or electronic governance is the use of electronic i.e. information and communication technology (ICT) tools for governance by the government for the citizens. The ICT tools include the Internet, Local Area Network (LAN), mobiles, etc. The use of ICT facilitates providing access to information for the citizens by the government. The government has set up ICT enabled services, like, registration of birth/death certificate, booking of railway tickets, RTI application submission, etc. E-governance empowers the citizens socially and economically, and enhances their lives by providing them with quality information and better services. It enables citizens to access information directly, without paying any money to a middleman or a tout. It ushers transparency in the system. The services of the e-governance portal can be broadly classified into two broad categories : to provide information and to provide online services.

2.3.1 Initiative

In India, the main thrust for e-Governance was provided by the launching of NICNET in 1987 - the national satellite-based computer network. This was followed by the launch of the District Information System of the National Informatics Centre (DISNIC) program to computerize all district offices in the country for which free hardware and software was offered to the State Governments. NICNET was extended via the State capitals to all district headquarters by 1990. In the ensuing years, computerization, tele-connectivity and internet connectivity led to establishment of a large number of e-Governance initiatives, both at the union and state levels. The formulation of National e-Governance Plan (NeGP) by the Department of Electronics and Information Technology (DEITY) and Department of Administrative Reforms and Public Grievances (DAR&PG) in 2006 has boosted the e-Governance process.

2.3.2 E-Governance Sites

Some of the Central initiatives for e-governance include:

- National e-Governance Plan (NeGP)

- National e-Governance Division (NeGD)
- e-Governance Infrastructure
- Mission Mode Projects
- Citizens Services
- Business Services
- Government Services
- Projects and Initiatives
- R&D in e-Governance

Several state governments have also taken e-governance initiatives in the area of agriculture, commercial taxes, district, employment exchange, land records, municipalities, gram panchayats, police, road transport, treasuries, etc.

Some of the key e-governance sites of India are as follows:

1. **india.gov.in** (The National Portal of India) : This is the National Portal of India, which provides a single window access to information and services being provided by the various Indian Government entities Fig. 2.3 (a), (b). The content in this portal is the result of a collaborative effort of various Indian government ministries and departments, at the central/state/district level. This portal is Mission Mode Project under the National E-Governance Plan, designed and maintained by National Informatics Centre (NIC), DeitY, MoCIT, Government of India.

Figure 2.3(a): india.gov.in website

2. **goidirectory.nic.in (Government of India Web Directory) :** This portal is a one-point source to access all Indian Government websites at all levels and from all sectors. It includes sites of different states and union territories of India, sites of government departments like judiciary and legislature. It also provides information about the various sectors like education and agriculture Figure 2.3 (c), (d).

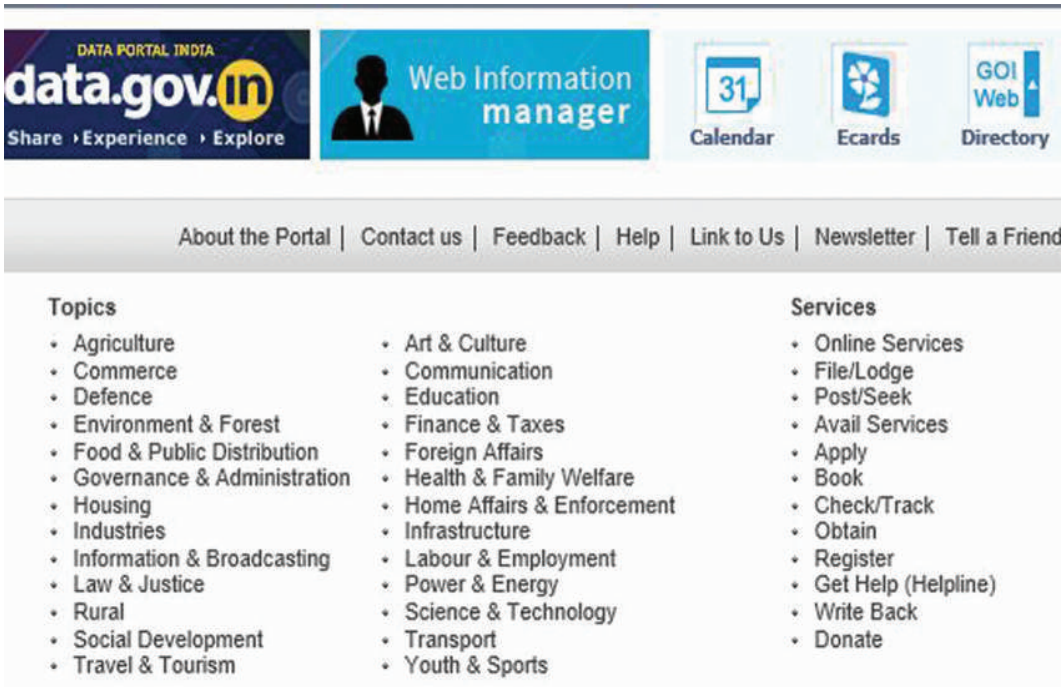


Figure 2.3(b): india.gov.in website

Figure 2.3(c): goidirectory.nic.in website

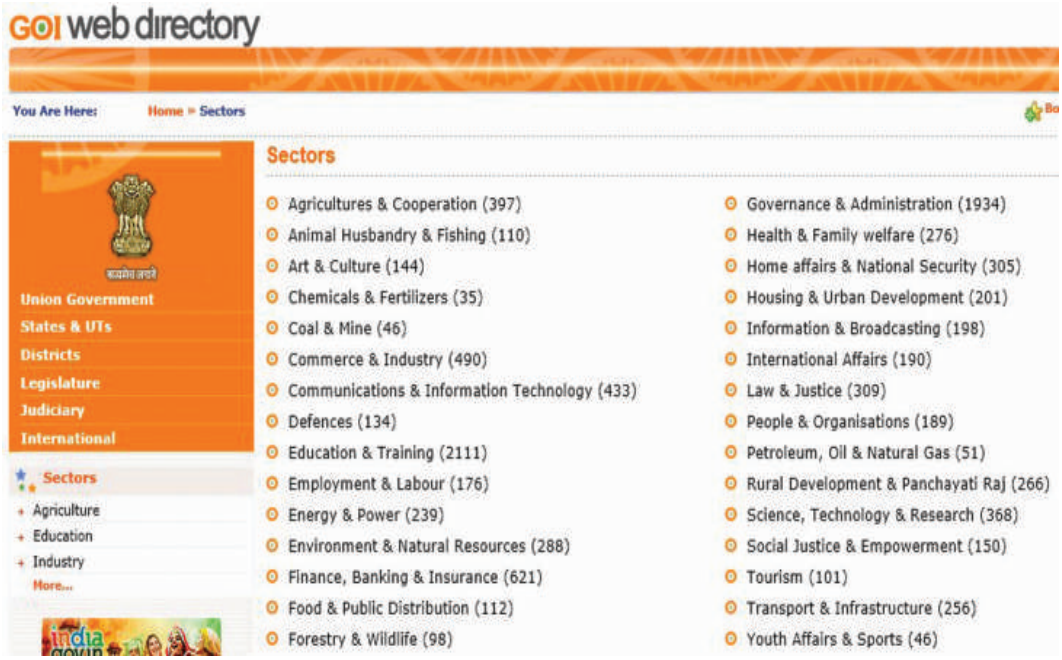


Figure 2.3(d): Sectors included in goidirectory.nic.in

The e-governance initiatives in India are discussed at http://india.gov.in/e-governance .

2.4 Online Shopping and Bill payments

Online shopping is an e-commerce application where the customer can purchase goods over the Internet. There are several websites where the customer can go for online shopping. The customer can purchase all kinds of items, like, books, TV, mobile phones, dresses, shoes, cosmetics, jewellery and greeting cards. The customer can view the goods that are displayed along with the details of the goods. The customer can select the goods to be purchased and store them in their online shopping basket. The payment for the goods is to be made using debit or credit card. The goods are then delivered at the address specified by the customer.

2.4.1 Benefits of Online Shopping

Online shopping is useful in many situations. Here, we discuss some of the situations :

- The customer does not have enough time to visit a store and purchase goods.
- The store where you will get what you need may be very far off.
- The money spent in travelling to the store, parking the car, etc., is much more than overhead if any in online shopping.
- The product you may require is not available at your market.
- You want to send a product to your friend; you can do online shopping and provide the receiver's address. You save your time and energy for delivery of the product to your friend's house located in a different city.
- You do not have time during the shopping hours. You can do online shopping from your office, home, a friend's place, anywhere, any time. Thus there are no fixed shopping hours.

- Online shopping also has bargain offers for the customer delight.
- Some online stores allow customers to rate their services and the items. This can be used to know the quality of service an online store is providing you. You can also know about the popularity of the item you are purchasing.

Since online shopping involves use of credit card or debit card for payments, and there is a need to create a user account. The precautions to be taken when doing online shopping are same as those required during online reservation as discussed in the earlier sections.

2.4.2 How it Works

The general outline of the working of an online shopping site is presented here. This may vary for different sites and can be customized for different users. The steps for online shopping are as follows:

- Open the shopping site, from where you want to shop online.
- If you just want to see the products and not buy any, you can simply browse the site
- If you wish to buy goods, then you may need to create your user account. It requires filling in a registration form with your details along with a user name and password. For a site, the user account needs to be created only once. Next time, you can simply login with your username and password. Some shopping sites may allow you to make a purchase without creating an account, or allow you to logon using your facebook or google account.
- To buy products, login with your user name and password.
- Just like you have a shopping basket when you physically go to a shop, online shopping stores also provide you with a shopping cart or basket. You can select the products that you wish to buy and add them to your shopping cart.
- Having selected your products, you can view what you have selected and the total amount to be paid by you.
- You may delete any product from the shopping cart, or proceed for payment.
- Read the terms and conditions before making the payment.
- Some examples of online shopping stores :
  - Flipkart for buying products in categories like movies, music, games, mobiles, cameras, computers, healthcare and personal products, home appliances and electronics, stationery, perfumes, toys, apparels, shoes, books, cosmetics and mobiles.
  - Ebay for buying and selling products in categories like, movies, music, games, mobiles, cameras, computers, healthcare and personal products, home appliances and electronics, stationery, perfumes, toys, apparels, shoes, books, cosmetics and mobiles.

- **Some Other Sites are:** yebhi.com, myntra.com, ibibo.com, snapdeal.com and infibeam.com Figure 2.4(a).

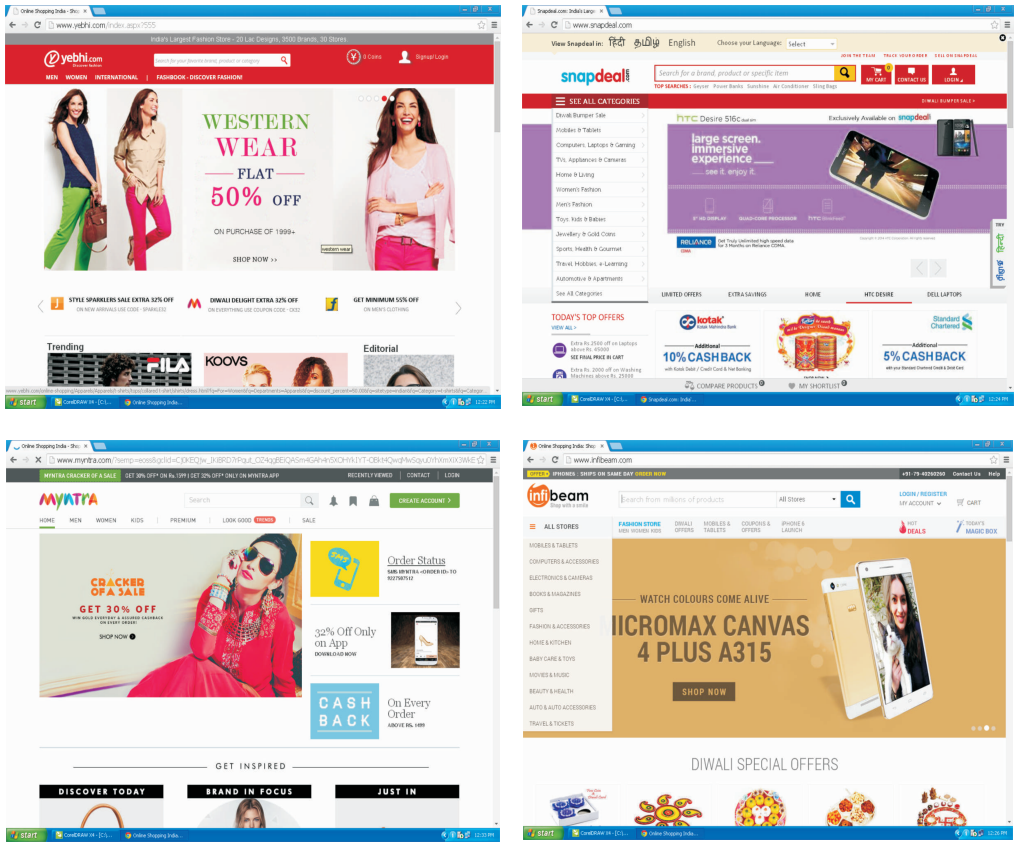


Figure 2.4(a): Some online shopping sites

2.4.3 Bill Payments

Online shoppers generally make payments using their debit card or credit card. However, different websites enable the user to make payments via alternative methods like net banking, e-gift voucher, cash on delivery and wire transfer. Always read the terms and conditions while making the payments. Bill payments can also be made for several government services like electricity bill, water bill, property tax and income tax payments.

2.4.4 Case Study: Online Shopping Using an Online Shopping Website

In this section, we shop at flipkart.com to demonstrate how online shopping can be done on a typical shopping site. For demonstration purpose the flipkart site is chosen but there is no preference for the same. The steps to be performed are as follows :

- Check you have a web browser and an Internet connection on your computer.
- Open the browser and type the website address of Flipkart i.e. https://www.flipkart.com/in the address bar. Press <Enter>. Alternatively search for the website using a search engine like google and click on the link.
- A web page opens, as shown in Figure 2.4 (b).

- For doing any kind of transaction, there is a need to open a user account.
- Click on the Sign up link. Window opens Figure 2.4(c).
- Provide your email Address which will be used as your login name. Enter a password. Make sure you enter a strong password. Re-enter the password.
- Click Sign Up Now.
- If you have been successfully signed up, a new window opens Figure 2.4(d).
- On clicking on your login name, you can see the folders available to you, like, My account, My orders and My wallet Figure 2.4(e).

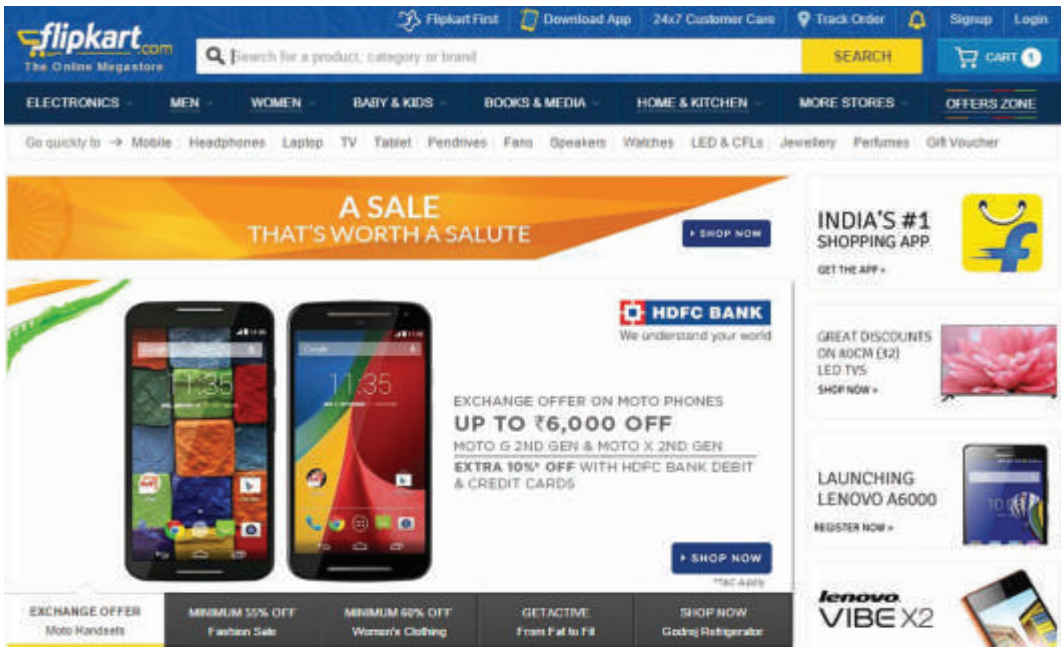


Figure 2.4(b): Flipkart: Home page

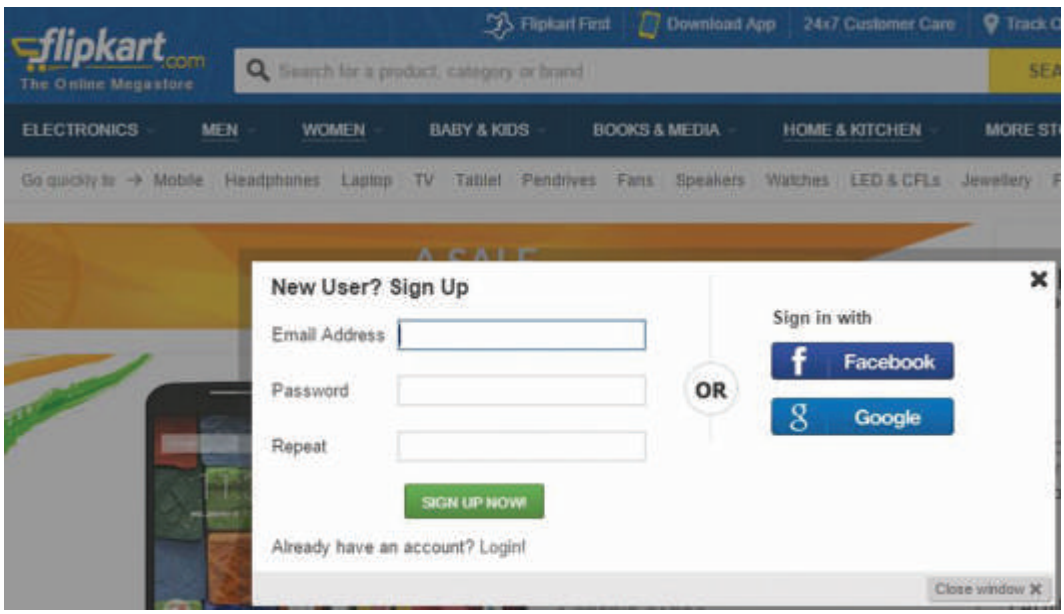


Figure 2.4(c): Flipkart: Signup page

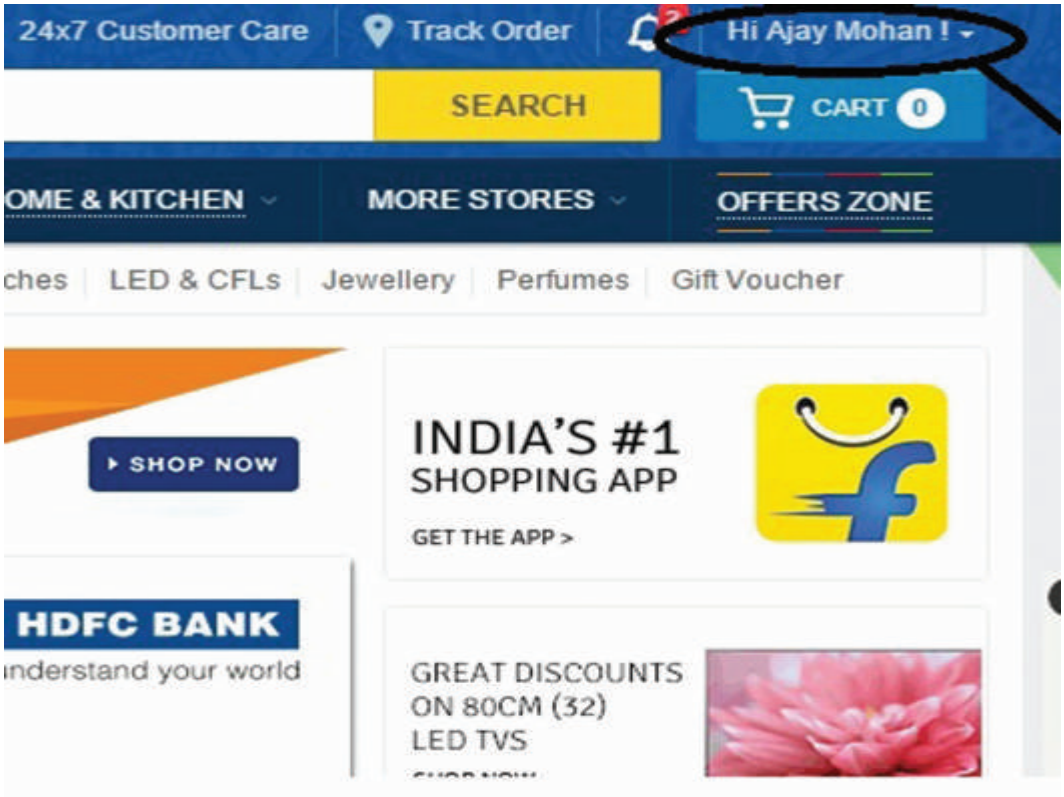


Figure 2.4(d): Flipkart: User logged in

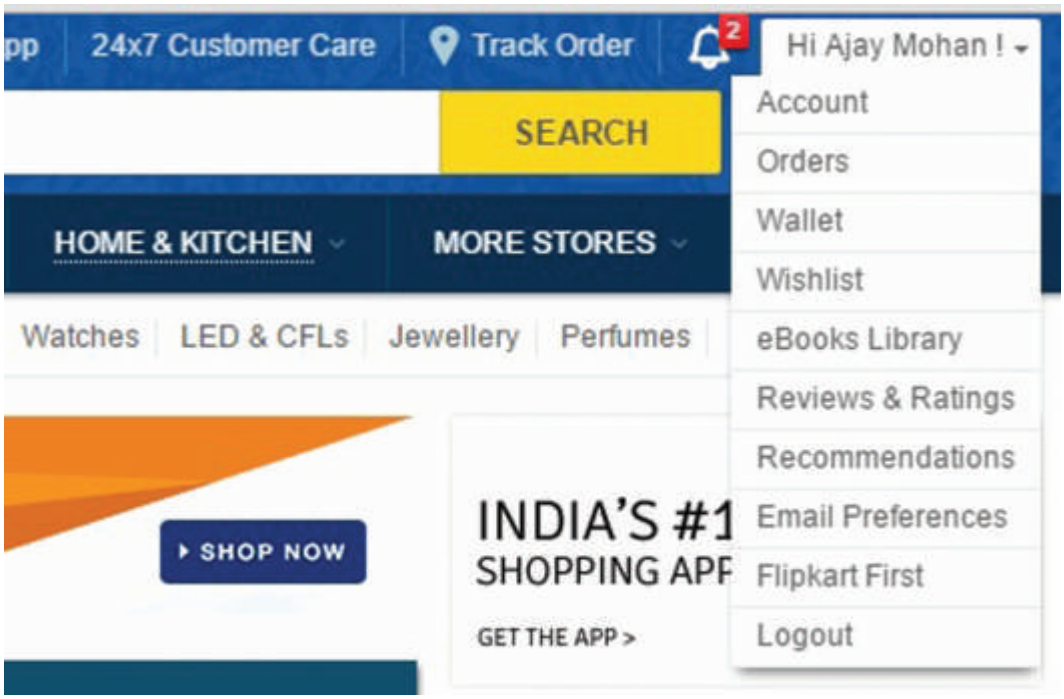


Figure 2.4(e): Flipkart: Options for logged in user

- The products in Flipkart are organized in categories like, Clothing, Footwear, Books & e-books, Mobiles and Tablets, Cameras.

- If you want to buy a mobile, bring your cursor on the Mobiles & Tablets, a window pops up Figure 2.4(f). You can choose your preference, like model, pricing and type.
- You can select different models of the product and compare them. For this, select the model and click on <Add to Compare>. Once you have selected, click on <Compare>. You can now view the comparison of the selected models Figure 2.4(g).
- The catalog can be browsed to purchase the product. Once you have decided on the item to be purchased, click on Buy Now. The product gets included in your shopping cart Figure 2.4(h).
- You can proceed to select more products before making the payment. Having added all the products you want to purchase, review your shopping cart. Then click on <Place order>.

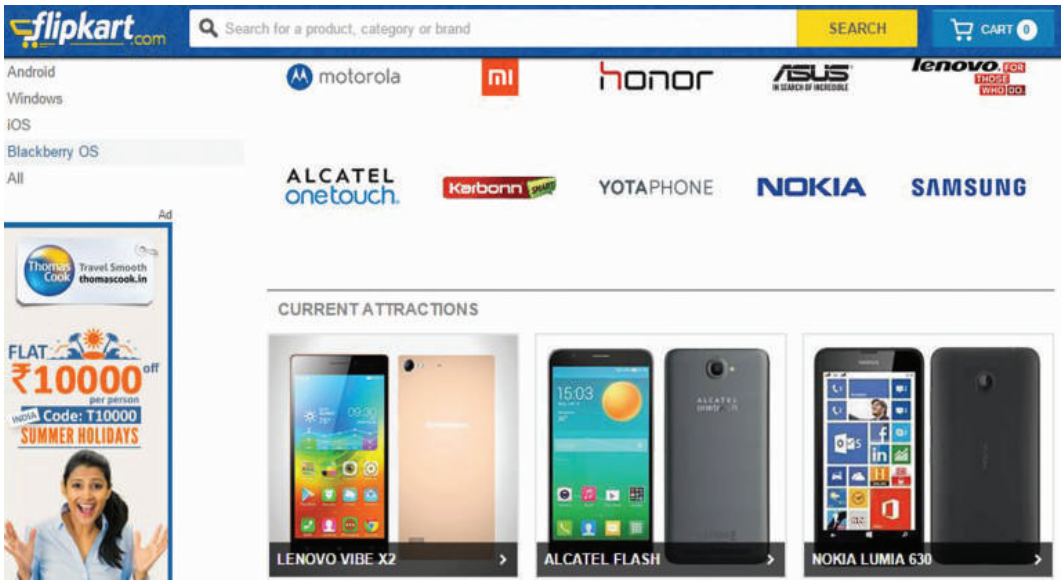


Figure 2.4 (f): Flipkart: Electronics

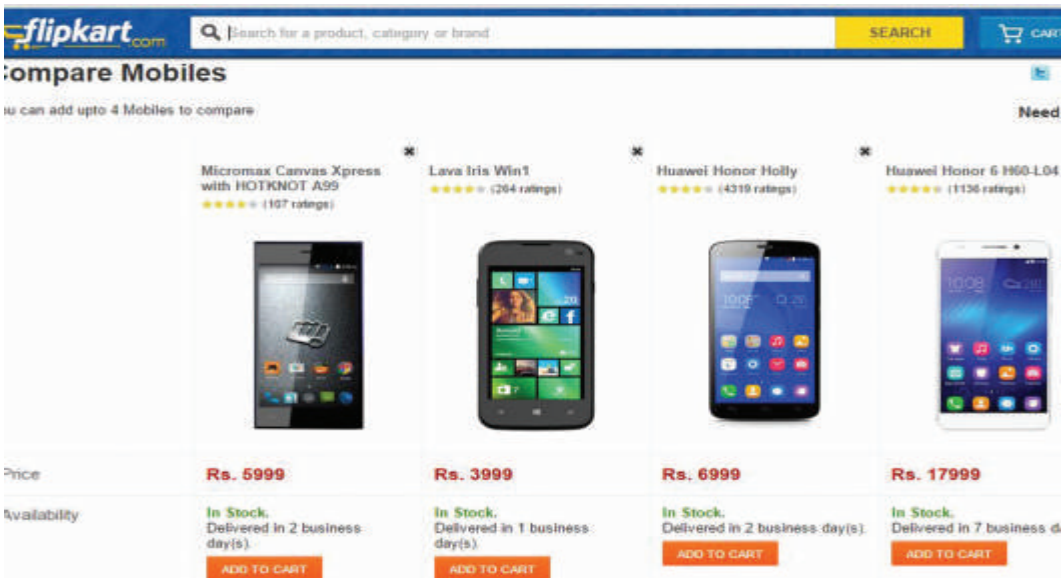


Figure 2.4(g): Flipkart: Compare Products

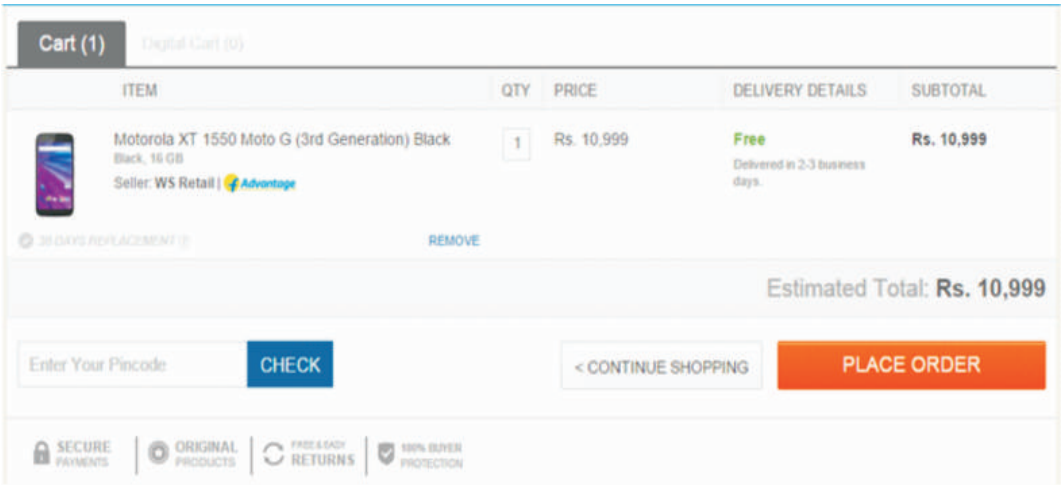


Figure 2.4(h): Flipkart: Shopping Cart

- For making payments, Flipkart allows different modes of payment - credit card, debit card, Net Banking, Cash on Delivery or e-gift voucher. Click on the method using which you want to make payment. Enter the details as required in various fields. Click <Pay> to make the payments. To complete the transaction, appropriate windows will be displayed based on the procedure to be followed for the payment.

2.5 Online Courses, Tutorials and Tests

Online courses and tutorials help the user in learning like using software, video game, or a lesson. In online tutoring, the students and the teachers are at physically different locations, connected to each other via Internet. Online learning has many benefits. It provides freedom to the learner to learn at their convenient location, convenient time and at their own pace of learning. Also, there is no discrimination of age, caste, gender, background, or the qualification of the learner. The tutorials may be enriched with audio and video information, which makes learning more enjoyable and attractive. Also, tutorials and tests are available for almost all topics, sometimes in regional languages also.

The tutorials may require different kinds of interaction with the user, for example,

- Video/slide tutorials where the video or slides of lectures are available. The learner can visit the site and view the tutorials.
- Interactive tutorials require the user to follow the instructions, perform the desired action and get a feedback.
- Webinars are real time lectures, where users participate in the lecture, may ask questions, and get their queries answered.

2.5.1 Online Educational Sites

There exist several sites for online learning like, http://www.khanacademy.org/, http://www.eagetutor.com/.

1. *Khan Academy* is an educational website created in 2006 by educator. The website

hosts video tutorials on different subjects, like, mathematics, history, physics, chemistry, civics, and economics. The learner can view the video lectures at their convenience Figure 2.5(a).

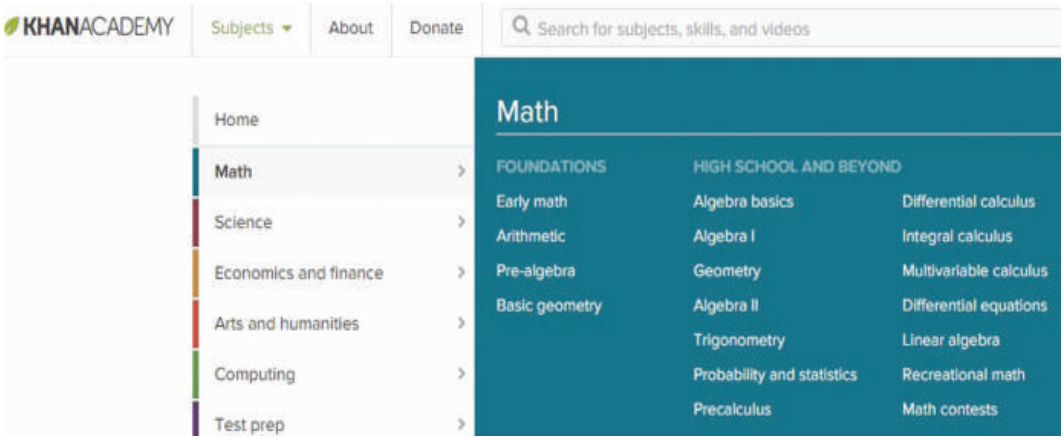


Figure 2.5(a): Khan Academy website

2. *udacity.org* is a site which offers online courses for development of technical skills. It claims to provide projects built by technical leaders like Google and AT & T. The site provides verified certificates for a fee Fig. 2.5(b).

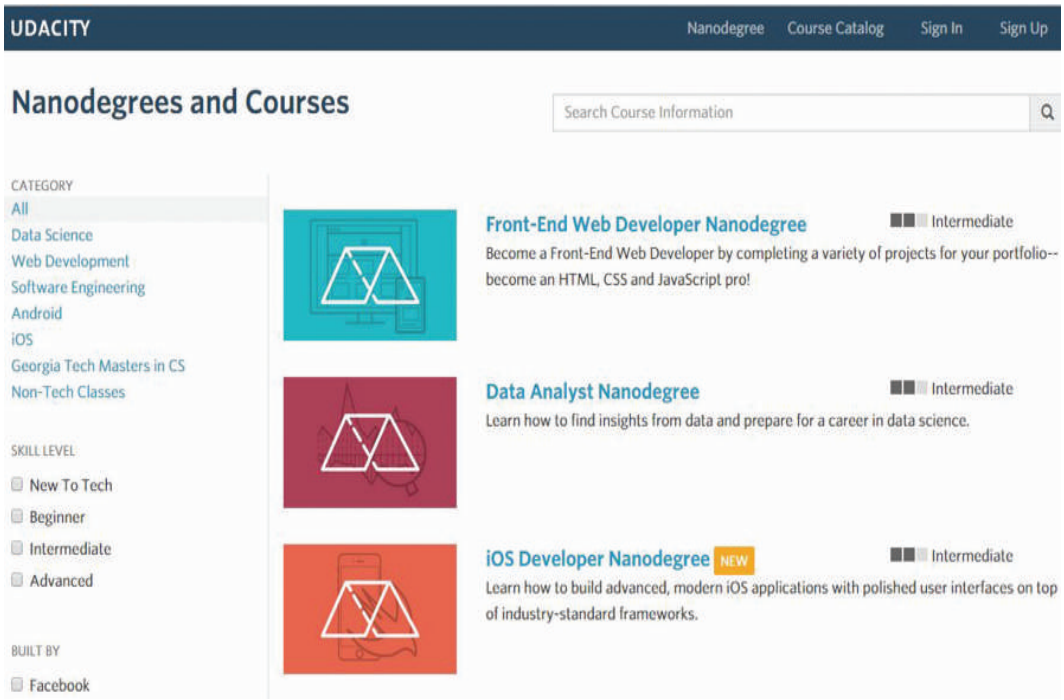


Figure 2.5(b): Udacity website

3. *coursera.org* and *edx.org* provide high quality online courses for free, in collaboration with various universities across the globe. They aim to provide free online education through its partners world wide. The courses include topics from hunanities, science, and engineering including courses at school level. These sites provide verified certificates for several courses for a fee Figure 2.5(c).

