# Chapter One

# Introduction to Object-Oriented Programming

## Programming Paradigm

**Programming paradigm** is a way of conceptualizing what it means to perform computation and how tasks to be carried out on a computer should be structured and organized. There are two types of programing paradigm: Procedural Oriented Paradigm and Object-oriented programming. Paradigm is used to describe how a program is constructed.

1. **POP** follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. Each step is carried out in order in a systematic manner so that a computer can understand what to do. The program is divided into small parts called functions and then it follows a series of computational steps to be carried out in order.
2. **OOP** is a high-level [programming language where a program](http://www.differencebetween.net/technology/difference-between-abstraction-and-encapsulation/)is divided into small chunks called objects using the object-oriented model, hence the name. This paradigm is based on objects and classes.

***Comparison between OOP and POP***

1. **Definition**

OOP stands for Object-oriented programming and is a programming approach that focuses on data rather than the algorithm, whereas POP, short for Procedure-oriented programming, focuses on procedural abstractions.

1. **Programs**

In OOP, the program is divided into small chunks called objects which are instances of classes, whereas in POP, the main program is divided into small parts based on the functions.

1. **Accessing Mode**

Three accessing modes are used in OOP to access attributes or functions – ‘Private’, ‘Public’, and ‘Protected’. In POP, on the other hand, no such accessing mode is required to access attributes or functions of a particular program.

1. **Focus**

The main focus is on the data associated with the program in case of OOP while POP relies on functions or algorithms of the program.

1. **Execution**

In OOP, various functions can work simultaneously while POP follows a systematic step-by-step approach to execute methods and functions.

1. **Data Control**

In OOP, the data and functions of an object act like a single entity so accessibility is limited to the member functions of the same class. In POP, on the other hand, data can move freely because each function contains different data.

1. **Security**

OOP is more secure than POP, thanks to the data hiding feature which limits the access of data to the member function of the same class, while there is no such way of data hiding in POP, thus making it less secure.

1. **Ease of Modification**

New data objects can be created easily from existing objects making object-oriented programs easy to modify, while there’s no simple process to add data in POP, at least not without revising the whole program.

1. **Process**

OOP follows a bottom-up approach for designing a program, while POP takes a top-down approach to design a program.

1. **Examples**

Commonly used OOP languages are C++, Java, VB.NET, etc. Pascal and FORTRAN are used by POP.

## Programming Language

Programming languages are languages that programmers use to write code/program to instruct a  
computer. They are languages designed for programming a computer. Computers do not  
understand human languages, so programs must be written in a language a computer can use.  
There are hundreds of programming languages, and they were developed to make the  
programming process easier for people. Programming languages are classified into three  
categories.

These may be divided into three general types:

1. Machine languages
2. Assembly languages
3. High-level languages

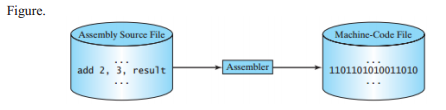
**Machine Languages**:

Machine language is a set of primitive instructions built into every computer. The instructions are in the form of binary code, so you have to enter binary codes for various instructions. Program with native machine language is a tedious process. Moreover the programs are highly difficult to read and modify. For example, to add two numbers, you might write an instruction in binary like this: 1101101010011010

***Assembly Language***

Programming in machine language is a tedious process. Moreover, programs written in  
machine language are very difficult to read and modify. For this reason, assembly language  
was created in the early days of computing as an alternative to machine languages. Assembly  
language uses a short descriptive word, known as a **mnemonic**, to represent each of the  
machine-language instructions. For example, the mnemonic add typically means to **ADD**  
numbers and **SUB** means to subtract numbers. To add the numbers 2 and 3 and get the result,  
you might write an instruction in assembly code like this: add 2, 3, result

Assembly languages were developed to make programming easier. Since the  
computer cannot understand assembly language, another program called an **assembler** is  
used to translate assembly-language programs into machine code, as shown in the following

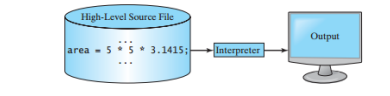


Writing code in assembly language is easier than in machine language. However, it is still  
tedious to write code in assembly language. Writing in assembly requires that you know  
how the CPU works. Assembly language is referred to as a ***low-level language***, because  
assembly language is close in nature to machine language and is machine dependent.

