

UNIT I

Introduction :

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions :

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

History :

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java

was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Following are given significant points that describe the history of Java.

1) James Gosling

, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.

4) After that, it was called Oak and was developed as a part of the Green project.

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

7) Why had they chose the name Java for Java language? The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called Java one of the Ten Best Products of 1995.

12) JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)

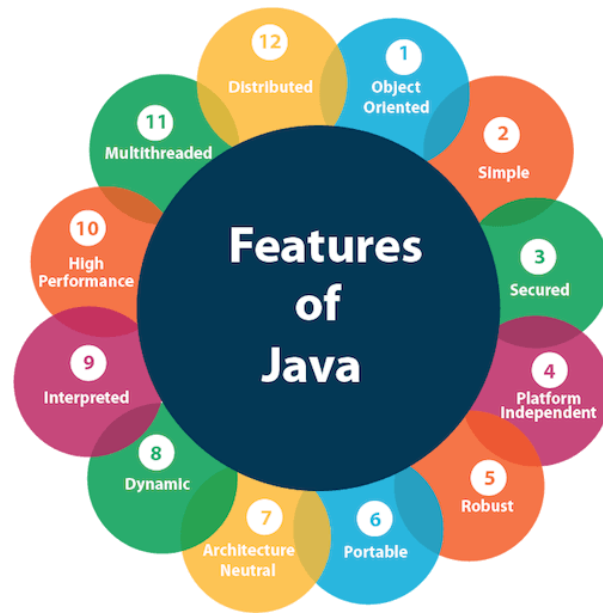
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018)
14. Java SE 12 (March 2019)
15. Java SE 13 (September 2019)
16. Java SE 14 (Mar 2020)
17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (to be released by March 2022)

Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

A list of the most important features of the Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).

- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

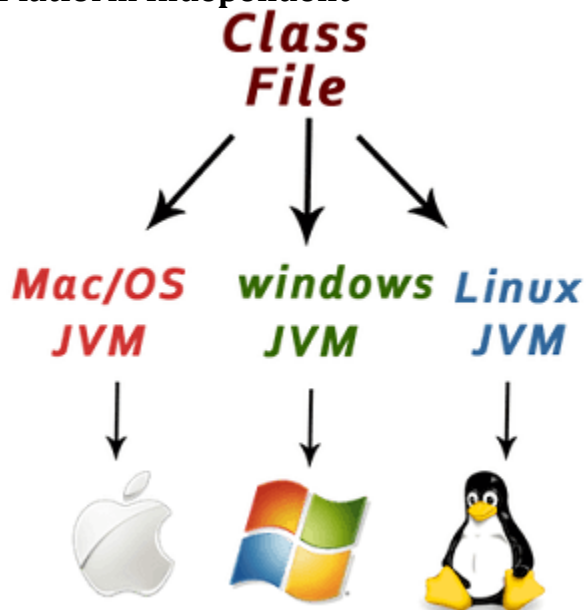
Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

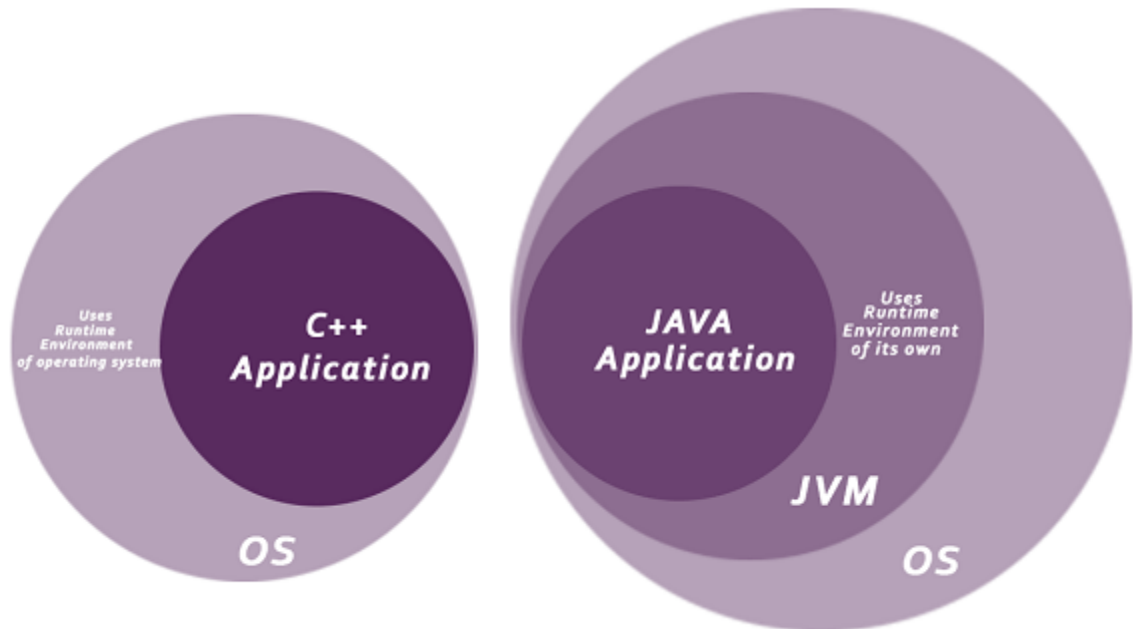
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**



- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

The English meaning of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.

- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

C++ vs Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the <u>C programming language</u> .	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the <u>goto</u> statement.	Java doesn't support the goto statement.

Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using <u>interfaces in java</u> .
Operator Overloading	C++ supports <u>operator overloading</u> .	Java doesn't support operator overloading.
Pointers	C++ supports <u>pointers</u> . You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on	Java has built-in <u>thread</u> support.

	third-party libraries for thread support.	
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (<code>/** ... */</code>) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the <u>inheritance</u> tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language.	Java is also an <u>object-oriented</u> language.

	However, in the C language, a single root hierarchy is not possible.	However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from <code>java.lang.Object</code> .
--	--	--

Difference between JDK, JRE, and JVM

We must understand the differences between JDK, JRE, and JVM before proceeding further to [Java](#). See the brief overview of JVM [here](#).

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the differences between the JDK, JRE, and JVM.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: *specification*, *implementation*, and *instance*.

The JVM performs the following main tasks:

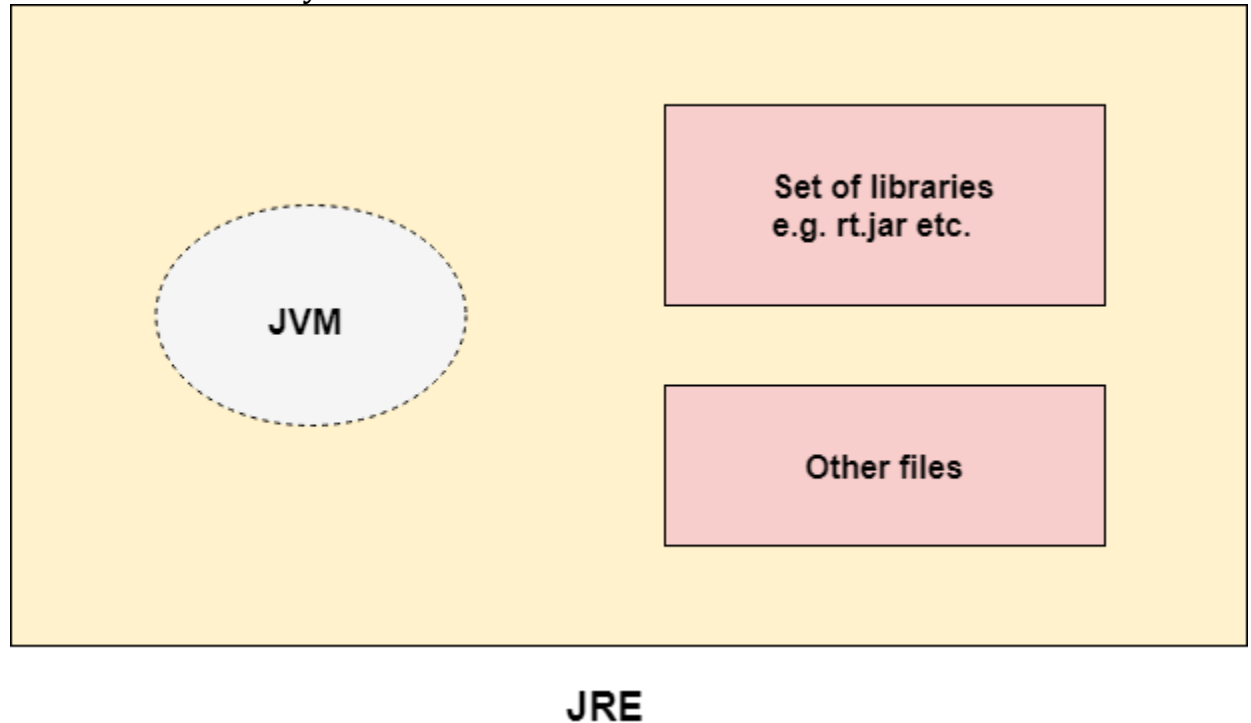
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which

