PEARSON

The class of JAVA

Pravin M. Jain

The class of JAVA

Pravin M. Jain

Director-Technical
Zen Softech

PEARSON

Delhi • Chennai • Chandigarh

Contents

Foreword

It is my pleasure to write the foreword for the first edition of the book entitled "The `class` of JAVA" by Pravin M. Jain. I have known Pravin M. Jain since long. I am visualizing the days, way back in 1983, when he was one of my students of T.Y. B.Sc. class at M.S. University, Baroda. It was then itself that I had identified him as one of the brilliant students.

By now, Pravin has established himself as a well-versed, experienced, excellent corporate trainer on Java. He imparts training to experienced developers. He has his own conceptual, elegant and lucid style of teaching. Thus, I have invited him several times as a resource person to teach Java programming to teachers participating in refresher courses or who deliver expert lecture series to students of the department. This book is the outcome of our desire to make available a good, proper, easy-to-understand text book on core Java to students all over India.

The Java Programming language has come a long way from the first release in 1995. Now, it is no more about just applets. Whole hosts of Java technologies are available for everything: be it Web applications, desktop applications, mobile applications or frameworks for database access, unit-testing or report generation. Hence, Java has become one of the most widely used programming languages today, and this book provides an in-depth understanding of the basics of Java.

Pravin has covered all the topics needed for an introductory course on Java. What is noteworthy about the book is that it not only teaches programming but also shows good programming practices, a quality cherished and looked for by qualitative employers.

While going through the book, I found that various constructs of the Java programming language have been introduced in a systematic, step-by-step incremental manner in Chapters 5 to 11. These chapters are truly a reflection of the title of the book, which explains the details of every part relating to a class of Java. The chapters on Exception and Threads also discuss the details of execution of a thread in Java. The exercise in Chapter 5 starts with a simple example of creating an `Account` class, and has been systematically modified in the subsequent chapters to incorporate newly introduced topics, and finally a multi-threaded network-based server is created. These exercise sequence is useful in learning to apply the newly introduced topics.

If you are a student keen on learning the art of Object-oriented Programming using Java or a teacher looking for a good text book to teach the Java Programming Language course, I think you need this book!

Dr Savita Gandhi
Professor and Head
Department of Computer Science
Gujarat University

Preface

The Title

The title of the book has three meanings. The `class` of JAVA is a book that was written from the notes of the various lectures on Java given by the author in a class room. The `class` of JAVA also means that this book talks about the meaning of the `class` in the Java Programming Language. The `class` of JAVA tries to highlight the class of Java as compared with the other programming languages.

The Inspiration

This book has been written because of inadequate books being used in the various educational institutions within India.

Who Should Read This Book

This book is meant for students, who wish to learn Object-oriented Programming using Java. The book assumes that the student is familiar with some programming using the C language. Students would be able to learn the art of Object-oriented Programming. This book can be used by students who do not have prior knowledge of Object-oriented Programming Language.

Conventions Used in This Book

The following conventions are used in this book:

- All Java keywords are shown as `keyword`, e.g. `class`
- All inline codes are shown as `in line code.`
- All individual class names are shown as `ClassName`
- All method names are shown as `methodname()`
- All individual code pieces are shown as `code pieces`

Structure of the Book

The goal of this book is to serve as a good text for a course on Core Java for university students and teachers, which can be adopted as a text book for the Java Programming Language

course. Another important goal of this book is to provide a document that deals with the basics of the Java Programming Language in an easy-to-understand manner, while not leaving out the depths and correctness in understanding. The book has been organized in twenty-three chapters.

Chapter 1 provides a brief introduction to the history of Java, a brief idea about the meanings of platform independence, the Object-oriented programming concepts like data-encapsulation, inheritance and polymorphism. It also hints at programmming with types.

Chapter 2 gives an introduction to developing applications using the Java programming language. It uses the standard `Hello world` application as an example. It also talks about the common utilities that are part of the JDK.

Chapter 3 provides details of the various data types available in Java. It explains the meaning of the reference data type and also explains how arrays in Java differ from the arrays in C language.

Chapter 4 provides details of the various operators and commonly used statements.

Chapter 5 gives a step-by-step introduction to the most common members, which can be defined in a class. It explains the meaning of the members of a class like the instance variable, class variables, methods, static methods, constructors and initializer blocks. It also explains the concept of method overloading.

Chapter 6 deals with the topic of inheritance of classes. The various aspects are related to using inheritance and also the concept of method overriding.

Chapter 7 explains the concept of abstract data types. It explains about the abstract classes and methods and also about the concept of multiple inheritance and the interfaces available in Java.

Chapter 8 explains the purpose of the various methods in the Object class, which is the super-class of all the classes in Java.

Chapter 9 explains the packages in Java. It explains the use of the package and the import statement and explains how import is used by the Java compiler. It also explains about how the CLASSPATH environment should be set and used. It also explains the meanings of the various access specifiers that are available in Java.

Chapter 10 shows the usage of some of the common classes from the `java.lang` package.

Chapter 11 explains the use of the `throw` statement and the exception handling-related keywords, `try and catch` and finally in Java. It also explains the difference between the checked and unchecked exceptions and also explains the use of the `throws` keyword.

Chapter 12 explains the various forms of nested classes and interfaces and how such nested classes and interfaces can be defined and their usage.

Chapter 13 shows the usage of some of the commonly used classes from the `java.util` package.

Chapter 14 shows the usage of the `Stream` classes from the `java.io` package for the purpose of input and output. It also explains the use of `File` class for the purpose of file management.

Chapter 15 explains some networking concepts and the usage of some classes from the `java.net` package.

Chapter 16 explains the concept of multi-theading. It explains how to create applications with multiple threads.

Chapters 17 to 19 show how to build GUI applications in the Java programming language and the various GUI-related APIs. Event handling is explained in Chapter 19.

Chapter 20 show the usage of the models in the `JTable` and `JTree` swing components.

Chapter 21 shows the details of the `Applet` class and how `Applet` works within the browser.

Chapter 22 shows the usage of the JDBC APIs and how to connect to a database from a Java application and how to execute SQL on any database.

Chapter 23 explains how to define annotations and how to annotate programming elements in a Java source code.

Acknowledgements

The production of any book involves valid contributions from a number of persons. I would like to thank my publishers Pearson Education for their encouragement, insight and strong support. I thank Pradeep Banerjee and Jennifer Sargunar for the editing and production of this project.

