# If base class has 2 parameter constructor and child class has 1 param meter construtor and default constructor is not defined

# Then If default constructor is not present in parent class it will throw exception

# How to tackle this exception

# Define default constructor in parent class

**package** \_NewOne;

**public** **class** Base {

**int** a;

**int** b;

Base(){}

Base(**int** a) {

**this**.a = a;

}

}

**package** \_NewOne;

**public** **class** Child **extends** Base {

Child(**int** x,**int** y)

{

System.***out***.println("Ram");

}

}

# Define super in child class constructor

**package** \_NewOne;

**public** **class** Base {

**int** a;

**int** b;

Base(**int** a) {

**this**.a = a;

}

}

**package** \_NewOne;

**public** **class** Child **extends** Base {

Child(**int** x,**int** y)

{

**super**(y);

System.***out***.println("Ram");

}

}

# Why Should I Write Getters and Setters?

<https://dzone.com/articles/why-should-i-write-getters-and-setters>

**I understand, but generally, we do not write anything in getters/setters. We just return and set the field, which is same as exposing a field as public. So why are you saying all of this?**

To answer this question, I say by writing getters/setters, we create a provision to add any validation method in the future, currently, there is no validation, but if anything goes wrong in the future we just add validation logic in the setter.

Like settet with validation

public void setName(String name) {

if(name == null ){

throw new NullPointerException("Name can't be null");

}

this.name = name;

}

public void setId(int id) {

if(id == 0 ){

throw new IllegalArgumentException("id can't be zero");

}

this.id = id;

}

**SOLID Principle**

## ****S is for Single Responsibility Principle-**** a class or module should do one thing only

## ****Open/Close Principle-**** Open/Closed Principle states that code entities should be open for extension, but closed for modification.

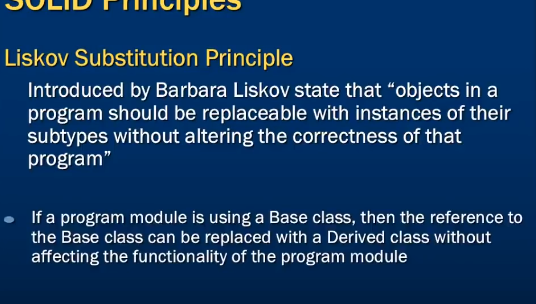
1. Liskov Substitution Principle- The LSP says, basically, that any child type of a parent type should be able to stand in for that parent without things blowing up.

Animal an = **new Animal()—it can be replced with child type object**

Animal an = **new Dog()—**

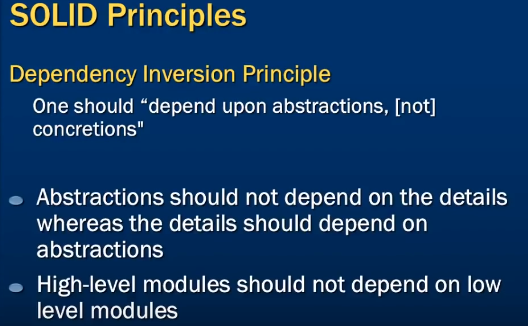
Parent objct can be replaced by child object

The principle 'LSP' states that references of BASE CLASS could be replaced with derived classes of BASE CLASS, without causing any adverse impact on the system.



1. **Interface Segregation Principle-** says that you should favor many, smaller, client-specific interfaces over one larger, more monolithic interface.

**5.Dependency Inversion-** is one of the last principles we are going to look at. The principle states that: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

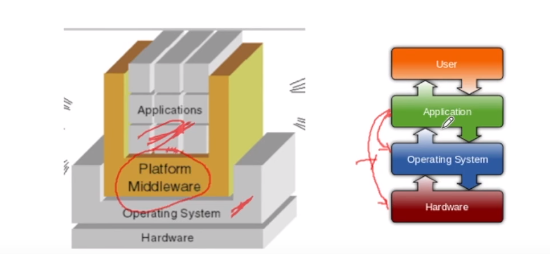
****

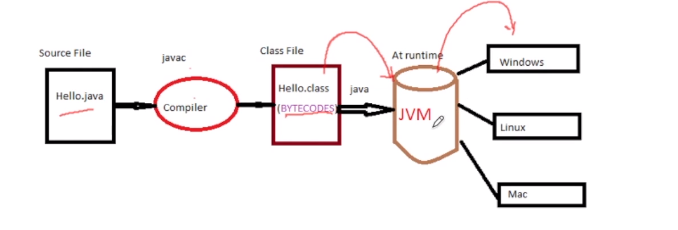
**how to mock static methods using mockito**

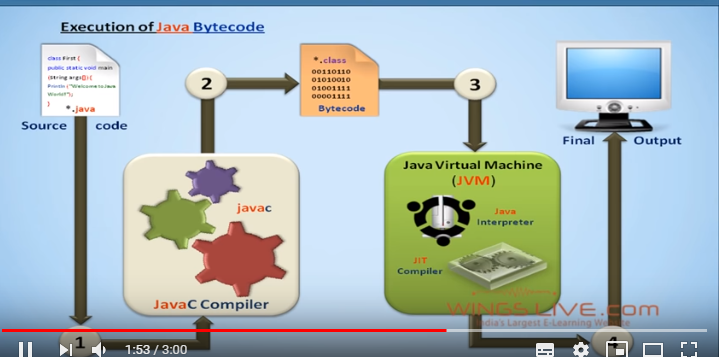
**Mockito** allows us to create **mock** objects. Since **static method** belongs to the class, there is no way in **Mockito** to **mock static methods**. However, we can use PowerMock along with **Mockito** framework to **mock static methods**.

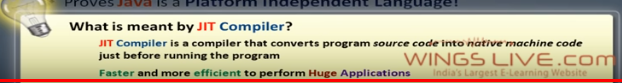
# [How is Java platform-independent when it needs a JVM to run?](https://stackoverflow.com/questions/2748910/how-is-java-platform-independent-when-it-needs-a-jvm-to-run)

This is because of the magic of Byte Code which is OS indepedent. When java compiler compiles any code then it generates the byte code not the machine native code(unlike C compiler). Now this byte code needs an interpreter to execute on a machine. This interpreter is JVM. So JVM reads that byte code(that is machine indepedent) amd execute it. Different JVM is designed for different OS and byte code is able to run on different OS.

****

****

****

****

**if parent class dont have default constructor and parametrized constoruct is defined then child class construtor must be parametrized and it need to call parent class parameterized consrutor explicitely**

**if parent class have both default constructor and parametrized constoructor is defined then child class construtor need not to be parametrized, things wll run normal way**

Multiprocessing

* When one system is connected to more than one processor which collectively work for the completion of the task, it is called as multiprocessing systems.

Like chrome open multiple tabs

Multithreading

* "Multithreading is a conceptual programming paradigm where a process is divided into a number of sub-processes called as threads. Each thread is independent and has its own path of execution with enabled inter thread communication."
* "Thread is the path followed while executing a program. Each thread has its own program counter, stack and register."

Like Excel sheet it insert update at same time

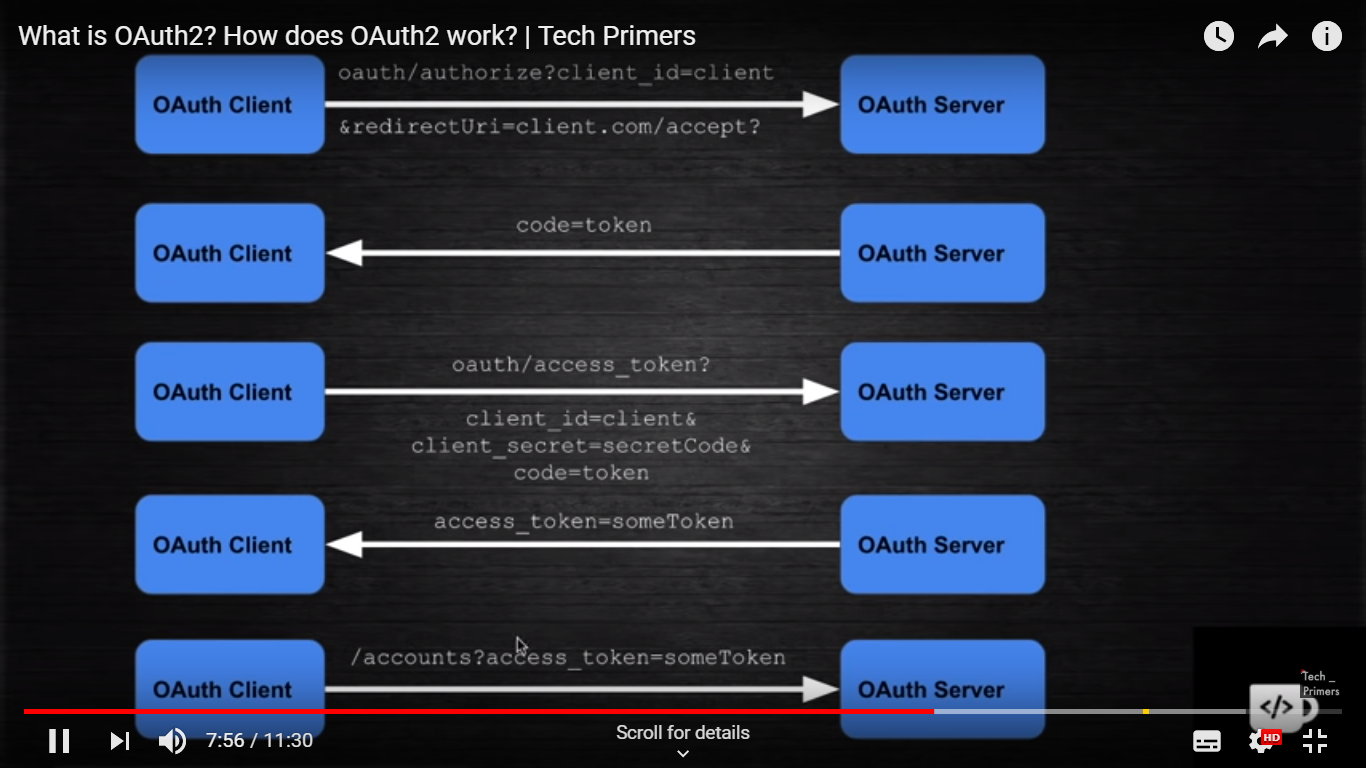
Multiprocessing vs Multithreading

Creating a process can **consume time** and **even exhaust** the system resources. However creating threads is **economical** as threads belonging to the same process share the belongings /resources of that process.

* Benefits of Multi threading include increased responsiveness. Since there are multiple threads in a program, so if one thread is taking too long to execute or if it gets blocked, the rest of the threads keep executing without any problem. Thus the whole program remains responsive to the user by means of remaining threads.

**OAUTH-** OAuth is an open-standard authorization protocol or framework that provides applications the ability for “secure designated access.” For example, you can tell Facebook that it’s OK for ESPN.com to access your profile or post updates to your timeline without having to give ESPN your Facebook password. This minimizes risk in a major way: In the event ESPN suffers a breach, your Facebook password remains safe.

OAuth doesn’t share password data but instead uses authorization tokens to prove an identity between consumers and service providers. OAuth is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.



**Anonymous Class-** without name and return type class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

// anonymous inner class is generated with taste$1 class so 3 class genareted 1.TasteOfDish 2.AnonymouseClass 3.AnonymouseClass$

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

// anonymous inner class is generated with taste$1 class so 3 class genareted 1.TasteOfDish 2.AnonymouseClass 3.AnonymouseClass$

TasteOfDish obj=**new** TasteOfDish()

{ **public** **void** taste()

{

System.***out***.println("sweet");

}};

obj.taste();

TasteOfDish obj1=**new** TasteOfDish();

obj1.taste();

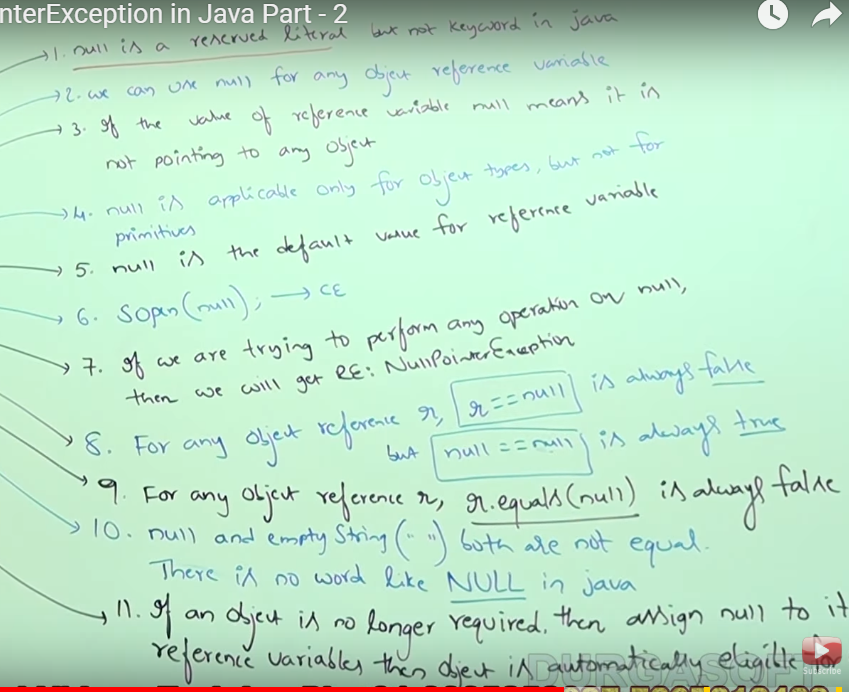
**class** TasteOfDish

{**public** **void** taste()

{System.***out***.println("salty");

}

}

****

//System.out.println("ada");

String k=**null**;

String k1=null;

System.***out***.println(k) 🡪o/p null

String k3 =k+k1;

System.out.println(k3)// o/p nullnull

System.out.println(k.concat("Ram")); nullnullRam

System.***out***.print(**null**); Compiletime Error

**Java 8**

**Predefined Functional Interfaces**

|  |
| --- |
| **Predicate** Represents a predicate (boolean-valued function) of one argument.  Predicate<Integer> p1= i->i%2==0; |
|  |
| **Function** Represents a function that accepts one argument and produces a result. |
|  |

|  |
| --- |
| **Consumer** Represents an operation that accepts a single input argument and returns no result. |
|  |

**Supplier** Represents a supplier of results.

**Two Argument Predefined Functional Interfaces**

BiPredicate

BiFunction

BiConsumer

**Primitive Function Interface**

IntPredicate

IntFunction

IntConsumer

1.Functional Programming

2.Introduction of Optional  
3. Defender Methods  
 4.Lambda Expressions

5.Stream Api foreach loop

[forEach() method in Iterable interface](https://www.journaldev.com/2389/java-8-features-with-examples#iterable-forEach)

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

        List<String> gamesList = **new** ArrayList<String>();

        gamesList.add("Football");

        gamesList.add("Cricket");

        gamesList.add("Chess");

        gamesList.add("Hocky");

        System.out.println("------------Iterating by passing method reference---------------");

        gamesList.forEach(System.out::println);

    }

}

1. [default and static methods in Interfaces](https://www.journaldev.com/2389/java-8-features-with-examples#interface-default-static-method)
2. [Functional Interfaces and Lambda Expressions](https://www.journaldev.com/2389/java-8-features-with-examples#functional-interface-lambdas)
3. [Java Stream API for Bulk Data Operations on Collections](https://www.journaldev.com/2389/java-8-features-with-examples#java-stream-api)
4. [Java Time API](https://www.journaldev.com/2389/java-8-features-with-examples#java8-time) .zone base time and datetime split
5. [Collection API improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-collection)
6. [Concurrency API improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-concurrency)
7. [Java IO improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-io)
8. [Miscellaneous Core API improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-core)

**Introduction of Optional class: Used to deal with nullPoinetr exception**

The main benefit of Optional is to Avoid null pointer exception: There is another class named Optional in the util package, as it is used to avoid the null pointer exception , If the value is present then it will return the true value otherwise it will show the false value  As it is boolean the value must be between false and true .

// Java program without Optional Class

public class OptionalDemo{

public static void main(String[] args) {

String[] words = new String[10];

String word = words[5].toLowerCase();

System.out.print(word);

}

}

Output :

Exception in thread "main" java.lang.NullPointerException

import java.util.Optional;

public class OptionalDemo{

    public static void main(String[] args) {

        String[] words = new String[10];

        Optional<String> checkNull =

                      Optional.ofNullable(words[5]);

        if (checkNull.isPresent()) {

            String word = words[5].toLowerCase();

            System.out.print(word);

        } else

            System.out.println("word is null");

    }

}

O/P- word is null

And the output of the code will be : Do Any Work implementation in the class Do More Work implementation in the interface

**PATH (For Windows Users)**

When we launch any program from command line prompt, operating system use PATH environment variable to locate executable programs. In other words, PATH maintains a list of directories for searching executable programs such as ".exe", ".bat" or ".com".

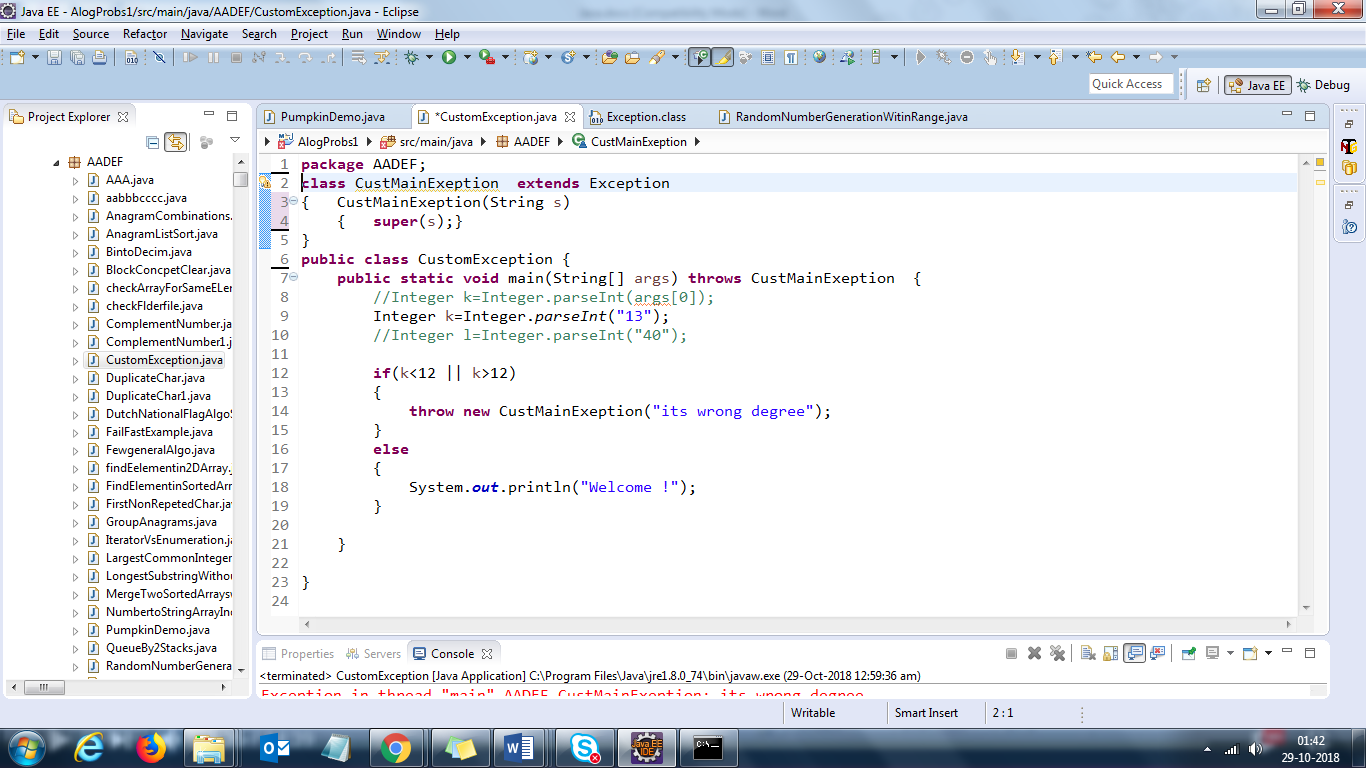
For Example: When you run command javac Hello.java or java Hello, Windows searches the program in the current working directory and all the directories listed in the PATH for Java Compiler "javac.exe" and Java Runtime "java.exe".

In case of failure it shows error given below:

PATH maintains a set of directories. The directories are separated by semi-colon ';'.

For Java applications, PATH must include JDK's "bin" directory (e.g., "c:\Program Files\java\jdk1.6.0\_xx\bin"), which contains JDK programs such as Java Compiler "javac.exe" and Java Runtime "java.exe".

**Custom Exception**



**Block Execution**

Static Block Parent Class

Static Block Child Class

Normal block Parent class

Constructor of parent class

Normal block Child class

Constructor of child class

Iterator<String> itr = al.iterator();

**while**(itr.hasNext())

{

**if**(itr.next()==3)

{

/\*Basically because that's how it's designed to

work. If you delete an element from the list,

the iterator doesn't know about it, and when

it tries to access another element in the list,

the list has changed and an error is raised.