are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JDK

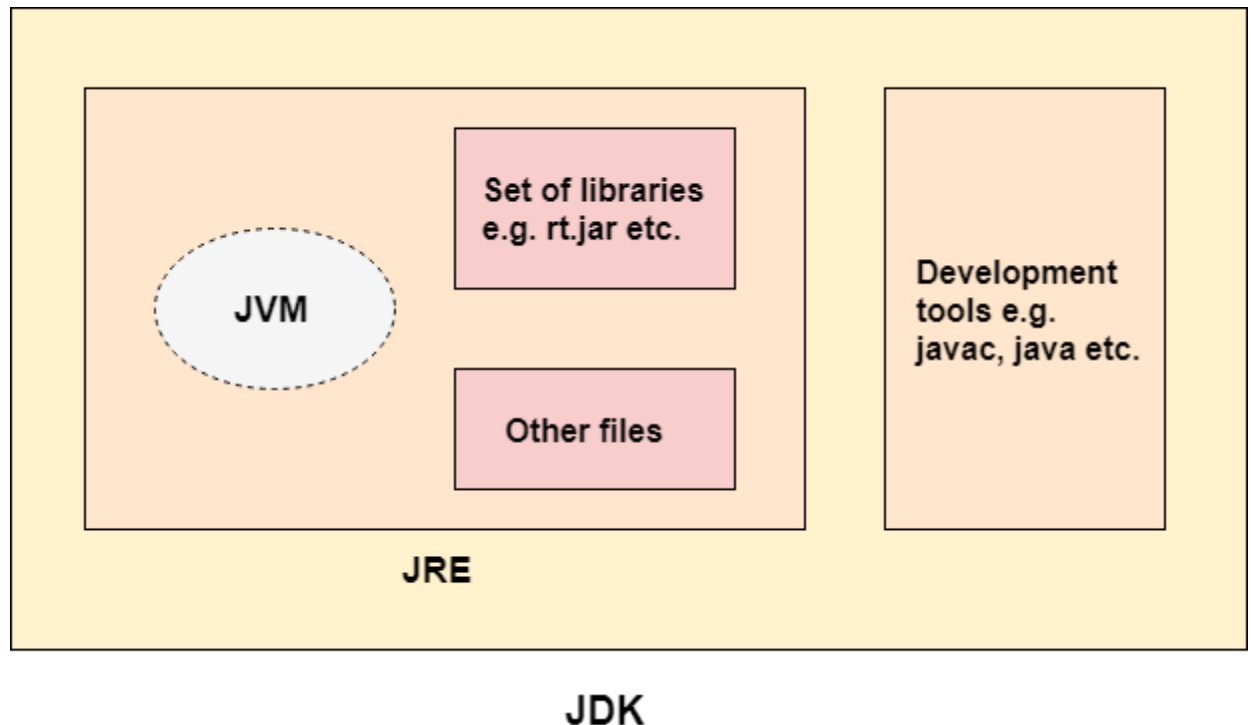
JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an

archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



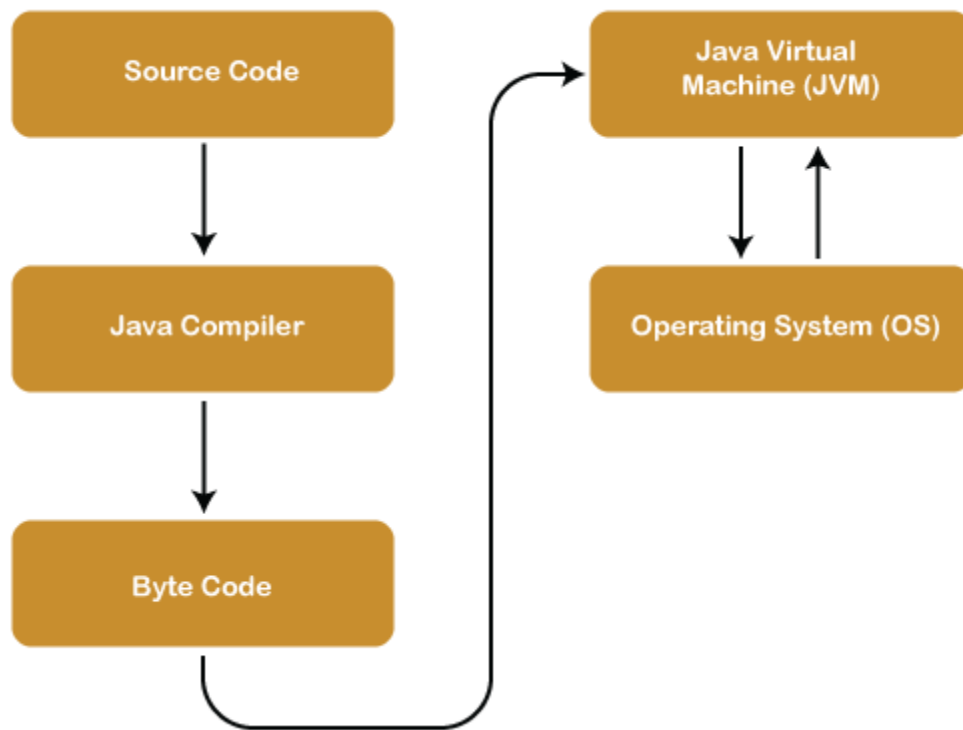
Java Architecture :

Java Architecture is a collection of components, i.e., **JVM**, **JRE**, and **JDK**. It integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. **Java Architecture** explains each and every step of how a program is compiled and executed.

Java Architecture can be explained by using the following steps:

- There is a process of compilation and interpretation in Java.
- Java compiler converts the Java code into byte code.
- After that, the JVM converts the byte code into machine code.
- The machine code is then executed by the machine.

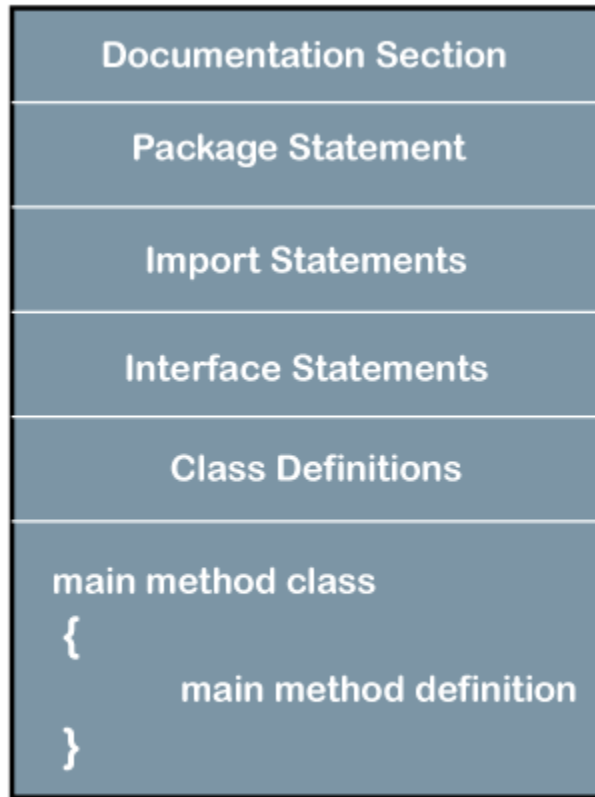
The following figure represents the **Java Architecture** in which each step is elaborate graphically.



- Now let's dive deep to get more knowledge about Java Architecture. As we know that the Java architecture is a collection of components, so we will discuss each and every component into detail.

Structure of Java Program

Java is an object-oriented programming, **platform-independent**, and **secure** programming language that makes it popular. Using the Java programming language, we can develop a wide variety of applications. So, before diving in depth, it is necessary to understand the **basic structure of Java program** in detail. In this section, we have discussed the basic **structure of a Java program**. At the end of this section, you will be able to develop the Hello world Java program, easily.



Structure of Java Program

Let's see which elements are included in the structure of a Java program. A typical structure of a Java program contains the following elements:

- Documentation Section
- Package Declaration
- Import Statements
- Interface Section
- Class Definition
- Class Variables and Variables
- Main Method Class
- Methods and Behaviors

Documentation Section

The documentation section is an important section but optional for a Java program. It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version,**

program name, company name, and description of the program. It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, we use **comments**. The comments may be **single-line, multi-line, and documentation** comments.

- **Single-line Comment:** It starts with a pair of forwarding slash (`//`). For example:

1. `//First Java Program`

- **Multi-line Comment:** It starts with a `/*` and ends with `*/`. We write between these two symbols. For example:

1. `/*It is an example of`

2. `multiline comment*/`

- **Documentation Comment:** It starts with the delimiter (`/**`) and ends with `*/`. For example:

1. `/**It is an example of documentation comment*/`

Package Declaration

The package declaration is optional. It is placed just after the documentation section. In this section, we declare the **package name** in which the class is placed. Note that there can be **only one package** statement in a Java program. It must be defined before any class and interface declaration. It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes package belongs to a single parent directory. We use the keyword **package** to declare the package name. For example:

package javatpoint; `//where javatpoint is the package name`

1. **package** com.javatpoint; `//where com is the root directory and javatpoint is the subdirectory`

Import Statements

The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. The import statement represents the class stored in the other package. We use the **import** keyword to import the class. It is written before the class declaration and after the package statement. We use the

import statement in two ways, either import a specific class or import all classes of a particular package. In a Java program, we can use multiple import statements. For example:

1. **import** java.util.Scanner; //it imports the Scanner class only
2. **import** java.util.*; //it imports all the class of the java.util package

Interface Section

It is an optional section. We can create an **interface** in this section if required. We use the **interface** keyword to create an interface. An interface is a slightly different from the class. It contains only **constants** and **method** declarations. Another difference is that it cannot be instantiated. We can use interface in classes by using the **implements** keyword. An interface can also be used with other interfaces by using the **extends** keyword. For example:

1. **interface** car
2. {
3. **void** start();
4. **void** stop();
5. }

Class Definition

In this section, we define the class. It is **vital** part of a Java program. Without the class, we cannot create any Java program. A Java program may contain more than one class definition. We use the **class** keyword to define the class. The class is a blueprint of a Java program. It contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method. For example:

1. **class** Student //class definition
2. {
3. }

Class Variables and Constants

In this section, we define variables and **constants** that are to be used later in the program. In a Java program, the variables and constants are defined just after the class definition. The variables and constants store values of the parameters. It is used during the execution of the program.