I am indebted to Prof. Dr Savita Gandhi, Head of Department, Department of Computer Science, Gujarat University, for writing the foreword for this book and also for her keen interest in this project, right from its inception.

I would like to thank, Prof. Dr S. Rama Mohan, Department of Applied Mathematics, Maharaja Sayajirao University of Baroda, who introduced me to the world of Java technologies, right back in the year 1998 since when I have been constantly working in the technology, teaching and consulting services to corporates.

I would like to thank my beloved teacher and friend, Shri Arvind Vaish of ORG Systems, who during the foundation years of my school and college, provided me with the vision and the direction and made me whatever I am today. He helped me with my academic subjects and was instrumental in developing my logical and analytical abilities. Thank you sir, for your guidance and I am extremely happy with my career today.

I am thankful to Shri Hitesh Dalal and Shri Devang Dalal of Info Education Services, Baroda, for providing me with the first teaching platform in Java. Before this teaching assignment, I had been involved mainly in the development of softwares, and it was this assignment that proved my excellence at teaching young university students. Till today this has been my main involvement.

I would further like to acknowledge the contributions of Dr A. K. Aggarwal, Director, School of Computer Science, Windsor University, Canada and Prof. Amit Johri of IEEE Inc., USA, who under the aegis of IEEE Computer Society, Gujarat, formed the Java User Group, Baroda, which was convened by me with the presence of almost all the industries as members of Baroda.

I also remember my days and work experience at CMC Limited, which provided me with an excellent experience of technology issues in the corporate sector, and I would like to thank all my seniors and colleagues at CMC. My special thanks are due to Shri A. V. Veerkar, Shri Vinayak Pandit and Late Shri Dinesh Gupta.

I would like to thank Sreejit Purushothaman of Zen Softech and Saket Jain of Credit Suisse, who right from the beginning took personal interest in this project and helped me in all aspects of the book.

I would also like to thank Prof. Dr V. A. Kalamkar, Head of Department, Department of Computer Applications, Faculty of Science, Maharaja Sayajirao University of Baroda, who gave valuable suggestions regarding the sequencing of chapters in the book.

The final tribute and appreciation, however, is reserved for my parents, Shri Madan Kumar Jain and Shrimati Meena Kumari Jain, and sister Dr Medha Jain, who during the rough times reminded me that there is likely to be a light at the end of the tunnel. Their inspiration and love have been instrumental in providing me with the necessary support while writing this book.

I sincerely hope that this book The `class` of JAVA will meet the needs of all my valued readers.

Pravin Jain

About the Author

Pravin Jain obtained his Master of Computer Applications degree from Maharaja Sayajirao University of Baroda having had an excellent academic career in 1988. He worked with CMC Limited as Senior Software Engineer for nine years. During his tenure at CMC Limited, he had an opportunity to work on various prestigious projects including the DIDS project for the Bombay Stock Exchange and the SCADA project for ONGC.

Pravin opted for an enterpreneural career starting his own software development company in 1995, to which he added the training division in 1999. He was the first Sun Certified Java Programmer in Baroda having cleared SCJP in September 1999. He was actively associated with CMC Limited in a project of Indian Space Research Organization for the CARTOSAT in 2003.

He is a life member at the Computer Society of India. He is a visiting professor, a project guide and an external examiner for MCA, BE Computer Science and Engineering, PGDCA and BCA of Maharaja Sayajirao University of Baroda. He provides expert lectures on Java technologies at Rollwala Computer Center, Gujarat University, Charusat University, Changa and SVIT, Vasad.

Today, Pravin devotes his professional time towards conducting training in Java programming, working on corporate projects and visiting universities.

Pravin is a member of the board of studies at Gujarat University, where he is involved in the development of course syllabi for various subjects with his academic and corporate experience. He knows the needs of the corporates and has been able to frame the syllabi accordingly.

Since 1995, he is the Director-Technical of Zen Softech, which he set up in Baroda.

He can be contacted via e-mail at pravin@classofjava.com.

CHAPTER 1

Object-oriented Programming and Introduction to Java

In the mid-1980s, C was one of the most popular programming language. By the late 1980s, object-oriented concepts were well established; and C++ became the language of choice for C programmers wishing to use object-oriented concepts in their application development. C++ is basically C language in addition to some object-oriented features. C++ retains everything from the C language, i.e. all C codes are valid C++ codes. The primary idea behind extending the C language was that the existing C programmers would not need to learn a new language and syntax. The C++ language is quite large, and the maintenance of the code written in C++ is difficult. We often come across systems written in C++, which follow a mix of the procedural as well as object-oriented programming structures. It is really difficult to maintain such code.

The Java programming language has a syntax similar to C, but it is not an extension of C. Originally, development of Java was started under "*Project Green*," by a group of Sun engineers led by James Gosling (a computer scientist at Sun Microsystems, also known as the father of Java). This language was supposed to be used as a programming language for consumer electronics. James Gosling called the new language "*Oak*," after the tree outside his window. Later it was found that "*Oak*" was already trademarked by Oak Technologies. It was then renamed as "*Java*". For some reason, Java (or Oak), did not at that time become very popular as a language for consumer electronics. However, Java really took off as a language for development on the Web, after Sun Microsystems showcased the use of applets in a browser at *Sunworld '95* on May 23, 1995 (this date is also considered to be the birth date of Java). The first version of Java was later released in 1996. Since then Java has been a very popular programming language for the Web. Due to its use in the World Wide Web, a lot of people seem to think that Java is meant exclusively for developing applications on the Web. However, despite its popularity in Web applications, Java is a general-purpose programming language, and it is not meant exclusively for the Web.

1.1 WHY IS JAVA SO POPULAR?

The Java programming language has a number of features that make it the language of choice for most developers. The reason why Java is so popular is that it is object oriented, platform independent, does not use pointers, has support for multi-threading, has a robust exception handling mechanism, has good security features, allows us to create applets and servlets for use on the web, etc. However, most importantly Java is popular because Java is easy. Unless a language is easy to learn and use, it cannot become as popular as Java. Java enables us to do easily things that would normally be difficult in other programming languages. Using Java, we can very easily, with a few lines of code, add network capabilities and multi-threading to our application.

The primary reasons for the popularity of Java are as follows:

- Java is platform independent.
- Java provides applets and servlets for use on the Web.
- Java is an Object-oriented Programming Language.
- Java is easy.

Let us understand each of these features.

