**Immutable objects** are instances whose state doesn’t change after it has been initialized.

For example, String is an immutable class and once instantiated its value never changes.

***Why String class is Immutable?***

String immutable Benefits

1.[String pool](http://www.journaldev.com/797/what-is-java-string-pool) is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space because different String variables can refer to same String variable in the pool. If String would not have been immutable, then String interning would not have been possible because if any variable would have changed the value, it would have been reflected to other variables also.

2. If String is not immutable then it would cause severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](http://www.journaldev.com/741/java-socket-server-client-read-write-example) host and port details passed as String. Since String is immutable it’s value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.

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

4. Strings are used in [java classloader](http://www.journaldev.com/349/java-interview-questions-understanding-and-extending-java-classloader) and immutability provides security that correct class is getting loaded by Classloader. For example, think of an instance where you are trying to load java.sql.Connection class but the referenced value is changed to myhacked.Connection class that can do unwanted things to your database.

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

Above are some of the reasons I could think of that shows benefits of String immutability. It’s a great feature of Java String class and makes it special

**To create a class immutable, you need to follow following steps:**

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables
4. Make all **mutable fields final** so that it’s value can be assigned only once.
5. Initialize all the fields via a constructor performing deep copy.
6. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

**Security**

String is widely used as parameter for many java classes, e.g. network connection, opening files, etc. Were String not immutable, a connection or file would be changed and lead to serious security threat. The method thought it was connecting to one machine, but was not.

Mutable strings could cause security problem in Reflection too, as the parameters are strings.

**Here is a code example:**

boolean connect(string s)

{

if (!isSecure(s))

{

throw new SecurityException();

}

//here will cause problem, if s is changed before this by using other references.

causeProblem(s);

}

**Requirement of String Pool**

String pool (String intern pool) is a special storage area in Java heap. When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object and returning its reference.

The following code will create only one string object in the heap.

String string1 = "abcd";

String string2 = "abcd";

**Immutable objects are naturally thread-safe**

Because immutable objects cannot be changed, they can be shared among multiple threads freely. This eliminates the requirements of doing synchronization.

***In summary, String is designed to be immutable for the sake of efficiency and security. This is also the reason why immutable classes are preferred in general.***

[equals() and hashCode() Contract](http://www.programcreek.com/2011/07/java-equals-and-hashcode-contract/)

HashCode is designed to improve performance. The contract between equals() and hasCode() is that:  
1. If two objects are equal, then they must have the same hash code.  
2. If two objects have the same hashcode, they may or may not be equal.

**String intern()**

intern() method is to be used on Strings constructed with new String()

string intern() method will first find from the String pool, if it finds, then it will return the object that points to that, or will add a new String into the pool.

**1. How to compare strings? Use "==" or use equals()?**

In brief, "==" tests if references are equal and equals() tests if values are equal. Unless we want to check if two strings are the same object, we should always use equals().

**Can we use string for switch statement?**

Yes to version 7. From [JDK 7](http://openjdk.java.net/projects/jdk7/features/), we can use string as switch condition. Before version 6, we can not use string as switch condition.

**switch** (str.toLowerCase()) {

**case** "a":

value = 1;

**break**;

**case** "b":

value = 2;

**break**;

}

**How to convert string to int?**

|  |  |
| --- | --- |
| **int n = Integer.parseInt("10");**  **How to count # of occurrences of a character in a string?**  Use StringUtils from apache commons lang.   |  | | --- | |  | |

**int** n = StringUtils.*countMatches*("11112222", "1");

**How to split a string with white space characters?**

|  |
| --- |
| String[] strArray = aString.split("**\\**s+"); |

***Is jvm is platform independent?***

JVM refers Java Virtual Machine. JVM is a program which will convert byte code instructions into machine language   
instructions understandable by Micro processor. Java program write once, later on run anywhere.  
  
**JVM is a system dependent, because it was developed in C language, where as class file is a system independent.**   
  
Sun Microsystems has developed different jvms for different operating systems. Hence, you can write a single Java   
program and generate the class file which can be executed on any other operating system.

***The JVM must translate the bytecode into machine language, and since the machine language depends on the operating system being used, it is clear that the JVM is platform (operating system) dependent.***

***This fact can be verified by trying to download the JVM - we will be given a list of JVM's corresponding to different operating systems, and you will obviously pick whichever JVM is targeted for the operating system that we are running.***

Yes, generally a JVM is platform dependent in the sense that it is implemented for the specific platform.

## Machine language is OS dependent

JVM must translate the bytecode into machine language, and since the machine language depends on the operating system being used, it is clear that the *JVM is platform (operating system) dependent* – in other words, the JVM is **not** platform independent.

## The JVM is not platform independent

The key here is that the JVM depends on the operating system – so if you are running Mac OS X you will have a different JVM than if you are running Windows or some other operating system. This fact can be verified by trying to download the JVM for your particular machine – when trying to download it, you will be given a list of JVM’s corresponding to different operating systems, and you will obviously pick whichever JVM is targeted for the operating system that you are running.

***Is the Java Virtual Machine written in C language?***

Yes. Because C language is capable of memory management and accessing Hardware. that is why JVM uses C to do all its memory management,threading ,Garbage Collection mechanism and keep java Developer away from all these issues. That is why Java is so popular (everything is done by JVM) So no headache to Java Devloper.

******

* Public are accessible from everywhere.
* Protected are accessible by the classes of the same package and the subclasses residing in any package.
* Default are accessible by the classes of the same package.
* private are accessible within the same class only.

***How HashMap Internally works?***

***---------------------------------------------***  
HashMap works on the principle of Hashing .To understand Hashing , we should understand the three terms first   i.e  *Hash Function , Hash Value and Bucket .*

**What is Hash Function , Hash Value  and Bucket ?**  
  
***Hash function***

***-----------------***

hashCode() function  which returns an integer value is the **Hash function**.

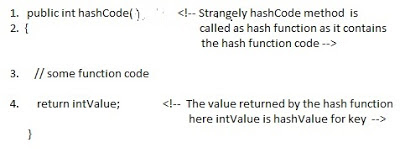
The important point to note that ,  this method is present in [Object class ( Mother of all class )](http://javahungry.blogspot.com/2013/06/object-class-and-methods-in-java-example-explanation.html).  
  
This is the code for the hash function(also known as hashCode method) in Object Class :  
  
    public native int hashCode();

The most important point to note from the above line :

***Hash Value***

hashCode method return  int value .So the **Hash value** isthe int value returned by the hash function **.**

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

}

**Hash map** **works on the principle of hashing**  
  
HashMap get(Key k) method calls hashCode method on the key object and applies returned hashValue to its own static hash function to find a bucket location(backing array) where keys and values are stored in form of a **nested class called Entry (Map.Entry)** .