We can also decide and define the scope of variables by using the modifiers. It defines the life of the variables. For example:

1. **class** Student //class definition
2. {
3. String sname; //variable
4. **int** id;
5. **double** percentage;
6. }

Main Method Class

In this section, we define the **main() method**. It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

1. **public static void** main(String args[])
2. {
3. }

For example:

1. **public class** Student //class definition
2. {
3. **public static void** main(String args[])
4. {
5. //statements
6. }
7. }

Sample Java Program:

In this section, we will learn how to write the simple program of Java. We can write a simple hello Java program easily after installing the JDK.

To create a simple Java program, you need to create a class that contains the main method. Let's understand the requirement first.

Creating Hello World Example

Let's create the hello java program:

```
class Simple{  
    public static void main(String args[]){  
        System.out.println("Hello Java");  
    }  
}
```

Save the above file as Simple.java.

To compile: javac Simple.java

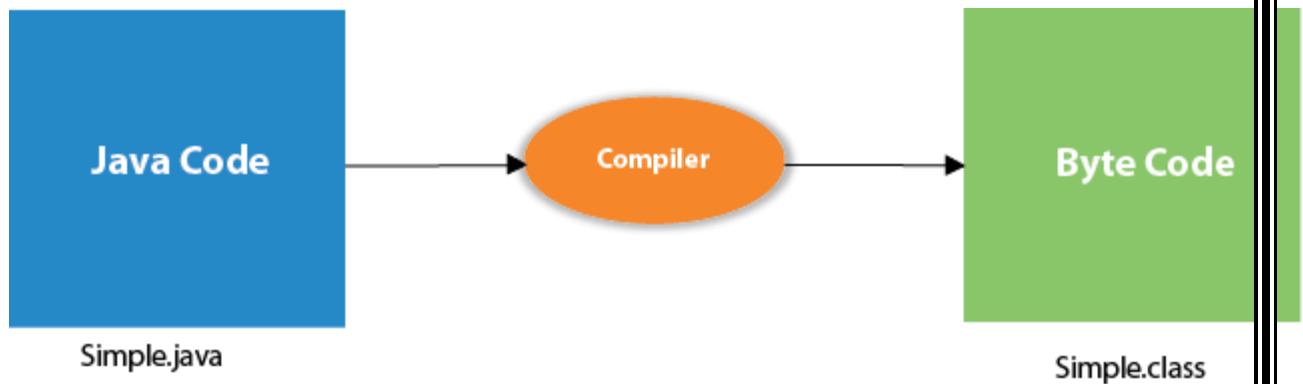
To execute: java Simple

Output:

Hello Java

Compilation Flow:

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.



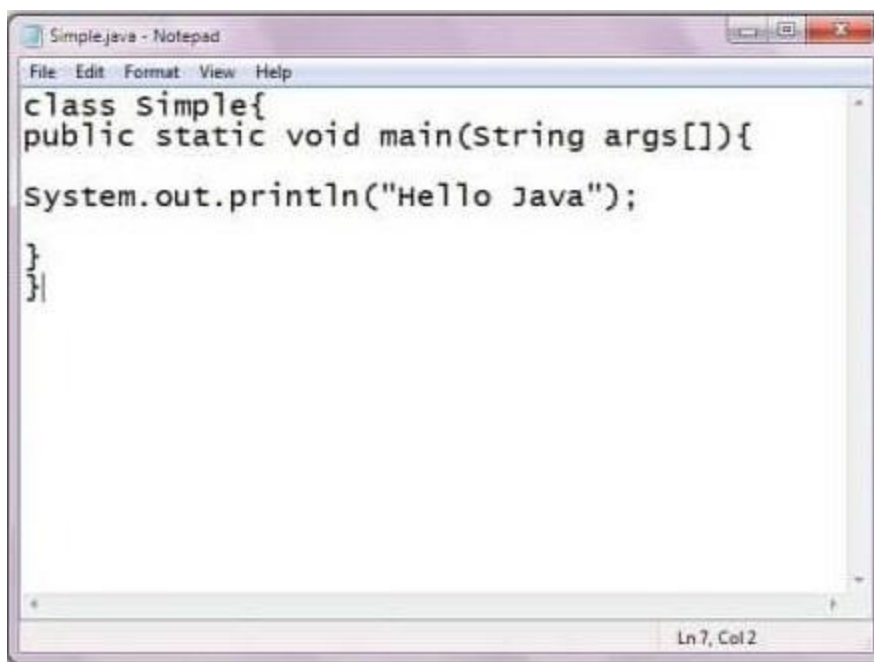
Parameters used in First Java Program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.

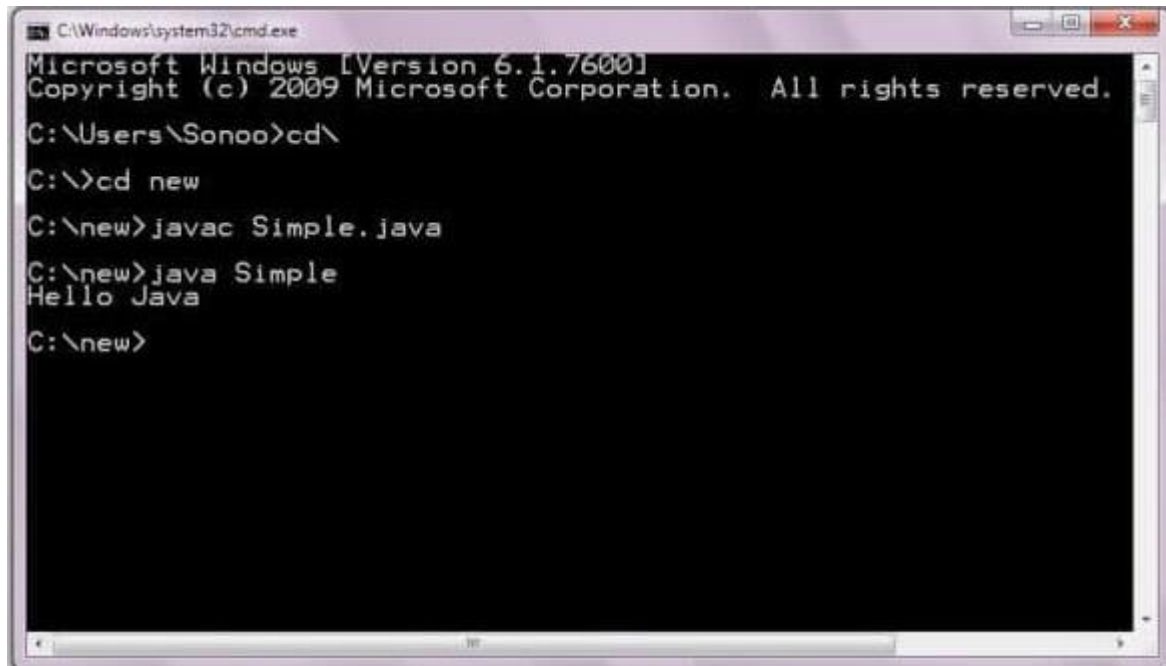
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> Notepad** and write a simple program as we have shown below:



```
Simple.java - Notepad
File Edit Format View Help
class Simple{
public static void main(String args[]){
System.out.println("Hello Java");
}
}
Ln 7, Col 2
```

