# 1

# GETTING STARTED

In the software industry across the globe, there is a need to develop large and complex software. This software should be platform independent, internet enabled, easy to modify, secure, and robust. To meet these requirements object-oriented paradigm has been advocated. Based on this paradigm, Java programming language emerges as the best programming environment. Java is used for mobile programming, internet programming, and many other applications compatible with distributed systems. This book aims to cover the essential components of Java programming so that the readers can improve their skills, cope with the demands of the IT industry and solve the problems in their own field of study.

## 1.0 Introduction

*Necessity*

IT industry operates in such a diverse environment that software programs should not be confined to a single PC. Rather, they should move from a single user environment to mainframes, to networks, to network of networks, and so on. A giant program can be developed which integrates numerous mini programs which were developed by geographically distributed programmers, and all of them can be connected online. Such a distributed programming techniques and environments are no longer a dream but a reality, and it is Java, the Internet programming language, which makes it possible.

*Scope*

Java was developed by Sun Microsystems Inc. with the intent to create a dynamic, object-oriented programming (OOP) language, suitable for using the same types of development tasks as C and C++, but without the difficulties and bugs common to those languages. This is why Java has been touted as "a better C++"; it contains all the essential OOP features, but with lower complexities than C++. Sun describes Java as a "simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language."

*Your learning*

In this chapter we shall briefly highlight the various features of Java programming, its superiority over other programming languages, and the tools available for its programming. At the end, you will be able to run your first program in Java.

## 1.1 Concept of Programming

*Application systems*

Computer programming is used to build a software system (e.g., Microsoft Word, Library Information System, Internet Explorer browser, mobile app, printer's driver software, to name a few). These software systems take an input and then produce an output (Figure 1.1). Note that input/output can be of any form like text, audio, video, handwritten character, and so on.

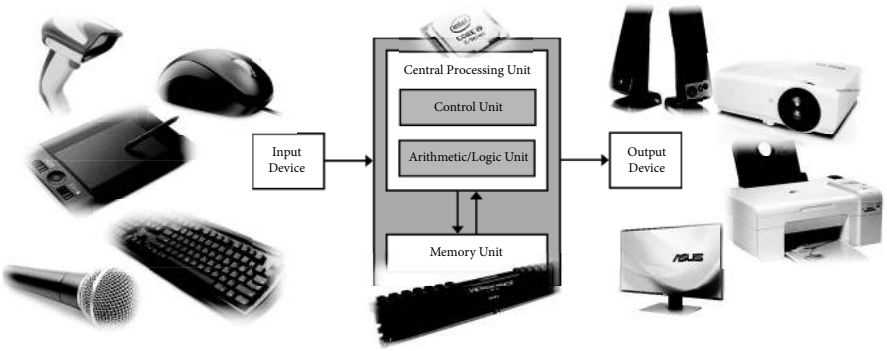


Figure 1.1 Computing system

### 1.1.1 Computer and Its Working

*Programming*

In order to process an input and produce an output, a program is required. A program contains of steps to transform input to a corresponding output. A computing system follows a basic architecture, popularly called Von Neumann architecture, to run programs. According to this architecture, a program is loaded into the memory of the computer. This program is then executed by the Central Processing Unit (CPU), which consists of a control unit, an arithmetic unit, and a logic unit.

*Program translation*

Thus, a program is a set of steps (or, instructions to the CPU). A program is written using a programming language. There are different kinds of programming languages, such as machine language (first generation language or 1GL), assembly language (second generation language or 2GL), high-level language (third generation language or 3GL), and so on. But a

computer can understand instructions only in machine language. Hence, a program in 2GL or 3GL has to be translated into machine language. That is done by using a program translator, also called a compiler or interpreter (Figure 1.2). As shown in the figure, an assembler translates a program in assembly language to machine language and a compiler or interpreter translates a program in high-level language to machine language.

