**JSP tutorial-**

<http://www.tutorialspoint.com/jsp/jsp_syntax.htm>

[**String vs StringBuffer vs StringBuilder in Java**](http://javarevisited.blogspot.in/2011/07/string-vs-stringbuffer-vs-stringbuilder.html)

<http://javarevisited.blogspot.in/2011/07/string-vs-stringbuffer-vs-stringbuilder.html>

1) String is immutable while **StringBuffer and StringBuilder is mutable** object.

2) **StringBuffer is**[**synchronized**](http://www.blogger.com/goog_1642539054)while **StringBuilder is not** which makes StringBuilder faster than StringBuffer.

3) Concatenation operator "+" is internal implemented using either StringBuffer or StringBuilder.

4) Use String if you require [immutability](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html), use Stringbuffer in java if you need mutable + [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) and use StringBuilder in Java if you require mutable + without thread-safety.

# [How to avoid deadlock in Java Threads](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html)

<http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html>

Path and Classpath are operating system level environment variales. Path is used define where the system can find the executables(.exe) files and classpath is used to specify the location .class files.

**What is the difference between static and instance variable?**

Static Variable= Defined at Class Level   
Instance Variable =Defined at Instace/Object Level.   
  
Static Variables are initialized ,loaded with Class itself.   
  
But instance variable initialized when Object for that Class is instantiated.   
  
In other words - For 1 class ,its different objects can have different values for same instance variable.(Each instance =can have its own value)   
But for same class ,static variable (since belongs to class itself ) -same value is shared by each object.

**Instance versus static methods**

Each Java method is either an instance method or a static method. Any method not marked with the static keyword is an instance method. Lastly, all Java methods must be defined within a class. In order to call an instance method, you must first have an instance of the class within which the method you want to call is defined. Suppose we want to call an instance method named "method1" defined in class "ClassA". We could do the following:   
  
ClassA x = new ClassA();   
x.method1();   
  
If method1 were instead static, then we would call the method by referring directly to the class itself:   
  
ClassA.method1();

**How to define a constant variable in Java?**

The variable should be declared as static and final. So only one copy of the variable exists for all instances of the class and the value can't be changed also.  
static final int PI = 3.14; is an example for constant.

# [Will a static block execute without main method?](http://stackoverflow.com/questions/17889220/will-a-static-block-execute-without-main-method)

[**http://stackoverflow.com/questions/17889220/will-a-static-block-execute-without-main-method**](http://stackoverflow.com/questions/17889220/will-a-static-block-execute-without-main-method)

# [Which is the first method called when Java executes a program?](http://stackoverflow.com/questions/4194407/which-is-the-first-method-called-when-java-executes-a-program)

[**http://stackoverflow.com/questions/4194407/which-is-the-first-method-called-when-java-executes-a-program**](http://stackoverflow.com/questions/4194407/which-is-the-first-method-called-when-java-executes-a-program)

It's usually main. But in this program, it's pain:

public class WhatThe {

public static final int x = pain();

public static int pain() {

System.out.println("pain!");

return 0;

}

public static void main(String[] args) {

System.out.println("main");

}

}

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

|  |
| --- |
| Ans)Yes, one of the way is static block but in previous version of JDK not in JDK 1.7. |

**class** A3{

**static**{

  System.out.println("static block is invoked");

    System.exit(0);

}

}

Output: static block is invoked (if not JDK7)

**Test code in the below link**

[**http://www.javatpoint.com/opr/test.jsp?filename=A3**](http://www.javatpoint.com/opr/test.jsp?filename=A3)

**In JDK 7, the outputs for the below samples are as follows:**

**Sample1**

class A3{

static{

System.out.println("static block is invoked");

System.exit(0);

}

}

**Compile by: javac A3.java**

**Run by: java A3**

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

**Sample 2**

class A3{

public static void main(String[] args){

System.out.println("Main block is invoked");

}

static{

System.out.println("static block is invoked");

// System.exit(0);

}

}

**Compile by: javac A3.java**

**Run by: java A3**

static block is invoked  
Main block is invoked

**Sample 3**

**class A3{**

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

**System.out.println("Main block is invoked");**

**}**

**static{**

**System.out.println("static block is invoked");**

**System.exit(0);**

**}**

**}**

**Compile by: javac A3.java**

**Run by: java A3**

static block is invoked

**Sample 4**

**class A3{**

**public void m1 (String arg1){**

**arg1="I am the second string";**

**}**

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

**A3 a3=new A3();**

**String arg2="im the first string";**

**a3.m1(arg2);**

**System.out.println(arg2);**

**}**

**static{**

**System.out.println("static block is invoked");**

**//System.exit(0);**

**}**

**}**

**Compile by: javac A3.java**

**Run by: java A3**

static block is invoked  
im the first string

**How is final different from finally and finalize()?**

final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited, final method can't be overridden and final variable can't be changed.   
  
finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.   
  
finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

**I want to print "Hello" even before main() is executed. How will you acheive that?**

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the main()method. And it will be executed only once.

