

# UNIT - 1

OBJECT ORIENTED PARADIGM

INTRODUCTION TO JAVA

# Object Orientation Concepts

---

## *What is the need to learn about object orientation concepts?*

As you have already seen here: introduction to java, Java is an object oriented programming language. Java follows or is built upon the object oriented paradigm. Generally, every programming language supports at least one of the programming paradigms.

To understand a Java program and its structure, it is essential to know about the object orientation concepts.

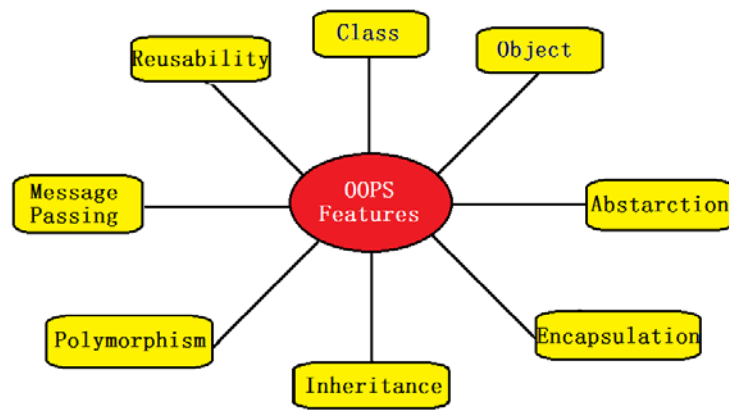
## **Structured Paradigm Vs Object Oriented Paradigm**

In programming languages following structured paradigm or procedural paradigm instructions are written in linear blocks or using functions. We can say that in procedural or structured paradigm, “Code allows access to data”.

In object oriented paradigm, the program is written using objects and classes. A problem can be solved through interactions between objects. We can say that in object oriented paradigm, “Data allows access to code”.

## **Object Oriented Paradigm**

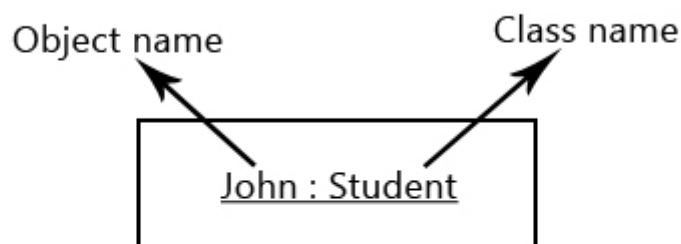
The fundamental unit in object oriented paradigm is an object. Let's look at an object in detail.



**Object:**

Object is a real world thing or entity that is tangible or intangible. An object consists of properties and/or behavior. For example consider a student object whose name is John. Student object contains properties or characteristics like name, age, gender, date of birth etc and behavior like eat, sleep, register, write exam etc.

A student object can be represented on paper or electronically as shown below. I am using the industry standard language UML (Unified Modeling Language) for representing the student object.



Syntax:  
 <Object-name> : <Class-name>

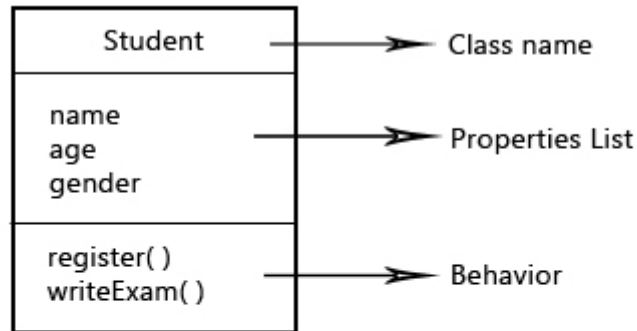
**Class:**

It is common for most of the Java programs to have similar objects. Similar means objects having same properties and behavior. So, a class can be defined as a mold or template for creating objects. A class can also be defined as a collection of similar objects.

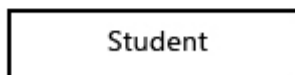


In the above example, the class is the mold (in yellow color) and the object is the paw (blue color). All the objects (blue color) will have the same size, shape etc.

Considering the above student example, let's think of three student objects: John, Mary and Krishna. All of these three students will have similar properties like name, age, gender etc and similar behavior like register, write exam etc. These three objects can be grouped into a class named Student. A class can be represented diagrammatically (using UML) as shown below:



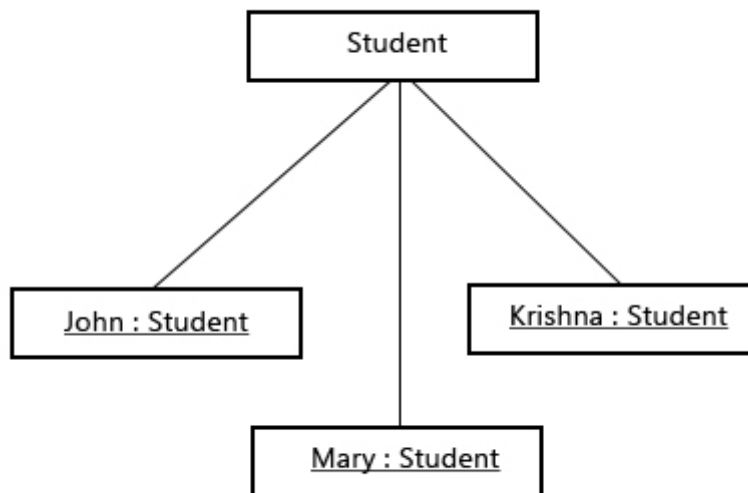
(a) Expanded Form



(b) Abbreviated Form

Startertutorials.com

Now we can represent the class Student and the three objects John, Mary and Krishna of the class Student as shown below:



Startertutorials.com

By now you will be able to distinguish between an object and a class. There is some more terminology you must know. Above, I said that object contains properties and behavior. A property is a characteristic which can be used to describe the object and will generally be a noun. Behavior is defined as a set of operations that can be performed by the object or on the object. Operations are generally verbs.

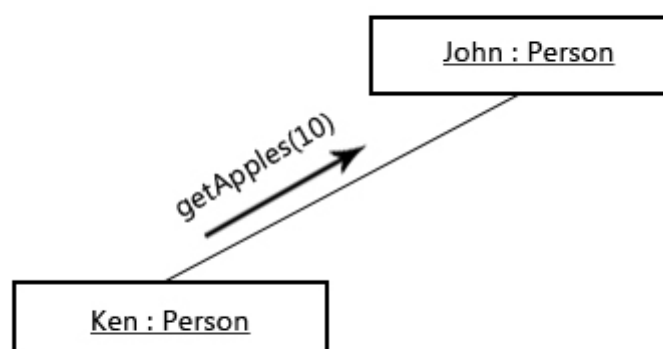
**Object state:**

The set of values assigned to the properties of an object is known as the state of an object. For example we know that for the John object there are three properties: name, age and gender. Then, the state of John object will be John, 21 and male which are the values of name, age and gender respectively.

**Messages:**

In object oriented world objects don't sit idle. They communicate with one other by passing messages. A message is a request to carry out a task. A task can be anything from displaying a message to retrieving data from database and process it.

An object can send messages to those objects only which can process the messages. For example, let's think that an object Ken of Person class wants his dad (John) to purchase apples for him. To carry out that task Ken sends the message `getApples(10)` to John. This can be viewed diagrammatically as shown below:



Startertutorials.com

It is the responsibility of John object to process the message `getApples(10)` and give response back to Ken object. The response message is not mandatory. In the above message `getApples(10)`, 10 is a parameter. A parameter is a mechanism to pass additional information along with the message. In the above example, 10 reflects the number of apples Ken wants.

## Object Orientation Principles

---

Following are the principles of object oriented paradigm (model):

### **Abstraction**

Abstraction is first of the object orientation principles. It is defined as hiding the complex details and presenting only with the relevant or necessary details. Abstraction is used to manage complexity while writing programs. Encapsulation and inheritance are used to implement abstraction while writing programs. Abstraction represents the external behavior of the object (what the object does) and encapsulation represents how it is implemented internally by the object.

As a real world example for abstraction let's consider the car example:



When you want to purchase a car, the car dealer gives you the details of available colors, seating capacity, engine, gearbox, type of steering and other kinds of general details. The car dealer never tells you specific (complex) details of material being used for the hood, nuts and bolts and such.



The above details are not needed. These details are hidden by the car dealer. Like this, you can think of other examples in the real world. Abstraction is everywhere.

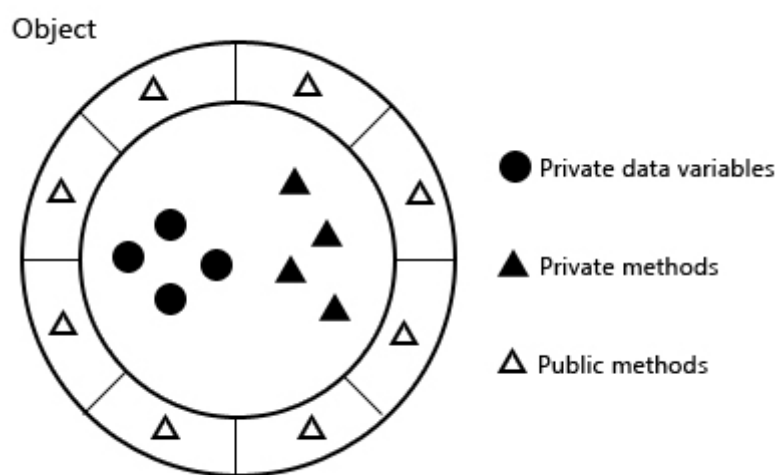
### Encapsulation

Encapsulation is the fundamental principle to achieve abstraction in object oriented programming. Abstraction is conceptual and encapsulation is a way to implement



abstraction. Wherever you can find abstraction you can say that encapsulation is there but encapsulation may not always provide abstraction.

Encapsulation is defined as wrapping up or combining the data and code into a single unit. In the definition data refers to variables and code refers to methods (functions). Every object contains private data (variables) and public methods which allows other code to access the private data. Private methods are optional. We can view an object in OOP as shown below:



Startertutorials.com

As a real world example for encapsulation let's consider a television. We access the television by using a remote which provides an interface to communicate with the television. Television manufacturer is hiding its components (data) under the shell and is providing public interface (remote control) to access its components.