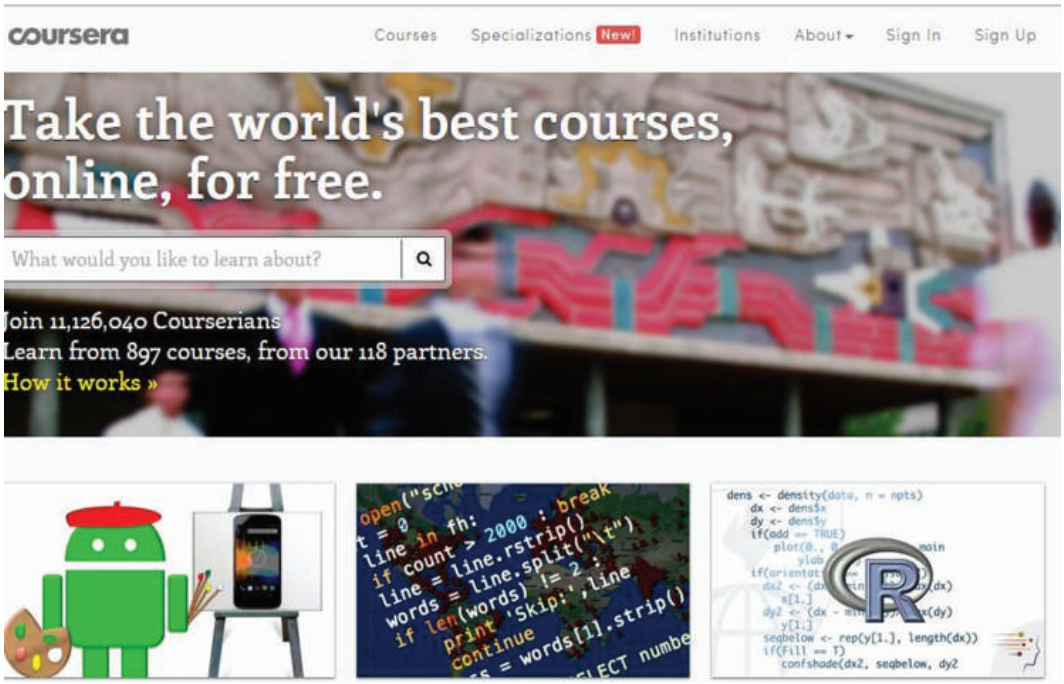


Figure 2.5(c): Coursera website

4. *w3schools.com* offers free tutorials in all web development technologies like HTML, CSS, Javascript, PHP etc. Figure 2.5(d).

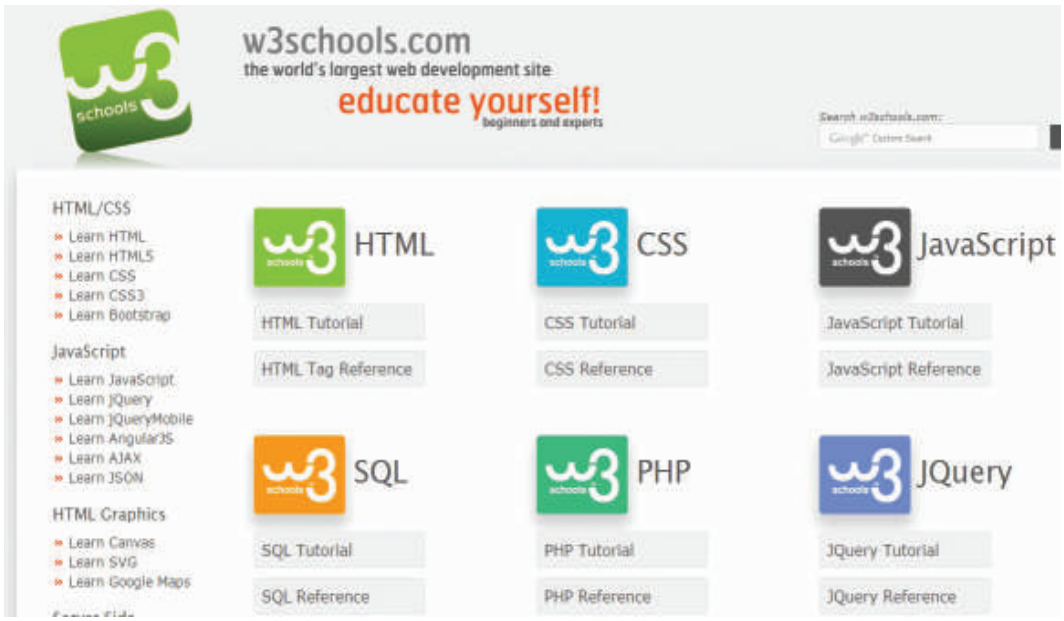


Figure 2.5(d): w3schools.com - Website

5. *GCFLearnFree.org* creates and provides quality, innovative online learning opportunities to anyone who wants to improve the technology, literacy and math skills needed to be successful in both work and life. It has 750 different lessons provided absolutely free. GCFLearnFree.org is a worldwide leader in online education.

6. *http://www.ncert.nic.in/index.html* is a NCERT portal that provides online learning resources in the form of e-books, journals, question papers, children books, etc. Fig. 2.5(e)

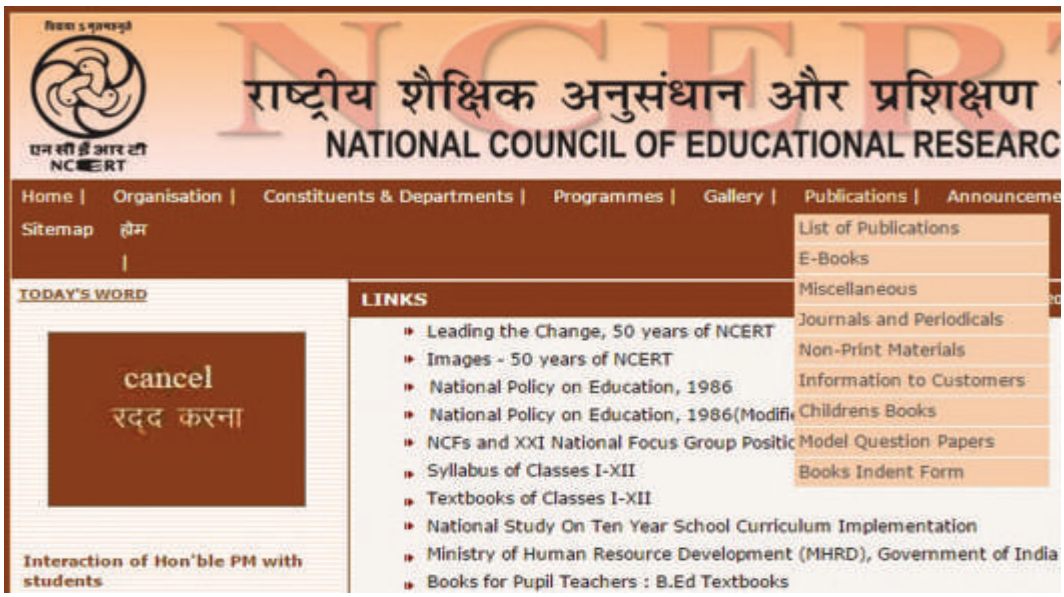
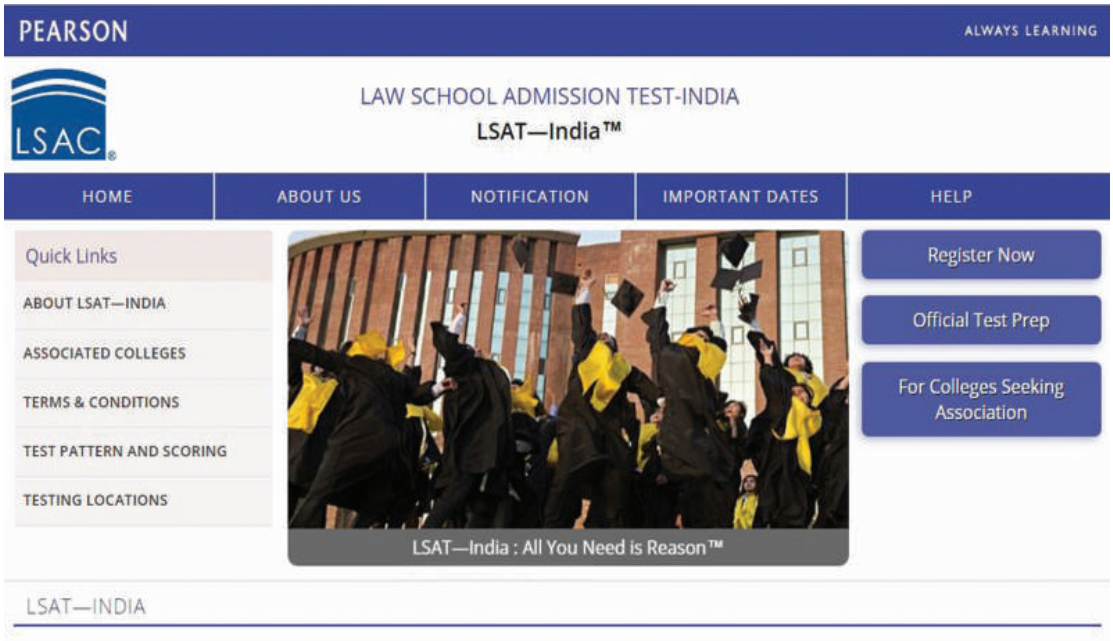


Figure 2.5(e): ncert website

In addition to learning, there are several sites that allow the students to test their knowledge. These websites hosts tests in different subjects. The learner can take a test and get evaluated. For example, http://www.british-study.com/quizzes/grammar-test.php fig. 2.5(f) is a site that allows a learner to test their English grammar, for free. http://www.pearsonvueindia.com/ lsatindia/overview.html#2 is site for standardized test of reading and verbal reasoning skills designed by the USA-based Law School Admission Council (LSAC) for use by law schools in India. This is a paid site Figure 2.5(g).

Figure 2.5(f)



The LSAT – India is a test of reasoning and reading skills, not a test to see whether you happened to have memorized the right facts or equations. The theory behind the LSAT – India is democratic and inclusive. It holds that students acquire critical thinking skills over their educational lifetimes, and that these skills are the most important for the study of law. Good critical thinking skills may be acquired in

Figure 2.5(g): Some Testing sites

2.6 Project Management - Web Based Application Development

Web based application development involves the process of developing web applications. A web application is a program that is stored on a remote server. It is online and is delivered over the Internet through a web browser interface. A web application may be of different kinds, like, a quiz, a game, or a bill calculator.

A project is a task that is undertaken to create a unique product, service, or result. Some of the key characteristics of a project are as follows:

- A project has a beginning and an end. The extent of the project is defined. It has boundaries.
- A project requires finite resources that are required to complete the project.
- A project has a specific time frame. It has a definite beginning and end dates.
- A project is complete when its end objectives are achieved. The objective is specific and identifiable.

Project Management is the process of achieving the objectives of the project through a set of activities defined within the time frame, to produce the end result. It is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements.

2.6.1 Project Essentials and Tips

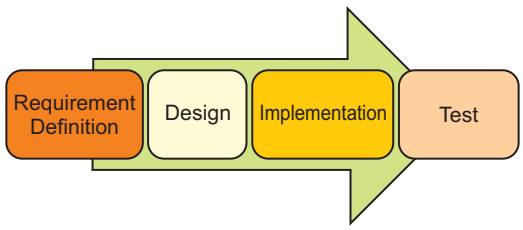
We have learnt about so many different web applications in the previous sections. In this section, a structured and simplified process is described for the development of web based applications. Let us discuss how the web applications are created.

2.6.1.1 Phases in a Web Application Project

The process of development of web based applications is broadly categorized in four phases. During the development process, one needs to follow the following four main phases of development:

1) Requirements Definition Phase
2) Design Phase
3) Implementation Phase
4) Testing Phase

Additionally, for all these phases there is a Planning and Monitoring phase. A project plan lists the activities that are required to be monitored during project execution. For each activity information useful for its monitoring is defined, like, duration, costs, schedule, resources required, and risk involved.



For developing web based applications, we discuss the steps to be performed during the four phases of development as follows:

1) **Requirement Definition Phase:** In this phase we identify the problem statement for which the web application is to be developed. Identify the scope of the problem. The following issues need to be addressed in this phase.

   (a) Feasibility: Is the project feasible?

   (i) Check whether the project is technically feasible (is it possible to do it?).

   (ii) Check whether the project is economically feasible (is it profitable?).

   (iii) If feasible, proceed further, otherwise there is no need to proceed with the project.

   (b) Scope: The focus is on **"what"** the application must do. To define the scope of the application,

   (i) Compile a detailed list with a clear description of application features.

   (ii) Establish the goals that the solution must achieve.

   (iii) Recognizing the limitations that are placed on the project.

2) **Design Phase:** The focus is on **"how"** the application is to be designed. This consists of design of:

   (a) A Map of the Web Application Site: It contains essential information about the structure of the site - the pages and the relationships between them. Thus, we

have a map with pages and a path connecting them. We have to simply implement the pages and follow the path to connect them.

(b) Database: Design the application database.

(i) Identify the database tables that will be required in the application.

(ii) Decide the table structures: For each table we need to identity the attributes of the tables, their data types, the size of the columns in the tables, the relationship between tables.

(c) Page Structure: Design the structure of page.

Identify the main sections of the page like header, main content and sidebar.

3) **Implementation Phase:** Create backend database, frontend and the connectivity between them.

a) Backend Database: Create the database and write SQL code defining tables, attributes and relationships, as per the requirement of the application.

b) Frontend: Develop the frontend of the application as per the requirement. Use the Page Structure and the Page Layout you defined in Design phase to implement the frontend code. Identify the inputs that are required to be taken from the user in the Front-End of the application. Decide the kind of buttons you will provide to the user, to minimize the user's typing effort, like, Radio buttons, checkbox, list and combo box.

c) Data Connectivity: Establish the data connectivity between the Front-End interface and Back-End Database.

4) **Testing Phase:** Test the complete application (Front-End and Back-End) with multiple sample sets of data. Find all the application bugs and fix them. Test each page of the application and test each feature of the application. Since fixing an error involves modifying the code, validate the modified code also. After final validation, the application is ready for release.

We shall now discuss three case studies that will use the project management steps of the web based application development.

2.6.2 Case Study - Online Game

First, let us define the problem statement for the online game. Here, we create an online game called Hangman. It is a simple guessing game where a player has to guess the word which may belong to a category like movie or country. Let us create this game.

Problem Statement: Hangman is a popular word game. In this game, a player selects a category like country, capitals, animals, birds and movie names. The game displays some number of blanks to the player. The player starts guessing the letter to fill in the blanks and guess the correct word. This game gives seven chances, for guessing to a user.

1. **Requirement Definition:** The problem statement is identified.

a) **Feasibility:** The students are encouraged to discuss in groups and identify the technical requirements for this project Evaluate whether it is feasible to implement within the scope of resources available in school. The following steps are based on the assumption that the project is feasible.

b) **Scope :** The application must do the following:

(i) Ask the player to select a category.

(ii) The game selects a word from that category randomly and displays the appropriate number of dashes and spaces to represent the phrase (depending on the number of letters in the word) to the player.

(iii) The player makes guesses. The guess is either a hit (successful) or a miss (fail). If it is success, the letter is written in the dash appropriately. The program should allow the user to make a total of seven guesses.

2. **Design:**

a) **A Map of the Web Application Site:** The students are encouraged to design structure of the sets by identifying the pages for this site. Determination of relationships between these pages will be crucial in design of website, for example, a page will be designed for front end interface containing leads to other pages. Implementation will follow this design structure.

b) **Database:** It is decided to create the following tables in the database, along with the attributes of the tables, their data types, the sizes of the columns in the tables:

- Category
- Words_in_Category

TABLE: Category

S. No.	Name	Type	Remarks
1	Category_Id	Varchar (10)	Unique id for each category
2	Category_Name	Varchar (20)	Name of the category

TABLE: Words_in_Category

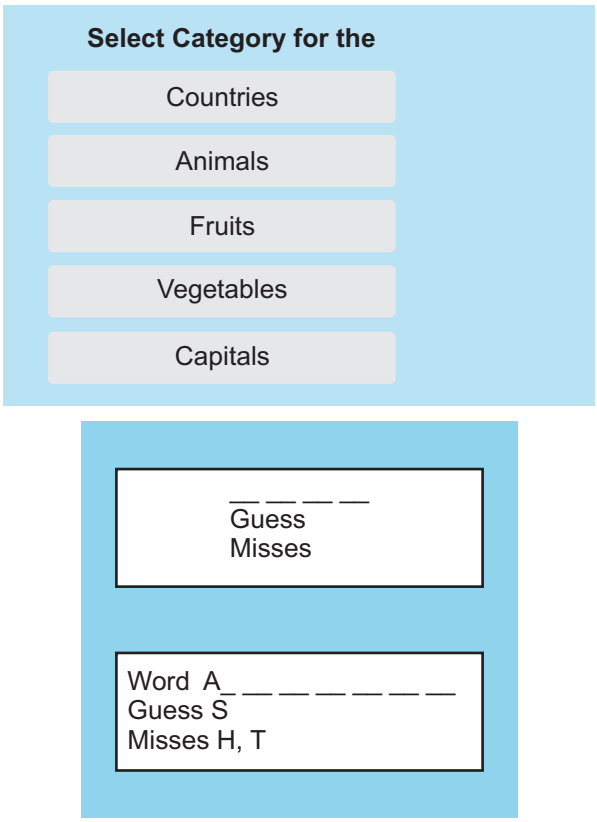
S. No.	Name	Type	Remarks
1	Category_Id	Varchar (10)	Unique id for each category
2	Word	Varchar (15)	Words for the category

c) **Page Structure:** Design the structure of page

It is found that the inputs required from the user are as follows:

- Selection of Category for the game
- Guessing the letters

The following front-end interface can be designed.



3. **Implementation:** Create backend database, frontend, and the connectivity between them.
4. **Test:** Having implemented the complete application, test the application with random data. Test each feature and functionality of the application. Fix the bugs, if found and retest.

2.6.3 Case Study - Online Quiz

Let us now define the problem statement for an online quiz. The pen-paper method of quiz is time consuming, less flexible and involves a lot of human effort in terms of correction, generating results, searching records, result analysis, etc. To solve these problems, a web-based online quiz is to be created.

Problem Statement: The online quiz web application allows a number of students to take the quiz at a time, displays the results as the test gets over. The result is generated automatically. A first time user has to register to get a login and password. The user then uses this login and password to take the test and also to see the results. Once logged in, the user can select the category for which to take the quiz. The user then attempts the quiz. When the quiz is complete, a report is generated based on the correct answers. The Administrator has the privilege to create, modify and delete the quiz questions.

1. **Requirement Definition:** The problem statement is identified.

   a) **Feasibility:** The students are encouraged to discuss in groups and identify the

technical requirements for this project and evaluate whether it is feasible to implement it within the scope of resources available in school. The following steps are based on the assumption that the project is feasible.

b) Scope: The application must do the following-

(i) Allow a user to register

(ii) Provide login and password to the user

(iii) Allow user to select a category of quiz

(iv) Display the quiz questions for the selected category, to a valid user

(v) Generate a report to provide result of the quiz.

(vi) Allow administrator to create, modify, and delete questions.

2. Design:

a) A Map of the Web Application Site: The students are encouraged to design structure of the sets by identifying the pages for this site. Determination of relationships between these pages will be crucial in design of website, for example, a page will be designed for front end interface containing leads to other pages. Implementation will follow this design structure.

b) Database: It is decided to create the following tables in the database, along with the attributes of the tables, their data types, the sizes of the columns in the tables:

- Admin_Login
- Quiz information (Here, we take two subjects)
- User_Info
- Result

ADMINISTRATOR

TABLE: Admin_Login

S.No.	Name	Type	Remarks
1	Username	Varchar (10)	Administrator username for login (Primary Key)
2	Password	Varchar (10)	Administrator password for login

TABLE: Quiz Info

S.No.	Name	Type	Remarks
1	Question_No	Number (10)	The question number
2	Subject	Varchar (10)	Subject name, e.g. English, Maths
3	Question	Varchar (40)	The quiz question

4	Option 1	Varchar (20)	1st option of quiz answer
5	Option 2	Varchar (20)	2nd option of quiz answer
6	Option 3	Varchar (20)	3rd option of quiz answer
7	Option 4	Varchar (20)	4th option of quiz answer
8	Answer	Varchar (10)	Correct answer from option1-4

TABLE: User_Info

S.No.	Name	Type	Remarks
1	Username	Varchar (10)	Name of user for login purpose (Primary Key)
2	Password	Varchar (10)	User password for login
3	Name	Varchar (10)	User name
4	Email_id	Varchar (10)	User email_id
5	Gender	Varchar (10)	User gender
6	Dob	Varchar (10)	User date of birth

TABLE: Result

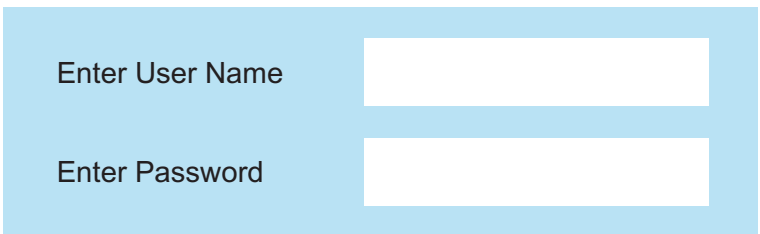
S.No.	Name	Type	Remarks
1	Username	Varchar (10)	Name of user for login purpose
2	Marks	Number (10)	Marks scored by user
3	Category	Varchar (10)	Quiz category which user attempted

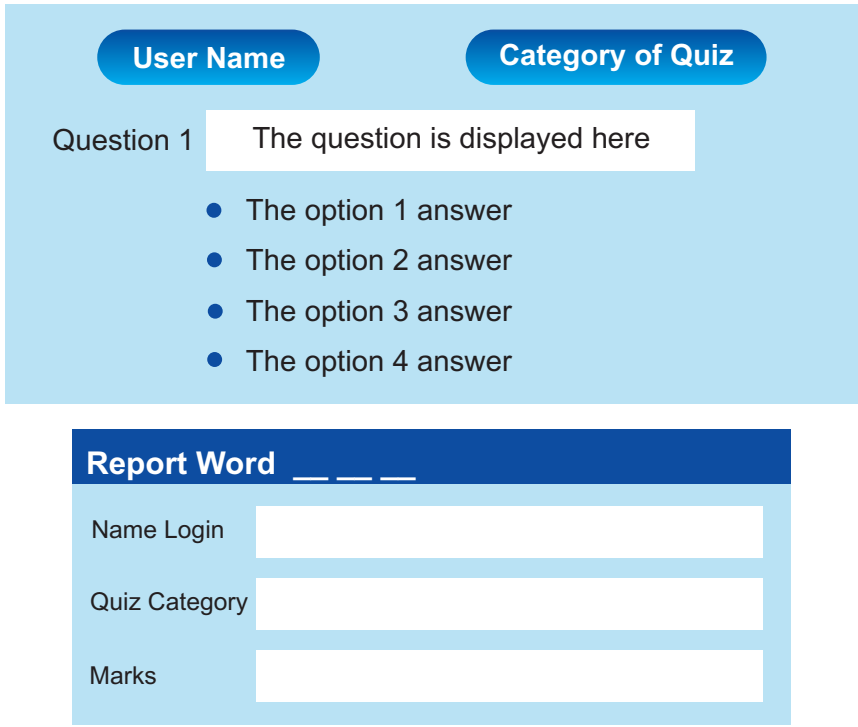
c) **Page Structure:** Design the structure of page

It is found that the inputs required from the user are as follows:

- Username and Password
- Selection of Category for which quiz is to be attempted
- The quiz is displayed to the user. The user selects the answer for each question.

The following front-end interface can be designed.





3. **Implementation:** Create backend database, frontend, and the connectivity between them.
4. **Test:** Having implemented the complete application, test the application with random data. Test each feature and functionality of the application. Fix the bugs, if found and retest.

2.6.4 Case Study - Online Bill Calculator

Let us now define the problem statement for the online bill calculator. The proper record keeping and reporting is need of the hour. The interface should be interactive, and easy to use. To provide user with a facility to calculate the bill online, a Web-based online bill calculator is to be created.

Problem Statement: The online bill calculator calculates bill for an individual user. It allows a user to enter the bill details. A first time user has to register to get a login and password. Once the user is logged in, the user can enter details of the bill and the bill is generated. The Administrator has the privilege to add, edit, or delete a user account.

1. **Requirement Definition:** The problem statement is identified.

   a) **Feasibility:** The students are encouraged to discuss in groups and identify the technical requirements for this project evaluate whether it is feasible to implement it within the scope of resources available in school. The following steps are based on the assumption that the project is feasible.

   b) **Scope:** The application must do the following:

      (i) Allow a user to register

(ii) Provide login and password to the user

(iii) Allow entry of user information

(iv) Allow entry of bill information

(v) Display the generated bill.

(vi) Allow Administrator to add, edit or delete a user.

2. Design:

a) A Map of the Web Application Site: The students are encouraged to design structure of the sets by identifying the pages for this site. Determination of relationships between these pages will be crucial in design of website, for example, a page will be designed for front end interface containing leads to other pages. Implementation will follow this design structure.

b) Database: It is decided to create the following tables in the database, along with the attributes of the tables, their data types, the sizes of the columns in the tables as shown below:

- User_info
- Bill_info

TABLE: User_Info

S.No.	Name	Type	Remarks
1	Username	Varchar (10)	Name of user for login purpose (Primary Key)
2	Password	Varchar (10)	password for login
3	Name	Varchar (20)	Name of user
4	Email_id	Varchar (40)	Address details of user
5	City	Varchar (20)	City of user
6	State	Varchar (20)	State of user
7	Country	Varchar (20)	Country of user
8	Pin_Code	Number (10)	Pin code of city
9	Mobile_Number	Number (15)	Mobile number of user

TABLE: Bill_Info

S.No.	Name	Type	Remarks
1	Username	Varchar (10)	Name of user for login purpose (Primary Key)
2	Name	Varchar (20)	Name of customer

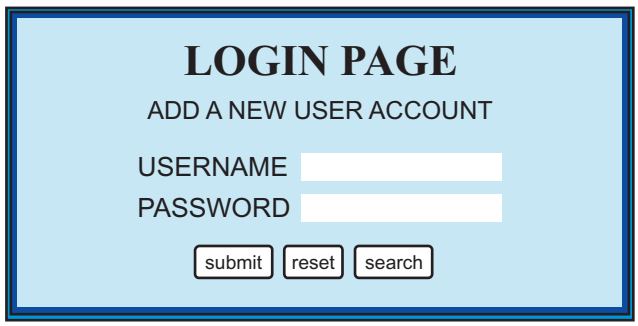
3	Address	Varchar (40)	Address details of customer
4	City	Varchar (20)	City of customer
5	State	Varchar (20)	State of customer
6	Country	Varchar (20)	Country of customer
7	Pin_Code	Number (10)	Pin code of city of customer
8	Month_Dispatch	Number (5)	Month of dispatch of material
9	Duration of Bill	Number (5)	Bill duration in days
10	Mobile_Number	Number (15)	Mobile number of user

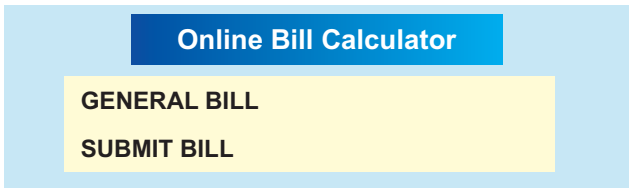
c) Page Structure: Design the structure of page

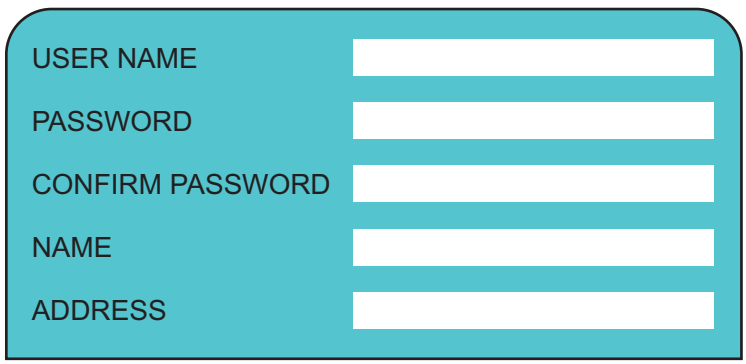
It is found that the inputs required from the user are as follows:

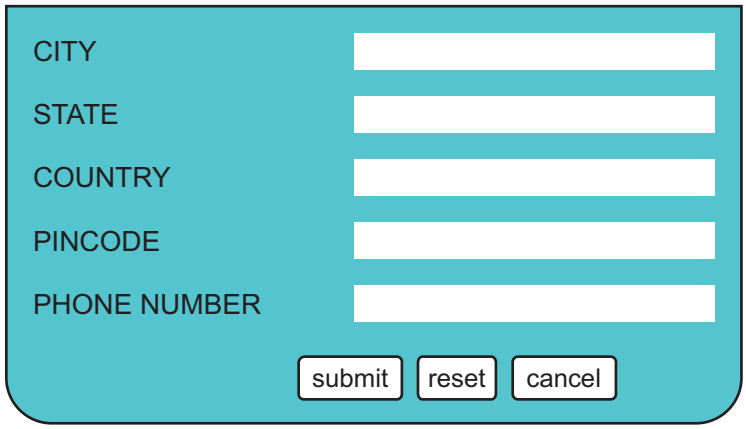
- Username and Password
- Interface to select the option of generating or submitting a bill
- Interface to add a user
- Interface to add Bill information

The following front-end interface can be designed.









NAME	
ADDRESS	
CITY	
STATE	
COUNTRY	
PINCODE	
PHONE NUMBER	
MOBILE NUMBER	
MONTH OF DISPATCH	
DURATION OF BILL	
AMOUNT	
TAX	
TOTAL AMOUNT	
LAST DATE WITH FINE	
FINE	

3. **Implementation:** Create backend database, frontend, and the connectivity between them.

4. **Test**: Having implemented the complete application, test the application with random data. Test each feature and functionality of the application. Fix the bugs, if found and retest.

Exercise:

Q1. Analyse the following scenarios and describe all the steps of the project development cycle:

(i) Online tutoring

(ii) Online matrimonial bureau

(iii) Online ticket booking

Q2. Explore the following sites for online shopping:

- Amazon.com
- Homeshop18.com
- Ebay.in

Unit - 3: Fundamentals of Java Programming

3.1 Introduction to Java

Welcome to “Fundamentals of Java Programming”! In this unit, you will learn to write and execute Java programs. But before you start writing one, it is important to know what a program is. A computer program is a sequence of instructions given to the computer in order to accomplish a specific task. The instructions tell the computer what to do and how to do it. Programming is the preparation and writing of programs for computers.

Programmers write their programs in a high level programming language such as Java, C++, Python, Ruby or Scala. However, as you are already aware, a computer only understands its own language called “machine language”.**Therefore, a compiler is needed** to translate high level program code into machine language code that will be understood by the computer. After a program is compiled, the machine code can be executed on the computer, say Windows, for which it was compiled. If the program is to be executed on another platform, say Mac, the program will first have to be compiled for that platform and can then be executed.

Java is a very popular high level programming language and has been used widely to create various types of computer applications such as database applications, desktop applications, Web based applications, mobile applications, and games among others. A Java compiler instead of translating Java code to machine language code, translates it into Java **Bytecode** (a highly optimized set of instructions). When the bytecode (also called a Java class file) is to be run on a computer, a Java interpreter, called the **Java Virtual Machine (JVM)**, translates the bytecode into machine code and then executes it. The advantage of such an approach is that once a programmer has compiled a Java program into bytecode, it can be run on any platform (say Windows, Linux, or Mac) as long as it has a JVM running on it. This makes Java programs **platform independent** and highly **portable**.

To write a Java program, you will need a Text Editor (for writing the code)and a Java compiler (for compiling the code into bytecode). There are a wide variety of **Java Integrated Development Environments (IDEs)** available in the market that come equipped with a text editor and a Java compiler thus simplifying writing, compiling and executing Java programs. Most of them are freely downloadable from the Internet. We will be using the open source and free **Java Net Beans IDE** for writing Java programs.

So let's dive straight in and write a simple Java program that prints a Welcome Message (“Hello World”) on the screen. Since this will be your first Java program, you will first need to setup the programming environment (Refer to **Appendix A** for installation and starting Net Beans).

Step 1: Double click the Net Beans icon on your computer to start Net Beans. It will open the Net Beans IDE (Figure 3.1(a).

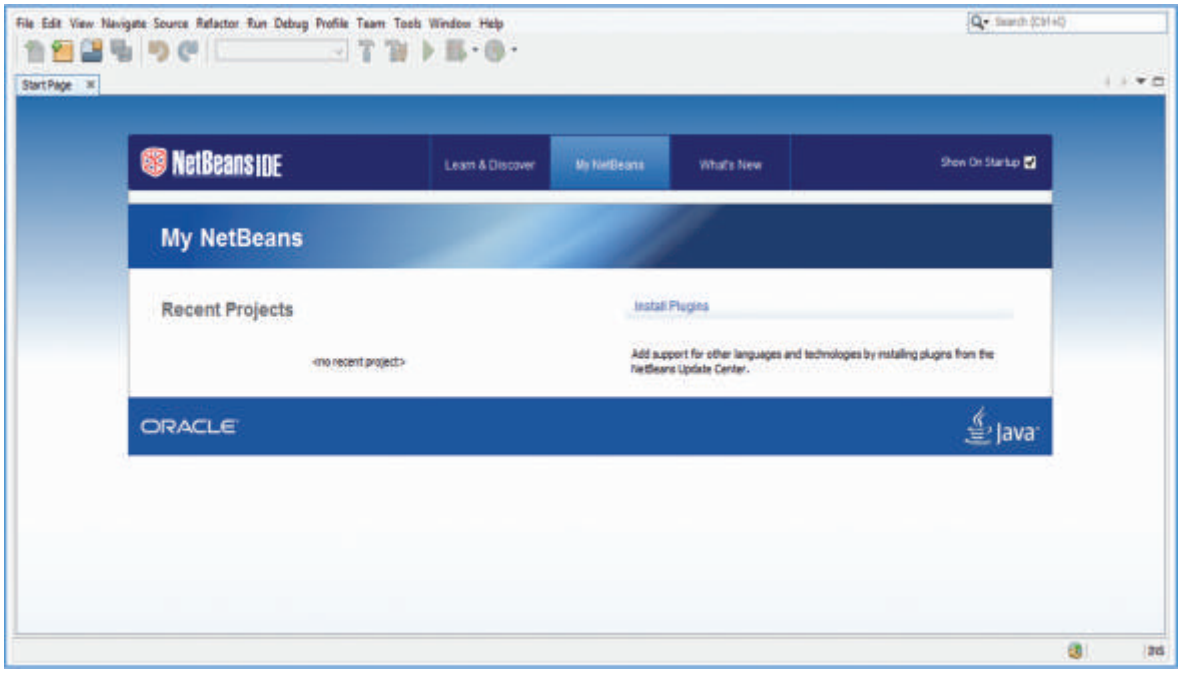


Figure 3.1(a): Java NetBeans IDE

Step 2: All software development in NetBeans is organized in the form of Projects, so we begin a new Project. In the IDE click **File> New Project (Ctrl + Shift + N)** Figure 3.1(b).