So we have concluded that from the previous line that **Both key and value is stored in the bucket as a form of  Entry object** . So thinking that Only value is stored  in the bucket is not correct and will not give a good impression on the interviewer .

**HashMap is an array of Entry objects:**  
Consider HashMap as just an array of objects.

**static class Entry<K,V> implements Map.Entry<K,V>**

**{**

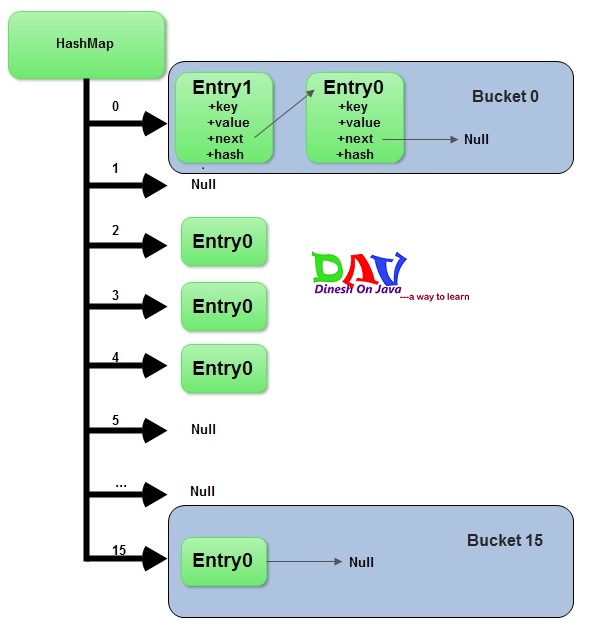
**final K key;**

**V value;**

**Entry<K,V> next;**

**final int hash;**

**}**



Each Entry object represents key-value pair. Field next refers to other Entry object if a bucket has more than 1 Entry.

Sometimes it might happen that ***hashCodes*** for 2 different objects are the same.

In this case 2 objects will be saved in one bucket and will be presented as ***LinkedList***. The entry point is more recently added object. This object refers to other object with next field and so one.

Last entry refers to null.When you create ***HashMap*** with default constructor.

**HashMap hashMap = new HashMap();**

Array is gets created with size 16 and default 0.75 load balance.

**Adding a new key-value pair**

1. Calculate ***hashcode*** for the key
2. Calculate position hash % (arrayLength-1)) where element should be placed(bucket number)
3. If you try to add a value with a key which has already been saved in ***HashMap***, then value gets overwritten.
4. Otherwise element is added to the bucket. If bucket has already at least one element - a new one is gets added and placed in the first position in the bucket. Its next field refers to the old element.

**What *put()* method actually does:**  
Before going into **put()** method’s implementation, it is very important to learn that instances of ***Entry*** class are stored in an array. ***HashMap*** class defines this variable as:

**/\*\***

**\* The table, resized as necessary. Length MUST Always be a power of two.**

**\*/**

**transient Entry[] table;**

code implementation of ***put()*** method:

**public V put(K key, V value)**

**{**

**if (key == null)**

**return putForNullKey(value);**

**int hash = hash(key.hashCode());**

**int i = indexFor(hash, table.length);**

**for (Entry<k , V> e = table[i]; e != null; e = e.next)**

**{**

**Object k;**

**if (e.hash == hash && ((k = e.key) == key || key.equals(k)))**

**{**

**V oldValue = e.value;**

**e.value = value;**

**e.recordAccess(this);**

**return oldValue;**

**}**

**}**

**modCount++;**

**addEntry(hash, key, value, i);**

**return null;**

**}**

**Step1-** First of all, key object is checked for null. If key is null, value is stored in table[0] position. Because hash code for null is always 0.  
  
**Step2**- Then on next step, a hash value is calculated using key’s hash code by calling its **hashCode()** method. This hash value is used to calculate index in array for storing Entry object. JDK designers well assumed that there might be some poorly written ***hashCode()*** functions that can return very high or low hash code value. To solve this issue, they introduced another hash() function, and passed the object’s hash code to this hash() function to bring hash value in range of array index size.  
  
**Step3-** Now ***indexFor(hash, table.length)*** function is called to calculate exact index position for storing the Entry object.  
  
**Step4**- Here comes the main part. Now, as we know that two unequal objects can have same hash code value, how two different objects will be stored in same array location [called bucket].  
  
Answer is **LinkedList**. If you remember, Entry class had an attribute “next”. This attribute always points to next object in chain. This is exactly the behavior of LinkedList.  
  
So, in case of collision, Entry objects are stored in LinkedList form. When an Entry object needs to be stored in particular index, HashMap checks whether there is already an entry?? If there is no entry already present, Entry object is stored in this location.  
  
If there is already an object sitting on calculated index, its next attribute is checked. If it is null, and current Entry object becomes next node in LinkedList. If next variable is not null, procedure is followed until next is evaluated as null.  
  
What if we add the another value object with same key as entered before. Logically, it should replace the old value. How it is done? Well, after determining the index position of Entry object, while iterating over LinkedList on calculated index, HashMap calls equals method on key object for each Entry object. All these Entry objects in LinkedList will have similar hash code but equals() method will test for true equality. If key.equals(k) will be true then both keys are treated as same key object. This will cause the replacing of value object inside Entry object only.  
  
In this way, HashMap ensure the uniqueness of keys.

1. Key object is checked for null. If key is null then it will be stored at table[0] because hash code for null is always 0.
2. Key object’s hashcode() method is called and hash code is calculated. This hashcode is used to find index of array for storing Entry object. It may happen sometimes that, this hashcode function is poorly written so JDK designer has put another function called hash() which takes above calculated hash value as argument.If you want to learn more about hash() function, you can refer [hash and indexFor method in hashmap](http://javapostsforlearning.blogspot.in/2014/02/hash-and-indexfor-method-in-hashmap.html).
3. indexFor(hash,table.length)  is used to calculate exact index in table array for storing the Entry object.
4. As we have seen in our example, if two key objects have same hashcode(which is known as **collision**) then it will be stored in form of linkedlist.So here, we will iterate through our linkedlist.

* If there is no element present at that index which we have just calculated then it will directly put our Entry object at that index.
* If There is element present at that index then it will iterate until it gets Entry->next as null.Then current Entry object become next node in that linkedlist
* What if we are putting same key again, logically it should replace old value. Yes,it will do that.While iterating it will check key equality by calling equals() method(**key.equals(k)**), if this method returns true then it replaces value object with current Entry’s value object.

**How *get()* methods works internally**

**public V get(Object key)**

**{**

**if (key == null)**

**return getForNullKey();**

**int hash = hash(key.hashCode());**

**for (Entry<k , V> e = table[indexFor(hash, table.length)]; e != null; e = e.next)**

**{**

**Object k;**

**if (e.hash == hash && ((k = e.key) == key || key.equals(k)))**

**return e.value;**

**}**

**return null;**

**}**

As you got the understanding on put functionality of hashmap. So to understand get functionality is quite simple. If you pass any key to get value object from hashmap.