In our real world example, television components are analogous to private member variables and public methods are analogous to buttons on remote control of the television.

Data hiding or information hiding is the side effect of encapsulation. Whenever we use encapsulation, we are implicitly hiding the data of an object which is known as data hiding. Encapsulation helps programmers to write code which is extensible.

Note: Don't worry about variables and methods. I will explain briefly about them in Java basics section.

## **Inheritance**

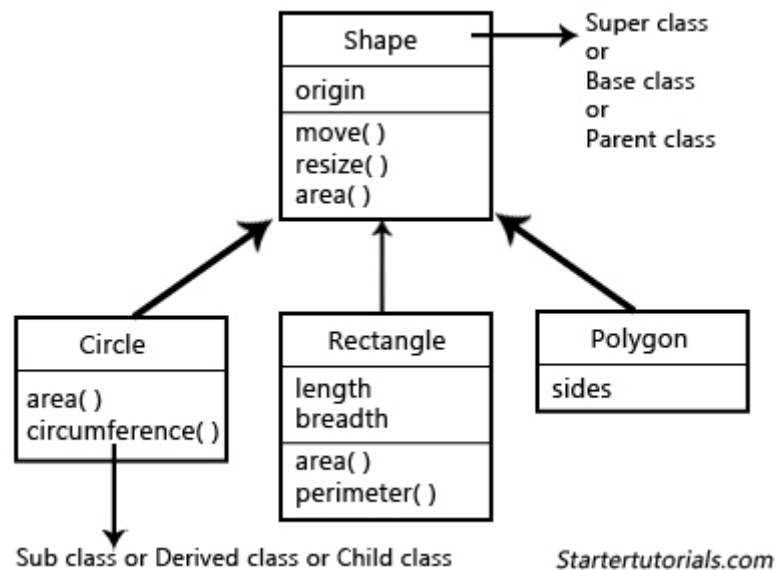
Another way to implement abstraction is by using inheritance. In inheritance, classes are arranged as hierarchies. You might come across situations where even though objects have similar properties are quite different. In such cases, it is better to move all the common properties and behavior into one common class and the specific properties and behavior into special or specific classes.

Inheritance is defined as one object derives or inherits the properties and behavior of another object. As said above common properties and behavior are moved into a common class also known as super class or base class or parent class. Specific properties and behavior are moved into specialized classes also known as sub class or derived class or child class.

As a real world example, consider a child inheriting the properties (land and other assets) from their parents.

As another example, let's consider that we are developing a program to work with geometrical objects like circle, rectangle etc. In the application I can create these objects and

perform operations like move, re size etc. These operations are common for all the objects. So, we can use inheritance in this context. The resultant class hierarchy will be as shown below:



Inheritance promotes re-usability.

## Polymorphism

Polymorphism means many forms. More generally it is defined as one interface multiple implementations. It means that based on the situation or different factors, the same thing exhibits multiple behaviors or the same thing can be used to carry out different tasks. Here thing is used as a general element.

As a real world example for polymorphism, let's consider the animal Chameleon. This animal changes its color to reflect the surroundings to hide from its prey. It is really cool!

Based on the color of the surroundings the same thing (Chameleon) can change its color from one to another.



In every well written java program, we can find encapsulation, inheritance and polymorphism.

# Java Buzzwords

---

1) **Simple:** Java designers built the language in such way that it is very easy to write programs using Java's syntax. It also allows experienced programmers who know C and C++ to move away to Java more quickly since Java's syntax was derived from C and object orientation features were derived from C++. If you already know about object orientation principles it will be much easier to write programs using Java.

2) **Object Oriented:** Java is a object oriented programming language which allows the users to solve the given problem in terms of classes and objects. Writing the solution in terms of classes and objects produces code which is easier to write, maintain and reuse.

3) **Robust:** Errors in Java programs does not crash the virtual machine or hurt the underlying operating system. Several built-in Java features help the users to create reliable programs. Java is a strongly typed language which requires the programmer to declare the type of a variable before using it. Java also has built-in memory management mechanism known as garbage collector. Java also provides exception handling to handle runtime errors. Pointers available in C/C++ are not supported by Java which may lead to memory errors.

4) **Multithreaded:** Java allows us to create interactive programs, which requires threads. Multithreading enables a program to carry out multiple tasks at the same time. For example, consider listening to music and writing something on a piece of paper at the same time. Java provides multiprocess synchronization primitives so that a programmer can be free from the worries of providing complex synchronization solutions.