As displayed in the above diagram, write the simple program of Java in notepad and saved it as Simple.java. In order to compile and run the above program, you need to open the command prompt by **start menu -> All Programs -> Accessories -> command prompt**. When we have done with all the steps properly, it shows the following output:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>cd\
C:\>cd new
C:\new>javac Simple.java
C:\new>java Simple
Hello Java
C:\new>
```

To compile and run the above program, go to your current directory first; my current directory is c:\new. Write here:

To compile: `javac Simple.java`

To execute: `java Simple`

In how many ways we can write a Java program?

There are many ways to write a Java program. The modifications that can be done in a Java program are given below:

1) By changing the sequence of the modifiers, method prototype is not changed in Java.

Let's see the simple code of the main method.

1. **static public void** main(String args[])

2) The subscript notation in the Java array can be used after type, before the variable or after the variable.

Let's see the different codes to write the main method.

1. **public static void** main(String[] args)
2. **public static void** main(String []args)
3. **public static void** main(String args[])

3) You can provide var-args support to the main() method by passing 3 ellipses (dots)

Let's see the simple code of using var-args in the main() method. We will learn about var-args later in the Java New Features chapter.

1. **public static void** main(String... args)

4) Having a semicolon at the end of class is optional in Java.

Let's see the simple code.

1. **class** A{
2. **static public void** main(String... args){
3. System.out.println("hello java4");
4. }
5. };

Valid Java main() method signature

1. **public static void** main(String[] args)
2. **public static void** main(String []args)
3. **public static void** main(String args[])
4. **public static void** main(String... args)
5. **static public void** main(String[] args)
6. **public static final void** main(String[] args)
7. **final public static void** main(String[] args)

Invalid Java main() method signature

1. **public void** main(String[] args)
2. **static void** main(String[] args)
3. **public void static** main(String[] args)
4. **abstract public static void** main(String[] args)

Resolving an error "javac is not recognized as an internal or external command"?

If there occurs a problem like displayed in the below figure, you need to set a path. Since DOS doesn't recognize javac and java as internal or external command. To overcome this problem, we need to set a path. The path is not required in a case where you save your program inside the JDK/bin directory. However, it is an excellent approach to set the path. Click here for [How to set path in java](#).

How to set path in Java

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

There are two ways to set the path in Java:

1. Temporary
2. Permanent

1) How to set the Temporary Path of JDK in Windows

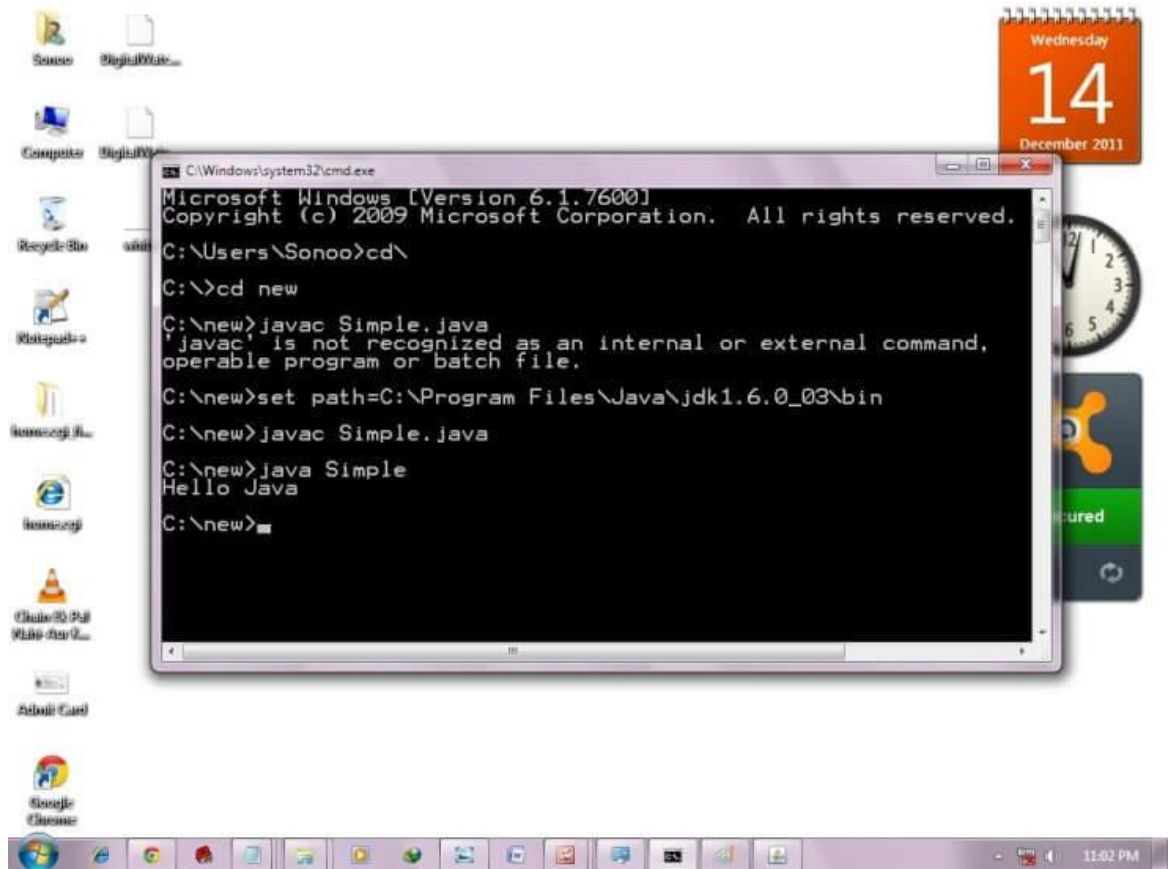
To set the temporary path of JDK, you need to follow the following steps:

- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=copied_path

For Example:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

Let's see it in the figure given below:



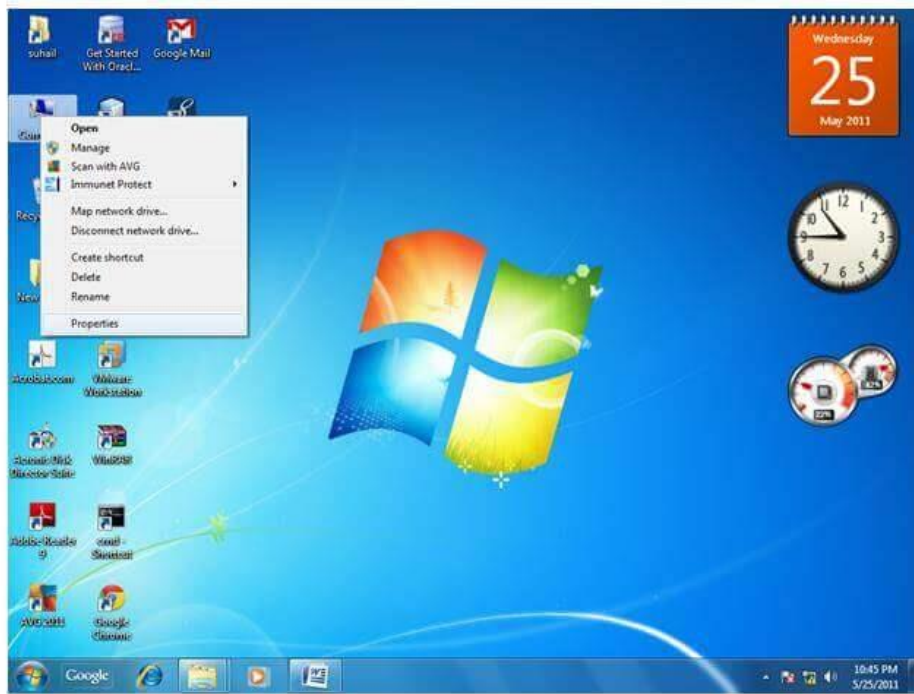
2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

For Example:

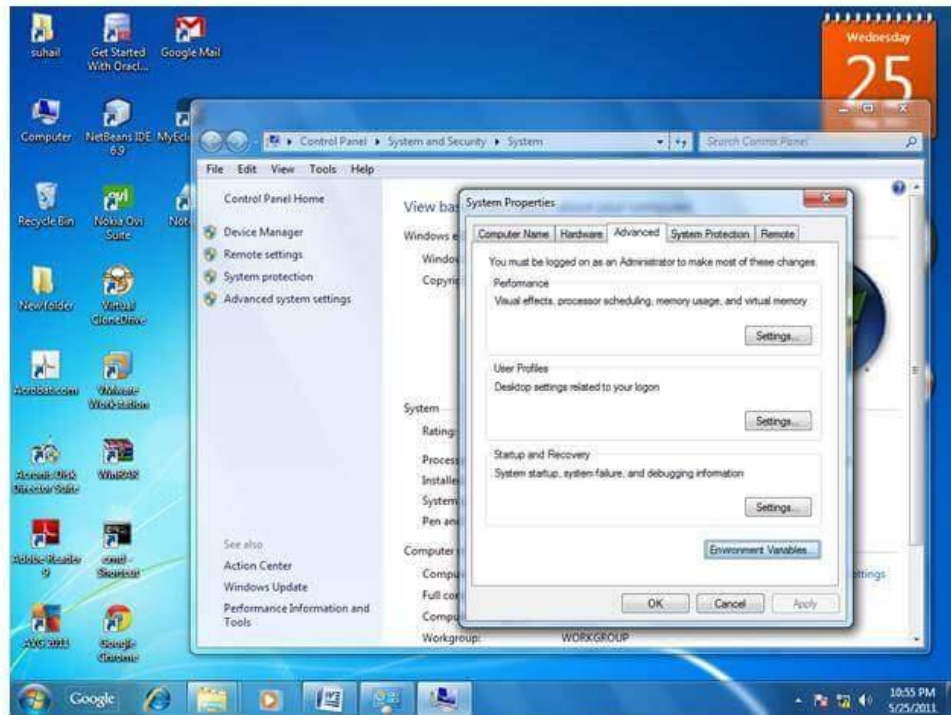
1) Go to MyComputer properties



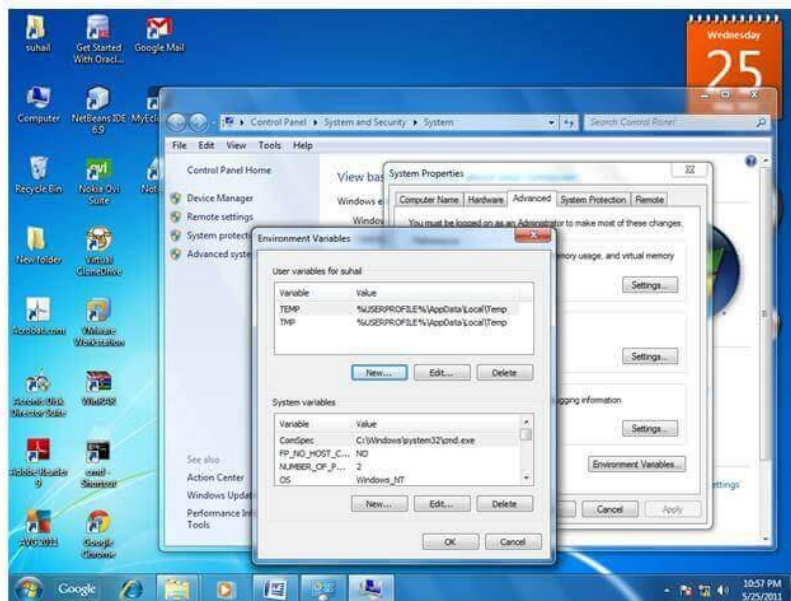
2) Click on the advanced tab



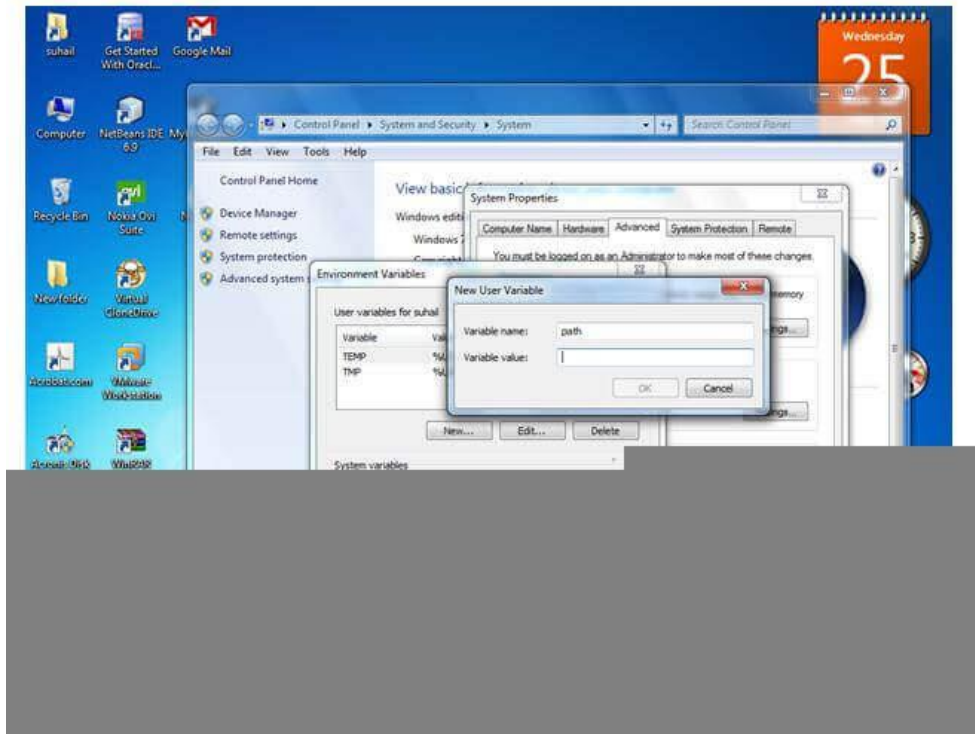
3) Click on environment variables



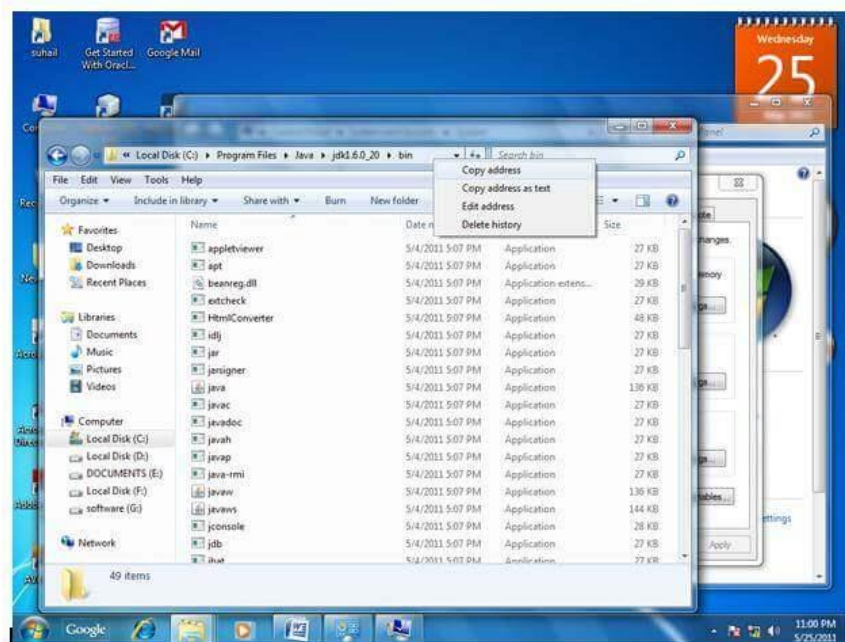
4) Click on the new tab of user variables



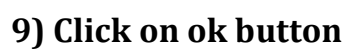
5) Write the path in the variable name

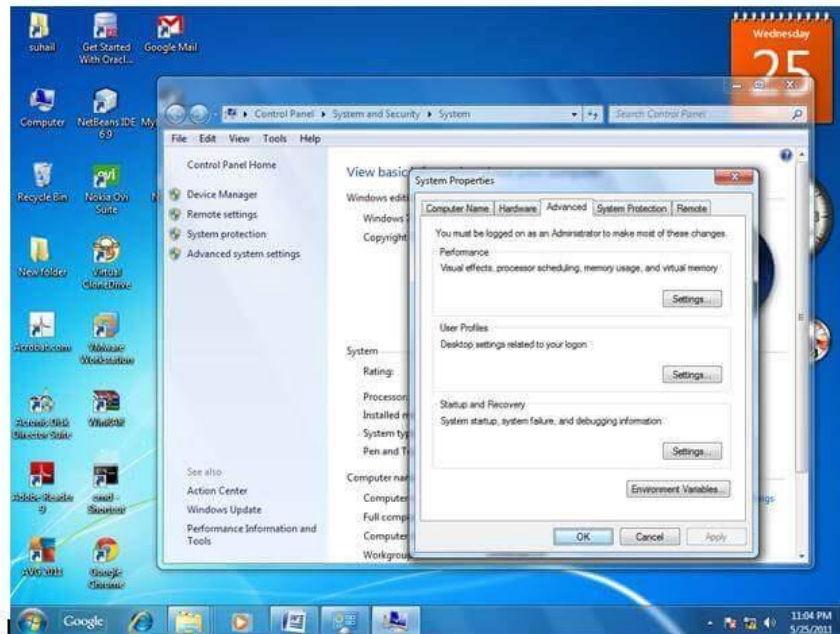


6) Copy the path of bin folder



7) Paste path of bin folder in the variable value





Now your permanent path is set. You can now execute any program of java from any drive.

Classes and Objects

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

Main.java

Create a class named "Main" with a variable x:

