# Programming in Java

Sachin Malhotra
*Associate Professor*
*IMS Ghaziabad*

Saurabh Choudhary
*IT Consultant and Corporate Trainer*

*Former Head, IT Department*
*IMS Ghaziabad*



OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trade mark of
Oxford University Press in the UK and in certain other countries.

Published in India by
Oxford University Press
Ground Floor, 2/11, Ansari Road, Daryaganj, New Delhi 110002, India



First Edition published in 2010
Second Edition published in 2014
Revised Second Edition published in 2018



ISBN-13: 978-0-19-948414-0
ISBN-10: 0-19-948414-7

Typeset in Times New Roman
by Ideal Publishing Solutions, Delhi
Printed in India by Magic International (P) Ltd., Greater Noida

# Features of the Book

**Example 16.2 First Servlet Program**

```
L1  import javax.servlet.*;
L2  import javax.servlet.http.*;
L3  import java.io.*;
L4  public class FirstServlet extends HttpServlet
    {
L5  public void service(ServletRequest req, Servlet Response res) throws
    ServletException,IOException
    {
L6      res.setContentType("text/plain");
L7      PrintWriter pw = res.getWriter();
L8      pw.println("My First Servlet is running");
    }}
```

**Explanation**

**L1–3** The packages `javax.servlet.*`, `javax.servlet.http.*`, and `java.io.*`have to be imported to create an `HttpServlet`.

**L4** The class `FirstServlet` must be a public class and it must inherit `HttpServlet`, as http protocol is used for communication between client and server. So to handle http request from client and generate http response for client, we have to create an `HttpServlet`. data to be sent to the client has to specified with the help of a method `res.setContentType ("text/plain")`. In other words, the MIME type (stands for multipurpose internet mail extension) has to set. Nowadays, web pages contain text, images, and multimedia. A servlet informs the browser about the type of data it will be sending to browser. The servlet in our example is transmitting plain text, so

**Programs are followed by line-by-line explanations to provide an in-depth understanding of the whole program**

**16.5 INTRODUCTION TO JAVA SERVER PAGES**

Java server pages(JSP), in contrast to servlets, is basically a page that contains embedded within html tags. Servlet is a Java program where html tags are embedd code or html responses are generated through Java. JSP files have an extension. js

**16.6 JAVA BEANS**

Java beans provides a standard format for writing Java classes. Java bean is a reusab component. Once it is designed and created, it can be used over and over again in mar applications as per their requirements. Java Beans can be used by IDE and other Ja

**16.3 SERVLETS**

Servlets are Java server-side programs that accept client's request (usually http reques them and generate (usually http response) responses. The requests originate from cl browser and are routed to a servlet located inside an appropriate webserver. Servle

**16.8 REMOTE METHOD INVOCATION**

Distributed computing allows parts of the system to be residing in separate machines different places. It allows business logic and data to be accessed from remote location anywhere by any one. RMI helps in accomplishing this by allowing objects runni machine to be accessed by the clients running in different machines.

**The last chapter covers topics related to Advanced Java. It provides a glimpse of JSP, database handling, RMI, Servlets, and Java Beans**

**Key notes in the text highlight important concepts**

**Note** Higher priority threads will always preempt the lower priority threads. Actu
how the priorities of threads set by the JVM are mapped to the operating
happen that a higher priority might not be considered higher by the operatir
actually depends on the operating system and it varies from OS to another.

## 14.14 PRACTICAL PROBLEM: CITY MAP APPLET

`CityMap` applet shows map of a city (top view) with five buttons namely hospitals, shopping malls, police station, post office, and stadium. If a user presses the hospital button, all hospitals are shown on the map with a specific color and likewise for malls, police station, post office and stadium.