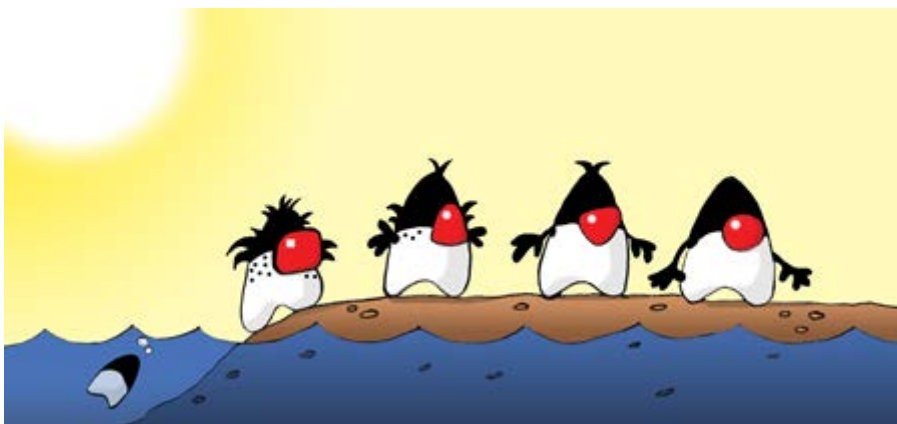
5) **Architecture Neutral:** Java compiler generates bytecode which when run on any machine gives the same output. Java's goal is to enable programmers to develop programs which has WORA (Write Once, Run Anywhere) property.

- 6) Interpreted:** The bytecode generated by the Java compiler goes as input to the Java Virtual Machine (JVM) which is the runtime engine that interprets bytecode into machine code (0's and 1's) line by line. Interpretation speeds up the development process and simplifies debugging process.
- 7) High performance:** Many virtual machines use a JIT (Just-In-Time) compiler that converts the bytecode into platform-specific instructions as the program executes. This helps to improve the performance of the Java programs.
- 8) Distributed:** Java provides libraries for handling TCP/IP, FTP and other network protocols which enables programmers to easily develop distributed applications. Java also provides Remote Method Invocation (RMI) which enables a program to access methods across a network.
- 9) Dynamic:** Java programs maintain non-trivial amount of run-time type information which is used to verify and access objects at run time. This allows to dynamically link small fragments of bytecode during execution of the application.
- 10) Portable:** Portability is achieved through Java's architecture neutrality and through a strict definition of the language. For example, Java's integer primitive type always means a signed 2's complement 32-bit integer. Whereas in C/C++, integer can be unsigned and its size varies according to the system's register size.
- 11) Secure:** Java's sandbox security model available in applets prevents the Java program from accessing the resources of the user's system. This prevents malicious programs to gain access to sensitive information available on the user's system.

# Evolution of Java

---

The latest version of Java till date is Java SE 8 (in short version 8) which was released in March, 2014. The initial version (JDK Alpha and Beta) was released in 1995. Java has made quite a journey from version 1 to 8. Let's look in detail what were the improvements or features added in each version in this article Evolution of Java.



You might think what's with the character in the above picture and why is it here? Well the character that you can see above is Java's mascot Duke.

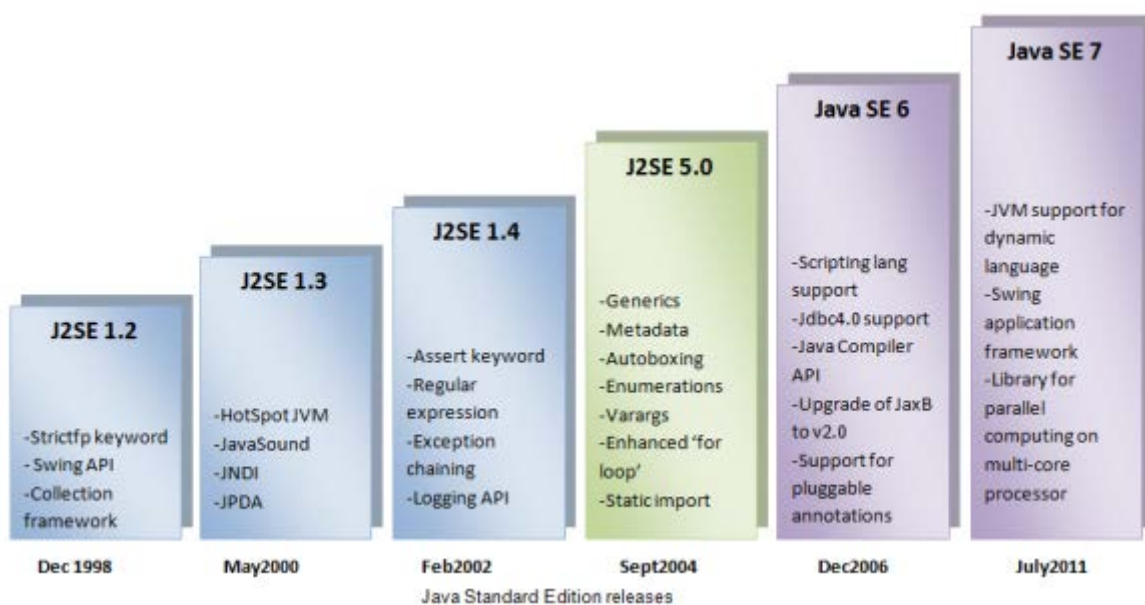
Before moving into the details of Java's versions, let's look at the background details of how and why Java came into existence.

In 1991, a research group at Sun Microsystems, as a part of their Green project, was working to develop software to control consumer electronic devices (embedded systems). The goal was to connect them in a network and establish communication between them. The first prototype developed by Sun Microsystems was a device called Star 7 which looked like a remote. The initial intent was to use C++ to control the Star 7. But, as a result of frustration

with C++, one of the members of Green project, James Gosling invented a new language called Oak which was later renamed to Java.