```
public class Main {  
    int x = 5;
```



```
}
```

Create an Object

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

Example :

Create an object called "myObj" and print the value of x:

```
public class Main {  
  
    int x = 5;  
  
    public static void main(String[] args) {  
  
        Main myObj = new Main();  
  
        System.out.println(myObj.x);  
  
    }  
}
```

Multiple Objects

You can create multiple objects of one class:

Example :

```
public class Main {  
  
    int x = 5;  
  
  
    public static void main(String[] args) {  
  
        Main myObj1 = new Main(); // Object 1  
  
        Main myObj2 = new Main(); // Object 2  
  
        System.out.println(myObj1.x);  
  
    }  
}
```

```
        System.out.println(myObj2.x);  
    }  
}
```

Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory/folder:

- Main.java
- Second.java

Main.java

```
public class Main {  
    int x = 5;  
}
```

Second.java

```
class Second {  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        System.out.println(myObj.x);  
    }  
}
```

When both files have been compiled:

C:\users\YourName\javac Main.java

C:\users\YourName\javac Second.java

Run the Second.java file:

C:\users\YourName\java Second.java

And the output will be:

5

Class Attributes

we used the term "variable" for x in the example (as shown below). It is actually an **attribute** of the class. Or you could say that class attributes are variables within a class:

Example :

Create a class called "Main" with two attributes: x and y:

```
public class Main {  
    int x = 5;  
    int y = 3;  
}
```

Another term for class attributes is **fields**.

Accessing Attributes

You can access attributes by creating an object of the class, and by using the dot syntax (.):

The following example will create an object of the Main class, with the name myObj. We use the x attribute on the object to print its value:

Example :

Create an object called "myObj" and print the value of x:

```
public class Main {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();
```

```
        System.out.println(myObj.x);  
    }  
}
```

Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

Example :

```
public class Main {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Main myObj1 = new Main(); // Object 1  
        Main myObj2 = new Main(); // Object 2  
        myObj2.x = 25;  
        System.out.println(myObj1.x); // Outputs 5  
        System.out.println(myObj2.x); // Outputs 25  
    }  
}
```

Multiple Attributes

You can specify as many attributes as you want:

```
public class Main {  
    String fname = "John";  
    String lname = "Doe";  
    int age = 24;
```

```
public static void main(String[] args) {  
    Main myObj = new Main();  
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);  
    System.out.println("Age: " + myObj.age);  
}  
}
```

Using Multiple Classes

Like we specified in the [Classes chapter](#), it is a good practice to create an object of a class and access it in another class.

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory:

- Main.java
- Second.java

Main.java

```
public class Main {  
    public void fullThrottle() {  
        System.out.println("The car is going as fast as it can!");  
    }  
  
    public void speed(int maxSpeed) {  
        System.out.println("Max speed is: " + maxSpeed);  
    }  
}
```

Second.java

```
class Second {
```

```
public static void main(String[] args) {  
    Main myCar = new Main();    // Create a myCar object  
    myCar.fullThrottle();    // Call the fullThrottle() method  
    myCar.speed(200);    // Call the speed() method  
}  
}
```

When both files have been compiled:

C:\users\YourName\javac Main.java

C:\users\YourName\javac Second.java

Run the Second.java file:

C:\users\YourName\java Second.java

And the output will be:

The car is going as fast as it can!

Max Speed is 200

Constructor :

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

Example :