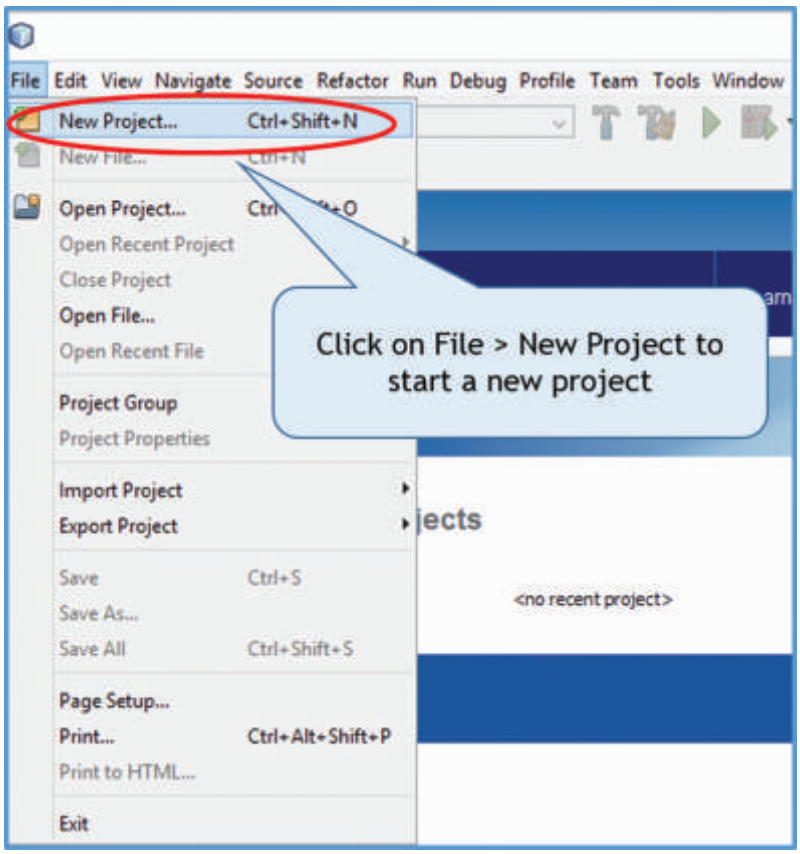


Figure 3.1(b): Create a New Project

Step 3: In the **New Project** dialog box that appears, under the **Categories** list, select **Java** and under the **Projects** list select **Java Application** (they should be already selected). Click on **Next** to create a new Java Application Project Figure 3.1(c).

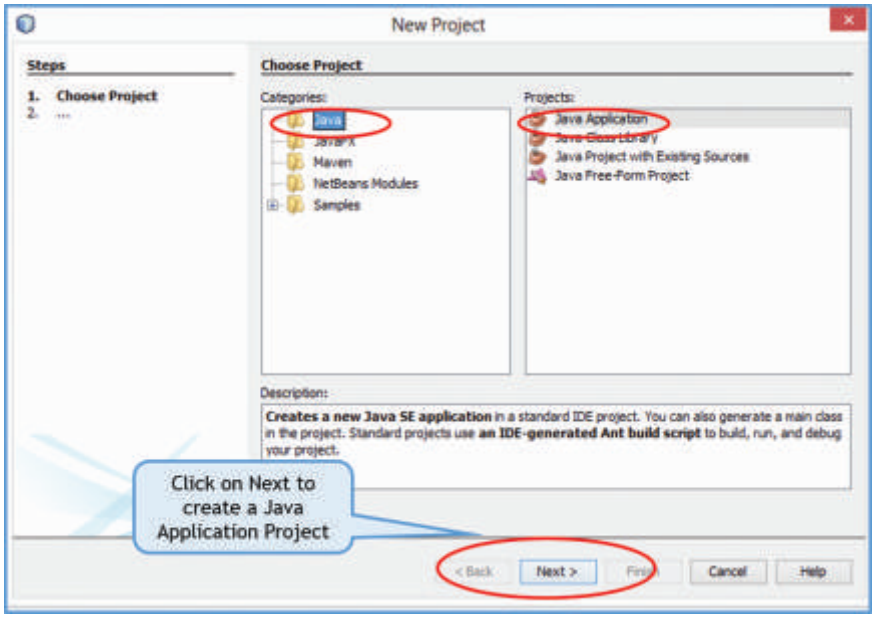


Figure 3.1(c): Create a New Java Application

Step 4: In the **New Java Application** dialog box that appears, in the **Project Name** field, type a name for the Project Figure 3.1(d). Here we have named the Project "`HelloWorld`". When you type the name of the project, the **Project Folder** field also changes automatically. So does the **Create Main Class** field. You can optionally change the **Project Location** and the **Project Folder** by clicking the **Browse** button. Click on **Finish** to finish creating the Java Application Project and to return to the IDE.

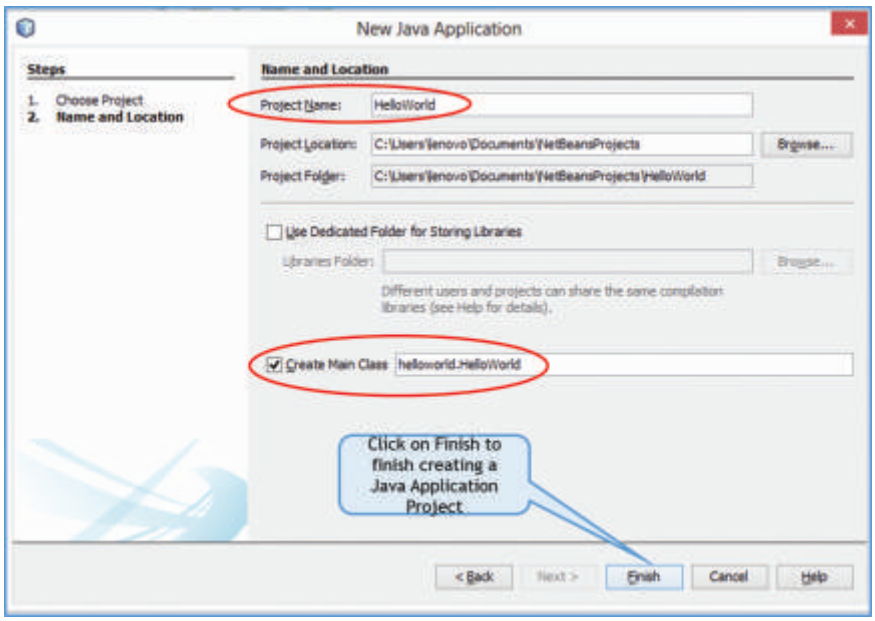


Figure 3.1(d): Finish creating new Java Application

On the left side of the NetBeans IDE, observe the **Projects** tab Figure 3.1(e). (If you can't see it, click **Windows > Projects** on the menu bar).

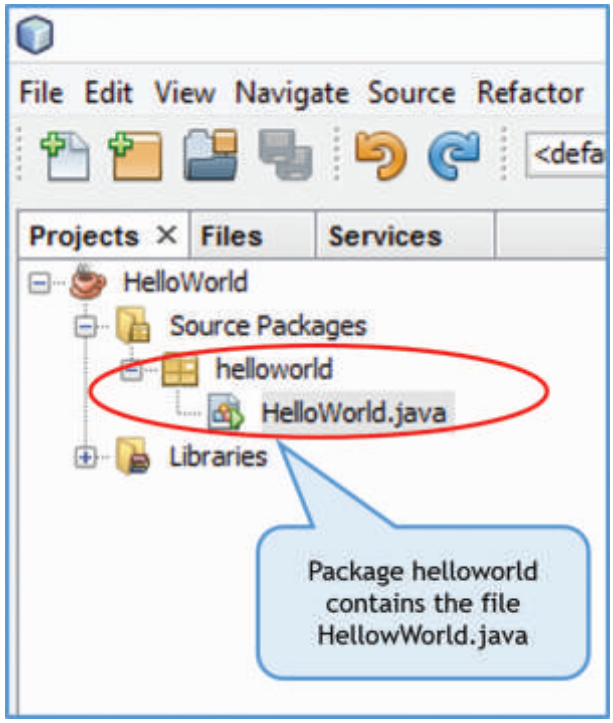


Figure 3.1(e): Projects Tab

Notice that a source code file called `HelloWorld. Java` has been created in a package called `Helloworld programs`. On the right side of the IDE is the code editor window where the Java program code in the `HelloWorld. Java` file is displayed. NetBeans has already filled in some code for you Figure 3.1(f).

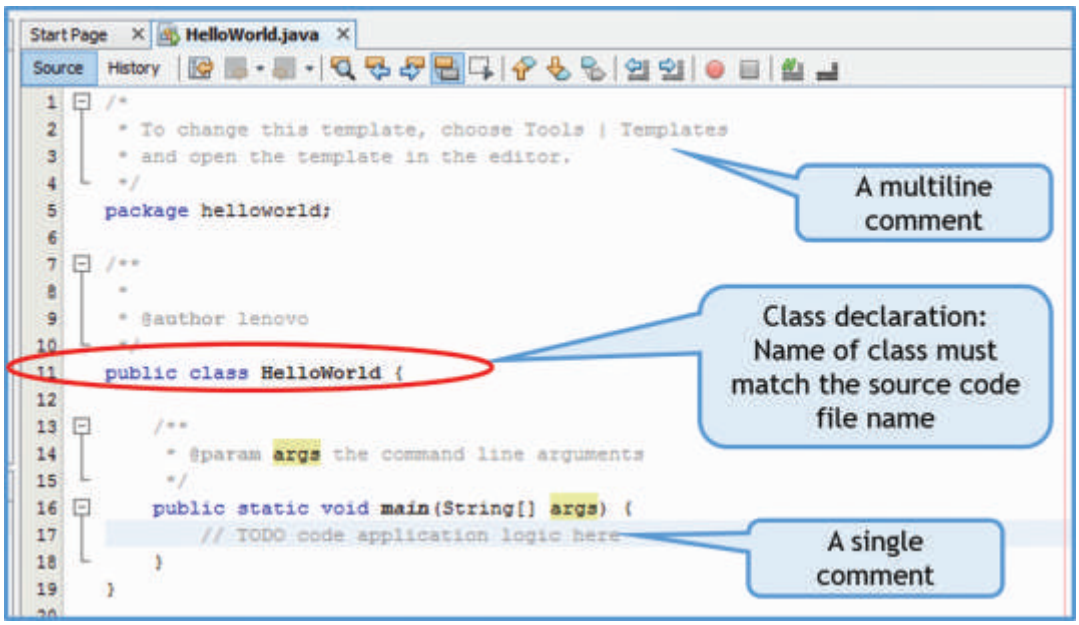


Figure 3.1(f): Code Editor Window

Observe that quite a bit of the code in the code editor is greyed out. All the grey parts are **comments**. Comments are used in code by programmers to *document* their programs - to provide explanatory notes to other people who read the code. This is especially useful in the real world, where large programs are written by one programmer and maintained by other programmers. You can write comments in a Java program in the following two ways:

- Beginning a comment line with two consecutive forward slashes (//)
- Writing the comment between the symbols /* and */

The former method is used for single line comments while the latter is generally preferred for multiple line comments. Comments are used for enhancing the readability of code and are ignored by the compiler when compiling a file.

If we ignore the comments, the `Helloworld.Java` program is as below:

```
package HelloWorld;
public class HelloWorld {
      public static void main (String[] args) {
   }
}
```

Notice that the first line in the program is a **package** statement. A package in java is a group of related classes. In our program this statement declares that the `HelloWorld`. Java program is part of the `HelloWorld` package. All classes in a package can share their data and code.

The next line declares a class called `Helloworld.Java` demands that the class name should be the same as the source file name. However the package name can be different from either the class or the source file name. NetBeans, by default, names the package with the same name as the project, except that it is in all lower case.

The contents of a class are enclosed within curly braces. Within the contents of the `Helloworld` class, a **method** called main has been declared. A method is a group of statements written to perform a specific task. The method body is enclosed within a pair of curly braces and contains the statements that the method will execute. Main is a special method that every Java application must have. When you run a program, the statements in the main method are the first to be executed. The main method in the `HelloWorld` class has only a single line comment within it.

```
// TODO code application logic here
```

In the program that we are building, we want to instruct the computer to display a welcome message in the Java output window. So within the main method we will write a single line of code that does just that. For this we will use the most common pre-built Java output method `System.out.println().`

Step 5: In the Code Editor window, find the line"// TODO code application logic here". Click on the right end of that line. Press Enter and type in the following line of code

```
System.out.println("Hello World");
```

Notice that as you type the dot after the word System followed by a dot, NetBeans will display a list of available options Figure 3.1(g).

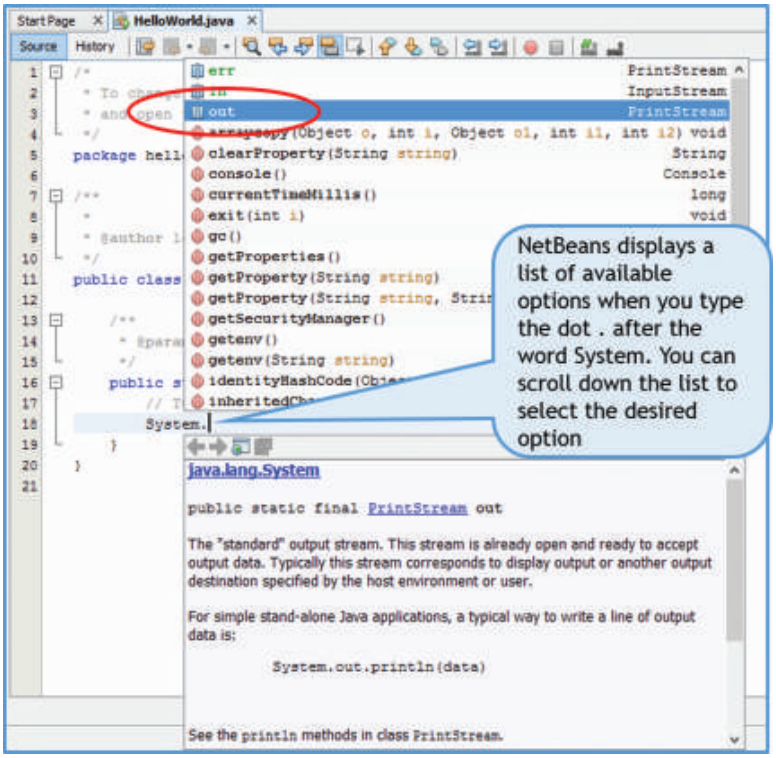


Figure 3.1(g): NetBeans Displays a List of Available Methods

At this point, you can continue to type the word **out** or alternatively, you can scroll down the displayed list to reach the word **out** and then either double click or press enter. When you type the dot after the word **out**, another list appears. Again either select `println` from the list or continue to type. After you complete the line, your code editor window should appear as in Figure 3.1(h).

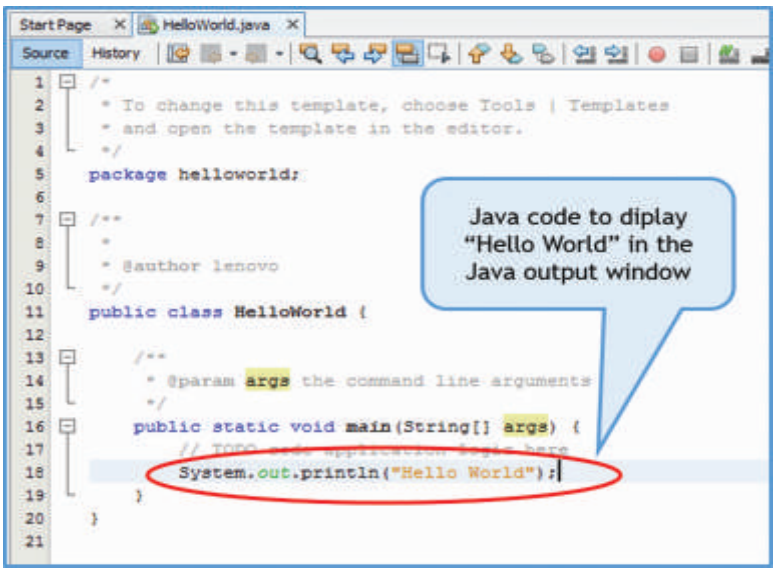


Figure 3.1(h): The Helloworld.java Program

Don't forget to type the **semicolon** at the end of the statement. All Java statements must end with a semicolon. If you make any errors while typing your code, for example, if you forget a semicolon at the end of the statement or if you misspell a keyword, NetBeans will warn you with a glyph - an exclamation mark in a red circle, in the left margin of the line Figure 3.1(i). If you bring the cursor near the warning, you will see helpful hints to correct the errors. You can use these hints to correct your errors.

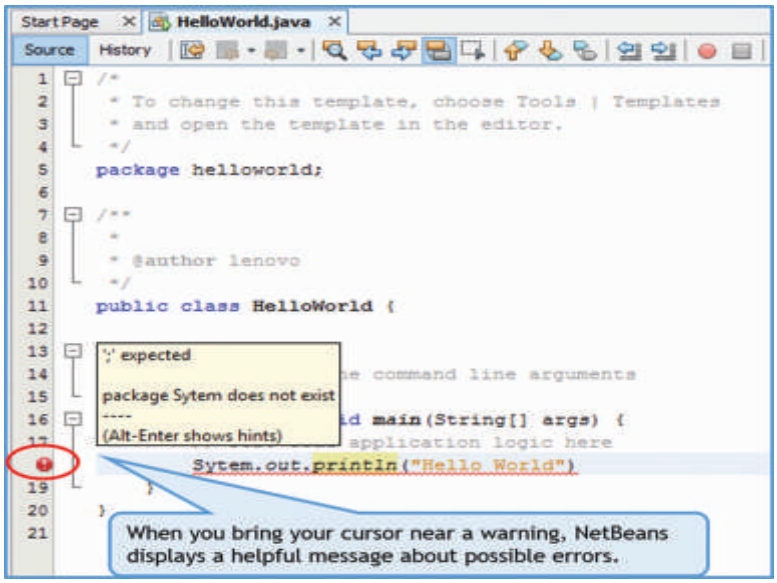


Figure 3.1(i): NetBeans Warning

Step 6: In the IDE toolbar, click on **File > Save (Ctrl + S)** or **File > Save All (Ctrl + Shift + S)** to save the `HelloWorld.Java` program Figure 3.1(j). Because NetBeans has a default **Compile on Save** feature, you do not have to compile your program manually, it is compiled automatically when you save it.

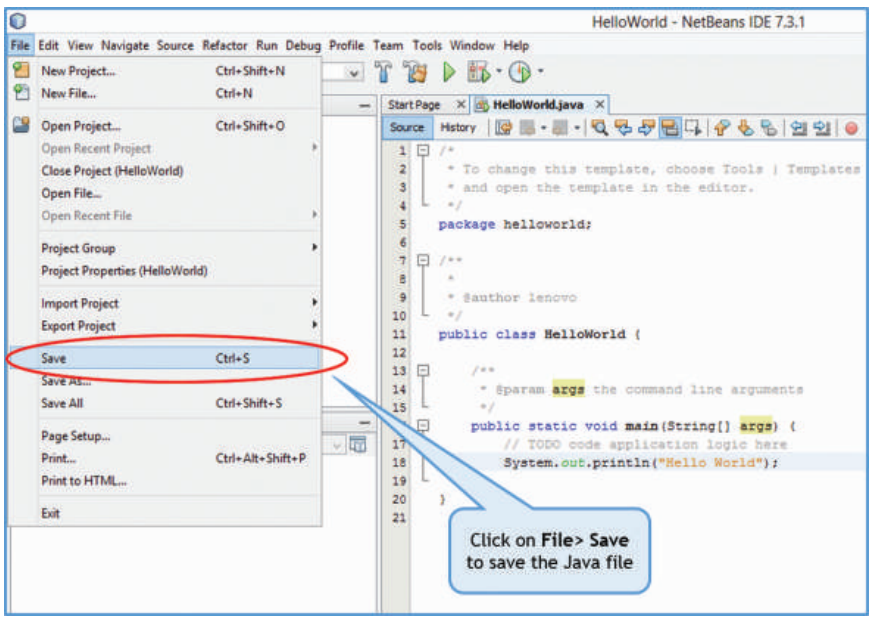


Figure 3.1(j): Saving a File

Step 7: In the IDE toolbar, click on **Run > Run Main Project (F6)**or **Run >Run File (Shift + F6)** to execute the Java Program Figure 3.1(k).

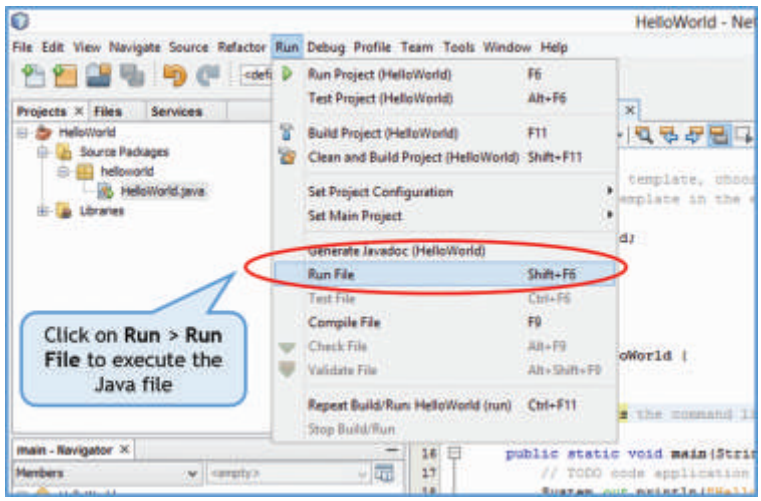


Figure 3.1(k): Executing a Java Program

You can also run your program by clicking the green arrow button on the toolbar Figure 3.1(l).

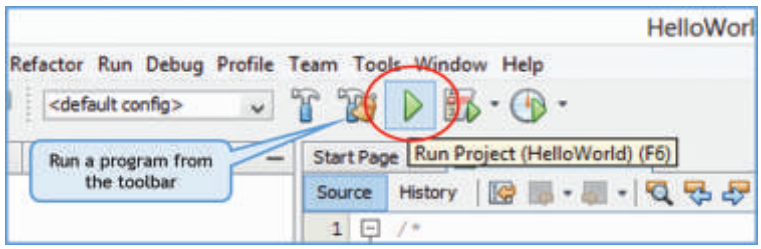


Figure 3.1(l): Executing a Java Program from the Toolbar

If there are no errors in your program, the NetBeans IDE compiles and then executes your program. You should now see the program output (the “Hello World” message) in the Output Display Window near the bottom of the IDE as in Figure 3.1(m).

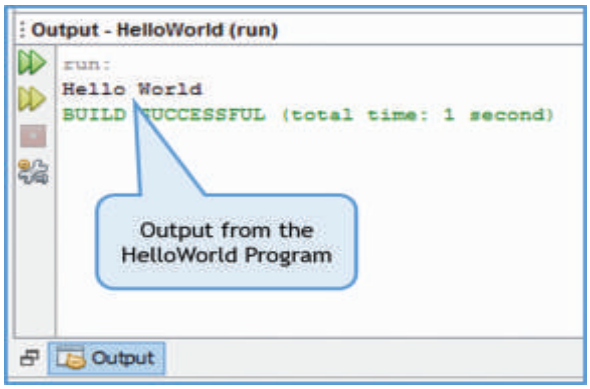


Figure 3.1(m): NetBeans Output Window

The `System.out.println("HelloWorld");` statement printed the line “`Hello World`” on the screen followed by a *newline* causing the subsequent line to be printed on the next line. Try changing the program code to

```
System.out.print("Hello World");
```

and run the program again. You should see the output as in Figure 3.1(n). Do you notice the difference from the previous run? This time a *newline* is not printed after the "`Hello World`" causing the subsequent line to be printed on the same line.

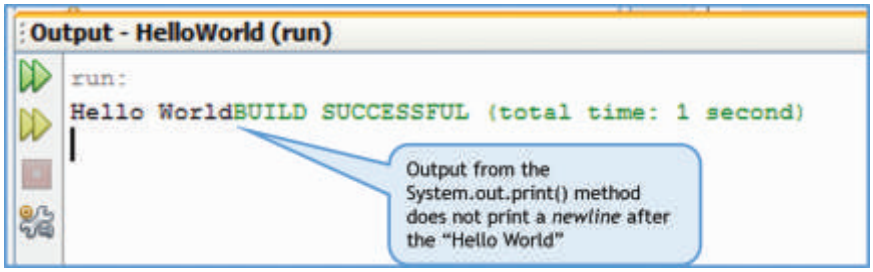


Figure 3.1(n): Output from the System.out.print() Method

Now, you have successfully written and executed your first Java Program. **Congratulations, you are now a Java programmer!**

3.2 Data Types and Variables

In the previous section, you learnt how to write and execute a simple Java program. This time let us write a program that does something a little more useful than just display a message. We will write a program that handles data. Given the marks obtained by a student and the total number of marks for an exam, our program will calculate the percentage obtained by the student. And we will calculate the percentage for two students.

3.2.1 Variables

To store the program data we will use **variables.** A variable is a placeholder for data that can change its value during program execution. Technically, a variable is the *name* for a storage location in the computer's internal memory. The *value* of the variable is the contents at that location. In our percentage calculator program, we will use three variables named `marks_obtained`, `total_marks` and `percentage` (Variable names have to be a single word, that's why we have added an underscore between the words "`marks`" and "`obtained`", "`total`" and "`marks`").

All data variables in Java have to be declared and initialized before they are used. When declaring variables, we have to specify the **data type** of information that the member will hold – integer, fractional, alphanumeric, and so on. The type of a variable tells the compiler, how much memory to reserve when storing a variable of that type.

In our percentage calculator program, the variable `total_marks` can have only integer values. So we declare it to be of type `int` – a keyword in Java that indicates an integer data type. The variables `marks_obtained` and `percentage` are declared of type **double**, since they can have fractional values (floating point numbers). Inside the main method, we declare these variables, we also assign them values using the = operator as shown below:

```
int total_marks = 400;
double marks_obtained = 346;
double percentage = 0.0;
```

To calculate the percentage, we construct an expression using the `total_marks` and `marks_obtained` variables and assign the value to the percentage variable. The * operator is used for multiplication and the / operator for division.

```
percentage = (marks_obtained/total_marks)*100;
```

To display the percentage in the IDE output window, we use our old friend – the `System.out.prinltn()` method.

```
System.out.println("Student1's Percentage = "+percentage);
```

Notice that the variable to be displayed is not put within the double quotes. To print the word "`Percentage`", we put it inside double quotes, to display the value stored in the `percentage` variable, we put it outside the double quotes. The + operator within the `System.out.println` statement, concatenates the string given in double quotes and the value of the variable.

To calculate the percentage for another student, we reuse the variables. We change the value of the variable `marks_obtained` and calculate the percentage again. There is no need to change the `total_marks` because its value remains the same as before.

```
marks_obtained = 144;
percentage = (marks_obtained/total_marks)*100;
System.out.println("Student2's Percentage = "+percentage);
```

To write the percentage calculator program in NetBeans, follow the steps given below:

Step 1: First, we will create a new Class file within the package HelloWorld (that was created in the previous section). Click on **`File > New File (Ctrl + N)`** to create a new file Figure 3.2(a).

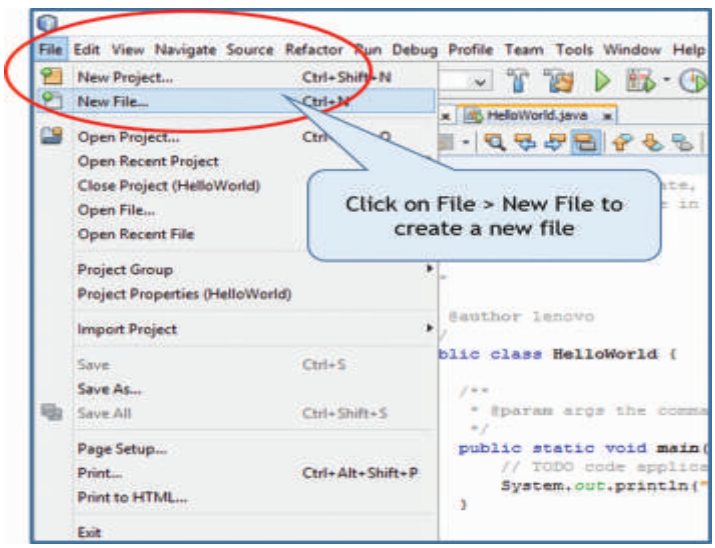


Figure 3.2(a) Creating a New File

Step 2: In the **New File** dialog box that appears, under the **Categories** list, select **Java** and under the **File Types** list select **Java Class** (they should be already selected). Click on **Next** to create a new Java Class file Figure 3.2(b).

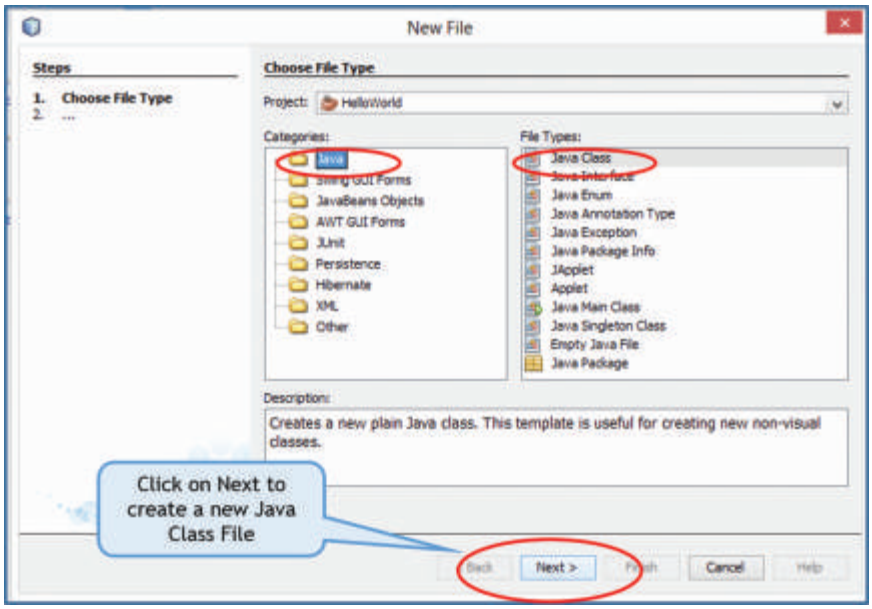


Figure 3.2(b): Creating a New Java Class File

Step 3: In the **New Java Class** dialog box that appears, type a name in the **Class Name** text box Figure 3.2(c). For our program, we type `Percentage Calculator`. When you type the name of the class, the **Created File** field also changes automatically. Make sure the **Package** field displays the `HelloWorld` package, if not click on the drop down list icon next to the field and select the `HelloWorld` package. Click on **Finish** to finish creating the New Java Class File and to return to the IDE.

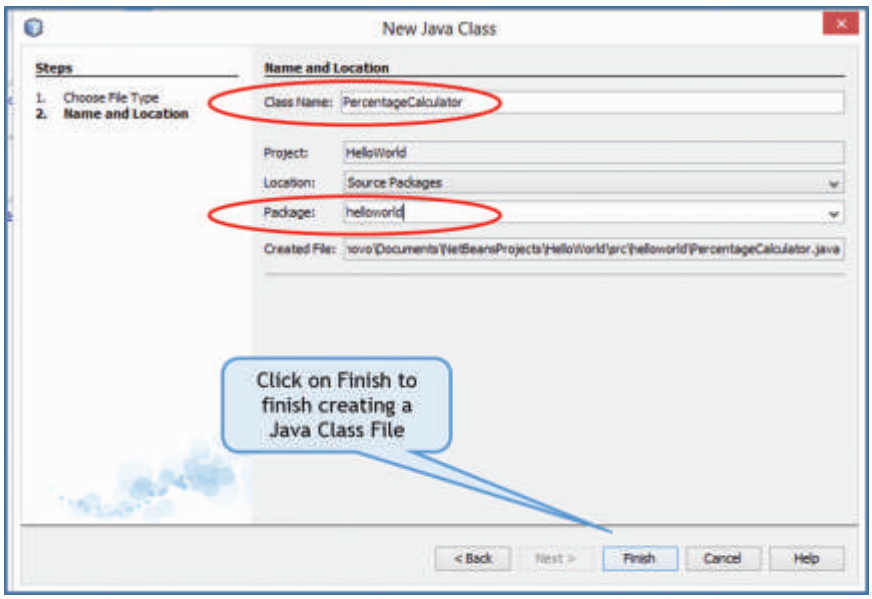


Figure 3.2(c): Finish Creating a New File

Observe that in the Projects tab, a file called `PercentageCalculator.java` has been added to the `HelloWorld` package Figure 3.2(d).

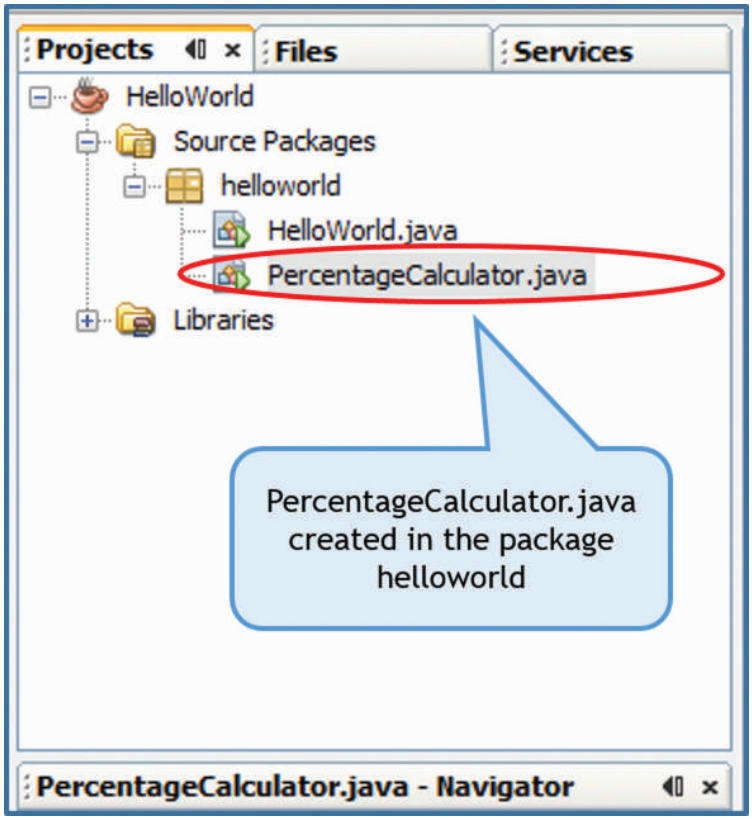


Figure-3.2(d): New File Added to the Package

Step 4: Click in the Code Editor Window under the `Percentage Calculator.Java` tab and type the code as shown in Figure 3.2(e). The comments inserted by NetBeans have been deleted for simplicity.

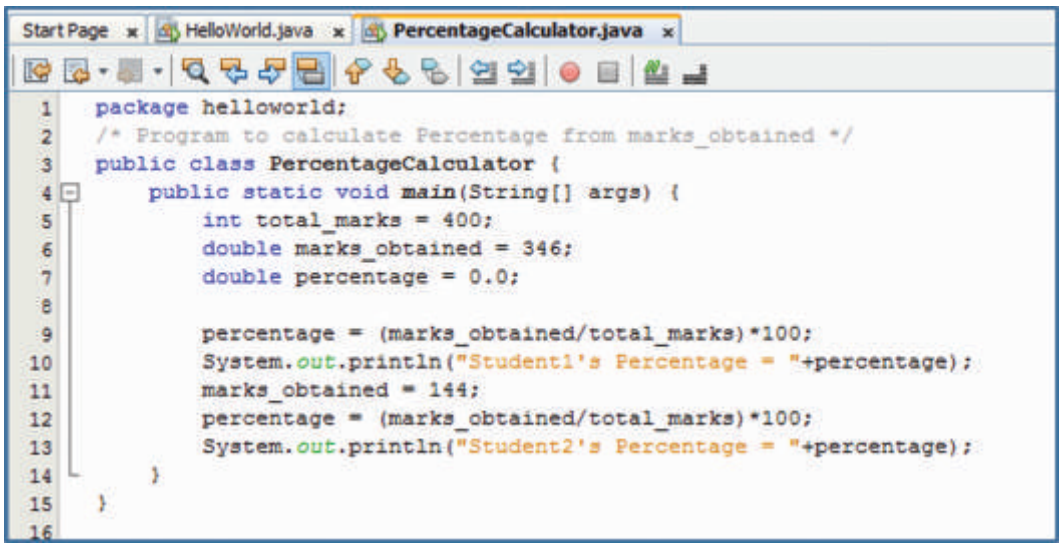


Figure 3.2(e): The Percentage Calculator Program

Step 5: Click **File > Save (Ctrl+S)**to save the file.