1. Key object is checked for null. If key is null then value of Object resides at table[0] will be returned.
2. Key object’s hashcode() method is called and hash code is calculated.
3. indexFor(hash,table.length)  is used to calculate exact index in table array using generated hashcode for getting the Entry object.
4. After getting index in table array, it will iterate through linkedlist and check for key equality by calling equals() method and if it returns true then it returns the value of Entry object else returns null.

## Key points to Remeber:

* HashMap has a inner class called Entry which stores key-value pairs.
* Above Entry object is stored in Entry[ ](Array) called table
* An index of table is logically known as bucket and it stores first element of linkedlist
* Key object’s hashcode() is used to find bucket of that Entry object.
* If two key object ‘s have same hashcode , they will go in same bucket of table array.
* Key object ‘s equals() method is used to ensure uniqueness of key object.
* Value object  ‘s equals() and hashcode() method is not used at all

**Volatile Keyword**

**What is Volatile variable in Java?**

volatile variable in Java is a special variable which is used to signal threads, Volatile variables values is going to be updated by multiple thread inside Java application.

By making a variable volatile using volatile keyword in Java, application programmer ensures that its value should always been read from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) and thread should not used cached value of that variable from their own stack.

**Important point related to volatile keyword in Java**

1) Volatile keyword can only be applied to variable; it cannot be applied to class or method. Using volatile keyword along with class and method is compiler error.

**When to use Volatile variable in Java**

This is the most important thing to learn while learning about *volatile variable in Java*. When to use volatile variable in Java is also a famous multi-threading interview question in Java. here are some of the scenario where you can use volatile variable in Java :  
  
1)if Any variable which is shared between multiple threads and it ensure that all thread must see latest/updated value of volatile variable.  
  
2) A signal to compiler and JIT to ensure that compiler does not change ordering or volatile variable and moves them out of synchronized context.  
  
3) We want to save cost of synchronization as volatile variables are less expensive than synchronization.

When a field is declared as volatile, then, the Java Memory Model ensures that all threads will “see” the updated value.

## Difference between String and StringBuffer in Java

1) First and most significant *difference between String and StringBuffer in Java* is that [String is immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html) while StringBuffer is mutable.

What this means is, performing any operation on String will create new String object while modifying StringBuffer object won't create new object.  
  
2) If we are using  + operator for concatenating multiple String than we should not be worried much because  based upon Java implementation call to + operator is replaced with either StringBuffer or StringBuider based upon [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) implements Java 1.5 or lower version.  
  
3) StringBuffer.append()method is used to perform String concatenation in Java.  
  
4) Creating StringBuffer from String is easy, as StringBuffer accept an String input. Similarly converting Stringbuffer to String is also easy by using toString() method in Java.  
  
5)Another significant difference between String and StringBuffer is that StringBuffer and String does not share same type hierarchy, means you can not cast String to StringBuffer in Java. any such attempt will result in ClassCastException in Java.  
  
That's all on [Difference between String and StringBuffer in Java](http://javabuddy.hubpages.com/hub/Difference-between-String-and-StringBuffer-in-Java). Most important difference to remember between String and StringBuffer is mutability.

**A class can not be declared with synchronized keyword. Then, why we call classes like Vector, StringBuffer are synchronized classes.?**

Any classes which have only synchronized methods and blocks are treated as synchronized classes. Classes like Vector, StringBuffer have only synchronized methods. That’s why they are called as synchronized classes.

***Can we declare a class as private?***  
Ans: yes… we can declare a class as a private… but it is must be a inner class of the class…. for example

class A

{

private class B

{

}  
}  
here class B can accessible with in class A… successfully

**Can we declare a class as protected.?**

We can’t declare an outer class as protected. But, we can declare an inner class (class as a member of another class) as protected.

**Can we declare an abstract method as private also.?**

No, abstract methods cannot be private. They must be public or protected or default so that they can be modified further.

**Exception:**

**An exception is an abnormal condition which occurs during run time and disrupts the normal flow of the program.**

**How YOU CONTROL THE VERSIONING OF YOUR APPLICATION?**

***If super class method is not throwing any exceptions, then it can be overrided with any unchecked type of exceptions, but cannot be overrided with checked type of exceptions.***

**class SuperClass**

**{**

**void methodOfSuperClass()**

**{**

**System.*out*.println("Super class method is not throwing any exceptions");**

**}**

**}**

**class SubClass extends SuperClass**

**{**

**@Override**

**void methodOfSuperClass() throws ArrayIndexOutOfBoundsException**

**{**

**System.*out*.println("can be overrided with any unchecked Exception");**

**}**

**}**

**class SubClassOne extends SuperClass**

**{**

**@Override**

**void methodOfSuperClass() throws NumberFormatException, NullPointerException, RuntimeException**

**{**

**System.*out*.println("Can be overrided with any number of Unchecked Exceptions");**

**}**

**}**

**class SubClassTwo extends SuperClass**

**{**

**@Override**

**void methodOfSuperClass() throws SQLException**

**{**

**//Compile time error**

**//Can not be overrided with checked exception**

**}**

**}**

***If a super class method is throwing unchecked exception, then it can be overrided in the sub class with same exception or any other unchecked exceptions but cannot be overrided with checked exceptions.***

**class SuperClass**

**{**

**void methodOfSuperClass() throws ArrayIndexOutOfBoundsException**

**{**

**System.*out*.println("Super class method is throwing Unchecked exception");**

**}**

**}**

**class SubClass extends SuperClass**

**{**

**@Override**

**void methodOfSuperClass() throws ArrayIndexOutOfBoundsException**

**{**

**System.*out*.println("Can be Overrided with same unchecked exception");**

**}**

**}**

**class SubClassOne extends SuperClass**

**{**

**@Override**

**void methodOfSuperClass() throws NumberFormatException, NullPointerException, RuntimeException**

**{**

**System.*out*.println("Can be overrided with any other Unchecked Exceptions");**

**}**

**}**

**class** SubClassTwo **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** IOException

{

//Compile time error

//Can not be overrided with checked exception

}

}

***If super class method is throwing checked type of exception, then it can be overrided with same exception or with it’s sub class exceptions i.e we can decrease the scope of the exception, but cannot be overrided with it’s super class exceptions i.e you cannot increase the scope of the exception.***

**class** SuperClass

{

**void** methodOfSuperClass() **throws** IOException

{

System.***out***.println("Super class method is throwing checked exception");

}

}

**class** SubClass **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** IOException

{

System.***out***.println("Can be Overrided with same checked exception");