```
public class Main {  
    int x; // Create a class attribute  
  
    // Create a class constructor for the Main class  
    public Main() {
```

```
x = 5; // Set the initial value for the class attribute x
}

public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main (This
    will call the constructor)
    System.out.println(myObj.x); // Print the value of x
}
}
```

// Outputs 5

Constructor Parameters

Constructors can also take parameters, which is used to initialize attributes.

The following example adds an **int y** parameter to the constructor. Inside the constructor we set x to y (x=y). When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:

Example :

```
public class Main {
    int x;

    public Main(int y) {
        x = y;
    }
}
```

```
}

public static void main(String[] args) {

    Main myObj = new Main(5);

    System.out.println(myObj.x);

}

}
```

// Outputs 5

You can have as many parameters as you want:

Example :

```
public class Main {

    int modelYear;

    String modelName;

    public Main(int year, String name) {

        modelYear = year;

        modelName = name;

    }

    public static void main(String[] args) {

        Main myCar = new Main(1969, "Mustang");

        System.out.println(myCar.modelYear + " " + myCar.modelName);

    }

}
```



```
}  
}
```

```
// Outputs 1969 Mustang
```

Example :

```
public class Main {  
    int modelYear;  
    String modelName;  
  
    public Main(int year, String name) {  
        modelYear = year;  
        modelName = name;  
    }  
  
    public static void main(String[] args) {  
        Main myCar = new Main(1969, "Mustang");  
        System.out.println(myCar.modelYear + " " + myCar.modelName);  
    }  
}
```

```
// Outputs 1969 Mustang
```

Method Overloading

With **method overloading**, multiple methods can have the same name with different parameters:

Example :

```
int myMethod(int x)
```

```
float myMethod(float x)
```

```
double myMethod(double x, double y)
```

Consider the following example, which has two methods that add numbers of different type:

Example :

```
static int plusMethodInt(int x, int y) {  
    return x + y;  
}
```

```
static double plusMethodDouble(double x, double y) {  
    return x + y;  
}
```

```
public static void main(String[] args) {  
    int myNum1 = plusMethodInt(8, 5);  
    double myNum2 = plusMethodDouble(4.3, 6.26);  
    System.out.println("int: " + myNum1);  
    System.out.println("double: " + myNum2);  
}
```

Instead of defining two methods that should do the same thing, it is better to overload one.

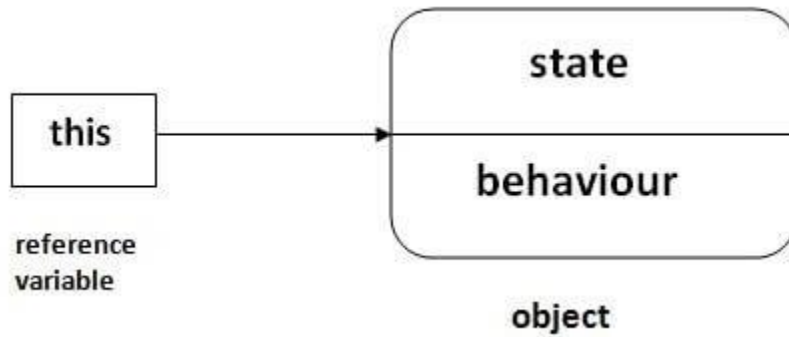
In the example below, we overload the `plusMethod` method to work for both `int` and `double`:

Example :

```
static int plusMethod(int x, int y) {  
    return x + y;  
}  
  
static double plusMethod(double x, double y) {  
    return x + y;  
}  
  
public static void main(String[] args) {  
    int myNum1 = plusMethod(8, 5);  
    double myNum2 = plusMethod(4.3, 6.26);  
    System.out.println("int: " + myNum1);  
    System.out.println("double: " + myNum2);  
}
```

this keyword

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



Usage of Java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Suggestion: If you are beginner to java, lookup only three usages of this keyword.

Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01

this can be used to refer current class instance variable.

04

this can be passed as an argument in the method call.

02

this can be used to invoke current class method (implicitly)

05

this can be passed as argument in the constructor call.

03

this() can be used to invoke current class Constructor.

06

this can be used to return the current class instance from the method

1) **this**: to refer current class instance variable

The **this** keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use **this** keyword by the example given below:

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** rollno,String name,**float** fee){
6. rollno=rollno;
7. name=name;
8. fee=fee;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}

```
11.}
12.class TestThis1{
13.public static void main(String args[]){
14.Student s1=new Student(111,"ankit",5000f);
15.Student s2=new Student(112,"sumit",6000f);
16.s1.display();
17.s2.display();
18.}}
```

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
1. class Student{
2.   int rollno;
3.   String name;
4.   float fee;
5.   Student(int rollno,String name,float fee){
6.     this.rollno=rollno;
7.     this.name=name;
8.     this.fee=fee;
9.   }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11.}
12.
13.class TestThis2{
14. public static void main(String args[]){
15. Student s1=new Student(111,"ankit",5000f);
16. Student s2=new Student(112,"sumit",6000f);
17. s1.display();
```

```
18.s2.display();
19.}}
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
1. class Student{
2.   int rollno;
3.   String name;
4.   float fee;
5.   Student(int r,String n,float f){
6.     rollno=r;
7.     name=n;
8.     fee=f;
9.   }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11.}
12.
13. class TestThis3{
14. public static void main(String args[]){
15. Student s1=new Student(111,"ankit",5000f);
16. Student s2=new Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19.}}
```

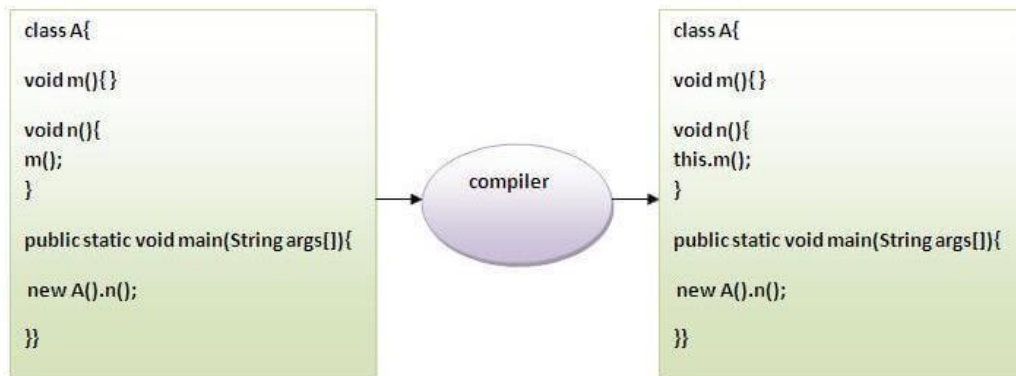
Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example



1. **class** A{
2. **void** m(){System.out.println("hello m");}
3. **void** n(){
4. System.out.println("hello n");
5. //m();//same as this.m()
6. **this**.m();
7. }
8. }
9. **class** TestThis4{
- 10.**public static void** main(String args[]){
- 11.A a=**new** A();
- 12.a.n();
- 13.}}

Output:

hello n

hello m

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
1. class A{
2. A(){System.out.println("hello a");}
3. A(int x){
4. this();
5. System.out.println(x);
6. }
7. }
8. class TestThis5{
9. public static void main(String args[]){
10.A a=new A(10);
11.}}
```

Output:

```
hello a
10
```

Calling parameterized constructor from default constructor:

```
1. class A{
2. A(){
3. this(5);
4. System.out.println("hello a");
5. }
6. A(int x){
7. System.out.println(x);
8. }
```

```
9. }
10. class TestThis6{
11. public static void main(String args[]){
12. A a=new A();
13. }}
```

Output:

```
5
hello a
```

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
1. class Student{
2. int rollno;
3. String name,course;
4. float fee;
5. Student(int rollno,String name,String course){
6. this.rollno=rollno;
7. this.name=name;
8. this.course=course;
9. }
10. Student(int rollno,String name,String course,float fee){
11. this(rollno,name,course); //reusing constructor
12. this.fee=fee;
13. }
14. void display(){System.out.println(rollno+" "+name+" "+course+" "+fee
    );}
15. }
16. class TestThis7{
17. public static void main(String args[]){
```

```
18.Student s1=new Student(111,"ankit","java");
19.Student s2=new Student(112,"sumit","java",6000f);
20.s1.display();
21.s2.display();
22.}}
```

Output:

```
111 ankit java 0.0
112 sumit java 6000.0
```

Rule: *Call to this() must be the first statement in constructor.*

```
1. class Student{
2.   int rollno;
3.   String name,course;
4.   float fee;
5.   Student(int rollno,String name,String course){
6.     this.rollno=rollno;
7.     this.name=name;
8.     this.course=course;
9.   }
10.  Student(int rollno,String name,String course,float fee){
11.    this.fee=fee;
12.    this(rollno,name,course);//C.T.Error
13.  }
14.  void display(){System.out.println(rollno+" "+name+" "+course+" "+fee
    );}
15. }
16. class TestThis8{
17.  public static void main(String args[]){
18.    Student s1=new Student(111,"ankit","java");
19.    Student s2=new Student(112,"sumit","java",6000f);
20.    s1.display();
21.    s2.display();
22.  }}
```

Output:

Compile Time Error: Call to this must be first statement in constructor

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
1. class S2{
2.     void m(S2 obj){
3.         System.out.println("method is invoked");
4.     }
5.     void p(){
6.         m(this);
7.     }
8.     public static void main(String args[]){
9.         S2 s1 = new S2();
10.        s1.p();
11.    }
12.}
```

Output:

method is invoked

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
1. class B{
2.     A4 obj;
3.     B(A4 obj){
4.         this.obj=obj;
```

```

5.  }
6.  void display(){
7.    System.out.println(obj.data); //using data member of A4 class
8.  }
9.  }
10.
11. class A4{
12.   int data=10;
13.   A4(){
14.     B b=new B(this);
15.     b.display();
16.   }
17.   public static void main(String args[]){
18.     A4 a=new A4();
19.   }
20.}

```

Output:10

6) this keyword can be used to return current class instance

We can return this keyword as a statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```

1. return_type method_name(){
2.   return this;
3. }

```

Example of this keyword that you return as a statement from the method

```

1. class A{
2.   A getA(){
3.     return this;
4.   }
5.   void msg(){System.out.println("Hello java");}

```

```
6. }
7. class Test1{
8. public static void main(String args[]){
9. new A().getA().msg();
10.}
11.}
```

Output:

Hello java

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
1. class A5{
2. void m(){
3. System.out.println(this); //prints same reference ID
4. }
5. public static void main(String args[]){
6. A5 obj=new A5();
7. System.out.println(obj); //prints the reference ID
8. obj.m();
9. }
10.}
```

Output:

A5@22b3ea59
A5@22b3ea59

Scope :

In Java, variables are only accessible inside the region they are created. This is called **scope**.

