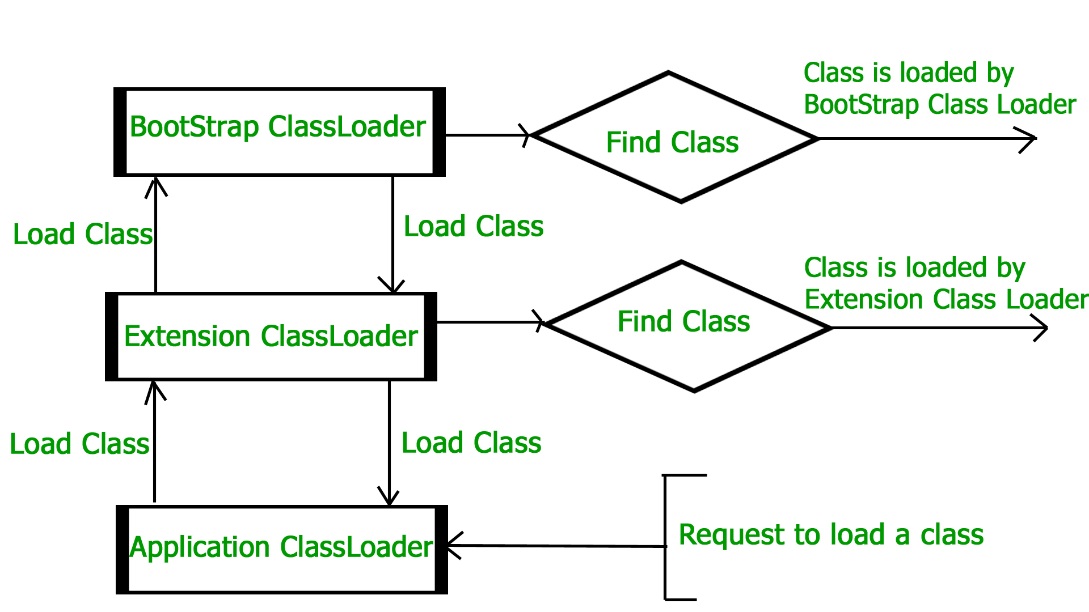
**Java Class Loader :** JVM follow Delegation-Hierarchy principle to load classes. System class loader delegate load request to extension class loader and extension class loader delegate request to boot-strap class loader. If class found in boot-strap path, class is loaded otherwise request again transfers to extension class loader and then to system class loader. At last if system class loader fails to load class, then we get run-time exception java.lang.ClassNotFoundException.



**Bootstrap class loader** : Every JVM implementation must have a bootstrap class loader, capable of loading trusted classes. It loads core java API classes present in JAVA\_HOME/jre/lib directory. This path is popularly known as bootstrap path. It loads **rt.jar**

**Extension class loader** : It is child of bootstrap class loader. It loads the classes present in the extensions directories JAVA\_HOME/jre/lib/ext(Extension path) or any other directory specified by the java.ext.dirs system property. It is implemented in java by the sun.misc.Launcher$ExtClassLoader class.

**System/Application class loader** : It is child of extension class loader. It is responsible to load classes from application **class path**. It internally uses Environment Variable which mapped to java.class.path. It is also implemented in Java by the sun.misc.Launcher$AppClassLoader class.

**Interpreter** : It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.

**Just-In-Time Compiler(JIT)** : It is used to increase efficiency of interpreter.It compiles the entire bytecode and changes it to native code so whenever interpreter see repeated method calls,JIT provide direct native code for that part so re-interpretation is not required,thus efficiency is improved.

**Why String is immutable or final ?**

**String pool** is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space **because different String variables can refer to same String variable in the pool.**

If String is not immutable then it would **cause severe security threat to the application**. For example, database username, password are passed as String to get database connection and in socket programming host and port details passed as String. Since String is immutable it’s value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.

Since String is immutable, **it is safe for multithreading** and a single String instance can be shared across different threads. This avoid the usage of synchronization for thread safety, Strings are implicitly thread safe.