}

}

**class** SubClassOne **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** FileNotFoundException

{

System.***out***.println("Can be overrided with checked Exception with lesser scope");

}

}

**class** SubClassTwo **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** NullPointerException, ArrayIndexOutOfBoundsException, FileNotFoundException

{

System.***out***.println("Can be overrided with any unchecked exceptions and checked exception with lesser scope");

}

}

**class** SubClassThree **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** Exception

{

//Compile time error

//Can not be overrided with checked exception with higher scope

}

}

***One more example with overriding a method throwing both checked and unchecked exceptions.***

**class** SuperClass

{

**void** methodOfSuperClass() **throws** IOException, ClassNotFoundException, NumberFormatException

{

System.***out***.println("Super class method is throwing both checked and unchecked exceptions");

}

}

**class** SubClass **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** IOException, ClassNotFoundException

{

System.***out***.println("Can be Overrided with same checked exceptions");

}

}

**class** SubClassOne **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** FileNotFoundException

{

System.***out***.println("Can be overrided with checked Exception with lesser scope");

}

}

**class** SubClassTwo **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** NullPointerException, ArrayIndexOutOfBoundsException, FileNotFoundException

{

System.***out***.println("Can be overrided with any other unchecked exceptions and checked exception with lesser scope");

}

}

**class** SubClassThree **extends** SuperClass

{

@Override

**void** methodOfSuperClass() **throws** Exception

{

//Compile time error

//Can not be overrided with checked exception with higher scope

}

}

1)  Checked Exceptions

2)  Unchecked Exceptions.

## 1) Checked Exceptions :

Checked exceptions are the exceptions which are known during compile time. These are the exceptions that are checked at compile time. They are also called compile time exceptions.

These exceptions must be handled either using try-catch blocks or using throws clause. If not handled properly, they will give compile time error.

All sub classes of java.lang.Exception except sub classes of RunTimeException are checked exceptions.

The following example throws checked exception. But it is not handled, so it gives compile time error.

**public** **class** CheckedException

{

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

{

**try**

{

Class.forName("AnyClassName");

}

**catch** (ClassNotFoundException ex)

{

System.out.println("ClassNotFoundException will be caught here");

}

}

}

or

**public** **class** CheckedException

{

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

{

Class.forName("AnyClassName");

}

}

## 2) Unchecked Exceptions :

Unchecked exceptions are the exceptions which are known  at run time. They cannot be known at compile time because they occur only at run time. That’s why they are also called **Run Time Exceptions**.

All the sub classes of RunTimeException and all sub classes of Error class are unchecked exceptions.

If any statement in the program throws unchecked exceptions and you are not handling them either using try-catch blocks or throws clause, then it does not give compile time error. Compilation will be successful but program may fail at run time. Therefore, to avoid premature termination of the program, you have to handle them properly.

**public** **class** UncheckedException

{

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

{

**int** i = Integer.*parseInt*("Unchecked Exception");

//Above statement throws NumberFormatException which is an unchecked exception

}

}

**public** **class** UncheckedException

{

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

{

**try**

{

**int** i = Integer.parseInt("Unchecked Exception");

//Above statement throws NumberFormatException which is an unchecked exception

}

**catch** (NumberFormatException e)

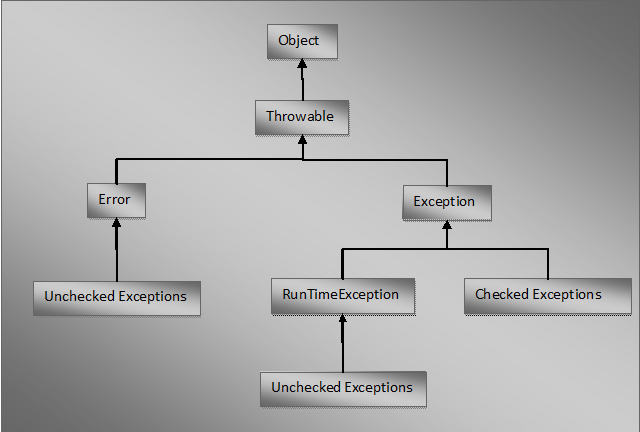
{

System.out.println("NumberFormatException will be caught here");

}

}

}

****

## Difference Between Checked And Unchecked Exceptions :

|  |  |
| --- | --- |
| **Checked Exceptions** | **Unchecked Exceptions** |
| They are known at compile time. | They are known at run time. |
| They are checked at compile time. | They are not checked at compile time. Because they occur only at run time. |
| These are compile time exceptions. | These are run time exceptions. |
| If  these exceptions are not handled properly in the application, they give compile time error. | If these exceptions are not handled properly, they don’t give compile time error. But application will be terminated prematurely at run time. |
| All sub classes of java.lang.Exception Class except sub classes of RunTimeException are checked exceptions. | All sub classes of RunTimeException and sub classes of java.lang.Error are unchecked exceptions. |

**user defined exceptions in java OR Customized exceptions**.

User defined exceptions must extend any one of the classes in the hierarchy of exceptions.

**public** **class** ExceptionHandling

{

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

{

Scanner sc = **new** Scanner(System.***in***); //Declaring Scanner variable to take input from user

System.***out***.println("Enter Your Age");

**int** age = sc.nextInt(); //Taking input from user

**try**

{

**if**(age < 0)

{

**throw** **new** Exception(); //throws an Exception if age is negative

}

}

**catch**(Exception ex)

{

System.***out***.println(ex); //Prints exception description

}

}

}

When user enters negative value in the above example, it throws an exception and prints exception description which user may not understand. So, Let’s make this program more user friendly. Modify above example so that user can understand why the exception has occurred. To do this, create one sub class to Exception class and override toString() method.

//Defining Our own exception by extending Exception class

**class** AgeIsNegativeException **extends** Exception

{

String errorMessage;

**public** AgeIsNegativeException(String errorMessage)

{

**this**.errorMessage = errorMessage;

}

//Modifying toString() method to display customized error message

@Override

**public** String toString()

{

**return** errorMessage;

}

}

Above defined exception is called **user defined exception or customized exception.** Now throw this customized exception when user enters negative value.

**public** **class** ExceptionHandling

{

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

{

Scanner sc = **new** Scanner(System.***in***); //Declaring Scanner variable to take input from user

System.***out***.println("Enter Your Age");

**int** age = sc.nextInt(); //Taking input from user

**try**

{

**if**(age < 0)

{

**throw** **new** AgeIsNegativeException("Age can not be negative"); //throws AgeIsNegativeException if age is negative

}

}

**catch**(AgeIsNegativeException ex)

{

System.***out***.println(ex); //Output : Age can not be negative

}

}

}

Now, this prints “Age can not be negative” when user enters a negative value. This makes the user understand easily why the error has occurred.