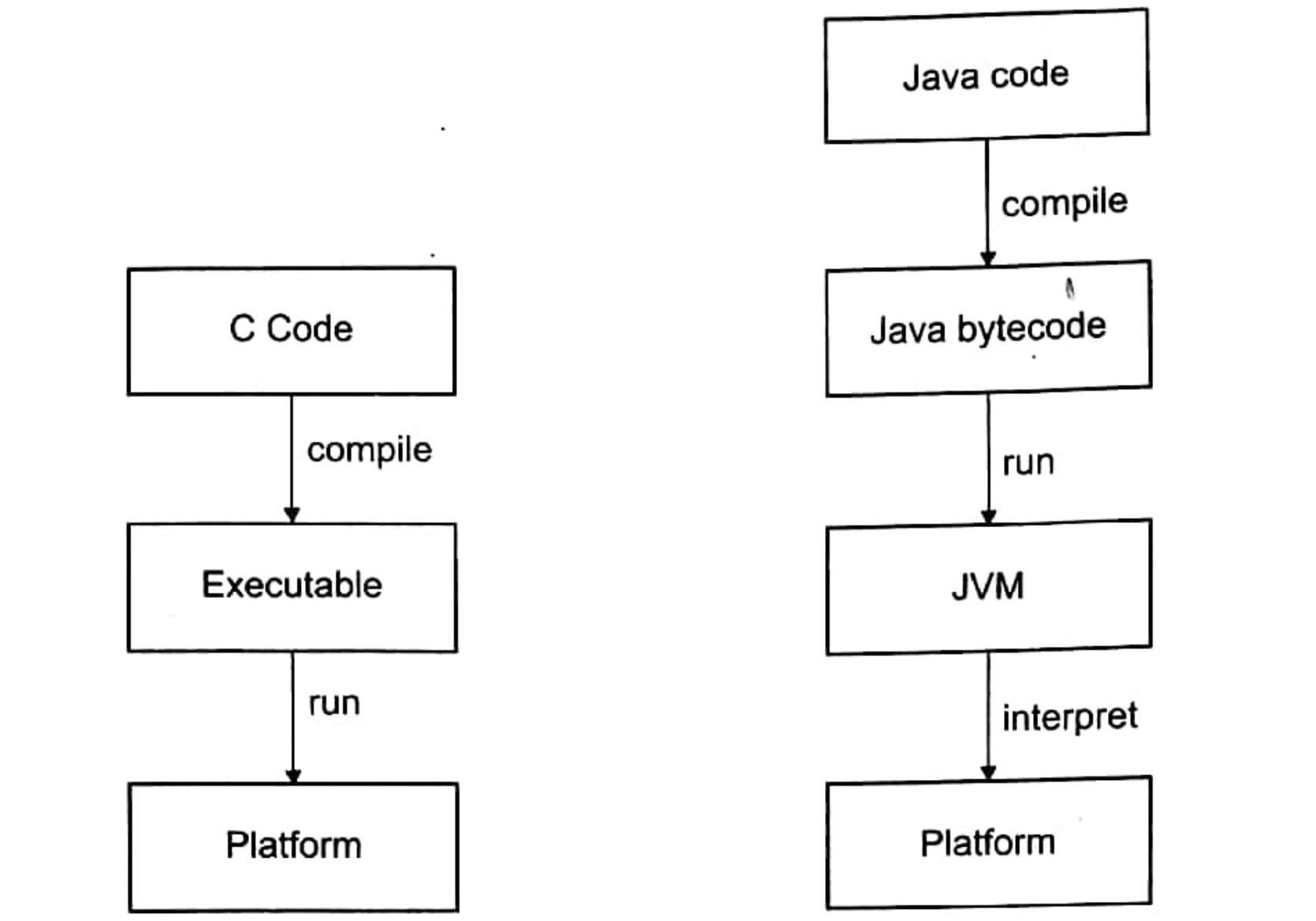
1.1.1 Platform Independent

What is meant by platform independence? Does it mean that it runs on any platform? Even C code can run on any platform. We just need to compile it for each platform. Here a *platform* refers to a combination of computer hardware and the operating system on which the code is being executed.

To understand platform independence, let us look at the steps taken for developing and executing an application using the C programming language and compare it with the steps required in case of Java programming language.

A C source file is first compiled and an executable file is created. What does an executable file contain? An executable is made up of instructions (machine code), which are from a specific instruction set (platform dependent). The executable generated by the C compiler contains instructions, which are specific to the platform for which the code is compiled, and so it will run only on the platform for which it has been compiled.

In the case of Java the process is not much different. A Java source file is also first compiled. The output of compilation is not an executable that would work only on the platform on which it is compiled, but it is the *Java bytecode*. The Java bytecode is similar to an executable, and it is made up of instructions from a specific instruction set. In this case, the instruction set is that used by the Java Virtual Machine (JVM). Independent of where (on which platform) we compile a Java source file, the Java bytecode generated is the same. This Java bytecode can run on any platform for which the JVM is available. At the runtime, the Java bytecode is interpreted from JVM instructions to platform-specific instructions and is executed by a JVM interpreter. Since the Java bytecode is interpreted, it would run a bit slower, compared to executing code from a platform-specific executable file. This difference in speed is negligible because the Java runtime environment, which is interpreting the Java bytecodes, has been optimized using various techniques including the use of a just in time (JIT) compiler. The



(a) Compilation and execution for C (b) Compilation and execution for Java

Figure 1.1 Compilation and execution for C and Java codes

optimization is a result of lots of efforts being put into it over many years. So in fact, Java is one platform. The Java code is written to be run only on one platform, and that is the JVM.

When we use the term Java, it can have two different meanings. The first is the Java Programming Language and the second is the Java Platform (JVM). There are two separate specifications maintained for each. One is for the language itself and is known as *The Java Language Specification* and the other is maintained for the platform and is known as *The Java Virtual Machine Specification*. The code written using the Java Programming Language is targeted to run only on the Java Platform. Like the Java programming language, now-a-days there are various scripting languages that are also getting compiled to be run on the Java Platform, e.g. JRuby, Jython, etc. (Figure 1.1).

1.1.2 Applets and Servlets for the Web

Let us look at some of the Java components that are most commonly used in Web-based applications. Applet and Servlet are the most commonly used Java components in Web-based applications. Web-based applications normally are based on a client and server interaction over the Internet. Most commonly the web applications are dependent on a Web server on the server side and a browser on the client side. There are various protocols for interaction between the client and the server. Most commonly this interaction uses the Hypertext Transfer Protocol (HTTP) protocol. The HTTP protocol is mainly used to transfer different types of documents from the server to the client whenever a client requests for them. The most common type of document downloaded from the server to the client is the Hypertext Markup Language (HTML) document. Other types of content can also be downloaded by the client from a Web server using the HTTP protocol.