Step 6: Click **Run> Run File (Shift + F6)** to execute the program. The output of the program appears in the Output window as shown in Figure 3.2(f).

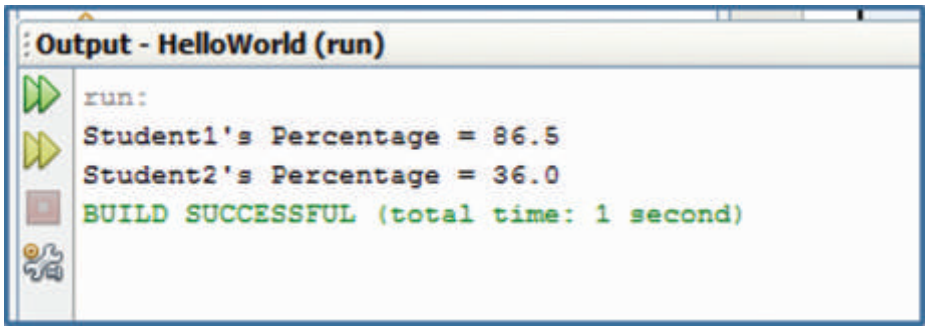


Figure 3.2(f): Output from the Percentage Calculator Program

As you may have noticed, the variables in a program act as placeholders for data handled by the program. Variables can change their value during program execution. The variables that you have seen in this section are **local variables**. Such variables are available only inside the methods where they are declared.

3.2.2 Primitive Data Types

Besides the two Java data types (int, double) that you have seen in the `Percentage Calculator` program, we can use six more data types. In all, Java supports eight primitive data types as in Table 1.

Table 1: Java Primitive Data Types

Data Type	Type of values	Size
byte	Integer	8-bit
short	Integer	16-bit
int	Integer	32-bit
long	Integer	64-bit
float	Floating Point	32-bit
double	Floating Point	64-bit
char	Character	16-bit
boolean	True or False	1-bit

STYLE TIP – Variable Names

1. Variable names can begin with either an alphabetic character, an underscore (_), or a dollar sign ($). However, convention is to begin a variable name with a letter. They can consist of only alphabets, digits, and underscore.

2. Variable names must be one word. Spaces are not allowed in variable names. Underscores are allowed. "`total_marks`" is fine but "`total marks`" is not.
3. There are some **reserved words** in Java that cannot be used as variable names, for example - int.
4. Java is a case-sensitive language. Variable names written in capital letters differ from variable names with the same spelling but written in small letters. For example, the variable name "percentage" differs from the variable name "`PERCENTAGE`" or "`Percentage`".
5. It is good practice to make variable names meaningful. The name should indicate the use of that variable.
6. You can define multiple variables of the same type in one statement by separating each with a comma. For example, you can define three integer variables as shown:

```
int num1, num2, num3;
```

3.2.3 String Variables

In the `Percentage Calculator` program of the previous section, we used variables to store numeric data. Quite often we want variables to store textual data, for example, the name of a student.

A variable of the primitive data type **char** can be used to store a single character. To assign a value to a char variable we enclose the character between single quotes.

```
char middle_name = 'M';
```

To store more than one character, we use the **String** class in Java. To assign a text value to a String variable we enclose the text between double quotes. For example,

```
String first_name = "Mayank";
String last_name = "Saxena";
```

To print char or String variables, we can use the `System.out.println()` method.

```
System.out.println(first_name+" "+ middle_name+" "+last_name);
```

The output from the statement given above is:

```
Mayank M Saxena
```

Note the use of the + operator. In Java, when used with numbers as operands, the + operator adds the numbers. When used with Strings as operands, the + operator concatenates the strings together. Also, note that the single space String literal (" ") is used so as to print a gap between the first and middle name and middle and last name.

3.3 Operators

Operators are special symbols in a programming language and perform certain specific

operations. You have already seen two operator * and /. Table 2 lists the Java Arithmetic operators, Table 3 lists the Relational operators, Table 4 lists the Assignment operators, and Table 5 lists the Logical operators available in Java.

Table 2 – Arithmetic Operators

Operator	Description	Explanation	Example (int a = 20, b = 30)	Result
+	Addition	Returns the sum of values of operands	a+b	50
-	Subtraction	Returns the difference of values of operands	a-b	-10
*	Multiplication	Returns the product of values of operands	a*b	600
/	Division	Returns the quotient of division of values of operands	b/a	1
%	Modulus	Returns the remainder of division of values of operands	a%b	10
++	Increment	Increments the operand by 1	a++ Or ++a	21
--	Decrement	Decrements the operand by 1	a-- Or --a	29

Table 3 – Relational Operators

Operator	Description	Explanation	Example	Result (int a=20, b=30)
==	equal to	Returns true if values of a and b are equal, false otherwise	a==b	false
!=	Not equal to	Returns true if values of a and b are not equal, false otherwise	a!=b	true
>	Greater than	Returns true if value of a is greater than that of b, false otherwise	a>b	false
<	Less than	Returns true if value of a is less than that of b, false otherwise	a<b	true
>=	Greater than or equal to	Returns true if value of a is greater than or equal to that of b, false otherwise	a>=b	false
<=	Less than or equal to	Returns true if value of a is greater than or equal to that of b, false otherwise	a<=b	true

Table 4 – Assignment Operators

Operator	Description	Explanation	Example	Result (int a=20, b=30)
=	Simple Assignment	Assigns value of left side operand to right side operand	a=b	a becomes 30
+=	Add and Assignment	Adds value of left side operand to right side operand and assigns the result to the right side operand Same as a = a+b	a+=b	a becomes 50 (20+30)
-=	Subtract and Assignment	Subtracts value of left side operand from right side operand and assigns the result to the right side operand Same as a = a-b	a-=b	a becomes-10(20-30)
*=	Multiply and Assignment	Multiplies value of left side operand to right side operand and assigns the result to the right side operand Same as a = a*b	a*=b	a becomes 600(20*30)
/=	Divide and Assignment	Divides value of left side operand by right side operand and assigns the quotient to the right side operand Same as a = a/b	a/=b	a becomes 0(20/30)
%=	Modulus and Assignment	Divides value of right side operand by left side operand and assigns the remainder to the right side operand Same as a = a%b	a%=b	a becomes 20(20%30)

Table 5 – Logical Operators

Operator	Description	Explanation	Example	Result (int a=20, b=30)
&&	Logical AND	Returns true if values of both a and b are true, false otherwise	a&&b	false
||	Logical OR	Returns true if values of either a or b is true, false otherwise	a||b	true
!	Logical NOT	Returns true if value of a false, true otherwise	!a	false

Java also provides Bitwise operators which are beyond the scope of this text.

3.4 Control Flow

Recall, that a program is nothing but a sequence of instructions. Java executes the instructions in sequential order, that is, one after the other. However, sometimes you might

want to execute an instruction only if a condition holds true. Or you may want to execute a set of instructions repeatedly until a condition is met. Java provides selection structures for the former and repetition structures for the latter.

3.4.1 Selection Structures

In real life, you often select your actions based on whether a condition is `true` or `false`. For example, if it is raining outside, you carry an umbrella, otherwise not.

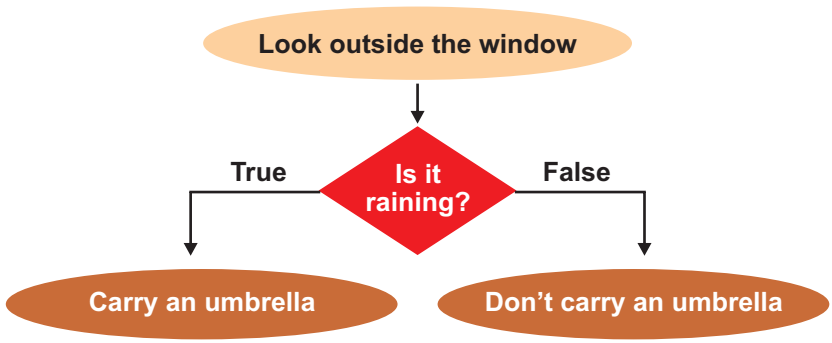


Figure 3.4(a): Selection in Real Life

Similarly in a program, you may want to execute a part of the program based on the value of an expression. Java provides two statements – the `if else` statement and the `switch` statement for executing a block of code conditionally (A block of code is a sequence of statements enclosed in curly braces).

3.4.2 The if Else Statement

The `if` statement in Java lets us execute a block of code depending upon whether an expression evaluates to true or false. The structure of the Java `if` statement is as below:

```
if (expression) {
   statements
}
```

The expression given in the round brackets after the if keyword, is a condition – an expression constructed using relational and logical operators. If the condition evaluates to true, the block of code following `if` is executed, otherwise not. If there is only one statement in the block after the `if`, the curly braces are optional. Let us enhance our `PercentageCalculator` program. If the percentage secured by the student is greater than or equal to 40%, we will declare the student as passed. So, in our percentage calculator program, we write,

```
if (percentage >= 40) {
   System.out.println ("PASSED");
}
```

If we also want to display "FAILED" when the student has scored less than 40%, we use the **if else** statement. The structure of the Java if else statement is as shown below:

```
if(expression) {
   statements
}
else {
   statements
}
```

As before, the block of code following `if` is executed if the conditional expression evaluates to `true`. The block of code following the **else** is executed if the conditional expression evaluates to false. In our `Percentage Calculator` program, we can write,

```
if (percentage >= 40) {
   System.out.println("PASSED");
}
else {
   System.out.println("FAILED");
}
```

Sometimes, you may want to execute a block of code based on the evaluation of two or more conditional expressions. For this, you can combine expressions using the logical `&&` or `||` operators. Let's say, we want to print the division with which the student has passed, then, we can write the following code. The first if statement uses the logical AND (`&&`) operator to combine two relational expressions.

```
if ((percentage >= 40)&& (percentage < 60)) {
   System.out.println("PASSED with II Division");
}
if (percentage >= 60) {
   System.out.println("PASSED with I Division ");
}
if (percentage < 40) {
   System.out.println("FAILED");
}
```

`if-else` statements can also be nested – you can have another `if-else` statement within an outer `if` or `else` statement. The example below is a nested if.

```
if (percentage >= 40) {
   System.out.println("PASSED");
   if (percentage>= 75) {
      System.out.println("With Distinction!");
   }
```

```
}
else {
   System.out.println("FAILED");
}
```

You can see the complete code listing of the new Percentage Calculator program in Figure 3.4(b) and its output in Figure 3.4(c).

Start Page | HelloWorld.java | PercentageCalculator.java | NewPercentageCalculator.java

```
1  package helloworld;
2  public class NewPercentageCalculator {
3      public static void main(String[] args) {
4          int total_marks = 400;
5          double marks_obtained = 346;
6          double percentage = 0.0;
7
8          percentage = (marks_obtained/total_marks)*100;
9          System.out.println("Student1's Percentage = "+percentage);
10         if (percentage >= 40) {
11                 System.out.println("PASSED");
12                 if (percentage  >= 75) {
13                    System.out.println("With Distinction!");
14                 }
15         }
16         else {
17                 System.out.println("FAILED");
18           }
19         marks_obtained = 144;
20         percentage = (marks_obtained/total_marks)*100;
21         System.out.println("Student2's Percentage = "+percentage);
22         if (percentage >= 40) {
23                 System.out.println("PASSED");
24                 if (percentage  >= 75) {
25                    System.out.println("With Distinction!");
26                 }
27         }
28         else {
29                 System.out.println("FAILED");
30           }
31     }
32 }
```

Figure 3.4(b): New Percentage Calculator

Output - HelloWorld (run)

```
run:
Student1's Percentage = 86.5
PASSED
With Distinction!
Student2's Percentage = 36.0
FAILED
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 3.4(c): Output from New Percentage Calculator

3.4.3 The Switch Statement

The switch statement is used to execute a block of code matching one value out of many possible values. The structure of the Java switch statement is as follows:

```
switch (expression) {
case constant_1 : statements;
                  break;
case constant_2 : statements;
                  break;
…
…
default         : statements;
                  break;
}
```

Within the `switch` statement, as you can see, there can be many case statements. The expression is compared with the constants in each case group and the matching case group is executed. If the expression evaluates to some `constant = constant_1`, the statements in the case group `constant_1` are executed. Similarly, if the expression evaluates to `constant_2`, the statements in the case group `constant_2` are executed. The `break` statement after each case group terminates the `switch` and causes execution to continue to the statements after the `switch`. If there is no match for the expression with any case group, the statements in the `default` part are executed.

The expression in the `switch` statement must evaluate to `byte, short, int, or char`.

The program code below demonstrates usage of the switch statement.

```
public class SwitchDemo {
   public static void main (String[ ] args) {
      int today = 5;
      String day = "";
      switch (today) {
          case 1: day = "Monday"
                  break;
          case 2: day = "Tuesday";
                  break;
          case 3: day = "Wednesday";
                  break;
          case 4: day = "Thursday";
                  break;
          case 5: day = "Friday";
                  break;
```

```
            case 6: day = "Saturday";
                    break;
            case 7: day = "Sunday";
                    break;
            default: day = "Incorrect Day!
                    break;
        }
        System.out.println (day);
    }
}
```

Figure 3.4(d) displays the output from the `SwitchDemo` program. Since the expression in the `switch`, is the variable `today` which is equal to 5, the case corresponding to the constant 5 is matched. The statements in the case 5 are executed which set the variable day to "`Friday`". The break after the assignment causes the `switch` to terminate. Next, execution of the statement after the `switch`, i.e. the `System.out.println()` statement takes place which prints the value of the variable day.

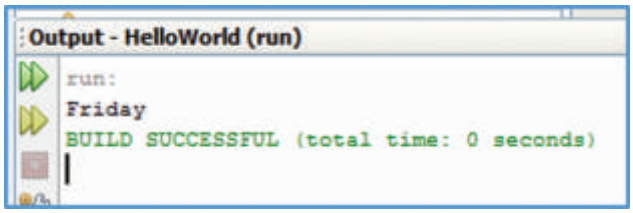


Figure 3.4(d): Output from Switch Demo Program

Can you guess what the output would be if the variable day was set to 10? Since 10 does not match any of the case statements, the default case would be executed and the output would be "`Incorrect Day!`".

3.4.4 Repetition Structures

In real life you often do something repeatedly, for example, consider a task such as reading a book, first you open the book, and then repeatedly - read a page; flip the page – until you get to the end of the book, then close the book.

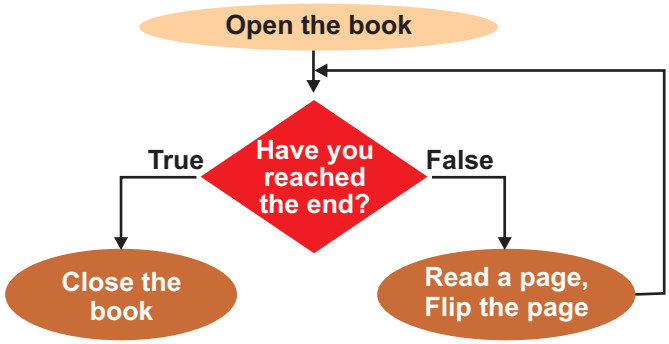


Figure 3.4(e): Repetition in Real Life

Similarly, when writing programs, you might need to perform the same sequence of statements repeatedly until some condition is met. The ability of a computer to perform the same set of actions again and again is called **looping**. The sequence of statements that is repeated again and again is called the **body** of the loop. The **test conditions** that determine whether a loop is entered or exited is constructed using relational and logical operators. A single pass through the loop is called an **iteration**. For example, a loop that repeats the execution of the body three times goes through three iterations.

Java provides three statements – the `while` statement, the do `while` statement, and the for statement `for` looping.

3.4.5 The While Statement

The `while` statement evaluates the test before executing the body of a loop. The structure of the Java `while` statement is as shown:

```
while (expression)
{
   statements
}
```

Let us write a program to print the squares of numbers from 1 to 5. The following steps need to be performed.

1. Initialize number = 1
2. Test if the number <= 5
3. If yes, print the square of the number;
   Increment number (number = number + 1)
   Go back to step 2
4. If no, exit the loop.

The tasks that need to be performed repeatedly are in step 3 – these constitute the body of the loop. The test condition is in step 2. The corresponding Java code is given below:

```
public class WhileDemo {
   public static void main (String[ ] args) {
      int number = 1;
      while (number <= 5) {
          System.out.print ("Square of " + number);
          System.out.println (" = " + number*number);
          ++number;
      }
   }
}
```

Note the use of the ++ operator to increment the value of number. The statement `++number` or `number++;` has the same effect as the statement `number = number + 1;`

The output from the `WhileDemo` program is displayed in Figure 3.4(f).

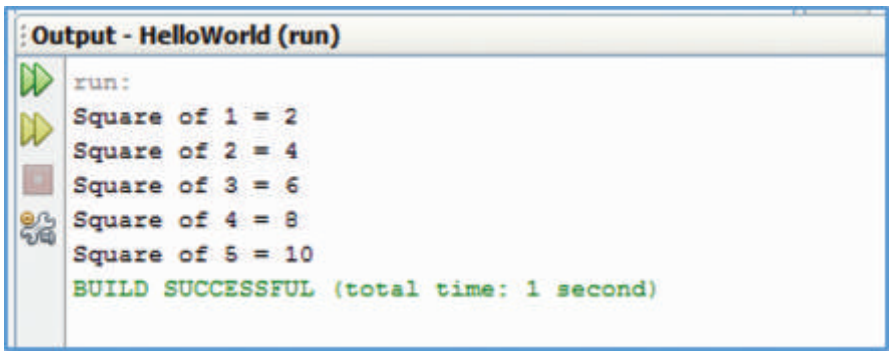


Figure 3.4(f): Output from While Demo Program

STYLE TIP: LOOPS

1. Indent your loops, begin the statements in the body of the loop with a tab/ spaces inside the curly braces. This will make the code easier to read, understand and debug. The NetBeans IDE automatically provides code indentation.

Indented Code	Non Indented Code

`int number = 1;`
`while (number >=10)`
`{`
`  System.out.println(nu`
`  mber);`
`  number = number + 1;`
`}`
	`int number = 0;`
	`while (number >=0)`
	`{`
	`System.out.println(numb`
	`er);`
	`number = number + 1;`
	`}`

2. If there is only one statement in the body of the loop, the set of curly braces enclosing the body can be omitted. For example,

```
while ( number < another_number)
   ++number;
```

3.4.6 The Do While Statement

The do `while` statement evaluates the test after executing the body of a loop. The structure of the Java do `while` statement is as shown:

```
do
{
   statements
} while (expression);
```

Let us write the same program to print the squares of numbers from 1 to 5. This time we will use a do-while loop. The following steps need to be performed.

1. Initialize number = 1
2. Print the square of the number
3. Increment number (number = number + 1)
4. Test if the number <= 5
5. If yes, Go back to step 2
   If no, Exit the loop

The tasks that need to be performed repeatedly are in steps 2 and 3 – these constitute the body of the loop. The test condition is in step 4.The corresponding Java code is as below:

```
public class DoWhileDemo {
   public static void main (String[ ] args) {
   int number = 1;
   do {
      System.out.print ("Square of " + number);
      System.out.println (" = " + number*number);
      ++number;
   }  while (number <= 5);
}
}
```

The output of the `DoWhileDemo` program is the same as Figure 3.4f. You might be wondering if both the `while` and `do-while` loops give the same output, what is the use of having two types of loop constructs. Well, consider the case when the variable number is initialized to 6.

```
int number = 6;
```

The `while` loop would first test whether number `<= 5`. Since `6 > 5`, the condition is `false` and the loop body would not be entered and as a result nothing is displayed in the output window. However in the case of the do-while loop, since the body of the loop is executed before the test condition, the square of 6 is printed `("Square of 6 = 36")`, then, number is incremented to 7. Now the test for number >= 5 is made. Since `7 > 5`, the condition is false and the loop is exited. To summarize, the `do-while` loop always executes at least once whereas the `while` loop executes zero or more times.

Table 6 summarizes the difference between a `while` and a `do-while` loop.

Table 6 – Difference between while and do-while loop

`while` LOOP	`do while` LOOP
A `while` loop is an entry controlled loop – it tests for a condition prior to running a block of code	A `do-while` loop is an exit control loop - it tests for a condition after running a block of code

A `while` loop runs zero or more times Body of loop may never be executed	A `do-while` loop runs once or more times but at least once
The variables in the test condition must be initialized prior to entering the loop structure.	Body of loop is executed at least once It is not necessary to initialize the variables in the test condition prior to entering the loop structure.

`while (condition)`
`{`
`  statements`
`}`
	`do {`
	`  statements`
	`} while (condition);`

Common Coding Errors: Loops

1. Infinite Loops

This type of error occurs when the loop runs forever, the loop never exits. This happens when the test condition is always true.

For example,

```
int number = 1;
while (number <= 5)
{
   System.out.print("Square of " + number);
   System.out.println(" = " + number*number);
}
```

This loop will run forever since number never changes – it remains at 1.

2. Syntax Errors

- Do not forget the semi colon at the end of the test condition in a `do-while` loop. Otherwise you will get a compiler error.
- Do not place a semi colon at the end of the test condition in a `while` loop.

For example,

```
int x = 1;
while ( x <=5 );
   x = x +1;
```

This loop is an infinite loop - The semicolon after the while causes the statement to be repeated as the null statement (which does nothing). If the semi colon at the end is be removed the loop will work as expected.

3.4.7 The for Statement

The for loop is the most widely used Java loop construct. The structure of the Java for statement is as below:

```
for (counter=initial_value; test_condition;change counter)
{
    statements
}
```

Semicolons separate the three parts of a for loop:

- The *initial_value* initializes the value of the loop counter.
- The *test_condition* tests whether the loop should be executed again. The loop is exited when the test condition fails.
- The *step* updates the counter in each loop iteration.

For example, consider the following loop that prints the square of numbers from 1 to 5:

```
for (int number = 1; number<= 5; ++number)
{
    System.out.print("Square of "+ number);
    System.out.println(" = "+ number*number);
}
```

When the loop begins, the variable number is set to 1. Then the *test_condition* tests whether the loop variable, number <=5. If it is, the body of the loop is executed – the square of 1 is printed. The *step* then increments count by 1. The *test_condition* is tested again to decide whether to execute the body. If it is less than 5, the square of the variable number is printed. When number exceeds 5, the loop is exited. The output from the program is the squares of numbers from 1 to 5.

In English, we read the loop above as follows:

for number is equal to 1; number is less than or equal to 5; execute the body of the loop; increment number; loop back to the test condition.

The loop given above counts up the loop index. It is an **incrementing** loop. We can also count down in a loop for a **decrementing** loop. The following decrementing loop will execute 5 times.

```
for (int number = 5; number >= 1; number = number-1)
{
    System.out.print("Square of " + number);
    System.out.println(" = " + number*number);
}
```

The loop index can be incremented or decremented by any value, not just 1. The following loop increments the loop index by 2. It displays all odd numbers between 1 and 20.

```
for ( int count = 1; count <= 20; count = count +2)
{
    System.out.println(count);
}
```

The loop index can begin with any value, not necessarily 1. The following loop also iterates 5 times and prints number from 5 to 9.

```
for ( int count = 5; count < 10; count++)
{
    System.out.println(count);
}
```

The counter in the loop iterates with count values – 5,6,7,8,9 for a total of five times. Notice that the test condition is count < 10 instead of count <= 10. If it was the latter, the loop would have iterated 6 times for loop index count as 5,6,7,8,9,10.

STYLE TIP: For LOOP

1. If there is only one statement in the body of the loop, the set of curly braces enclosing the body can be omitted. For example,

```
for ( int count = 5; count < 10; count++)
    System.out.println (count);
```

2. You can use multiple items in the initialization and the updation part of the loop by separating them with the comma operator.

   For example,

```
for ( x = 0, y = 10; x < 10; x++, y--)
    System.out.println("x = " + x + "y = " + y);
```

3. Do remember to use indentation to align the body of the loop, it will make it easier for you to debug your code.

Common Coding Errors: The for Loop

1. Initial value is greater than the limit value and the loop increment is positive.

   For example,

```
for ( int count = 5; count <= 1; count++)
```

   In this case, body of the loop will never be executed

2. Initial value is lesser than the limit value and the loop increment is negative.

   For example,

```
for ( int count = 1; count >= 5; count --)
```

   In this case also, body of the loop will never be executed.

3. Placing a semicolon at the end of a for statement:

```
for ( int count = 1; count <= 5; count++);
```

```
{
    //body of loop
}
```

This has the effect of defining the body of the loop to be empty. The statements within the curly braces will be executed only once (after the for loop terminates) and not as many times as expected.

4. Executing the loop either more or less times than the desired number of times.

   For example, the following loop iterates 4 times and not the intended 5 times because it exits when count = 5.

   ```
   for ( int count = 1; count < 5; count ++)
   ```

   The correct way to loop five times would be to test for count **<=** 5.

   Such errors are known as off by one errors.

5. Using a loop index (declared within the loop) outside the loop.

   ```
   for ( int count = 1; count < 5; count ++)
   {
       System.out.println(count);
   }
       System.out.println(count);    //error!!
   ```

   The scope of the variable count is only within the body of the loop. It is not visible outside the loop.

3.5 Arrays

Previously you learned about variables that are place holders for a single value. Arrays are variables that can hold more than one value, they can hold a list of values of the same type. For example, to store the marks of a student, we used a variable:

```
double marks_obtained = 346;
```

To store the marks of another student, we changed the variable `marks_obtained` as follows.

```
marks_obtained = 144;
```

By changing `marks_obtained`, we have lost information about the first student's marks. What if we wanted to save the first student's marks for later processing? We would need two variables, say marks1 and marks2. But, what if there were 5 students in the class. We would need 5 variables!

```
double marks1 = 346, mark2 = 144, marks3 = 243,
marks4=256.5, marks5 = 387.5;
```

Java provides a simpler way to store the marks of 5 students. We use an array and define it as below:

```
double[] marks;
```

The [] brackets after the double data type tell the Java compiler that instead of a single value, the marks variable is an array that can hold more than one value, each of type `double`. To specify that the array can hold up to 5 marks, we specify the size of the array using the keyword new as shown below:

```
marks = new double[5];
```

The two statements – declaring an array and specifying its size can also be done in one statement.

```
double[] marks = new double[5];
```

Now we can store five marks in the array, each element of the array is indexed by its position, starting from 0. So the five elements of the array are available at positions 0 to 4 as given below:

```
marks[0], marks[1], marks[2], marks[3], marks[4]
```

Note that the array index goes from 0 to n-1 for an array of size n and not from 1 to n. We can initialize the array **statically** (that is at compile time) as shown below:

```
double[]marks = {346, 144, 103, 256.5, 387.5};
```

The statement above creates an array of double of size 5, and elements are assigned with the numbers given in the curly braces. Components of an array may be assigned value using an assignment statement. For example,

```
inti i = 3;
marks[3] = 210.75;
```

To print all the marks, we can use a `for` loop, varying the loop index from 0 to 4.

```
for (int i = 0; i< 5; i++)
   System.out.println(marks[i]);
```

An array variable has a useful property, the `length property`, which is the size of the array. For example, for the marks array, `marks.length` will be 5. We can thus code the previous for loop to print all the elements of an array as:

```
for (int i = 0; i <marks.length; i++)
   System.out.println(marks[i]);
```

The following code shows the use of an array to print the class report card for the five students.

```
for (int i = 0; i <marks.length; i++) {
   double percentage = (marks[i]/total_marks)*100;
   String result;
   if (percentage >= 40)
      result = "Passed";
   else
      result = "Failed";
   System.out.print((i+1)+"\t");
```

```
System.out.print(marks[i]+"\t");
System.out.print(percentage+"\t\t");
System.out.println(result);
}
```

Notice the use of `\t` to print a tab between the numbers in the print statement. Also note that the roll number of the student has to be printed as (i+1) since the array index begins from 0 but roll numbers begin from 1. Figure 3.5(a) shows the complete program listing and Figure 3.5b displays the output from the `ArrayDemo` program.

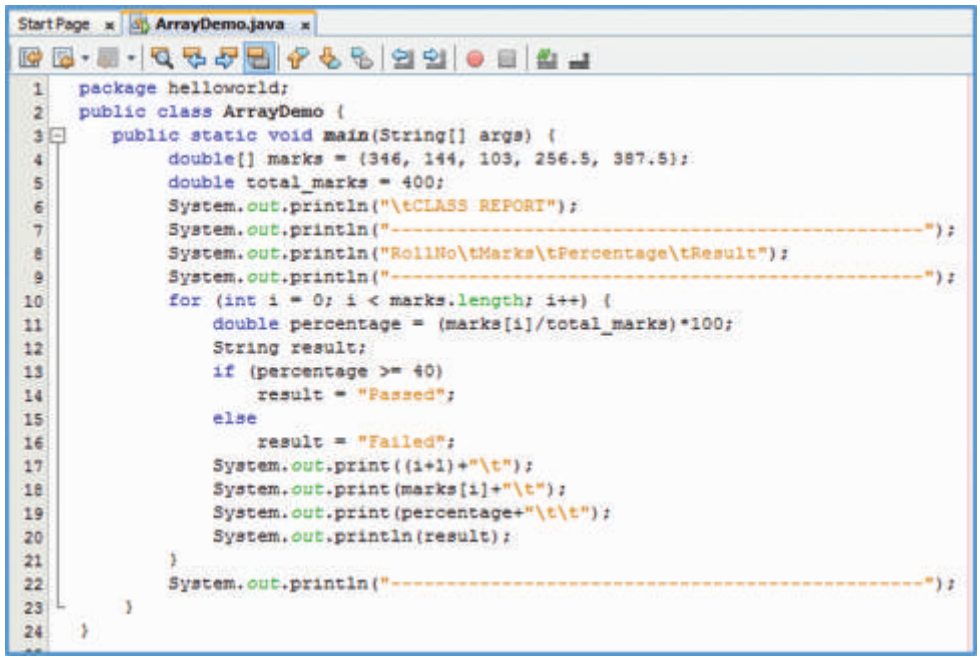


Figure 3.5(a): Array Demo Program

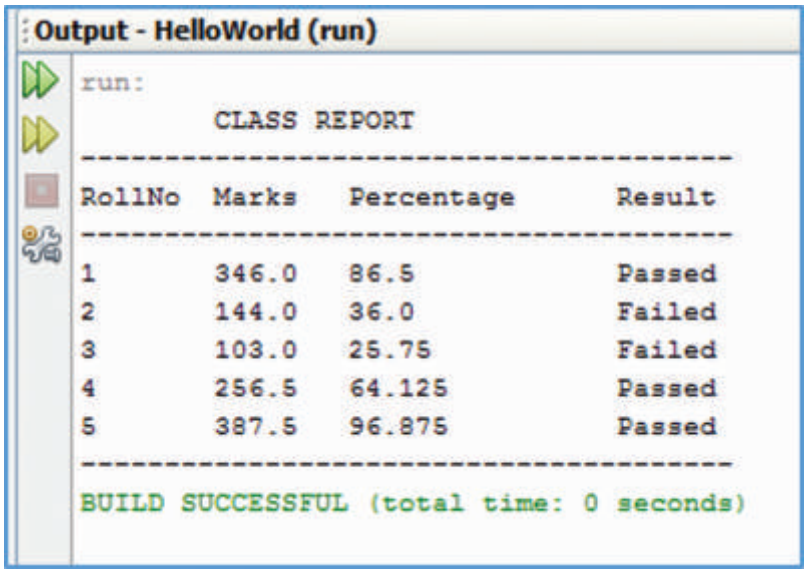


Figure 3.5(b): Output from Array Demo Program

Common Coding Errors: Arrays

1. In an off by one error, the loop index if varied from 0 to n instead of 0 to n-1. For example, if the array size is 5, then `loop index = 5` is an off by one error since array index can go only from 0 to 4.

```
double [] ] marks = new double [5];
for (int i = 0; i <= 5; i++) {
   System.out.print(marks[i]);
}
```

In this case an Array index out of bounds error occurs and the program terminates unexpectedly with an error as below.

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
at javaprograms.ArrayDemo.main(ArrayDemo.java:11)
Java Result: 1
```

3.6 User Defined Methods

A method in Java is a block of statements grouped together to perform a specific task. A method has a *name*, a *return type*, an optional *list of parameters*, and a *body*. The structure of a Java method is as below:

```
return_type method_name(list of parameters separated by commas)
{
    statements
    return statement
}
```

The method name is a word followed by round brackets. The parameter list placed within the round brackets allows data to be passed into the method. This list is a comma separated list of variables (as many as needed) along with their data types. The variables of the parameter list act just like regular local variables inside the method body. The method body is enclosed within a pair of curly braces and contains the statements of the method. The statements end with an optional `return` statement that returns a value back to the calling method. The return type is the type of the value that the method returns.

Let us write a method that given the `length` and `breadth` of a rectangle as parameters returns the `area` of the rectangle.

```
static double rectangle_area (double length, double breadth)
{
    return (length * breadth);
}
```

The name of the method is `rectangle_area`. The method has two parameters – both of type `double`. The body of the method has only a single statement, one that calculates the area based on the parameters and returns it. The return type from the method is of type

`double`. We have declared this method as a **static** method. The `static` modifier allows us to call this method without an invoking object. (You will learn more about objects later).

A method is called/invoked from another method. When a method is called, control is transferred from the calling method to the called method. The statements inside the called method's body are executed. Control is then returned back to the calling method.

Let us call/invoke the `rectangle_area` method from the `main` method.

```
public static void main(String[] args) {
    double a = 0;
    a = rectangle_area(45.5, 78.5);
    System.out.println("Area of the rectangle = "+ a);
}
```

When we call the `rectangle_area` method, we supply the length and breadth of the rectangle as **arguments** in the method call. The arguments with which the `rectangle_area` method is called are copied into its parameters. In this case, 45.5 is copied into the parameter `length` and 78.5 is copied into the parameter `breadth`.

```
double rectangle_area (double length, double breadth)

    double a = rectangle_area (45.5, 78.5);
```

The method is executed and the return value from the method is copied into the variable a. In this case, a gets the value 3571.75, the product of 45.5 and 78.5.

```
double rectangle_area (double length, double breadth)
3571.75
    double a = rectangle_area(45.5, 78.5);
```

The `System.out.prinltn()` then displays the area in the output window.

```
Area of the rectangle = 3571.75
```

If we can call the `rectangle_area` method with different arguments and we will get a different result.

Figure 3.6(a) shows the complete program listing of the `MethodDemo` program and Figure 3.6(b) displays the output from the program.