//Defining Our own exception class by extending ArithmeticException class

**class** InvalidWithdrawlMoneyException **extends** ArithmeticException

{

//Overriding toString() method of ArithmeticException as per our needs

@Override

**public** String toString()

{

**return** "You don't have that much of money in your account";

}

}

//Using above customized ArithmeticException

**public** **class** ExceptionHandling

{

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

{

**int** balance = 5000; //Initializing the balance

Scanner sc = **new** Scanner(System.***in***); //Scanner variable to take input from user

System.***out***.println("Enter Withdrawl Money");

**int** withdrawlMoney = sc.nextInt(); //taking input from the user

**try**

{

//checking withdrawl money with the balance

//if withdrawl money is more than the balance,

//then it throws Exception

**if**(withdrawlMoney > balance)

{

**throw** **new** InvalidWithdrawlMoneyException();

}

**else**

{

System.***out***.println("Transaction Successful");

}

}

**catch**(InvalidWithdrawlMoneyException ex)

{

//InvalidWithdrawlMoneyException will be caught here

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

}

}

}

We can throw modified exception using anonymous inner class also. Whenever exception occurs, create anonymous inner class, override toString() method and throw the exception. No need to define exception class separately. Above example can be written using anonymous inner classs as,

**public** **class** ExceptionHandling

{

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

{

**int** balance = 5000; //Initializing the balance

Scanner sc = **new** Scanner(System.in); //Scanner variable to take input from user

System.out.println("Enter Withdrawl Money");

**int** withdrawlMoney = sc.nextInt(); //taking input from the user

**try**

{

//checking withdrawl money with the balance

//if withdrawl money is more than the balance,

//then it throws Exception

**if**(withdrawlMoney > balance)

{

//throwing exception using anonymous inner class

**throw** **new** ArithmeticException()

{

@Override

**public** String toString()

{

**return** "You don't have that much of money in your account";

}

};

}

**else**

{

System.out.println("Transaction Successful");

}

}

**catch**(ArithmeticException ex)

{

System.out.println(ex);

}

}

}

|  |  |
| --- | --- |
| **Errors** | **Exceptions** |
| Errors in java are of type java.lang.Error. | Exceptions in java are of type java.lang.Exception. |
| All errors in java are unchecked type. | Exceptions include both checked as well as unchecked type. |
| Errors happen at run time. They will not be known to compiler. | Checked exceptions are known to compiler where as unchecked exceptions are not known to compiler because they occur at run time. |
| It is impossible to recover from errors. | we can recover from exceptions by handling them through try-catch blocks. |
| Errors are mostly caused by the environment in which application is running. | Exceptions are mainly caused by the application itself. |
| Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, ClassCastException, NullPointerException |

## What is the difference between throw, throws and Throwable in java?

## throw In Java :

**Throw** is a keyword in java which is used to throw an exception manually. Using throw keyword, you can throw an exception from any method or block. But, that exception must be of type **java.lang.Throwable** class or it’s sub classes. Below example shows how to throw an exception using throw keyword.

**class** ThrowAndThrowsExample

{

**void** method() **throws** Exception

{

Exception e = **new** Exception();

**throw** e; //throwing an exception using 'throw'

}

}

## throws In Java :

**throws** is also a keyword in java which is used in the method signature to indicate that this method may throw mentioned exceptions. The caller to such methods must handle the mentioned exceptions either using try-catch blocks or using throws keyword. Below is the syntax for using throws keyword.

return\_type method\_name(parameter\_list) **throws** exception\_list

{

//some statements

}

**class** ThrowsExample

{

**void** methodOne() **throws** SQLException

{

//This method may throw SQLException

}

**void** methodTwo() **throws** IOException

{

//This method may throw IOException

}

**void** methodThree() **throws** ClassNotFoundException

{

//This method may throw ClassNotFoundException

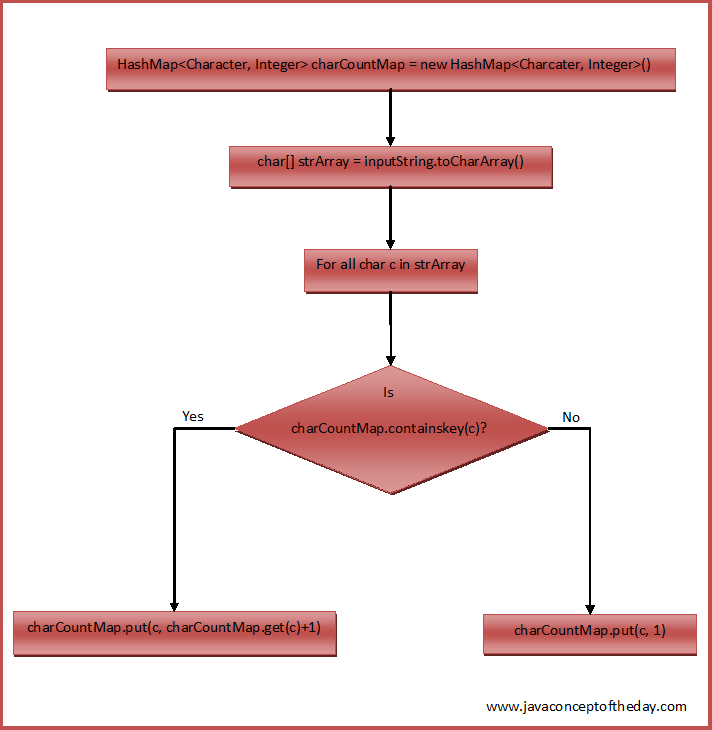
}

}

## Throwable In Java :

**Throwable** is a super class for all types of errors and exceptions in java. This class is a member of **java.lang** package. Only instances of this class or it’s sub classes are thrown by the java virtual machine or by the throw statement. The only argument of catch block must be of this type or it’s sub classes. If you want to create your own customized exceptions, then your class must extend this class. Click [here](http://javaconceptoftheday.com/hierarchy-exceptions-java/) to see the hierarchy of exception classes in java.