When we want to retrieve some document from the Internet, we normally type the corresponding URL in the Web browser. A typical URL could be something like

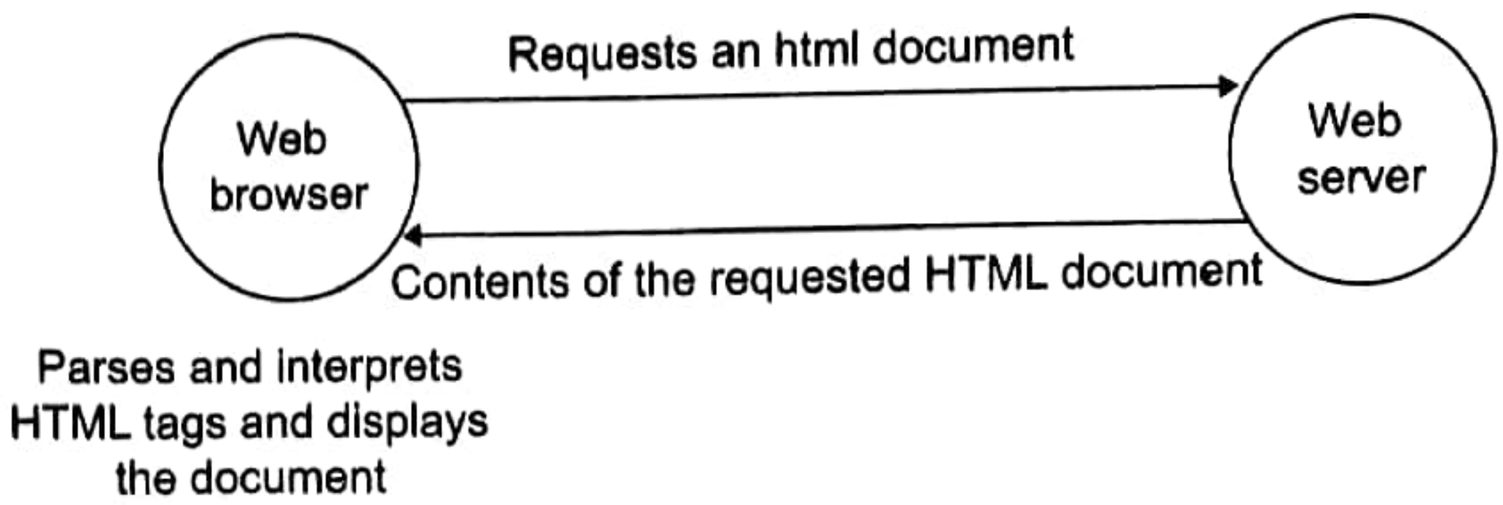


Figure 1.2 Interaction between the Web browser and the Web server for getting HTML content

`http://developers. sun.com/prodtech/index.html`. This URL has two parts, the initial part identifies the server and the protocol to be used for communicating with the server; while the second part identifies the resource to be accessed on the server. Here, for example, `http://developers.sun.com` identifies the server host name as `developers.sun.com`, and `http` identifies the protocol to be used for communicating with the server. The part `/prodtech/index.html` identifies the resource on the HTTP server. The Web browser would now establish a connection with the HTTP server on the machine identified by the first part of the URL. It then requests the HTTP server to provide the content of the file /prodtech/index.html. The server, in turn, would send the contents of the HTML file to the Web browser. The Web browser on receiving the contents of the HTML file would then parse and interpret the HTML tags and display the contents (Figure 1.2).

While parsing the HTML content, if the browser encounters some tag like `<img src= "/im/vnv1_sunlogo.gif" alt="sun.com" width="55" height="24" border= "0" >`, it would again make a connection with the web server and ask for the contents of the image file "/im/vnv1_sunlogo.gif". The Web browser knows how to display gif and jpeg files. Therefore, in this case, when it comes across a reference to image file "/im/vn1_sunlogo.gif", the browser downloads the gif file from the server and renders it appropriately. Like the `<img ...>` tag, an HTML file may contain the `<applet ...>` tag.

What is an applet? An applet is a specific type of Java bytecode, which can be embedded inside a HTML page and can be downloaded and run by a Web browser. Let us try to understand the working of applets over the Web. Applets and their working are explained in detail in Chapter 21.

To understand the working of an applet, let us once again look at the interaction between the Web browser and the Web server. A Web browser normally requests an HTML document (resource) from the Web server and then waits for the content of the HTML document to arrive from the server. On receiving the contents of an HTML document, the browser interprets the HTML tags in the HTML document and displays the document. Some of the HTML tags may require the browser to request some other types of files (resources) from the Web server. For example, when the browser, while interpreting the HTML tags, comes across a tag for an image like `<img src=''url for image file'' ...>` then the Web browser again sends a request to the Web server requesting for the contents of the image file like gif, jpeg, etc. The Web browser on receiving the contents of the image file displays the image (Figure 1.3).

The Web browser is a very versatile client and understands the various types of resources like images, audio and multimedia. The browser's handling of an applet is similar to the way it handles an image.

In case the browser comes across a tag for an applet like `<applet code="name of applet class" ...>`, it sends a request to the Web server requesting the contents of the applet