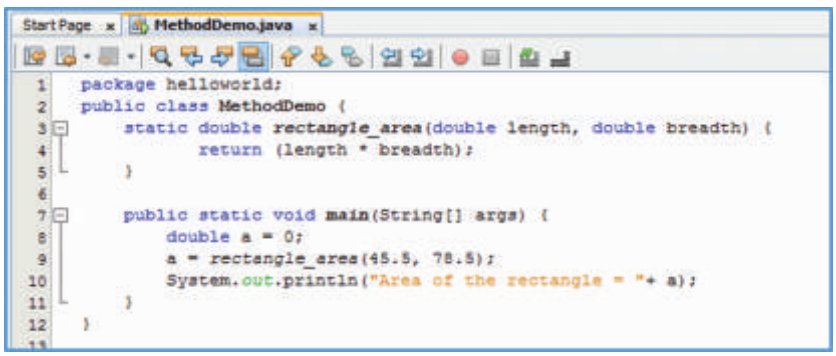


Figure 3.6(a): User Defined Method Demo Program

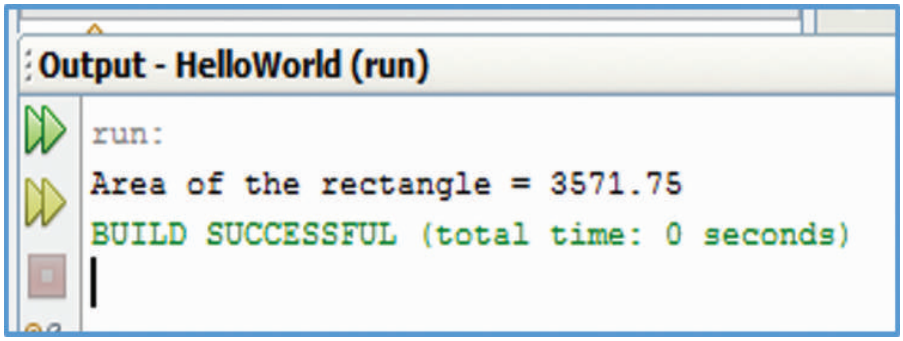


Figure 3.6(b): Output from User Defined Method Demo Program

A special return type `void` can be used if a method does not return any value. For example, a method that just prints a string need not have a return value can use `void` as the return type.

```
void print_message(String message) {
   System.out.println("The message is: "+ message);
}
```

This method can be called with a statement such as

```
print_message("This is a secret message!");
```

3.7 Object Oriented Programming

By now you know quite a bit of Java programming. Now we begin to look at Java's most fundamental feature – **Classes** and **Objects**. Java is an **Object Oriented Programming (OOP) language**. In an OOP language, a program is collection of *objects* that interact with other objects to solve a problem. Each object is an instance of a *class.*

Imagine you are creating a database application for a bookstore. You will need to store data about all the books in the store. So, each book will become an object in your program. Further, each book will have certain characteristics or attributes such as its Title, Author, Publisher, and Price. You may also want to perform certain actions on a book such as display its details on the computer screen or find its price. In an OOP language, such as Java, an entity such as a book in the example is referred to as a **class**. A class is a physical or logical entity that has certain attributes. The title, author, publisher, genre and price are the **data members** of the class Book. Displaying book details and getting its price are **method members** of the class Book. Method members can get, set or update the class data members.

Class Book

Data Members:

Title

Author

Publisher

Genre

Price

Method Members:

display()

getPrice()

To describe one particular book in the library, you would create an **object** of the class book and fill in all its data members. For example one book in the library would be an object populated with the following data members:

Book: book1

Title: Game of Thrones

Author: George R Martin

Publisher: Harper Collins

Genre: Fiction

Price: 450

Another book would be another object populated with the following data members:

Book: book2

Title: Fundamentals of Database Systems

Author: Shamkant Navathe

Publisher: Pearson

Genre: Educational

Price: 400

You can think of a class as a template from which objects are created. All objects of the same class have the same type of data members.

Method members of a class are invoked on an object to perform the action associated with that method. For example, to display the details of book1 you would call the method member `display()` of the class with book1 as the invoking object. To display the details of book2 you would call the same method member `display()` but with the invoking object as book2.

3.8 Class Design

We are now ready to create a class in Java. A class in Java begins with the keyword `class` followed by the name of the class. The body of the class is enclosed within curly braces. The body contains the definitions of the data and method members of the class. The data members are defined by specifying their type. The method members of the class are defined just like the user defined methods we saw earlier. The method members have access to all the data members of the class and can use them in their body. The data members of a class are like **global** variables – they can be accessed by all the method members of the class. Figure 3.8(a) shows how to code the `class Book` in Java.

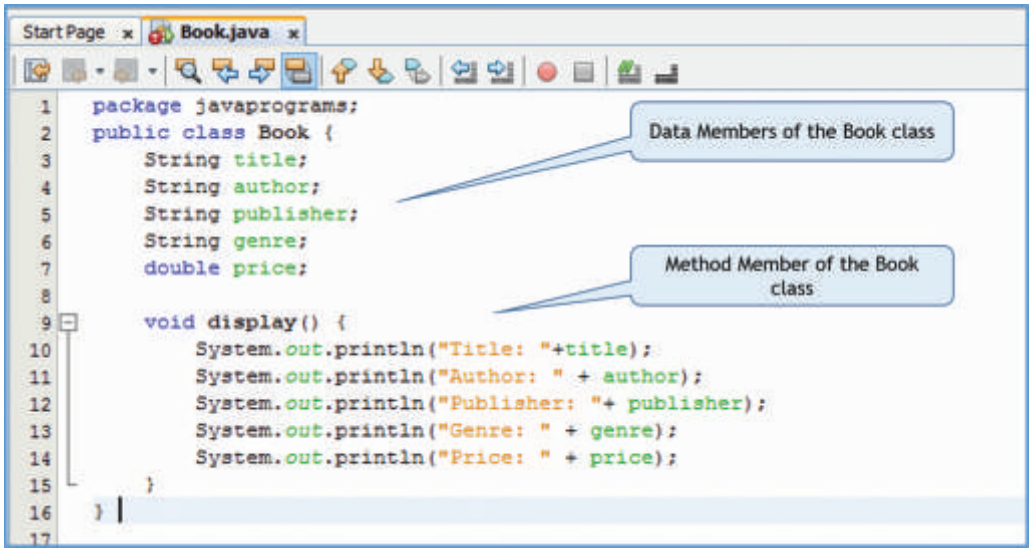


Figure 3.8(a): The Book Class

3.8.1 Constructors

A data member that is declared but not initialized before using, is assigned a default value by the compiler usually either zero or null. However, it is good programming practice to always initialize variables before using them.

A special method member called the **constructor** method is used to initialize the data members of the class (or any other initialization is to be done at time of object creation). The constructor has the same name as the class, has no return type, and may or may not have a parameter list. Whenever a new object of a class is created, the constructor of the class is invoked automatically. We do not call the constructor explicitly.

Let us define a constructor for the `Book class` that takes parameters corresponding to each data member and uses the parameters to initialize the data members Figure 3.8(b).

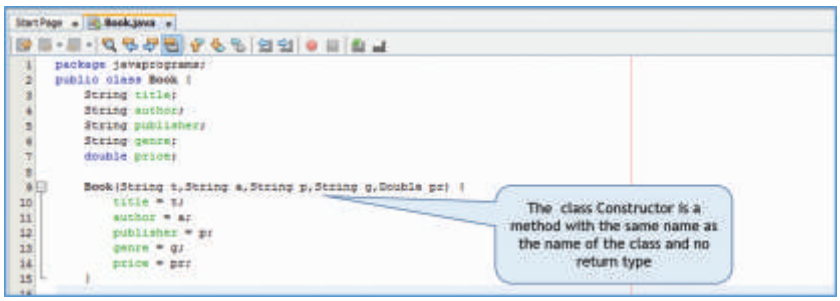


Figure 3.8(b): The Book Class Constructor

We can now use our constructor to create an object of the `Book class`.

```
Book book1 = new Book("Game of Thrones",
                      "George R Martin",
                      "Harper Collins",
                      "Fiction", 450.0);
```

This has the effect of creating a new object `book1` with its data members set as per the parameter list of the constructor.

Note that once we have defined a parameterized constructor, we will not be able to create a `Book object` without any parameters, that is, with a statement like:

```
Book book3 = new Book();
```

The compiler will complain that the methods actual and formal arguments differ in length. If you want to still be able to create a book object without specifying any parameters, you will have to also define a parameter-less constructor as below.

```
Book() {
   title = "";
   author = "";
   publisher = "";
   genre = "";
   price = 0;
}
```

The parameter-less constructor above initializes all the data members of the book with default values (null string in case of `String` parameters and 0 for the `double` parameter).

Once the class is defined, we can create objects of the class and access its members.To create an object of a class we use the keyword new. We can create objects of the `class Book` in a Java class file of our package. So first we create a new Java class file, called `BookTest.java`, in our package. Then in the main method of the `BookTest class`, we create a `Book object` Figure 3.8(c). The following statement creates an object named book1 of the `class Book`.

```
   public static void main(String[] args) {
   Book book1 = new Book();
}
```

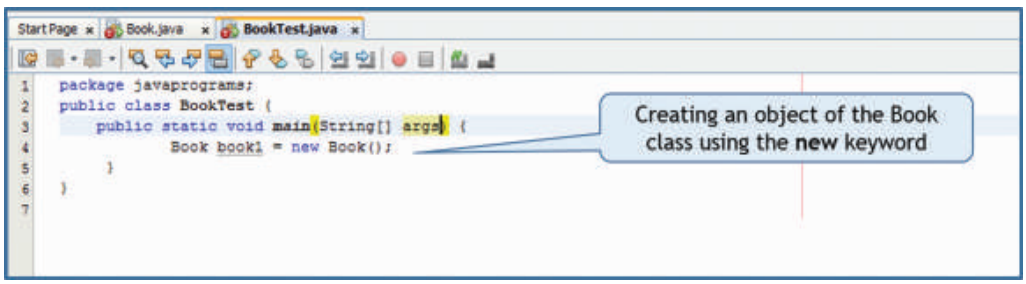


Figure 3.8(c): The BookTest Program

Data members and method members of an object are accessed using the dot (.)operator. As soon as you type the name of the object followed by a dot, `NetBeans` will prompt you with a list of available data and method members that you can access Figure 3.8(d).

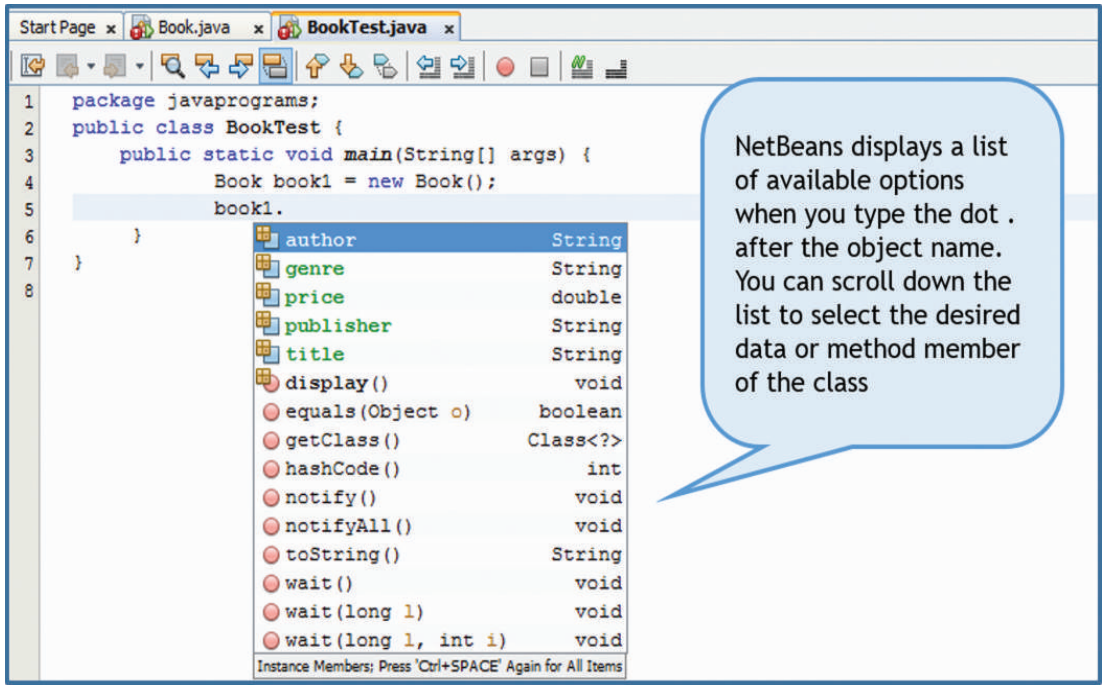


Figure 3.8(d): List of available members of the Book Class

Note that the data members and method member that you defined in the Book class are available in the list. Also note that a number of methods that you did *not* define in the Book class, such as, `hashCode()`, `wait()` and so on, are also available in the list. This is because all Java classes inherit from the base class **Object**. These extra methods are defined in the base class and are therefore available to all Java classes. Objects of `class Book` can invoke the method member defined in the class Figure 3.8(e). For example, in the following statement, the object book1 invokes the method member `display()`.

```
book1.display();
```

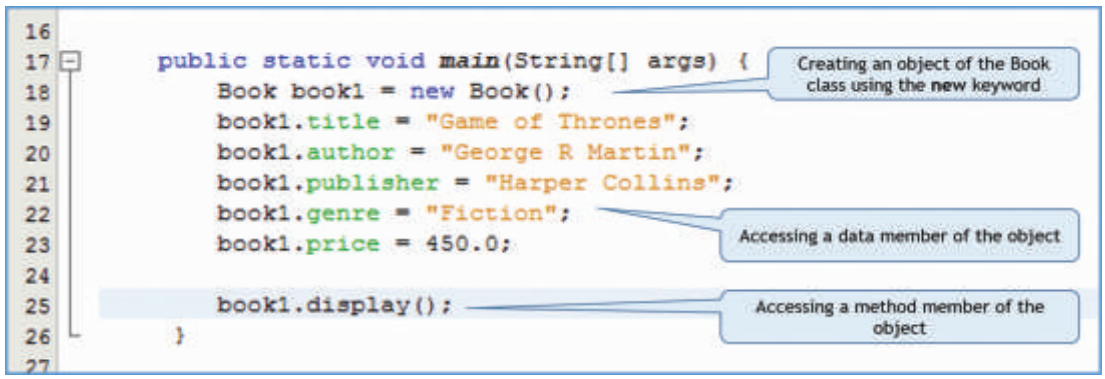


Figure 3.8(e): Accessing the Members of an Object

The `display()` method as per its method body prints the data members of its invoking object, in this case `book1`. Figure 3.8(f) displays the result of executing the `BookTest` program.

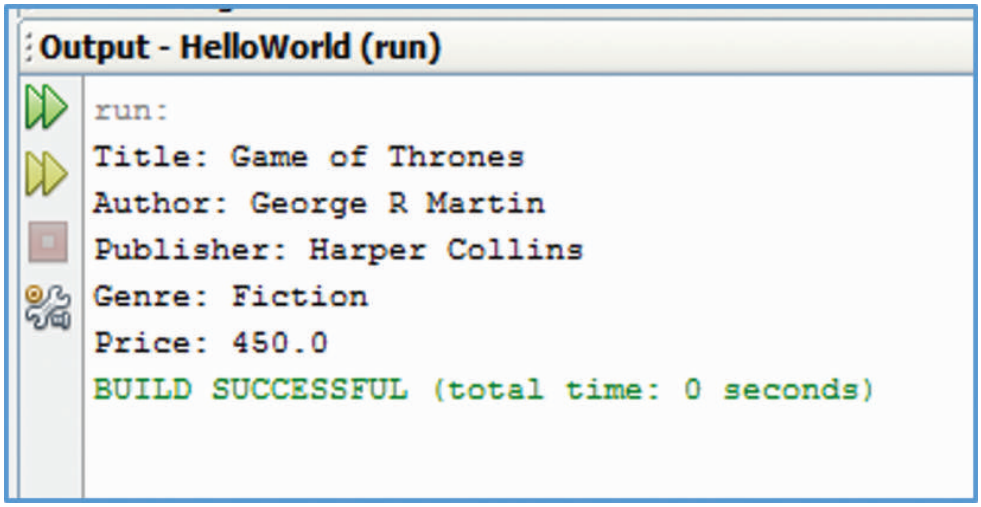


Figure 3.8(f): Output from the BookTest Program

In the `BookTest.java` program, try to create another object book2, initialize its data members, and then call `book2.display()`. Run the modified program to verify that now `book2`'s data members are displayed.

3.8.2 Access Modifiers

Data members of a class can be accessed from outside the class by default. However, it is generally not good programming practice to allow data members to be accessed outside the class. By allowing objects to change their data members arbitrarily, we lose control over the values being held in them. This makes debugging code harder and our code vulnerable to security issues. To make a data member or a method member of a class visible only within the class, we add the keyword `private` before its declaration Figure 3.8(g).

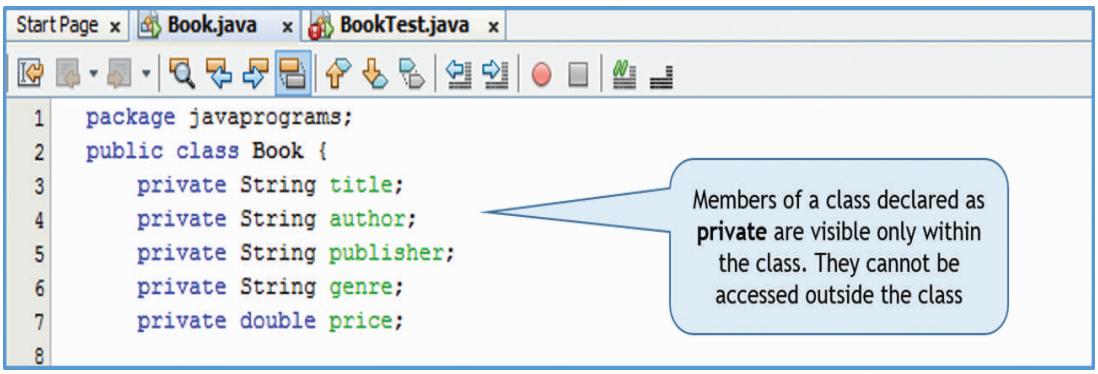


Figure 3.8(g): Private Access Modifier

Private members of a class cannot be accessed outside the class. Declaring the data members of the `Book class` private, we won't be allowed access to them in the `BookTest` class Figure 3.8(h).

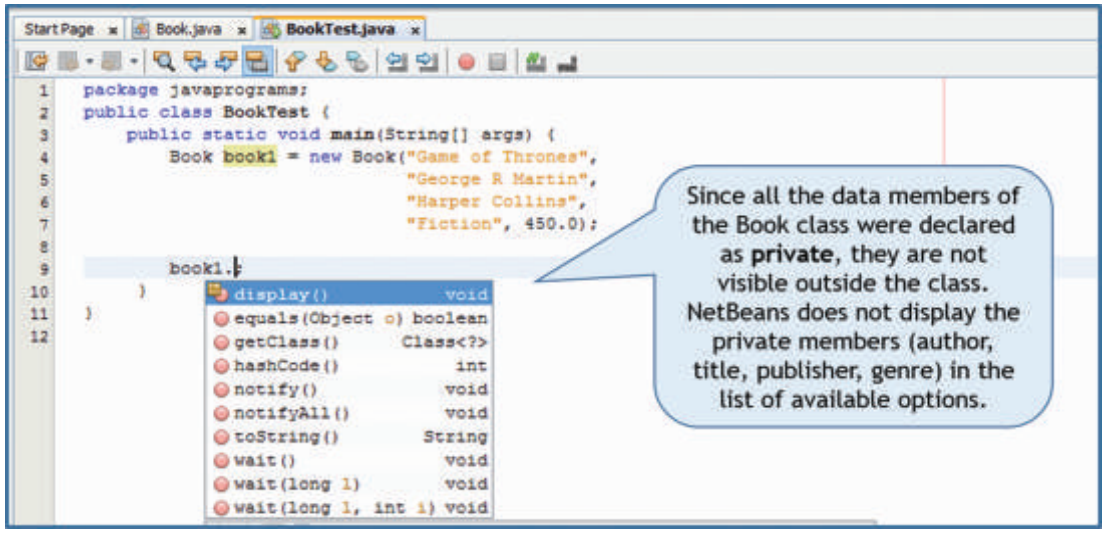


Figure 3.8(h): Private Data Members are not visible outside the class

As opposed to private, a public member of the class has visibility both within and outside the class. By default, all members of a class are public.

3.8.3 Getter and Setter Methods

Private data members of a class cannot be accessed outside the class however, you can give controlled access to data members outside the class through getter and setter methods. A getter method returns the value of a data member. For example we could define a getter method in the `Bookclass` for the `price` data member as given below:

```
double getPrice ( ) {
   return price;
}
```

Similarly, we define a setter method but control how the `price` is set. We do not allow a book `price` to become lower than 100.0.

```
void setPrice(double newprice) {
   if (newprice < 100)
      System.out.println("Price cannot be set lower than
      100!");
   else
      price = newprice;
}
```

In the `BookTest class`, the getter and setter methods can be used to retrieve and set a new price for a book (Figure 3.8(i)). Let's say the `BookTest` class wants to discount the `price` of the `book1` object by 20%, we first get the book's price using the `getPrice()` method, use the price to determine a discounted price for the book, and then use the **`setPrice()`** method to change the price of the book. The code for the same is as follows:

```
double bookprice = book1.getPrice();
bookprice = bookprice - 20*bookprice/100;
book1.setPrice(bookprice);
```

Now `book1's` data member `price` is set to 360.0 Figure 3.8(j). However if a discount caused a book object's price to fall below 100, the setter method would not change the `price`. Without a controlled setter function, the `BookTest` class could have changed the book's price arbitrarily.

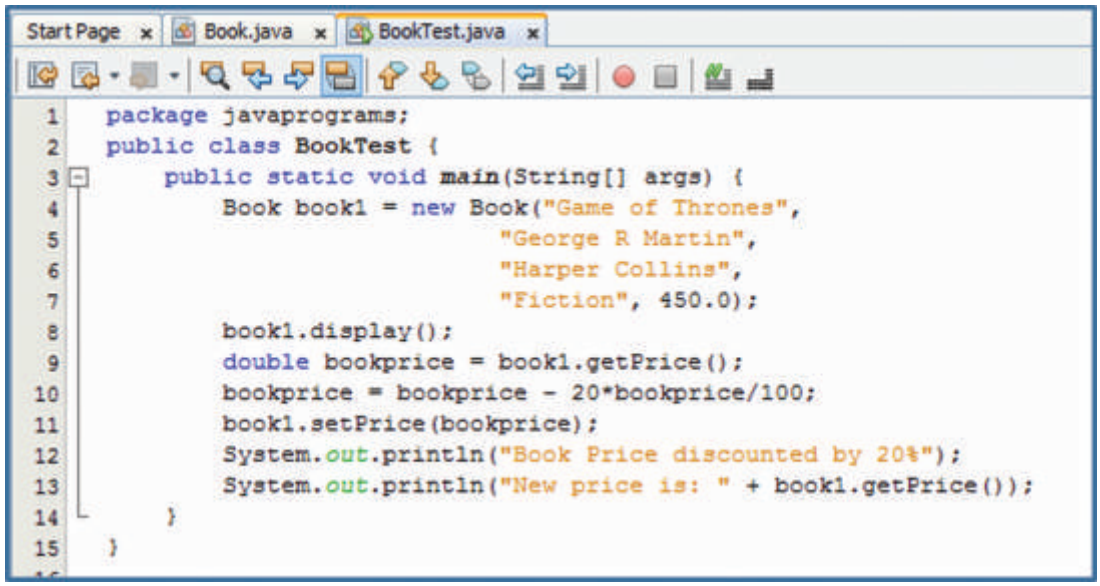


Figure 3.8(i): Use of Getter and Setter Methods

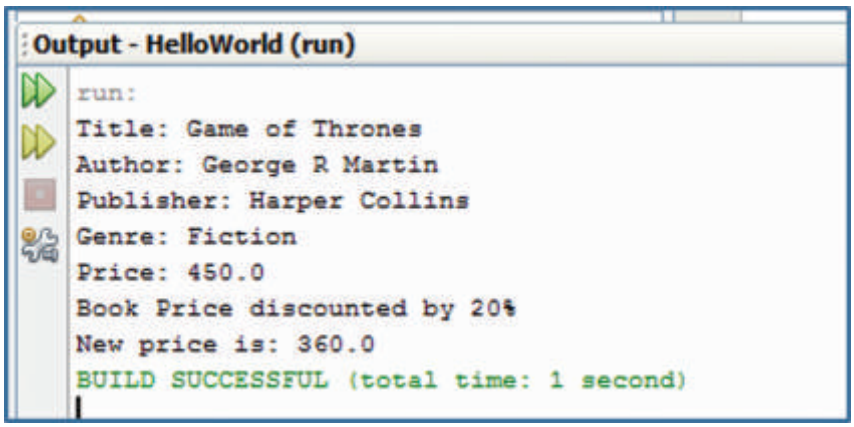


Figure 3.8(j): Output from use of Getter and Setter Methods

3.9 Java Libraries

The power of Java comes from the hundreds of Java classes that are already prebuilt and can be used in your programs. To use a prebuilt class and associated methods in those class, all you have to do is to use the keyword **import** to import the class from the package in which it is contained into your space. The `import` statements must appear before any class definitions in the file. In the following sections, we will look at some useful classes.

3.9.1 Data Input

The programs we have written up to now haven't been interactive. A program is interactive if it is able to take input from the user and respond accordingly. Let us write a program to take user input. To take user input we use the prebuilt Scanner class. This class is available in the `java.util` package. First we import this class,

```
import java.util.Scanner;
```

Now we create an object of the class `Scanner`. The constructor of this class requires the source from which input is to be taken. Since we will take input from the console, we use the `System.in` object.

```
Scanner user_input = new Scanner(System.in);
```

Then, we invoke the `next()` method of the Scanner class that returns the token read from the input stream as a `String` object.

```
String name = user_input.next();
System.out.println("Hello "+ name);
```

To read numeric data input, again a Java prebuilt class comes to our rescue. This time we use the `Integer class`. The class has a static method `parseInt()` that takes a `String` as parameter and returns the equivalent integer.

```
String age_string = user_input.next( );
int age = Integer.parseInt(age_string);
System.out.print("In 5 years you will be "+ (age +5));
System.out.println(" years old.");
```

Since the Integer class is in the `java.lang` package, which is already imported for all classes, we do not need to import the package. Similar to the Integer class, there is the **Float** class with a `parseFloat()` method, a `Double` class with a `parseDouble()` method, and other such classes that you can use to read a `String` and convert it to the required data type.

Start Page | DataInputDemo.java

```
1  package javaprograms;
2  import java.util.Scanner;
3  public class DataInputDemo {
4
5      public static void main(String[] args) {
6          Scanner user_input = new Scanner( System.in );
7          System.out.print("Enter your name: ");
8          String name = user_input.next( );
9          System.out.print("Enter your age: ");
10         String age_string = user_input.next( );
11         int age = Integer.parseInt(age_string);
12
13         System.out.println("Hello "+ name);
14         System.out.print("In 5 years you will be "+ (age +5));
15         System.out.println(" years old.");
16     }
17 }
```

Figure 3.9(a): Data Input Demo Program

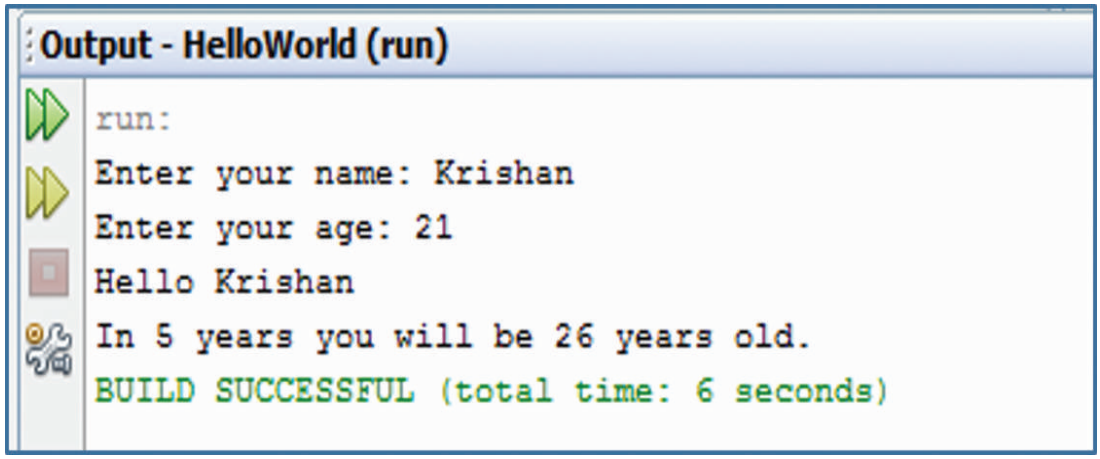


Figure 3.9(b): Output from the Data Input Demo Program

Figures 3.9(a) and 3.9(b) show the Data Input Demo program and its output respectively.

3.9.2 Array Manipulation

In the previous section you learned about some useful prebuilt classes for data input. In this section, we will explore the prebuilt Arrays class from the `java.util` package for array handling. The `Arrays class` has a number of useful methods. Let us start by using the `sort()` method to sort an array of integers in ascending order.

First we **import** `java.util.Arrays` class. Then in the `main()` method, we invoke the `Arrays.sort()` method on the array we need to sort.

```
double[] marks = {103, 144, 256.5,346, 387.5};
Arrays.sort(marks);
```

The marks array after sorting becomes = {103.0, 144.0, 256.5, 346.0, 387.5}. Sorting makes it easier for us to find the lowest and highest marks obtained by a student. To print the lowest marks, we can now write

```
System.out.println(marks[0]);
```

To print the highest marks,we can write

```
System.out.println(marks[marks.length-1]);
```

The same method can be used to sort an array of Strings in alphabetic order.

```
String[] names =
{"Sarthk","Saumya","Mayank","Mudit","Shiva","Anju",
"Savita"};
Arrays.sort(names);
```

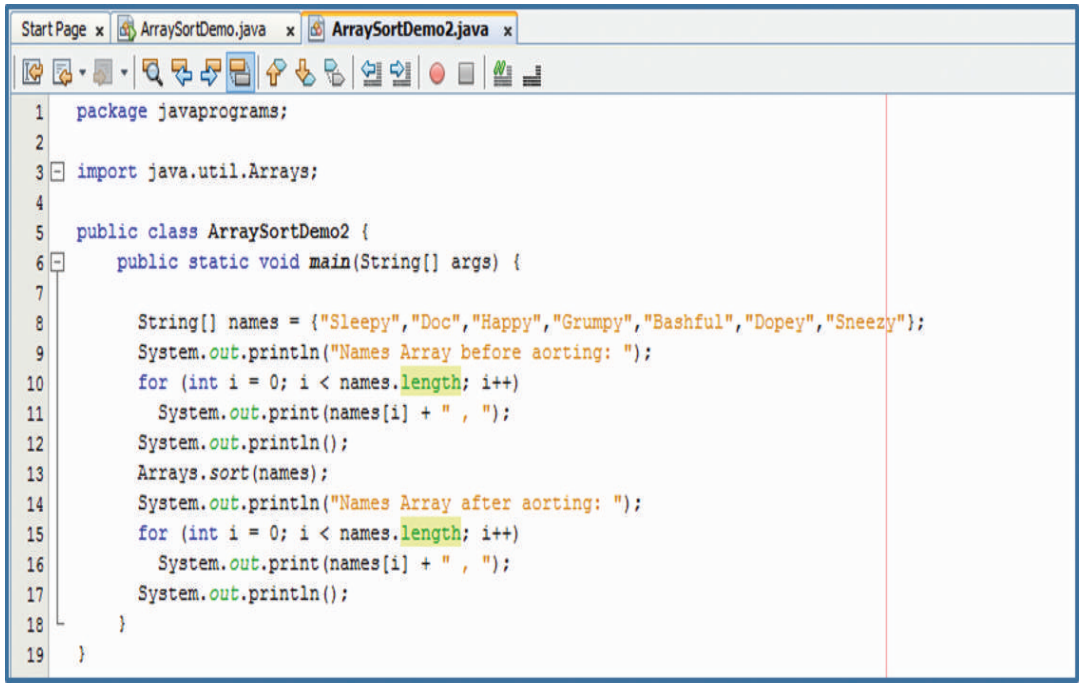


Figure 3.9(c): The ArraySortDemo2 Program

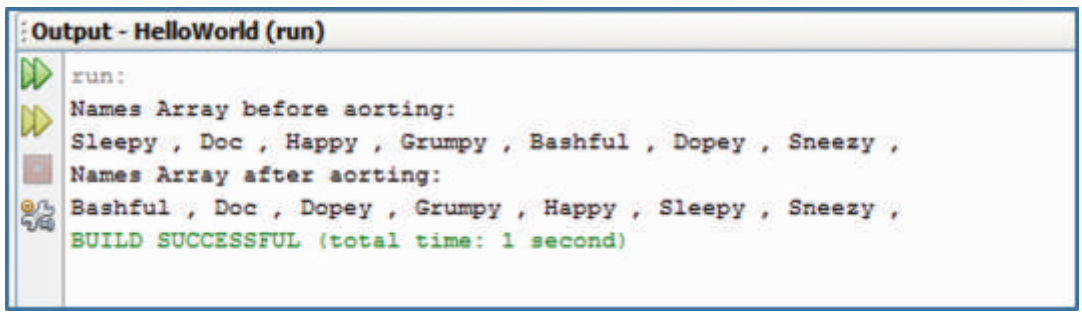


Figure 3.9(d): Output from the ArraySortDemo2 Program

Figures 3.9(c) and 3.9(d) show the `ArraySortDemo2` program and its output respectively.

The `binarySearch()` method of the `Arrays` class helps us search for a specific element in the array. The parameters it needs are the array to be searched and the key element to be searched. The method returns the index of the array where the key is present. If the key is not found in the array, the `binarySearch` method returns -1.

```
double[] marks = {103, 144, 256.5, 346, 387.5};
int key = 346;
int index = Arrays.binarySearch(marks,key);
```

Note that the array must be sorted before invoking `binarySearch()`. If it is not sorted, the results are undefined

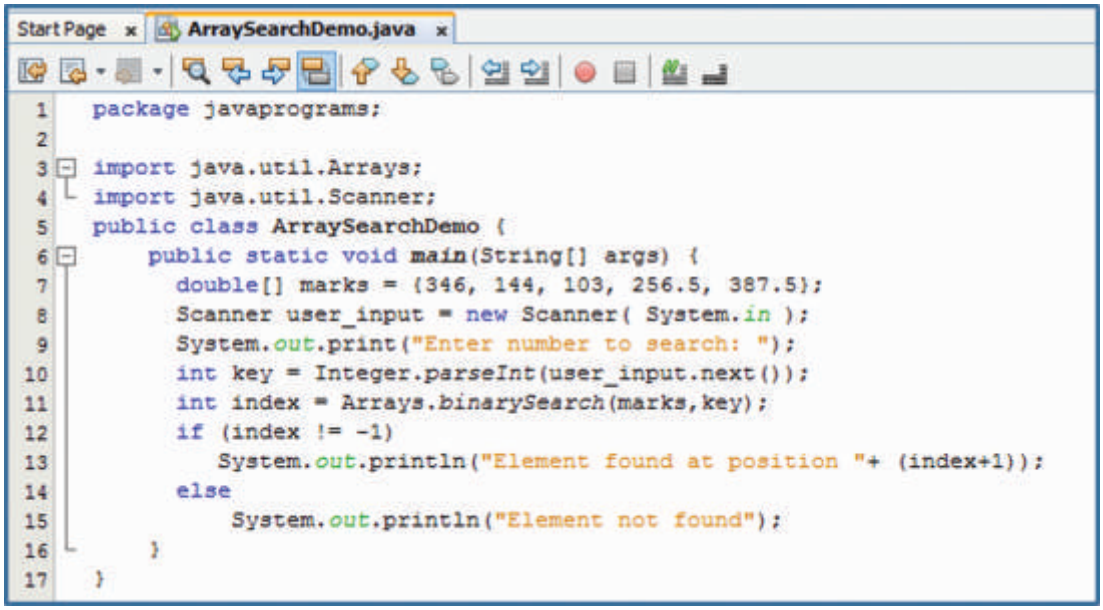


Figure 3.9(e): The ArraySearch Demo Program

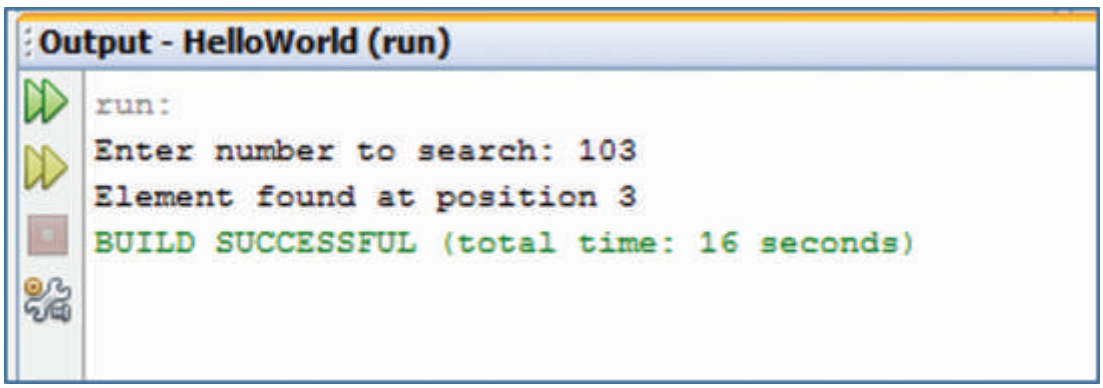


Figure 3.9(f): Output from the Array Search Demo Program

Figures 3.9(e) and 3.9(f) show the `ArraySearchDemo` program and its output respectively.