Since String is immutable, **its hashcode is cached at the time of creation** and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and it’s processing is fast than other HashMap key objects.

**String intern**

public String intern()

When the intern() method is invoked on a String object it looks the string contained by this String object in the pool, if the string is found there then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

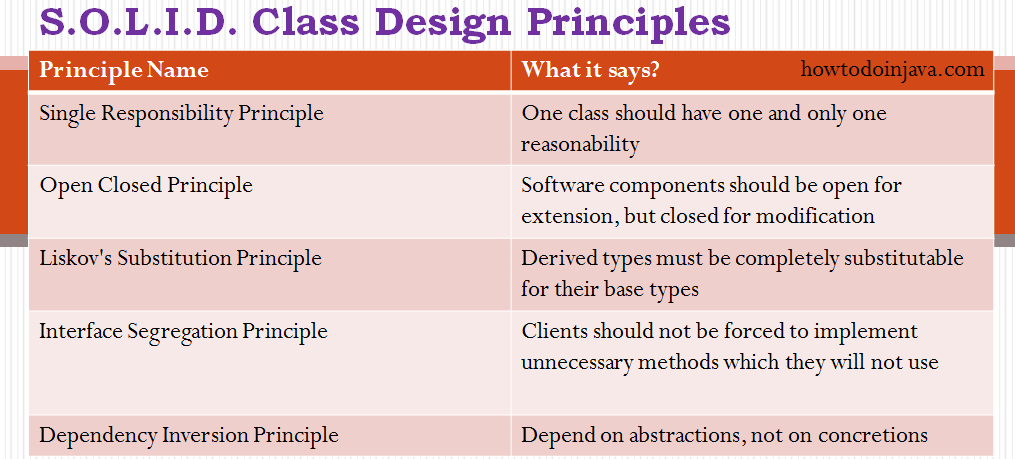
String s1 = "amit";

String s2 = **new** String("amit");

System.***out***.println(s1==s2); // false

String s3 = s2.intern();// returns string from string pool

System.***out***.println(s1==s3); //true



**ClassNotFoundException** is a runtime exception that is thrown when an application tries to load a class at runtime using the **Class.forName()** or **loadClass()** or **findSystemClass()** methods ,and the class with specified name are not found in the classpath.

public static void main(String[] args)

{

try

{

Class.forName("oracle.jdbc.driver.OracleDriver");

}catch (ClassNotFoundException e)

{

e.printStackTrace();

}

}

**NoClassDefFoundError** is an error that is thrown when the Java Runtime System tries to load the definition of a class, and that class definition is no longer available. The required class definition was present at compile time, but it was missing at runtime.

class A

{

// some code

}

public class B

{

public static void main(String[] args)

{

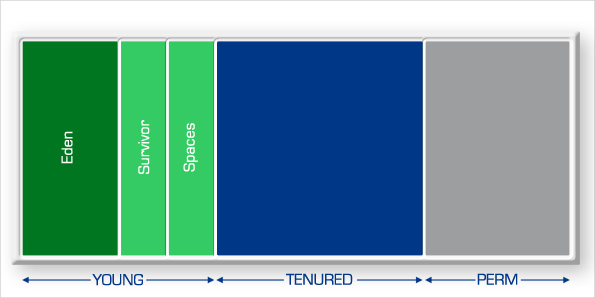
A a = new A();

}

}

**JVM Architecure**





**Garbage Collector Types**

**1. Serial GC** – A single thread collector. Mostly applies to small applications with small data usage. Can be enabled by specifying the command line option: -XX:+UseSerialGC

**2. Parallel GC** – Even from the naming, the difference between Serial and Parallel would be that Parallel GC uses multiple threads to perform the garbage collecting process. This GC type is also known as the throughput collector. It can be enabled by explicitly specifying the option: -XX:+UseParallelGC

**3. Mostly concurrent GC** – If you remember, earlier in this article, it was mentioned that the garbage collecting process is actually pretty expensive, and when it runs, all thread are paused. However, we have this mostly concurrent GC type, which states that it works concurrent to the application. However, there is a reason why it is “mostly” concurrent. It does not work 100% concurrently to the application. There is a period of time for which the threads are paused. Still, the pause is kept as short as possible to achieve the best GC performance. Actually, there are 2 types of mostly concurrent GCs:

**Enum**

Enum is type-safe you can not assign anything else other than predefined Enum constants.