**Example 14.18** `CityMap.java`

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

/*<applet code = "CityMap2.class" width=650 height=600></applet>*/

public class CityMap2 extends Applet
 {
   Button b1,b2,b3,b4,b5;
```

**Practical programming examples to showcase how the concepts discussed in a particular chapter are implemented in practice**

### Programming Exercises

1. Write a program to connect to a database and retrieve all the data. The database type (Access or Oracle), driver name, database name, DSN, etc. have to be fed by the user.
2. Write a servlet program that fetches all the data from client and stores it in a database
4. Write a servlet that automatically redirects the client to another page.
5. Write a servlet that ensures authenticated users have access to important pages. The user name and password should be stored in a database and whenever a user tries to access the servlet,

### Review Questions

1. What is the difference between `Statement`, `PreparedStatement`, and `CallableStatement`?
2. Explain the different types of JDBC drivers.
3. Explain the lifecycle of a Servlet.
4. Differentiate get and post requests.
5. Explain the role of registry services in RMI.
6. Explain the following:
   (a) Http Redirects
   (b) Cookie
   (c) Stubs and skeletons
   (d) ResultSet
   (e) ResultSet metadata
7. Explain all the steps used for establishing a

### Objective Questions

1. Which packages contain the JDBC API?
   (a) java.jdbc (b) java.sql
   (c) javax.jdbc (d) javax.sql
2. Which class is used to establish a database connection?
   (a) Class (b) DriverManager
   (c) Statement (d) ResultSet
4. Which of the following is used for calling stored procedures?
   (a) Statement
   (b) PreparedStatement
   (c) CallableStatement
   (d) Connection
5. Which of the following methods return a Connec-

**A variety of chapter-end exercises that include both subjective as well as objective questions**

# Preface to the Revised Second Edition

Java is an easy-to-learn, versatile, robust, portable, and secure language with rich user interfaces. It has set up new benchmarks in the software development world ranging from desktop to web-based enterprise applications to mobile and embedded applications. Since its inception in 1995, it has come a long way by continuously evolving itself and, in the process, changing the style of programming the world over. Java is not only found in laptops or data centres, it is also widely used in cell phones, SIM cards, smart cards, printers, routers and switches, set-top boxes, ATMs, navigation systems, to name a few, and remains one of the top choices for most popular Cloud platforms. According to the latest and most popular programming indexes and ranking, Java continues to be the preferred choice of developers.

This revised second edition of *Programming in Java* conforms to Java Standard Edition 8. It is significant in the sense that this major release comes bundled with plenty of enhancements which were long overdue. To list a few noticeable enhancements, Java 8 includes support for functional programming, lambdas, enhancements to interfaces – default and static methods, and much more. These new topics are appropriately explained in separate appendices with suitable examples.



## New to the Revised Second Edition

This edition has been updated to provide greater topical coverage as well as to incorporate Java 8 enhancements. The most noticeable changes are as follows:

- Appendices on major Java 8 enhancements – functional programming with lambdas and static and default methods in interfaces.
- Appendices on regular expressions, stack and heap usage in Java, and differences between pointers and references
- This edition is supplemented with a rich online resource centre that provides a "Prelude to Java 9"

## Key Features

The most prominent feature of this book has been the line-by-line explanation section under each program. They facilitate in-depth understanding of the whole program. We have retained this feature in the revised second edition as well. It has been well appreciated by the users. Other noticeable features include the following:

- A recap of object-oriented programming concepts before introducing the concepts of Java
- Plenty of user-friendly programs and key notes at appropriate places to highlight important concepts
- A variety of chapter-end exercises that include subjective as well as objective questions

## Content and Structure

This book comprises 16 chapters and seven appendices. A brief outline of each chapter is as follows.

*Chapter 1* focuses on the object-oriented concepts and principles. It provides real-life mapping of concepts and principles besides depicting them pictorially. In addition to this, the chapter also provides an introduction to Unified Modeling Language (UML), which is a modeling language to show classes, objects, and their relationship with other objects.

*Chapter 2* introduces Java and its evolution from its inception to its current state. Besides introducing the features of Java, it also tells you about the structure of JDK (Java Development Kit) and the enhancements made to Java in its latest versions. It describes how to install and run the JDK that is in turn required for executing a Java program.

*Chapter 3* describes the basic programming constructs used in Java such as variables, datatypes, and identifiers. Java reserved keywords are also depicted in this chapter. The operators (arithmetic, relational, boolean, etc.) that act on variables are also explained in this chapter. For each set of operators, we have provided sufficient examples along with their explanation and output. Apart from variables and operators, this chapter focuses on statements like if and other loops available in Java (for, while, do…while, and for...each).

*Chapter 4* deals with classes and objects. A lot of practical problems and their solutions have been discussed in this chapter. It begins with how to define classes, objects, and method creation. Method overloading is also discussed. Later, it emphasizes on the differences between instance variables/methods and class variables and methods. Finally, a discussion about arrays, this keyword, and command-line arguments is also provided.

*Chapter 5* focuses on inheritance and its uses. How it is realized in Java is discussed in this chapter. Apart from this, polymorphism concepts are visualized through method overriding and super keyword. How practical programming problems are solved through super keyword forms a major part of this chapter. Towards the end of the chapter, some related concepts like abstract classes are also discussed.

*Chapter 6* covers interfaces, packages, and enumeration. It highlights the differences between abstract classes and interfaces and their practical usages with examples. The role of packages in Java and their creation and usage is also discussed. In-depth coverage of a predefined package *java.lang* is included in this chapter along with some of the famous classes such as String, StringBuffer, StringBuilder, and Wrapper classes.