Why the name Oak? Well, as per rumors, there happened to be a Oak tree outside the window at which James Gosling was working. Hence the name Oak. Later they found out that there was already a language by the name Oak. So, they renamed it to Java.

Now, let's look at the version history which is the core of evolution of Java.



The initial version of Java was **JDK 1.0** which was released in 1996. Unlike other languages which evolve in small increments, Java continued to evolve at a faster pace. By the time JDK 1.0 was released, the developers of Java had already created JDK 1.1.

**JDK 1.1** was released in 1997. Java 1.1 (JDK 1.1) added many new library elements, redefined the way events are handled and reconfigured many of the features of the 1.0 library.

The next major release was **Java 2** (also known as **J2SE 1.2**) which was released in 1998. 1.2 is the internal version number for Java library. With Java 2, Sun repackaged the Java product as J2SE (Java 2 Platform Standard Edition). Major changes in J2SE 1.2 are:



- Introduced Swing framework
- Introduced Collections framework
- Enhancements to JVM and other programming tools.
- Deprecated `suspend()`, `resume()` and `stop()` methods of the `Thread` class.

The first major upgrade made to the original version Java 2 was **J2SE 1.3** which was released in 2000. This version introduced new APIs like JNDI, JPDA and JavaSound. Although this version introduced a small set of changes, they are not very important.

Next major upgrade **J2SE 1.4** which was released in 2002, further enhanced Java. Several important features were added in this release. They are:

- Keyword `assert`
- Chained exceptions
- Channel-based I/O subsystem
- Changes to Collections framework and networking classes
- Logging API

Next release of Java was **J2SE 5** which was released in 2004. Compared to other releases this one was revolutionary. J2SE 5 fundamentally expanded the scope, power and range of the language. The major improvements and features in this release are:

- Generics
- Annotations
- Autoboxing and auto-unboxing

- Enumerations
- Enhanced version of for loop, for-each loop
- Variable length arguments
- Static import
- Formatted I/O
- Concurrency utilities

Next release was **Java SE 6** which was released in 2006. Note that Sun changed the name of the Java platform. The official product name was Java Platform, Standard Edition 6. The Java Developer's kit was called JDK 6. The internal developer version number is 1.6. This version did not add any major features. But, it enhanced the API libraries, added several new packages and made improvements to the runtime.

Next major release was **Java SE 7** (codenamed Dolphin) which was released in 2011 after Sun Microsystems was acquired by Oracle. Java SE 7 contains many new features and enhancements which developed as a part of Project Coin. The new features added in version 7 are as follows:

- A String can now control a switch statement.
- Binary integer literals.
- Underscores in numeric literals.
- An expanded try statement, called try-with-resources, that supports automatic resource management.
- Type inference (using diamond operator) when constructing a generic instance.

- Enhanced exception handling
- Compiler warnings associated with some types of varargs methods have been improved.

Several enhancements were also made in Java SE 7. They are:

Addition of Fork/Join framework for NIO (New I/O) which supports parallel programming.

Upgrade to the Java runtime system that supports non-Java languages.



Next major release is **Java SE 8** (unofficially codenamed as Spider) which was released in 2014. Java 8 is a giant step forward for the Java programming language. A lot of features and enhancements were included. Among them the important ones are:

- Lambda expressions
- Method references
- Default Methods
- A new stream API
- Optional class (java.util package)

- *A new Date/Time API*
- *Nashorn, the new JavaScript engine*
- *Removal of permanent generation (in memory)*

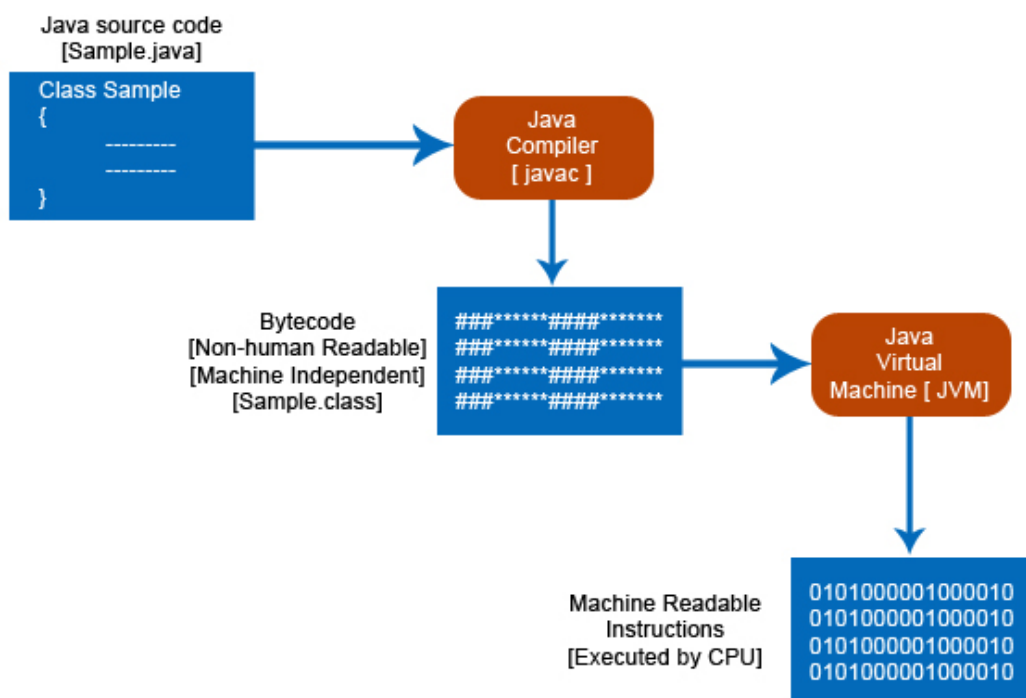
*With the introduction of Lambdas, method references and default methods on interfaces Java incorporates support for functional programming without losing the clarity and simplicity of Java.*