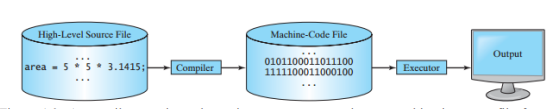
***High-Level Languages***

They are platform-independent, which means that you can write a program in a high level  
language and run it in different types of machines. High-level languages are English-like  
and easy to learn and program. The instructions in a high-level programming language are called  
***statements***. For example, the following is a high-level language statement that computes the area  
of a circle with a radius of 5: area = 5 \* 5 \* 3.1415;

There are many high-level programming languages, and each was designed for a specific  
purpose. Some popular ones are C, C++, C#, VB and Java. But for this course Java is our focus. A program written in a high-level language is called a ***source program or source code***.  
Because a computer cannot understand a source program, a source program must be translated  
into machine code for execution. The translation can be done using another programming tool  
called an ***interpreter or a compiler.*** An interpreter reads one statement from the source code,  
translates it to the machine code or virtual machine code, and then executes it right away, as  
shown in the following figure.



A compiler translates the entire source code into a machine-code file, and the machine-code file  
is then executed, as shown in figure below.



The difference between compiling and interpreting is as follows. Compiling translates the high level code into a target language code as a single unit. Interpreting translates the individual steps in a high-level program one at a time rather than the whole program as a single unit. Each step is executed immediately after it is translated.

## Overview of OO principles

All object-oriented programming languages provide mechanisms that help you implement the object-oriented model.

**Object:** An object is a self-contained abstraction of an item. The item could be a student record, employee record, a customer record or any screen window. An object includes all the data necessary to represent the item and the methods that manipulate the data. The perfect object knows everything about itself, including how to get input, give output and manipulate its data. Otherwise, an object is an *instance* of a class. It can be uniquely identified by its name and it defines a state that is represented by the values of its attribute at a particular point of time.

**Class:** Class is user-defined data type that is used to implement an abstract object, giving us the capability to use OOP with Java. A class includes members. A member can be either data known as a data member or a method, known as a member method.

**Members:** A class has one or more variable types, called members. The two types of members are *data members* and *member methods*.

* *Data members*: data members of a class are exactly like the variables in structures.

eg. Class Car {

String name; // data member

Color color;

}

* *Member methods*: are methods/ functions defined within a class that act on the data members in the class.

eg. class First{

Void new\_line ()//this is a member method

{

System.out.println ();

System.out.println (“Hello”);

}

}

**Class member visibility:** Each member function and data member of a class has an attribute known asits *visibility*. A class member’s visibility determines who can use the member.Java has four levels of visibility as follows.

* public
* private
* protected
* default

Public and private visibilities will be discussed here. But protected visibility will be presented in the inheritance section.

***-*** *eg. class Employee{*

*private String empName;*

*private String address;*

*private String city;*

*}*

Here, the data members of the Employee class are declared as private. Private members can be accessed only by a member method of that class. They can’t be accessed by a non-member method; the data is hidden from the outside world.

***-*** *eg. class Employee{*

*public String empName;*

*public String address;*

*public Sring city;*

*}*

This class contains the class visibility label public. By declaring a class member public, we make that member accessible to both member and nonmember methods.

**Encapsulation:** it is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. The code and data that constitute a class are called members of the class. When code and data are linked together in this fashion, an object is created. In other words, an object is the device that supports encapsulation. Within an object, code, data, or both may be private to that object or public.

**Polymorphism:** Polymorphism is the quality that allows one interface to access a general class of actions. It is a method the same as another in spelling but with different behavior. Generally, polymorphism refers to the ability to appear in many forms. Polymorphism is often expressed by the phrase “one interface, multiple methods.”