Enum can implement interface.

Enum cannot extends class because all Enum by default extend abstract base class java.lang.Enum.

Enum implements Comparable interface, which is main requirement to be used in Sorted Collection like [TreeSet and TreeMap](http://java67.blogspot.com/2012/08/difference-between-treemap-and-treeset-java.html).

We can use Enum in Switch case in Java.

public void developerState(DayOfWeek today){

switch(today){

case MONDAY:

System.out.println("Hmmmmmmmm");

break;

case TUESDAY:

System.out.println("Hmmmm");

break;

case FRIDAY :

System.out.println("Yeahhhhhh");

break;

}

}

**Ways to create object**

1. Using New keyword
2. Class.forName()

Class cls = Class.forName("NewInstanceExample");

1. Using clone()
2. Using Deserialization
3. Using newInstance() method of Constructor

Constructor<ReflectionExample> constructor

                = ReflectionExample.class.getDeclaredConstructor();

            ReflectionExample r = constructor.newInstance();

**InstanceOf vs isInstance()**

**instanceof** operator and **isInstance()** method both are used for checking the class of the object. But main difference comes when we want to check the class of object dynamically. In this case **isInstance()** method will work. There is no way we can do this by **instanceof** operator.

**instanceof** operator and **isInstance()** method both return a boolean value.

i instanceof Integer

Class.forName(c).isInstance(obj);

**New operator vs newInstance()**

New operator is used to create objects, but if we want to decide type of object to be created at runtime, there is no way we can use new operator

 Object obj = Class.forName(c).newInstance();

**Class.forName()** method return class **Class** object on which we are calling **newInstance()**method which will return the object of that class which we are passing as command line argument.  
If the passed class doesn’t exist then **ClassNotFoundException** will occur.  
**InstantionException** will occur if the passed class doesn’t contain default constructor as **newInstance()**method internally calls the default constructor of that particular class.  
**IllegalAccessException** will occur if we don’t have access to the definition of specified class definition.

**How to make an object eligible for garbage collection**

**Object created inside a method** : When a method is called it goes inside the stack frame. When the method is popped from the stack, all its members dies and if some objects were created inside it then these objects becomes unreachable or anonymous after method execution and thus becomes eligible for garbage collection

**Reassigning the reference variable:**When reference id of one object is referenced to reference id of some other object then the previous object has no any longer reference to it and becomes unreachable and thus becomes eligible for garbage collection.

**Nullifying the reference variable :**When all the reference variables of an object are changed to NULL, it becomes unreachable and thus becomes eligible for garbage collection.

**Anonymous object :**The reference id of an anonymous object is not stored anywhere. Hence, it becomes unreachable.

 new Test("t1");

**Island of isolation**

* Object 1 references Object 2 and Object 2 references Object 1. Neither Object 1 nor Object 2 is referenced by any other object. That’s an island of isolation.
* Basically, an island of isolation is a group of objects that reference each other but they are not referenced by any active object in the application. Strictly speaking, even a single unreferenced object is an island of isolation too.

**Parent class constructor throws checked exception**

If parent class constructor throws checked exception then child class constructor should throw the checked exception.

**Advantage of Functional Interface/Lambda Expression**

**Prevent Singleton Breaking :**

To prevent from reflection use enum.

To prevent from deserialization implement readResolve() method.

class Singleton implements Serializable

{

    // public instance initialized when loading the class

    public static Singleton instance = new Singleton();

    private Singleton()

    {

        // private constructor

    }

    // implement readResolve method

**protected Object readResolve()**

**{**

**return instance;**

**}**

}

**Fail-fast and Fail-safe**

Fail-safe iterators means they will not throw any exception even if the collection is modified while iterating over it.

Whereas Fail-fast iterators throw an exception(ConcurrentModificationException) if the collection is modified while iterating over it