# Life cycle of a Java Program

Life cycle of a java program tells us what happens right from the point when we type source code in a text editor to the point that source code is converted into machine code (0's and 1's).

There are three main stages in the life cycle of a java program. They are:

- Editing the program
- Compiling the source code
- Executing the byte code



Startertutorials.com

First, you will start with typing the program in a text-editor (ex: notepad, notepad++, wordpad, textedit etc). After completing editing of the program, we have to save the file. While saving the file you should remember that the file must be saved with .java extension.

For example, let's think that I had written a Java program which contains a single class `Sample` (more on classes in future posts). It is a good convention to save the file with the name of the class. So, as per my example, the file will be saved as **Sample.java**.

Second step is compilation. The name of the Java compiler is **javac**. The input to the compiler is Java source code which is available in `Sample.java`. The output of the compiler is **machine independent or platform independent code which is known as bytecode**. The file which is generated after compilation is `.class` file. As per my example, the bytecode file will be `Sample.class`.

Command to compile Java program is: **javac filename.java**

Last step is execution. The bytecode generated by the compiler will be executed by Java Virtual Machine (JVM). Input to the JVM is bytecode and output is machine code (0's and 1's) which will be executed by the CPU of the local machine.

Command to execute a Java program is: **java ClassName**

## Creating Compiling and Executing a Java Program

---

First step is to type the Java source code into a text editor and save it with `.java` extension.

Type or copy/paste the below code in your favorite text editor.

```
//Hello World Java Program
class Hello
{
    public static void main(String[] args)
```

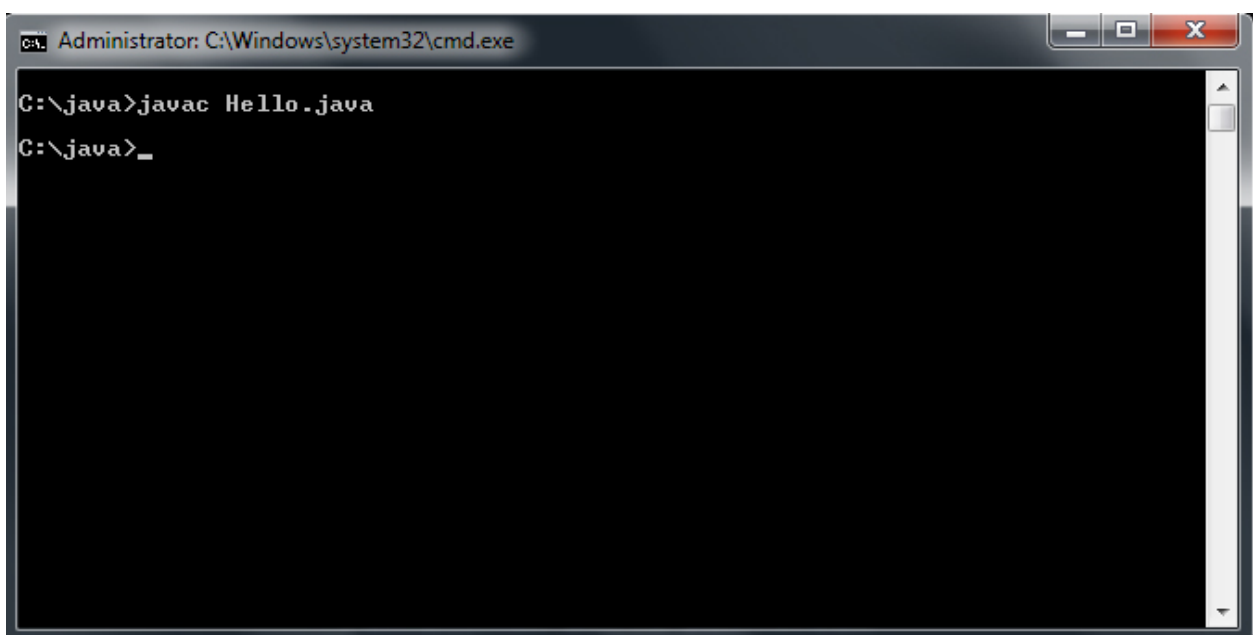
```
{  
    System.out.println("Hello World");  
}  
}
```

Above Java program prints Hello World on to the console (command prompt in Windows). After typing the above program save the file as Hello.java. Note that file name Hello is same as the class name. It is good practice to save the file with the same name as class name but is not mandatory. Let's assume that the file is saved in the path C:/java. So, full path to the source file is C:/java/Hello.java.