**Inheritance:** Inheritance is the process by which one object can acquire the properties of another object. It allows programmers to customize a class for a specific purpose, without actually modifying the original class (the superclass).The derived class (subclass) is allowed to add methods or redefine them.

## Overview of Java programing language

In 1991, James Gosling led a team at Sun Microsystems that developed the first version of Java (which was not yet called Java).This first version of the language was designed for programming home appliances, such as washing machines and television sets.Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given the significant points that describe the history of Java.

* Firstly, it was called **"Greentalk"** by James Gosling, and file extension was .gt
* After that, it was called **Oak** and was developed as a part of the Green project.
* In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

## Types of java programming

### There are two kinds of Java programs, applets and applications.

### Java application program:

* Run on your computer like any other program.
* May have a windowing interface or use simple console I/O.

Applet:

* Is meant to be run from a Web browser.
* Can be sent to another location on the Internet and run there.
* always use a windowing interface, but not all programs with a windowing interface are applets
* Can also be run with a program known as an applet viewer.

## Key feature of Java

As stated in the Java language white paper by Sun, Java is simple, object oriented, distributed,  
interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, and  
dynamic.

1. **Java Is Simple:**  No language is simple, but Java is a bit easier than the popular object-oriented programming language C++, which was the dominant software-development language before Java. Java is partially modeled on C++, but greatly simplified and improved. Java uses automatic memory allocation and garbage collection, whereas C++ requires the programmer to allocate memory and collect garbage. Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.
2. **Java Is Object-Oriented**: Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques. Software systems developed using procedural programming languages are based on the paradigm of procedures. Object oriented programming models the real world in terms of objects. Everything in the world can be modeled as an object. A circle is an object, a person is an object, and a Window icon is an object. Even a loan can be perceived as an object. A Java program is object-oriented because programming in Java is centered on creating objects, manipulating objects, and making objects work together. One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.
3. **Java Is Distributed:** Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.
4. **Java Is Interpreted**: You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM). Most compilers, including C++ compilers, translate programs in a high-level language to machine code. The code can only run on the native machine. If you run the program on other machines, it has to be recompiled on the native machine. For instance, if you compile a C++ program in Windows, the executable code generated by the compiler can only run on the Windows platform. With Java, you compile the source code once, and the bytecode generated by a Java compiler can run on any platform with a Java interpreter. The Java interpreter translates the bytecode into the machine language of the target machine.
5. **Java Is Robust**: Robust means reliable. No programming language can ensure complete reliability. Java puts a lot of emphasis on early checking for possible errors. Java compilers can detect many problems that would first show up at execution time in other languages. Java has eliminated certain types of error-prone programming constructs found in other languages. Java has a runtime exception handling feature to provide programming support for robustness.
6. **Java Is Secure**: As an Internet programming language, Java is used in a networked and distributed environment. If you download a Java applet (a special kind of program) and run it on your computer, it will not damage your system because Java implements several security mechanisms to protect your system against harm caused by stray programs. Java implements several security mechanisms to protect your system against harm caused by stray programs
7. **Java Is Architecture-Neutral**: Java is interpreted. This feature enables Java to be *architecture-neutral*, or to use an alternative term, *platform-independent*. With a Java Virtual Machine (JVM), you can write one program that will run on any platform Using Java, developers need to write only one version that can run on every platform. You can write a program once and run it anywhere. Write once, run anywhere With a Java Virtual Machine (JVM), you can write one program that will run on any platform.
8. **Java Is Portable**: Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled. Moreover, there are no platform-specific features in the Java language.
9. **Java's Performance**: Java’s performance Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.
10. **Java Is Multithreaded**: Multithreading is a program’s capability to perform several tasks simultaneously. Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading.
11. **Java Is Dynamic**: Java was designed to adapt to an evolving environment. New code can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

## The Java Language Specification, API, JDK, and IDE

Java syntax is defined in the Java language specification, and the Java library is defined in the Java **API**. The **JDK** is the software for developing and running Java programs. An **IDE** is an integrated development environment for rapidly developing programs. Computer languages have strict rules of usage. If you do not follow the rules when writing a program, the computer will not be able to understand it. The Java language specification and the Java API define the Java standards. The Java language specification is a technical definition of the Java programming language’s syntax and semantics. The application program interface (API), also known as library, contains predefined classes and interfaces for developing Java programs. The API is still expanding.