But if you remove an element through the iterator,

then the itertor knows about the removal, makes

the appropriate adjustment to its data structures,

and continues.

\*/

// itr.remove();// No error

al.remove(3);//Error

// al.add(8);//Error fail fast

}

**Answer:**Member variables cannot be overridden. In other words, Variables are resolved at compile-time and methods at run-time.

**Member Variables**

**Question 7)**Guess the output of below code

|  |  |
| --- | --- |
|  | public class PumpkinDemo {        public static void main(String[] args) {          Shape s = new Circle();          System.out.println(s.name);      }  }    class Shape{      String name = "Shape";  }    class Circle extends Shape{      String name = "Circle";  } |

**Output:**

|  |  |
| --- | --- |
| 1 | Shape |

**Answer:**Member variables cannot be overridden. In other words, Variables are resolved at compile-time and methods at run-time.

**Exception Handling**

**Question 8)**Can overridden method throw different exception than the one being thrown in parent class method. For Example, Will below code compile successfully?

|  |  |
| --- | --- |
|  | import java.io.FileNotFoundException;  import java.io.IOException;    public class PumpkinDemo {        public static void main(String[] args) throws Exception{          Shape s = new Circle();          s.draw();      }  }    class Shape{      public void draw() throws IOException      {          System.out.println("Shape");      }  }    class Circle extends Shape{      public void draw() throws FileNotFoundException      {          System.out.println("Circle");      }  } |

**Answer:**

While overriding a method, you can compress the scope of checked exception but you cannot widen it. Also you can’not throw any other checked exception which is not being thrown in parent class method.

Here, FileNotFoundException is a child class of IOException. So, above code will compile successfully and it will give **Circle**as output.

If we change FileNotFoundException to generic Exception in above code, then it will give compile time error saying 'Exception Exception is not compatible with throws clause in Shape.draw()'

**Question 9)**Will below code compile successfully?

|  |  |
| --- | --- |
| \  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public class PumpkinDemo {        public static void main(String[] args){          Shape s = new Circle();          s.draw();      }  }    class Shape{      public void draw() throws ArithmeticException      {          System.out.println("Shape");      }  }    class Circle extends Shape{      public void draw() throws RuntimeException      {          System.out.println("Circle");      }  } |

**Answer:**Yes. Overridden methods can throw any RuntimeException irrespective of its scope unlike checked exception.

**Exception-handling-in-method-overriding-with-example/**

**Rule**:  An overriding method (the method of child class) can throw any [unchecked exceptions](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/), regardless of whether the overridden method (method of base class) throws exceptions or not. However the overriding method should not throw [checked exceptions](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) that are new or broader than the ones declared by the overridden method. The overriding method can throw those checked exceptions, which have less scope than the exception(s) declared in the overridden method.

**https://beginnersbook.com/2014/01/exception-handling-in-method-overriding-with-example/**

**Try block cant be standalone it should aligned with catch or finally at least one**

The **constructors** that can take arguments are called **parameterized constructors**.

# [How HashSet Works Internally In Java?](http://javaconceptoftheday.com/how-hashset-works-internally-in-java/)

**HashSet** uses HashMap internally to store its objects. Whenever you create a HashSet object, one **HashMap** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as **keys** of this HashMap object. The value associated with those keys will be a **constant**.

Whenever you insert an element into HashSet using **add()** method, it actually creates an entry in the internally backing HashMap object with element you have specified as it’s key and constant called “**PRESENT**” as it’s value. This “PRESENT” is defined in the HashSet class as below.

**Few Thump rules:**

* If two objects are same then they must return same value in hashcode() and equals() method whenever invoked.
* It is not necessary that two different object must have different hashcode values. it might be possible that they share common hash bucket...so hashcode is overriden

# [What's the meaning of System.out.println in Java?](https://stackoverflow.com/questions/3406703/whats-the-meaning-of-system-out-println-in-java)

System is a class, that has a public static field out. So it's more like

class System

{

public static PrintStream out;

}

class PrintStream

{

public void println ...

}

This is a slight oversimplification, as the PrintStream class is actually in the java.io package, but it's good enough to show the relationship of stuff.

**COllections vs COLLECTION**

**Collection is a base interface for most collection classes, whereas Collections is a utility class. I recommend you read the documentation.**

If so, the Collections class is a utility class having static methods for doing operations on objects of classes which implement the Collection interface. For example, Collections has methods for finding the max element in a Collection. Method like reverse, swap, add etc. 1) Collections.max()

2) Collections.min()

3) Collections.sort()

4) Collections.synchronizedCollection()

The Collection interface defines methods common to structures which hold other objects. List and Set are subinterfaces of Collection, and ArrayList and HashSet are examples of concrete collections.

The Collection interface contains methods that perform basic operations, such as int size(), boolean isEmpty(), boolean contains(Object element), boolean add(E element), boolean remove(Object element), and Iterator<E> iterator().

# What does it mean that "Javascript is asynchronous"?

JavaScript is not entirely asynchronous. Certain parts of the language are asynchronous. The difference between synchronous code and asynchronous code is that synchronous code executes from the top of a code block to the bottom in the order it was written. For example this block

We can write synchronous as well as asynchronous code in JavaScript.Usually ajax is used for writing asynchronous code.Callback functions along with ajax are used for implementing asynchronous functionality.You make ajax call to a server and when you get the response you call the callback function.

**Call-back function is a function which is passed as argument to other function:**

For example in the following code the callback function will be called after the ajax call:

function asyncMethod(callbackMethod) {

$.get(text.aspx", function(response) { callbackMethod(response); }); }

When you execute something synchronously, you wait for it to finish before moving on to another task. When you execute something asynchronously, you can move on to another task before it finishes.

JavaScript is not asynchronus. AJAX and Jquery are asynchronus. Asynchronus communication means that once the request is made, the thread doesn't wait for the response but rather involves itself in some other task. In case of AJAX / Jquery, multiple servers calls are made asynchronously and there are difference threads / processes that handles the response , as and when they come and update the page.

**Python Vs Java**

A language is **Statically typed** if the type of a variable is known at compile time. For some languages this means that you as the programmer must specify what type each variable is (e.g.: Java, C, C++, Scala); The main advantage here is that all kinds of checking can be done by the compiler, and therefore a lot of trivial bugs are caught at a very early stage.

**Dynamic** Typing: Perl is an **dynamic** typed **language**. Variables need not be initialized before they are used in code.

A language is **dynamically typed** if the type is associated with run-time values, and not named variables/fields/etc. This means that you as a programmer can write a little quicker because you do not have to specify types every time (unless using a statically-typed language with type inference). Example: Perl, Ruby, Python

**Python programs** are generally expected to run slower than Java programs, but they also take much less time to develop. Python programs are typically 3-5 times shorter than equivalent Java programs. This difference can be attributed to Python's built-in high-level data types and its dynamic typing.

For example, a Python programmer wastes no time declaring the types of arguments or variables, and Python's powerful polymorphic list and dictionary types, for which rich syntactic support is built straight into the language, find a use in almost every Python program. Because of the run-time typing, Python's run time must work harder than Java's. For example, when evaluating the expression a+b, it must first inspect the objects a and b to find out their type, which is not known at compile time. It then invokes the appropriate addition operation, which may be an overloaded user-defined method. Java, on the other hand, can perform an efficient integer or floating point addition, but requires variable declarations for a and b, and does not allow overloading of the + operator for instances of user-defined classes.

**Here are various thread pools in java:**

* Single Thread Executor : A thread pool with only one thread. So all the submitted tasks will be executed sequentially. Method : Executors.newSingleThreadExecutor()
* Cached Thread Pool : A thread pool that creates as many threads it needs to execute the task in parrallel. The old available threads will be reused for the new tasks. If a thread is not used during 60 seconds, it will be terminated and removed from the pool. Method : Executors.newCachedThreadPool()
* **Fixed Thread Pool : A thread pool with a fixed number of threads. If a thread is not available for the task, the task is put in queue waiting for an other task to ends. Method : Executors.newFixedThreadPool()**
* Scheduled Thread Pool : A thread pool made to schedule future task. Method : Executors.newScheduledThreadPool()
* Single Thread Scheduled Pool : A thread pool with only one thread to schedule future task. Method : Executors.newSingleThreadScheduledExecutor()

**Char to String**

**1.** String stringValueOf = String.valueOf('c'); // most efficient

2. String stringValueOfCharArray = String.valueOf(new char[]{x});

3. String characterToString = Character.toString('c');

4. String characterObjectToString = new Character('c').toString();

5. String concatBlankString = 'c' + "";

**Char to int**

**char** c='a';

1. **char** c2='1';
2. **int** a=c;
3. **int** b=c2;
4. **int** a=Integer.parseInt(String.valueOf(c));

**String to Integer**

String number = "10";

int result = Integer.parseInt(number);

int result = Integer.parseInt(number); 10;

String number = "10a";

int result = Integer.parseInt(number);

System.out.println(result); o/P java.lang.NumberFormatException: For input string: "10a"

**Integer to String**

String str1 = Integer.toString(a);

 int d = 1234;

    String str4 = new Integer(d).toString();

**String to Integer**

String k="8";

**int** j=Integer.*parseInt*(k);

Integr to String

**int** i=9;

1.String s1 = String.*valueOf*(i);

2. Integer obj = **new** Integer(i);

3. String str4 = obj.toString();

3. String str5 = **new** Integer(i).toString();

**int** i=7;

String k =Integer.*toString*(i);

int is a primitive data type while Integer is a Reference or Wrapper Type (Class) in Java.

after java 1.5 which introduce the concept of **autoboxing** and **unboxing** you can initialize both int or Integer like this.

int a= 9

Integer a = 9 // both valid After Java 1.5.

why Integer.parseInt("1"); but not int.parseInt("1"); ??

Integer is a Class defined in jdk library and parseInt() is a static method belongs to IntegerClass

They're two completely different things.

.length is a property on **arrays**. That isn't a method call.

.length() is a method call on String.

.size() if for collections;

**int** a[] = **new** **int**[5];

a[0] = 10;

String s ="Test";

System.***out***.println(s.length());5

**char**[] bc=s.toCharArray();

System.***out***.println(bc.length);

ArrayList v = **new** ArrayList();

v.add("a");

v.add(4.5);

v.add(5);

System.***out***.println(v.size());//3

Set v = **new** HashSet();

v.add("a");

v.add(4.5);

v.add(5);

System.***out***.println(v.size());//3

Given code will throw exception

Unreachable catch block for ArithmeticException. It is already handled by the catch block for Exception

**try**

{

**int** i=1/0;

}

**catch** (Exception e)

{

System.***out***.print("z");

}

**catch** (ArithmeticException ee)

{

System.***out***.print("z");

# }

# [Difference between wait() and sleep()](https://stackoverflow.com/questions/1036754/difference-between-wait-and-sleep)

**sleep()** is a method which is used to hold the process for few seconds or the time you wanted but in case of **wait()** method thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

The **wait**() is used for inter-thread communication while **sleep**() is used to introduce pause on execution, generally.

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.

Important Points

* Overloaded methods MUST change the argument list
* Overloaded methods CAN change the return type
* Overloaded methods CAN change the access modifier
* Overloaded methods CAN declare new or broader checked exceptions
* A method can be overloaded in the same class or in a subclass.

### Rules for method overriding:

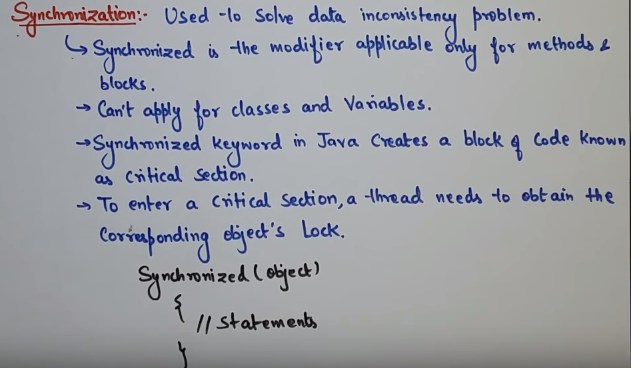
* In [java](http://crunchify.com/category/java-tutorials/), a method can only be written in Subclass, not in same class.
* The argument list should be exactly the same as that of the overridden method.
* The return type should be the same or a subtype of the return type declared in the original overridden method in the super class.
* The access level cannot be more restrictive than the overridden method’s access level. For example: if the super class method is declared public then the overridding method in the sub class cannot be either private or protected.
* It is possible to relax the restriction, but not to make it more restrictive:
* public abstract class A {
* protected void method();
* }
* public class B extends A {
* @Override
* public void method(){ // OK
* }
* }
* public class C extends A {
* @Override
* private void method(){ // not allowed
* }
* Instance methods can be overridden only if they are inherited by the subclass.
* A method declared final cannot be overridden.
* A [method declared static](http://crunchify.com/java-static-methods-variables-static-block-and-class-with-example/) cannot be overridden but can be re-declared.
* If a method cannot be inherited then it cannot be overridden.
* A subclass within the same package as the instance’s superclass can override any super class method that is not declared private or final.
* A subclass in a different package can only override the non-final methods declared public or protected.
* An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
* Constructors cannot be overridden

# Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.



### Points to remember for Synchronized block

* Synchronized block is used to lock an object for any shared resource.
* Scope of synchronized block is smaller than the method.

Collection : The root [interface](http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html) of [Java Collections Framework](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html).

Collections : A utility class that is a member of the [Java Collections Framework](https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html).

**static binding**

When type of the object is determined at compiled time(by the compiler), it is known as static binding. **Method overloading** is the best example of static binding

If there is any private, final or static method in a class, there is static binding.

**Dynamic binding**

When type of the object is determined at run-time, it is known as dynamic binding.

like overriding, **Method overriding** is the best example of **dynamic binding**.

**Abstract classes have constructors**

Yes when we define a **class** to be an **Abstract Class** it cannot be instantiated but that does not mean an**Abstract class** cannot **have** a **constructor**. Each **abstract class** must **have** a concrete subclass which will implement the **abstract** methods of that **abstract class**

Abstract classes have constructors and those constructors are always invoked when a concrete subclass is instantiated. We know that when we are going to instantiate a class, we always use constructor of that class. Now every constructor invokes the constructor of its **super class with an implicit call to super()**

equals method will throw compile time exception with primitive variable

Integer a=5;

Integer b=5

a.equals(b)= true;

a.==b, true;

**but if**

**int a=5;**

Int b=5

a.equals(b)--complietime exceptions;

**noclassdeffounderror vs classnotfoundexception java**

**ClassNotFoundException** is thrown when the class cannot be found in the classpath.

**NoClassDefFoundError** is an error that occurs when a particular class is present at compile time, but was missing at run time.

 NoClassDefFoundError indicates that the classloader (in this case java.net.URLClassLoader), which is responsible for dynamically loading classes, cannot find the .class file for the class that you're trying to use.

e.g This may also happen if your jar file is dependent on some other jar files for their woking and you have not added them in your project.

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

**Beware that if your static initialize block throws Exception than you may get**[**java.lang.NoClassDefFoundError**](http://javarevisited.blogspot.sg/2011/06/noclassdeffounderror-exception-in.html)**when you try to access the class which failed to load.**

package SelePracticePack;

import org.openqa.selenium.WebDriver;

import org.testng.Assert;

import org.testng.annotations.Test;;

public class SimpleCalculator

{

static int undefined = 1 / 0;

}

like i have war file which has

# Java String intern

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from pool memory, if it is created by new keyword.

String s1=**new** String("hello");

String s2="hello";

String s3=s1.intern();

//returns string from pool, now it will be same as s2

String s5=s2.intern();

String s4=s1.intern();

System.***out***.println(s1==s3); false

System.***out***.println(s4==s1);false

System.***out***.println(s1.equals(s4));

System.***out***.println(s1==s2);//false because reference is different

System.***out***.println(s2==s3);//true because reference is same

1. }}

String a = "abc"; // 1 Object: "abc" added to pool

String b = "abc"; // 0 Object: because it is already in the pool

String c = new String("abc"); // 1 Object

String d = new String("def"); // 1 Object + "def" is added to the Pool

String e = d.intern(); // (e==d) is "false" because e refers to the String in pool

String f = e.intern(); // (f==e) is "true"

**//Total Objects: 4 ("abc", c, d, "def").**

**Static and Instance variable/Method Access**

**Non static methods can access static/Non static variables.** Static methods can access only static variables or methods directly without creating object.ex:public static void main(String arg[])

* Instance methods can access instance variables and instance methods directly.
* **Instance methods can access class variables and class methods directly.**
* Class methods can access class variables and class methods directly.
* Class methods cannot access instance variables or instance methods directly—they must use an object reference. Also, class methods cannot use the this keyword as there is no instance for this to refer to.
* **Note:**Java is platform independent but jvm is platform dependent.

# [Why can other methods be “static” but a constructor cannot?](https://stackoverflow.com/questions/7780406/why-can-other-methods-be-static-but-a-constructor-cannot)

Constructor is used to create Objects.

Static is generally which is same for all objects.

So, if we have had static constructors creation of one object would affect all the other existing objects.

Static methods only reference to static variables. Therefore all the initial parameters which you are giving to create an object would change for all objects. It is no point creating similar objects for no use.

**Constructor** is the property of an object while **static** has nothing to do with object. That's why there is nothing like **static constructor**

# Marker interface in Java

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Colonnable and Remote interface. All these interfaces are empty interfaces.

public interface Serializable

{

// nothing here

}

Examples of Marker Interface which are used in real-time applications :

**Cloneable interface** : Cloneable interface is present in java.lang package. There is a method clone() in [Object](http://www.geeksforgeeks.org/object-class-in-java/) class. A class that implements the Cloneable interface indicates that it is legal for clone() method to make a field-for-field copy of instances of that class.

**Serializable interface** : Serializable interface is present in java.io package. It is used to make an object eligible for saving its state into a file. This is called [Serialization](http://quiz.geeksforgeeks.org/serialization-in-java/).  
Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializab

/\*Process of writing State of an object to File or in network supported form is called serialization

or from java supported form into either file supported form or Netwrok supported form

we can write binary data in file but not object diectly so we use

ObjectOutPutStream

Static Variable. Static variables belong to a class and not to any individual instance.

The concept of serialization is concerned with the object's current state. Only data

associated with a specific instance of a class is serialized, therefore static member fields are

ignored during serialization.Dec 29, 2013

e

\*/

**Remote interface** : Remote interface is present in java.rmi package. A remote object is an object which is stored at one machine and accessed from another machine. So, to make an object a remote object, we need to flag it with Remote interface. Here, Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine.Any object that is a remote object must directly or indirectly implement this interface. RMI ([Remote Method Invocation](http://www.geeksforgeeks.org/remote-method-invocation-in-java/)) provides some convenience classes that remote object implementations can extend which facilitate remote object creation.