*Chapter 7* discusses exceptions in detail. Apart from explaining in detail the five keywords (try, catch, throw, throws, and finally) used in handling exceptions, it also discusses how a user can create his own exceptions and handle them. Concepts such as exception, encapsulation, and enrichment are also explained in this chapter. Besides these, the new facilities provided by Java like assertions and logging are also discussed.

*Chapter 8* covers multithreading concepts, its states, and priorities. It also discusses in detail the inter-thread communication and synchronization concepts. Methods, such as wait(), notify(), and notifyAll(), have also been discussed.

*Chapter 9* emphasizes on the essentials of I/O concepts like how standard input can be taken and how output is delivered to the standard output. A few main classes of the java.io package are discussed with examples and their usages. Console class, used for taking user input, is also

discussed. What is the use of making objects persistent and how will it be done is discussed towards the end of the chapter.

*Chapter 10* discusses the java.util package in detail. Interfaces, such as Map, Set, and List, have been discussed in detail as well as their subclasses such as LinkedList, ArrayList, Vector,HashSet, HashMap, TreeMap. Java 5 introduced a new feature named 'Generics' which forms the core of the *java.util* package. This concept along with its application has been covered in detail.

*Chapter 11* explains how network programming can be done in Java. In-depth coverage of sockets is extended in this chapter. Client and server concept is illustrated by the programs created. TCP and UDP clients and server and their interactions are demonstrated. The concept of multithreading is merged with socket and illustrated to create server programs. Some main classes such as URL, URL connection, and network interface are also discussed.

*Chapter 12* focuses on applets, its lifecycle, methods, and how they are different from applications. Besides providing an in-depth coverage of *java.applet* package, some of the classes of *java.awt* package are also discussed as they are very useful in creating applets such as Graphics class, Font class, Color class, and FontMetric class. All these classes are discussed and supported by an example for each of them.

*Chapter 13* talks about event handling in Java. Basically for creating effective GUI applications, we need to handle events and this forms the basis of this chapter. The event handling model is not only discussed but applied throughout the chapter. All the approaches to event handling have been discussed such as Listener interfaces, Adapter classes, inner classes, and anonymous inner classes.

*Chapter 14* focuses on GUI creation through *java.awt* package. It has an in-depth coverage of containers and components. Containers, such as Frame and Window, and components, such as Label, Button, TextField, Choice, Checkbox, and List, are discussed in detail. How the components can be arranged in a container is also discussed.

*Chapter 15* shows how to create more advanced and lightweight GUI applications in Java. More advanced layouts such as SpringLayout have been discussed. Lightweight components, such as JButton, JLabel, JCheckBox, JToggleButton, JList, JScrollPane, and JTabbedPane, have been discussed. How to create Dialogs is also discussed. The pluggable look and feel of Java is explained in detail.

*Chapter 16* focuses on advanced Java concepts such as servlets, JDBC, and RMI. An introduction to the advanced technologies has been discussed. This chapter is equipped with numerous figures showing how to install the necessary software required for executing an advanced Java program. The chapter also provides a step-by-step and simplified approach on how to learn advanced concepts.

*Appendix A* demystifies the *this* keyword and discusses its internal working.

*Appendix B* discusses in detail the stack and heap memory usage in Java with practical implementation.

*Appendix C* gives a brief comparison of Java references vis-à-vis C pointers.

*Appendix D* provides a brief discussion of pattern matching using regular expressions.

*Appendix E* provides a step-by-step introduction to default and static methods introduced in Java 8 interfaces.

*Appendix F* provides an in-depth coverage of functional programming and its use through lambdas in Java.

*Appendix G* includes a list of interview questions along with their answers which provides an overview of the industry scenario and their requirements.

## ACKNOWLEDGEMENTS

Several people have been instrumental throughout this tiring, yet wonderful journey. First of all, we would like to express our sincere gratitude to our families without whose support, patience, and cooperation, this book would not have been possible and we would not have been what we are today.

We are extremely thankful to our fraternal colleagues and friends for their endless support, motivation and suggestions in revising this book. We would also like to thank our readers for their valuable feedback, which has helped us in shaping this edition.

**Sachin Malhotra**
**Saurabh Choudhary**

# Detailed Contents

*Preface to the Revised Second Edition vi*

**1. Introduction to OOP 1**

1.1 Introduction 1
1.2 Need of Object-Oriented Programming 2
1.2.1 Procedural Languages *2*
1.2.2 Object-Oriented Modeling *2*
1.3 Principles of Object-Oriented Languages 3
1.3.1 Classes *3*
1.3.2 Objects *3*
1.3.3 Abstraction *3*
1.3.4 Inheritance *4*
1.3.5 Encapsulation *4*
1.3.6 Polymorphism *5*
1.4 Procedural Language vs OOP 5
1.5 OOAD Using UML 6
1.6 Applications of OOP 9

**2. Getting Started With Java 12**

2.1 Introduction 12
2.2 History of Java 13
2.3 Java's Journey: From Embedded Systems to Middle-Tier Applications 13
2.4 Java Essentials 14
2.5 Java Virtual Machine 15
2.6 Java Features 16
2.6.1 Platform Independence *16*
2.6.2 Object Oriented *16*
2.6.3 Both Compiled and Interpreted *17*
2.6.4 Java is Robust *18*
2.6.5 Java Language Security Features *18*
2.6.6 Java is Multithreaded *20*
2.6.7 Other Features *20*
2.7 Program Structure 21
2.7.1 How to Execute a Java Program *21*
2.7.2 Why Save as Example.Java? *22*
2.7.3 Explanation *22*
2.8 Java Improvements 23
2.8.1 Java 5.0 Features *23*
2.8.3 Java 6 Features *25*
2.8.4 Java 7 Features *26*
2.8.4 Brief Comparison of Different Releases *27*
2.9 Differences between Java and C++ 28
2.10 Installation of JDK 1.7 29
2.10.1 Getting Started With the JDK *29*
2.10.2 JDK Installation Notes *29*
2.10.3 Exploring the JDK *37*
2.11 Integrated Development Environment 39

**3. Java Programming Constructs 42**

3.1 Variables 42
3.2 Primitive Data Types 42
3.3 Identifier 44
3.3.1 Rules for Naming *44*
3.3.2 Naming Convention *44*
3.3.3 Keywords *45*
3.4 Literals 45
3.5 Operators 48
3.5.1 Binary Operators *48*
3.5.2 Unary Operators *54*
3.5.3 Ternary Operator *54*
3.6 Expressions 55
3.7 Precedence Rules and Associativity 55
3.8 Primitive Type Conversion and Casting 57
3.9 Flow of Control 61