There are **three editions** of the Java JDK:

* Java 2 Standard Edition(**J2SE**)- **–** can be used to develop client-side standalone application
* Java 2 Enterprise Edition(**J2EE**) **–** can be used to develop server side application such as java servlet , java erver page(JSP) and java sever face(JSF)
* Java 2 Micro Edition (**J2ME**) **–** can be used to develop application for mobile devices like cell phones.

The two principal products in the Java platform are: Java Development Kit (JDK) and Java  
Runtime Environment (JRE).

* The *JDK* is a superset of the JRE, and contains everything that is in the JRE, plus  
  tools such as the compilers and debuggers necessary for developing applets and  
  applications.
  + Java Platform = JDK (because JRE is subset of JDK )
  + JRE = JVM + Java Packages of Classes (like util, math, lang, awt, swing  
    etc) + runtime libraries.
* If you have the JDK installed, you don’t need to install the JRE separately to run  
  any Java software.
* JVM is platform dependent

**Integrated Development Environment (IDE)** is provided by the Java development tool for  
rapidly developing Java programs. Editing, compiling, debugging, and online help are integrated  
in one graphical user interface.

* Just enter source code in one window or open an existing file in a window, then  
  click a button, menu item, or function key to compile and run the program.

**JDK edition**

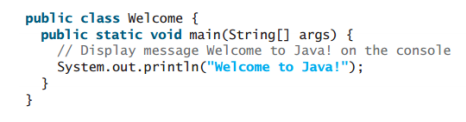
* J2SE, J2EE, J2ME

**Java Development Tools**

Three major development tools are:

* Any text editor, JCreator, Dr java, Intellij, Netbeens, Eclipse , Jedit, jGrasp,blueJ ,**etc…**

## Simple java program



Every Java program must have at least one class. A class is a construct that defines data and  
methods. Each class has a name. By convention, class names start with an uppercase letter.

## Creating compiling and executing java program

The Java programming-development process consists of creating/modifying source code,  
compiling, and executing programs. You can use any text editor (such as notepad) or IDE   
(such as JCreator) to create and edit a Java source-code file. Like any other programming  
language, Java has its own syntax, and you need to write code that obeys the syntax rules.  
The Java compiler will report syntax errors if your program violates the syntax rules. If there  
are no syntax errors, the compiler generates a bytecode file with a .class extension. The bytecode is similar to machine instructions but is architecture-neutral and can run on any platform that has a JVM. This is one of Java's primary advantages: Java bytecode can run on a variety of hardware platforms and operating systems.

### 

### Figure 1.The Java program-development process consists of repeatedly creating/modifying source code, compiling, and executing programs. The Java language is a high-level language, but Java bytecode is a low-level language. The *bytecode* is similar to machine instructions but is architecture neutral and can run on any platform that has a *Java Virtual Machine (JVM)*, as shown in Figure 1.16b. Rather than a physical machine, the virtual machine is a program that interprets Java bytecode. This is one of Java’s primary advantages: *Java bytecode can run on a variety of hardware platforms and operating systems.* Java source code is compiled into Java bytecode and Java bytecode is interpreted by the JVM. Your Java code may use the code in the Java library. The JVM executes your code along with the code in the library. To execute a Java program is to run the program’s bytecode. You can execute the bytecode on any platform with a JVM, which is an interpreter. It translates the individual instructions in the bytecode into the target machine language code one at a time rather than the whole program as a single unit. Each step is executed immediately after it is translated.

### 