You can visit **http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html** to explore all the methods available for the Array class.

3.9.3 String Manipulation

In the previous section, we learned to manipulate arrays using Java prebuilt classes. In this section we learn to manipulate Strings using the String class present in the java.lang package.

The first method we will use from the String class is the `toUpperCase()` method. This method converts a string to all uppercase letters.

```
String myString = "Hello World";
System.out.println("UpperCase: " + myString.toUpperCase());
```

The output of the code given above will be:

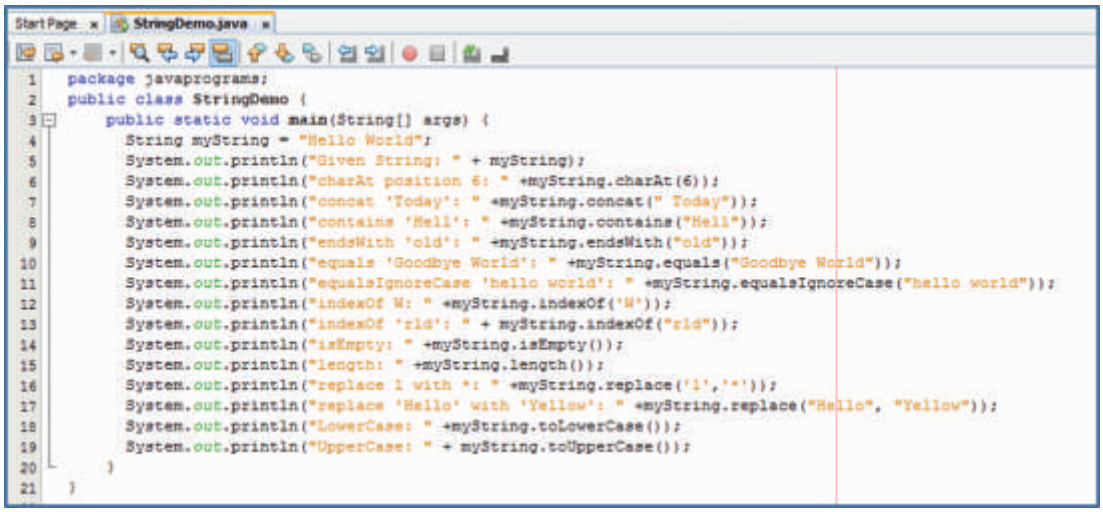
```
HELLO WORLD
```

The following table shows the commonly used `String` class methods and the result of applying them on the string "`Hello World`".

Method	Description	Application myString = "Hello World"	Output
`char charAt (int index)`	Return the character at the given index	`myString. charAt(6)`	`W`
`String concat (String str)`	Concatenate the specified string at the end of this string	`myString.concat ("Today")`	`Hello World Today`
`boolean contains (String s)`	Returns true if this string contains the specified substring	`myString.contains ("Hell")`	`True`
`boolean endsWith (String suffix)`	Test whether this string end with the given suffix	`myString.endsWith ("old")`	`False`
`boolean equals (Object anObject)`	Compare this string with specified object	`myString.equals ("Goodbye World")`	`False`
`boolean equalsIgnoreCase (String another)`	Compare this string with specified string ignoring case	`myString. equalsIgnoreCase ("hello world")`	`True`
`int indexOf (int c)`	Return the index of the first occurrence of given character	`myString. indexof ('W')`	`6`
`int indexOf (String str)`	Return the index of the first occurrence of given substring	`myString. indexof ("rld")`	`8`
`boolean isEmpty()`	Returns true if the length of this string is 0	`myString. isEmpty()`	`False`
`int length()`	Returns the length of the string	`myString. length ()`	`11`
`String replace(char oldChar, char newChar)`	Returns a new string after replacing all occurrences of oldChar in this string with newChar	`myString.replace ('l','*')`	`He**o Wor*d`
`String replace(String oldStr, String newStr)`	Returns a new string after replacing all occurrences of oldStr in this string with newStr	`myString.replace ("Hello", "Yellow")`	`Yellow World`

String toLowerCase()	Converts all of the characters in this String to lower case	myString.toLowerCase ( )	hello world
String toUpperCase()	Converts all of the characters in this String to upper case	myString.toLowerCase ( )	HELLO WORLD

Figures 3.9(g) and 3.9(h) show the String Demo program and its output respectively.



```
package javaprograms;
public class StringDemo {
    public static void main(String[] args) {
      String myString = "Hello World";
      System.out.println("Given String: " + myString);
      System.out.println("charAt position 6: " +myString.charAt(6));
      System.out.println("concat 'Today': " +myString.concat(" Today"));
      System.out.println("contains 'Hell': " +myString.contains("Hell"));
      System.out.println("endsWith 'old': " +myString.endsWith("old"));
      System.out.println("equals 'Goodbye World': " +myString.equals("Goodbye World"));
      System.out.println("equalsIgnoreCase 'hello world': " +myString.equalsIgnoreCase("hello world"));
      System.out.println("indexOf W: " +myString.indexOf('W'));
      System.out.println("indexOf 'rld': " + myString.indexOf("rld"));
      System.out.println("isEmpty: " +myString.isEmpty());
      System.out.println("length: " +myString.length());
      System.out.println("replace l with *: " +myString.replace('l','*'));
      System.out.println("replace 'Hello' with 'Yellow': " +myString.replace("Hello", "Yellow"));
      System.out.println("LowerCase: " +myString.toLowerCase());
      System.out.println("UpperCase: " + myString.toUpperCase());
    }
}
```

Figure 3.9(g): The String Demo Program

```
Output - HelloWorld (run)
run:
Given String: Hello World
charAt position 6: W
concat 'Today': Hello World Today
contains 'Hell': true
endsWith 'old': false
equals 'Goodbye World': false
equalsIgnoreCase 'hello world': true
indexOf W: 6
indexOf 'rld': 8
isEmpty: false
length: 11
replace l with *: He**o Wor*d
replace 'Hello' with 'Yellow': Yellow World
LowerCase: hello world
UpperCase: HELLO WORLD
BUILD SUCCESSFUL (total time: 2 seconds)
```

Figure 3.9(h): Output from the String Demo Program

You can visit **http://docs.oracle.com/javase/6/docs/api/java/lang/String.html** to explore all the methods available for the String class.

3.10 Exception Handling

By now you may have written quite a few programs. Some of your programs when executed may have terminated unexpectedly with runtime errors. The errors could have occurred because an array index reference was out of range, or an attempt was made to divide an integer by zero, or there was insufficient computer memory and so on. Such an error situation that is unexpected in the program execution and causes it to terminate unexpectedly is called an `exception`. As a programmer, you should anticipate exceptions in your program that could lead to unpredictable results and handle the exceptions. Consider the program in Figure 3.10(a) that has a method called `divide()`. The method takes two numbers as parameters, divides them and returns the quotient:

```
int divide(int dividend, int divisor) {
   return dividend/divisor;
}
```

Since division by 0 in integer arithmetic causes a program to terminate prematurely, a call to the function divide such as `divide(10,0)`; will cause the program to fail Figure 3.10(b).

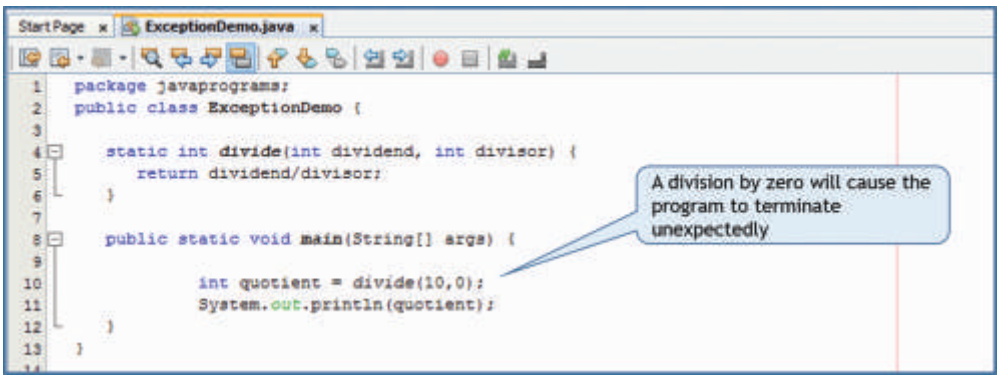


Figure 3.10(a): The Exception Demo Program

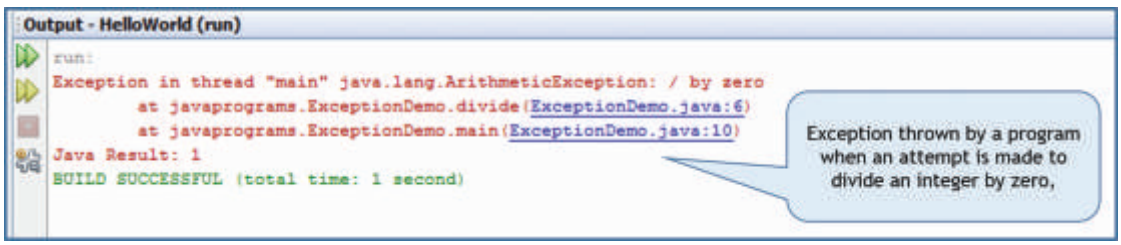


Figure 3.10(b): Output from the ExceptionDemo Program

Note: If the method `divide()` had used floating point numbers instead of integers, there would not have been an error, since in floating-point arithmetic, division by zero is allowed — it results in infinity, which would be displayed as `Infinity`.

Java provides an exception handling mechanism so that a program is able to deal with exceptions, and continue executing or terminate gracefully. The basic idea in exception

handling is to

- Denote an exception block - Identify areas in the code where errors can occur
- Catch the exception - Receive the error information
- Handle the exception - Take corrective action to recover from the error

Java provides the following keywords to handle an exception:

1. **try** – A `try` block surrounds the part of the code that can generate exception(s).
2. **catch** – The catch blocks follow a `try` block. A `catch` block contains the exception handler - specific code that is executed when the exception occurs. Multiple `catch` blocks following a `try` block can handle different types of exceptions.

The structure of a `try-catch` statement block for exception handling is as below:

```
try {
   // Part of the program where an exception might occur
}
catch (exceptiontype1 argument1) {
   // Handle exception of the exceptiontype1
}
catch (exceptiontype2 argument2) {
   // Handle exception of the exceptiontype2
}
finally {
   //Code to be executed when the try block exits
}
```

The `try` block is examined during execution to detect any exceptions that may be thrown by any statements or any calls to methods within the block. If an exception is thrown, an exception object is created and thrown. The program execution stops at that point and control enters the `catch` block whose argument matches the type of the exception object thrown. If a match is found the statements in that `catch` block are executed for handling the exception.

If no exception is thrown during execution of the statements in the `try` block, the catch clauses that follow the try block are not executed. Execution continues at the statement after the last catch clause.

The optional `finally` block is always executed when the try block exits. This means the `finally` block is executed whether an exception occurs or not. Programmers use this block to put in *clean up* code, for example, freeing up resources allocated in the `try` block.

The following code fragment handles a division by zero exception:

```
try {
      int quotient = divide(10,0);
      System.out.println(quotient);
    } catch (Exception e) {
```

```
System.out.println(e.getMessage());
}
```

We catch an exception of the type `Exception`. An object of the class `Exception` is a "catch all" exception that returns a general error message encountered during program execution. The exception handler in the `catch` block uses the `getMessage()` method of the `Exception` class to print the error that caused the exception.

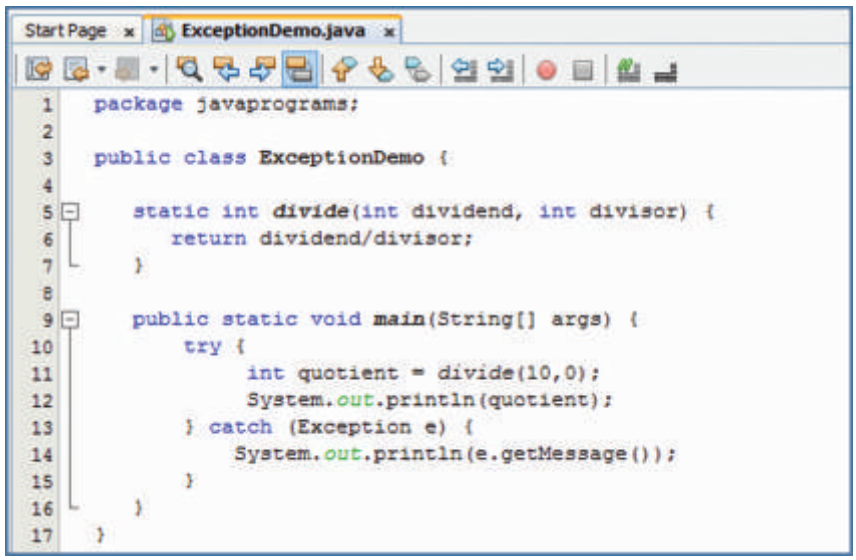


Figure 3.10(c): Exception Handling Demo Program

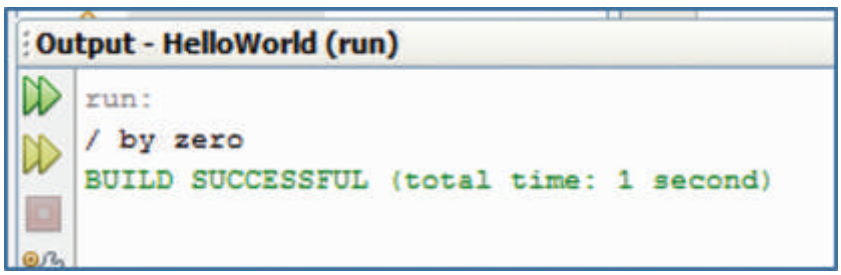


Figure 3.10(d): Output from the Exception Handling Demo Program

Figures 3.10(c) and 3.10(d) show the Exception Handling Demo program and its output respectively.

3.11 Database Connectivity

In this section, we will learn to connect a MySQL database to a Java program and get back the results from executing an SQL query on the database. Connecting a database to a Java program is easy with NetBeans since it allows us to connect directly to a MySQL server.

3.11.1 Connecting to the MySQL Server in NetBeans

We first need to configure NetBeans to Register and connect a MySQL Server. Follow the steps below to do just that.

Step 1: Click on the Services tab located on the left side of the NetBeans IDE. Right

click the **Databases** node and select **Register MySQl Server** (Figure 3.11(a)).

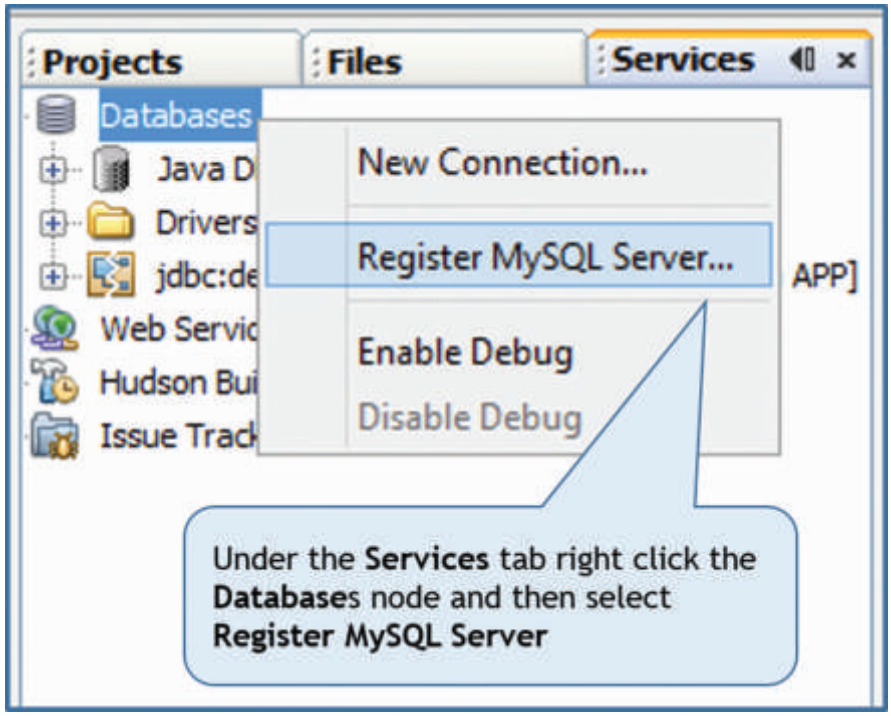


Figure 3.11(a): Register MySQL Server

Step 2: In the **MySQL Server Properties** Dialog Box that opens up, type in the **Administrator User Name** (if not displayed). Also type in the **Administrator Password** for your MySQl Server. Check the Remember Password checkbox and click on **OK** Figure 3.11(b).

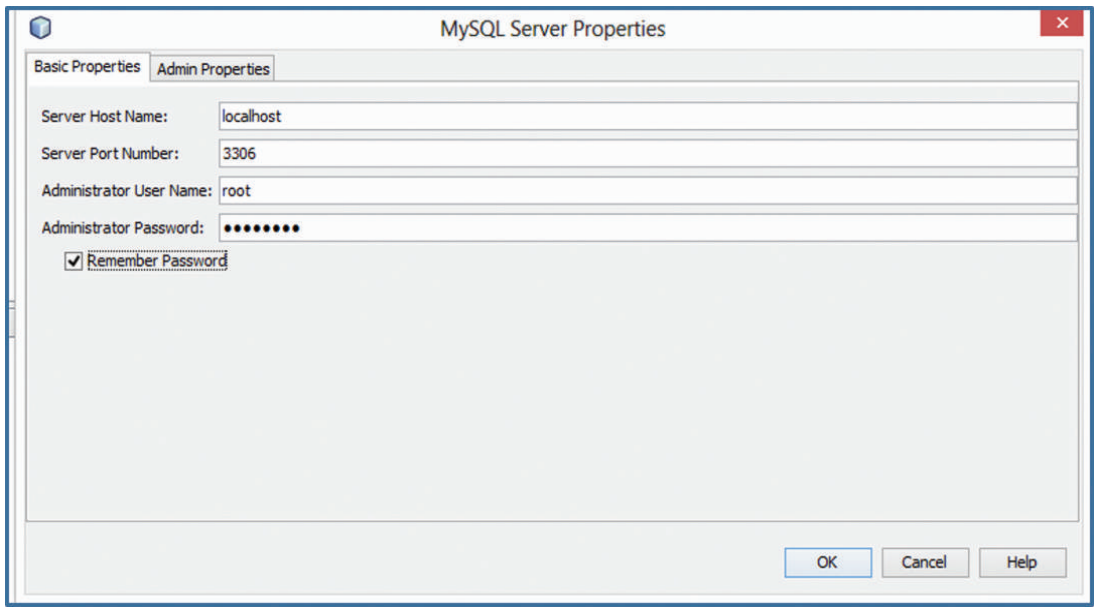


Figure 3.11(b): MySQL Server Properties Window

Step 3: In the same MySQL Server Properties Dialog Box, click on the **Admin**

Properties tab.

In the **Path/URL to admin tool field**, type or browse to the location of your MySQL Administration application **mysqladmin** (you will find it in the bin folder of your MySQL installation directory).

In the **Path to start command**, type or browse to the location of the MySQL start command **mysqld** (you will find it in the bin folder of your MySQL installation directory).

In the **Path to stop command** field, type or browse to the location of the MySQL stop command **mysqladmin** (you will find it in the bin folder of your MySQL installation directory). In the **Arguments** field, type **-u root stop** to grant root permissions for stopping the server.

When finished, the Admin Properties tab should appear similar to Figure 3.11(c). Click on OK.

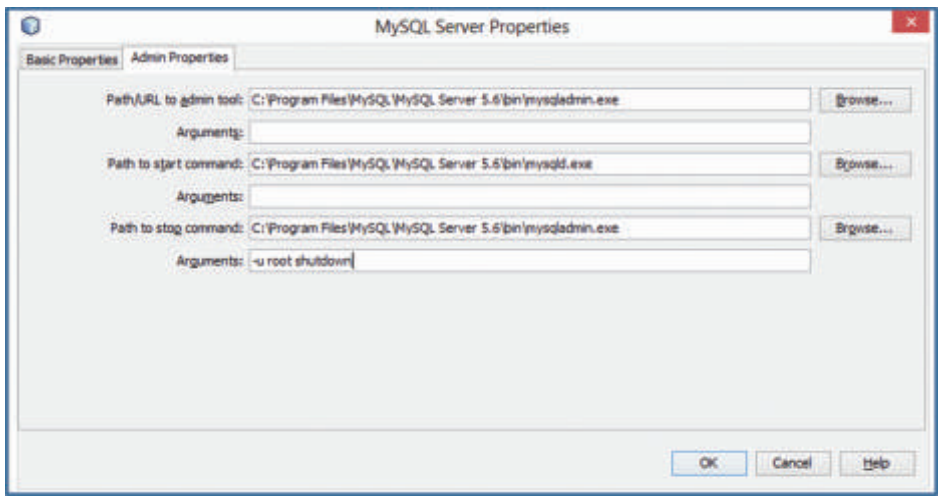


Figure 3.11(c): MySQL Server Properties Window (Admin Properties Tab)

The MySQL Server should now appear under the Database node in the Services tab in the NetBeans IDE Figure 3.11(d). However, it is shown disconnected.

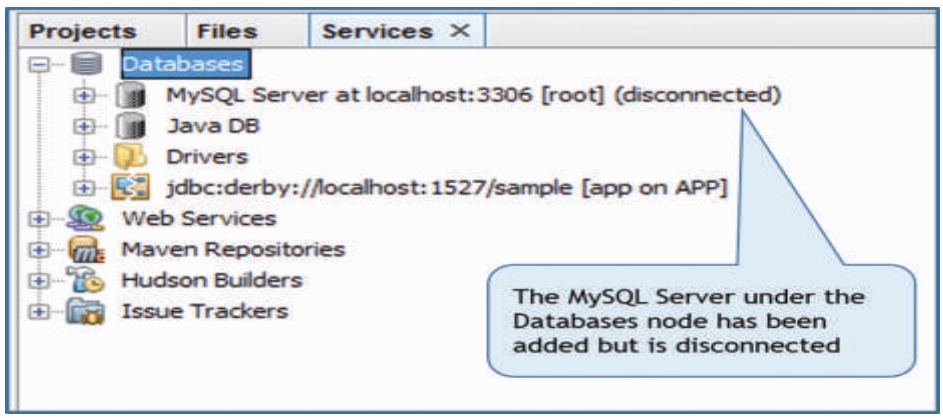


Figure 3.11(d): MySQL Server added to the Databases

Step 4: To connect the MySQL Server to NetBeans, under the Databases node, right click the **MySQL Server at localhost:3306 [root] (disconnected)** and select **Connect** Figure 3.11(e).

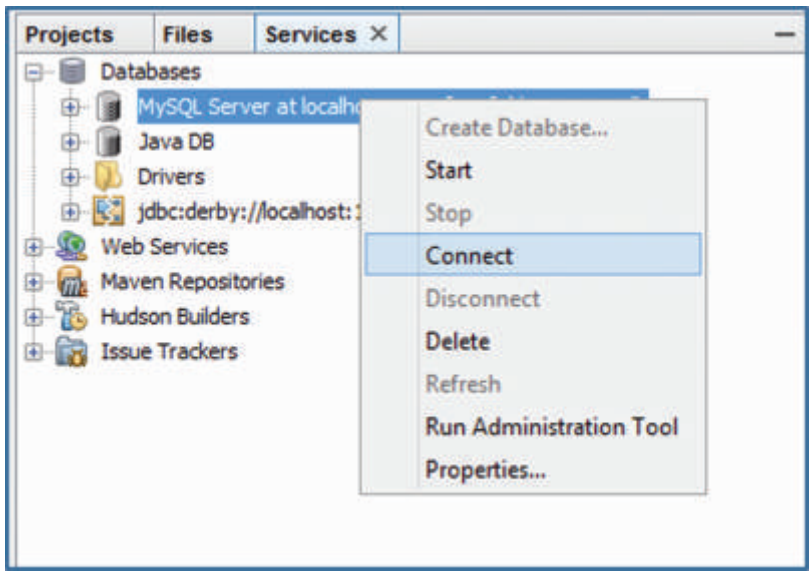


Figure 3.11(e): Connect to the MySQL Server

Note: The MySQL Server should be up and running before you connect to it through the NetBeans IDE. If it is not, start the MySQL Server by running the **mysqld** command from the bin folder of the MySQL installation directory. Then, attempt Step 4.

Step5: When the server is connected you should see the [disconnected] removed from the **MySQL Server at localhost:3306 [root]** database. You should also be able to expand the MySQL Server node by Clicking on the + sign to view all the available MySQL databases Figure 3.11(f).

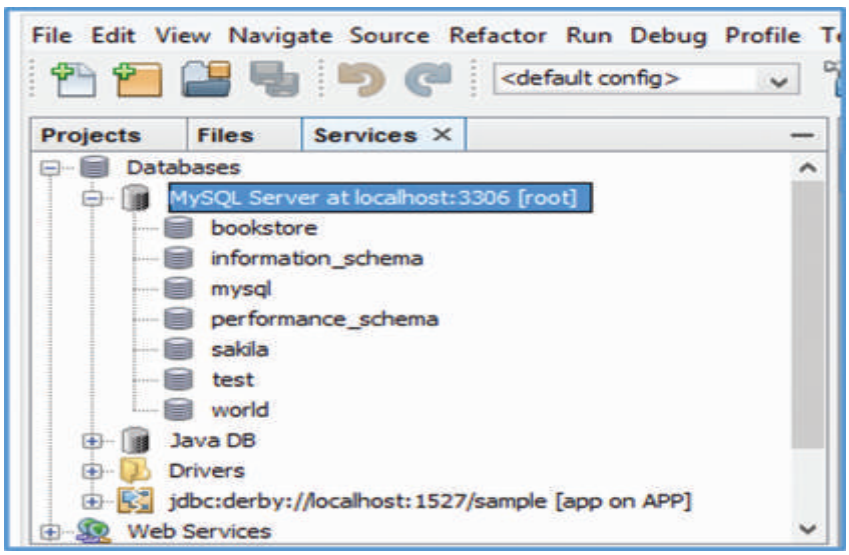


Figure 3.11(f): MySQL databases

That completes connecting the MySQL Server to NetBeans.

3.11.2 Adding the MySQL Connector JAR to the NetBeans Libraries

Before writing the Java program to connect to the database, we also need toadd the mysql connector JAR file to the Libraries in our project. The following steps show you how to do just that:

Step1: Under the **Projects** tab, right click on the **Libraries** node and select **ADD JAR/ Folder…** Figure 3.11(g).

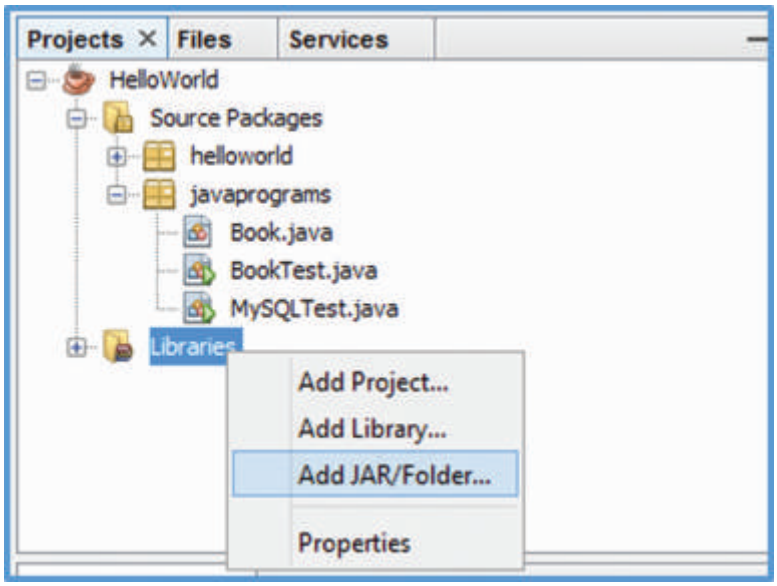


Figure 3.11(g): Add JAR/Folder to Libraries

Step 7: In the **Add JAR/Folder** dialog box that appears, navigate to the your **NetBeans Installation Folder**. Then navigate to the **/ide/modules/ext** folder and select the **mysql-connector-java-5.1.23-bin.jar** file. Click on **Open** Figure 3.11(h).

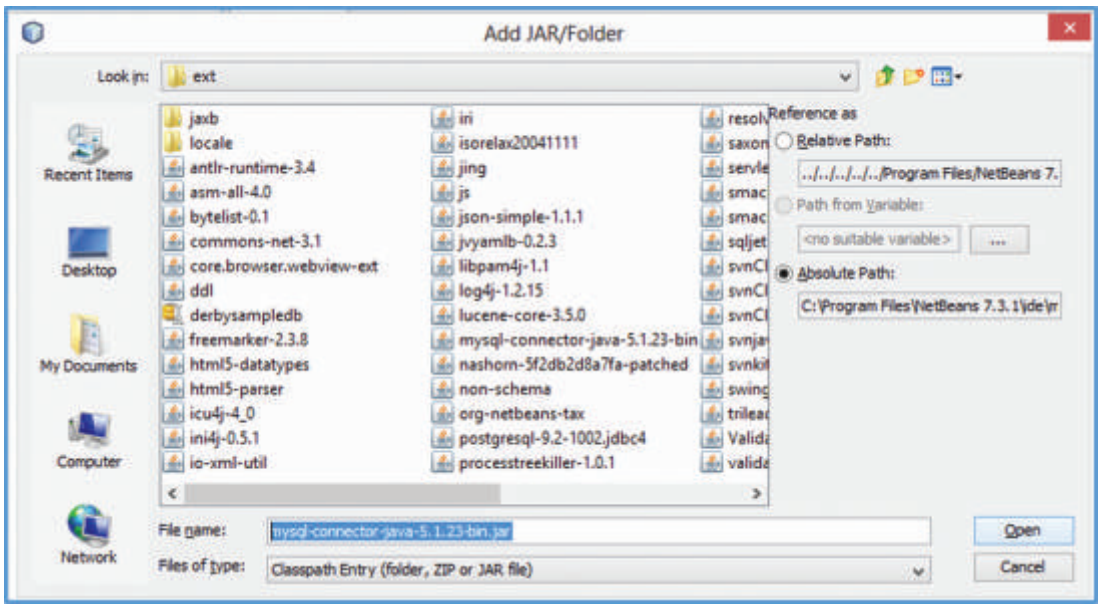


Figure 3.11(h): Add the mysql-connector-java-5.1.23-bin.jar

Now, expand the Libraries node (click on the + sign on its left), the mysql connector jar should have been added to the NetBeans Libraries Figure 3.11(i).

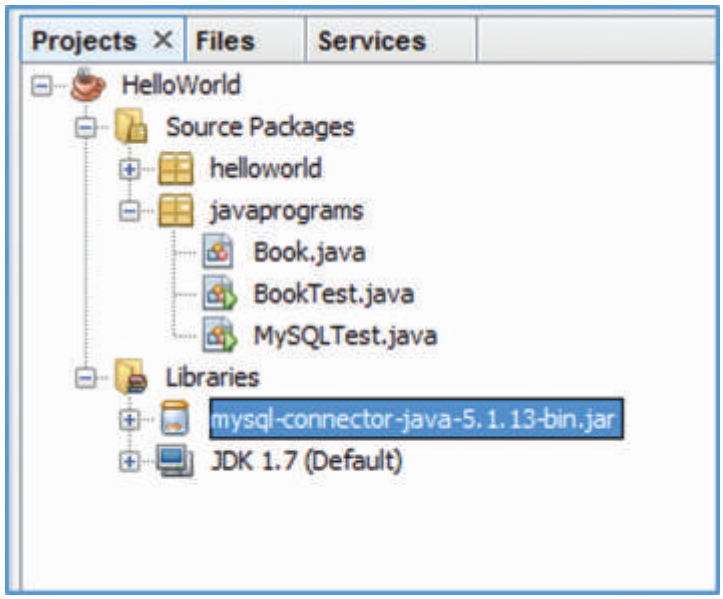


Figure 3.11(i): Libraries

3.11.3 Database Connection from Java

We are now ready to write a Java program to connect to a database and execute a SQL query on the database. In our MySQL database, we have already created a database called bookstore and a table called book within it. The columns of this table are (title, author, publisher, genre, and price). We have also inserted 5 rows into the table Figure 3.11(j).

```
mysql> select * from book;
+---------------------------------+-----------------+-----------------+-------------+-------+
| title                           | author          | publisher       | genre       | price |
+---------------------------------+-----------------+-----------------+-------------+-------+
| Effective Java                  | Joshua Bloch    | Pearson         | Educational |   500 |
| Fundamentals of Database Systems | Shamkant Navathe | Pearson         | Educational |   400 |
| Game of Thrones                 | George R Martin | Harper Collins  | Fiction     |   450 |
| Hold My Hand                    | Durjoy Dutta    | Penguin         | Fiction     |   150 |
| Programming with Java           | E Balagurusamy  | Tata McGraw-Hill | Educational |   220 |
+---------------------------------+-----------------+-----------------+-------------+-------+
5 rows in set (0.00 sec)
```

Figure 3.11(j): Table book from the Bookstore Database

We will now learn how to retrieve data from the MySQL table book in the bookstore database from a Java program.

All the classes that we need are in the java.sql package. So we import the required classes as in Figure 3.11(k).

First, to establish a database connection to the MySQL Server, we invoke the `getconnection()` method of the `DriverManager` class. This method needs three parameters –URL of the database, username, password to connect to the database.

Each database driver has a different syntax for the URL. The MySQl URL has a hostname, the port, and the database name. In our program we construct a String with `hostname` as *`localhost`*, port number as *3306*, and the database name as *bookstore*.

```
String dbURL = "jdbc:mysql://localhost:3306/bookstore";
```

We also assign the username and `password`, this has to be the same username and password that is used for starting the MySQL Server.

```
String username ="root";
String password = "password";
```

Next, we invoke the `getconnection()` method using the URL, `username`, and password:

```
Connection dbCon =DriverManager.getConnection(dbURL,
username, password);
```

Next, we use the Connection object returned by the `getconnection()` method and invoke the `createStatement()` method. This method returns a `Statement` object for sending SQL statements to the database.

```
Statement stmt = dbCon.createStatement();
```

Next, we invoke the `executeQuery()` method of the `Statement` object to execute an SQL query. This method returns a single `ResultSet` object. The `ResultSet` is a table of data returned by a specific SQL statement.

```
String query ="select * from book";
ResultSet rs = stmt.executeQuery(query);
```

Finally, we use the `next()` method of the `ResultSet` object to iterate through all the rows of data returned by the query. When there are no more rows left, the `next()` method will return false. Since we know there are 5 columns in the book table, we use a for loop and the `getString()` method of the `ResultSet` object to print all the five columns.

```
while(rs.next()){
   for (int i = 1; i <=5; i++) {
      System.out.print(rs.getString(i));
      System.out.print("|");
   }
   System.out.println();
}
```

All the statements described above have to be put in a try catch block to catch Exceptions (of the `Exception` type `SQLException`) that can occur while connecting or fetching data from the database.