# StringTokenizer

In Java, you can StringTokennizer class to split a String into different tokenas by defined delimiter.(space is the default delimiter). Here’re two StringTokennizer examples :

public static void main(String[] args) {

String str = "This is String , split by StringTokenizer, created by mkyong";

StringTokenizer st = new StringTokenizer(str);

System.out.println("---- Split by space ------");

while (st.hasMoreElements()) {

System.out.println(st.nextElement());

}

System.out.println("---- Split by comma ',' ------");

StringTokenizer st2 = new StringTokenizer(str, ",");

while (st2.hasMoreElements()) {

System.out.println(st2.nextElement());

}

}

}

*Output*

This

is

String

,

split

by

StringTokenizer,

created

by

mkyong

---- Split by comma ',' ------

This is String

split by StringTokenizer

created by mkyong

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Iterator vs Enumeration 1.Only major difference between Enumeration and iterator is Iterator has a remove() method  while Enumeration doesn't.  2. Also **Iterator**is more secure and safe as compared to **Enumeration**because it  does not  Allow other thread to modify the collection object while some thread is iterating over it and throws **ConcurrentModificationException.** 3. Iterator also allows you to remove elements from collection during traversal but Enumeration doesn't allow that, it doesn't got the remove () method.  4) Enumeration is a legacy interface used to traverse only the legacy classes like Vector, HashTable and Stack. Where as Iterator is not a legacy code which is used to traverse most of the classes in the collection framework. For example, ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet, HashMap, LinkedHashMap, TreeMap etc. 5) Iterator is a fail-fast in nature. i.e it throws ConcurrentModificationException if a collection is modified while  iterating other than it’s own remove() method. Where as Enumeration is fail-safe in nature. It doesn’t throw any exceptions if a collection is modified while iterating.          Iterator<String> itr = myList.iterator();          while(itr.hasNext()){              System.out.println(itr.next());          }  **Map** is three **collections**: Keys, values and key-value pairs.  It's **part** of the **collection** framework but it doesn't implement the  java.util.**Collection interface**. ... The **Map interface** is **not** an extension  of the **Collection interface**. However, it is a structure for keeping **collections**  **Wrapper class**  Java Enum  **Enum in java** is a data type that contains fixed set of constants.  It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.  Java Enums can be thought of as classes that have fixed set of constants.   1. **class** EnumExample1{ 2. **public** **enum** Season { WINTER, SPRING, SUMMER, FALL } 4. **public** **static** **void** main(String[] args) { 5. **for** (Season s : Season.values()) 6. System.out.println(s); 8. }}  Can Enum extend any class in Java?  |  | | --- | | **Answer:** | | Enum can not extend any class in java, the reason is by default, Enum extends abstract base class java.lang.Enum. Since java does not support multiple inheritance for classes, Enum can not extend another class. |   - See more at: http://www.java2novice.com/java\_interview\_questions/can-enum-extend/#sthash.31eLEm5T.dpuf 4) Why Map interface does not extend Collection interface? A good answer to this interview question is “**because they are incompatible**“. Collection has a method add(Object o). Map can not have such method because it need key-value pair. There are other reasons also such as Map supports keySet, valueSet etc. Collection classes does not have such views.  All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.  **Can constructor be final**  So making a **constructor final** is not required, so it is not used with **constructor**. When you  set a method as **final**, it means : "You don't want any class override it", but **constructor**  by JLS(**Java Language Specification** ) definition **can**'t overridden, so it is clean.  We **can**'t make **constructor** to be **final** because **Constructor** is never inherited.  **Why can't constructors be final, static, or abstract in Java?**  **When you set a method as final it means:** "You don't want any class override it." But the constructor (according to the Java Language Specification) can't be overridden, so it is clean.  **When you set a method as abstract it means**: "The method doesn't have a body and it should be  Implemented in a child class." But the constructor is called implicitly when the new keyword is  Used so it can't lack a body.  **When you set a method as static it means**: "The method belongs to the class, not a particular  Object." But the constructor is implicitly called to initialize an object,  So there is no purpose in having a static constructor.  **Access Private variable and set the value(By Reflection)** [What does Class<?> mean in Java?](https://stackoverflow.com/questions/9921676/what-does-class-mean-in-java) Class is a parametrizable class, hence you can use the syntax Class<T> where T is a type. By writing Class<?>, you're declaring a Class object which can be of any type (? is a wildcard). The Class type is a type that contains metainformation about a class.  Class<?> clazz = TestSetPrivate.**class**;  Object cc = clazz.newInstance();  Field[] fld = TestSetPrivate.**class**.getDeclaredFields(); // for axis the fields  **for**(Field fld1:fld)  {  fld1.setAccessible(**true**);  System.***out***.println("field :"+fld1.getName());  System.***out***.println("type :"+fld1.getType());  System.***out***.println("value :"+fld1.get(**new** TestSetPrivate()));  **if**(fld1.getName().equals("strn"))  {  fld1.setAccessible(**true**);  fld1.set(cc, "Mohan");  String str1 = (String) fld1.get(cc);  System.***out***.println("new value: " + str1);      }    }  **To achieve encapsulation in Java −**   * Declare the variables of a class as private. * Provide public setter and getter methods to modify and view the variables values.   Abstract Vs. Interface  Difference between the Two What is an Abstract Class? An abstract class is a special kind of class that cannot be instantiated. So, why we need a class  that cannot be instantiated? An abstract class is only to be inherited from. In other words,  it only allows other classes to inherit from it but cannot be instantiated. The advantage is that  it enforces certain common behavior or properties across multiple classes who inherit the  abstract class.  It contains non abstract and Abstract methods both What is an Interface? An interface is an entity that is defined by the word Interface. An Interface only contains signature  of the methods whose implementation is to be provided by the classes who implements that Interface.it can’t be instantiated Difference Between Interface and Abstract Class  1. Main difference is methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behavior. 2. Variables declared in a Java interface is by default final. An abstract class may contain   non-final variables.   1. Members of a Java interface are public by default. A Java abstract class can have the   usual flavors of class members like private, protected, etc..   1. Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”. 2. An interface can extend another Java interface only, an abstract class can extend another   Java class and implement multiple Java interfaces.   1. A Java class can implement multiple interfaces but it can extend only one abstract class. 2. Interface is absolutely abstract and cannot be instantiated; A Java abstract class   also cannot be instantiated, but can be invoked if a main() exists.   1. In comparison with java abstract classes, java interfaces are slow as it   requires extra indirection. [Why can't an interface have constructors and destructors?](http://stackoverflow.com/questions/7226493/why-cant-an-interface-have-constructors-and-destructors)Interfaces are contracts, not implementations, so no need to construct or destroy them. You onlyConstruct and destroy concrete types which could implement interfaces. and similar to Abstract classes we are unable to create Object to Interface.  <http://www.allapplabs.com/interview_questions/java_interview_questions_2.htm>  **What is the difference between a constructor and a method?**  A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the new operator. A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator  Method is inheritable but constructor not   |  |  |  |  | | --- | --- | --- | --- | |  |  | | | |  | \*\*\*\*\*  Overloading vs overriding  Q: **Method Overloading?**  Ans:  Method overloading means having two or more methods with the same name but different signatures in the same scope. These two methods may exist in the same class or anoter one in base class and another in derived class.   Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.  Important Points   * Overloaded methods MUST change the argument list * Overloaded methods CAN change the return type * Overloaded methods CAN change the access modifier * Overloaded methods CAN declare new or broader checked exceptions * A method can be overloaded in the same class or in a subclass   The reason is that overloads in Java are only allowed for methods with different *signatures*.  The return type is not part of the method signature, hence cannot be used to distinguish overloads.  **Constructor**   1. They should be declared in the public section. 2. They are invoked automatically when the object are created. 3. They do not have return types, not even void and therefore, and they cannot return values. 4. They cannot be inherited, though a derived class can call the base class constructor. 5. Like other c++ functions, they can have default arguments. 6. Destructor cannot be virtual. 7. We cannot refer to their addresses. 8. An object with a destructor (or constructor) cannot be used as a member of a union. 9. They make implicit calls to the operators new and delete when memory allocation or de-allocation is required.   New Operator | | | |  | |  |   **Interface** is not a class, it is a set of rules, and cannot be instantiated, then it cannot contain any  volatile data container inside. Only constants can be set inside of interface, although it is  discouraged, because declaring constants in interfaces [violates encapsulation approach](http://en.wikipedia.org/wiki/Constant_interface).  **Interface variables are static because Java interfaces cannot be instantiated** in their own  right; the value of the variable must be assigned in a static context in which no instance exists. The final modifier ensures the value assigned to the **interface variable is a true constant that cannot be re-assigned by program code.**and have in mind that an interface is used to show 'what' your going to have to implement not how to. so the variables should be final ( cause the non-static variables are not related to the whole specification of a class ). | |  |   War-So a War is a .jar, but it contains web application components and is laid out according to a specific structure. A .war is designed to be deployed to a web application server such as Tomcat or Jetty or a Java EE server such as JBoss or Glassfish.  A .war file has a specific structure in terms of where certain files will be. Other than that, yes, it's just a .jar.  Jar-A .war file is a Web Application Archive which runs inside an application server while a .jar is  Java Application Archive that runs a desktop application on a user's machine.  The main difference between a JAR and a WAR is that a JAR is not really a single purpose  format, while a WAR is. With a JAR file, you can package lots of different things. Usually you  package up Java classes, but you could put pretty much anything into a JAR file, and you can  create JARs that have whatever files and directory formats you want. For instance, Oracle  (at least in 8.1.x) actually uses JAR files to distribute the Oracle software on the install CDs,  and IBM uses them to distribute an installable version of the documentation for WebSphere.  Technically you can put JSP pages, HTML pages, servlet classes, etc. in a JAR, but you  generally don't -- you use a WAR.  WAR files are JAR files with a specific format. This format is defined in the servlet spec (which you can look at on the java.sun.com site). Essentially, the WAR file is a standard format for web applications that has specific directories and specific files. This includes a WEB-INF directory, a WEB-INF/web.xml file used to describe the application, a WEB-INF/lib directory for JAR files used by the application, and a WEB-INF/classes directory for class files that aren't distributed in a JAR. You would put the pages (JSPs and HTML) in the WAR as well. Then, you can distribute your application as one file, instead of as a collection of images, HTML pages, and Java classes.  2. What do you mean by platform independence?  Platform independence means that we can write and compile the java code in one platform (eg Windows) and can execute the class in any other supported platform eg (Linux,Solaris,etc).  3. What is a JVM?  JVM is Java Virtual Machine which is a run time environment for the compiled java class files.    First, let's have a clear idea of the following terms  Javac is Java Compiler -- Compiles your Java code into **Bytecode**  JVM is Java Virtual Machine -- Runs/ Interprets/ translates Bytecode into **Native Machine Code**  JIT is Just In Time Compiler -- Compiles the given bytecode instruction sequence to machine code at **runtime** before executing it natively. It's main purpose is to do heavy optimizations in performance.  **So now, Let's find answers to your questions..**  1)JVM: is it a compiler or an interpreter? -- **Ans:** Interpreter  2)what about JIT compiler that exist inside the JVM? -- **Ans:** If you read this reply completly, you probably know it now  3)what exactly is the JVM? -- **Ans:**   * JVM is a virtual platform that resides on your RAM * Its component, **Class loader** loads the .class file into the RAM * The **Byte code Verifier** component in JVM checks if there are any access restriction violations in your code. (This is one of the principle reasons why java is secure) * Next, the **Execution Engine** component converts the Bytecode into executable machine code   Hope this helped you..  4. Are JVM's platform independent?  ***JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.***  5. What is the difference between a JDK and a JVM?  JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.  JDK = JRE + JVM.  **JRE (java run-time environment)**  It's needed to run Java programs. You can't compile Java programs with it .  For example: a regular computer user who wants to run some online games then will need JRE in his system to run Java programs.  **JVM (java virtual machine)**  As you might know it run the bytecodes. It make Java platform independent because it executes the.class file which you get after you compile the Java program regardless of whether you compile it on Windows, Mac or Linux.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | |  | | --- | |  |   7. What is the base class of all classes?  java.lang.Object  9. Is Java a pure object oriented language?  **Java uses primitive data types** and hence is not a pure object oriented language   |  | | --- | |  |   13. What are instance variables?  Instance variables are those which are defined at the class level. Instance variables need not be  initialized before using them as they are automatically initialized to their default values.  **1.** An instance variable is *one per Object*, every object has its **own copy of instance variable.**  public class Test{  int x = 5;  }  Test t1 = new Test();  Test t2 = new Test();  Both t1 and t2 will **have its own copy of** x.  **2.** A static variable is *one per Class*, **every object of that class shares the same Static variable.**  15. Should a main() method be compulsorily declared in all java classes?  No not required. main() method should be defined only if the source class is a java application.  18. What is the arguement of main() method?  main() method accepts an array of String object as arguement. | | |  | | |  | | --- | |  |   19. Can a main() method be overloaded?  Yes. You can have any number of main() methods with different method signature and  implementation in the class.  yes **we can overload main method**. . Yes, **main method can** be **overloaded**. **Overloaded main method** has to be called from inside the "public static void**main**(String args[])" as this is the entry point when the  20. Can a main() method be declared final?  Yes. Any inheriting class will not be able to have its own default main() method.  21. Does the order of public and static declaration matter in main() method?  No. It doesn't matter but void should always come before main().  23. What is a package?  Package is a collection of related classes and interfaces. Package declaration should be first statement in a java class.  24. Which package is imported by default?  java.lang package is imported by default even without a package declaration.  26. Can a class be declared as protected?  A class can't be declared as protected. only methods can be declared as protected.  <http://way2java.com/packages/access-specifiers-accessibility-permissions-restrictions/>  . Fields, methods and constructors declared protected in a superclass can be accessed only by subclasses in other packages.  odifier | Class | Package | Subclass | World  ————————————+———————+—————————+——————————+———————  public | ✔ | ✔ | ✔ | ✔  ————————————+———————+—————————+——————————+———————  protected | X | ✔ | ✔ | ✘  ————————————+———————+—————————+——————————+———————  no modifier | ✔ | ✔ | ✘ | ✘  ————————————+———————+—————————+——————————+———————  private | ✔ | ✘ | ✘ | ✘  27. What is the access scope of a protected method?  A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.  31. Can you give few examples of final classes defined in Java API?  java.lang.String, java.lang.Math are final classes.  32. 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.  When **JVM** trying to delete the objects, then some objects refuse to delete if they held any resource. If our object opened up some resources, and we woluld like to close them before our object is deleted. So before **Garbage** operation we have to clean up the resources which the object held on, that clean up code we have to put in ***finalize()*** method.  **Finalization *was* meant to clean up resources acquired by the object (not memory, but other resources, e.g. file handles, ports, DB connections etc.). However, it did not really work out :-(**   * **it is unpredictable when finalize() will be called** * in fact, there is no guarantee that finalize() will be called ever!   49. Why is an Interface be able to extend more than one Interface but a Class  Can’t extend more than one Class?  Basically Java doesn't allow multiple inheritance, so a Class is restricted to extend only one Class. But an Interface is a pure abstraction model and doesn't have inheritance hierarchy like classes(do remember that the base class of all classes is Object). So an Interface is allowed to extend more than one Interface. Why Java doesn't support multiple inheritanceFirst reason is ambiguity around Diamond problem, consider a class A has foo() methodand then B and C derived from A and has there own foo() implementation and now class Dderive from B and C using multiple inheritance and if we refer just foo() compiler will not beable to decide which foo() it should invoke. This is also called Diamond problem because structure on this inheritance scenario is similar to 4 edge diamond, see below              A foo()             / \            /   \     foo() B     C foo()            \   /             \ /              D             foo()  In my opinion even if we remove the top head of diamond class A and allow multiple inheritances we will see this problem of ambiguity.  51. Can a class be defined inside an Interface?  Yes it's possible.  52. Can an Interface be defined inside a class?  Yes it's possible.  61. What value does read() return when it has reached the end of a file?  The read() method returns -1 when it has reached the end of a file.  62. Can a Byte object be cast to a double value?  No, an object cannot be cast to a primitive value.  m21  A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.  66. When can an object reference be cast to an interface reference?  An object reference be cast to an interface reference when the object implements the referenced  interface.  Byte b=(**double**) 5000;//not possible  **double** c=(**byte**) 599;/possble  In other words….An interface reference can point to any object of a class that implements this  interface i.e. see the example below:  interface Foo{ void display(); }  public class TestFoo implements Foo{  void display(){ System.out.println(“Hello World”); }  public static void main(String[] args){ Foo foo = new TestFoo(); foo.display(); }  }  **Thread and a Process**  The processes and threads are independent sequences of execution,  the typical difference is that threads run in a shared memory space,  while processes run in separate memory spaces.  The typical difference is that threads (of the same process) run in a shared memory space,  1) A program in execution is often referred as process. A thread is a subset(part) of the process.  2) A process consists of multiple threads. A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.  3) A process is sometime referred as task. A thread is often referred as lightweight process.  4) A process has its own address space. A thread uses the process’s address space and share it with the other threads of that process.  6) A thread can communicate with other thread (of the same process) directly by using methods like wait(), notify(), notifyAll(). A process can communicate with other process  by using [**inter-process communication**](http://en.wikipedia.org/wiki/Inter-process_communication).  8) Threads have control over the other threads of the same process. A process does not  have control over the sibling process, it has control over its child processes only.  Threads are sometimes called *lightweight processes*. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process.  Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.  67. Which class is extended by all other classes?  The Object class is extended by all other classes.  68. Which non-Unicode letter characters may be used as the first character of an identifier?  The non-Unicode letter characters $ and \_ may appear as the first character of an identifier  70. What is casting?  There are two types of casting, casting between primitive numeric types and casting between object references. Casting between numeric types is used to convert larger values, such as double values, to smaller values, such as byte values. Casting between object references is used to refer to an object by a compatible class, interface, or array type reference.  Casting is a method of assigning a value of one type to a variable of a more specific type.   Here is an example: String extends Object, making "String" the more specific type (subclass).   Object iAmAString = "avalue";  String str = (String)iAmAString;   The (String) in parenthesis is how you perform the cast from Object to String. It basically assures the compiler that you, as the programmer, know that the actual value extends the expected type.   If iAmAString's value was not actually a string, Java would throw a ClassCastException at runtim [Java does not support private/protected inheritance like C++?](http://programmers.stackexchange.com/questions/190918/why-java-does-not-support-private-protected-inheritance-like-c) Protected is specifier which is only used for members , methods and variable not for classes and interfaces 80. Can an anonymous class be declared as implementing an interface and extending a class?  An anonymous class may implement an interface or extend a superclass, but may not be declared to do both.  **85. Is null a keyword?**  The null value is not a keyword. its literals  *90. Does a class inherit the constructors of its superclass?*  ***A class does not inherit constructors from any of its superclasses.***  A **constructor from a subclass can *call* constructors of superclass, but they're not inherited as such. only memebers inherited and costrutor is not member of class**  91. Name the eight primitive Java types.  The eight primitive types are byte, char, short, int, long, float, double, and boolean.  93. What is the difference between a while statement and a do while statement?  A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do while statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do whilestatement will always execute the body of a loop at least once.  100. What is the diffrence between inner class and nested class?  When a class is defined within a scope od another class, then it becomes inner class. If the access modifier of the inner class is static, then it becomes nested class.  All below method are different means all are overloaded  **public** **static** **void** tet(**int** a, **int** b)  {  System.***out***.println(a-b);  }  **public** **static** **void** tet(**float** a, **float** b)  {    }  **public** **static** **void** tet(Integer a, Integer b)  {  System.***out***.println(a+b);  }    Like Below Case Not possible  **public** **void** ovTest(**int** a,**int** b)  {}    **public** **int** ovTest(**int** a,**int** b)  { return a; }  And give is Possible  **public** **void** ovTest(**int** a,**int** b)  {}    **public** **int** ovTest(**int** a)  { return a; }  115. What is constructor chaining and how is it achieved in Java ?  Calling another constructor in the same class from another constructor is called constructor chaining. By using this() we can call another constructor in the same class. Incase we want to call another constructor, this() should be the first line in the constructor. Below example shows code for constructor chaining. Java Constructor Chaining Sample Code  |  | | --- | | **Code:** | | [?](http://www.java2novice.com/java_constructor_examples/constructor_chaining/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package com.myjava.constructors;    public class MyChaining {        public MyChaining(){          System.out.println("In default constructor...");      }      public MyChaining(int i){          this();          System.out.println("In single parameter constructor...");      }      public MyChaining(int i,int j){          this(j);          System.out.println("In double parameter constructor...");      }        public static void main(String a[]){          MyChaining ch = new MyChaining(10,20);      }  } | |  Example Output In default constructor...  In single parameter constructor...  In double parameter constructor...  120. Can a for statement loop indefinitely?  Yes, a for statement can loop indefinitely. For example, consider the following: for(;;);  **public** **static** **void** main(String[] args) {    **for**(;;)  System.***out***.println("Hello");    The default value of an String type is null.  Marker interface in Java is interfaces with no field or methods or in simple word empty interface in java is called marker interface.  Serialization   |  |  | | --- | --- | | **Serialization** is the process of converting an object into a stream of bytes in order to store the  object or transmit it to memory, a database, or a file. Its main purpose is to save the state of  an object in order to be able to recreate it when needed. The reverse process is called deserialization.  Image result What is the need of Serialization?  * To send the state of an object over the network. * To save the state of an object in a file. * The state of an object needs to be manipulated as a stream of bytes.    Since the static variables are class variables and not instance variable so static  variables are not saved as part of the serialization.  **Always remember that static variable directly related to class and not the objects , hence as variable a does not belong to object, it does not get serializable**  The value of a static field is not touched by serialization or deserialization. Hence whatever value the class has for the static field that will reflect on the deserialized instance.  Let us analyze the situation as you went to a super market and ordered some thing. After the purchase, the cashier summarized the amount and you have not so much of amount at that time. But you can take it from an ATM counter (I am saying a situation where credit/debit card payments not possible. Only ready cash want to be paid.). So you just leave the shop for taking the money and the shopkeeper want to keep your bill as a pending one and should deal with the billing of other customers. So keeping a bill or any thing as a pending one is a serious issue and will slow the process. You can keep the items in memory and can also easily forget them when a power failure occurred. So every thing should be persists in a file form is the best option. | | |  |  |   126. What is a transient variable?  Transient variable is a variable that may not be serialized.  128. What is the difference between the >> and >>> operators?  The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have  been shifted out.  **can we define class inside interface java?** A class which is defined inside an interface, doesn't have to be declared as public andstatic but it then it is implicitly public and static. **interface** myInterface  {  **int***i* = 0;  **void** m2();  **public class** aClass  {  **public** aClass()  {  System.*out*.println("constructor of aClass = " + *i*);  }    //void m2(){};    **void** m4()  {  System.*out*.println("i m m4");    }  }  }  /\*class bClass extends myInterface.aClass  {  void m2()  {  System.out.println("interface method of m2 = " );  }  } \*/  **Public class** classinsideinterfaces **implements** myInterface  {  **publicvoid** m2()  {  System.*out*.println("interface method of m2 = " );  }  **publicstaticvoid** main(String[] arg) {  myInterface.aClass innerObj = **new** myInterface.aClass();  classinsideinterface obj = **new** classinsideinterface();  obj.m2();  }}  **A class method that implements an interface is always public. But what is a better**  **Convention?**  Two Points About Interface   1. All the Methods Inside the interfaces are public .  2. All the variables inside the interfaces are by default public static final .  Protected accessibility means that the particular entity is accessible only within the package. This will not be available in another package.  <http://stackoverflow.com/questions/15939002/protected-access-modifier-in-java>   |  |  |  | | --- | --- | --- | | |  | | --- | | publicclassMyBase {      publicintx;      publicvoidshow() {          System.out.println("x ="+ x);  } | |  | | |  |  | | --- | --- | | 1  2  3  4  5  6  7 | classMyDerived extendsMyBase {      publicinty;      publicvoidshow() {          System.out.println("x = "+ x);          System.out.println("y = "+ y);      }  } | |  |   Type safe code can access only the memory locations that it has permission to execute.  Type safe code can never access any private members of an object.  Type safe code ensures that objects are isolated from each other and are therefore safe  for inadvertent or malicious corruption. CLR performs a mandatory type safety check,  called verification [What is meant by immutable?](http://stackoverflow.com/questions/279507/what-is-meant-by-immutable) Immutable means that once an object of that Class has been created it can't be altered. Difference between String and StringBuffer/StringBuilder in Java Well, the most important difference between String and StringBuffer/StringBuilder in java is that **String object is immutable** whereas **StringBuffer/StringBuilder objects are mutable.**  By immutable, we mean that the value stored in the String object cannot be changed. Then the next question that comes to our mind is “If String is immutable then how am I able to change the contents of the object whenever I wish to?” . Well, to be precise it’s not the same String object that reflects the changes you do. Internally a new String object is created to do the changes.  So suppose you declare a String object:  String myString = “Hello”;  Next, you want to append “Guest” to the same String. What do you do?  myString = myString + ” Guest”;  When you print the contents of myString the output will be “Hello Guest”. Although we made  use of the same object(myString), internally a new object was created in the process. So,  if you were to do some string operation involving an append or trim or some **Finally,**  **whats the difference between StringBuffer and StringBuilder?**  StringBuffer and StringBuilder have the same methods with one difference and that’s  of synchronization. StringBuffer is synchronized( which means it is thread safe and  hence you can use it when you implement threads for your methods) whereas  StringBuilder is not synchronized( which implies it isn’t thread safe).  StringBuffer came in 1,0 and Stringbuilder came in 1. 5 java  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.    **What is serialization?**  When we want to transport an object through network then we have to convert the object into a stream of bytes. The process of converting an object into a stream of bytes is called Serialization [What is thread Safe in java](http://stackoverflow.com/questions/6324085/what-is-thread-safe-in-java)- As others have pointed out, thread safety means that a piece of code will work without errors if it's  used by more than one thread at once. Thread safety is a little bit more than just making sure  your shared data is accessed by only one thread at a time. You have to ensure sequential access  to shared data, while at the same time avoiding [race conditions](http://en.wikipedia.org/wiki/Race_condition), [deadlocks](http://en.wikipedia.org/wiki/Deadlock), [livelocks](http://en.wikipedia.org/wiki/Livelock#Livelock),  and [resource starvation](http://en.wikipedia.org/wiki/Resource_starvation). . Thread Safety in Java Thread safety in java is the process to make our program safe to use in multithreaded  Environment, there are different ways through which we can make our program thread safe.   * Synchronization is the easiest and most widely used tool for thread safety in java. * Use of Atomic Wrapper classes from *java.util.concurrent.atomic* package. For example AtomicInteger * Use of locks from *java.util.concurrent.locks* package. * Using thread safe collection classes, check this post for usage of   [ConcurrentHashMap](https://www.journaldev.com/122/java-concurrenthashmap-example-iterator) for thread safety.   * Using volatile keyword with variables to make every thread read the data from   memory, not read from thread cache.  privateint myInt =0;  publicintAddOne()  {  int tmp = myInt;  tmp = tmp +1;  myInt = tmp;  return tmp;  }  Now thread A and thread B both would like to execute AddOne(). but A starts first and reads the value of myInt (0) into tmp. Now for some reason the scheduler decides to halt thread A and defer execution to thread B. Thread B now also reads the value of myInt (still 0) into it's own variable tmp. Thread B finishes the entire method, so in the end myInt = 1. And 1 is returned. Now it's Thread A's turn again. Thread A continues. And adds 1 to tmp (tmp was 0 for thread A). And then saves this value in myInt. myInt is again 1.  So in this case the method AddOne was called two times, but because the method was not implemented in a thread safe way the value of myInt is not 2, as expected, but 1 because the second thread read the variable myInt before the first thread finished updating it.   * **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class. * **Class -** A class can be defined as a template/blue print that describes the behaviors/states that object of its type support. | |  |  | |  |  |  |  |   What is thread? A thread is an independent path of execution in a system.  **What is multi-threading?**  Multi-threading means various threads that run in a system.  **How does multi-threading take place on a computer with a single CPU?**  The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks execute sequentially.  When an object is not initialized, the default value is null  16-bit and larger bit patterns.  **What are wrapped classes?**  Wrapped classes are classes that allow primitive types to be accessed as objects.  **What is the purpose of finalization?**  The purpose of finalization is to give an unreachable object the opportunity to perform any  clean-up processing before the object is garbage collected.  **How many methods in Object class?**  This question is not asked to test your memory. It tests you how well you know Java. 11 in total plus constructor clone()  equals() & hashcode()  getClass()  finalize()  wait() & notify()  toString()  Here are the few other Checked Exceptions –  Which come at compile time   * SQLException * IOException(FileNotFound,EOF Expetion) * DataAccessException * ClassNotFoundException * InvocationTargetException   Unchecked - except compile time exception all are Unchecked exception   * NullPointer Exception * IndexOutOfBounds Exception * Arithmetic Exception * IllegalArgument Exception * VM Errors - Like StackOverFlow,OutOfMemoryError * Linkage Error   Throw vs Throws  Throws- Used for Suppress The error  Public void MethodA() Throws IOExpetion{}  What is the purpose of the System class?The purpose of the System class is to provide access to system resources.  ***Main difference between throw and throws* is** in there usage and functionality. where throws  is used in method signature to declare Exception possibly thrown by any method, throw is  actually used to throw Exception in Java code, here is an example of both throw and throws keyword which makes it easy to understand difference between them.  The keyword **throw** is used inside method body to invoke an exception and **throws clause** is used in method declaration (signature).  If we see syntax wise than **throw** is followed by an instance variable and **throws** is followed by exception class names.  **Throw:**  static{  try {  throw new Exception("Something went wrong!!");  } catch (Exception exp) {  System.out.println("Error: "+exp.getMessage());  }  }  ....  Throws:  public void sample() throws ArithmeticException{  //Statements  .....  //if (Condition : There is an error)  ArithmeticException exp = new ArithmeticException();  throw exp;  ...  }  public void **shutdown**() throws **IOException**{          throw new IOException("Unable to shutdown");   }  Usage of super Keyword  1. super is used to refer immediate parent class instance variable. 2. super() is used to invoke immediate parent class constructor. 3. super is used to invoke immediate parent class method. 4. **class** Vehicle{ 5. **int** speed=50; 6. } 8. **class** Bike **extends** Vehicle{ 9. **int** speed=100; 11. **void** display(){ 12. System.out.println(speed);//will print speed of Bike 13. } 14. **public** **static** **void** main(String args[]){ 15. Bike b=**new** Bike(); 16. b.display(); 18. } 19. }   Output:100  In the above example Vehicle and Bike both class have a common property speed.  Instance variable of current class is refered by instance bydefault, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class instance variable.  Solution by super keyword  //example of super keyword  class Vehicle  {  int speed=50;  }  class Bike extends Vehicle{  int speed=100;  void display(){  System.out.println(super.speed);//will print speed of Vehicle now  }  public static void main(String args[])  {  Bike b=new Bike();  b.display();  }  }  Output:50 VM  |  | | --- | | JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime  Environment in which java bytecode can be executed. | | JVMs are available for many hardware and software platforms (so JVM is plateform dependent). |  JRE  |  | | --- | | JRE stands for Java Runtime Environment. It is the implementation of JVM and physically exists. |  JDK  |  | | --- | | JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development  tools.  Java Development Kit (JDK) is a program development environment for writing [Java](http://searchsoa.techtarget.com/definition/Java)[applet](http://searchsoa.techtarget.com/definition/applet)s and  applications. It consists of a runtime environment that "sits on top" of the[operating system](http://searchcio-midmarket.techtarget.com/definition/operating-system) layer as  well as the tools and programming that developers need to compile, debug, and run applets and  applications written in the Java language. |   **JVM vs JRE**  The **JRE** is the environment within which the virtual machine runs.  **JRE** is the container, **JVM** is the content.  Java Runtime Environment contains **JVM**,  class libraries, and other supporting files. It does not contain any development tools  such as compiler, debugger, etc   *JDBC*  a [Java](http://www.webopedia.com/TERM/J/Java.html) [API](http://www.webopedia.com/TERM/A/API.html) that enables Java [programs](http://www.webopedia.com/TERM/P/program.html)to execute [SQL](http://www.webopedia.com/TERM/S/SQL.html) [statements](http://www.webopedia.com/TERM/S/statement.html). This allows Java programs to interact with any SQL-compliant [database](http://www.webopedia.com/TERM/D/database.html). Since nearly all [relational database management systems (DBMSs)](http://www.webopedia.com/TERM/R/relational_database.html) support SQL, and because Java itself runs on most[platforms](http://www.webopedia.com/TERM/P/platform.html), JDBC makes it possible to write a single database application that can run on different platforms and interact with different [DBMSs](http://www.webopedia.com/TERM/D/database_management_system_DBMS.html).  JDBC is similar to [ODBC](http://www.webopedia.com/TERM/O/ODBC.html), but is designed specifically for Java programs, whereas ODBC is language-independent.  **Applet** -- An applet is a little [application](http://searchsoftwarequality.techtarget.com/definition/application). Prior to the World Wide Web, the built-in writing and drawing programs that came with Windows were sometimes called "applets." On the Web, using [Java](http://searchsoa.techtarget.com/definition/Java), the [object-oriented programming](http://searchsoa.techtarget.com/definition/object-oriented-programming) language, an applet is a small program that can be sent along with a Web page to a user. Java applets can perform interactive animations, immediate calculations, or other simple tasks without having to send a user request back to the server.  **Default Behaviour**  class MyClass   // package private  {     int field;    // package private field     void calc() {  // package private method     }  }  interface MyInterface  // package private  {     int field1;         // static final public     void method1();     // public abstract  } [Implemeting 2 interfaces in a class with same method.Which interface method is overridden?](http://stackoverflow.com/questions/2801878/implemeting-2-interfaces-in-a-class-with-same-method-which-interface-method-is-o) If a type implements two interfaces, and each interface define a method that  has identical signature, then in effect there is only one method, and they are  not distinguishable. If, say, the two methods have conflicting return types,  then it will be a compilation error. This is the general rule of inheritance,  method overriding, hiding, and declarations,  and applies also to possible conflicts not only between 2 inherited interface   methods, but also an interface and a  super class method, or even just conflicts due to type erasure of generics.  **interface** ITest {  **void** Test();  }  **interface** ITest2 {  **void** Test();  }  **class** Student **implements** ITest,ITest2  {  **void** Test() {  System.out.println("ITest.Test");  }    **void** main(String[] args) {    Student cls = **new** Student();  cls.Test();  }    }  }  }  **If return type is different, it will not compile**  }  But in dot not its different  Public class Implementing Class : AClass1, IClass1, IClass2  {  publicoverridestringMethod()  {  return"AClass1";  }  stringIClass1.Method()  {  return"IClass1";  }  stringIClass2.Method()  {  return"IClass2";  }  }  So when calling from different class you will have to type cast the object into required Interface or Abstract class.  Implementing Class implementingClass = newImplementingClass();  ((AClass1)implementingClass).Method();  Jump to page : **1** | [2](http://www.indiabix.com/technical/core-java/2) | [3](http://www.indiabix.com/technical/core-java/3) | [4](http://www.indiabix.com/technical/core-java/4) | [5](http://www.indiabix.com/technical/core-java/5) | [6](http://www.indiabix.com/technical/core-java/6) | [7](http://www.indiabix.com/technical/core-java/7) | [8](http://www.indiabix.com/technical/core-java/8) | [9](http://www.indiabix.com/technical/core-java/9) | [10](http://www.indiabix.com/technical/core-java/10) | [11](http://www.indiabix.com/technical/core-java/11) | [12](http://www.indiabix.com/technical/core-java/12) | [13](http://www.indiabix.com/technical/core-java/13) | [14](http://www.indiabix.com/technical/core-java/14) | [15](http://www.indiabix.com/technical/core-java/15) | [16](http://www.indiabix.com/technical/core-java/16) | [17](http://www.indiabix.com/technical/core-java/17) | [18](http://www.indiabix.com/technical/core-java/18) | [19](http://www.indiabix.com/technical/core-java/19) | [20](http://www.indiabix.com/technical/core-java/20) | [21](http://www.indiabix.com/technical/core-java/21) | [22](http://www.indiabix.com/technical/core-java/22) | |  |