Next step is to compile the source code file. Open the command prompt (black window) and navigate to the location C:/java. Type the following command to compile the java file:

```
javac Hello.java
```

If there are no errors in the Java program, the program compiles successfully and you will see the prompt again as shown below:



```
Administrator: C:\Windows\system32\cmd.exe  
C:\>cd java  
C:\java>javac Hello.java  
C:\java>_
```

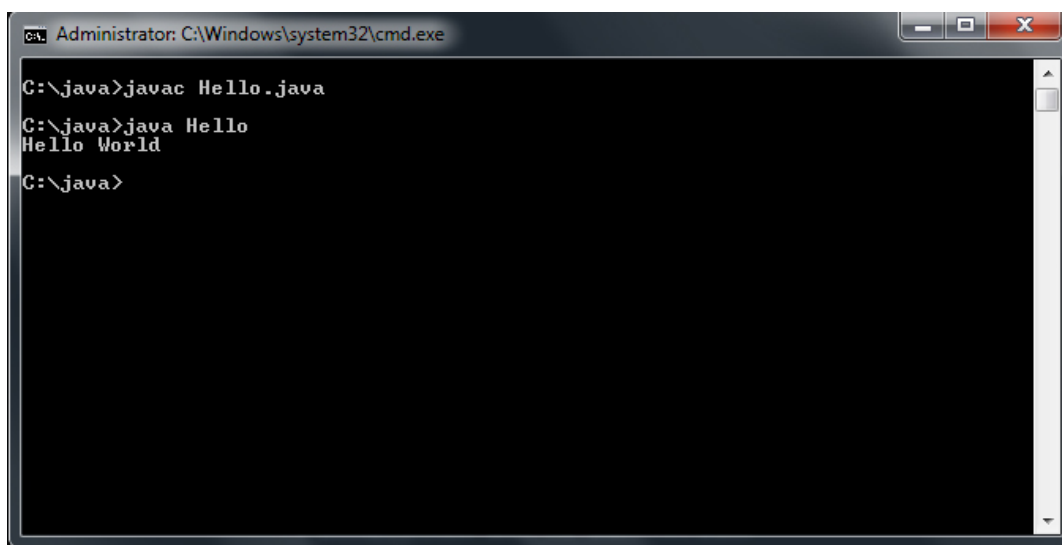
After compiling the program, you can see the generated file `Hello.class` which contains the bytecode in the location `C:/java`.

Next step is to execute the program. To execute the program, type the following command in the command prompt:

```
java Hello
```

Take care that `.class` is not added after the class name `Hello`.

You can see the output `Hello World` as shown below:



```
Administrator: C:\Windows\system32\cmd.exe
C:\java>javac Hello.java
C:\java>java Hello
Hello World
C:\java>
```

Alright! Now that we have executed our first Java program, we will look closer at the Java program to discuss various elements and their use.

## Comments

In Line 1, you can see comments. In Java, we can write comments at any line in the program. There are different types of comments. The one you see in the above program is a single line comment. A single line comment starts with `//` followed by anything you want to write. Comments are useful for writing something which will be useful for the developer or other users. Remember that Java compiler ignores comments.



## **class Keyword**

In Lines 2 to 8, you can see a class declaration. Don't worry too much about classes. They will be explained in detail in a separate post. For now remember this. Every Java program must contain at least one class. A class can be declared in Java by using the class keyword.

## **main Method**

In Lines 4 to 7, you can see the main method. Again, don't worry too much. The main method is an entry point for every Java program. For executing a Java program, we have to write a main method. When JVM executes the bytecode, it searches for the main method and starts the execution from that line.

## **System.out.println**

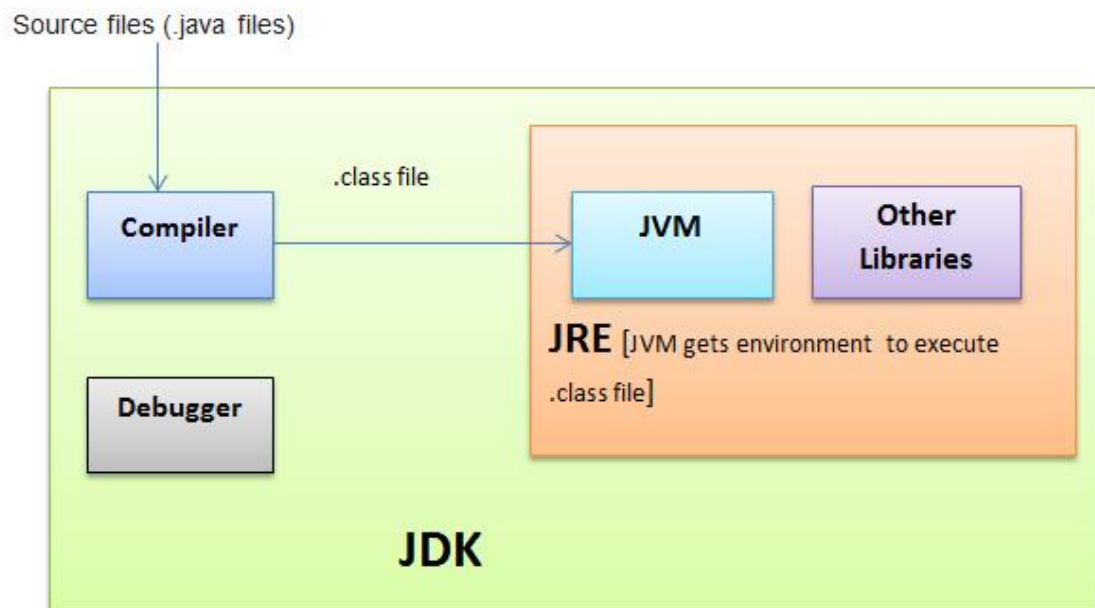
It's normal in a program to do computations, reading data from the user and printing the results. Reading data from the user in the console window (command prompt) is called Standard Input and printing/writing data to the console is called Standard Output. In the above Hello World program, I am printing to the console by using println method which is provided by the out object of PrintStream class and the out object is made accessible by System class. Let's say that whenever we want to print something on to the console, we have to use System.out.println.