Figures 3.11(k) and 3.11(l) show the complete listing of the MySQL Demo program and its output respectively.

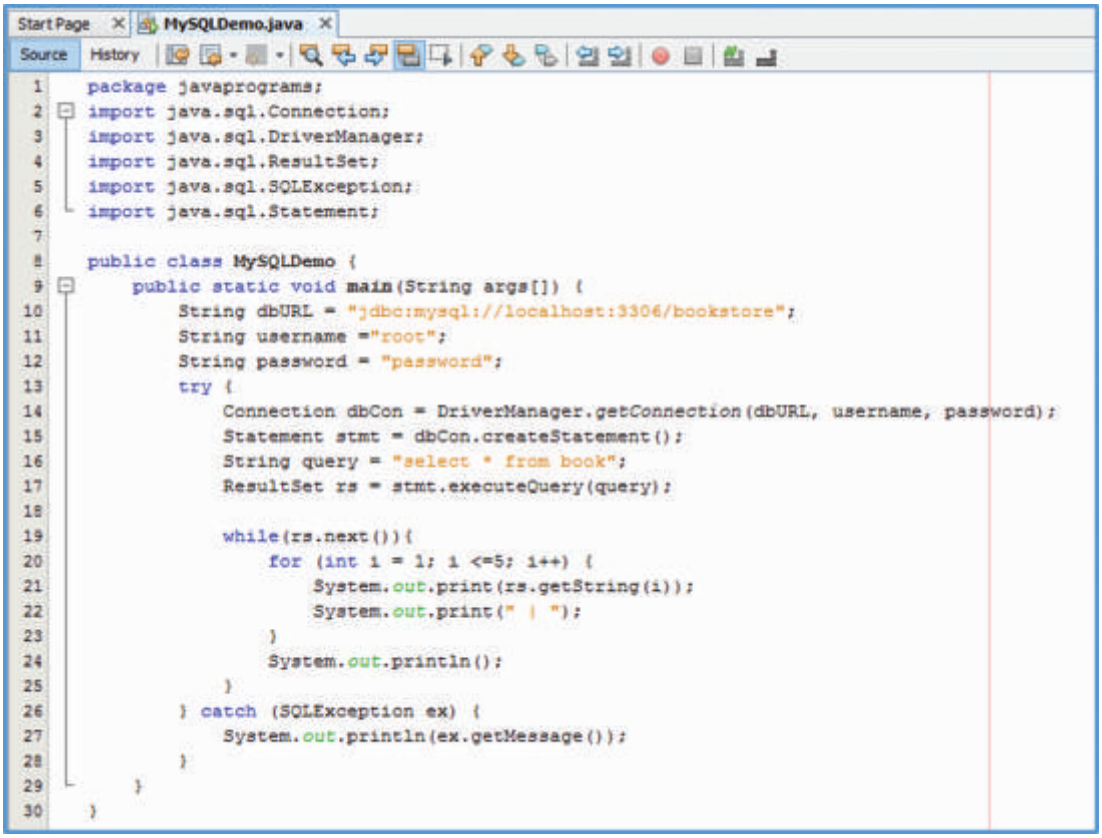


Figure 3.11(k): The MySQL Demo Program

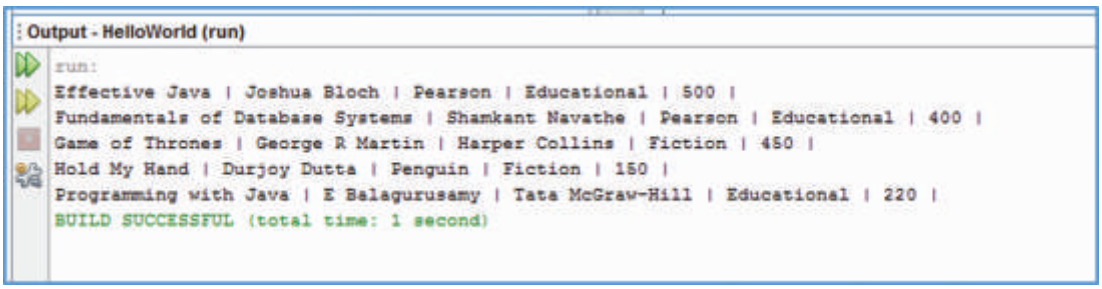


Figure 3.11(l): Output from the MySQL Demo Program

Once you complete running your database program, you can disconnect the MySQL Server from NetBeans. Under the Databases node, right click the **MySQL Server at localhost:3306 [root]** and select **Disconnect.**

3.12 Assertions, Threads, and Wrapper Classes

In this section, we will look at some advanced features of Java.

3.12.1 Assertions

An assertion is a useful mechanism for effectively identifying/detecting and correcting

logical errors in a program. When developing your Java programs, it is good programming practice to use assert statements to debug your code. An assert statement states a condition that should be true at a particular point during the execution of the program.

There are two ways to write an assertion

assert `expression;`

assert `expression1 : expression2`

The first statement evaluates `expression` and throws an AssertionError if `expression` is false. The second statement evaluates `expression1` and throws an AssertionError with `expression2` as the error message if `expression1` is false.

The program fragment below demonstrates usage of the `assert` statement.

```
assert age >= 18:"Age not Valid";
```

When this statement is executed, we assert that the value of the variable age should be >= 18. If it is not, an AssertionError is thrown and the error message "`Age not Valid`" is returned.

Note that, since assertions reduce runtime performance, they are disabled by default. To enable assertions at runtime, you can enable them from the command line by using the –ea option.

```
java –eaAssertionDemo
```

Alternatively, to enable assertions in NetBeans, Right click on your project>Properties> Run>VMOptions. Type–ea in the text box next to VM Options and click OK.

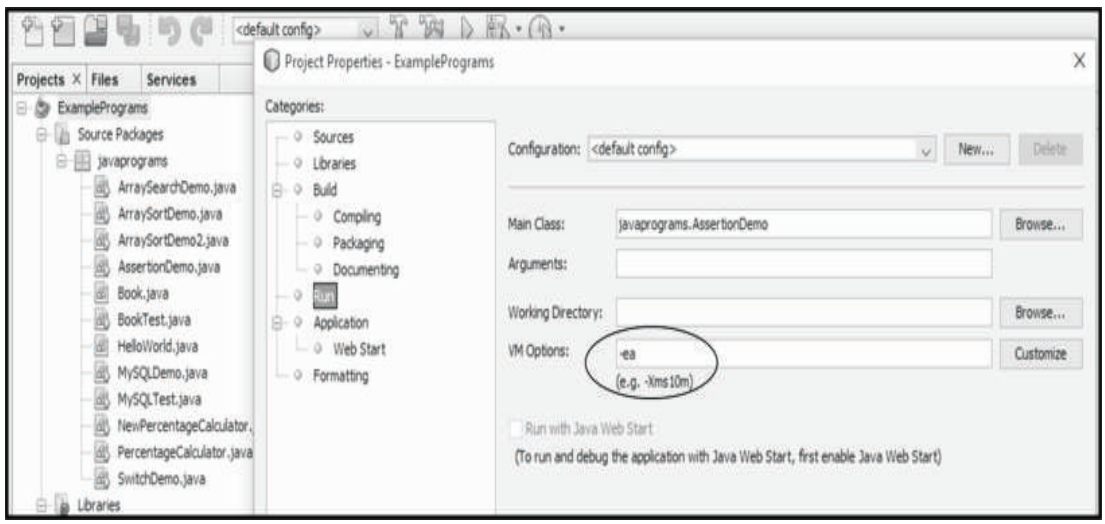


Figure 3.12(a): Enabling Assertions in NetBeans

You can see the complete code listing of the `AssertionDemo` program in Figure3.12(b) and its output in Figure 3.12(c).

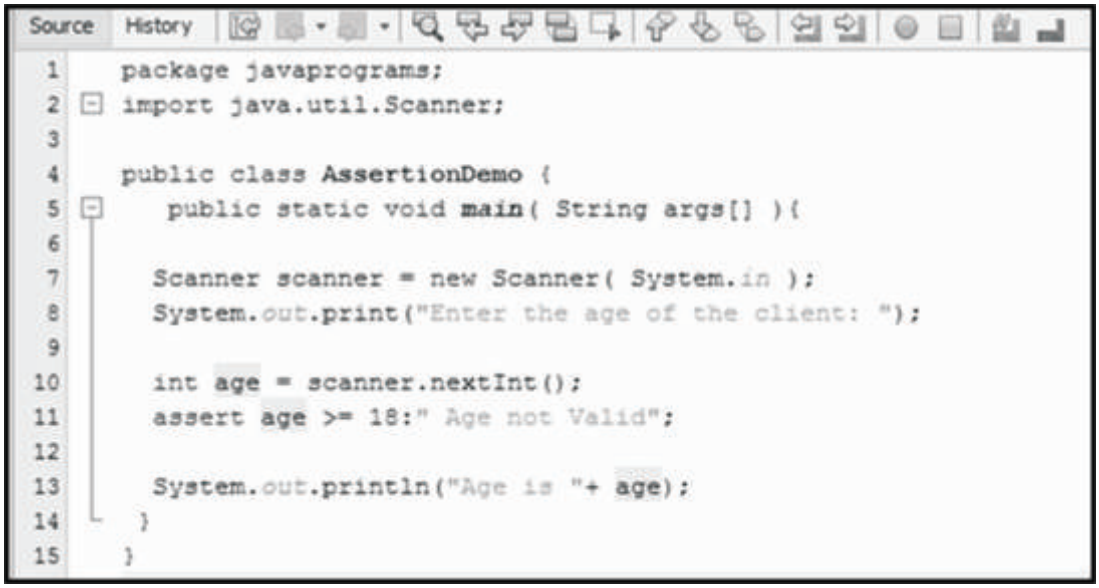


Figure 3.12(b): The AssertionDemo Program

Figure 3.12(c): Output from the AssertionDemo Program

3.12.2 Threads

A multithreaded program is one that can perform multiple tasks concurrently so that there is optimal utilization of the computer's resources. A multithreaded program consists of two or more parts called threads each of which can execute a different task independently at the same time.

In Java, threads can be created in two ways

1. By extending the Thread class
2. By implementing the Runnable interface

The first method to create a thread is to create a class that extends the Thread class from the java.lang package and override the run() method. The run() method is the entry point for every new thread that is instantiated from the class.

```
public class ExtendThread extends Thread {
   public void run() {
      System.out.println("Created a Thread");
```

```
        for (int count = 1; count <= 3; count++)
            System.out.println("Count="+count);
    }
}
```

To create a thread, instantiate it from the ExtendThread class, and to start its execution, call the start() method of the Thread class.

```
public static void main(String args[]) {
ExtendThread t1 =new ExtendThread();
t1.start();
}
```

The program in Figure 3.12(d) demonstrates creation of two threads using the ExtendThread class and Figure 3.12(e) shows its corresponding output. As you can see, the operating system alternates the execution of both the threads, that is, both the threads execute simultaneously.

Start Page | ExtendThread.java

Source | History

```
1   package javaprograms;
2
3   public class ExtendThread extends Thread {
4       public void run() {
5           System.out.println("Created a Thread");
6           for (int count = 1; count <= 3; count++)
7             System.out.println("Count="+count);
8       }
9
10      public static void main(String args[]){
11          ExtendThread t1 =new ExtendThread();
12          ExtendThread t2 =new ExtendThread();
13
14          t1.start();
15          t2.start();
16      }
17  }
```

Figure 3.12(d): The ExtendThread Demo Program

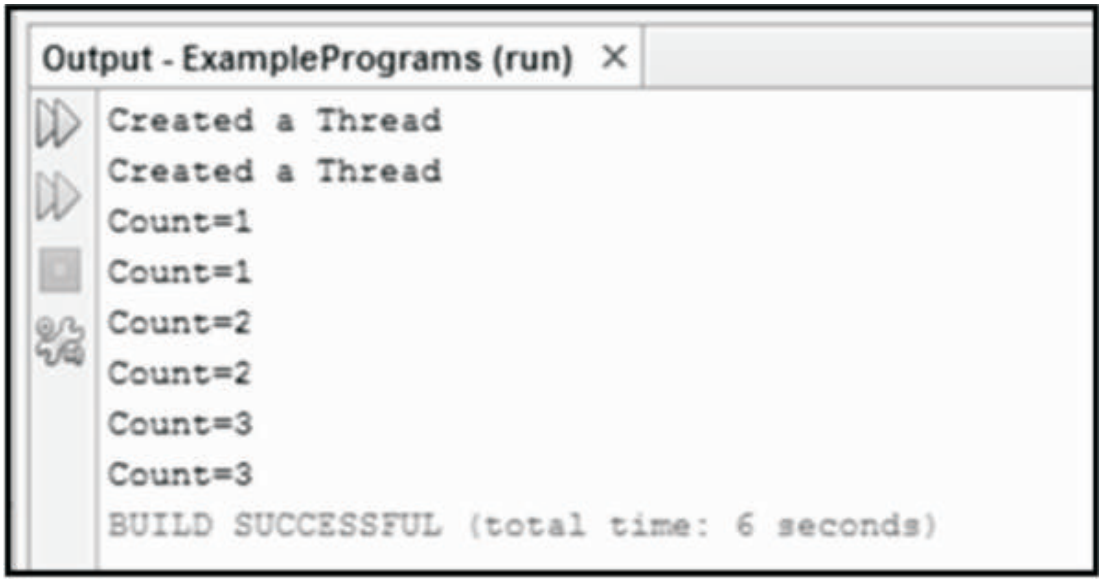


Figure 3.12(e): Output from the ExtendThread Program

Classes that you create by extending the Thread class cannot be extended further. The second method to create a thread is to create a class that implements the Runnable interface and override the run() method. Implementing the Runnable interface gives the flexibility to extend classes created by this method.

```
   public class RunnableDemoimplements Runnable {
      public void run() {
         System.out.println("Created a Thread");
         for (int count = 1; count <= 3; count++)
            System.out.println("Count="+count);
   }
}
```

To create a thread, first instantiate the class that implements the Runnable interface, then pass that object to a Thread instance. As before, to start the execution of the thread call the start() method.

```
   public static void main(String args[]){
      RunnableDemo r = new RunnableDemo();
   Thread t1 = new Thread(r);
      t1.start();
   }
```

The program in Figure 3.12(f) demonstrates creation of two threads using the RunnableDemo class. The output from the program is the same as that in Figure 3.12(e).

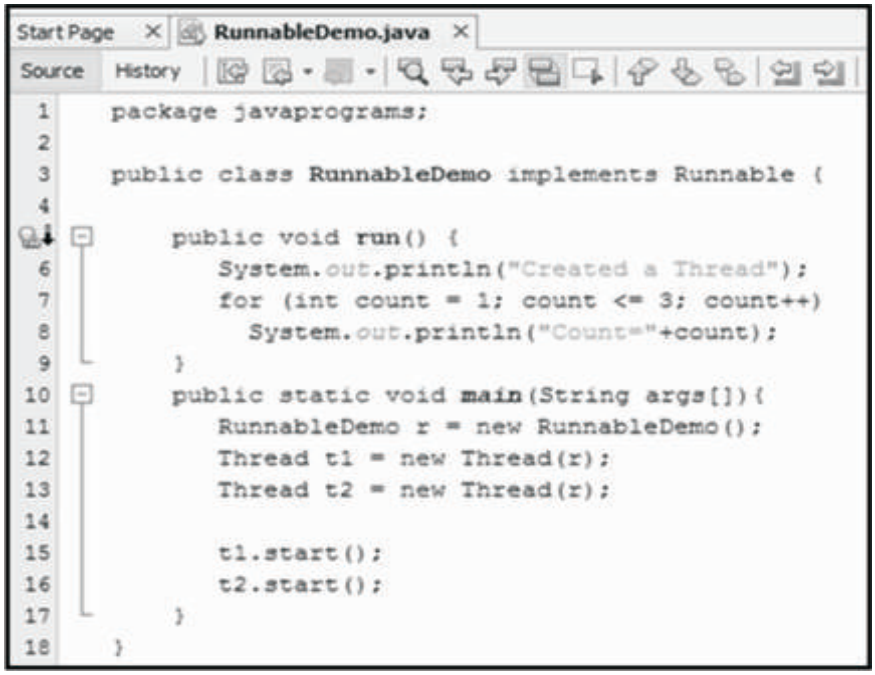


Figure 3.12(f): The RunnableDemo Program

When many threads are running, there is no guarantee of the order in which the threads will be executed. However, if you want some thread to have a higher priority than others, you can change its priority level using the `setPriority()` method of the Thread class. The Priority levels can range from 1 to 10. The `sleep()` method causes the thread to suspend execution by the specified number of milliseconds parameter. A thread can enter a wait state by invoking the `wait()` method. This method is useful when you have multiple threads running but you want one of them to start execution only when another one finishes and notifies the first one to resume execution.

3.12.3 Wrapper Classes

By default, the primitive datatypes (such as `int`, `float`, and so on) of Java are passed by value and not by reference. Sometimes, you may need to pass the primitive datatypes by reference. That is when you can use wrapper classes provided by Java. These classes wrap the primitive datatype into an object of that class. For example, the Integer wrapper class holds an int variable.

Consider the following two declarations:

```
int     a = 50;
Integer b = new Integer(50);
```

In the first declaration, an int variable is declared and initialized with the value 50. In the second declaration, an object of the class Integer is instantiated and initialized with the value 50. The variable a is a memory location and the variable b is a reference to a memory location that holds an object of the class Integer. Figure 3.12(g) illustrates the difference between an int and an Integer variable.

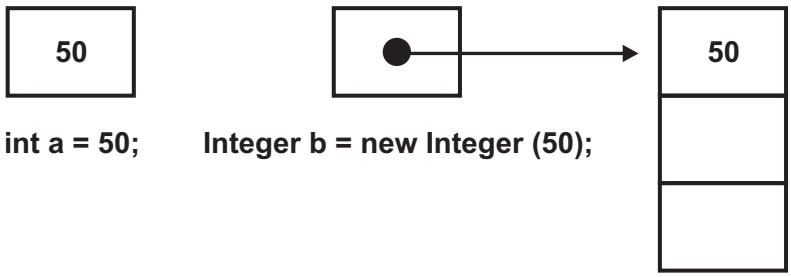


Figure 3.12(g): Memory assignment of Primitive datatype versus Wrapper Class

Access to the value of a wrapper class object can be made through getter functions defined in the class. For example, the `intValue()` member function of the Integer wrapper class allows access to the int value held in it.

```
int c = a + b.intValue();
```

Another useful function defined in the Integer wrapper class lets you convert a string into its integer value. The following statement converts the string "3456" into the integer3456 and stores it in the int variable d.

```
int d = Integer.parseInt("345");
```

Note that the `parseInt` method is a static member of the Integer class and can be accessed using the name of the class, that is, without creating an instance of the class. Similar to the parseInt method, the toString() method allows conversion from an integer value to a String as shown in the statement below.

```
String s = Integer.toString(3456);
```

Similar to the Integer wrapper class for the int datatype, each of the eight primitive types has a wrapper class defined for it (in the `java.lang` package) all of which are imported by default in Java programs. Table 3.12(g) lists the wrapper classes for the corresponding datatype.

Primitive Datatype	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
int	Integer
float	Float
double	Double
long	Long
short	Short

Table 3.12(g): Wrapper Classes

Exercise:

Q1. What is Java Bytecode?

Q2. Explain the difference between a Class and an Object with an example.

Q3. What is a constructor? Why is it used?

Q4. How are exceptions handled in Java?

Q5. How can threads be created in Java? Briefly explain with an example.

Lab Exercises:

Q1. Write a program in Java to implement the formula.

area = length * width * height

Q2. Write a program in Java to find the result of the following expressions.

(Assume a = 20 and b = 30)

i) a%b

ii) a /= b

iii) (a + b * 100) /10

iv) a && b

v) a++

Q3. Write a program in Java to print the square of every alternate number of an array.

Q4. Write a program in Java to create class Triangle with the data members base, height, area and color. The members base, height, area are of type double and color is of type string. Write getter and setter methods for base, height and color, and write method to compute_area (). Create two object of class Triangle, compute their area, and compare their area and color. If area and color both are same for the objects then display "Matching Triangles" otherwise display "Non matching Triangles".

Q5. Write a program in Java to enable user to handle divide by zero exception.

Unit - 4: Work Integrated Learning IT – DMA

4.1 Introduction

Previous chapters provided you an insight into how database management concepts are helpful for organizing the data in meaningful ways. They facilitate fast retrieval of information as and when required. You learnt data manipulation language SQL for creating, selecting and modifying the data in database. You also learnt regarding various web applications that use databases for managing the data.

In this chapter, you will learn about various work areas that use database management systems. Also, we shall develop a shopping application using database management systems and java. During the course of discussion, we have raised some questions and encourage you to think about their solution and explore related issues.

4.2 Identification of Potential Work Areas

Database management systems have found application in various domains as they provide efficient storage and fast retrieval of data. Following are a few domains where database applications may be used:

(A) Education

- For storing information such as student details, marks and result.
- For storing information about faculty and staff members.
- For storing details about school/college such as infrastructure details, department and offered course details.

(B) Banking

- For storing information about customers. For example,
  - (i) Personal details such as address, age, PAN card, occupation, contact numbers.
  - (ii) Accounts and loans related information.
  - (iii) Information regarding daily transactions.
- For storing employee details such as their personal information, salary, leaves taken, joining date, retirement year.

(C) Hospitals

- For maintaining information regarding patients such as their personal details, health records, hospitalization date.
- For storing information regarding doctors, nurses, staff members, rooms, medical equipment, and medicines.

(D) Government Sector

- For storing details of electoral roll, all types of taxes (Income tax, sales tax, house tax etc.), criminal records.
- For storing details of PAN cards, AADHAR cards, vehicle registration, birth/death certificate registration.

(E) Companies

- For storing information regarding employees such as name, address, contact number, salary, position, joining date.
- For maintaining information regarding the projects handled by them.
- For keeping track of infrastructure, sales, and investments.

(F) E-Commerce

- For storing information regarding products such as price, quantity, quality, manufacturing date, seller.
- For storing information regarding customers such as name, contact number, address and their orders.

(G) Airlines

- For storing information about flight details such as arrival time, departure time, fares, passenger capacity, number of bookings.
- For keeping track of online and offline reservations.

(H) Railways

- For storing information about train details such as arrival time, departure time, fares, passenger capacity, number of bookings.
- For keeping track of online and offline reservations.

(I) Telecommunications

- For storing information regarding communication networks, customers, call records, and their bills.
- For storing information about the plans offered and plans subscribed by the users.

(J) Hotels

- For storing information regarding guests, their check-in and check-out times, duration of stay, room number allocated.
- For keeping track of booked and available rooms.
- For storing information regarding hotel staff, menu items, and infrastructure.

Various applications in above mentioned work areas can be developed to cater their respective needs. For example,

(i) College/school management application.

(ii) Online reservation application for airline, railways, buses, movie and hotels.

(iii) Application for online tutorials and tests.

(iv) Hospital management application.

(v) Company management application.

(vi) Bank management application.

(vii) Shopping application.

(viii) Bill payment and bill generation application.

(ix) Library management application.

(x) Real estate management application.

(xi) Inventory control application.

(xii) Shop management application.

(xiii) Restaurant management application.

(xiv) Telecom management application.

(xv) Insurance management application.

4.3 A Shopping Website - A Case Study

In this section, we shall study database management application used in **XYZ Shopping Website** that allows users to shop online by providing 24 X 7 access. The website offers a wide range of products ranging from books, clothes, stationery, kitchen appliances to electronic items. A customer can select items to be purchased by putting them in shopping cart. The website also allows users to make a choice of the product by performing comparative analysis of the products. The order can be completed by choosing to pay through Cash on Delivery option, credit/debit card, or net banking. Once the payment is made, goods will be delivered at the address specified by the customer Figure 4.3(a) shows the layout of the website.

Figure 4.3(a): XYZ Shopping Website

Left panel of the website shows various categories of products. In the center, there are pictures of some of its products. Right panel shows details of offers, link to contact, login, and track order.

4.3.1 Entities Involved

Now, let us have a look at types of information required to be maintained in the database for creating the shopping application. We have identified below some of the entities that would be useful for such an application. For each entity, a separate table is to be created using **create** SQL command. The primary key is underlined in each case.

(i) CATEGORY: This table stores categories of products available. It will store information such as category id, name of category, and its description.

Schema: CATEGORY(Category_id, Name, Description)

Name	Type	Remarks
Category_id	INT(10)	Category number (Primary Key)
Name	VARCHAR(20)	Name of category
Description	VARCHAR(40)	Description of category

Figure 4.3(b): Category Table

(ii) PRODUCT: This table stores details of products available for shopping. It will store information such as product_id, name, category_id, price, quantity, discount, brand, color, size, seller_id and its description.

Schema: PRODUCT (Product_id, Name, Category_id, Price, Quantity, Discount, Brand, Color, Size, Seller_id, Description)

Name	Type	Remarks
Product_id	INT(10)	Product number (Primary Key)
Name	VARCHAR(20)	Name of the product
Category_id	INT(10)	Category to which product belongs (Foreign Key)
Price	INT(10)	Price of product
Quantity	INT(10)	Number of items left in stock

Discount	INT(3)	Discount available for the product
Brand	VARCHAR(20)	Brand to which product belongs
Color	VARCHAR(20)	Color of the product
Size	VARCHAR(20)	Size of the product
Seller_id	INT(10)	Seller of the product (Foreign Key)
Description	VARCHAR(20)	Description of product

Figure 4.3(c): Product Table

(iii) CUSTOMER: This table stores customer details specified by him either at login time or during order. It will store information such as customer_id, password, first name, last_name, address, and email_id and contact_num.

Schema: CUSTOMER (Customer_id, Password, First_name, Last_name, Address, Email_id, Contact_num)

Name	Type	Remarks
Customer_id	INT(10)	Customer number (Primary Key)
Password	VARCHAR(30)	Login password of customer
First_name	VARCHAR(20)	First name of customer
Last_name	VARCHAR(20)	Last name of customer
Address	VARCHAR(50)	Address of customer
Email_id	VARCHAR(20)	Email id of customer
Contact_num	INT(10)	Contact number of customer

Figure 4.3(d): Customer Table

(iv) SELLER: This table stores details about seller of the products. It will store information such as seller_id, and rating of seller.

Schema: SELLER (Seller_id, Rating)

Name	Type	Remarks
Seller_id	INT(10)	Seller number (Primary Key)
Rating	INT(1)	Rating of seller

Figure 4.3(e): Seller Table

(v) WISH_LIST: This table stores details about product, customer wishes to buy in future. It will store information such as customer_id and product_ids of products in his wish list. Note, the primary key of this table is a combination of both its attributes the cutomer_id and product_id.

Schema: WISH_LIST (Customer_id, Product_id)

Name	Type	Remarks
Customer_id	INT(10)	Customer number (Primary Key and Foreign Key)
Product_id	INT(10)	Product number (Primary Key and Foreign Key)

Figure 4.3(f): Wish List Table

(vi) ORDER: This table stores details of all the orders placed till date. It will store information such as order id, customer_id of user who gave the order, product_id, shipment_id, order_date, and current_status.

Schema: ORDER (Order_id, Customer_id, Product_id, Shipment_id, Order_date, Current_status)

Name	Type	Remarks
Order_id	INT(10)	Order number (Primary Key)
Customer_id	INT(10)	Customer who gave the order (Foreign Key)
Product_id	INT(50)	Product ids of products ordered (Maximum 5) (Foreign Key)
Shipment_id	INT(10)	Shipment number (Foreign Key)
Order_date	DATE	Date of order
Current_status	VARCHAR(20)	Pending or Delivered

Figure 4.3(g): Order Table

(vii) SHIPMENT: This table stores details of shipment used for dispatching a product. It will store information such as shipment_id, details and shipment_date.

Schema: SHIPMENT(Shipment_id, Details, Shipment_date)

Name	Type	Remarks
Shipment_id	INT(10)	Shipment number (Primary Key)
Details	VARCHAR(30)	Shipment details such as name, location from which order is dispatched
Shipment_date	DATE	Date of shipment

Figure 4.3(h): Shipment Table

(viii) PAYMENT: This table stores details of payment made for an order. It will store information such as payment_id, order_id, payment_amount, payment_date, and mode of payment.

Schema: PAYMENT (Payment_id, Order_id, Payment_amount, Payment_date, Mode)

Name	Type	Remarks
Payment_id	INT(10)	Payment number (Primary Key)
Order_id	INT(10)	Order number for which payment is made (Foreign Key)
Payment_amount	INT(10)	Amount of payment
Payment_date	DATE	Date of payment
Mode	VARCHAR(20)	Mode of payment such as debit card, credit card, net banking, cash on delivery, E-gift voucher.

Figure 4.3(i): Payment Table

Now, let us examine how these entities are linked with each other. Figure 4.3(j) shows relationships between these entities, in an informal manner.

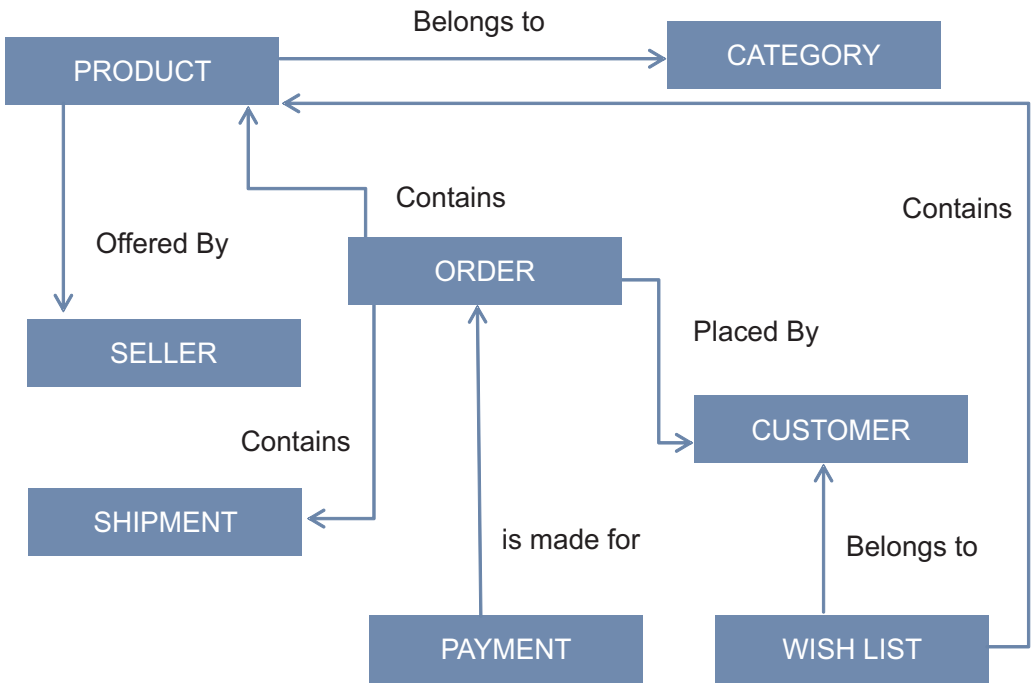


Figure 4.3(j): Relationship Between Entities

4.3.2 Functionality

Now, let us have a look at how shopping application may use and manage database. Given below are some important functions that are performed by XYZ Shopping website:

1. First, all the categories whose products are to be made available for shopping are decided by XYZ Company. For storing the finalized categories in the CATEGORY

table, function **insert_category** of class **Category** defined in Figure 4.3(k) may be used. At later point in time, if the company wishes to come up with more categories, it may do so by adding them in table using the same function. Also, the company may decide to either delete or modify the categories using functions **delete_category** and **modify_category** respectively. If the customer selects any category on the website, function **display_category** is invoked to display all the products in that category. The skeleton of class and the related functions for managing the CATEGORY table are given in Figure 4.3(k).

```
package shopping_application;
public class Category {
      public void insert_category(String categ_tuple[])
      {
            // Add functionality to insert a row of category
            // named categ_tuple.
            // SQL command to be used: insert
      }
      public void delete_category(int categ_id)
      {
            // Add functionality to delete a category with id categ_id.
            // SQL command to be used: delete
      }
      public void modify_category(int categ_id, String attr, String
      new)
      {
            // Add functionality to change value of attribute attr
            // of category with id categ_id to new.
            // SQL command to be used: update
      }
      public void display_category(int categ_id)
      {
            // Add functionality to display all the products in
            // category with id categ_id.
            // SQL command to be used: select
      }
}
```

Figure 4.3(k): Class for Table Category

QUESTION

Suppose that numbers of products belonging to a category are quite large. Customer may not be interested in some products at all. What do you think, should be the order of display of products? Should it be based on popularity, price or any other criteria? For each criterion, figure out what else is required to be stored in database.

2. Next, the list of products along with associated details are to be decided for each category and inserted in PRODUCT table using **insert_product** function of class **Product** as shown in Figure 4.3(l).Similarly, company at later point in time may add more products by calling the same function. Functions **delete_product** and

modify_product may respectively be used to delete or modify any product, as and when required. If the customer selects any product on the website, then function **display_product** may be used to display the details of product. The skeleton of class and the related functions for managing the PRODUCT table are given in Figure 4.3(l).

```
package shopping_application;
public class Product {
      public void insert_product(String prod_tuple[])
      {
            // Add functionality to insert a row of product named
            // prod_tuple.
            // SQL command to be used: insert
      }
      public void delete_product(int prod_id)
      {
            // Add functionality to delete a product with id prod_id.
            // SQL command to be used: delete
      }
      public void modify_product(int prod_id, String attr, String new)
      {
            // Add functionality to change value of attribute attr
            // of product with id prod_id to new.
            // SQL command to be used: update
      }
      public void display_product(int prod_id)
      {
            // Add functionality to display details of the products
            // with id prod_id
            // SQL command to be used: select
      }
}
```

Figure 4.3(l): Class for Table Product

QUESTION

Identify at least two other functionalities that should be supported by class Product and write routines for them.

3. The products inserted above contain **seller_id** of the Seller offering the product for selling. So, we also need to specify information regarding all the sellers in the table SELLER. This may be achieved using function **insert_seller** of class **Seller** as shown in Figure 4.3(m). Further, functions **delete_seller** and **modify_seller** may be used to delete or modify any seller's information, as and when required. If the customer selects any product on the website, then function **display_seller** may be used to display the details of seller offering the product. Suppose same product is offered by more than one seller. In such a case, you need to list all the sellers along

with their details. Here, declaring **Product_id** and **Seller_id** as the composite key will serve the purpose. The skeleton of class and the related functions for managing the SELLER table are given in Figure 4.3(m).

```
package shopping_application;
public class Seller {
      public void insert_seller(String seller_tuple[])
      {
            // Add functionality to insert a row of seller named
            // seller_tuple.
            // SQL command to be used: insert
      }
      public void delete_seller(int seller_id)
      {
            // Add functionality to delete a seller with id seller_id.
            // SQL command to be used: delete
      }
      public void modify_seller(int seller_id, String attr, String new)
      {
            // Add functionality to change value of attribute attr
            // of seller with id seller_id to new.
            // SQL command to be used: update
      }
      public void display_seller(int seller_id)
      {
            // Add functionality to display details of the seller
            // with id seller_id
            // SQL command to be used: select
      }
}
```

Figure 4.3(m): Class for Table Seller

QUESTION

Suppose a customer want to provide rating to a seller. Is there any need of special routine for that? Specify how can you achieve this.

4. Whenever a customer visits the shopping website, he can either sign in or can specify his details while placing the order. If he is the existing user, he may login using interface given in Figure 4.3(n). If the customer is not yet registered, and wants to get registered, he may use the interface provided in Figure 4.3(o). It may be noted that almost same interface (excluding password field) is used when unregistered user specifies his details during placement of order. The details specified by user are saved into CUSTOMER table using function **insert_customer** of class **Customer**. These details are used whenever customer makes any order. For example, shipping address specified can be used for delivery purpose. At any point in time, customer may change his personal details. Function **modify_customer** may be used for the same.