***String class***

****

**class EachCharCountInString**

**{**

**static void characterCount(String inputString)**

**{**

**//Creating a HashMap containing char as a key and occurrences as a value**

**HashMap<Character, Integer> charCountMap = new HashMap<Character, Integer>();**

**//Converting given string to char array**

**char[] strArray = inputString.toCharArray();**

**//checking each char of strArray**

**for (char c : strArray)**

**{**

**if(charCountMap.containsKey(c))**

**{**

**//If char is present in charCountMap, incrementing it's count by 1**

**charCountMap.put(c, charCountMap.get(c)+1);**

**}**

**else**

**{**

**//If char is not present in charCountMap,**

**//putting this char to charCountMap with 1 as it's value**

**charCountMap.put(c, 1);**

**}**

**}**

**//Printing the charCountMap**

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

**}**

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

**{**

***characterCount*("Java J2EE Java JSP J2EE");**

***characterCount*("All Is Well");**

***characterCount*("Done And Gone");**

**}**

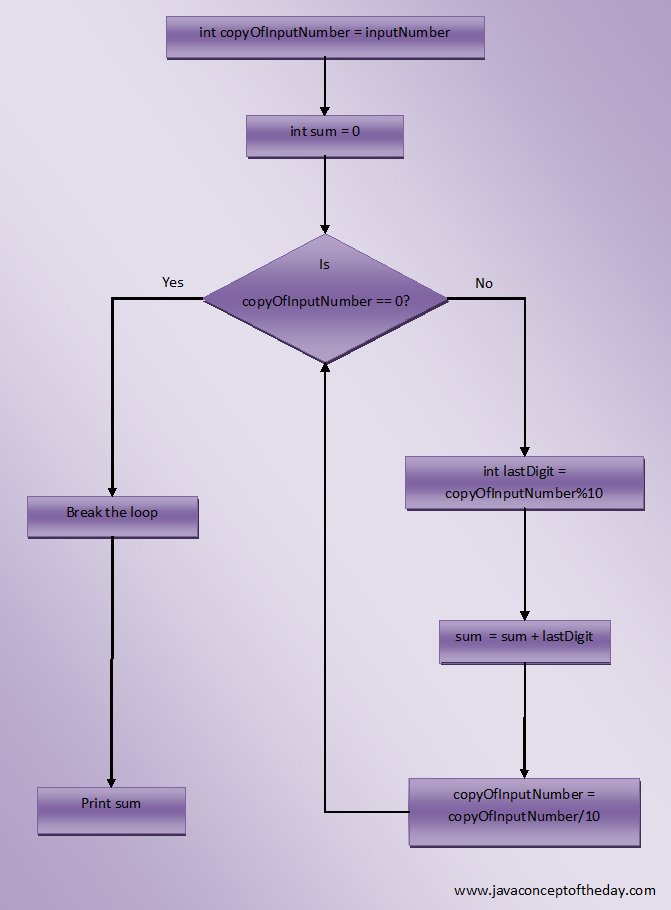
**}**

**Output :**

{E=4, 2=2, v=2, =4, P=1, S=1, a=4, J=5}  
{W=1, =2, e=1, s=1, A=1, l=4, I=1}  
{D=1, d=1, =2, G=1, e=2, A=1, n=3, o=2}

Sum of all the digit of a number:

----------------------------------------

****

**public class MainClass**

**{**

**static void sumOfAllDigits(int inputNumber)**

**{**

**//Creating a copy of input number**

**int copyOfInputNumber = inputNumber;**

**//Initializing sum to 0**

**int sum = 0;**

**while (copyOfInputNumber != 0)**

**{**

**//Getting last digit of the input number**

**int lastDigit = copyOfInputNumber%10;**

**//Adding last digit to sum**

**sum = sum + lastDigit;**

**//Removing last digit from the input number**

**copyOfInputNumber = copyOfInputNumber/10;**

**}**

**//Printing sum**

**System.*out*.println("Sum Of All Digits In "+inputNumber+" = "+sum);**

**}**

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

**{**

***sumOfAllDigits*(47862);**

***sumOfAllDigits*(416872);**

***sumOfAllDigits*(5674283);**

***sumOfAllDigits*(475496215);**

**}**

**}**

Sum Of All Digits In 47862 = 27  
Sum Of All Digits In 416872 = 28  
Sum Of All Digits In 5674283 = 35  
Sum Of All Digits In 475496215 = 43

I am looking for an opportunity where i can found more challenging work to improve my work skills knowledge and communition.

I am looking for an opportunity where i can explore my technical skill to improve the company growth as well as my personal growth.

***Write a java program to find the duplicate words and their number of occurrences in a string?***

**public class duplicateWordsInString**

**{**

**static void duplicateWords(String inputString)**

**{**

**//Splitting inputString into words**

**String[] words = inputString.split(" ");**

**//Creating one HashMap with words as key and their count as value**

**HashMap<String, Integer> wordCount = new HashMap<String, Integer>();**

**//Checking each word**

**for (String word : words)**

**{**

**//whether it is present in wordCount**

**if(wordCount.containsKey(word.toLowerCase()))**

**{**

**//If it is present, incrementing it's count by 1**

**wordCount.put(word.toLowerCase(), wordCount.get(word.toLowerCase())+1);**

**}**

**else**

**{**

**//If it is not present, put that word into wordCount with 1 as it's value**

**wordCount.put(word.toLowerCase(), 1);**

**}**

**}**

**//Extracting all keys of wordCount**

**Set<String> wordsInString = wordCount.keySet();**

**//Iterating through all words in wordCount**

**for (String word : wordsInString)**

**{**

**//if word count is greater than 1**

**if(wordCount.get(word) > 1)**

**{**

**//Printing that word and it's count**

**System.*out*.println(word+" : "+wordCount.get(word));**

**}**

**}**

**}**

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

**{**

***duplicateWords*("Bread butter and bread");**

***duplicateWords*("Java is java again java");**

***duplicateWords*("Super Man Bat Man Spider Man");**

**}**

**}**

**Write a java program to count the number of words in a string?**

**class CountTheWords**

**{**

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

**{**

**System.*out*.println("Enter the string");**

**Scanner sc = new Scanner(System.*in*);**

**String s=sc.nextLine();**

**String[] words = s.trim().split(" ");**

**System.*out*.println("Number of words in the string = "+words.length);**

**}**

**}**

One more method to count the number of words in a string.

**class** CountTheWords

{

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

{

System.out.println("Enter the string");

Scanner sc = **new** Scanner(System.in);

String s=sc.nextLine();

**int** count = 1;

**for** (**int** i = 0; i < s.length()-1; i++)

{

**if**((s.charAt(i) == ' ') && (s.charAt(i+1) != ' '))

{

count++;

}

}

System.out.println("Number of words in a string = "+count);

}

}

**Write a java program to count the total number of occurrences of a given character in a string without using any loop?**

**class** CountCharacterOccurence

{

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

{

String s = "Java is java again java again";

**char** c = 'a';

**int** count = s.length() - s.replace("a", "").length();

System.***out***.println("Number of occurances of 'a' in "+s+" = "+count);

}

}

# Algorithm to reverse a string array in O(n/2) complexity.

There are many ways of reversing a string but best possible case should be considered keeping space and time complexities in mind.