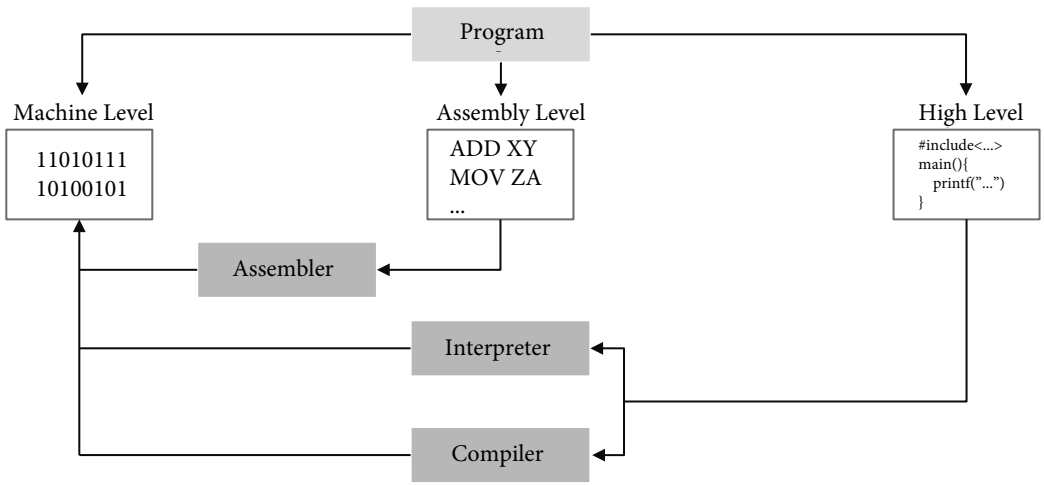


Figure 1.2 Program translation schemes

*Java programming*

Like different high-level languages (e.g., C, C++, Pascal, Python, Java), machine languages also vary from one CPU to another (e.g., Intel, Sun, IBM, Macintosh), and from one operating system (OS) to another (e.g., Windows, Solaris, MacOS, Android). Different assemblers, compliers or interpreters are required according to the different programming environments (CPU or OS). In contrast, the approach in Java is different. Here, a Java compiler translates a Java program into Java byte codes (according to a virtual machine, which is a hypothetical CPU). These byte codes are then interpreted by a Java interpreter to run the program. So, an application developed using Java is in the form of byte codes. This application, then, can be executed in any hardware or operating system and thus programming becomes architecture neutral or platform independent.

### 1.1.2 Programming Paradigms

Out of the numerous programming paradigms, two are followed more often in industries: function-oriented programming paradigms and object-oriented programming paradigms.

*Concept of function*

A function in its general sense is a mapping procedure from an input to an output. It takes one set of data as an input and produces another set of data as output. For example, $x = GCD(m,n)$ is a function which returns $x$ as the greatest common divisor given two input numbers $m$ and $n$. In another example, given an array of numbers, a sorting function on this data set will produce another array of numbers. A function in its simplest form can do one functional task. If a function is to complete more complex tasks, then it can be decomposed into a number of smaller functions (called sub functions, sub sub-functions, and so on). For example, there can be a function to find if a person's email address is available on the Internet repository. Here, the basic function is *search(…)*, which can be composed of *retrieve(…)* followed by *sort(…)* and then *check(…)*. Note that *sort(…)*, *check(…)* functions can be in terms of some other smaller functions, and so on.

*Function-oriented programming*

A programming principle which is based on concept of function is called function-oriented programming. To build a system, we have to build a set of functions. The system itself is a function, which can be composed as a set of functions. For example, in a bank ATM system, withdraw money, deposit money, balance enquiry, PIN change, and so on, are functions. The concept of function-oriented programming can be better understood through Figure 1.3. There are several functions in this figure which are centered around some data to be processed. Here, data is globally available to the functions. Thus, in function-oriented programming, a system is a collection of functions.