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| Using *Enumeration*, you can only traverse the collection. You can’t do any modifications to collection while traversing it. | Using *Iterator*, you can remove an element of the collection while traversing it. |
| *Enumeration* is introduced in JDK 1.0 | *Iterator* is introduced from JDK 1.2 |
| *Enumeration* is used to traverse the legacy classes like *Vector*, *Stack* and *HashTable*. | *Iterator* is used to iterate most of the classes in the collection framework like *ArrayList*, *HashSet*, *HashMap*, *LinkedList* etc. |
| Methods : *hasMoreElements()*and *nextElement()* | Methods : *hasNext()*, *next()* and *remove()* |
| *Enumeration* is fail-safe in nature. | *Iterator* is fail-fast in nature. |
| *Enumeration* is not safe and secured due to it’s fail-safe nature. | *Iterator*is safer and secured than *Enumeration*. |

**Possible Duplicate:**[**What's the meaning of System.out.println in Java?**](http://stackoverflow.com/questions/3406703/whats-the-meaning-of-system-out-println-in-java)

I was looking for the answer of what System, out and println are in System.out.println() in the Java. I searched and found a different answer like these:

* System is a built-in class present in java.lang package. This class has a final modifier, which means that, it cannot be inherited by other classes. It contains pre-defined methods and fi elds, which provides facilities like standard input, output, etc.
* out is a static final field (ie, variable)in System class which is of the type PrintStream (a built-in class, contains methods to print the different data values). static fields and methods must be accessed by using the class name, so ( System.out ).
* out here denotes the reference variable of the type PrintStream class.
* println() is a public method in PrintStream class to print the data values. Hence to access a method in PrintStream class, we use out.println() (as non static methods and fields can only be accessed by using the refrence varialble)
* System is static class and cannot be instantiated
* out is a reference variable defined in System
* println() is the method used to print on standard output.

**Passing String array as argument to main() method**

# Or [What is the need of String array in the main method of JAVA](http://stackoverflow.com/questions/4506634/what-is-the-need-of-string-array-in-the-main-method-of-java)

Consider you have a method some thing like this   
  
myMethod(int){ }   
  
You can invoke this method only by passing an int like   
  
myMethod(5) (or)   
myMethod(a) // "a" holding some int value   
  
You can never invoke this method like this   
  
myMethod("Hello");   
  
Similary JVM always invokes the main method which takes String array as an argument. If it cant find one then a exception would be thrown.

* ‘java takes command line arguments from user so as to execute the program,that's why we always mention arguments for main() , now java always assumes that the argumetns are of String type which may be appropriately converted to any other type later. that is why we pass string array to hold the command line argumetns supplied by the user.

’

**Static**

Static means that the variable or method marked as such is available at the class level. In other words, you don't need to create an instance of the class to access it.

a static member is at the class level. Therefore, the said member is loaded into memory by the JVM once for that class (when the class is loaded). That is, there aren't *n* instances of a static member loaded for *n* instances of the class to which it belongs.

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class.Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object. A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a nonstatic method. In other words, you can't change a static method into an instance method in a subclass.

Does this definition make sense ?

privatestaticfinalString string = "Constant string";

It makes sense, yes. This is how constants are defined in Java.

* final means that the variable cannot be reassigned - i.e. this is the only value it can have
* static means that the same value is accessible from each instance of the class (it also means it can be accessed even without an instance of the class where it is declared)

(private here means this constant is accessible only to the current class (all of its instances and its static methods))

|  |  |
| --- | --- |
|  |  |
| **A:** | What Is the Static Keyword in Java? In Java, a *static* member is a member of a class that isn’t associated with an instance of a class. Instead, the member belongs to the class itself. As a result, you can access the static member without first creating a class instance.  The two types of static members are static fields and static methods:   * **Static field:**A field that’s declared with the static keyword, like this:   private static int ballCount;  The position of the static keyword is interchangeable with the positions of the *visibility keywords* (private and public, as well as protected). As a result, the following statement works, too:  static private int ballCount;  As a convention, most programmers tend to put the visibility keyword first.  The value of a static field is the same across all instances of the class. In other words, if a class has a static field named CompanyName, all objects created from the class will have the same value for CompanyName.  Static fields are created and initialized when the class is first loaded. That happens when a static member of the class is referred to or when an instance of the class is created, whichever comes first.   * **Static method: A method declared with the static keyword. Like static fields, static methods are associated with the class itself, not with any particular object created from the class. As a result, you don’t have to create an object from a class before you can use static methods defined by the class.**   The best-known static method is main, which is called by the Java runtime to start an application. The main method must be static, which means that applications run in a static context by default.  One of the basic rules of working with static methods is that you can’t access a nonstatic method or field from a static method because the static method doesn’t have an instance of the class to use to reference instance methods or fields. |
|  | Static variable are common to all objects !!??  A static variable is a class variable and not for any particular object.  Static variables are created/loaded during the class loading time.  Static variable are accessed using the class name which denotes that the variables are not dependent of any object but the class itself.  [**TOP**](http://www.allapplabs.com/interview_questions/java_interview_questions.htm#top) |

**Java Static Classes**

• For java classes, only an inner class can be declared using the static modifier.

• For java a static inner class it does not mean that, all their members are static. These are called nested static classes in java.

Can constructor be Private?

Yes Ofcourse

# Writing Final Classes and Methods

You can declare some or all of a class's methods *final*. You use the final keyword in a method declaration to indicate that the method cannot be overridden by subclasses. The Object class does this—a number of its methods are final.

class **PersonalLoan**{

 public final String getName(){

     return "personal loan";

 }

}

class **CheapPersonalLoan** extends **PersonalLoan**{

    @Override

    public final String getName(){

        return "cheap personal loan"; **//compilation error: overridden method is final**

    }

}

**Note that you can also declare an entire class final. A class that is declared final cannot be subclassed. This is particularly useful, for example, when creating an immutable class like the String class.**

final class **PersonalLoan**{

}

class **CheapPersonalLoan** extends **PersonalLoan**{  //compilation error: cannot inherit from final class

}

**Why top level class can not be static**

no sense to do this anyway.because the JVM directly accesses ur top class without instantiating it. Though for accessing the instance data members, the class must need to be instantiated.

What the static boils down to is that an instance of the class can stand on its own. Or, the other way around: a non-static inner class (= instance inner class) cannot exist without an instance of the outer class. Since a top-level class does not have an outer class, it can't be anything but static.

Because *all* top-level classes are static, having the static keyword in a top-level class definition is pointless.

17. Why is the main() method declared static?

Create an instance of the java class where it is declared. Any static java method can be called without having to instantiate the class that defines that static method. The reason we wouldn't want toforce the JVM to create an instance of the java class to call the main() method is because creating this instance may be expensive and/or have unwanted side effects and/or require constructorarguments that the JVM doesn't know how to provide

33. Can a class be declared as static?

We can not declare top level class as static, but only inner class can be declared static.

**What are the differences between static and non-static nested classes?**  
Following are major differences between static nested class and non-static nested class. Non-static nested class is also called Inner Class.

**1)** Nested static class doesn’t need reference of Outer class, but Non-static nested class or Inner class requires Outer class reference.

**2)** Inner class(or non-static nested class) can access both static and non-static members of Outer class. A static class cannot access non-static members of the Outer class. It can access only static members of Outer class.

**3)**An instance of Inner class cannot be created without an instance of outer class and an Inner class can reference data and methods defined in Outer class in which it nests, so we don’t need to pass reference of an object to the constructor of the Inner class. For this reason Inner classes can make program simple and concise.