Steps to solve this algorithm

* take two pointers, one pointing to the start of the array and another to the end of the array and swap the values at respective pointers.
* While traversing the array increment starting pointer and decrement ending pointer and continue swap the values at respective pointers
* continue this till we reach the middle of the array
* **Space Complexity:** No additional space is required as we are just swapping the letters within the same array  
  **Time Complexity:**Order of this algorithm: O(n/2), where is ‘n’ is the length of the input array

***public class ReverseString***

***{***

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

***{***

***char[] arr = "welcome to coding algorithms".toCharArray();***

***System.out.println("Length of char[] arrays is:-"+arr.length);***

***// reverse the string***

***reverse(arr, 0, arr.length / 2, arr.length);***

***System.out.println(new String(arr));***

***}***

***private static void reverse(char[] arr, int wordIFirstIndex, int wordMidIdx,int wordLastIdx)***

***{***

***for (; wordIFirstIndex < wordMidIdx; wordIFirstIndex++)***

***{***

***// swap first letter with the last letter in the***

***char tmp = arr[wordIFirstIndex];***

***arr[wordIFirstIndex] = arr[wordLastIdx - 1];***

***arr[wordLastIdx - 1] = tmp;***

***wordLastIdx--;***

***}***

***}***

***}***

## *Write a java program to remove all white spaces from a string.?*

**class RemoveWhiteSpaces**

**{**

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

**{**

**String str = "Core Java jsp servlets jdbc struts";**

**//1. Using replaceAll() Method**

**String strWithoutSpace = str.replaceAll("\\s", "");**

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

**//Output : CoreJavajspservletsjdbcstrutshibernatespring**

**//2. Without Using replaceAll() Method**

**char[] strArray = str.toCharArray();**

**StringBuffer sb = new StringBuffer();**

**for (int i = 0; i < strArray.length; i++)**

**{**

**if( (strArray[i] != ' ') && (strArray[i] != '\t') )**

**{**

**sb.append(strArray[i]);**

**}**

**}**

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

**//Output : CoreJavajspservletsjdbcstrutshibernatespring**

**}**

**}**

[***Check Whether One String Is Rotation Of Another?***](http://javaconceptoftheday.com/check-one-string-is-rotation-of-another/)

If s1 and s2 are two given strings, then write a java program to check whether s2 is a rotated version of s1?

**Examples :**

If “JavaJ2eeStrutsHibernate” is a string then below are some rotated versions of this string.

“StrutsHibernateJavaJ2ee”, “J2eeStrutsHibernateJava”, “HibernateJavaJ2eeStruts”.

**Solution :**

Step 1 : Check whether s1 and s2 are of same length. If they are not of same length then s2 is not rotated version of s1.

Step 2 : s3 = s1 + s1;

If s1 = “JavaJ2eeStrutsHibernate” then s3 = “JavaJ2eeStrutsHibernateJavaJ2eeStrutsHibernate”.

Step 3 : Check whether s3 contains s2 using contains() method of String class. If it contains then s2 is rotated version of s1.

**public class MainClass**

**{**

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

**{**

**String s1 = "JavaJ2eeStrutsHibernate";**

**String s2 = "StrutsHibernateJavaJ2ee";**

**//Step 1**

**if(s1.length() != s2.length())**

**{**

**System.out.println("s2 is not rotated version of s1");**

**}**

**else**

**{**

**//Step 2**

**String s3 = s1 + s1;**

**//Step 3**

**if(s3.contains(s2))**

**{**

**System.out.println("s2 is a rotated version of s1");**

**}**

**else**

**{**

**System.out.println("s2 is not rotated version of s1");**

**}**

**}**

**}**

**}**

## *What Is Anagram?*

Two strings are called anagrams if they contain same set of characters but in different order. For example, **“Dormitory – Dirty Room”**, **“keep – peek”,  “School Master – The Classroom”** are some anagrams.

**public class AnagramProgram**

**{**

**static void isAnagram(String s1, String s2)**

**{**

**//Removing all white spaces from s1 and s2**

**String copyOfs1 = s1.replaceAll("\\s", "");**

**String copyOfs2 = s2.replaceAll("\\s", "");**

**//Initially setting status as true**

**boolean status = true;**

**if(copyOfs1.length() != copyOfs2.length())**

**{**

**//Setting status as false if copyOfs1 and copyOfs2 doesn't have same length**

**status = false;**

**}**

**else**

**{**

**//Changing the case of characters of both copyOfs1 and copyOfs2 and converting them to char array**

**char[] s1Array = copyOfs1.toLowerCase().toCharArray();**

**char[] s2Array = copyOfs2.toLowerCase().toCharArray();**

**//Sorting both s1Array and s2Array**

**Arrays.sort(s1Array);**

**Arrays.sort(s2Array);**

**//Checking whether s1Array and s2Array are equal**

**status = Arrays.equals(s1Array, s2Array);**

**}**

**//Output**

**if(status)**

**{**

**System.*out*.println(s1+" and "+s2+" are anagrams");**

**}**

**else**

**{**

**System.*out*.println(s1+" and "+s2+" are not anagrams");**

**}**

**}**

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

**{**

***isAnagram*("Mother In Law", "Hitler Woman");**

***isAnagram*("keEp", "peeK");**

***isAnagram*("SiLeNt CAT", "LisTen AcT");**

***isAnagram*("Debit Card", "Bad Credit");**

***isAnagram*("School MASTER", "The ClassROOM");**

***isAnagram*("DORMITORY", "Dirty Room");**

***isAnagram*("ASTRONOMERS", "NO MORE STARS");**

***isAnagram*("Toss", "Shot");**

***isAnagram*("joy", "enjoy");**

**}**

**}**

***What is the difference between Runnable and Callable Interace?***

**Callable vs Runnable**

Though both the interfaces are implemented by the classes who wish to execute in a different thread of execution, but there are few differences between the two interface which are:-

* A Callable<V> instance returns a result of type V, whereas a Runnable instance doesn't
* A Callable<V> instance may throw checked exceptions, whereas a Runnable instance can't

# [*Remove duplicates from an array java*](http://www.instanceofjava.com/2014/12/remove-duplicates-from-array-java.html)

**package** com.instanceofjavaforus;

**import** java.util.Arrays;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**public** **class** RemoveDupArray

{

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

{

// A string array with duplicate values

String[] data ={ "E", "C", "B", "E", "A", "B", "E", "D", "B", "A" };

System.***out***.println("Original array : " + Arrays.*toString*(data));

List<String> list = Arrays.*asList*(data);

Set<String> set = **new** HashSet<String>(list);

System.***out***.print("After removing duplicates: ");

String[] resarray = **new** String[set.size()];

set.toArray(resarray);

**for** (String ele : resarray)

{

System.***out***.print(ele + ", ");

}

}

}

Output

----------

**OutPut:**

**Original array         : [E, C, B, E, A, B, E, D, B, A]**

**After removing duplicates: D, E, A, B, C**

***How to convert an array to ArrayList in Java***

String[] asset = {"equity", "stocks", "gold", "foreign exchange","fixed income", "futures", "options"};

List<String> assetList = Arrays.asList(asset);

# <http://javarevisited.blogspot.in/2011/09/generics-java-example-tutorial.html>

*1) This method returns a*[*List view*](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) *of underlying array.  
  