Figure 4.3(n): Login Interface

Figure 4.3(o): Sign Up Interface

The skeleton of class and the related functions for managing the CUSTOMER table is given in Figure 4.3(p).

```
package shopping_application;
public class Customer {
      public void insert_customer(String customer_tuple[])
      {
            // Add functionality to insert a row of customer named
            // customer_tuple.
            // SQL command to be used: insert
      }
      public void modify_customer(int customer_id, String attr, String
      new)
      {
            // Add functionality to change value of attribute attr
            // of customer with id customer_id to new.
            // SQL command to be used: update
      }
}
```

Figure 4.3(p): Class for Table Customer

QUESTION

Suppose a customer wants to delete his account. What kind of functionality needs to be added to the class Customer? Specify the routine for it.

QUESTION

Figure out what extra information regarding customer can be added to the database CUSTOMER. Specify how this information can be used. Is it possible to write any more functions associated with the newly added information?

5. While browsing products on the website, customers may wish to buy certain products at a later point in time. At this point, he may add certain products to the wish list. For achieving this, function **insert_wish** of class **WishList** may be invoked. Also, customer may want to delete certain products from the wish list. This may be achieved using function **delete_wish**. At later point in time, customer may view his wish list using function **display_wish**. The skeleton of class and the related functions for managing the WISH LIST table are given in Figure 4.3(q).

```
package shopping_application;
public class WishList{
      public void insert_wish(String wish_tuple[])
      {
            // Add functionality to insert a row of wish named
            // wish_tuple.
            // SQL command to be used: insert
      }
      public void delete_wish(int customer_id, String wish)
      {
            // Add functionality to delete a wish of customer with id
            // customer_id.
            // SQL command to be used: delete
      }

      public void display_wish(int customer_id)
      {
            // Add functionality to display wish list of customer with
            // id customer_id.
            // SQL command to be used: select
      }
}
```

Figure 4.3(q): Class for Table Wish List

6. When a customer adds products to be purchased in the shopping cart. He has to go through series of events as described in Figure 4.3(r). In case the customer is not already logged in, he is asked to either login or provide his details. After that, he may choose to pay through any of the available payment modes as specified in Figure 4.3(s). All the details of payment are stored in table PAYMENT using function **insert_payment** of class **Payment**. In case, customer choses to pay cash on

delivery, attribute **Payment_date** is set to the date on which product is delivered. The skeleton of class and the related functions for managing the PAYMENT table are given in Figure 4.3(s).

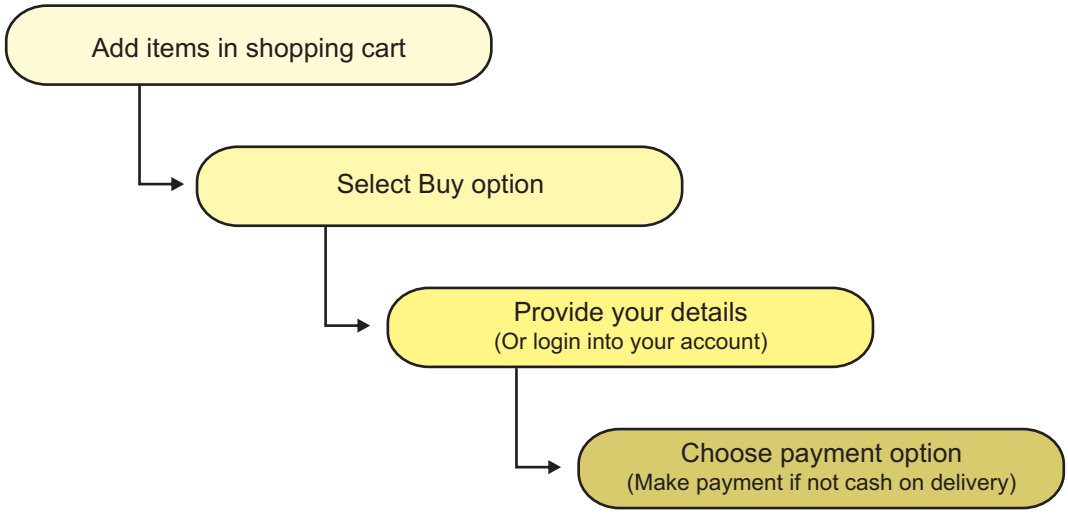


Figure 4.3(r): Sequences of events for Order Completion

```
package shopping_application;
public class Payment{
      public void insert_payment(String payment_tuple[])
      {
            // Add functionality to insert a row of payment named
            // payment_tuple.
            // SQL command to be used: insert
      }
}
```

Figure 4.3(s): Class for Table Payment

QUESTION

Suppose if the customer cancels the order, payment should be refunded to him. For achieving this, is there any need to make changes in PAYMENT table. If yes, specify those changes. Also, write a function

7. All the details of order are stored in table ORDER using function **insert_order** of class **Order** as shown in Figure 4.3(t). If user wishes to cancel order, he can do so by choosing cancel order option. In such a scenario, function **delete_order** will be invoked. If user wants to track his order, he can do so by providing the **order_id**. After doing so, function **display_order** will display all the relevant details of his order along with current status. The skeleton of class and the related functions for managing the ORDER table are defined in Figure 4.3(t).

```
package shopping_application;
public class Order{
      public void insert_order(String order_tuple[])
      {
            // Add functionality to insert a row of order named
            // order_tuple.
            // SQL command to be used: insert
      }
      public void delete_order(int order_id)
      {
            // Add functionality to delete order with id order_id.
            // SQL command to be used: delete
      }
      public void display_order(int order_id)
      {
            // Add functionality to display details of the order with
            // id order_id.
            // SQL command to be used: select
      }
}
```

Figure 4.3(t): Class for Table ORDER

QUESTION

Suppose, number of products specified by user while making order, exceeds five. How will you take care of this situation in function insert_order of class Order?

8. Finally, details of shipment of order are stored in table SHIPMENT using function **insert_shipment_details** of class **Shipment**. The skeleton of class and the related functions for managing the SHIPMENT table are defined in Figure 4.3(u).

```
package shopping_application;
public class Shipment{
      public void insert_shipment_details(String order_shipment[])
      {
            // Add functionality to insert a row of shipment named
            // order_shipment.
            // SQL command to be used: insert
      }
}
```

Figure 4.3(u): Class for Table SHIPMENT

QUESTION

Is there any need of storing shipment details in another table? Can table ORDER be used for storing the same? What if this table is deleted from the database? Can the website still function without any flaw?

Exercise:

Q1. Choose at least one more application out of the ones that are listed at the beginning of chapter. Identify the data requirements and functionality to store and manage the data. Write commands for creating the table for each identified entity. Also, describe the relationship amongst these entities. Write your own java routines for the functionality you have identified.

Appendix - A

Installing and Staring NetBeans IDE

To install NetBeans IDE, first you have to download it and then follow the instructions given below to install it.

Step 1: Visit **https://netbeans.org/downloads/** in your browser.

Step 2: Click to select the appropriate NetBeans bundle to download (Figure 1).

Figure 1: Select the Appropriate NetBeans Bundle to Download

Step 3: Once you click on the Download button the screen as per Figure 2 will appear and the download will start automatically.

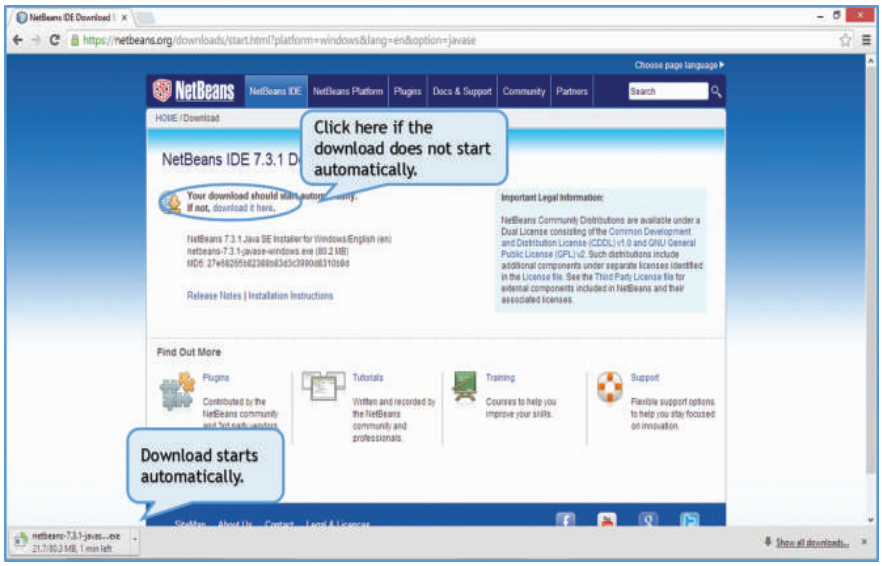


Figure 2: Download Starts Automatically

Step 4: Once the download completes, use the File Explorer to navigate to the file that was just downloaded. Double click on the file name to start the installation process (Figure 3).

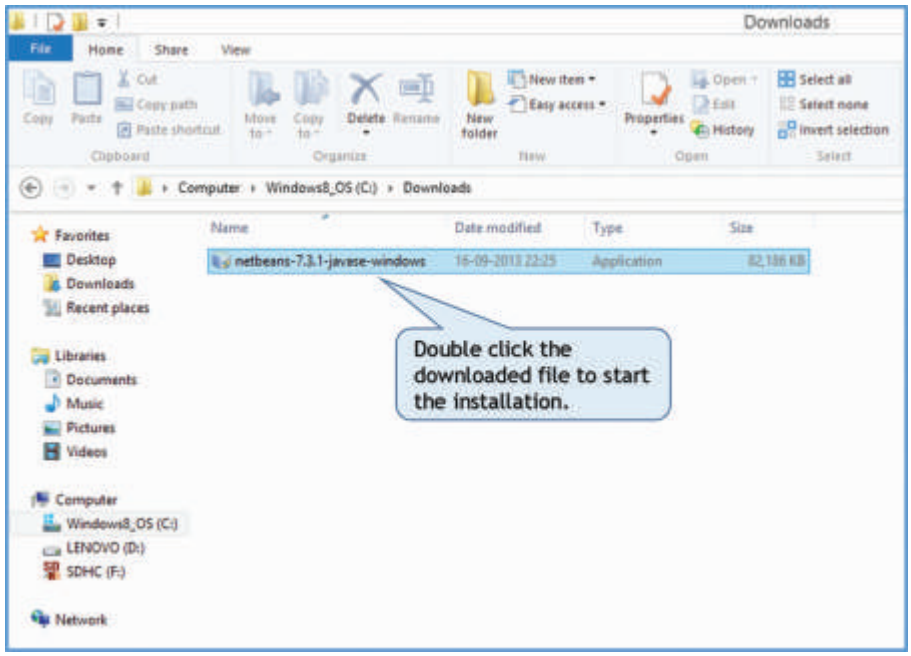


Figure 3: Double click on the file name to start the installation process

Step 5: You might see a User Account Control Window asking whether you want the program to make changes to your computer. Click on the "Yes" button. Once the installation begins, you will see the NetBeans IDE Installer window as in Figure 4.

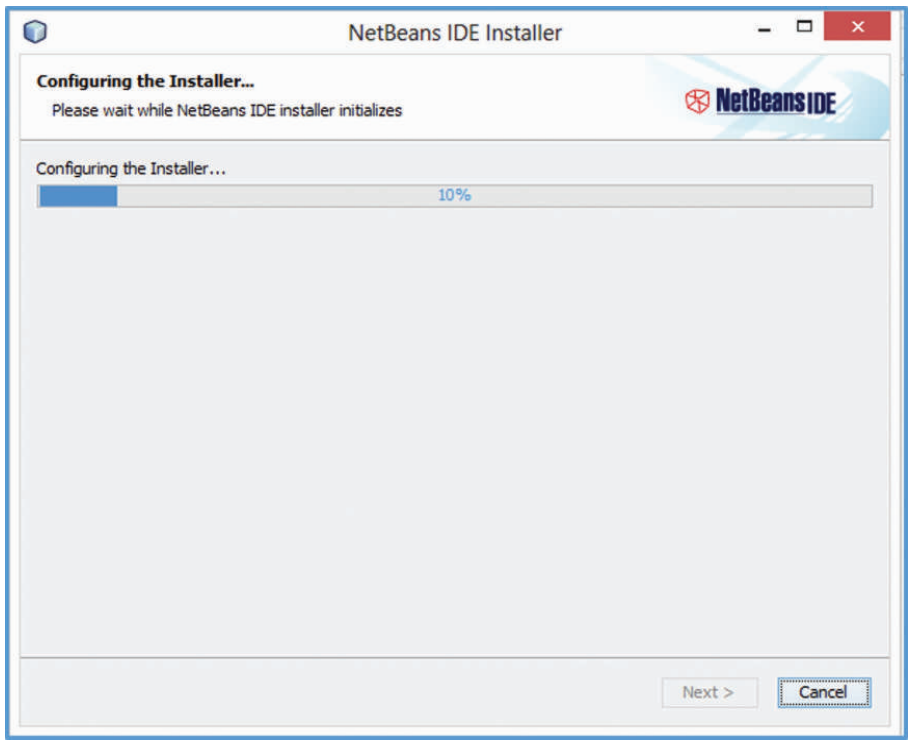


Figure 4: NetBeans IDE Installer

Step 6: Once the NetBeans IDE Installer has been configured, the Welcome screen appears, Click on Next (Figure 5).

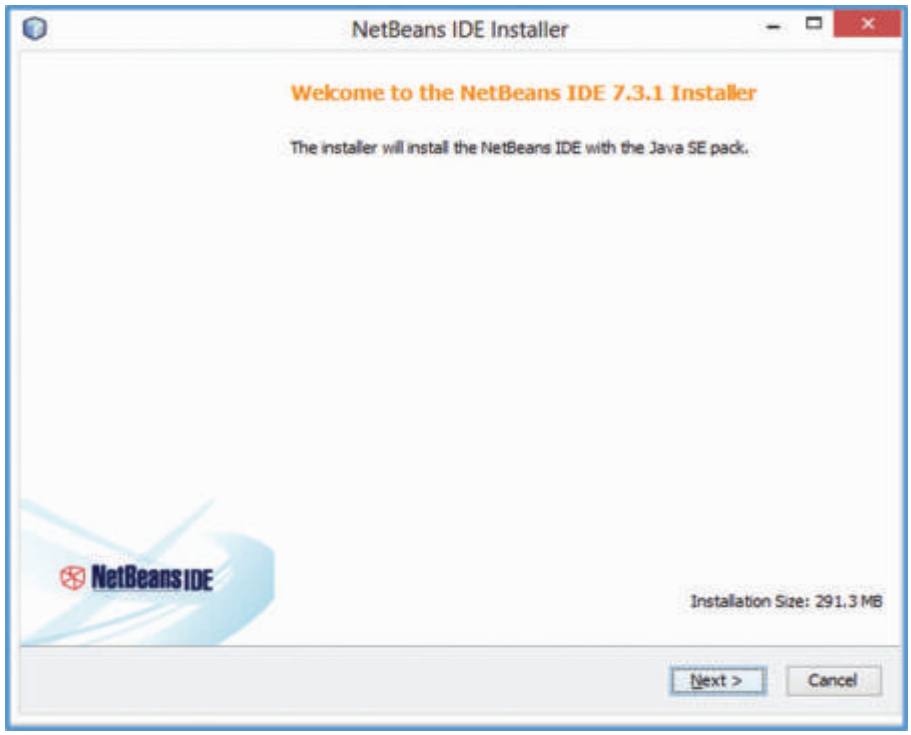


Figure 5: NetBeans IDE Installer Welcome Screen

Step 7: Check the selection box to Accept the License Agreement and click on Next (Figure 6).

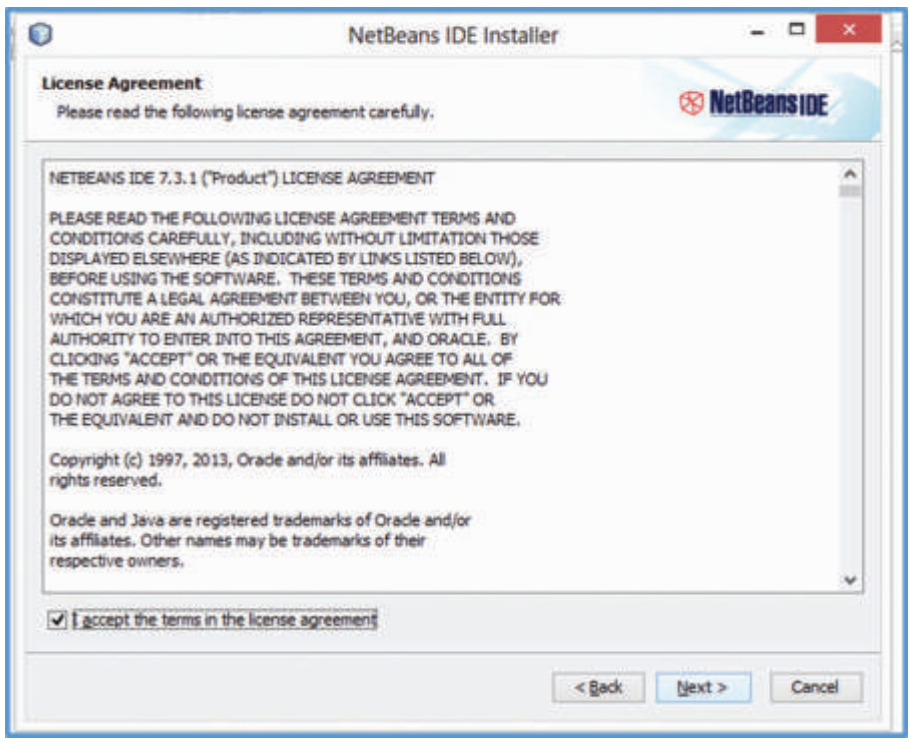


Figure 6: Accept the License Agreement

Step 8: In the JUnit License Agreement screen that appears, Click on Next (Figure 7).

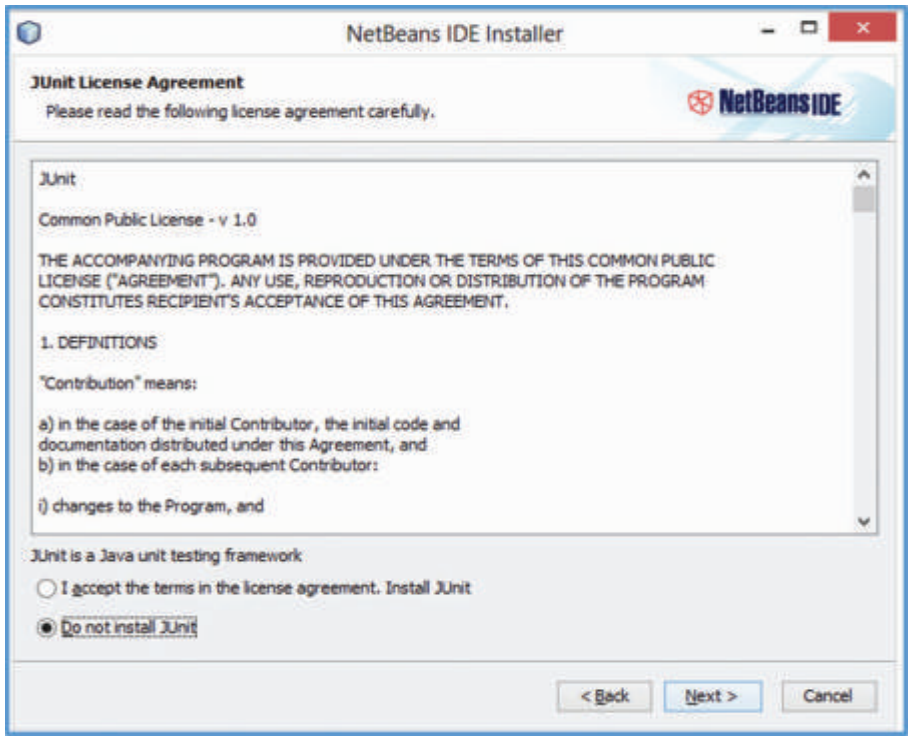


Figure 7: JUnit License Agreement

Step 9: Choose the Installation folder and Click on Next (Figure 8).

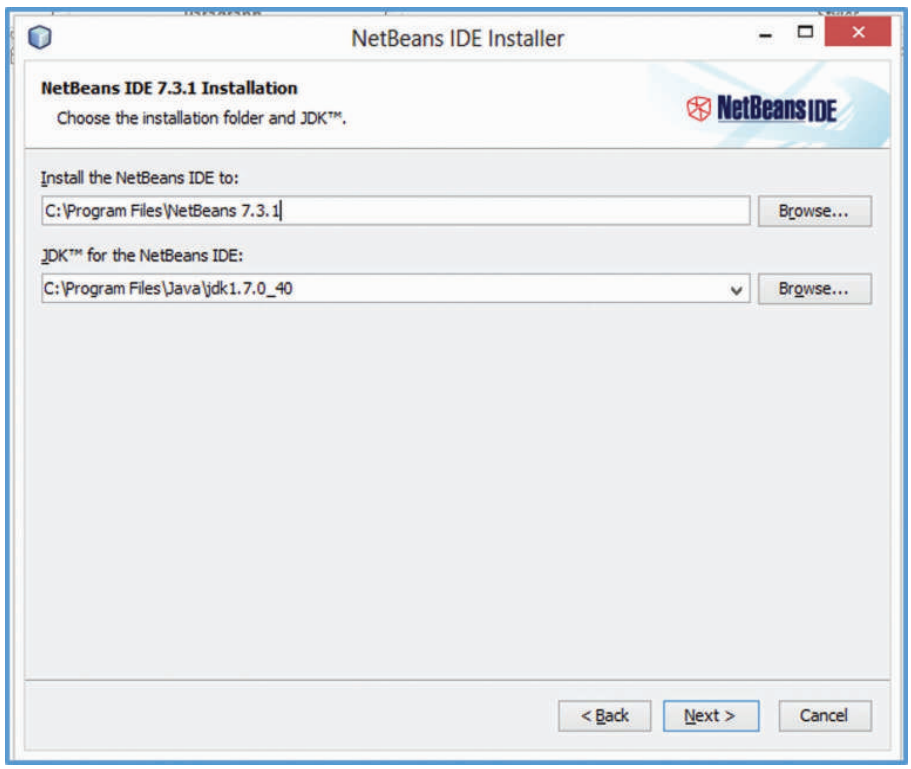


Figure 8: Choose the Installation folder

Step 10: Click the Install button to begin the Installation (Figure 9). The installation will begin as shown in Figure 10.

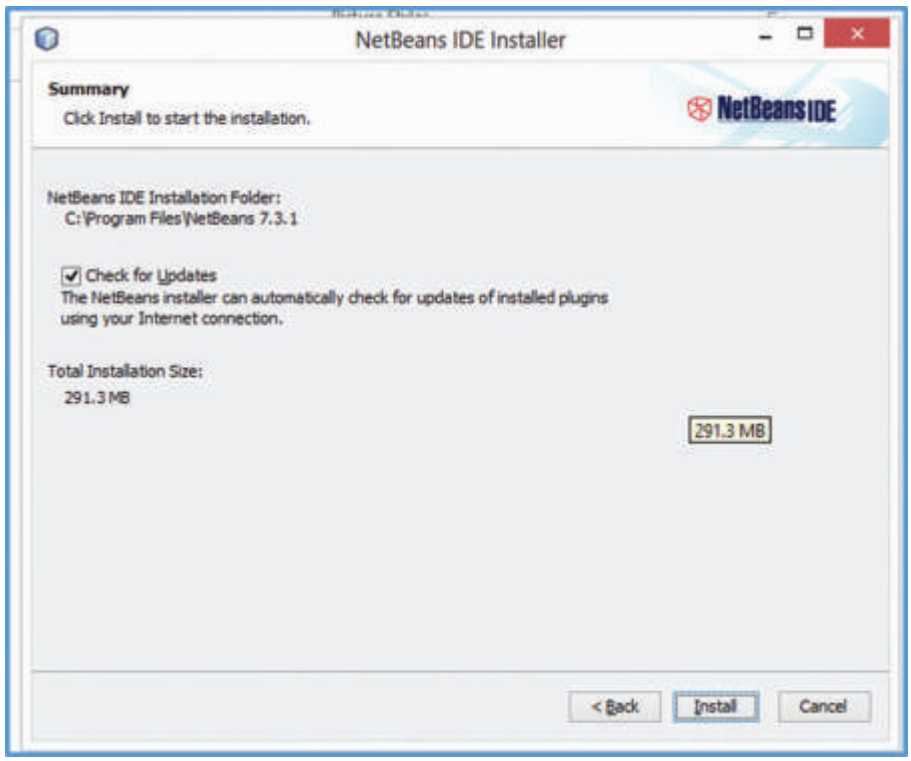


Figure 9: Begin the Installation

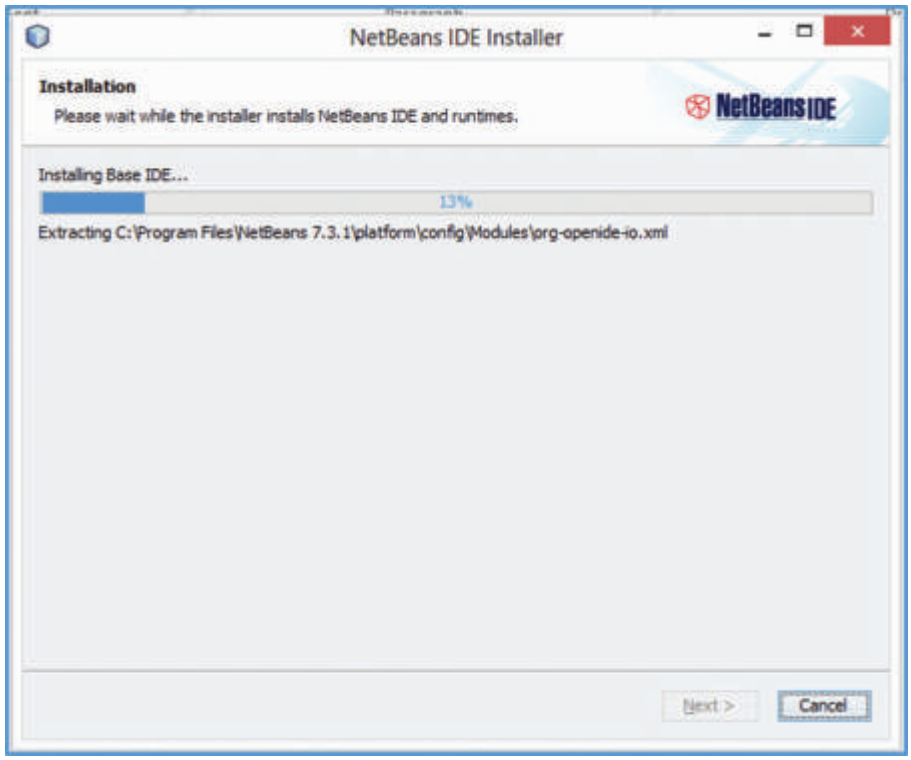


Figure 10: Installing Beans IDE

Step 11: Click on Finish to complete the Installation Process (Figure 11).

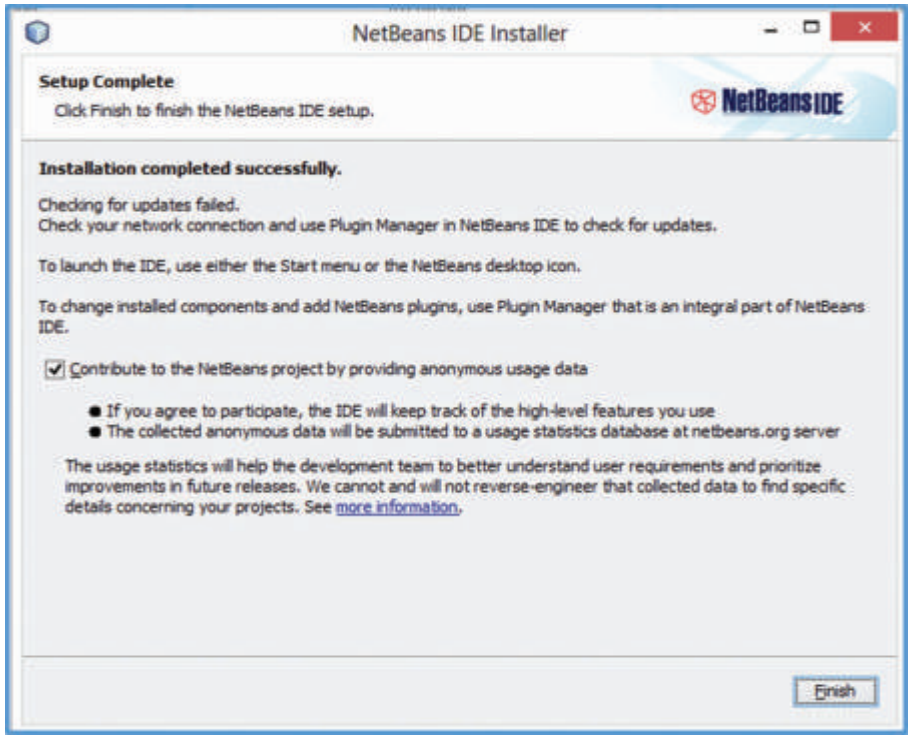


Figure 11: Finish to complete the Installation Process

Step 12: An icon to start NetBeans will be installed on your Desktop (Figure 12). Double click the icon to start the Java NetBeans IDE.

Figure 12: Double click the icon to start the Java NetBeans IDE

The required modules are loaded and NetBeans IDE started (Figure 13).

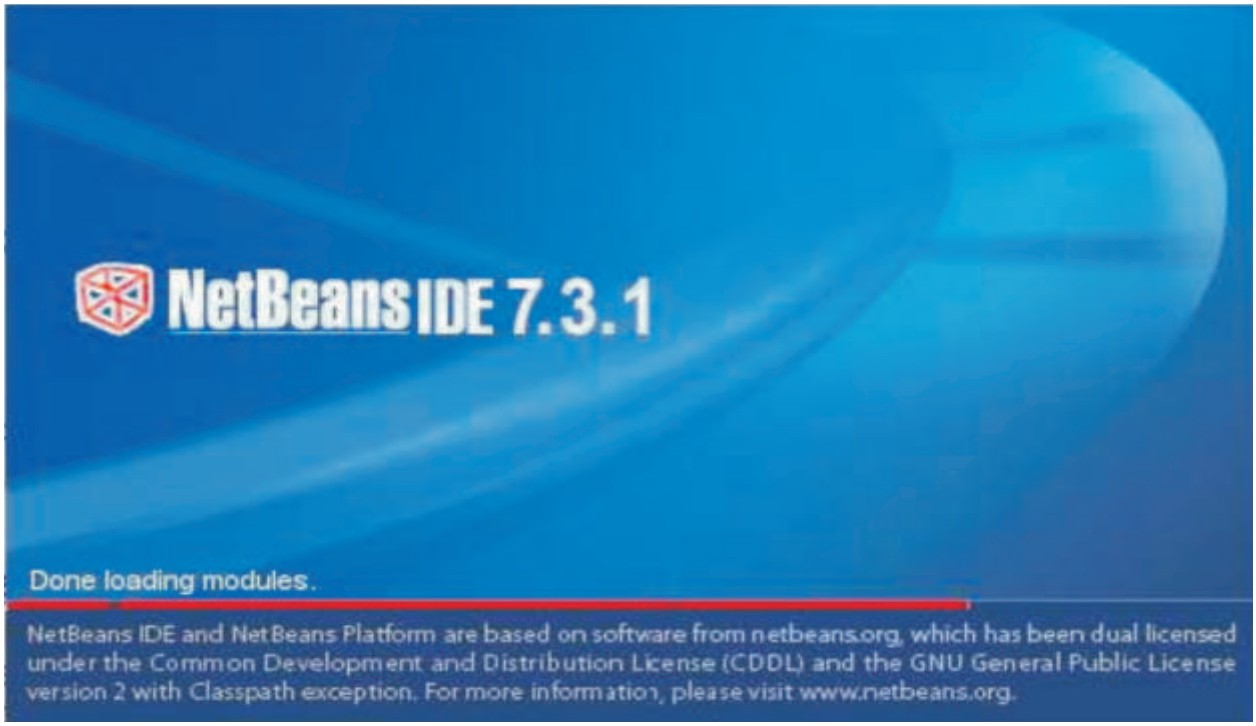


Figure 13: Loading Modules of NetBeans IDE

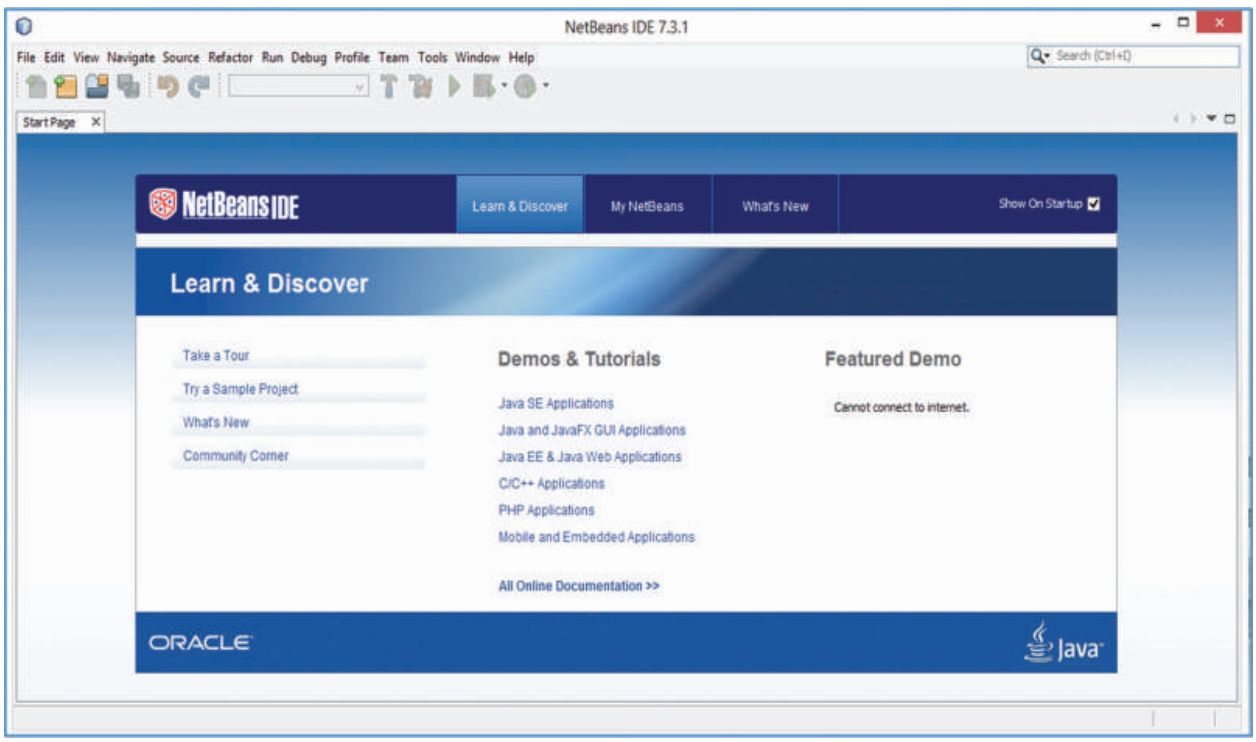


Figure 14: NetBeans IDE

Finally, the NetBeansIDE will be presented to you (Figure 14). As a beginner, you must take some time to familiarize yourself with the interface, and go through the appropriate demos and tutorials.

•BUILDING SKILLS•BUILDING FUTURE•

CENTRAL BOARD OF SECONDARY EDUCATION

Shiksha Kendra, 2, Community Centre, Preet Vihar, Delhi-110301
Tel: 011-22527183 • Fax: 011-22526580
E-mail: voc.cbse@gmail.com • Website: www.cbsevocational.in