<http://www.geeksforgeeks.org/static-class-in-java/>

|  |
| --- |
| /\* Java program to demonstrate how to implement static and non-static     classes in a java program. \*/  classOuterClass{     privatestaticString msg = "GeeksForGeeks";       // Static nested class     Public static class NestedStaticClass{           // Only static members of Outer class is directly accessible in nested         // static class         publicvoidprintMessage() {             // Try making 'message' a non-static variable, there will be           // compiler error           System.out.println("Message from nested static class: "+ msg);         }      }        // non-static nested class - also called Inner class      publicclassInnerClass{           // Both static and non-static members of Outer class are accessible in         // this Inner class         publicvoiddisplay(){            System.out.println("Message from non-static nested class: "+ msg);         }      }  }  classMain  {      // How to create instance of static and non static nested class?      publicstaticvoidmain(String args[]){           // create instance of nested Static class         OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();           // call non static method of nested static class         printer.printMessage();           // In order to create instance of Inner class we need an Outer class         // instance. Let us create Outer class instance for creating         // non-static nested class         OuterClass outer = newOuterClass();         OuterClass.InnerClass inner  = outer.newInnerClass();           // calling non-static method of Inner class         inner.display();           // we can also combine above steps in one step to create instance of         // Inner class         OuterClass.InnerClass innerObject = newOuterClass().newInnerClass();           // similarly we can now call Inner class method         innerObject.display();      }  } |

Output:

Message from nested static class: GeeksForGeeks

Message from non-static nested class: GeeksForGeeks

Message from non-static nested class: GeeksForGeeks

publicclassTest

{

staticclassInnerClass

{

publicstaticvoid InnerMethod()

{ System.out.println("Static Inner Class!"); }

}

publicstaticvoid main(String args[])

{

Test.InnerClass.InnerMethod();

}

}

//output: Static Inner Class!

**member that static methods can't be overridden! This doesn't mean they   
can't be redefined in a subclass, but redefining and overriding aren't the same thing.   
Let's take a look at an example of a redefined (remember, not overridden), static   
method:**

1. **class** Animal
2. {
3. **static** **void** doStuff()
4. {
5. System.out.print("a ");
6. }
7. }
8. **class** Dog **extends** Animal
9. {
10. **static** **void** doStuff()
11. {
12. // it's a redefinition,
13. // not an override
14. System.out.print("d ");
15. }
17. **public** **static** **void** main(String [] args)
18. {
19. Animal [] a = {**new** Animal(), **new** Dog(), **new** Animal()};
20. **for**(**int** x = 0; x < a.length; x++)
21. a[x].doStuff();
22. // invoke the static method
23. }
24. }
26. /\*
27. Running this code produces the output:
28. a a a
29. \*/

overriding happens at runtime VS Redefining happens at compile time. 

Overriding means the Object's type dictates which method to use, and not the reference variable type, and this happens in runtime VS Redefining means that the reference variable type determines this, and this is done at compile time.   
  
If you replaced static to normal in your example, you should get "a d a" as output.

Static Methods stick to the Class scope and not to the instance (or) object scope... That is static method cannot execute based on instance property variables.... Redefined method can be called only by specifying the correct Class which contains the redefined method.   
  
This is not the case for Overriding... No need of specifying the Class explicitly to call the overriding method...

Overriding happens at runtime VS Redefining happens at compile time.

The only difference with inherited static (class) methods and inherited non-static (instance) methods is that when you write a new static method with the same signature, the old static method is just hidden, not overridden.

<http://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/>

### Advantage of Java Method Overriding

* Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method Overriding is used for Runtime Polymorphism

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| 1) Method overloading is used to increase the readability of the program. | Method overriding is used to provide the specific implementation of the method that is already provided by its super class. |
| 2) method overloading is performed within a class. | Method overriding occurs in two classes that have IS-A relationship. |
| 3) In case of method overloading parameter must be different. | In case of method overriding parameter must be same. |

### Why we cannot override static method?

Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Just to add on, static methods are bound to class rather than objects, so runtime polymorphism is not possible at all.

Therefore you cannot override them.

Static methods in the same class and base classes

(and outer classes) are available, with noati

Qualification

Example- **package** package3;**ass** Base1 {**publicstaticvoid** display2() {

System.*out*.println("base Static or class method from Base");

}}

**class** Base **extends** Base1 {

**publicstaticvoid** display() {

System.*out*.println("base Static or class method from Base");

}

**publicvoid** print() {

System.*out*.println ("base Non-static or Instance method from Base");

}

**publicstaticvoid** display1() {

System.*out*.println("base Static or class method from Base");

}

}**publicclass** Test **extends** Base

**publicstaticvoid** display() {

System.*out*.println(" derived Static or class method from Derived");

}

**publicvoid** print() {

System.*out*.println("Derived Non-static or Instance method from ");

}

**publicstaticvoid** main(String args[ ]) {

Base obj1 = **new**Test();

Test tst= **new**Test();

Test.*display*();

Base.*display*();

*display2*();

*display1*();

}

}

34. When will you define a method as static?

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

35. What are the restriction imposed on a static method or a static block of code?

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

1. If you are writing utility classes and they are not supposed to be changed.
2. If the method is not using any instance variable.
3. If any operation is not dependent on instance creation.
4. If there is some code that can easily be shared by all the instance methods, extract that code into a static method.
5. If you are sure that the definition of the method will never be changed or overridden. As static methods can not be overridden.

Static methods cannot be overridden, they can however be hidden using the 'new' keyword. Mostly overriding methods means you reference a base type and want to call a derived method.

|  |
| --- |
|  |

**37. What is the importance of static variable?**

Static variables are class level variables where all objects of the class refer to the same variable. If one object changes the value then the change gets reflected in all the objects.

**59. What is a local, member and a class variable?**

Variables declared within a method are "local" variables.

Variables declared within the class i.e not within any methods are "member" variables (global variables).

Variables declared within the class i.e not within any methods and are defined as "static" are class variables.

publicstaticvoid main(string argc[])

public - This method can be accessed from anywhere.

**75. What modifiers may be used with an inner class that is a member of an outer class?**

A (non-local) inner class may be declared as public, protected, private, static, final, or abstract.

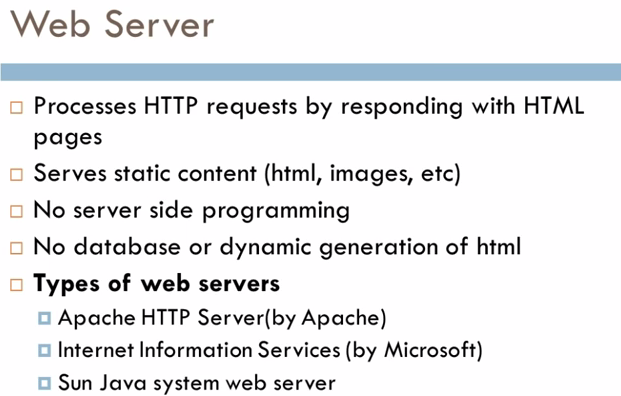
# [Why are interface variables static and final by default?](http://stackoverflow.com/questions/2430756/why-are-interface-variables-static-and-final-by-default)

**Since interface doesn't have a direct object, the only way to access them is by using a class/interface and hence that is why if interface variable exists,** it should be static otherwise it won’t be accessible at all to outside world. Now since it is static, it can hold only one value and any classes that extends it can change it and hence it will be all mess.

**Static - because Interface cannot have any instance. and final - because we do not need to change it.**

Q- System.out.println

Q- Webserver vs Application server



* Web Server is designed to **serve HTTP Content. App Server can also serve HTTP Content but is not limited to just HTTP**. It can be provided other protocol support such as RMI/RPC

Web Server is mostly designed to serve static content, though most Web Servers have plugins to support scripting languages like Perl, PHP, ASP, JSP etc. through which these servers can generate dynamic HTTP content.Both terms are very generic, one containing the other one and vice versa in some cases.

* **Web server**: serves content to the web using http protocol.
* **Application server**: hosts and exposes business logic and processes.
* Most of the application servers have Web Server as integral part of them, that means App Server can do whatever **Web Server is capable of. Additionally App Server have components and features to support Application level services such as Connection Pooling, Object Pooling, Transaction Support, Messaging services etc.**

**Connection pooling** means that **connections** are reused rather than created each time a **connection** is requested. To facilitate **connection** reuse, a memory cache of database **connections**, called a **connection pool**, is maintained by a **connection pooling** module as a layer on top of any standard JDBC driver product.

A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.

## As web servers are well suited for static content and app servers for dynamic content, most of the production environments have web server acting as reverse proxy to app server. That means while servicing a page request, static contents (such as images/Static HTML) are served by web server that interprets the request. Using some kind of filtering technique (mostly extension of requested resource) web server identifies dynamic content request and transparently forwards to app server Web Server -> Programming Environment

IIS : ASP (.NET)

Tomcat : Servlet

Jetty : Servlet

Apache : Php, CGI

## Application Servers -> Programming Environment

MTS : COM+

WAS : EJB

JBoss : EJB

WebLogic Application Server : EJB

The crucial difference is that application servers support some **distributed component** technology, providing features like remote invocation and distributed transactions, like **EJB** in Java world or **COM+**on Microsoft platform. Http server often support some more simple programming environments, often scripting, like ASP (.NET) in case of Microsoft or Servlet--based, including JSP and many other in case of Java or PHP and CGI in case of Apache.

Other capabilities like load-balancing, clustering, session-failover, connection pooling etc. that used to be in the realm of application servers, are becoming available on web servers as well directly or through some third party products.

Finally, it is worth noting that the picture is further distorted with "lightweight containers" like Spring Framework, that often supplement the purpose of application servers in more simple manner and without the application server infrastructure. And since distribution aspect in applications is moving from distributed component towards service paradigm and SOA architecture, there is less and less space left for traditional application servers.

1**. Application Server**supports **distributed transaction and EJB**. While Web Server only supports Servlets and JSP.

2. Application Server can contain web server in them. most of App server e.g. JBoss or WAS has Servlet and JSP container.

3. Though its not limited to Application Server but they used to provide services like **Connection pooling**, **Transaction management**, messaging, clustering, load balancing and persistence. Now Apache tomcat also provides connection pooling.

4. In terms of l*ogical difference between web server and application server*. web server is supposed to provide http protocol level service while application server provides support to web service and expose business level service e.g. EJB.

5. Application server are more heavy than web server in terms of resource utilization.

Q-Classptah vs Path’

path- where javac.exe is available which compiles the program

Classpth- find .class file

**PATH is used by the OS to find executables to run.**The PATH variable is a **Windows** system variable that Windows uses to find executable files **CLASSPATH is used by the JVM to find class files to run.**

 Another significant difference between PATH and CLASSPATH is that PATH can not be overridden by any Java settings but CLASSPATH can be overridden by providing command line option -classpath or -cp to both "java" and "javac" commands or by using Class-Path attribute in Manifest file inside JAR archive.

The CLASSPATH variable is a **Java** runtime system variable that Java uses to find class files. When Java tries to load a class, it will lookup its ClassLoader. If the class is not yet loaded, Java tries to find the class in the CLASSPATH variable. Sometimes the CLASSPATH may be appended from the runtime command line directly.

#### 4.1  PATH

When you launch a program from the command line, the operating system uses the PATH environment variable to search for the program in your local file system. In other words, PATH maintains a list of directories for searching executable programs.

Q- Jar war ear

Jar- only .class file

War – web application jsp servelet , html, js, css etc

Ear – Enterprise application EJB, JMS, servelet , jsp

Q- JDK= Jre + JVM

# Difference between Exception and Error

# : An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

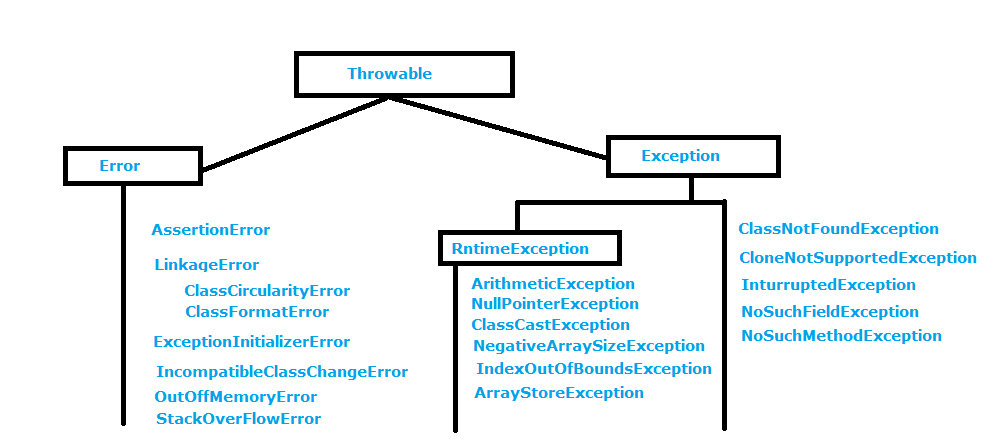
Exception are recoverable but Error not

Exception are caused by our program but error not

**Errors** are also unchecked exception & the programmer is not required to do anything with these. In fact it is a bad idea to use a

Try-catch clause for Errors. Most often, recovery from an Error is not possible & the program should be allowed to terminate.

**Examples include OutOfMemoryError, StackOverflowError, etc.**



## Error Vs Exception In Java :

1) Recovering from **Error** is not possible. The only solution to errors is to terminate the execution. Where as you can recover from **Exception** by using either try-catch blocks or throwing exception back to caller.

2) You will not be able to handle the **Errors** using try-catch blocks. Even if you handle them using try-catch blocks, your application will not recover if they happen. On the other hand, **Exceptions** can be handled using try-catch blocks and can make program flow normal if they happen.

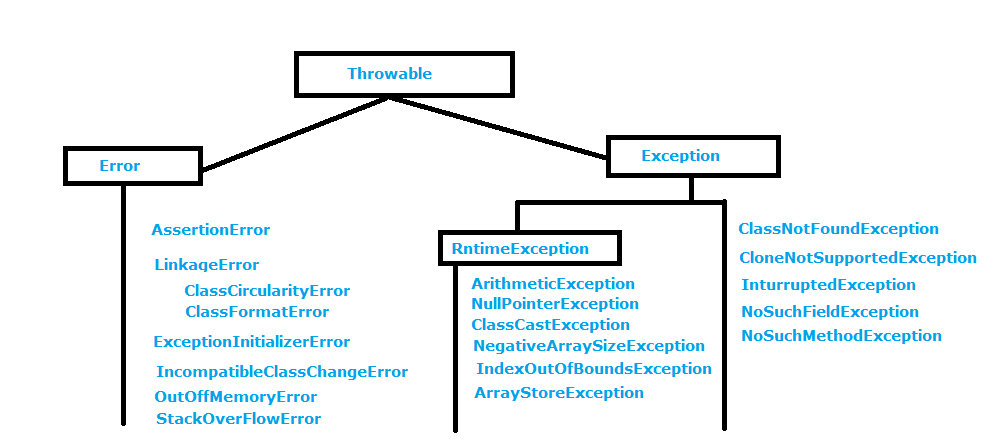
**An Error** "indicates serious problems that a reasonable application should not try to catch."

while

**An Exception** "indicates conditions that a reasonable application might want to catch."

Both **java.lang.Error** and **java.lang.Exception** classes are sub classes of **java.lang.Throwable**class**,** but there exist some significant differences between them. **java.lang.Error** class represents the errors which are mainly caused by the environment in which application is running. For example, **OutOfMemoryError** occurs when JVM runs out of memory or **StackOverflowError** occurs when stack overflows.

Where as **java.lang.Exception** class represents the exceptions which are mainly caused by the application itself. For example, **NullPointerException** occurs when an application tries to access null object or **ClassCastException** occurs when an application tries to cast incompatible class types. In this article, we will discuss the differences between Error and Exception in java.



1. **== vs equals**

== point to memory location ;

Equals – content of string and also is case sensitive;

1. **class** Teststringcomparison1{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=**new** String("Sachin");
6. String s4="Saurav";
7. System.out.println(s1.equals(s2));//true
8. System.out.println(s1.equals(s3));//true
9. System.out.println(s1.equals(s4));//false
10. }
11. }

### 2. Why string objects are immutable in java?

|  |
| --- |
| Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java. |

1. **public** **static** **void** main(String args[]){
2. String s="Sachin";
3. s.concat(" Tendulkar");//concat() method appends the string at the end
4. System.out.println(s);//will print Sachin because strings are immutable objects
5. }
6. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring)

Output:Sachin

1. **public** **static** **void** main(String args[]){
2. String s="Sachin";
3. S=s.concat(" Tendulkar");//concat() method appends the string at the end
4. System.out.println(s);//will print Sachin because strings are immutable objects
5. }
6. }

## Example of java substring

public class TestSubstring{

1. public static void main(String args[]){
2. String s="Sachin Tendulkar";
3. System.out.println(s.substring(6));//Tendulkar
4. System.out.println(s.substring(1,4));//ach
5. }
6. }

Out put is--- Tendulkar

ach

(0,4)0 is S 4th is H but end one is Excluded

# Java StringBuffer class

Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

# Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

Exceptions

2 types

1.Checked- All exception Except runtime is Checked Exception- it gives error in compile time

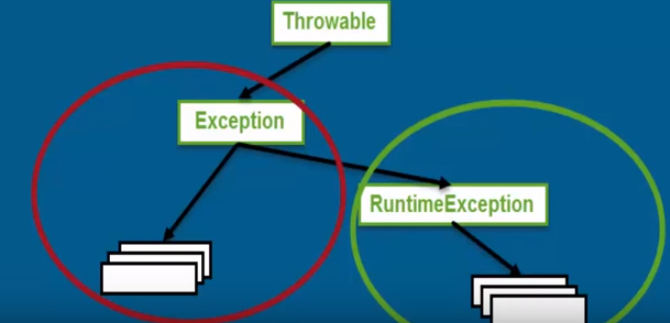
Like IoException,

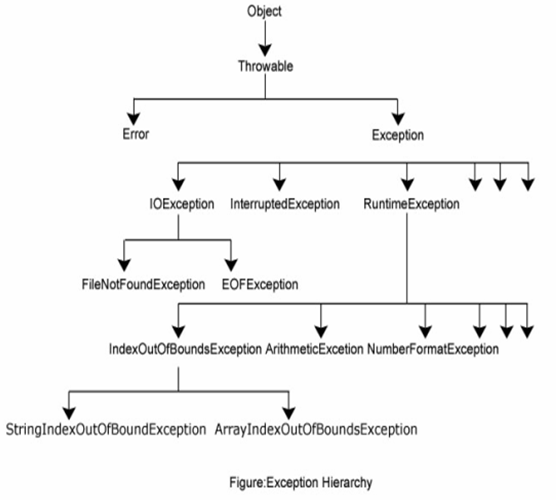
2.Unchecked—Runtime Exception which we we can not handled is Unchled exception

The exceptions that are not checked at compile time are called unchecked exceptions, classes that extends RuntimeException comes under unchecked exceptions. Examples of some unchecked exceptions are listed below.

Like we d

­­­­­­





## Error vs Exception in Java

As I said earlier, Main difference on Error vs Exception is that Error is not meant to catch as even if you catch it you can not recover from it. For example during [OutOfMemoryError](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html), if you catch it you will get it again because GC may not be able to free memory in first place. On the other hand Exception can be caught and handled properly.

2) Error are often fatal in nature and recovery from Error is not possible which is different in case of Exception which may not be fatal in all cases.

Both Error and Exception are derived from java.lang.Throwable in Java but main difference between Error and Exception is kind of error they represent. java.lang.Error represent errors which are generally can not be handled and usually refer catastrophic failure e.g. running out of System resources,

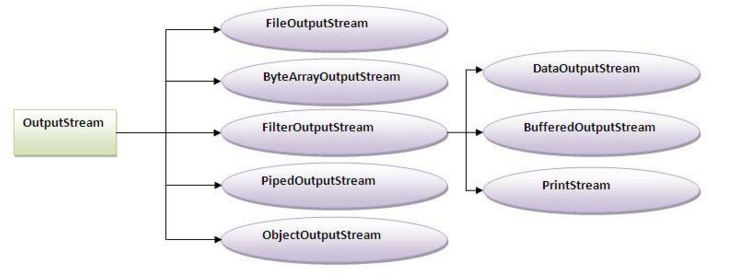
some examples of Error in Java are [java.lang.OutOfMemoryError](http://javarevisited.blogspot.sg/2011/09/javalangoutofmemoryerror-permgen-space.html) or [Java.lang.NoClassDefFoundError](http://java67.blogspot.sg/2012/08/what-is-noclassdeffounderror-in-java.html) and[java.lang.UnSupportedClassVersionError](http://java67.blogspot.sg/2012/10/how-to-fix-javalangunsupportedclassversionerror-major-minor-version-49-50-51.html). On the other hand java.lang.Exception represent errors which can be catch and dealt e.g. IOException which comes while performing I/O

Error-NoClassDefFoundError and java.lang.UnSupportedClassVersionError,OutOFmerroryError,StackOverFlowError

# Java I/O Tutorial

### OutputStream

Java application uses an output stream to write data to a destination, it may be a file,an array,peripheral device or socket.



### InputStream

Java application uses an input stream to read data from a source, it may be a file,an array,peripheral device or socket.



## Java FileOutputStream class

Java FileOutputStream is an output stream for writing data to a file.