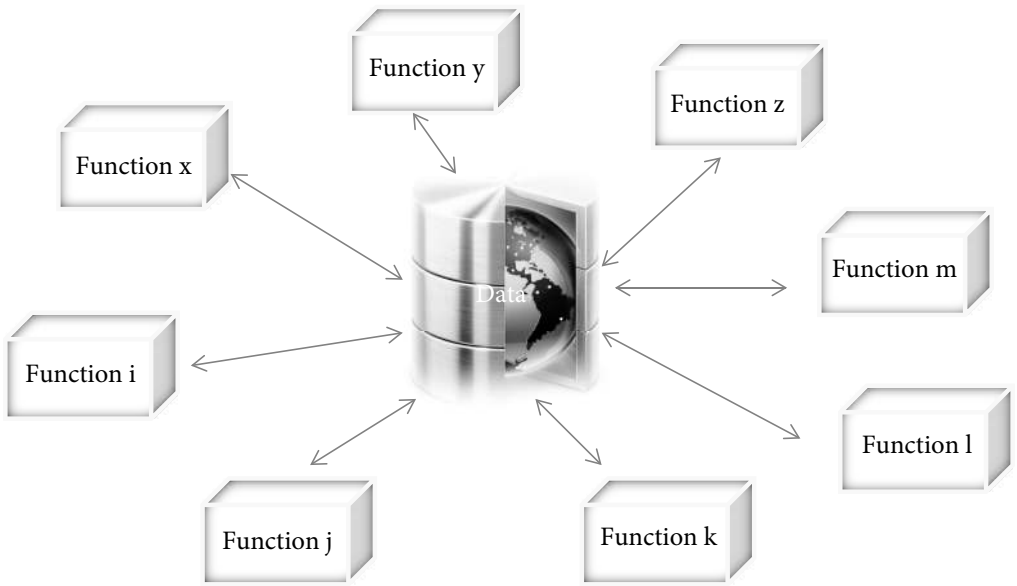


Figure 1.3 Function-oriented programming

*Concept of object*

Object-oriented programming, in contrast to function-oriented programming, is based on the concept of object. A function in function-oriented programming is a verb, whereas an object in object-oriented programming is a noun. In real world, a material thing that can be seen and touched, or a person or thing to which a specified action or feeling is directed, is called an object. A student, a car, and a book are some examples of objects. An object is defined by a set of attributes. For example, name, roll number, date of birth, and department are some attributes by which a student (object) can be defined. In real world, an object can send (or receive) message(s) to (or from) another object. For example, a student sends a message "issue" to a book and book can send a message "fine" to the student.

*Object-oriented programming*

Such a real-world scenario is modeled in object-oriented programming. A system's functionality is accomplished by means of communication of messages among the objects. On receiving a message, an object changes its state. States of an object are characterized by the change of value(s) of attribute(s). Each object has its data in it and also knows to how to process that data. Both data and operation on that data together define an object. A task is carried out by the invocation of operations by another object. Such a programming paradigm is illustrated in Figure 1.4.