3.9.1 Conditional Statements *62*
3.9.2 Loops *65*
3.9.3 Branching Mechanism *68*

**4. Classes and Objects 74**

4.1 Classes 74
4.2 Objects 75
4.2.1 Difference between Objects and Classes *76*
4.2.2 Why Should We Use Objects and Classes? *76*
4.3 Class Declaration in Java 77
4.3.1 Class Body *78*
4.4 Creating Objects 79
4.4.1 Declaring an Object *79*
4.4.2 Instantiating an Object *79*
4.4.3 Initializing an Object *80*
4.5 Methods 82
4.5.1 Why Use Methods? *82*
4.5.2 Method Types *82*
4.5.3 Method Declaration *83*
4.5.3 Instance Method Invocation *86*
4.5.4 Method Overloading *87*
4.6 Constructors 90
4.6.1 Parameterized Constructors *93*
4.6.2 Constructor Overloading *94*
4.7 Cleaning Up Unused Objects 96
4.7.1 The Garbage Collector *96*
4.7.2 Finalization *97*
4.7.3 Advantages and Disadvantages *97*
4.8 Class Variable and Methods—Static Keyword 97
4.8.1 Static Variables *98*
4.8.2 Static Methods *99*
4.8.3 Static Initialization Block *101*
4.9 `this` Keyword 103
4.10 Arrays 105
4.10.1 One-Dimensional Arrays *105*
4.10.2 Two-Dimensional Arrays *110*
4.10.3 Using `for-each` With Arrays *115*
4.10.4 Passing Arrays to Methods *115*
4.10.5 Returning Arrays from Methods *116*
4.10.6 Variable Arguments *117*
4.11 Command-line Arguments 118
4.12 Nested Classes 119
4.12.1 Inner Class *119*
4.12.2 Static Nested Class *122*
4.12.3 Why Do We Create Nested Classes? *124*
4.13 Practical Problem: Complex Number Program 124

**5. Inheritance 132**

5.1 Inheritance vs Aggregation 132
5.1.1 Types of Inheritance *133*
5.1.2 Deriving Classes Using Extends Keyword *135*
5.2 Overriding Method 137
5.3 `super` Keyword 141
5.4 `final` Keyword 146
5.5 Abstract Class 147
5.6 Shadowing vs Overriding 149
5.7 Practical Problem: `Circle` and `Cylinder` Class 151

**6. Interfaces, Packages, and Enumeration 156**

6.1 Interfaces 156
6.1.1 Variables in Interface *158*
6.1.2 Extending Interfaces *160*
6.1.3 Interface vs Abstract Classes *160*
6.2 Packages 161
6.2.1 Creating Packages *162*
6.2.2 Using Packages *164*
6.2.3 Access Protection *168*
6.3 `java.lang` Package 169
6.3.1 `java.lang.Object` Class *169*
6.3.2 Java Wrapper Classes *170*
6.3.3 String Class *174*
6.3.4 StringBuffer Class *179*
6.3.5 StringBuilder Class *180*
6.3.6 Splitting Strings *181*
6.4 Enum Type 183
6.4.1 Using Conditional Statements with an Enumerated Variable *185*
6.4.2 Using `for` Loop for Accessing Values *185*
6.4.3 Attributes and Methods Within Enumeration *186*
6.5 Practical Problem: Banking Example 187

**7. Exception, Assertions, and Logging 199**

7.1 Introduction 199
7.1.1 Exception Types *201*
7.2 Exception Handling Techniques 202
7.2.1 `try…catch` *203*
7.2.2 `throw` Keyword *206*

7.2.3 `throws` *207*
7.2.4 `finally` Block *209*
7.2.5 `try-with-resources` Statement *210*
7.2.6 Multi Catch *212*
7.2.7 Improved Exception Handling in Java 7 *213*
7.3 User-Defined Exception 215
7.4 Exception Encapsulation and Enrichment 216
7.5 Assertions 217
7.6 Logging 219

**8. Multithreading in Java 224**

8.1 Introduction 224
8.2 Multithreading in Java 225
8.3 `java.lang.Thread` 225
8.4 Main Thread 227
8.5 Creation of New Threads 228
8.5.1 By Inheriting the `Thread` Class *228*
8.5.2 Implementing the `Runnable` Interface *231*
8.6 Thread.State in Java 234
8.6.1 Thread States *235*
8.7 Thread Priority 240
8.8 Multithreading—Using `isAlive()` and `join()` 243
8.9 Synchronization 245
8.9.1 Synchronized Methods *246*
8.9.2 Synchronized Statements *246*
8.10 Suspending and Resuming Threads 246
8.11 Communication between Threads 248
8.12 Practical Problem: Time Clock Example 251

**9. Input/Output, Serialization and Cloning 256**

9.1 Introduction 256
9.1.1 `java.io.InputStream` and `java io.OutputStream` *257*
9.2 `java.io.File` Class 258
9.3 Reading and Writing Data 261
9.3.1 Reading/Writing Files Using Byte Stream *261*
9.3.2 Reading/Writing Console (User Input) *264*
9.3.3 Reading/Writing Files Using Character Stream *269*
9.3.4 Reading/Writing Using Buffered Byte Stream Classes *270*
9.3.5 Reading/Writing Using Buffered Character Stream Classes *272*
9.4 Randomly Accessing a File 273
9.5 Reading and Writing Files Using New I/O Package 276
9.6 Java 7 Nio Enhancements 278
9.7 Serialization 283
9.8 Cloning 285

**10. Generics, java.util and other API 296**

10.1 Introduction 296
10.2 Generics 301
10.2.1 Using Generics in Arguments and Return Types *304*
10.2.2 Wildcards *304*
10.2.3 Bounded Wildcards *306*
10.2.4 Defining Your Own Generic Classes *307*
10.3 Linked List 309
10.4 Set 311
10.4.1 Hashset Class *312*
10.4.2 Treeset Class *314*
10.5 Maps 315
10.5.1 Hashmap Class *315*
10.5.2 Treemap Class *317*
10.6 Collections Class 318
10.7 Legacy Classes and Interfaces 319
10.7.1 Difference between Vector and Arraylist *319*
10.7.2 Difference between Enumerations and Iterator *320*
10.8 Utility Classes: `Random` Class 320
10.8.1 Observer and Observable *322*
10.9 `Runtime` Class 326
10.10 Reflection API 328

**11. Network Programming 336**

11.1 Introduction 336
11.1.1 TCP/IP Protocol Suite *336*
11.2 Sockets 337
11.2.1 TCP Client and Server *338*
11.2.2 UDP Client and Server *342*
11.3 URL Class 344
11.4 Multithreaded Sockets 346
11.5 Network Interface 349

## 12. Applets 354

12.1 Introduction 354
12.2 Applet Class 355
12.3 Applet Structure 356
12.4 Example Applet Program 357
12.4.1 How to Run an Applet? *358*
12.5 Applet Life Cycle 359
12.6 Common Methods Used in Displaying the Output 361
12.7 `paint()`, `update()`, and `repaint()` 364
12.7.1 `paint()` Method *364*
12.7.2 `update()` Method *365*
12.7.3 `repaint()` Method *366*
12.8 More About Applet Tag 366
12.9 `getDocumentbase()` and `getCodebase()` Methods 369
12.10 Appletcontext Interface 370
12.10.1 Communication between Two Applets *371*
12.11 How To Use An Audio Clip? 372
12.12 Images in Applet 373
12.12.1 Mediatracker Class *375*
12.13 Graphics Class 377
12.13.1 An Example Applet Using Graphics *379*
12.14 Color 380
12.15 Font 382
12.16 Fontmetrics 386
12.17 Practical Problem: Digital Clock 390

## 13. Event Handling in Java 394

13.1 Introduction 394
13.2 Event Delegation Model 395
13.3 `java.awt.Event` Description 395
13.3.1 Event Classes *395*
13.4 Sources of Events 404
13.5 Event Listeners 404
13.6 How Does The Model Work? 406
13.7 Adapter Classes 410
13.7.1 How To Use Adapter Classes *410*
13.7.2 Adapter Classes in Java *412*
13.8 Inner Classes in Event Handling 413
13.9 Practical Problem: Cartoon Applet 416
13.9.1 Smiling Cartoon With Blinking Eyes (Part 1) *416*
13.9.2 Smiling Cartoon With Blinking Eyes (Part 2) *420*
13.9.3 Smiling Cartoon (Part 3) *423*

## 14. Abstract Window Toolkit 429

14.1 Introduction 429
14.1.1 Why Awt? *429*
14.1.2 `java.awt` Package *430*
14.2 Components and Containers 432
14.2.1 Component *432*
14.2.2 Components as Event Generator *433*
14.3 Button 434
14.4 Label 437
14.5 Checkbox 438
14.6 Radio Buttons 441
14.7 List Boxes 444
14.8 Choice Boxes 448
14.9 Textfield and Textarea 451
14.10 Container Class 455
14.10.1 Panels *455*
14.10.2 Window *456*
14.10.3 Frame *456*
14.11 Layouts 458
14.11.1 `FlowLayout` *459*