If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

**try**{

     FileOutputstream fout=**new** FileOutputStream("abc.txt");

     String s="Sachin Tendulkar is my favourite player";

**byte** b[]=s.getBytes();//converting string into byte array

     fout.write(b);

     fout.close();

     System.out.println("success...");

    }**catch**(Exception e){system.out.println(e);}

## Java FileInputStream class

Java FileInputStream class obtains input bytes from a file.It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

It should be used to read byte-oriented data for example to read image, audio, video etc.

### Example of FileInputStream class

import java.io.\*;

class SimpleRead{

 public static void main(String args[]){

  try{

    FileInputStream fin=new FileInputStream("abc.txt");

    int i=0;

    while((i=fin.read())!=-1){

     System.out.println((char)i);

    }

    fin.close();

  }catch(Exception e){system.out.println(e);}

 }

Example of Reading the data of current java file and writing it into another file

We can read the data of any file using the FileInputStream class whether it is java file, image file, video file etc. In this example, we are reading the data of C.java file and writing it into another file M.java.

**import** java.io.\*;

FileInputStream fin=**new** FileInputStream(**new** File("D:/AAA.txt"));

**int** i;

StringBuffer strContent = **new** StringBuffer("");

**while**( (i = fin.read()) != -1)

{

strContent.append((**char**)i);

}

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

}

# Java ByteArrayOutputStream class

Java ByteArrayOutputStream class is used to write data into multiple files. In this stream, the data is written into a byte array that can be written to multiple stream.

# Java BufferedOutputStream and BufferedInputStream

## Java BufferedOutputStream class

Java BufferedOutputStream class uses an internal buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

1. **import** java.io.\*;
2. **class** Test{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileOutputStream fout=**new** FileOutputStream("f1.txt");
5. BufferedOutputStream bout=**new** BufferedOutputStream(fout);
6. String s="Sachin is my favourite player";
7. **byte** b[]=s.getBytes();
8. bout.write(b);
10. bout.flush();
11. bout.close();
12. fout.close();
13. System.out.println("success");
14. }
15. }

## Java BufferedInputStream class

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

FileInputStream fin = **new** FileInputStream("D:/AAA.txt");

**int** i;

FileInputStream fin1 = **new** FileInputStream("D:/AAA.txt");

FileReader fr = **new** FileReader("D:/AAA.txt");

BufferedReader bin = **new** BufferedReader(fr);

**int** i1;

String l;

1st way

while ((l = bin.readLine()) != null) {

System.out.println(l);

}

2nd way

while((i=fr.read())!=-1)

System.out.print((char)i);

fr.close();

}

3rd way

BufferedInputStream bin1=**new** BufferedInputStream(fin);

**int** j;

**while**((j=bin.read())!=-1){

System.***out***.print((**char**)j);

}

# }

# Java FileWriter and FileReader (File Handling in java)

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java.

**Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.**

1. import java.io.\*;
2. class Simple{
3. public static void main(String args[]){
4. try{
5. FileWriter fw=new FileWriter("abc.txt");
6. fw.write("my name is sachin");
7. fw.close();
8. }catch(Exception e){System.out.println(e);}
9. System.out.println("success");
10. }
11. }

Output:

success...

## Java FileReader class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

### Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |

1. import java.io.\*;
2. class Simple{
3. public static void main(String args[])throws Exception{
4. FileReader fr=new FileReader("abc.txt");
5. int i=fr.read();
6. while((i=fr.read())!=-1)
7. System.out.println((char)i);
8. i=fr.read()
10. }
11. fr.close();
12. }

Output:

my name is sachin

/\* method 1

\* FileReader fr = new FileReader("E://Friends//Teest.txt");

int s=fr.read();

while(s!=-1)

{

System.out.print((char)s);

s=fr.read();

}

fr.close();

\*/

// method 2

File fl = **new**File("E:\\Friends\\Teest.txt");

FileReader fr = **new**FileReader(fl);

**char**[] s = **newchar**[(**int**) fl.length()];

fr.read(s);

**for** (**char** s1:s)

{

System.*out*.print(s1);

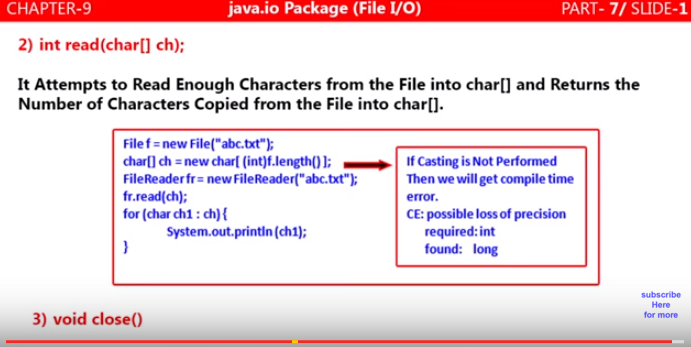
}

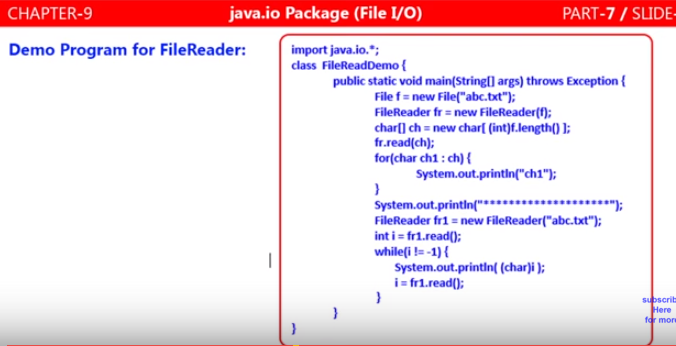
# Reading data from keyboard

There are many ways to read data from the keyboard. For example:

* InputStreamReader
* Console
* Scanner
* DataInputStream etc.

How to use File Reader





**FileWriter can be used 2 ways**

**1. Append – Depends on Construtor**

**2.Override**

**Publicstaticvoid**main(String[] args) **throws** IOException

{

String da="test is test";

File f = **new**File("E://Friends","Teest.txt");

**if**(!f.exists()){

f.createNewFile(); // **Create file**

}

FileWriter fw = **new**FileWriter(f,**true**);// **Append data in file**

FileWriter fw = **new**FileWriter(f);/// **Overrides in file**

BufferedWriter bw = **new**BufferedWriter(fw);

// Scanner scn = new Scanner(System.in);

//String ds=scn.nextLine();

//bw.write('n');

bw.write(da+'\n');

bw.close();

//System.out.println("done");

}

}

**Why FileReader and FileWriter Not Recomnded**



**Bufferedwriter**

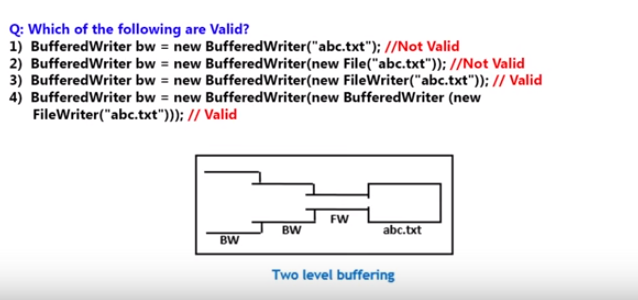
Bufferedwriter communicate with some filewriter not directly with file it can be writer, filewriter or can b bufferwriter as well

FileWriter file = newFileWriter("foo.txt");

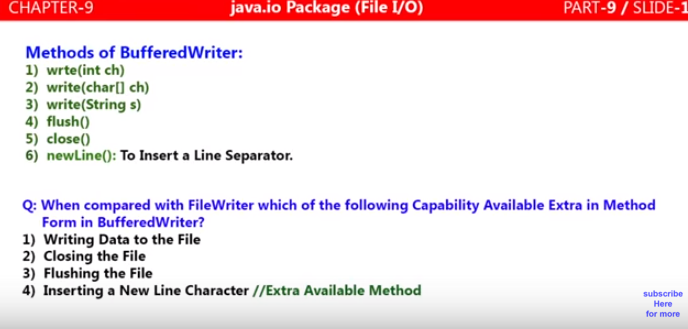
BufferedWriter bf = newBufferedWriter(file);

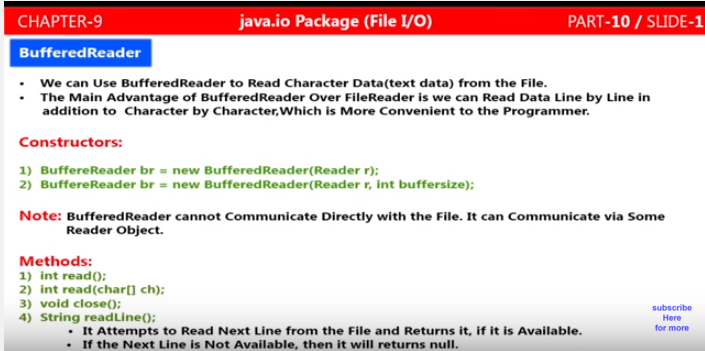
BufferedWriter bf = newBufferedWriter(Writer writer);

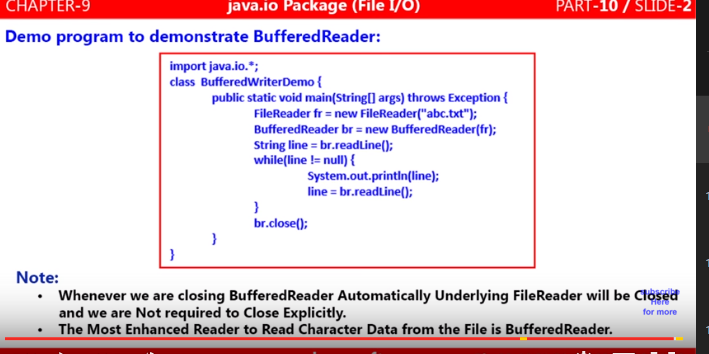
BufferedWriter bf = newBufferedWriter(Writer writer,int buffersize);



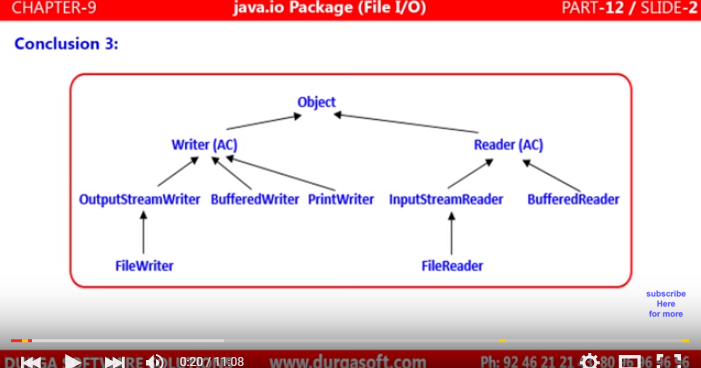
When we close bufferedwriter, filewriter is closed Automatically because its working inside Bufferreader

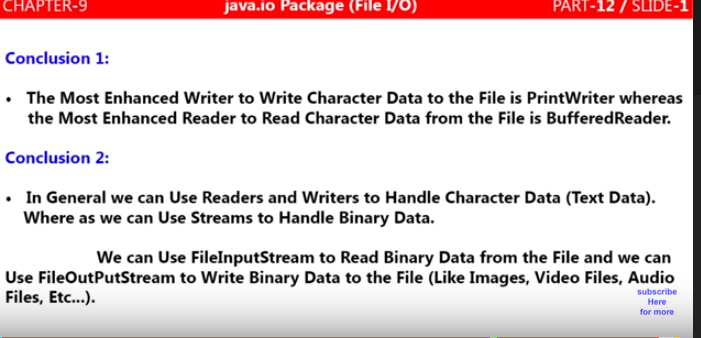






**Hirachical Classfication of files Reader/Writer**





**replace text in file**

File log= new File("log.txt");

String search = "textFiles/a.txt";

String replace = "replaceText/b.txt";

try{

FileReader fr = new FileReader(log);

String s;

String totalStr = "";

try (BufferedReader br = new BufferedReader(fr)) {

while ((s = br.readLine()) != null) {

totalStr += s;

}

totalStr = totalStr.replaceAll(search, replace);

FileWriter fw = new FileWriter(log);

fw.write(totalStr);

fw.close();

}

}catch(Exception e){

e.printStackTrace();

}

**package AADEF;**

**import java.io.BufferedReader;**

**import java.io.BufferedWriter;**

**import java.io.File;**

**import java.io.FileNotFoundException;**

**import java.io.FileReader;**

**import java.io.FileWriter;**

**import java.io.IOException;**

**import java.lang.reflect.Field;**

**import java.lang.reflect.InvocationTargetException;**

**import java.lang.reflect.Method;**

**import java.sql.SQLDataException;**

**import AProgramsPorblems.A\_MinMaxAlternativeWithNewArray;**

**class \_\_\_DD**

**{**

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

**// FileReader fr = new FileReader("D:/AAA.txt");**

**File f = new File("D:/AAA.txt");**

**try**

**{**

**String ENDL = System.getProperty("line.separator");**

**StringBuilder sb = new StringBuilder();**

**BufferedReader br = new BufferedReader(new FileReader(f));**

**String ln;**

**while((ln = br.readLine()) != null)**

**{**

**sb.append(ln.replace("RAM", "RAM1").replace("SAHU", "SAHU1")**

**).append(ENDL);**

**}**

**br.close();**

**BufferedWriter bw = new BufferedWriter(new FileWriter(f));**

**bw.write(sb.toString());**

**bw.close();**

**}**

**catch (IOException e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**Collections**

**Vector vs Array List**

But there are many differences between ArrayList and Vector classes that are given below.

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1) ArrayList is **not synchronized**.  **ArrayList:**   * Initial Capacity:10 * Load Factor:1 (when the list is full) * Growth Rate: current\_size + current\_size/2   **Vector:**   * Initial Capacity:10 * Load Factor:1 (when the list is full) * Growth Rate: current\_size \* 2 (if capacityIncrement is not defined) current\_size + capacityIncrement (if capacityIncrement is defined during vector initialization) | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity. | Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object. |
| 5) ArrayList uses **Iterator** interface to traverse the elements. | Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also. |

**Difference between HashTable and Hashmap**

1**. Hashtable not allowed any null keys or any null value** where as hashmap can have one null key and more null values..

**2. Hashtable is synchronized whereas** HashMap is non synchronized, synchronized means Hashtable is thread-safe and can be shared between multiple threads but HashMap can not be shared between multiple threads without proper synchronization

3. **one more notable difference between Hashtable and HashMap is that because of thread-safety and synchronization Hashtable is much slower than HashMap** if used in Single threaded environment. So if you don't need synchronization and HashMap is only used by one thread, it out perform Hashtable in Java.

HashMap<Interger,String> employeeHashmap=new HashMap<Integer,String>();

employeeHashmap.put(1,"Arpit");

employeeHashmap.put(2,null); // will work fine

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hashtable<Interger,String> employeeHashmap=new Hashtable<Integer,String>();

employeeHashmap.put(1,"Arpit");

employeeHashmap.put(2,null); //not allowed and will throw NullPointer exception at run time

Hashtable vs HashMap:

The following are the legacy classes defined by **java.util** package

1. Dictionary
2. HashTable
3. Properties
4. Stack
5. Vector

There is only one legacy interface called **Enumeration**

**Array vs ArrayList in Java**

Difference between Array and ArrayList in Java with example

1) First **and Major Difference between Array and ArrayList in Java is that Array is a fixed length data structure** while ArrayList is a variable length Collection class. You can’not change length of Array once created in Java but ArrayList re-size itself when gets full depending upon capacity and load factor. **Since ArrayList is internally backed by Array in Java, any resize operation in ArrayList will slow down performance as it involves creating new Array and copying content from old array to new array.**

2) Another difference between Array and ArrayList in Java is that you can not use Generics along with Array, as Array instance knows about what kind of type it can hold and throws ArrayStoreException, if you try to store type which is not convertible into type of Array. ArrayList allows you to use Generics to ensure type-safety.

3) You can also compare **Array vs ArrayList on How to calculate length of Array or size of ArrayList**. **All kinds of Array provides length variable which** denotes length of Array while **ArrayList provides size(**) method to calculate size of ArrayList in Java.

4) One more major difference between **ArrayList and Array is that, you can not store primitives in ArrayList, it can only contain Objects**. While Array can contain both primitives and Objects in Java. Though Autoboxing of Java 5 may give you an impression of storing primitives in ArrayList, it actually automatically converts primitives to Object. e.g.

Stores similar data of one type ,Arraylist can store heterogeneous data types - See more at: http://way2java.com/arrays/array-vs-arraylist/#sthash.mio9uKRN.dpuf

Array Cannot be synchronized, Arraylist Can be obtained a synchronized version

Array stores primitive data types and also objects and Arraylist Stores only objects –

|  |  |
| --- | --- |
| Array must declare element type | And arraylist element type is Object |
|  |  |

**What is the difference between ArrayList and LinkedList**

1. Insertions are easy and fast in LinkedList as compared to ArrayList because there is no need of resizing array and copying content to new array if array gets full which makes adding into ArrayList to amortized constant time of O(n) in worst case, while adding is O(1) operation in LinkedList in Java. ArrayList also needs to update its index if you insert something anywhere except at the end of array.

2. Removal is like insertions better in LinkedList than ArrayList. LinkedList won’t need to re-arrange elements on removal.

3. LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node

**Common in both**

Both ArrayList and LinkedList are implementation of List interface, which means you can pass either ArrayList or LinkedList if a method accepts List interface.

2) Both ArrayList and LinkedList are not synchronized, which means you can not shared them between multiple threads without external synchronization. See here to know How to make ArrayList synchronized in Java.

3) ArrayList and LinkedList are ordered collection e.g. they maintain insertion order of elements i.e. first element will be added on first position.

4) ArrayList and LinkedList also allows duplicates and null unlike any other List implementation e.g. Vector.

5) Iterator of both LinkedList and ArrayList are fail-fast which means they will throw ConcurrentModificationException if collection is modified structurally once Iterator is created. They are different than CopyOnWriteArrayList whose Iterator is fail-safe.

**How to use both**

ArrayList should be used where you have more retrieval than insertions and deletions. while the LinkedList should be used when you have more insertions and deletions as it won’t need resizing on insertions and deletions.

Use ArrayList in Java for all those situations where you need a non-synchronized index based access. ArrayList is fast and easy to use,

**Set vs List**

**List**

I*s an Ordered grouping of elements*

*List is used to collection of elements with duplicates.*

*New methods are defined inside List interface.*

**Set**

*Is an Unordered grouping of elements.*

*Set is used to collection of elements without duplicates.*

*No new methods are defined inside Set interface, so we have to use Collection interface methods only with Set subclasses.*

##############################################

**Set (Interface)**

Set is an un-ordered collection which doesn’t allows duplicate (no-duplicate) elements

We can iterate the values by calling iterator() method

Set s = new HashSet();

Iterator iter = s.iterator();

**List (Interface)**

List is an ordered collection which allows duplicate elements

We can iterate the values by calling iterator() method

List li = new ArrayList();

Iterator iter = li.iterator();

**Map (Interface)**

In Map we used to store the data in key and value pairs, we may have duplicate values but no duplicate keys

In Map we don’t have iterator() method, but we can get the keys by calling the method keySet()

Map m; // insert values

Set s = m.keySet();

// Get Map keys into the Set and then iterate this Set object normally

// m.keySet() returns Set object with Map keys

Iterator iter = s.iterator();

Ordered lists of element (unique or not)

Conform to Java's interface named List

Can be accessed by index

LinkedList

ArrayList

Lists of unique elements:

Conform to Java's interface named Set

Cannot be accessed by index

HashSet (unordered)

LinkedHashSet (ordered)

TreeSet (sorted by natural order or by provided comparator)

Both interfaces Set and List conform to Java's interface named Collection

List - ArrayList, LinkedList and Vector

Set - HashSet, TreeSet and LinkedHashSet

Map - HashMap, Hashtable and TreeMap

**Set (Interface)**

Set is an un-ordered collection which doesn’t allows duplicate (no-duplicate) elements

We can iterate the values by calling iterator() method

Stores data as object

Set s = new HashSet();

Iterator iter = s.iterator();

**List (Interface)**

List is an ordered collection which allows duplicate elements

We can iterate the values by calling iterator() method

List li = new ArrayList();

Iterator iter = li.iterator();

**Map (Interface)**

In Map we used to store the data in key and value pairs, we may have duplicate values but no duplicate keys

In Map we don’t have iterator() method, but we can get the keys by calling the method keySet()

Map m; // insert values

Set s = m.keySet();

// Get Map keys into the Set and then iterate this Set object normally

// m.keySet() returns Set object with Map keys

Iterator iter = s.iterator();

**Difference between HashSet and HashMap in Java**

Following are some differences between HashMap and Hashset:

|  |  |
| --- | --- |
| **Hash Map** | **Hash Set** |
| HashMap  is a implementation of Mapinterface | HashSet is an implementation of Set Interface |
| **HashMap Stores data in form of  key value pair** | **HashSet Store only objects** |
| **Put method is used to add element inmap** | **Add method is used to add element is Set** |
| In hash map hashcode value is calculated using key object | Here member object is used for calculating hashcode value which can be same for two objects so equal () method is used to check for equality if it returns false that means two objects are different. |
| HashMap is faster than hashset because unique key is used to access object | HashSet is slower than Hashmap |

# [HashSet vs. TreeSet vs. LinkedHashSet](http://www.programcreek.com/2013/03/hashset-vs-treeset-vs-linkedhashset/)

1. In a set, there are no duplicate elements. That is one of the major reasons to use a set. There are 3 implementations of Set: HashSet, TreeSet and LinkedHashSet. When and which to use is an important question. In brief, if we want a fast set, we should use HashSet; if we need a sorted set, then TreeSet should be used; if we want a set that can be read by following its insertion order, LinkedHashSet should be used.
2. Difference between HashSet ,LinkedHashSet and TreeSet

1. HashSet **not maintaining any order** ,LInkedHashSet maintaining the **insertion Order** ,TreeSet is maintaining the  **Sorting order**

2. **HashSet and LinkedHashSet uses equals() method in Java for comparison but TreeSet uses compareTo() method**

3. **HashSet and LinkedHashSet allows null values but TreeSet doesn't allow null values**

**2. HashSet vs. TreeSet vs. LinkedHashSet**

HashSet is Implemented using a hash table. Elements are not ordered. The add, remove, and containsmethods has constant time complexity O(1).