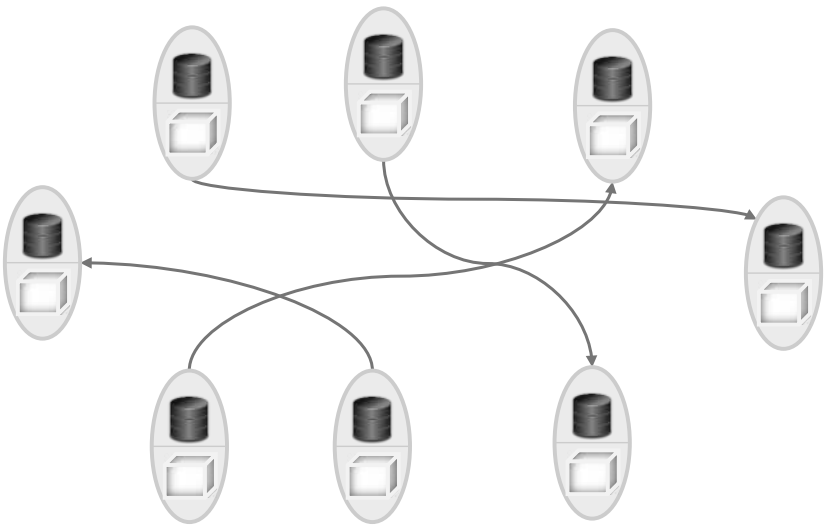

Figure 1.4 Object-oriented paradigm

In object-oriented programming, a system is a collection of objects. Unlike function-oriented programming, there is no global data here. Instead, the data is locally distributed among the objects.

Let us consider the system of getting an email address from the Internet. Here, a user "sends" a person's profile to a browser. The browser "retrieves" all email addresses "matching" the profile from the Internet. The browser then "sends" the list to the person. The person then "retrieves" the email address.

## 1.2 Object-Oriented Programming Paradigms

*OOP paradigms*

The programming principles followed in function-oriented programming include how to define a function, how a function can be decomposed to smaller functions, and how data can flow from one function to another. In contrast, object-oriented programming follows radically different programming paradigms. There are mainly four object-oriented programming paradigms: Encapsulation, Inheritance, information hiding and polymorphism. All these OOP paradigms are discussed in the following sections.

### 1.2.1 Encapsulation

*Examples of objects*

In object-oriented programming, the object is at the center of the stage. The object contains both data and the methods that operate the data. Encapsulation is the process of wrapping method and data together into a single unit. Figure 1.5 illustrates the encapsulations of two types of objects: a book and a borrower. The encapsulation procedure allows a programmer to define the type of object (such as books, borrowers). Such a type is called class. Thus, in Figure 1.5, *Book* and *Borrower* are two classes which define two types of objects. Notice that for simplicity, the body of the methods are not included in Figure 1.5.

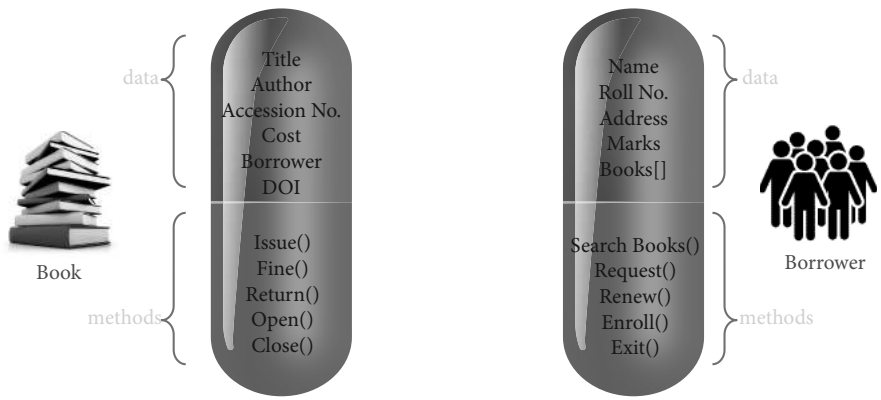


Figure 1.5 Encapsulation in OOP

### 1.2.2 Inheritance

*Generalized to specialized object*

Inheritance is a mechanism in which one object acquires all or some of the properties and behaviors of a parent object. This paradigm allows a programmer to extend a class into another. Such an extension is useful to update or modify the system without disturbing the existing system. It also allows code sharing.

Figure 1.6 illustrates the concept of inheritance. In this figure, *Book* is a class. Two child classes, namely *Text* and *Reference* are inherited from the parent class, *Book*. By this inheritance, the class *Text*, for example, inherits all the fields and methods in the class *Book*. In addition to these, the class *Text* has its own field and methods. The methods *Close()* and *Open()* are redefined in the class *Reference*.