14.11.2 `BorderLayout` *462*
14.11.3 `CardLayout` *465*
14.11.4 `GridLayout` *469*
14.11.5 `GridbagLayout` *471*
14.12 Menu 478
14.13 Scrollbar 483
14.14 Practical Problem: City Map Applet 487

## 15. Swing 495

15.1 Introduction 495
15.1.1 Features of Swing *496*
15.1.2 Differences between Swing and AWT *496*
15.2 JFrame 497
15.3 JApplet 500
15.4 JPanel 501
15.5 Components in Swings 502
15.6 Layout Managers 506
15.6.1 Springlayout *506*
15.6.2 Boxlayout *509*
15.7 JList and JScrollPane 510
15.8 Split Pane 513
15.9 JTabbedPane 514
15.10 JTree 516
15.11 JTable 521
15.12 Dialog Box 525
15.13 JFileChooser 529

15.14 JColorChooser 530
15.15 Pluggable Look and Feel 531
15.16 Inner Frames 539
15.17 Practical Problem: Mini Editor 545

**16. Introduction to Advanced Java 553**

16.1 Introduction to J2EE 553
16.2 Database Handling Using JDBC 553
16.2.1 Load the Driver *554*
16.2.2 Establish Connection *556*
16.2.3 Create Statement *556*
16.2.4 Execute Query *557*
16.2.5 Iterate Resultset *557*
16.2.6 Scrollable Resultset *559*
16.2.7 Transactions *560*
16.3 Servlets 562
16.3.1 Lifecycle of Servlets *562*
16.3.2 First Servlet *563*
16.3.3 Reading Client Data *567*
16.3.4 Http Redirects *571*
16.3.5 Cookies *572*
16.3.6 Session Management *574*
16.4 Practical Problem: Login Application 577
16.5 Introduction to Java Server Pages 589
16.5.1 JSP Life Cycle *589*
16.5.2 Steps in JSP Page Execution *590*
16.5.3 JSP Elements *590*
16.5.4 Placing Your JSP in the Webserver *593*
16.6 Java Beans 597
16.6.1 Properties of a Bean *597*
16.6.2 Using Beans Through JSP *601*
16.6.3 Calculatebean Example *602*
16.7 Jar Files 605
16.7.1 Creating a Jar File *605*
16.7.2 Viewing the Contents of a Jar File *606*
16.7.3 Extracting the Contents of Jar *607*
16.7.4 Manifest Files *607*
16.8 Remote Method Invocation 609
16.8.1 RMI Networking Model *609*
16.8.2 Creating an Rmi Application *610*
16.9 Introduction to EJB 613
16.9.1 Types of EJB *614*
16.9.2 EJB Architecture *615*
16.10 Hello World—EJB Example 616

*Appendix A:* `this` *Reference Demystified* *628*
*Appendix B: Stacks versus Heaps* *629*
*Appendix C: Pointer versus Reference Variables* *631*
*Appendix D: Regular Expressions* *634*
*Appendix E: Interfaces in Java 8* *646*
*Appendix F: Functional Programming with Lambdas* *651*
*Appendix G: Interview Questions* *668*
*Index* *675*

# Introduction to OOP

# 1

*Beauty is our weapon against nature; by it we make objects, giving them limit, symmetry, proportion. Beauty halts and freezes the melting flux of nature.*

**Camille Paglia**

**After reading this chapter, the readers will be able to**

- know what is object-oriented programming
- understand the principles of OOP
- understand how is OOP different from procedural languages
- comprehend the problems in procedural programming and how OOP overcomes them
- learn the applications of OOP
- use UML notations



## 1.1 INTRODUCTION

Object-oriented programming (OOP) is one of the most interesting and useful innovations in software development. OOP has strong historical roots in programming paradigms and practices. It addresses the problems commonly known as the *software crisis*. Software have become inherently complex which has led to many problems within the development of large software projects. Many software have failed in the past. The term 'software crisis' describes software failure in terms of

- Exceeding software budget
- Software not meeting clients' requirements
- Bugs in the software

OOP is a programming paradigm which deals with the concept of objects to build programs and software applications. It is modeled around the real world. The world we live in is full of objects. Every object has a well-defined *identity*, *attributes*, and *behavior*. Objects exhibit the same behavior in programming. The features of object-oriented programming also map closely to the real-world features like *inheritance*, *abstraction*, *encapsulation*, and *polymorphism*. We will discuss them later in the chapter.

## 1.2 NEED OF OBJECT-ORIENTED PROGRAMMING

There were certain limitations in earlier programming approaches and to overcome these limitations, a new programming approach was required. We first need to know what these limitations were.

### 1.2.1 Procedural Languages

In procedural languages, such as C, FORTRAN, and PASCAL, a program is a list of instructions. The programmer creates a list of instructions to write a very small program. As the length of a program increases, its complexity increases making it difficult to maintain a very large program. In the structured programming, this problem can be overcome by dividing a large program into different functions or modules, but this gives birth to other problems. Large programs can still become increasingly complex. There are two main problems in procedural languages—(i) the functions have unrestricted access to global data and (ii) they provide poor mapping to the real world.

Here are some other problems in the procedural languages. Computer languages generally have built-in data types: integers, character, float, and so on. It is very difficult to create a new data type or a user-defined data type. For example, if we want to work with dates or complex numbers, then it becomes very difficult to work with built-in types. Creating our own data types is a feature called *extensibility*: we can extend the capabilities of a language. Procedural languages are not extensible. In the traditional languages, it is hard to write and maintain complex results.

### 1.2.2 Object-Oriented Modeling

In the physical world, we deal with objects like person, plane, or car. Such objects are not like data and functions. In the complex real-world situations, we have objects which have some attributes and behavior. We deal with similar objects in OOP. Objects are defined by their unique *identity*, *state*, and *behavior*. The state of an object is identified by the value of its attributes and behavior by methods.

#### Attributes

Attributes define the data for an object. Every object has some attributes. Different types of objects contain different attributes or characteristics. For example, the attributes of a student object are name, roll number, and subject; and the attributes for a car object would be color, engine power, number of seats, etc. These attributes will have specific values, such as Peter (for name) or 23 (for roll number).

#### Behavior

The response of an object when subjected to stimulation is called its *behavior*. Behavior defines what can be done with the objects and may manipulate the attributes of an object. For example, if a manager orders an employee to do some task, then he responds either by doing it or not doing it. The wings of a fan start moving only when the fan is switched ON. Behavior actually determines the way an object interacts with other objects. We can say that behavior is synonym to functions or methods: we call a function to perform some task. For example, an Employee class will have functions such as adding an employee, updating an employee details, etc.