TreeSet is implemented using a tree structure(red-black tree in algorithm book). The elements in a set are sorted, but the add, remove, and contains methods has time complexity of O(log (n)). It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.

LinkedHashSet is between HashSet and TreeSet. It is implemented as a hash table with a linked list running through it, so it provides the order of insertion. The time complexity of basic methods is O(1).

Generics vs collections

1. Generic collection provide more benefit than collections because it is type safe and collection is not type-safe that is we can add any type of data in collection.   
   2. Generic collection types generally perform better than the corresponding nongeneric collection types
2. 3.Examples are  
     
   1.List is the generic class for ArrayList.  
   2.Dictionary is the generic class for Hashtable.
3. *Non-Generic collections* - These are the collections that can hold elements of different data types. It holds all elements as object type.  
   So it includes overhead of type conversions.  
      
   - *Generic collections* - These are the collections that can hold data of same type and we can decide what type of data that collections can hold.  
      
   Some advantages of *generic collections* - Type Safe, Secure, reduced overhead of type conversions.
4. Collections examples: ArrayList , IEnumerator, Hashtable , SortedList.  
     
   Generic Collections Examples List<T> and Dictionary<TKey, TValue>

Using generic collections is generally recommended, because you gain the immediate benefit of type safety without having to derive from a base collection type and implement type-specific members. In addition, generic collection types generally perform better than the corresponding nongeneric collection types (and better than types derived from nongeneric base collection types) when the collection elements are value types, because with generics there is no need to box the elements.

The following generic types correspond to existing collection types:

* [List](http://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.80).aspx) is the generic class corresponding to [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist(v=vs.80).aspx).
* [Dictionary](http://msdn.microsoft.com/en-us/library/xfhwa508(v=vs.80).aspx) is the generic class corresponding to [Hashtable](http://msdn.microsoft.com/en-us/library/system.collections.hashtable(v=vs.80).aspx).
* [Collection](http://msdn.microsoft.com/en-us/library/ms132397(v=vs.80).aspx) is the generic class corresponding to [CollectionBase](http://msdn.microsoft.com/en-us/library/system.collections.collectionbase(v=vs.80).aspx). **Collection** can be used as a base class, but unlike **CollectionBase** it is not abstract, making it much easier to use.
* [ReadOnlyCollection](http://msdn.microsoft.com/en-us/library/ms132474(v=vs.80).aspx) is the generic class corresponding to [ReadOnlyCollectionBase](http://msdn.microsoft.com/en-us/library/system.collections.readonlycollectionbase(v=vs.80).aspx). **ReadOnlyCollection** is not abstract, and has a constructor that makes it easy to expose an existing **List** as a read-only collection.
* The [Queue](http://msdn.microsoft.com/en-us/library/7977ey2c(v=vs.80).aspx), [Stack](http://msdn.microsoft.com/en-us/library/3278tedw(v=vs.80).aspx), and [SortedList](http://msdn.microsoft.com/en-us/library/ms132319(v=vs.80).aspx) generic classes correspond to the respective nongeneric classes with the same names.

**Non-Generic                        Similar Generic Type**  
ArrayList              List<T>  
Hashtable              Dictionary<TKey,TValue>  
SortedList             SortedList<TKey,TValue>  
Queue                  Queue<T>  
Stack                  Stack<T>  
IEnumerable            IEnumerable<T>  
ICollection            N/A (use IEnumerable<T> anything that extends it)  
N/A                    ICollection<T>   
IList                  IList<T>  
CollectionBase         Collection<T>  
ReadOnlyCollectionBase ReadOnlyCollection<T>  
DictionaryBase         N/A (just implement IDictionary<TKey,TValue>  
N/A                    SortedDictionary<TKey,TValue>  
N/A                    KeyedCollection<TKey,TItem>  
N/A                    LinkedList<T>

<http://www.vishwamohan.com/post/2007/07/08/Speed-Test-Generic-List-vs-ArrayList.aspx>

The only downside is you have to write extra code for that. However, ArrayList takes the second place and relatively good with respect to Collection, Data Table or XML. So the decision is based on how efficient you want? You have to also realize the fact that in ArrayList, you will have 2 issues .First, you have to always retrieve data based on index (does not clearly tells you which column you are reading unless you know the mapping ). Second, you pay the penalty of Boxing/Unboxing, because every element in the ArrayList is an object, you will have to convert them to their actual data type. The benefit is you have to write less code and still achieve the better performance.

# LinkedHashSet class in Java with Examples

A LinkedHashSet is an ordered version of [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/) that maintains a doubly-linked List across all elements. When the iteration order is needed to be maintained this class in used. When iterating through a [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/) the order is unpredictable, while a LinkedHashSet lets us iterate through the elements in the order in which they were inserted.when cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

# HashSet permits to have a single null value. HashMap permits single null key and any number of null values

# LinkedHashMap class in Java with Example

[HashMap in Java](https://www.geeksforgeeks.org/hashmap-treemap-java/) provides quick insert, search and delete operations. However it does not maintain any order on elements inserted into it. If we want to keep track of order of insertion, we can use LinkedHashMap.  
LinkedHashMap is like HashMap with additional feature that we can access elements in their insertion order.

**Similarities Both HashMap and HashSet**  
1) Both HashMap and HashSet are not synchronized which means they are not suitable for thread-safe operations unitl unless synchronized explicitly. This is how you can synchronize them explicitly:  
**HashSet:**

Set s = Collections.synchronizedSet(new HashSet(...));

Databse connection

}

**JDBC In JAVA**

**import**java.sql.DriverManager;

**import**java.sql.Connection;

**import**java.sql.ResultSet;

**import**java.sql.SQLException;

**import**java.sql.Statement;

**public class** JDBCTest {

**publicstaticvoid** main(String[] args)**throws**SQLException, ClassNotFoundException{

**Class**.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

Connection conn =DriverManager.getConnection("jdbc:sqlserver://HOSP\_SQL1.company.com;user=name;password=abcdefg;database=Test");

System.out.println("test");

Statement sta =conn.createStatement();

String Sql ="select \* from testing\_table";

ResultSet rs =sta.executeQuery(Sql);

**while**(rs.next()){

System.out.println(rs.getString("txt\_title"));

}

}

**JDBC Statement**

**Package** *import java.sq*

**Statement:**

|  |
| --- |
| Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. |
|  |

**PreparedStatement :**

|  |
| --- |
| Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. The PreparedStatement interface is a subinterface of Statement. It is used to execute   * recompilation and DB-side caching of the SQL statement leads to overall faster execution and the ability to reuse the same SQL statement in [batches](https://stackoverflow.com/questions/2467125/reusing-a-preparedstatement-multiple-times). * Automatic prevention of [SQL injection](http://en.wikipedia.org/wiki/SQL_injection) [attacks](http://unixwiz.net/techtips/sql-injection.html) by builtin escaping of quotes and other special characters. Note that this requires that you use any of the PreparedStatement setXxx()methods to set the values   parameterized query.  Let's see the example of parameterized query:   1. String sql="insert into emp values(?,?,?)";   **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once. |
|  |

**CallableStatement:** Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

### Commonly used methods of Statement interface:

|  |
| --- |
| **1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet. |
| **2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc. |
| **3) public boolean execute(String sql):** is used to execute queries that may return multiple results. |
| **4) public int[] executeBatch():** is used to execute batch of commands |

**Statement vs PreparedStatement**

1.PreparedStatement is better than Statement because of fast performance.

2.PreparedStatement helps us in preventing SQL injection attacks because it automatically escapes the special characters.

3.PreparedStatement allows us to execute dynamic queries with parameter inputs.

4.In case of PreparedStatement query is parsed, validated, translated and optimized only once and executed each time.

5.But query is parsed, validated, translated, optimized and executed every time when you execute

the query by Statement interface.

the query by Statement interface.

When we fire a query to be executed for a relational database, it goes through following steps.

1. Parsing of SQL query

2. Compilation of SQL Query

3. Planning and optimization of data acquisition path

4. Executing the optimized query and return the resulted data

When we use Statement, it goes through all the four steps but with PreparedStatement

first three steps are executed when we create the prepared statement. So execution of

query takes less time and more quick that Statement.

Another benefit of using PreparedStatement is that we can use Batch Processing through

addBatch() and executeBatch() methods. We can create a

single prepared statement and use it to execute multiple queries.

**Heap vs Stack**

### Java Heap Memory

Heap memory is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space. Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

### Java Stack Memory

Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method. Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as method ends, the block becomes unused and become available for next method.  
Stack memory size is very less compared to Heap memory.

stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call

**Difference between Heap and Stack Memory**

Based on the above explanations, we can easily conclude following differences between Heap and Stack memory.

heap stores objects stack stores primitive and ref data types

1. Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
2. If there is no memory left in the stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating an object, JVM will throw java.lang.OutOfMemoryError:

3. Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

4 Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.

1. Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at [Java Garbage Collection](http://www.journaldev.com/2856/java-jvm-memory-model-and-garbage-collection-monitoring-tuning).
2. Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.
3. We can use **-Xms** and **-Xmx** JVM option to define the startup size and maximum size of heap memory. We can use **-Xss** to define the stack memory size.
4. **When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.**
5. Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

**why stack is faster than heap in java**

**When Heap memory size is exhausted Outofmemory Error comes**

## How do threads interact with the stack and the heap? How do the stack and heap work in multithreading?

|  |
| --- |
|  |

In a multi-threaded application, each thread will have its own stack. But, all the different threads will share the heap. Because the different threads share the heap in a multi-threaded application, this also means that there has to be some coordination between the threads so that they don’t try to access and manipulate the same piece(s) of memory in the heap at the same time.

## Can an object be stored on the stack instead of the heap?

Yes, an object can be stored on the stack. If you create an object inside a function without using the “new” operator then this will create and store the object on the stack, and not on the heap. Suppose we have a C++ class called Member, for which we want to create an object. We also have a function called somefunction( ). Here is what the code would look like:

## Which is faster – the stack or the heap? And why?

The stack is much faster than the heap. This is because of the way that memory is allocated on the stack. Allocating memory on the stack is as simple as moving the stack pointer up.

## Which one should I use – the stack or the heap?

if you know the size of your data is very small. It’s better to use the heap when you know that you will need a lot of memory for your data, or you just are not sure how much memory you will need (like with a dynamic array).

**Abstractions vs. Encapsulation**

Abstractions is basically design where we take data from large amount of data

like for bank form we take 5-10 information of client instead of 1000 info thats abstraction

Abstraction is thought process and Encapsulation is implementation

Encapsulation- Encapsulation only expose the things what is necessary, Hide Complexity

Like Add in data base we use 2 months

1 first validate the data

2. Add in database

**Abstraction** allows us to represent complex real world in simplest manner. It is process of identifying the relevant qualities and behaviors an object should possess, in other word represent the necessary feature without representing the back ground details.

In Java, abstraction is achieved using Abstract classes and interfaces.

**Encapsulation** It is a process of hiding all the internal details of an object from the outside real world. The word Encapsulation, like Enclosing into the capsule. It restrict client from seeing its internal view where behavior of the ab

abstraction is implemented

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

**Memory leak in Java it is a situation where some objects are not used by application any more, but GC fails to recognize them as unused.** As a result, these objects remain in memory indefinitely reducing the amount of memory available to the application.

Memory leak occurs when programmers create a memory in heap and forget to delete it.  
Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate.

RAM can consume memory in 2 types

* 1. Either computation is high in program –not a memory leak
  2. Or Ram is Already holding memory but not using it- Memory leak

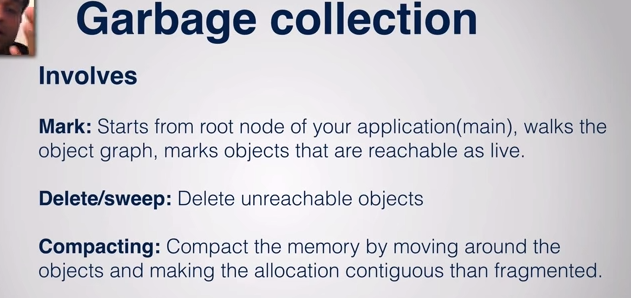
**How to Remove Memory leaks:**

1. Relase db connections

2.Use finally Block

3.Release Instances Stored

**Garbage Collection:**

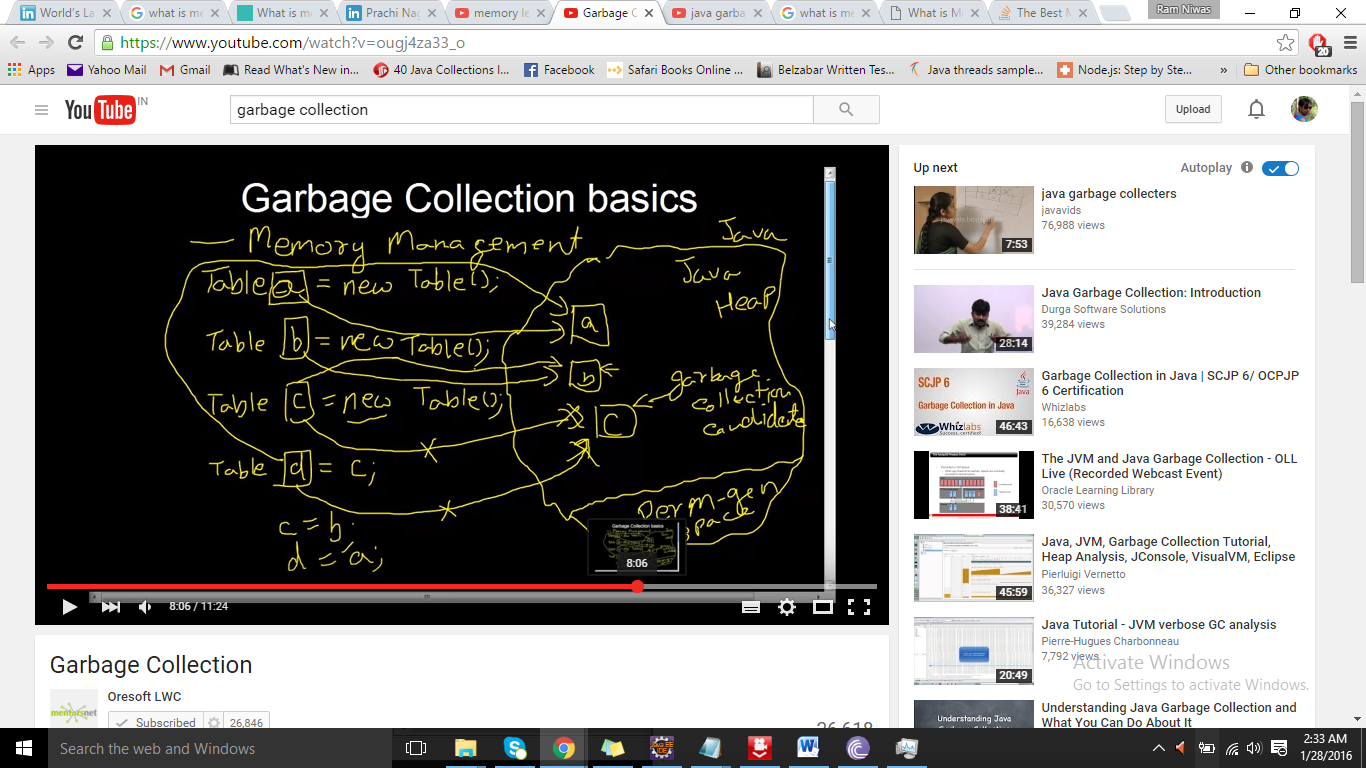
****

The **garbage collector** is a program which runs on the **Java** Virtual Machine which gets rid of objects which are not being used by a **Java** application anymore. It is a form of automatic memory

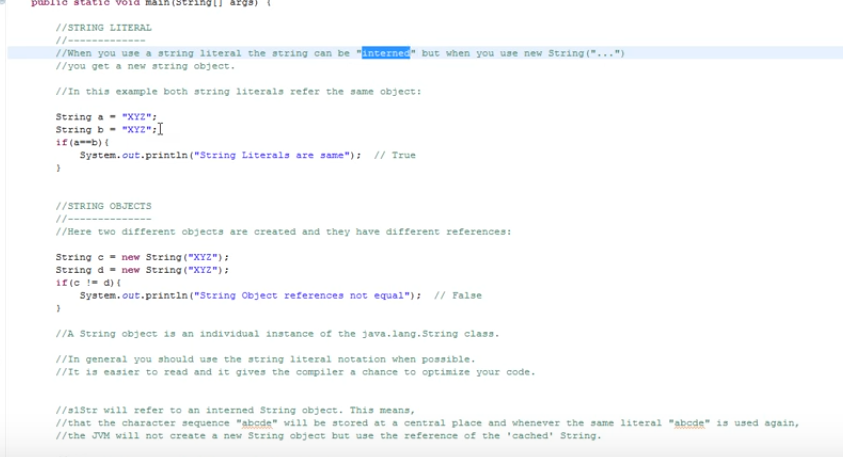
**Fnalize method**

It is a **method** that the [**Garbage Collector**](https://www.geeksforgeeks.org/garbage-collection-java/) always calls just **before** the deletion/destroying the object which is eligible for Garbage Collection, so as to perform **clean-up activity**. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is **not** a reserved keyword.

Once finalize method completes immediately Garbage Collector destroy that object. finalize method is present in Object class and its syntax is:



**String Literal vs String objects**

****

### How to create String object?

|  |
| --- |
| There are two ways to create String object:   1. By string literal 2. By new keyword |

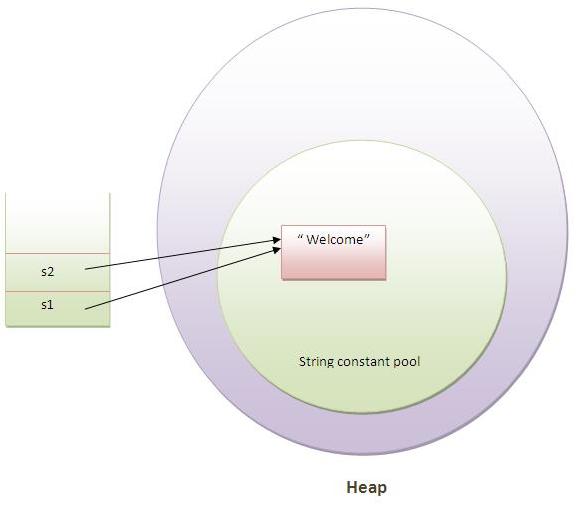
### 1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance



### Java String valueOf() method

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

**int** a=10;

String s=String.valueOf(a);

System.out.println(s+10);

Ans:1010

**String**  
  
Stringis *immutable*  ( once created can not be changed )object  . The object created as a String is stored in the  **Constant String Pool**  .   
Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously.  
String  once assigned can not be changed.

**StringBuffer**  
  
StringBufferis mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap . StringBuffer  has the same methods as the StringBuilder , but **each method in StringBuffer is synchronized**that is **StringBuffer is thread safe** .

But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property . Thus  StringBuilder is faster than the StringBuffer when calling the same methods of each class.

**StringBuilder**

**StringBuilder**  is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that**StringBuilder is also not thread safe.**  
StringBuilder is fast as it is not thread safe .

***String***                ***StringBuffer***        ***StringBuilder***  
----------------------------------------------------------------------------------                   
**Storage Area** | Constant String Pool           Heap                       Heap   
**Modifiable**     |  No (immutable)            Yes( mutable )          Yes( mutable )  
**Thread Safe**   |           Yes                                  Yes                              No

**Performance** |         Fast  (Performance wise,

StringBuffer is faster when performing concatenations.