2) List returned by this method would be fixed size.  
  
3) Most important point to note is when you change an element into this List corresponding element in original array will also be changed.  
  
4) Another important point is since List is fixed size, you can not add element into it. If you try you will get exception.*

***java.lang.UnsupportedOperationException.*** *5) This is the most efficient way of converting array to arraylist as per my knowledge because it doesn't copy the content of underlying array to create list.  
  
6) From Java 5 or JDK 1.5 onwards this method supports generic so you can generate type safe ArrayList from array. to know more about Generic see my post* [*How Generic works in Java*](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html)

***Note:***

*One of the most important point related to Arrays.asList() method is that it returns a fixed size List not a read only List, although you can not add() or remove() elements on this List you can still change existing elements by using set method. If you are using this to create read only List than its wrong, Use* [*Collections.unmodifiableList method to create read only collection in Java*](http://javarevisited.blogspot.sg/2012/07/create-read-only-list-map-set-example-java.html)*.*

String str ="1,2,45,69,87,15,24,23,10";

List list = Arrays.*asList*(str);

System.***out***.println("list-->"+list);

System.***out***.println("zeroth index-->"+list.get(0));

Output: zeroth index-->1,2,45,69,87,15,24,23,10

list.set(0, "amit");

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

Output:- [amit]

***Note:-After converting arrays element into the List object then all the arrays elements are stored at the zeroth index of the List.***

List<String> assetList = new ArrayList();

String[] asset = {"equity", "stocks", "gold", "foriegn exchange", "fixed income", "futures", "options"};

Collections.addAll(assetList, asset);

*1) Its not as fast as Arrays.asList() but more flexible.  
 2) This method actually copies the content of the underlying array into ArrayList provided  
3) Since you are* [*creating copy of original array*](http://java67.blogspot.sg/2012/07/copy-elements-from-list-to-set-in-java-collection-example.html)*, you can add, modify and remove any element without affecting original one.*

### Example of converting ArrayList to Array in Java

***ArrayList assetTradingList = new ArrayList();***

***assetTradingList.add("Stocks trading");***

***assetTradingList.add("futures and option trading");***

***assetTradingList.add("electronic trading");***

***assetTradingList.add("forex trading");***

***assetTradingList.add("gold trading");***

***assetTradingList.add("fixed income bond trading");***

***String [] assetTradingArray = new String[assetTradingList.size()];***

***assetTradingList.toArray(assetTradingArray);***

# How to convert Array to List in Java

Sometime you need to **convert an array variable to List or Collection** for some iterative (looping) process. Actually Java come with a handy function to achieve this – Arrays.asList.

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** ArrayToList {

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

String sArray[] = **new** String []{"Array 1", "Array 2", "Array 3"};

*//convert array to list*

List lList = **Arrays.asList(sArray);**

System.out.println(lList);

}

}

**1.List<Integer> list1 = new ArrayList<Integer>(Arrays.asList(ia)); //copy=case1**

**2.List<Integer> list2 = Arrays.asList(ia); =case 2**

Case 2 is better performance-wise BUT: it returns a List with an immutable size. Meaning you cannot add/remove elements to/from it:

***What design patterns are used in Struts?***

***Struts is based on model 2 MVC (Model-View-Controller) architecture.***

***Struts controller uses the command design pattern***

***Action classes use the adapter design pattern.***

***The process() method of the RequestProcessor uses the template method design pattern.***

***Struts also implement the following J2EE design patterns.***

***·         Service to Worker***

***·         Dispatcher View***

***·         Composite View (Struts Tiles)***

***·         Front Controller***

***·         View Helper***

***·         Synchronizer Token***

***Struts controller uses the Command design pattern and the action classes use the Adapter design pattern. The process() method of the RequestProcessor uses the Template method design pattern. Struts also implement the following J2EE design patterns.   
  
\* Service to Worker   
\* Dispatcher View   
\* Composite View (Struts Tiles)   
\* Front Controller   
\* View Helper   
\* Synchronizer Token***

# *What Design Pattern is used in HIBERNATE?*

* **Domain Model Pattern – An object model of the domain that incorporates both behavior and data.**
* **Data Mapper – A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.**
* **Proxy Pattern – for lazy loading**
* **Factory pattern – in Session Factory**

***Design Patterns being used in Spring framework***

## Spring : Design Patterns Used in Java Spring Framework

**Dependency injection/ or IoC (inversion of control)** *– Is the main principle behind decoupling process that Spring does*

**Factory Design Pattern** – *Spring uses* ***factory pattern*** *to create objects of beans using Application Context reference.*

*// Spring uses factory pattern to create instances of the objects  
BeanFactory factory = new XmlBeanFactory(new FileSystemResource("spring.xml"));  
Triangle triangle = (Triangle) factory.getBean("triangle");  
triangle.draw();*

**Proxy** **Design pattern** – used heavily in AOP, and remoting.

**Singleton** **Desing Pattern** – *by default, beans defined in spring config file (xml) are only created once. No matter how many calls were made using getBean() method, it will always have only one bean. This is because, by default all beans in spring are singletons.  
This can be overridden by using Prototype bean scope.Then spring will create a new bean object for every request.*

**Model View Controller** – *The advantage with Spring MVC is that your controllers are POJOs as opposed to being servlets. This makes for easier testing of controllers. One thing to note is that the controller is only required to return a logical view name, and the view selection is left to a separate ViewResolver. This makes it easier to reuse controllers for different view technologies.*

**Front Controller** – *Spring provides DispatcherServlet to ensure an incoming request gets dispatched to your controllers.*

**View Helper** *– Spring has a number of custom JSP tags, and velocity macros, to assist in separating code from presentation in views*.

**Template method** – *used extensively to deal with boilerplate repeated code (such as closing connections cleanly, etc..). For example JdbcTemplate, JmsTemplate, JpaTemplate*.

**Difference between NoClassDefFoundError and ClassNotFoundException?**

**---------------------------------------------------------------------------------------------------**

**Test t1 = new Test();**

**for the our hard coded class name, at run time if the corresponding .class file is not present then JVM will**

**throw RunTime Exception saying NoClassDefFoundError.**

**This is unchecked type of exception.**

**ClassNotFoundException**

**---------------------------------**

**Object obj = Class.forName(args[0]).newInstance();**

**At RunTime Dynamically provided class name, if the corresponding dot class in not found or available then Jvm will throw run time exception which saying ClassNotFoundException.**

**This is checked type of exception.**

#### 