**Note** If we try to represent the CPU of a computer in OOP terminology, then CPU is the object. The CPU is responsible for fetching the instructions and executing them. So fetching and executing are two possible functions (methods or behavior) of CPU. The place (attributes) where CPU stores the retrieved instructions, values and result of the execution (registers) will then be the attributes of the CPU.

## 1.3 PRINCIPLES OF OBJECT-ORIENTED LANGUAGES

OOP languages follow certain principles such as class, object, and abstraction. These principles map very closely to the real world.

### 1.3.1 Classes

A class is defined as the blueprint for an object. It serves as a plan or a template. The description of a number of similar objects is also called a *class*. An object is not created by just defining a class. It has to be created explicitly. Classes are logical in nature. For example, furniture does not have any existence but tables and chairs do exist. A class is also defined as a new data type, a user-defined type which contains two things: data members and methods.

### 1.3.2 Objects

Objects are defined as the instances of a class, e.g. table, chair are all instances of the class Furniture. Objects of a class will have same attributes and behavior which are defined in that class. The only difference between objects would be the value of attributes, which may vary. Objects (in real life as well as programming) can be physical, conceptual, or software. Objects have unique identity, state, and behavior. There may be several types of objects:

- *Creator objects*: Humans, Employees, Students, Animal
- *Physical objects*: Car, Bus, Plane
- *Objects in computer system*: Monitor, Keyboard, Mouse, CPU, Memory

### 1.3.3 Abstraction

Can you classify the following items?

- Elephant
- Television
- Table
- CD player
- Chair
- Tiger

How many classes do you identify here? The obvious answer anybody would give is three, i.e., Animal, Furniture, and Electronic items. But how do you come to this conclusion? Well, we grouped similar items like Elephant and Tiger and focused on the generic characteristics rather than specific characteristics. This is called *abstraction*. Everything in this world can be classified as living or non-living and that would be the highest level of abstraction.

Another well-known analogy for abstraction is a car. We drive cars without knowing the internal details about how the engine works and how the car stops on applying brakes. We are happy with the abstraction provided to us, e.g., brakes, steering, etc. and we interact with them. In real life, human beings manage complexity by abstracting details away. In programming, we manage complexity by concentrating only on the essential characteristics and suppressing implementation details.

### 1.3.4 Inheritance

Inheritance is the way to adopt the characteristics of one class into another class. Here we have two types of classes: *base class* and *subclass*. There exists a parent–child relationship among the classes. When a class inherits another class, it has all the properties of the base class and it adds some new properties of its own. We can categorize vehicles into car, bus, scooter, ships, planes, etc. The class of animals can be divided into mammals, amphibians, birds, and so on.

The principle of dividing a class into subclass is that each subclass shares common characteristics with the class from where they are inherited or derived. Cars, scooters, planes, and ships all have an engine and a speedometer. These are the characteristics of vehicles. Each subclass has its own characteristic feature, e.g., motorcycles have disk braking system, while planes have hydraulic braking system. A car can run only on the surface, while a plane can fly in air and a ship sails over water (see Fig. 1.1).

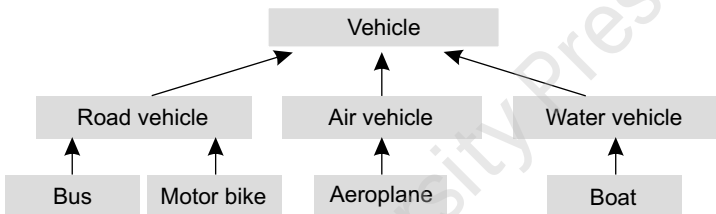


**Fig. 1.1** Inheritance

Inheritance aids in *reusability*. When we create a class, it can be distributed to other programmers which they can use in their programs. This is called *reusability*. Suppose someone wants to make a program for a calculator, he can use a predefined class for arithmetic operations, and then he need not define all the methods for these operations. This is similar to using library functions in procedural language. In OOP, this can be done using the inheritance feature. A programmer can use a base class with or without modifying it. He can derive a child class from a parent class and then add some additional features to his class.

### 1.3.5 Encapsulation

Encapsulation is one of the features of object-oriented methodology. The process of binding the data procedures into objects to hide them from the outside world is called *encapsulation* (see Fig. 1.2). It provides us the power to restrict anyone from directly altering the data. Encapsulation is also known as *data hiding*. An access to the data has to be through the methods of the class. The data is hidden from the outside world and as a result, it is protected. The details that are not useful for other objects should be hidden from them. This is called *encapsulation*. For example, an object that does the calculation must provide an interface to obtain the result. However, the internal coding used to calculate need not be made available to the requesting object.

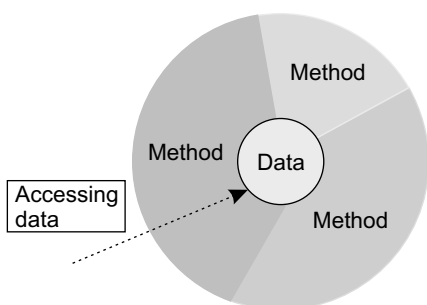


**Fig. 1.2** Diagrammatic Illustration of a Class to Show Encapsulation

### 1.3.6 Polymorphism

Polymorphism simply means many forms. It can be defined as the same thing being used in different forms. For example, there are certain bacteria that exhibit in more than one morphological form. In programming, polymorphism is of two types: *compile-time* and *runtime polymorphism*. Runtime polymorphism, also known as *dynamic binding* or *late binding*, is used to determine which method to invoke at runtime. The binding of method call to its method is done at runtime and hence the term *late binding* is used. In case of compile-time polymorphism, the compiler determines which method (from all the overloaded methods) will be executed. The binding of method call to the method is done at compile time. So the decision is made early and hence the term *early binding*. Compile-time polymorphism in Java is implemented by *overloading* and runtime polymorphism by *overriding*. In overloading, a method has the same name with different signatures. (A *signature* is the list of formal arguments that is passed to the method.) In overriding, a method is defined in subclass with the same name and same signature as that of parent class. This distinction between compile-time and runtime polymorphism is of *method invocation*. Compile-time polymorphism is also implemented by operator overloading which is a feature present in C++ but not in Java. Operator overloading allows the user to define new meanings for that operator so that it can be used in different ways. The operator (+) in Java is however an exception as it can be used for addition of two integers as well as concatenation of two strings or an integer with a string. This operator is overloaded by the language itself and the Java programmer cannot overload any operator.