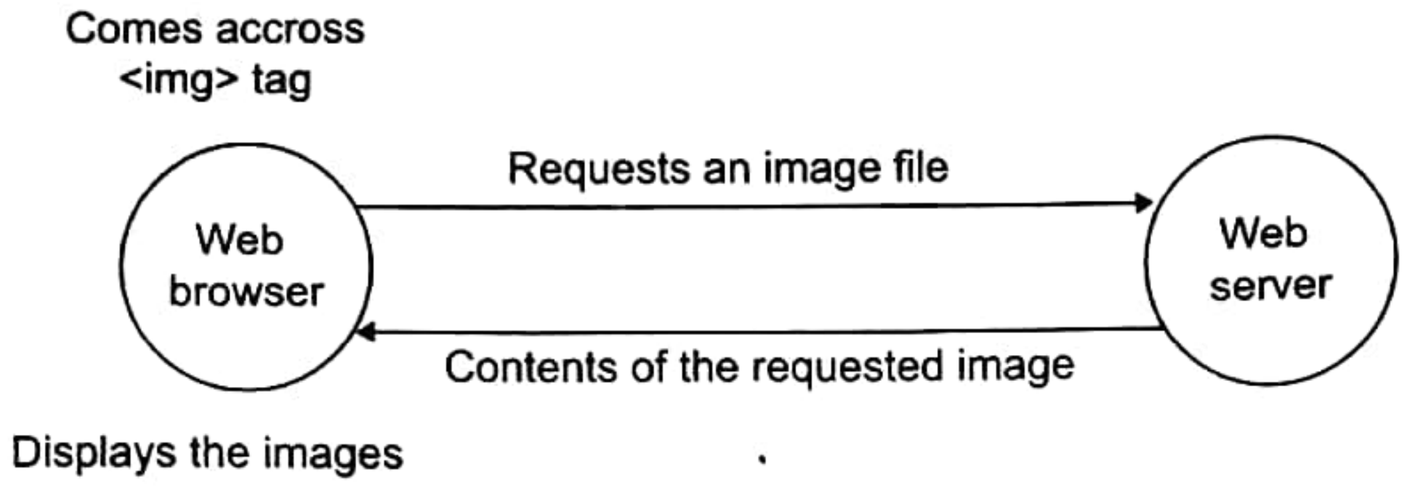


Figure 1.3 Interaction between the Web browser and the Web server for getting image content

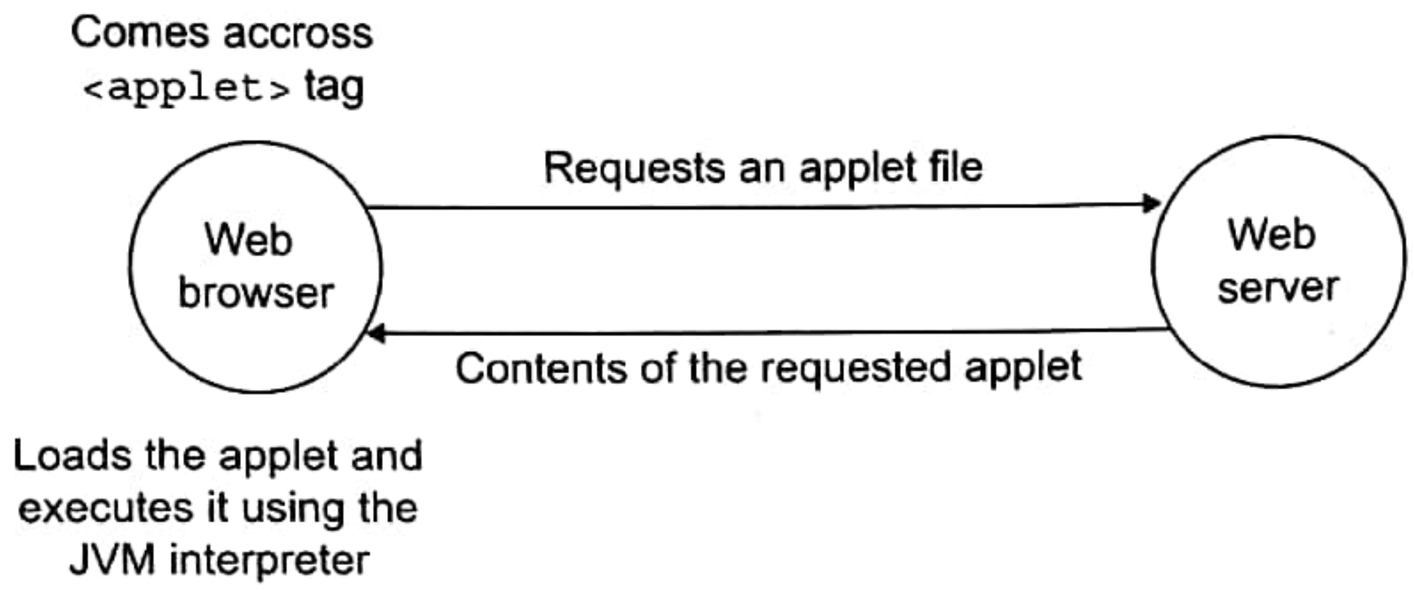


Figure 1.4 Interaction between the Web browser and the Web server for getting an Applet

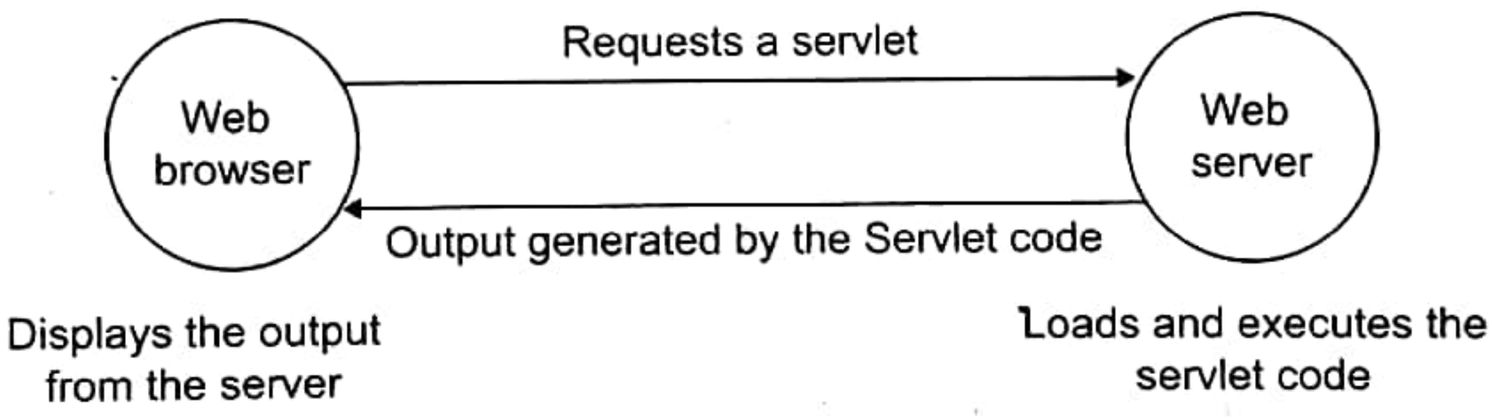


Figure 1.5 Interaction between the Web browser and the Web server for accessing a servlet

class. The applet class is made up of Java bytecodes. After receiving the contents of an applet class, the browser executes the Java bytecodes. For this, it is required that the Web browser be Java enabled (it has a JVM interpreter). This kind of handling of applets by the browser gives a lot of advantage in developing a client server application. Here the client application can be developed as an applet (Figure 1.4). The client code does not have to be installed on the client machines. Any machine connected to the Web with a Java-enabled browser can become a client. The client machine can have any platform. There is no requirement of maintaining various versions of the client for each platform. Also in case of upgrading the client code, there is no requirement of upgrading the code on each client machine. There is only one copy of the client code, which is maintained on the server.