**Can we declare a static variable inside a method?**

Static varaibles are class level variables and they can't be declared inside a method. If declared, the class will not compile.

**What is use of a abstract variable?**

Variables can't be declared as abstract. only classes and methods can be declared as abstract.

**Can you create an object of an abstract class?**

Not possible. Abstract classes can't be instantiated.

**Sample 5**

class A3{

int x=3;

public static void main (String args[]){

String s1=new String ("abc");

String s2=s1;

s1 +="d";

System.out.println("First: " + s1+ ".." +s2+".." + (s1==s2));

StringBuffer sb1=new StringBuffer("abc");

StringBuffer sb2=sb1;

sb1.append("d");

System.out.println("Second: " + sb1+ ".." +sb2+".." + (sb1==sb2));

}

}

**Compile by: javac A3.java**

**Run by: java A3**

2  
First: abcd..abc..false  
Second: abcd..abcd..true

**Important points on Interfaces and Abstract Classes:**

* Abstract Classes have abstract as well as non abstract methods.
* Abstract Classes cannot be instantiated. They definitely have to be extended and the inheriting sub class should implement all the methods. All methods of an abstract class must be overridden by the subclass which is why no method in an abstract class can be final since final methods cannot be overridden. If all the methods are not overridden, then the inheriting class must be declared abstract.
* There can be an abstract class without any abstract methods. It is basically to avoid instantiation of the class.
* Interfaces are implicitly Abstract i.e. need not use the Abstract keyword. All methods of an interface are abstract.
* A class implementing an interface must override all the methods of the interface else this class must be declared abstract.
* A class can extend only one class whereas a class can implement any number of interfaces. (An interface can extend more than one interface)
* An interface cannot implement another interface since interfaces can have only abstract methods.
* An interface cannot be final. Doing so will result in compilation error.
* Can a class be defined inside an Interface? Yes it's possible.
* Can an Interface be defined inside a class? Yes it's possible.
* an interface cannot be instantiated with the new operator; for example, Runnable a=new Runnable(); is not allowed.

About calendar class:

public abstract class **Calendar**  
implements **[Serializable](http://pic.dhe.ibm.com/infocenter/adiehelp/v5r1m1/topic/com.sun.api.doc/java/io/Serializable.html)**, **[Cloneable](http://pic.dhe.ibm.com/infocenter/adiehelp/v5r1m1/topic/com.sun.api.doc/java/lang/Cloneable.html)**  
extends [**Object**](http://pic.dhe.ibm.com/infocenter/adiehelp/v5r1m1/topic/com.sun.api.doc/java/lang/Object.html)

**What is the difference between creating String as new() and literal?**

When we create string with new() Operator, it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

String s = new String("Test");  
   
does not  put the object in String pool , we need to call String.intern() method which is used to put  them into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" Java automatically put that into String pool.

**Give the list of Java Object class methods. - cefghnntw**

**clone()** - Creates and returns a copy of this object.

**equals()** - Indicates whether some other object is "equal to" this one.

**finalize()** - Called by the garbage collector on an object when garbage collection

determines that there are no more references to the object.

**getClass()** - Returns the runtime class of an object.

**hashCode()** - Returns a hash code value for the object.

**notify()** - Wakes up a single thread that is waiting on this object's monitor.

**notifyAll()** - Wakes up all threads that are waiting on this object's monitor.

**toString()** - Returns a string representation of the object.

**wait()** - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

**What will happen if you call return statement or System.exit on try or catch block ? will finally block execute?**

This is a very *popular tricky Java question* and its tricky because many programmer think that finally block always executed. This question challenge that concept by putting return statement in try or catch block or calling System.exit from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put return statement in try block or catch block but finally block won't run if you call System.exit form try or catch.

# [Can a private method in super class be overriden in the sub-class?](http://stackoverflow.com/questions/11976446/can-a-private-method-in-super-class-be-overriden-in-the-sub-class)

[**http://stackoverflow.com/questions/11976446/can-a-private-method-in-super-class-be-overriden-in-the-sub-class**](http://stackoverflow.com/questions/11976446/can-a-private-method-in-super-class-be-overriden-in-the-sub-class)

[**http://stackoverflow.com/questions/2223386/why-doesnt-java-allow-overriding-of-static-methods**](http://stackoverflow.com/questions/2223386/why-doesnt-java-allow-overriding-of-static-methods)

**Can you override private or static method in Java ?**

Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java.  Anyway, you can not override private or static method in Java, if you create similar method with same return type and same method arguments that's called method hiding.

A parent's private method cannot be accessed or inherited by a child class, in line with principles of encapsulation. It is hidden.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *no modifier(package- private)* | Y | Y | N | N |
| private | Y | N | N | N |

**Parent class is extended by child (subclass) class:**

**Following are allowed:**

**Parent p=new Parent();**

**Child c= new Child();**

**Parent p =new Child(); (Here you are casting child class to Parent type.)**

**Keeping this in mind we can conclude that** An object reference can be cast to an interface reference when the object implements the referenced interface.