## 1.4 PROCEDURAL LANGUAGE VS OOP

Table 1.1 highlights some of the major differences between procedural and object-oriented programming languages.

**Table 1.1** Procedural Languages vs OOP

| Procedural Languages | OOP |
|---|---|
| • Separate data from functions that operate on them. | • Encapsulate data and methods in a class. |
| • Not suitable for defining abstract types. | • Suitable for defining abstract types. |
| • Debugging is difficult. | • Debugging is easier. |
| • Difficult to implement change. | • Easier to manage and implement change. |
| • Not suitable for larger programs and applications. | • Suitable for larger programs and applications. |
| • Analysis and design not so easy. | • Analysis and design made easier. |
| • Faster. | • Slower. |
| • Less flexible. | • Highly flexible. |
| • Data and procedure based. | • Object oriented. |
| • Less reusable. | • More reusable. |
| • Only data and procedures are there. | • Inheritance, encapsulation, and polymorphism are the key features. |
| • Use top-down approach. | • Use bottom-up approach. |
| • Only a function calls another. | • Object communication is there. |
| • Example: C, Basic, FORTRAN. | • Example: JAVA, C++, VB.NET, C#.NET. |

## 1.5 OOAD USING UML

An object-oriented system comprises of objects. The behavior of a system results from its objects and their interactions. Interaction between objects involves sending messages to each other. Every object is capable of receiving messages, processing them, and sending to other objects.

### Object-oriented Analysis and Design (OOAD)

It is an approach that models software as a group of interacting objects. A model is a description of the system that we intend to build. Each object is characterized by its class having its own state (attributes) and behavior. Object-oriented analysis (OOA) analyzes the functional requirements of a system and focuses on *what* the system should do. Object-oriented design (OOD) focuses on *how* the system does it. The most popular modeling language for OOAD is the *unified modeling language* (UML).

UML is a standard language for OOAD. It contains graphical notations for all entities (class, object, etc.) used in the object-oriented languages along with the relationship that exists among them. These notations are used to create models. UML helps in visualizing the system, thereby reducing complexity and improving software quality. The notations used for class and object are shown in Fig. 1.3. For example, consider an Employee class with attributes name, designation, salary, etc. and operations such as addEmployee, deleteEmployee, and searchEmployee.

The notation for employee class and its object is as follows:

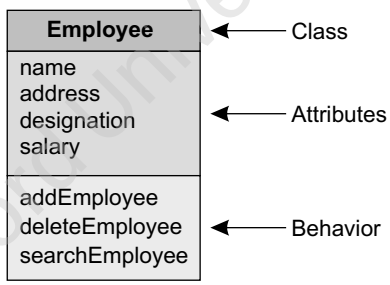


**Fig. 1.3** UML Notation for Class

The notation for an object is very much similar to the class notation. The class name underlined and followed by a colon represents an object (Fig. 1.4).

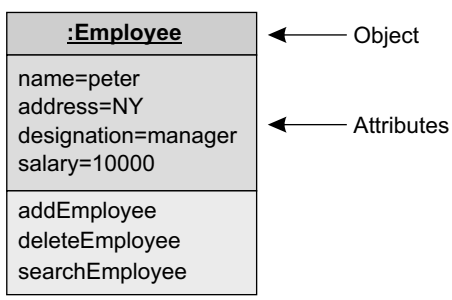


**Fig. 1.4** UML Notation for Object

An instance of a class can be related to any number of instances of other class known as multiplicity of the relation. One-to-one, one-to-many, and many-to-many are different types of multiplicities that exist among objects. The multiplicities along with their examples and respective notations are shown below. Figure 1.5(a) illustrates the generic notation for representing multiplicity in object-oriented analysis and design. One-to-one mapping is shown as a straight line between the two classes. Figure 1.5(b) shows the UML notation for demonstrating the one-to-one mapping. The 1..1 multiplicity depicted on the straight line (both ends) indicates a single instance of a class is associated with single instance of other class. Figure 1.5 shows that each country has a president and a president is associated with a country.

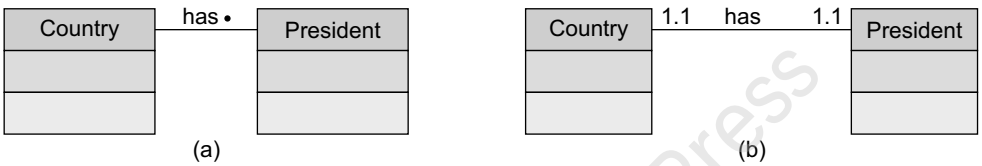


**Fig. 1.5** One-to-one Relationship

A country has many states and many states belong to a country. So there exists a one-to-many relationship between the two. This relationship is shown in Fig. 1.6. Part (a) of this figure shows the generic notation where a solid dot is indicated on the many side and both classes are joined by a straight line. Figure 1.6(b) shows the UML notation where 1..* indicates the one to many relationship between country and states. On the country end, a 1..1 multiplicity is placed to indicate one country and on states end, a 1..* is placed to indicate many states.

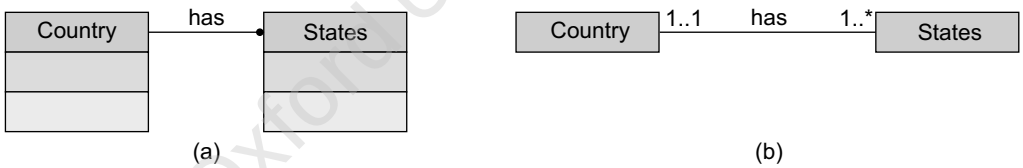


**Fig. 1.6** One-to-many Relationship

Let us take another example to explain many-to-many relationship. A teacher teaches many students and a student can be taught by many teachers. There exists a many-to-many relationship between them. Many-to-many relationship (Generic notation in OOAD) are represented by placing solid dots on both ends joined by a straight line as shown in Fig. 1.7(a). The respective notation in UML is shown in Fig. 1.7(b) where 1..* on both ends is used to signify many-to-many relationship.

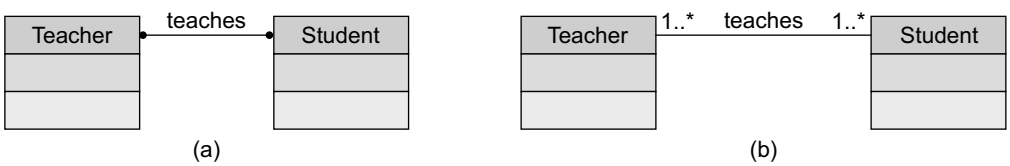


**Fig. 1.7** Many-to-many Relationship

Besides multiplicity of relations, the relationships can be of various types: *inheritance, aggregation, composition*. These relationships can be denoted in UML with links and associations. The links represent the connection between the objects and associations represent groups of links between classes. If a class inherits another class, then there exists a parent-child relationship between them. This relationship is depicted in UML as shown in Fig. 1.8. For example, Shape is the superclass, and the subclasses of Shape can take any shape, e.g., Square, Triangle, etc.

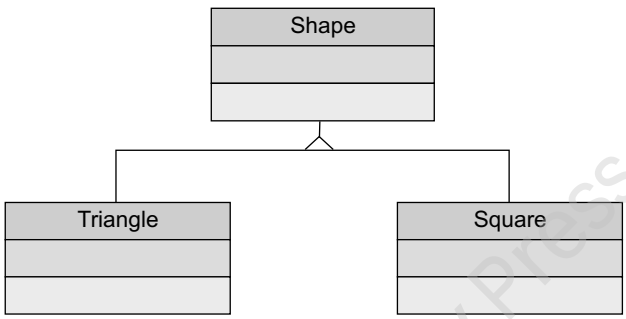


**Fig. 1.8** UML Diagram Depicting Inheritance

The above diagram can be extended to depict the OOP principle of polymorphism. Every shape will have a method named area() which would calculate the area of that shape. The implementation of area() method would be different for different shapes. For example, the formula for calculating area of a triangle is different from a square. So the implementation is different but the name of the method is same. This is polymorphism (one name many implementations). In Fig 1.9 below, the area() method is overridden by Triangle and Square classes.

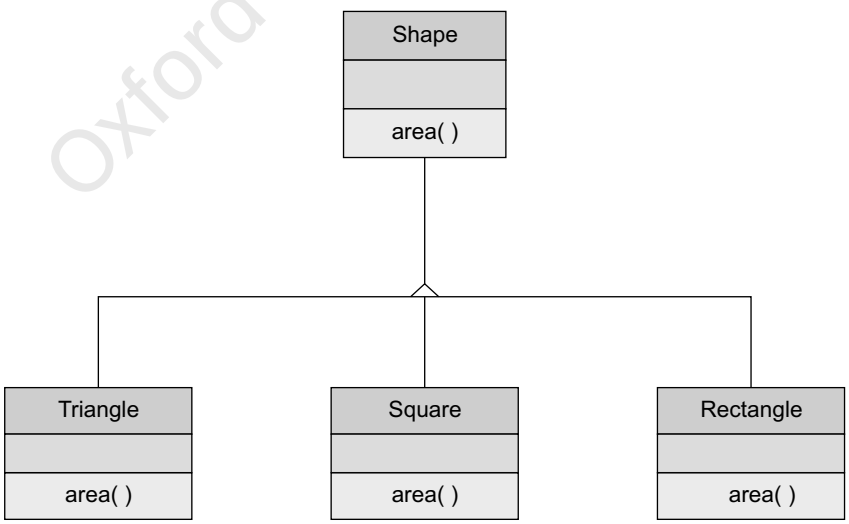


**Fig. 1.9** UML Diagram Depicting Polymorphism

Another kind of relationship that exists among objects is the part-of-relationship. When a particular object is a part of another object then we say that it is *aggregation*. For example,

car is an aggregation of many objects: engine, door, etc. and engine in turn is an aggregation of many objects, e.g., cylinder, piston, valves, etc. as shown in Fig. 1.10(a). A special kind of aggregation is composition where one object owns other objects. If the owner object does not exist, the owned objects also cease to exist. For example, the human body is a very good example of composition. It is a composition of different organs. The hands, feet, and internal organs such as the lung and intestine are also parts of the body owned by the body.