What is a servlet? The servlet is a resource on the Web server, which is made up of Java bytecodes and would remain on the server; it is not delivered to the client like the applet. The Web browser may request a servlet invocation. When the Web server receives a request for a servlet, the Web server will locate the appropriate servlet, which is a special type of Java bytecode, which can be embedded inside a Web container (server). The Web server will load and execute the servlet code. The output, which is generated by the servlet code is then sent back as content to the Web browser (Figure 1.5). This allows dynamic content coming to the

Web browser instead of static content coming from a file. The servlet code can make use of all the Java technologies like JDBC, RMI, EJB, etc. to generate the desired content for display on the Web browser. Therefore, a servlet is not the content, it is a content generator.

1.1.3 Object-oriented Programming

What is Object-oriented Programming?-How does it differ from the conventional procedural programming? The difference is in the approach to problem solving using the programming language. In case of the conventional procedural programming, we as programmers follow a top–down approach, where the entire problem to be solved is looked at from the top as one problem. We now start by breaking it into smaller parts (procedures and functions) and refine it further; we then break it into still smaller levels till we reach the details. Normally in this approach, there is not much of re-usability of code across applications. Here we are thinking more about the application and solving it, rather than the re-usability of the code. In case of Object-oriented Programming, the approach is to first look at the various types of data types that are participating in our application. Here we design data types that are not available yet, and write the code for it and test it, i.e. we are ready with the details first and then start writing the application that would in turn use the data types created. These data types that are created once are re-usable in other applications. The data types created are quite re-usable, and are not created for the purpose of a single application. These data types (known as class) created are more general purpose as they are not created for the current application alone.

What are the commonly known Object-oriented Programming Languages? Java, C++, Smalltalk, Object Pascal (used in Delphi), etc. Java is an Object-oriented Programming Language. It uses a syntax similar to C, but it is much different from C++. One of the major differences between C++ and Java is that Java does not have pointers, instead it always uses references to access objects of a class. The reference variables of Java and how they differ from pointers are given in Section 3.3.

Object-oriented Programming involves creating new data types. Let us look at the evolution of the data types along with the evolution of the programming languages. All programming languages basically provide for data types and have operators for each of the data types. If we look at the evolution of the programming languages from the perspective of data types and the operators for the data types, we find that the initial programming languages (low-level languages) had a fixed set of data types and their operators. The next generation of programming languages like (FORTRAN, COBOL, PASCAL, C, etc.) allowed the programmer to define new data types over and above the data types provided by the language, but these languages do not allow the programmer to define new operators for these programmer-defined data types. The Object-oriented Programming Languages allow programmers to create new data types and also allow the programmer to define operators for these data types. The operators are used for obtaining information about the specific instance of the data type or for manipulating the instance (Table 1.1).

TABLE 1.1 Data types and operators for programming languages

Programming Languages	Data types	Operators
Low-level languages	Fixed by language	Fixed by language
High-level languages	Programmer defined	Fixed by language
Object-oriented languages	Programmer defined	Programmer defined

The Object-oriented way of solving a problem is more closer to the real-world way of solving a problem. In the real world, any problem is solved by collaboration between various persons, where each person is responsible for certain types of activities. The interaction between persons takes place by passing messages between them. Let us consider an example to understand this. Let us say we would like to send flowers to our friend who is in a distant town; then how is this problem typically solved?

We probably go to some florist in our town, say "J K Florist" and give him a message "sendFlowers". In the message we also specify the name and address of our friend, the type of flowers to be delivered and the quantity. How does "J K Florist" start processing our request (message)? He starts by looking at the address of the recipient and finds out that it is not a local address. In this case, it would be more convenient for him to find a florist in the town of the recipient. So "J K Florist" looks up his list of florists with whom he can collaborate and locates some other florist near our friend's place. He then sends the message "sendFlowers" to the other florist along with other details like the name and address of the recipient, type of flowers and quantity. The other florist on receiving the message starts processing the request and finds that the recipient has a local address. So he now looks at the type of flowers to be delivered and, from his list of flower producers, locates a 'flower producer' who can provide him the type of flowers to be delivered. He now sends a message "getFlowers" to the local 'flower producer', specifing the type of flowers and the quantity required. After getting the flowers from the 'flower producer', the florist then takes the help of a 'delivery boy' to deliver the flowers to the recipient. To do this he gives the message "deliver", along with the recipient address and the flowers to be delivered, to the 'delivery boy' (Figure 1.6).

Let us analyze the above steps and associate them with the various Object-oriented programming terminologies.

Class and object: Here we gave the message "sendFlowers" to "J K Florist" and did not give it to "Happy Toys" (who sells toys). The reason is that "Happy Toys" will not understand the message "sendFlowers", whereas "J K Florist" understands the message "sendFlowers". Why does "J K Florist" understand the message "sendFlowers"? The reason is that "J K Florist"