**What is Externalizable?**

Externalizable is an Interface that extends Serializable Interface. And sends data into Streams in Compressed Format. It has two methods, writeExternal(ObjectOuput out) andreadExternal(ObjectInput in)

**What is an object's lock and which object's have locks?**

An object's lock is a mechanism that is used by multiple threads to obtain synchronized access to the object. A thread may execute a synchronized method of an object only after it has acquired the object's lock. All objects and classes have locks. A class's lock is acquired on the class's Class object.

Some 15 questions in the below link:

<http://bateru.com/news/2011/03/484/>

1. What is the purpose of serialization?  
   Answer: Serialization is the conversion of an object to a series of bytes, so that the object can be easily saved to persistent storage or streamed across a communication link. The byte stream can then be deserialised – converted into a replica of the original object.  
   [Source](http://stackoverflow.com/questions/447898/what-is-object-serialization) | [Example](http://java.sun.com/developer/onlineTraining/Programming/BasicJava2/serial.html)
2. What is the difference between JDK and JRE?  
   Answer: Java Development Kit (JDK) is the most widely used Java Software Development Kit. Java Runtime Environment (JRE) is an implementation of the Java Virtual Machine which executes Java programs.

**JVM JRE JDK**

[**http://bateru.com/news/2011/03/484/**](http://bateru.com/news/2011/03/484/)**’**

**JVM:**

It is a specification that provides runtime environment in which java bytecode can be executed.

The JVM performs following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

**JRE (JVM +set of libraries)**

It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

**JDK**

It contains JRE + development tools.

**What is a native method?**

A native method is a method that is implemented in a language other than Java.

**Is the ternary operator written x : y ? z or x ? y : z ?**

It is written x ? y : z.

**Does a class inherit the constructors of its superclass?**

A class does not inherit constructors from any of its superclasses.

Instanceof Syntax

obj **instanceof** Object

**What modifiers may be used with a top-level class?**

A top-level class may be public, abstract, or final.

**Can a method be overloaded based on different return type but same argument type ?**

No, because the methods can be called without using their return type in which case there is ambiquity for the compiler.

**Overloading needs different types /number of arguments**

**What happens to a static variable that is defined within a method of a class ?**

Can't do it. You'll get a compilation error.

**What is the difference between the Boolean & operator and the && operator?**

If an expression involving the Boolean & operator is evaluated, both operands are evaluated. Then the & operator is applied to the operand. When an expression involving the && operator is evaluated, the first operand is evaluated. If the first operand returns a value of true then the second operand is evaluated. The && operator is then applied to the first and second operands. If the first operand evaluates to false, the evaluation of the second operand is skipped.

**How are this() and super() used with constructors?**

this() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor.

**What is a transient variable?**

Transient variable is a variable that may not be serialized.

**Is sizeof a keyword?**

The sizeof operator is not a keyword.

**Collections:**

### HashMap

* It has pair values(keys,values)
* NO duplication key values
* unordered unsorted
* it allows one null key and more than one null values

### HashTable

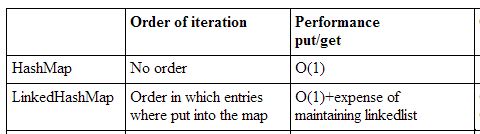
* same as hash map
* it does not allows null keys and null values

### LinkedHashMap

* It is ordered version of map implementation
* Based on linked list and hashing data structures

### TreeMap

* Ordered and sortered version
* based on hashing data structures



## Set Implementations

Being a Collection subtype all methods in the Collection interface are also available in the Set interface.

Since Set is an interface you need to instantiate a concrete implementation of the interface in order to use it. You can choose between the following Set implementations in the Java Collections API:

* java.util.EnumSet
* java.util.HashSet
* java.util.LinkedHashSet
* java.util.TreeSet

Each of these Set implementations behaves a little differently with respect to the order of the elements when iterating theSet, and the time (big O notation) it takes to insert and access elements in the sets.

HashSet is backed by a HashMap. It makes no guarantees about the sequence of the elements when you iterate them.

LinkedHashSet differs from HashSet by guaranteeing that the order of the elements during iteration is the same as the order they were inserted into the LinkedHashSet. Reinserting an element that is already in the LinkedHashSet does not change this order.

TreeSet also guarantees the order of the elements when iterated, but the order is the sorting order of the elements. In other words, the order in which the elements whould be sorted if you used a Collections.sort() on a List or array containing these elements. This order is determined either by their natural order (if they implement Comparable), or by a specific Comparator implementation.

There are also Set implementations in the java.util.concurrent package, but I will leave the concurrency utilities out of this tutorial.

Here are a few examples of how to create a Set instance:

Set setA = new EnumSet();

Set setB = new HashSet();

Set setC = new LinkedHashSet();

Set setD = new TreeSet();

**Vector and arraylist are same except that vector is synchronized and arraylist is not.**