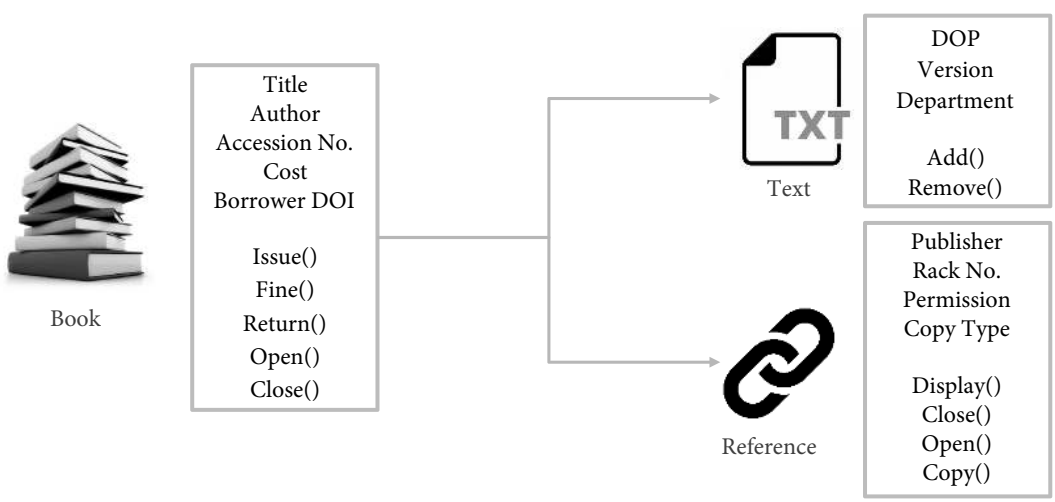


Figure 1.6 Inheritance in OOP

### 1.2.3 Information Hiding

This paradigm is allowed to limit the accessibility of fields and methods outside the class. This is particularly applicable when a class is inherited from another class. In such situations, the programmer may limit the access of some method and field to the methods in the child class. In object-oriented programming, such a mechanism is called information hiding.

Figure 1.7 illustrates the information hiding mechanism. In this case, we are talking about the class *Book*. It may be noted that there are access specifiers, namely *Public*, *Private*, and *Protected*. All fields and methods under *Public* access specifier are accessible to any methods belonging to any class. The field and methods which are under *Private* access specifier are accessible only to the methods inside the class *Book*, whereas the fields and methods under the *Protected* specifier are accessible to methods in the child class(es) of the class *Book*.

Book

Public
Title
Author
Protected
Account No.
Private
Cost
Public
Issues()
Returns()
Protected
Resave()
Private
Open()
Close()

Figure 1.7 Information hiding in OOP

### 1.2.4 Polymorphism

*Same name but different activity*

In object-oriented programming, polymorphism (from Greek, meaning "having multiple forms") is the characteristic of being able to assign a different meaning or usage to the same thing but in different contexts, specifically, to allow an entity such as a variable, a method, or an object to have more than one form. The idea of polymorphism in an object-oriented paradigm can be better understood from Figure 1.8. The figure includes four polymorphic representations of the method `Add(…)`. Here, the method name `Add(…)` has polymorphic behavior. When `Add(…)` is called with `x`, `y`, two integer numbers, it will add the value and return the result. However, the same method, with two documents as input, will merge the two documents into a single document.