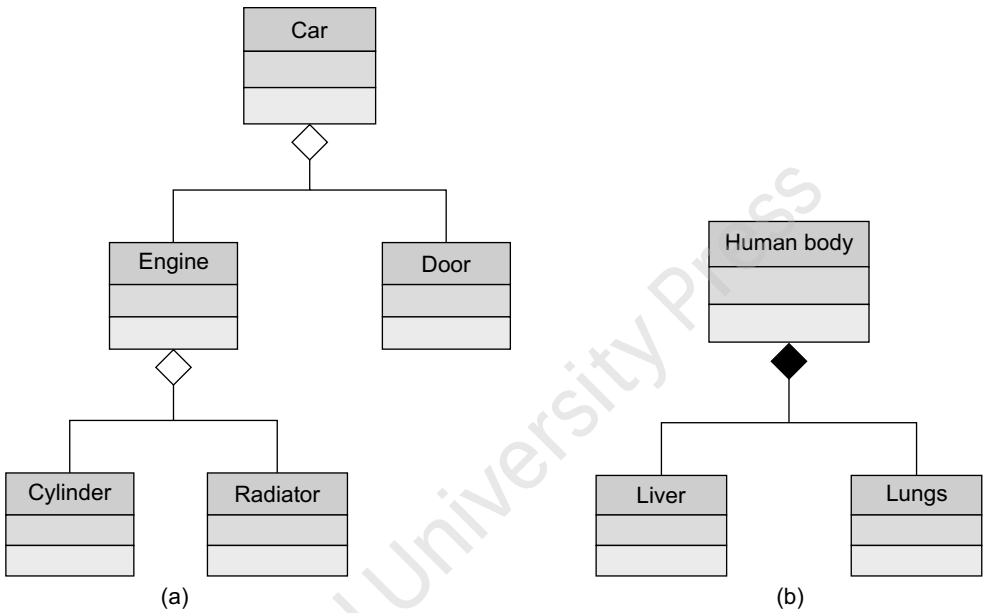


**Fig. 1.10** (a) Aggregation and (b) Composition

## 1.6 APPLICATIONS OF OOP

The basic thought behind object-oriented language is to make an object by combining data and functions as a single unit and then operate on that data. In procedural approach, the focus is on business process and the data needed to support the process. For example, in the late 1990s, a problem bothered every programmer, popularly known as the Y2K problem. Everybody related to the computer industry was afraid of what will happen past midnight 31 December 1999. The problem arises due to the writing convention of the year attribute. In early programming days, a programmer wrote a year in two digits, so there was a problem to distinguish the year 1900 from 2000 because if we write only the last two digits of a year, the computer cannot differentiate between the two. Nobody perceived this problem and used the date and year code as and when required, thus aggravating the problem. The solution to this problem was to analyze multiple lines of codes everywhere and change the year to four digits rather than two. It seems simple to change the state variable of year but analyzing a code of several thousands of lines to find how many times you have used date in your code is not an easy task.

If object-oriented programming language had been used, we could have created a Date class with day, month, and year attributes in it. Wherever the date functionality would be required,

a Date object would be created and used. At a later point of time, if a change is required, for example, the year of Date class needs to be changed to four digits, then this change would be incorporated in the class only and this change would automatically be reflected in all the objects of the Date class whenever they are created and used. So, the change would have to be done at one place only, i.e., the class and wherever the objects of the class are being used, the changes would be reflected automatically. There is no need to analyze the whole code and change it.

In OOP, we access data with the help of objects, so it is very easy to overcome a problem without modifying the whole system. Likewise, OOP is used in various fields, such as

- Real-time systems
- Artificial intelligence
- Expert systems
- Neural networks
- Database management

## SUMMARY

Object-oriented languages have become an ubiquitous standard for programming. They have been derived from the real world. OOP revolves around objects and classes. A class is defined as a group of objects with similar attributes and behavior. OOP is a programming paradigm which deals with the concepts of objects to develop software applications. Certain principles have been laid down by OOP which are followed by every OOP language. These principles are: inheritance, abstraction, encapsulation, and polymorphism.

We have presented a detailed comparison of procedural and object-oriented languages. For building large projects, a technique known as OOAD is used. Object-oriented analysis and design deals with how a system is modeled. OOA deals with what the system should do and OOD deals with how the system achieves what has been specified by OOA.

OOAD is realized with the help of a language known as UML. UML stands for unified modeling language; it is a standard language used for visualizing the software. An abstract model is created for the entire software using graphical notations provided by UML.

## EXERCISES

### Objective Questions

1. In an object model, which one of the following is true?
   (a) Abstraction, encapsulation, and multitasking are the major principles
   (b) Hierarchy, concurrency, and typing are the major principles
   (c) Abstraction, encapsulation, and polymorphism are the major principles
   (d) Typing is the major principle
2. Which one of the following is not an object-oriented language?
   (a) Simula (b) Java
   (c) C++ (d) C
3. The ability to hide many different implementations behind an interface is
   (a) Abstraction (b) Inheritance
   (c) Polymorphism (d) None of the above
4. Which one of the following terms must relate to polymorphism?
   (a) Static allocation (b) Static typing
   (c) Dynamic binding (d) Dynamic allocation
5. Providing access to an object only through its member functions, while keeping the details private is called
   (a) Information hiding (b) Encapsulation
   (c) Modularity (d) Inheritance
6. The concept of derived classes is involved in
   (a) Inheritance
   (b) Encapsulation
   (c) Data hiding
   (d) Abstract data types

7. Inheritance is a way to
   (a) Organize data
   (b) Pass arguments to objects of classes
   (c) Add features to existing classes without rewriting them
   (d) Improve data-hiding and encapsulation
8. UML is used for
   (a) Creating models
   (b) Representing classes, objects and their relationships pictorially
   (c) Reducing complexity and improving software quality
   (d) All the above
9. Which of the following is true about class?
   (a) Class possesses data and methods
   (b) Classes are physical in nature
   (c) Collection of similar type of objects is a class
   (d) Both (a) and (c)
10. Which of the following is true about procedural languages?
   (a) Debugging is easier
   (b) Analysis and design is easy
   (c) Less reusable
   (d) Difficult to implement changes

## Review Questions

1. Explain the importance of object-oriented programming languages.
2. Explain the difference between class and object.
3. Differentiate between procedural languages and OOP languages.
4. Write short notes on: (a) inheritance, (b) polymorphism, (c) abstraction, (d) encapsulation.
5. Differentiate between runtime and compite-time polymorphism.

## Programming Exercises

1. Identify the relevant classes along with their attributes for the following: A departmental store needs to maintain an inventory of cosmetic items which might be found there. You should include female as well as male cosmetic items. Keep information on all items such as item name, category, manufacturer, cost, date purchased, and serial number.
2. Identify the relevant classes along with their attributes from the following problem specification:

   A hospital wants to keep track of scheduled appointments of a patient with his doctor. When a patient is given an appointment, he should be given a confirmation that states the time and date of appointment along with the doctor's name. Meanwhile the doctor should also be informed about the patient details. Each doctor has one weekday as off-day and no patients should be assigned to a doctor on that day.



## Answers to Objective Questions

| | | | |
|---|---|---|---|
| **1.** (c) | **2.** (d) | **3.** (c) | **4.** (c) |
| **5.** (b) | **6.** (a) | **7.** (c) | **8.** (d) |
| **9.** (d) | **10.** (c) and (d) | | |