# Difference between JDK, JRE, and JVM

## JDK

JDK provides all the tools, executables, and binaries required to compile, debug, and execute a Java Program.

We can say that JDK is the superset of JRE since it contains JRE with Java compiler, debugger, and core classes.

## JVM

Java Virtual Machine , is responsible for converting the byte code(compiled code) to the machine-specific code.

JVM provides core java functions such as memory management, garbage collection, security, etc. JVM is customizable and we can use java options to customize it. For example, allocating minimum and maximum memory to JVM.

## JRE (Java Run Time Environment )

JRE is the implementation of JVM(this is virtual and does not exist physically , so this is its implementation ).

It provides a platform to execute java programs. JRE consists of JVM, Java binaries, and other classes (rt.jar have access to core classes e.g. java.lang.String, java.lang.Thread, java.util.ArrayList etc.

In windows, rt.jar will always reside under $JAVA\_HOME/jre/lib ) to execute any program successfully.

JRE doesn’t contain any development tools such as Java compiler, debugger, JShell, etc.

If you just want to execute a java program, you can install only JRE. You don’t need JDK because there is no development or compilation of java source code is required.





# Constructor

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

* Constructor name must be the same as its class name
* A Constructor must have no explicit return type
* A Java constructor cannot be abstract, static, final, and synchronized
* We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

# static keyword

The static keyword suggests that something belongs to the class than an instance of the class.

The static can be used with :

* Variable (also known as a class variable)
* Method (also known as a class method)
* Block

## static variable

The static variable can be used to refer to the common property of all objects (which is not unique for each object)

static variable gets memory only once in the class area at the time of class loading.

## 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.
* Also , Non-static field cannot be referenced from a static context/method.
* Non-static method should be able to access the static field.

**Restrictions for the static method**

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

## static block

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

## Why is the Java main method static ?

It is because the for JVM object should not required to call a static method.

# this keyword

this is a reference variable that refers to the current object.

## this can be used to refer current class instance variable.

If there is ambiguity between the instance variables and parameters

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

## this() can be used to invoke current class constructor.

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.

***Rule:*** Here the call to this() must be the first statement in constructor, when calling other from here.

**class** **Student**{

int roll ;

String name ;

**public** Student(String name ){

**this**.name = name ;

}

**public** Student(int roll){

**this**("Student " + roll ) ;

**this**.roll = roll ;

}

**public** String toString(){

**return** **this**.roll + " " + **this**.name ;

}

}

**class** **JavaMain** {

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

{

Io.initializeIO() ;

Student s = **new** Student(1) ;

Io.output.println(s) ;

Io.closeIO() ;

}

}

O/p :

1 Student 1

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

this keyword can also be passed as an argument in the method. It is mainly used in the event handling.

## this can be passed as argument in the constructor call.

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

In such case, return type of the method must be the class type (non-primitive).

# Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

**Types**

* single
* multilevel
* hierarchical

## Why multiple inheritance is not supported in java?

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes.

If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes.

So whether you have same method or different, there will be compile time error.

## Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

It increases the readability of the program.

There are two ways to overload the method in java

* By changing number of arguments
* By changing the data type

**Why Method Overloading is not possible by changing the return type of method only?**

Method overloading is not possible by changing the return type of the method only because of ambiguity.

The compiler does not consider the return type while differentiating the overloaded method, and it is not possible to decide which method to execute based on the return type alone.

For example, if two methods have the same signature but different return types, it would be confusing for the compiler to distinguish between the calls because at the time of invocation, the return type is not given.

The parameter list is the only way for the compiler to distinguish between the function calls, and the return value is received at the completion of the function.

## Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found.

If there are matching type arguments in the method, type promotion is not performed.

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

**class** **Test**{

void sum(int a,long b){System.out.println("a method invoked");}

void sum(long a,int b){System.out.println("b method invoked");}

}

**class** **JavaMain** {

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

{

Io.initializeIO() ;

Test t = **new** Test() ;

t.sum(20,20) ;

Io.output.println() ;

Io.closeIO() ;

}

}

error: reference to sum is ambiguous

t.sum(20,20) ;

^

both method sum(int,long) in Test and method sum(long,int) in Test match

## Method Overriding

If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

**Rules for Java Method Overriding**

* The method must have the same name as in the parent class.
* The method must have the same parameter as in the parent class.
* There must be an IS-A relationship (inheritance).

**Can we override static method?**

* We can declare static methods with the same signature in the subclass, but it is not considered overriding as there won’t be any run-time polymorphism.
* For class (or static) methods, the method according to the type of reference is called, not according to the object being referred, which means method call is decided at compile time.
* For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time.
* An instance method cannot override a static method, and a static method cannot hide an instance method.

class Base {  
  
 *// Static method in base class which will be hidden in subclass* public static void display() {  
 System.*out*.println("Static or class method from Base");  
 }  
  
 *// Non-static method which will be overridden in derived class* public void print() {  
 System.*out*.println("Non-static or Instance method from Base");  
 }  
}  
  
*// Subclass*class Derived extends Base {  
  
 *// This method is hidden by display() in Base* public static void display() {  
 System.*out*.println("Static or class method from Derived");  
 }  
  
 *// This method overrides print() in Base* public void print() {  
 System.*out*.println("Non-static or Instance method from Derived");  
 }  
}  
  
  
public class Example1 {  
 public static void main(String args[ ]) {  
 Base obj1 = new Derived();  
  
 *// As per overriding rules this should call to class Derive's static  
 // overridden method. Since static method can not be overridden, it  
 // calls Base's display()* obj1.*display*();  
  
 *// Here overriding works and Derive's print() is called* obj1.print();  
 }  
}

O/p :

Static or class method from Base

Non-static or Instance method from Derived

**Covariant Return Type**

Covariant return types in Java allow a subclass to override a method from its superclass with a more specific return type.

This feature was introduced in Java 5.

class A {

A get() {

return this;

}

}

class B extends A {

B get() {

return this;

}

}

In this example, the return type of the get() method of class A is A, but the return type of the get() method of class B is B. Both methods have different return types, but it is method overriding. This is known as covariant return type.