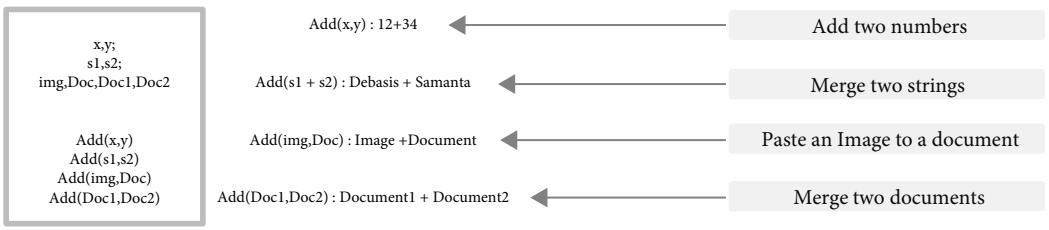


Figure 1.8 Polymorphism in OOP

## 1.3 About Java Programming

*Origin of Java*

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. This small team of Sun engineers is called the Green Team. The language was called "Greentalk" by James Gosling, and file extension of a program file was `.gt`.

*Starting point*

The language was originally designed for small, embedded systems in electronic appliances like set-top boxes, but it was a very advanced technology for the digital cable television industry at that time. Subsequently, the language was called "Oak" and was developed as a part of the Green project. The team members initiated this project to develop a language for digital devices. Later, this technology was used by Netscape as it was suitable for networking.

*The name Java*

The language was called "Oak" as oak is a symbol of strength and is the national tree of many countries like USA, France, Germany, and Romania. The team wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to speak. In 1995, Oak was renamed as "Java." Java is an island in Indonesia where coffee was first produced (called java coffee).

*Breakthrough*

In 1995, *Time* magazine called Java one of the "Ten Best Products of 1995." JDK (Java Development Kit) 1.0 was released on January 23, 1996.

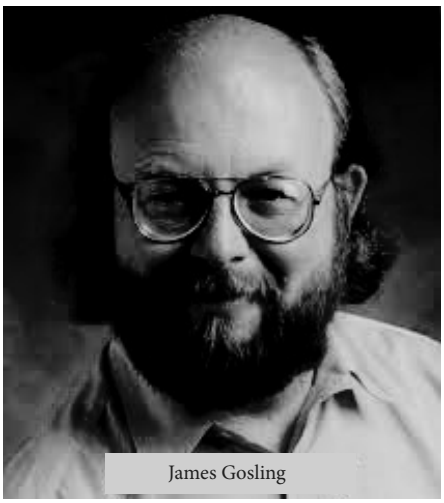



Figure 1.9 History of Java

### 1.3.1 Why Java?

*Definition of Java*

Java programming concept possesses all the advanced programming features of the recent programming languages. In addition, it has something special which makes it the first of its kind. The developer of Java described the Java language as: simple, portable, secure, high-performance, multithreaded, interpreted, platform independent, dynamic, architecture neutral, object-oriented, and robust. A few notable characteristics in Java programming language are summarized in Figure 1.10.

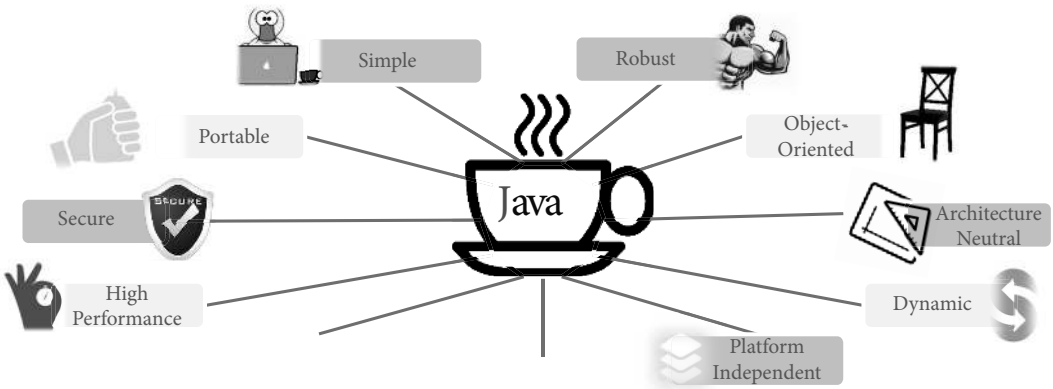


Figure 1.10 Characteristics of Java language