Method Scope

Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared:

Example :

```
public class Main {  
    public static void main(String[] args) {  
        // Code here CANNOT use x  
        int x = 100;  
  
        // Code here can use x  
        System.out.println(x);  
    }  
}
```

Block Scope

A block of code refers to all of the code between curly braces {}.

Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared:

Example :

```
public class Main {  
    public static void main(String[] args) {  
        // Code here CANNOT use x  
        { // This is a block  
            // Code here CANNOT use x  
        }  
    }  
}
```

```
int x = 100;

// Code here CAN use x

System.out.println(x);

} // The block ends here

// Code here CANNOT use x

}

}
```

Static Keyword :

The **static keyword** in [Java](#) is used for memory management mainly. We can apply static keyword with [variables](#), methods, blocks and [nested classes](#). The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

Understanding the problem without static variable

```
1. class Student{
2.     int rollno;
3.     String name;
4.     String college="ITS";
5. }
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Java static property is shared to all objects.

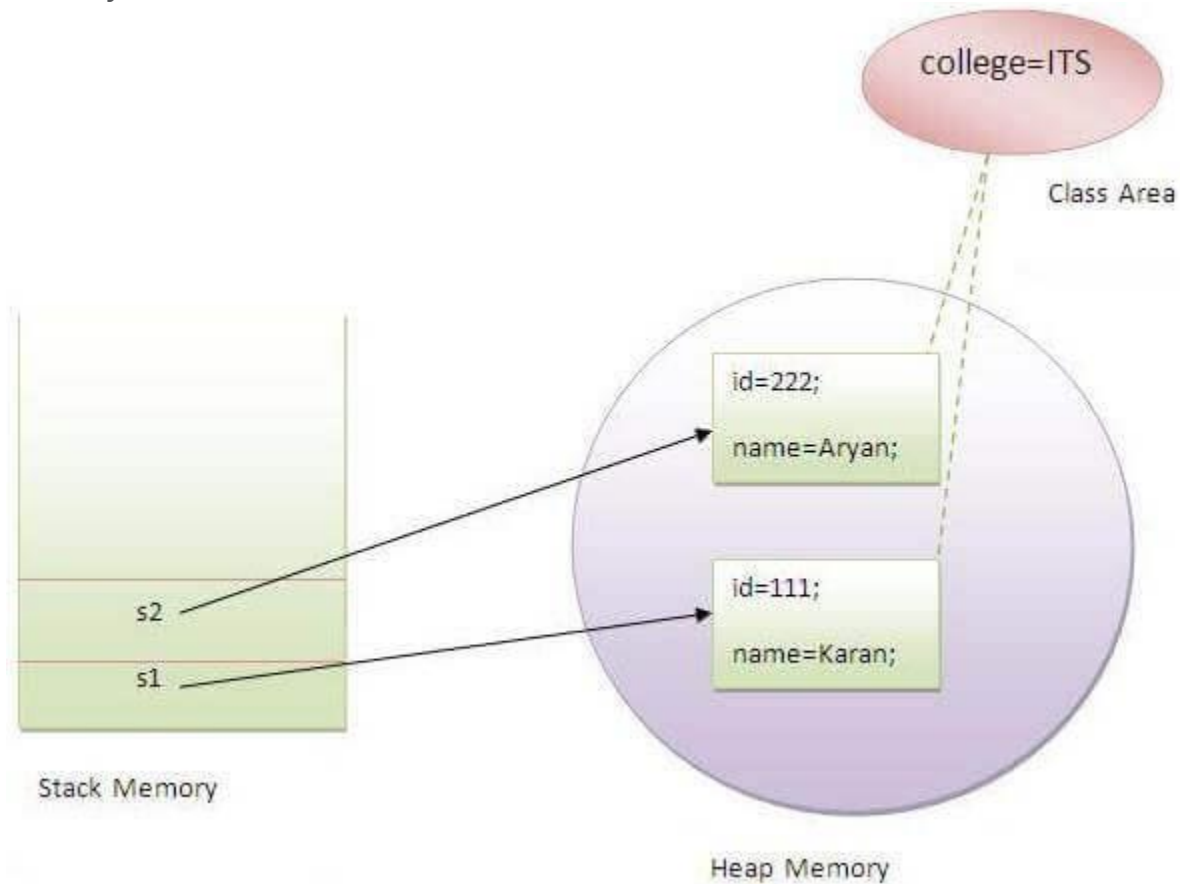
Example of static variable

```
1. //Java Program to demonstrate the use of static variable
2. class Student{
3.     int rollno;//instance variable
4.     String name;
5.     static String college ="ITS";//static variable
6.     //constructor
7.     Student(int r, String n){
8.         rollno = r;
9.         name = n;
10.    }
11.    //method to display the values
12.    void display (){System.out.println(rollno+" "+name+" "+college);}
13.}
14.//Test class to show the values of objects
15. public class TestStaticVariable1{
16.     public static void main(String args[]){
17.         Student s1 = new Student(111,"Karan");
18.         Student s2 = new Student(222,"Aryan");
19.         //we can change the college of all objects by the single line of code
20.         //Student.college="BBDIT";
21.         s1.display();
```

```
22. s2.display();  
23. }  
24. }
```

Output:

```
111 Karan ITS  
222 Aryan ITS
```



Program of the counter without static variable

In this example, we have created an instance variable named `count` which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect other objects. So each object will have the value 1 in the `count` variable.

1. `//Java Program to demonstrate the use of an instance variable`

```
2. //which get memory each time when we create an object of the class.
3. class Counter{
4. int count=0;//will get memory each time when the instance is created
5.
6. Counter(){
7. count++;//incrementing value
8. System.out.println(count);
9. }
10.
11. public static void main(String args[]){
12. //Creating objects
13. Counter c1=new Counter();
14. Counter c2=new Counter();
15. Counter c3=new Counter();
16. }
17. }
```

Output:

```
1
1
1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
1. //Java Program to illustrate the use of static variable which
2. //is shared with all objects.
3. class Counter2{
4. static int count=0;//will get memory only once and retain its value
5.
6. Counter2(){
7. count++;//incrementing the value of static variable
8. System.out.println(count);
```

```
9. }  
10.  
11. public static void main(String args[]){  
12. //creating objects  
13. Counter2 c1=new Counter2();  
14. Counter2 c2=new Counter2();  
15. Counter2 c3=new Counter2();  
16.}  
17.}
```

Output:

```
1  
2  
3
```

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example of static method

```
1. //Java Program to demonstrate the use of a static method.  
2. class Student{  
3.   int rollno;  
4.   String name;  
5.   static String college = "ITS";  
6.   //static method to change the value of static variable  
7.   static void change(){  
8.     college = "BBDIT";  
9.   }  
10. //constructor to initialize the variable
```

```

11. Student(int r, String n){
12.     rollno = r;
13.     name = n;
14. }
15. //method to display values
16. void display(){System.out.println(rollno+" "+name+" "+college);}
17.}
18.//Test class to create and display the values of object
19.public class TestStaticMethod{
20.     public static void main(String args[]){
21.         Student.change();//calling change method
22.         //creating objects
23.         Student s1 = new Student(111,"Karan");
24.         Student s2 = new Student(222,"Aryan");
25.         Student s3 = new Student(333,"Sonoo");
26.         //calling display method
27.         s1.display();
28.         s2.display();
29.         s3.display();
30.     }
31.}

```

Output:111 Karan BBDIT
 222 Aryan BBDIT
 333 Sonoo BBDIT

Another example of a static method that performs a normal calculation

```

1. //Java Program to get the cube of a given number using the static method
2.
3. class Calculate{
4.     static int cube(int x){
5.         return x*x*x;
6.     }
7.
8.     public static void main(String args[]){

```

```
9.  int result=Calculate.cube(5);
10. System.out.println(result);
11. }
12. }
```

Output:125

Restrictions for the static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

```
1. class A{
2.  int a=40;//non static
3.
4.  public static void main(String args[]){
5.   System.out.println(a);
6.  }
7. }
```

Output:Compile Time Error

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

Example of static block

```
1. class A2{
2.     static{System.out.println("static block is invoked");}
3.     public static void main(String args[]){
4.         System.out.println("Hello main");
5.     }
6. }
```

Output:static block is invoked
Hello main

Q) Can we execute a program without main() method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the [main method](#).