This is because when you concatenate a String,

you are creating a new object (internally)

every time since String is immutable.)                 Very slow                    Fast

String[] arr={"test1","test2","test3"};

String i="test1";

**if**(arr[0]==i)

{System.***out***.println("done");}

**else**{System.***out***.println("not done");}

}

o/p- done;

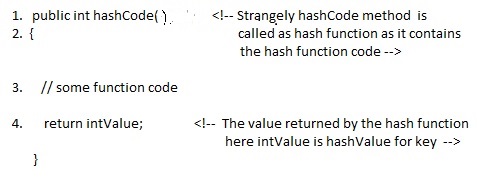
**How HashMap works in java**

hasmap works in hashing algorithm

hashCode() function  which returns an integer value is the **Hash function**. The important point to note that, this method is present in [Object class (Mother of all class)](http://javahungry.blogspot.com/2013/06/object-class-and-methods-in-java-example-explanation.html) .

The most important point to note from the above line :  hashCode method return  int value .

    So summarize the terms in the diagram below :



**What is bucket ?**   
A bucket is used to store key value pairs . A bucket can have multiple key-value pairs . In hash map, bucket used simple linked list to store objects .

**Code inside Java Api (HashMap class internal implementation) for HashMap get(Obejct key) method**

**1**. Public V get(Object key)

{

**2**. **if** (key ==**null**)

**3**. //Some code

**4**. **int** hash = hash(key.hashCode());

**5**. // if key found in hash table then return value

**6**. // else return null

}

HashMap get(Key k) method calls hashCode method  >>>returned hashValue >

\* Whenever we call get( Key k )  method on the HashMap object . First it checks that whether key is null or not .  Note that **there can only be one null key in HashMap .**  
 **If key is null , then Null keys always map to hash 0, thus index 0.**

If key is not null then , it will call hashfunction on the key object , see line 4 in above method i.e. key.hashCode()  ,so after key.hashCode() returns hashValue , line 4 looks like  
  
4.                int hash = hash(hashValue)  
  
 , and now ,it applies returned hashValue into its own hashing function .  
  
**We might wonder why we are calculating the hashvalue again using hash(hashValue).** Answer is ,It defends against poor quality hash functions.

now hashvalue is passed in indexFor method to fine bucket index

\*\*

\* Returns index for hash code h.

\*/

static int indexFor(int h, int length) {

return h & (length-1);

}

Now step 4 final  hashvalue is used to find the bucket location at which the Entry object is stored .**Entry object stores in the bucket like this (hash,key,value,bucketindex) .**

**Interviewer:    What if  when two different keys have the same hashcode ?**  
Solution, [equals() method](http://javahungry.blogspot.com/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html) comes to rescue.Here candidate gets puzzled. Since bucket is one and we have two objects with the same hashcode .Candidate usually forgets that bucket is a simple linked list.

**The bucket is the linked list effectively . Its not a LinkedList as in a java.util.LinkedList - It's a separate (simpler) implementation just for**

**the map .**

Answer is when an element is added/retrieved, same procedure follows:  
  
  
a. Using key.hashCode() [ see above step 4],determine initial hashvalue for the key  
  
b. Pass initial hashvalue as hashValue  in    hash(hashValue) function, to calculate the final hashvalue.  
  
c. Final hash value is then passed as a first parameter in the indexFor(int ,int )method .  
    The second parameter is length which is a constant in HashMap Java Api , represented by                             DEFAULT\_INITIAL\_CAPACITY  
  
    The default  value of DEFAULT\_INITIAL\_CAPACITY is 16 in HashMap Java Api .  
  
 indexFor(int,int) method  returns the first entry in the appropriate bucket. The linked list in the bucket is then iterated over - (the end is found and the element is added or the key is matched and the value is returned )  
  
  
Explanation about indexFor(int,int) is below :

/\*\*

\* Returns index for hash code h.

\*/

**static** **int** **indexFor**(**int** h, **int** length) {

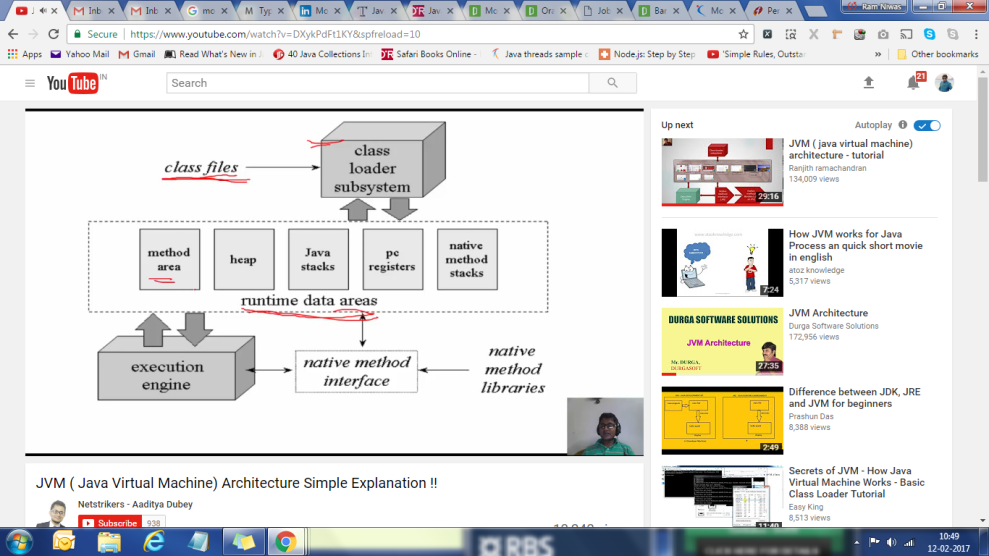
**return** h & (length-**1**);

}

**http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html**

HashMap internally stores mapping in the form of **Map.Entry** object which contains both key and value object. When you want to retrieve the object, you call [the get() method](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) and again pass the key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve the object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location. If there is only one object then it is returned and that's your value object which you have stored earlier. Things get little [tricky](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html) when collisions occur. It's easy to answer this question if you have read good books on data structure and algorithms like [this](http://www.amazon.com/Data-Structures-Algorithm-Analysis-Edition/dp/0132576279?tag=javamysqlanta-20)one. If you know how hash table data structure works then this is a piece of cake.  
http://www.jitendrazaa.com/blog/java/what-is-the-need-to-override-hashcode-and-equals-method/

**RUN time memory**

****

Static methods (in fact all methods) as well as static variables are stored in the **PermGen(method Area) section of the heap**, since they are part of the reflection data (class related data, not instance related).

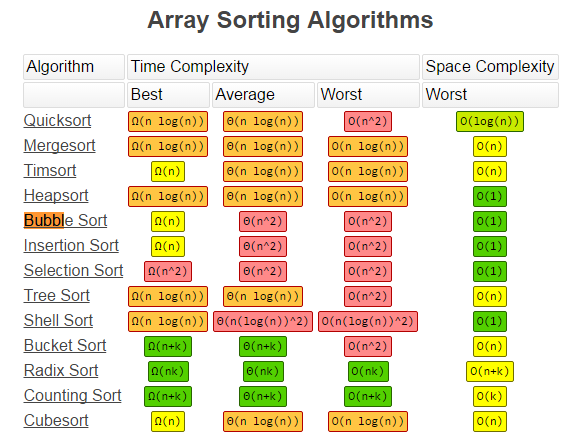
Class variables(Static variables) are stored as part of the Class object associated with that class. This Class object can only be created by JVM and is stored in permanent generation.

### 2)How many types of memory areas are allocated by JVM?

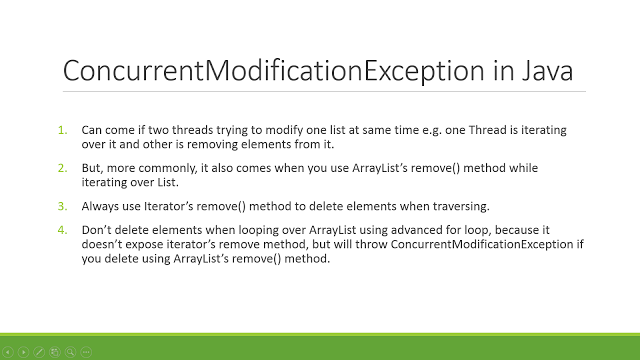
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Many types:  **1) Classloader : Classloader is a subsystem of JVM that is used to load class files.**   1. **Method Area** – All the **class level data** will be stored here, including **static variables**. There is only one method area per JVM, and it is a shared resource. 2. **Heap Area** – All the **Objects** and their corresponding**instance variables** and **arrays** will be stored here. There is also one Heap Area per JVM. Since the **Method** and **Heap areas** share memory for multiple threads, the data stored is not thread**safe.** 3. **Stack Area** – For every thread, a separate **runtime stack** will be created. For every **method call**, one entry will be made in the stack memory which is called as **Stack Frame**. All **local variables** will be created in the stack memory. The stack area is thread safe since it is not a shared resource. The Stack Frame is divided into three subentities:    1. **Local Variable Array** – Related to the method how many **local variables** are involved and the corresponding values will be stored here.    2. **Operand stack** – If any intermediate operation is required to perform, **operand stack** acts as runtime workspace to perform the operation.    3. **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data. 4. **PC Registers** – Each thread will have separate**PC Registers,** to hold the address of **current executing instruction** once the instruction is executed the PC register will be **updated** with the next instruction. 5. **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.   public int math (int x, int y){  A a = new A();  return (A.e + x + y);  }  }   Then we have:   Stack: x, y, a  Heap: instance a (it is A object), a.e = 1   (Note that a in stack points out to instance a in heap)   If instance a is no longer used, it is garbage collected  The class (method) area stores *code* - that's the code of your program. The heap stores object instances. For example:  public void MakeSomeFruit(){  Fruit myFruit=new Fruit();  }   * The MakeSomeFruit code is stored in the class area. * When executed, the actual Fruit instance it creates is stored in the Heap. * When executed, the myFruit *reference* variable is stored on the stack. That's just a number which points at the location of the instance in memory - an address.   Primitives are stored on the stack, objects in the heap space.  int j = 20; will be stored on the stack, because the variable itself holds the value and it's much faster, but if you wrap it: Integer p = new Integer(20); then it's stored in the heap space because it's now an object.  **The String Constant Pool.**  It is a special place where the collection of references to string objects are placed. What makes this so special? we will try to understand it now.  In our tutorial about String immutability, we have learned that string literals cannot be modified and multiple reference variable can refer to the same string literal. Let us first write a program to understand object comparison and references   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public class StringConstantPool {  public static void main(String[] args) {  String s = "prasad";  String s2 = "prasad";    System.out.println(s.equals(s2));  System.out.println(s == s2);  }  } | | 1  2 | True  True |   1) **Iterator** is used for traversing List **and** Set both. We can use **ListIterator** to traverse List only, we cannot traverse Set using **ListIterator**. 2) We can traverse in only forward direction using **Iterator**. Using **ListIterator**, we can traverse a List in both the directions (forward **and** Backward). |

**Runtime Polymorphism - Overriding**

**Compile time- Over loading**

****

java.util.**ConcurrentModificationException** is a very common exception when working with java collection classes. Java Collection classes are fail-fast, which means if the Collection will be changed while some thread is traversing over it using iterator, the iterator.next() will throw**ConcurrentModificationException**.Jun 18, 2016



|  |  |
| --- | --- |
|  | Use an [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) and call [remove()](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html#remove%28%29):  Iterator<String> iter = myArrayList.iterator();  while (iter.hasNext()) {  String str = iter.next();  if (someCondition)  iter.remove();  } |

**There are four different ways to create objects in java:**

**A**. Using new keyword  
This is the most common way to create an object in java. Almost 99% of objects are created in this way.

MyObject object = new MyObject();

**B**. Using Class.forName()  
If we know the name of the class & if it has a public default constructor we can create an object in this way.

MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();

**C**. Using clone()  
The clone() can be used to create a copy of an existing object.

MyObject anotherObject = new MyObject();

MyObject object = (MyObject) anotherObject.clone();

**D**. Using object deserialization

Object deserialization is nothing but creating an object from its serialized form.

ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyObject object = (MyObject) inStream.readObject();

**E:** Using [Class::newInstance()](http://docs.oracle.com/javase/8/docs/api/java/lang/Class.html#newInstance--) method. See [Oracle Tutorial](http://docs.oracle.com/javase/tutorial/reflect/member/ctorInstance.html).

Object object4 = CrunchifyObj.class.getClassLoader().loadClass("crunchify.com.example.CrunchifyObj").newInstance();

**F:** Use the [Constructor](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Constructor.html) class from the [java.lang.reflect](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html) package, part of [Java Reflection](http://docs.oracle.com/javase/tutorial/reflect/index.html) facility.

Class clazz = CrunchifyObj.class;

Constructor crunchifyCon = clazz.getDeclaredConstructors()[0];

CrunchifyObj obj = (CrunchifyObj) crunchifyCon.newInstance();

**String to Byte Arrays**

**String str = "www.journaldev.com";**

**//convert String to byte array**

**byte[] byteArr = str.getBytes();**

**Java 9 Features**

Private method in interface

**JAVA 8 New Features**

## [What is a queue? Explain types of queue.](http://www.ultimatitsolution.in/p/what-is-queue-explain-types-of-queue.html)

• Queue is a Linear Data Structure that works on First-in-First-Out (FIFO) principle.   
• It has two pointers, ‘Front’ that points to the beginning of the queue and ‘Rear’ that points to the end of the queue.  
• The ‘Front’ and ‘Rear’ pointers are manipulated constantly to always point to the beginning and end of queue.  
• It can be implemented using Arrays and Linked Lists (Recursive and Non-recursive) methods both.  
• Different types of queues:  
  
1.Simple or linear queue  
Linked lists are among the simplest and most common data structures.  
They can be used to implement several other common abstract data types, including lists(the abstract data type), stacks, queues, associative arrays, and S-expressions, though it is not uncommon to implement the other data structures directly without using a list as the basis of implementation.  
  
2.Circular queue:  
Another common implementation of a queue is a circular buffer. "Buffer" is a general name for a temporary storage location, although it often refers to an array, as it does in this case.  
A circular buffer, cyclic buffer or ring buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. This structure lends itself easily to buffering data streams.  
  
3.Priority queue:  
The Priority Queue ADT has the same interface as the Queue ADT, but different semantics.  
The semantic difference is that the item that is removed from the queue is not necessarily the first one that was added. Rather, it is whatever item in the queue has the highest priority.  
  
4.Dequeue:  
A double-ended queue (Dequeue) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail).  
It is also often called a head-tail linked list, though properly this refers to a specific data structure implementation.

element() and PEAK())

Remove() and poll()

OFFER() and Add()

**Difference between Fail Fast iterator and Fail Safe iterator**  
  
**Fail fast Iterator**  
  
Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification  of the collection . Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Fail Safe Iterator :

Fail-Safe iterators don’t throw any exceptions if the collection is modified while iterating over it. Because, they iterate on the clone of the collection not on the actual collection. So, any structural modifications done on the actual collection goes unnoticed by these iterators. But, these iterators have some drawbacks. One of them is that it is not always guaranteed that you will get up-to-date data while iterating. Because any modifications to collection after the iterator has been created is not updated in the iterator. One more disadvantage of these iterators is that there will be additional overhead of creating the copy of the collection in terms of both time and memory.

Two  issues associated with Fail Safe Iterator are :  
  
1. Overhead of maintaining the copied data structure i.e memory.  
  
2.  Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.

HashMap(FailFast) and ConcurrentHashMap(FailSafe)

**Interviewer : How  Fail  Fast Iterator  come to know that the internal structure is modified ?**  
Iterator read internal data structure (object array) directly . The internal data structure(i.e object array) should not be modified while iterating through the collection. To ensure this it maintains an internal  flag *"mods" .*Iterator checks the *"mods" flag* whenever it gets the next value (using hasNext() method and next() method). Value of *mods* flag changes whenever there is an structural modification. Thus indicating iterator to throw ConcurrentModificationException.

**String, Integer,File all are Immutable**

**NULL is a reference data type Literal not Keyword**

**null can be assigned to any reference type**

**all are ok**

String k =**null**;

Integer p=**null**;

, **int**[] s =**null**;

**int** y=(Integer) **null**;

**EOFException-**When you reach the end of a stream (end of file, or peer closes the connection):

* read() returns -1
* readLine() returns null
* readXXX() for any other X throws EOFException.

The stream is still open, but you should stop reading from it and close it.

1. if there is no data in a **STREAM** but you are trying to read...eg read methods of chain streams like DataInputStream, ObjectInputStream, RandomAccessFile throw EOFException if they are trying to read from FileInputStream but the FileInputStream is empty.

**Read Property File**

Properties prop = new Properties();

InputStream input = null;

try {

input = new FileInputStream("config.properties");

// load a properties file

prop.load(input);

// get the property value and print it out

System.out.println(prop.getProperty("database"));

System.out.println(prop.getProperty("dbuser"));

System.out.println(prop.getProperty("dbpassword"));

} catch (IOException ex) {

ex.printStackTrace();

} finally {

if (input != null) {

try {

input.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Load Factor : You can also say, load factor is a measure "Till what load, hashmap can allow elements to put in it before its capacity is automatically increased"**

**Default capacity :**

**Vector - 10, No Load factor increases by 100%**

**Arraylist - 10, No Load factor increases by 50%**

**Hashtable - 11,**

**Hashmap - 16, Load factor .75**

**Hashset - 16 Load factor .75**

**Example of calling private method from another class**

1. **public** **class** A {
2. **private** **void** message(){System.out.println("hello java"); }
3. }

*File: MethodCall.java*

1. **import** java.lang.reflect.Method;
2. **public** **class** MethodCall{
3. **public** **static** **void** main(String[] args)**throws** Exception{
5. Class c = Class.forName("A");
6. //Object o= c.newInstance();
7. Method m =c.getDeclaredMethod("message", **null**);
8. m.setAccessible(**true**);
9. m.invoke(o, **null**);
10. }
11. }
12. use setAccessible(true) on your Method object before using its invoke method.
13. import java.lang.reflect.\*;
14. class Dummy{
15. private void foo(){
16. System.out.println("hello foo()");
17. }
18. }
19. class Test{
20. public static void main(String[] args) throws Exception {
21. Dummy d = new Dummy();
22. Method m = Dummy.class.getDeclaredMethod("foo");
23. //m.invoke(d);// throws java.lang.IllegalAccessException
24. m.setAccessible(true);// Abracadabra
25. m.invoke(d);// now its OK
26. }
27. }

The method **compareTo**() is used for comparing two strings lexicographically. Each character of both the strings is converted into a Unicode value for comparison. If both the strings are equal then this method returns 0 else it returns positive or negative value.

# [How to integrate JIRA with Selenium WebDriver?](https://stackoverflow.com/questions/27684899/how-to-integrate-jira-with-selenium-webdriver)

I was able to work with JIRA with Selenium Web Driver based project by integrating [jira client](https://github.com/rcarz/jira-client/blob/master/README.md) as part of my utility classes. [This has nothing to do with selenium web driver]

Below example tells how to log defect in JIRA if any automated test gets failed. [This is just a sample. You can try mentioned scenario in question by yourself with reference taken from sample code below]

JiraClient jira = new JiraClient("<JiraUrl>", "<creds>");

public void createNewJiraIssue(ITestResult result, String projectName, String defectType, String defectSummary,

String defectDescription, String defectReporter, String defectAssignee) {

try {

if (result.getStatus() == ITestResult.FAILURE) {

/\* Create new issue \*/

Issue newIssue = jira.createIssue(projectName, defectType).field(Field.SUMMARY, defectSummary)

.field(Field.DESCRIPTION, defectDescription).field(Field.REPORTER, defectReporter)

.field(Field.ASSIGNEE, defectAssignee).execute();

}

} catch (JiraException ex) {

System.err.println(ex.getMessage());

if (ex.getCause() != null)

System.err.println(ex.getCause().getMessage());

}

}

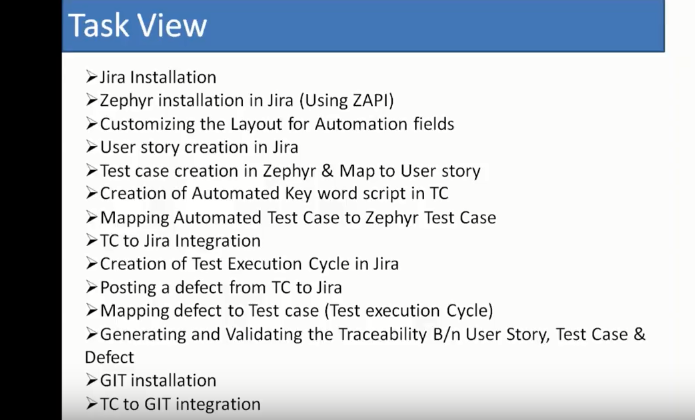
Replace JiraUrl and creds with valid Jira credentials.

For every failure, you can log defect in Jira. You may want to add some intelligence to it in future such as look for duplicate tickets before creating a new one.

**How to file bug in jira by Automation Script**

**https://www.youtube.com/watch?v=vLx4HHhVqzg**

**https://www.youtube.com/watch?v=Ok6vu6Wwf9M**

****