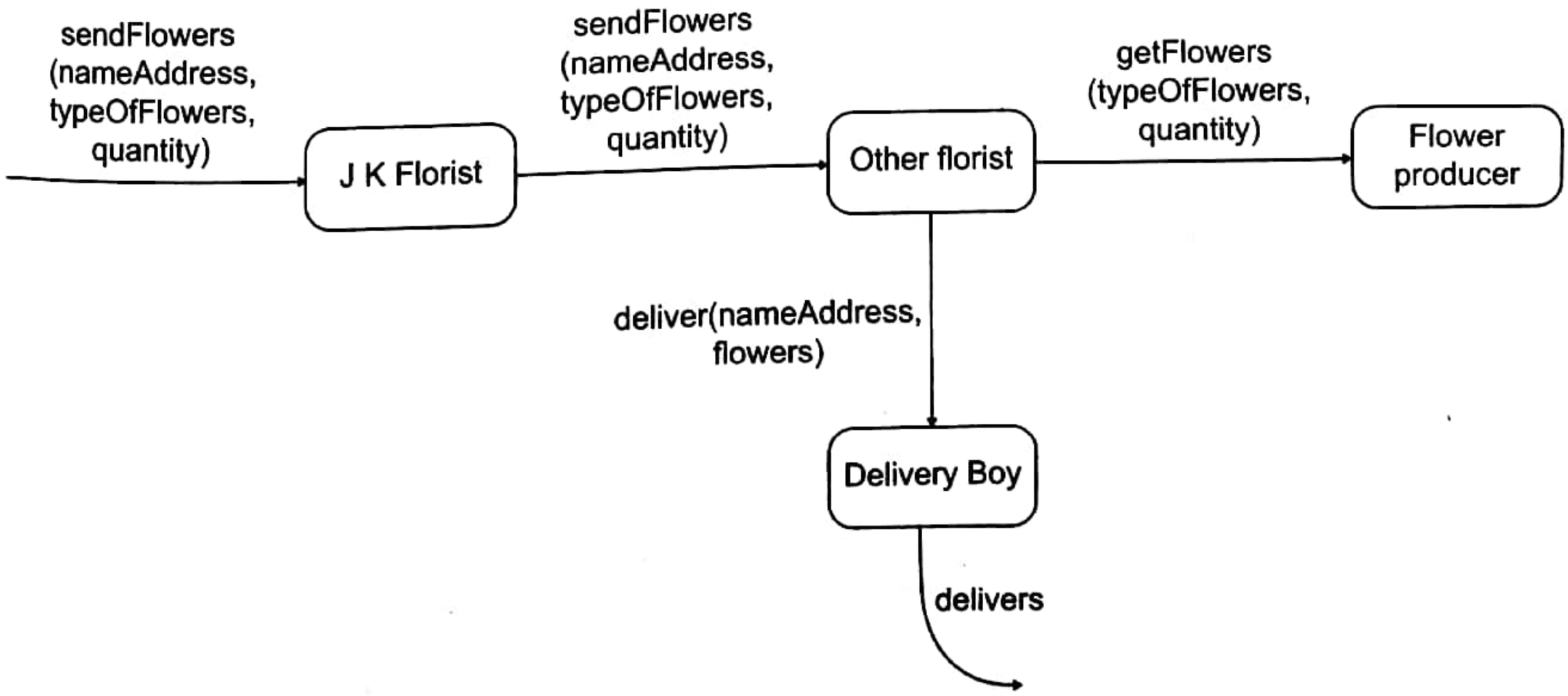


Figure 1.6 Interaction between various objects for solving a problem

is a florist. All florists will understand the message "sendFlowers". Here "J K Florist" is an instance of the class "Florist". "Florist" is a class, and has a well-defined set of messages that the instances of this class can understand. A class (data type) defines a list of messages (operators) understood by its instances. Messages understood by instances of a particular class are like the operators available for a data type. Here, the class is a programmer-defined data type.

Data encapsulation: Data encapsulation is the ability to hide the data structures used by a data type. Here, for instance, the florist uses a diary to maintain the list of addresses of florists in other towns. This diary is not accessible to others directly. What would happen if this diary were directly accessible to all? Here, the danger is not about the information being available to all from the diary, but there is a bigger danger since anyone would also be able to make entries (valid or invalid) into this diary, and the diary would lose its meaning. Instead, if it is accessible to only the florist then the florist can always answer any queries we have by looking up into the diary. In data encapsulation, the data structure is encapsulated by some code. The data structure is not directly accessible, but instead the data are accessed only by the code, which encapsulates them. The data are accessed by interaction with the instance, which would involve, invoking some code on the instance to fetch the data. Each message requires some set of inputs (which may be empty), thus defining a contract for using the message.

Inheritance: Again here we have more kinds of interaction with "J K Florist", not simply because he is a "Florist", but because he is a shopkeeper. We know, for example, that we will be asked for money as part of the transaction and in return for payment we will be given a receipt, i.e. he will be able to understand the messages like "acceptPayment" and "issueReceipt". These messages are understood by grocers, stationers, toy sellers and other shopkeepers. Since the class "Florist" is a more specialized form of the class "Shopkeeper", all messages that can be given to shopkeepers can also be given to all florists. Because all florists are shopkeepers, whatever is true of "Shopkeeper" is also true of "Florist" and hence of "J K Florist" (Figure 1.7).

Polymorphism: What is polymorphism? There are many forms. There are different types of polymorphism; what we commonly understand in programming are overloading and overriding. Let's take an example of a Plotter to understand overloading. What are the typical interactions with a plotter? A plotter would have interaction to draw various kinds of shapes. There would be an interaction to draw a circle, a message called drawCircle. What will be the inputs given to the plotter in order to draw a circle? The radius and the center point. So, the drawCircle message requires two inputs. Now, we have another requirement; we have got three points in a plane, and there is a unique circle that passes through it. So to ask the plotter to draw a circle we may have to compute the radius and center point of the circle from these three points and then interact with the plotter, but the plotter may additionally provide for another message, which can do the task of converting the three points into the radius and the center point and also support another way of using drawCircle. It can support a message, such that, if we give it three points in a plane, then also it should be possible to draw a circle. So we have two forms of interaction with the plotter for drawing a circle; we can either provide the radius and a center point or we may provide three points in a plane. This is overloading.