```
1. class A3{
2.     static{
3.         System.out.println("static block is invoked");
4.         System.exit(0);
5.     }
6. }
```

Output:

static block is invoked

Since JDK 1.7 and above, output would be:

Error: Main method not found in class A3, please define the main method as:
public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application

Access Modifiers :

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Static Keyword :

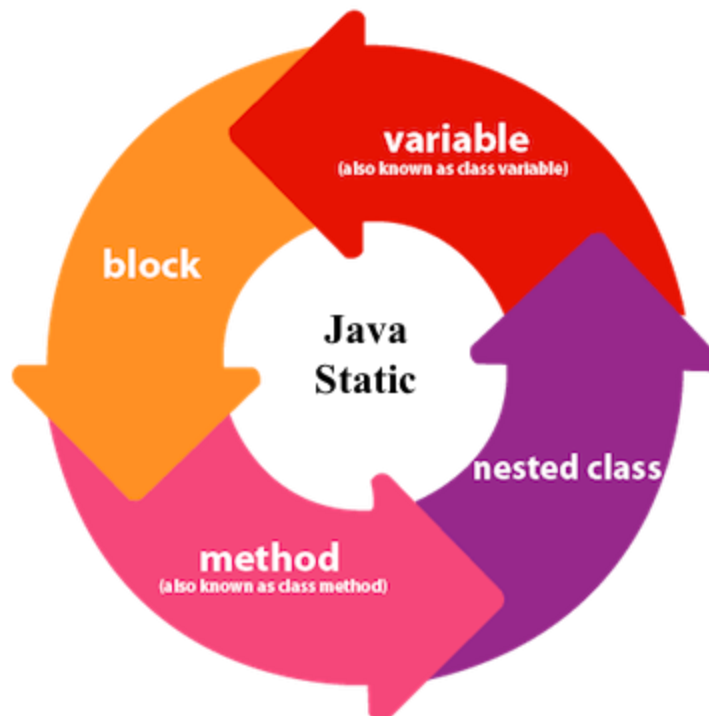
The **static keyword** in [Java](#) is used for memory management mainly. We can apply static keyword with [variables](#), methods, blocks and [nested classes](#). The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)

1) Java static variable

If you declare any variable as static, it is known as a static variable.



- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

Understanding the problem without static variable

1. **class** Student{
2. **int** rollno;

3. String name;
4. String college="ITS";
5. }

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Java static property is shared to all objects.

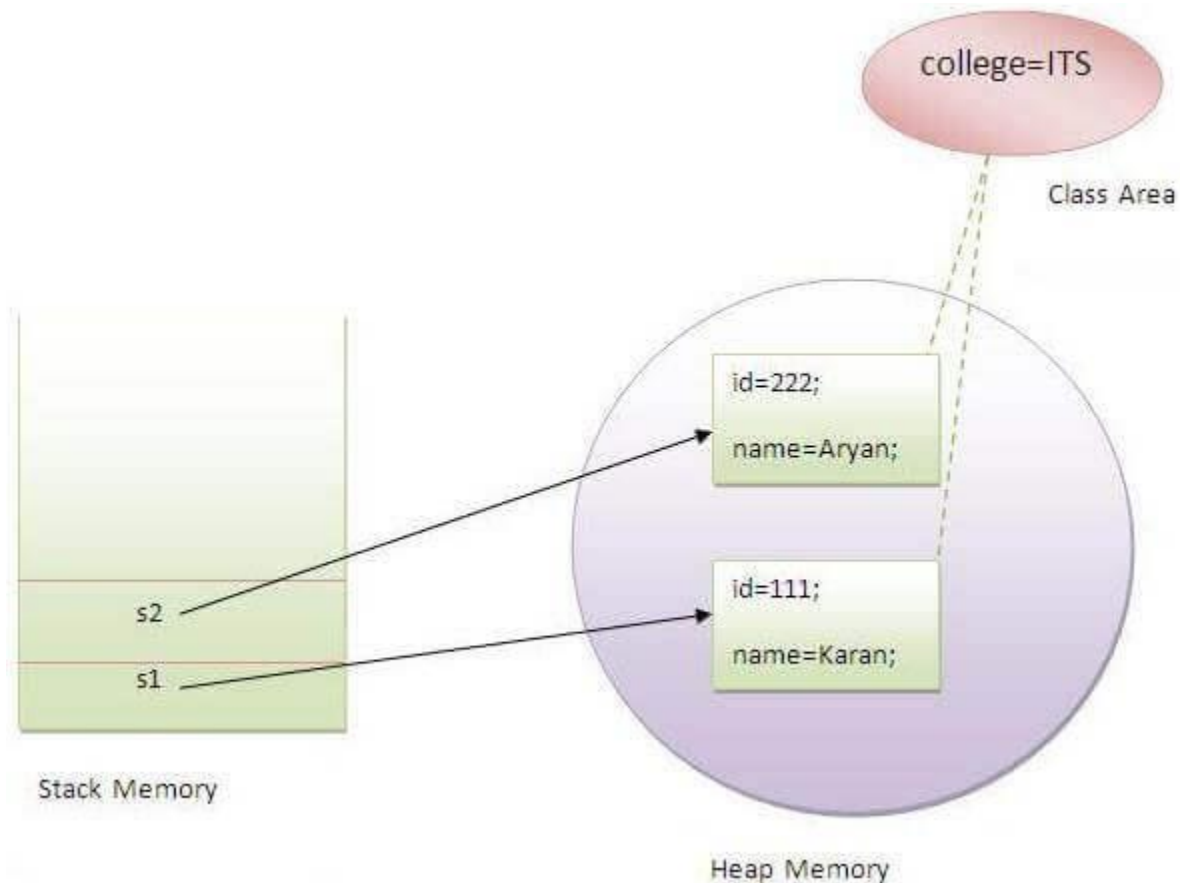
Example of static variable

1. //Java Program to demonstrate the use of static variable
2. **class** Student{
3. **int** rollno;//instance variable
4. String name;
5. **static** String college ="ITS";//static variable
6. //constructor
7. Student(**int** r, String n){
8. rollno = r;
9. name = n;
10. }
11. //method to display the values
12. **void** display () {System.out.println(rollno+" "+name+" "+college);}
- 13.}
- 14.//Test class to show the values of objects
- 15.**public class** TestStaticVariable1{
16. **public static void** main(String args[]){
17. Student s1 = **new** Student(111,"Karan");
18. Student s2 = **new** Student(222,"Aryan");
19. //we can change the college of all objects by the single line of code
20. //Student.college="BBDIT";
21. s1.display();
22. s2.display();
23. }
- 24.}

25. Output:

26. 111 Karan ITS

27. 222 Aryan ITS



Program of the counter without static variable

In this example, we have created an instance variable named `count` which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect other objects. So each object will have the value 1 in the `count` variable.

1. `//Java Program to demonstrate the use of an instance variable`
2. `//which get memory each time when we create an object of the class.`
3. `class Counter{`
4. `int count=0; //will get memory each time when the instance is created`
- 5.
6. `Counter(){`
7. `count++; //incrementing value`

```
8. System.out.println(count);
9. }
10.
11. public static void main(String args[]){
12. //Creating objects
13. Counter c1=new Counter();
14. Counter c2=new Counter();
15. Counter c3=new Counter();
16.}
17.}
```

Output:

```
1
1
1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
1. //Java Program to illustrate the use of static variable which
2. //is shared with all objects.
3. class Counter2{
4. static int count=0; //will get memory only once and retain its value
5.
6. Counter2(){
7. count++; //incrementing the value of static variable
8. System.out.println(count);
9. }
10.
11. public static void main(String args[]){
12. //creating objects
13. Counter2 c1=new Counter2();
14. Counter2 c2=new Counter2();
```

```
15. Counter2 c3=new Counter2();
```

```
16.}
```

```
17.}
```

```
18. Output:
```

```
19.1
```

```
20.2
```

```
21.3
```

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example of static method

```
1. //Java Program to demonstrate the use of a static method.
2. class Student{
3.     int rollno;
4.     String name;
5.     static String college = "ITS";
6.     //static method to change the value of static variable
7.     static void change(){
8.         college = "BBDIT";
9.     }
10.    //constructor to initialize the variable
11.    Student(int r, String n){
12.        rollno = r;
13.        name = n;
14.    }
15.    //method to display values
16.    void display(){System.out.println(rollno+" "+name+" "+college);}
17.}
18.//Test class to create and display the values of object
```

```
19. public class TestStaticMethod{
20.     public static void main(String args[]){
21.         Student.change();//calling change method
22.         //creating objects
23.         Student s1 = new Student(111,"Karan");
24.         Student s2 = new Student(222,"Aryan");
25.         Student s3 = new Student(333,"Sonoo");
26.         //calling display method
27.         s1.display();
28.         s2.display();
29.         s3.display();
30.     }
31. }
```

Output:111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT

Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

How can an object be unreferenced?



1) By nulling a reference:

1. Employee e=**new** Employee();
2. e=**null**;

2) By assigning a reference to another:

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

3) By anonymous object:

1. **new** Employee();

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. **protected void** finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

1. **public static void** gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Simple Example of garbage collection in java

1. **public class** TestGarbage1{
2. **public void** finalize(){System.out.println("object is garbage collected");}
3. **public static void** main(String args[]){
4. TestGarbage1 s1=**new** TestGarbage1();
5. TestGarbage1 s2=**new** TestGarbage1();
6. s1=**null**;
7. s2=**null**;
8. System.gc();
9. }
- 10.}

Output :

object is garbage collected
object is garbage collected