System and PrintStream are predefined classes which are provided by Java. All these predefined classes come bundled in packages. A package is a group of related classes. System class is available in java.lang package and PrintStream is available in java.io package.

# Java Virtual Machine

---

Before learning about JVM it is important to know about JDK (Java Development Kit) and JRE (Java Runtime Environment). Below figure shows the relationship between JDK, JRE, and JVM:



JDK provides programmers with a set of tools (like javac, debugger, javap, appletviewer etc..) for developing Java programs. JDK includes JRE.

JRE provides the run-time engine JVM along with the class libraries which contains the predefined functionality.

Using JDK programmers can create and run Java programs. But with JRE alone, programmers or users can only run already compiled Java programs. We cannot create Java programs using only JRE.

Java Virtual Machine (JVM) is an abstract computing machine that allows a computer to run programs written in Java. There are three concepts related to JVM:

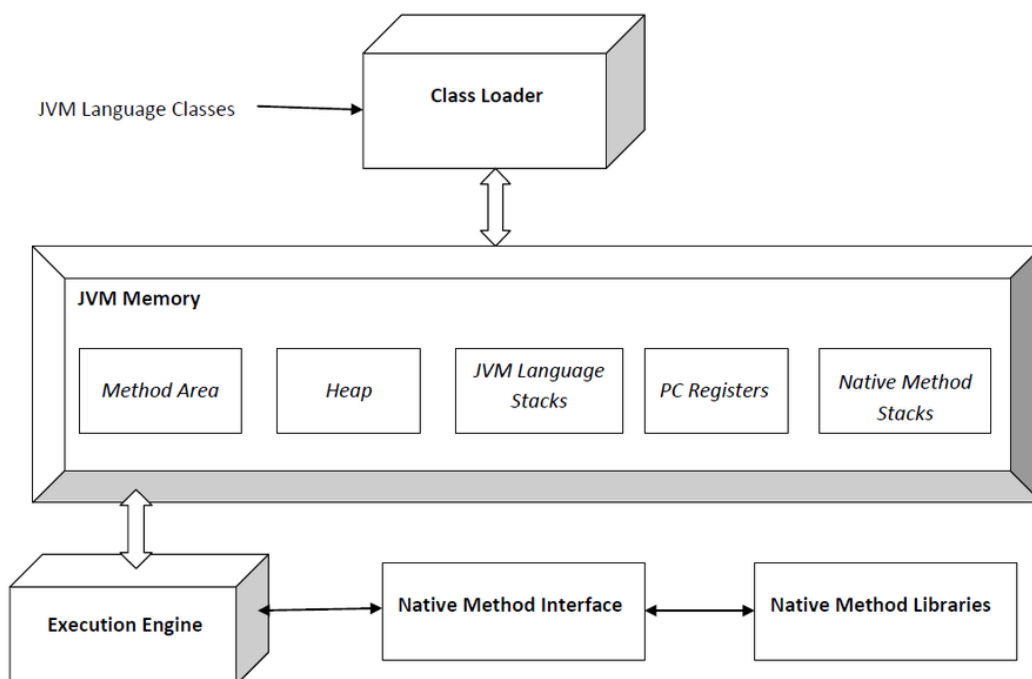
- Specification
- Implementation
- Instance

The JVM specification is a document which contains formal information about what a JVM implementation should contain. A single specification allows various interoperable implementations.

A JVM implementation is a computer program that meets the requirements given in JVM specification.

An instance of a JVM is an implementation running in a process that executes bytecode. Oracle's implementation of JVM specification is known as HotSpot. Other famous implementations are JRockit, Kaffe, IBM J9.

Overview of the architecture of JVM is as shown below:



## Class Loader

A class loader implementation is a program that should be able to perform the following activities:

- Loading: finds and imports the binary data for a type
- Linking: performs verification, preparation, and resolution (optional)
- Initialization: Invokes Java code that initializes class variables to their proper initial values

## Heap

The heap area of JVM is used for dynamic memory allocation. In HotSpot the heap is divided into generations:

- The young generation stores objects whose lifetime is short.
- The old generation stores objects which persist for longer durations.

The permanent generation area stores class definitions and other metadata. This area is removed in Java 8.