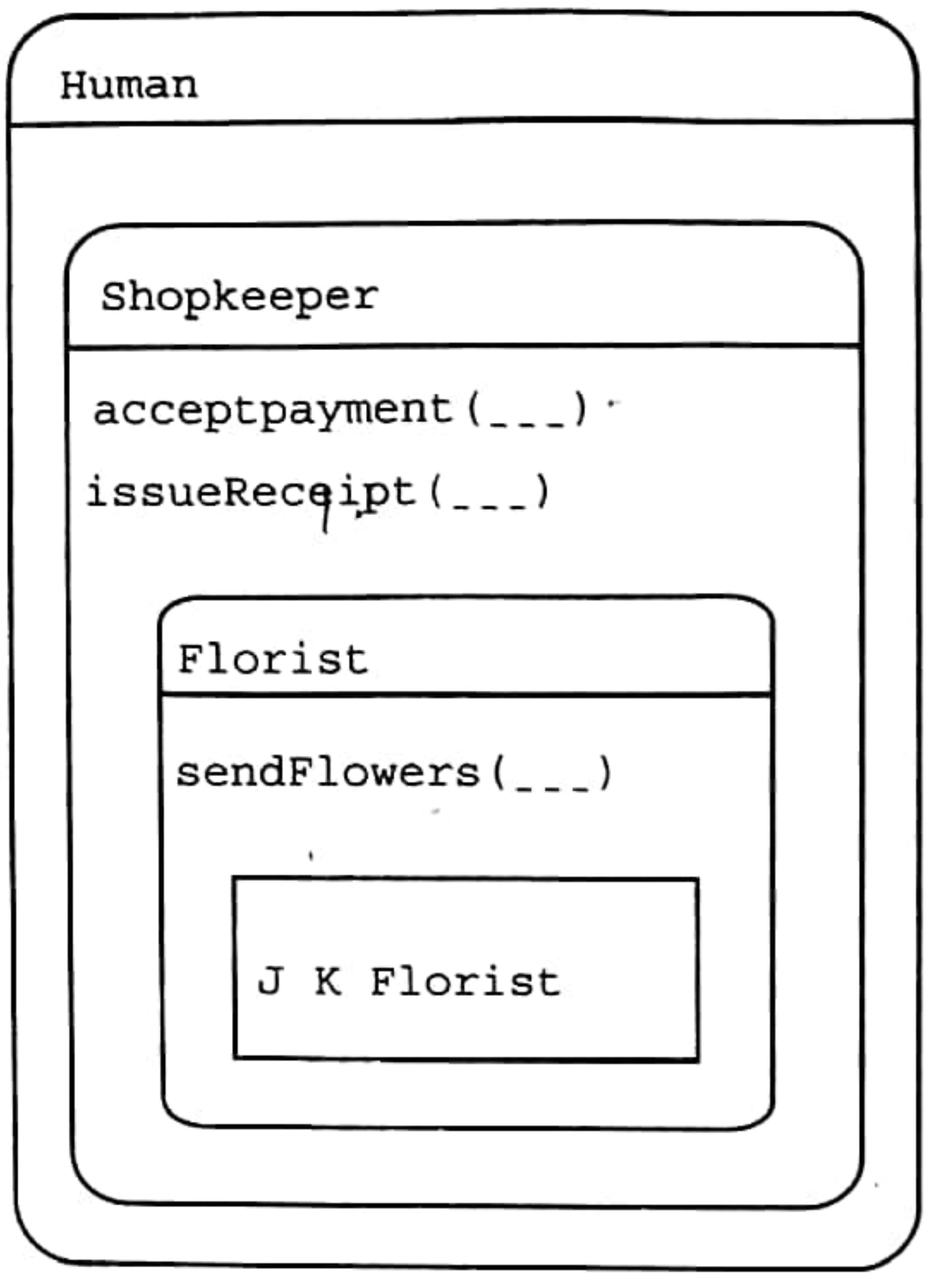


Figure 1.7 Class inheritance

Another form of polymorphism is found in case of overriding. Overriding involves inheritance. Overriding is a case of exception to the general rule. In case of inheritance the sub-class inherits all the contracts of the super-class. It even inherits the implementations of the contracts. Sometimes there are cases where the sub-class wants to have a different implementation of the contract, compared to the implementation as available from the super-class. This is where overriding of the contract is done. Also there is a concept of abstract data types, having abstract methods. These abstract methods signify the contracts that need to be supported by the sub-class. Here different sub-classes will have different implementations of the same contract. The abstract method is being implemented in different ways by different sub-classes, e.g. we have a class called Shape. Every Shape should have an area method, which could return the area of the shape. This area method would be implemented differently for different sub-classes of the Shape. Rectangle, Triangle, Circle, Pentagon, Hexagon and other sub-classes of Shape each would have its own different implementation of the area method. Therefore we find different implementations of the area method for Shape in its various sub-classes.

1.1.4 Easy

Java is an easy programming language. It has a syntax similar to C language. It has removed pointers and follows the Object Reference Model. In case of Java, the programmer does not have to bother about deallocation. Deallocation is taken care of by the garbage collector within the JVM. Java is easy in terms of programming for network applications and multi-threaded applications. An average Java programmer can easily write network applications and even use multi-threaded features in the JVM, which is difficult in most of the other programming languages. Writing multi-threaded applications normally requires high skill and experience in other languages.

1.2 JAVA COMMUNITY

An important thing to note about the Java development is that the Java development has always been in the open. There are well-defined specifications that are maintained for both the Java programming language and the JVM. These specifications are in the open. These specifications get revised as new ideas emerge, and most of the revisions are backward compatible, i.e. most of the features of the old versions would always work with the new versions. So applications developed using older versions would work on the newer versions. Sun Microsystems has its implementation of these specifications. These implementations of Java have always been in the open for scrutiny for over a decade.

The development of the Java application programming interface (API) is now in the open source. There are various open-source projects that are being developed at the java.net site. One of the communities at java.net is the OpenJDK community, which is now looking after the development of the JDK and its API. Several experts are members of this community, and they are continously contributing to the future development of Java. A lot of young professionals are gaining from the advices by the experts in this community.

1.3 RICH API

Therefore, Java is an Object-oriented Programming Language, where we do not have to worry about multiple platforms. It has a well-defined specification for the platform and the language. The Java programming language is well supported by a rich API. The API takes care of most of the requirements for almost all kinds of applications.

These are some of the reasons for the Java platform becoming the first choice for most of the application developers.

LESSONS LEARNED

- Java is not just a programming language; it is also a platform.
- The Java programming language works only on the Java platform.
- The Java Virtual Machine enables the same Java application to be used on heterogenous platforms.
- The Java applications are compiled into bytecodes, which are interpreted on the native platform at runtime.
- Object-oriented Programming is about identifying various data types from the real world, and the kind of interactions that are supported by those data types.
- Object-oriented Programming supports the concepts of data-encapsulation, inheritance and polymorphism.

EXERCISES

1. State which of the following are true or false:

(a) Java is a programming language meant for use on the Web only.

(b) Java is one platform.

(c) The relationship between Bird and Parrot is that the Parrot is an Object of the class Bird.

(d) A sub-class encapsulates less functionality than its super-class.

(e) Applets are deployed on the client machine.

(f) Servlets are deployed on the server machine.

2. Fill in the blanks:

(a) The birth date of Java is _____.

(b) _____ is known as the father of Java.

(c) The original development of Java was started under Project _____.

(d) Object is an instance of a _____.

(e) Java output generated by the Java compiler is _____.

(f) To optimize the execution of the Java bytecode, the JVM uses a _____ compiler.

3. Explain how platform independence is achieved in Java.

4. Explain the concept of data encapsulation, inheritance and polymorphism in Object-oriented Programming Languages.

5. Explain how applets are used in a browser.

6. Explain how servlets are used on the server side.

7. Observe the interactions involved in the process of booking a railway ticket. Identify the various objects involved and the interactions between the objects in order to solve the problem of booking a railway ticket.