JAVA … in short

Pt-1 : Core Java

Aditya Handadi - July 3, 2015

JAVA , founded in 1995 by James Gosling. Can be run on many platforms like UNIX, Windows and MAC OS. It is open source and free of cost. Eclipse IDE and Netbeans are amongst most popular tools to develop JAVA program.

1. Objects, Classes ,Methods and Instance Variables

Class is a template or blueprint to that describes a particular behavior that is defined for a particular type of objects to be created[1].

Objects are instance of a class, they have states and behaviors[1] . ex: if Car is a class, Toyota Prius can be an object of that class.

Method is basically a behavior. A class can contain many methods. Methods are written for classes only. This is where the whole business logic is written.

Variables are data types, offered by JAVA . Defined in class . ex: Integers, Strings. There are two times, primitive and non primitive.

2.Compiling and Executing Java

There are 2 ways, either use tools like Eclipse to execute or just notepad to write the code and compile and execute with JDK installed in system. The later method requires environment variables set to the specific JAVAHOME path.

Consider this program:

```
public class MyFirst{
     //method defined below
     public static void main(String args[]){
          System.out .println(“Hello”);
     }

}
```

This program once written in notepad , save as MyFirst.java . Then executing the following commands, executes the Java program:

- compiling

```
javac MyFirst.java
```

- execute

```
java MyFirst
```

```
Hello
```

3. Basic Syntax and Features

JAVA is case sensitive. Class Names always start with upper case letters. Method names with lower case letters. Program File Name(executable file) should always be same as the class name containing the main method. Only one main method allowed per class, but in a single package, there can be multiple main methods.
Identifiers, should begin with a letter or $ or _ . Cannot start with a number. Key word cannot be used as identifiers. List of keywords given below:

abstract
assert
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
enum
extends
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
try
void
volatile
while

4.Object Oriented Java

Abstraction , Encapsulation, Inheritance and Polymorphism. A class in Java looks like this:

```
public class MyClass{
     String Name;
     int age;
     String address;

     void setBehaviour1(){
          //logic goes here
     }

     void behaviour2(){
          //logic here
     }
}
```

Variables -

3 types: Local variables, Instance variables, Class variables. Local variables defined inside a block like within a method or constructors etc. Instance variables defined within a class, but outside a method. Class variables are like instance variables but written with “static” keyword.

Constructors -

Every class has a constructor even if we don't define it. Java compiler builds a default constructor if not defined any for that class. Constructor is identified with its classname. A class can have multiple constructors.

Any Java object can be instantiated with the “new “ keyword. Another way of instantiating is

```
Class cls = Class.forName("Student");
           Student ob1 = (Student) cls.newInstance(); [2]
```

5.Data Types

Primitive and Reference data types/Object data types.

Primitive Data Types:

byte - 8 bit 2’s complement signed integers. -128 to 127 . Default value = 0
short - 16 bit 2’s complement signed integers - -32768 to 32767 . Default value = 0
int - 32 bit —— “ ——- - -2^{32} to $2^{32}-1$. Default value = 0L.
float - 32 bit IEEE 754 floating point - default value - 0.0f .
double - 64 bit. 0.0d .
boolean - true or false.
char - 16 bit single unicode character . \u0000 to \uffff .

Object/Class Data Types:

Same data types are available in Integer, String, Boolean etc. These are basically used in Generics, or other related operations. Sometimes performance is kept in mind while we use such variables. Other data types can be user defined. Ex, creating an object from a class.

Converting String to Integers:

int foo = Integer.parseInt(“1234”) ; //value of foo will be 1234 in integer now.

Converting String to char:
char c = str.charAt(3) ; //will give 3rd character in string str.

Final and Static variables/methods:
http://stackoverflow.com/questions/13772827/difference-between-static-and-final

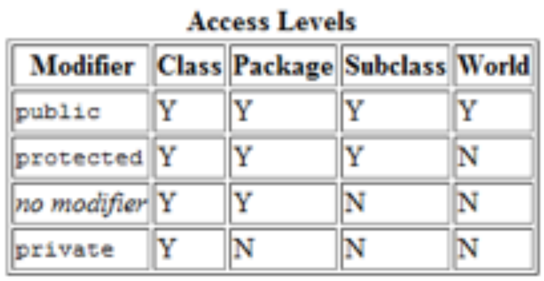
Define static methods in the following scenarios only:
1 If you are writing utility classes and they are not supposed to be changed.
2 If the method is not using any instance variable.

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.

6.Access Modifiers

Access Modifiers:

public, private, protected and default(no mod).


Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Non Access Modifiers:

static, final, abstract , synchronized and volatile (threads) .

Loops, DataTypes, If Else, Case statements just like C++.

7.Arrays

Declaring:

```
dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[];  //  works but not preferred way.
```

Initializing:

```
arrayRefVar = new dataType[arraySize];
```

or can be done in single line as :

```
dataType[] arrayRefVar = new dataType[arraySize];

OR

dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

Length & Size:

```
public class MyArrays {

      public static void main(String args[]){
            double[] myDouble = {2.3,4.5,6.5,4.45,6.23};

            System.out.println("Length myDouble : "+myDouble.length);

            List<Double> myDouble2 = new ArrayList<Double>();
            myDouble2.add(2.3);
            myDouble2.add(3.3);
            myDouble2.add(4.4);
            myDouble2.add(5.3);
            myDouble2.add(6.23);

            System.out.println("Size myDouble2 : "+myDouble2.size());

      }

}
```

Output:

```
Length myDouble : 5
Size myDouble2 : 5
```

As we see above, if we use java.lang.Double or Integer or Float etc, we use size(). Else var.length ;

for each loops:

```
// Print all the array elements
     for (double element: myList) {
        System.out.println(element);
     }
```

passing arrays into methods:

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

returning array from methods

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```

8.Date

Date() and Date(long milliseconds) can be used to create a Date object. date.toString() will give Date output. Comparing dates can be done in 3 ways:

- use boolean before(Date date) , boolean after(Date date) , and equals() to compare.
- use getTime() and compare using two milliseconds values.
- use date1.compareTo(date2) .

Using the Simple Date Formatter :

```
public class DateDemo {
   public static void main(String args[]) {

      Date dNow = new Date( );
      SimpleDateFormat ft =
      new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

      System.out.println("Current Date: " + ft.format(dNow));
   }
}
```

output:

```
Current Date: Sun 2004.07.18 at 04:14:09 PM PDT
```

Simple Date Formatter codes:

http://www.tutorialspoint.com/java/java_date_time.htm [1]

Converting String into Dates:

```
String string = "January 2, 2010";
DateFormat format = new SimpleDateFormat("MMMM d, yyyy",
Locale.ENGLISH);
Date date = format.parse(string);
System.out.println(date); // Sat Jan 02 00:00:00 GMT 2010 [2]
```

Timers:

```
long start = System.currentTimeMillis( );
long end = System.currentTimeMillis( );
long diff = end - start;
```

9. Sorting Techniques

Selection Sort : time complexity O(n^2) .

```
/* a[0] to a[n-1] is the array to sort */
           int i,j;
           int iMin;

           for (j = 0; j < n-1; j++) {
             iMin = j;

               for ( i = j+1; i < n; i++) {
                   if (a[i] < a[iMin]) {
                      iMin = i;
                   }
               }
               if(iMin != j) {
                   swap(a[j], a[iMin]);
               }
           }
```

Insertion Sort : time complexity O(n^2) .

```
            double temp;
            int j;
            for(int i=1;i< myDouble.length-1 ;i++){
                 temp = myDouble[i];
                 j = i ;
                 while(j>0 && myDouble[j-1]>temp){
                      myDouble[j] = myDouble[j-1];
```

```
                    }

  void bubble(int a[], int l, int r)
   { for (int i = l; i < r; i++)
       for (int j = r; j > i; j--)
         compexch(a[j-1], a[j]);
   }

                    myDouble[j] = temp;
               }
```

Bubble Sort : O(n^2):

Quick Sort / Partition Sort : time complexity O(n^2) - worst case . O (n log n) average case.

```
#include<iostream>
using namespace std;

void quickSort(int arr[], int left, int right)
 {
  int i = left, j = right;
  int tmp;
  int pivot = arr[abs((left + right) / 2)];
  cout<<"pivot is"<<pivot<<endl;

  /* partition */
  while (i <= j) {
        while (arr[i] < pivot)
              i++;
        while (arr[j] > pivot)
              j--;
        if (i <= j) {
              tmp = arr[i];
              arr[i] = arr[j];
              arr[j] = tmp;
              i++;
              j--;
    }
}
cout<<"recursion"<<endl;
/* recursion */
if (left < j)
    quickSort(arr, left, j);

if (i< right)
    quickSort(arr, i, right);
}
```

```
void merge(int a[], int l, int m, int r)
 {
   int i, j;
   static int aux[maxN];
   for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
   for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
   for (int k = l; k <= r; k++) {
     if (aux[j] < aux[i])
       a[k] = aux[j--];
     else
       a[k] = aux[i++];
     }
 }

void mergesort(int a[], int l, int r)
 { if (r <= l) return;
   int m = (r+l)/2;
   mergesort(a, l, m);
   mergesort(a, m+1, r);
   merge(a, l, m, r);
 }
```

http://stackoverflow.com/questions/11975214/quick-sort-code-explanation

Merge Sort : time complexity O(n log n) even in worst case.

10. Reading Values from Console

Let's say we have input :

```
 /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your
class should be named Solution. */
      Scanner sc = new Scanner(System.in);

      //int[] intArr = new int[maxN];

      int n = sc.nextInt();
      long sum = 0L;
      try{
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String[] splited = br.readLine().split("\\s+");

      for(int i=0;i<n ;i++){
            long x = Long.valueOf(splited[i]).longValue();
            sum += x;
      }

      System.out.println(sum);

      sc.close();
      }catch(Exception e){
            e.printStackTrace();
      }
```

```
5
1000000001 1000000002 1000000003 1000000004 1000000005
```

This can be read either by scanner as shown , or BufferedInputReader.
Output:

```
5000000015
```

Reading Arrays :

```
int[][] arr = new int[maxN][maxN];

     for(int i=0;i< n;i++){
          String str = br.readLine();
          String[] strsplit = str.split("\\s+");
          for(int j=0;j<n;j++){
               arr[i][j] = Integer.valueOf(strsplit[j]);
          }
     }
```

Rounding up Decimal Places:

Let’s say we have a value : 0.923456

To round up to x decimal places we can use 3 methods:

- String.format(“%.<x>g%n”, 0.923456); ex: String.format("%.5g%n", 0.912300); for 5 decimal places
- `DecimalFormat df = new DecimalFormat("#.#####"); df.format(0.912385);`
- (double)Math.round(value * 100000) / 100000 ; 10^x for x decimal places.

11.Break & Continue Tag

Without any adornment, `break` will just break out of the innermost loop. Thus

```
while (true) { // A
    while (true) { // B
```

```
        break;
    }
}
```
Will loop forever, because the `break` only exists loop `B`.

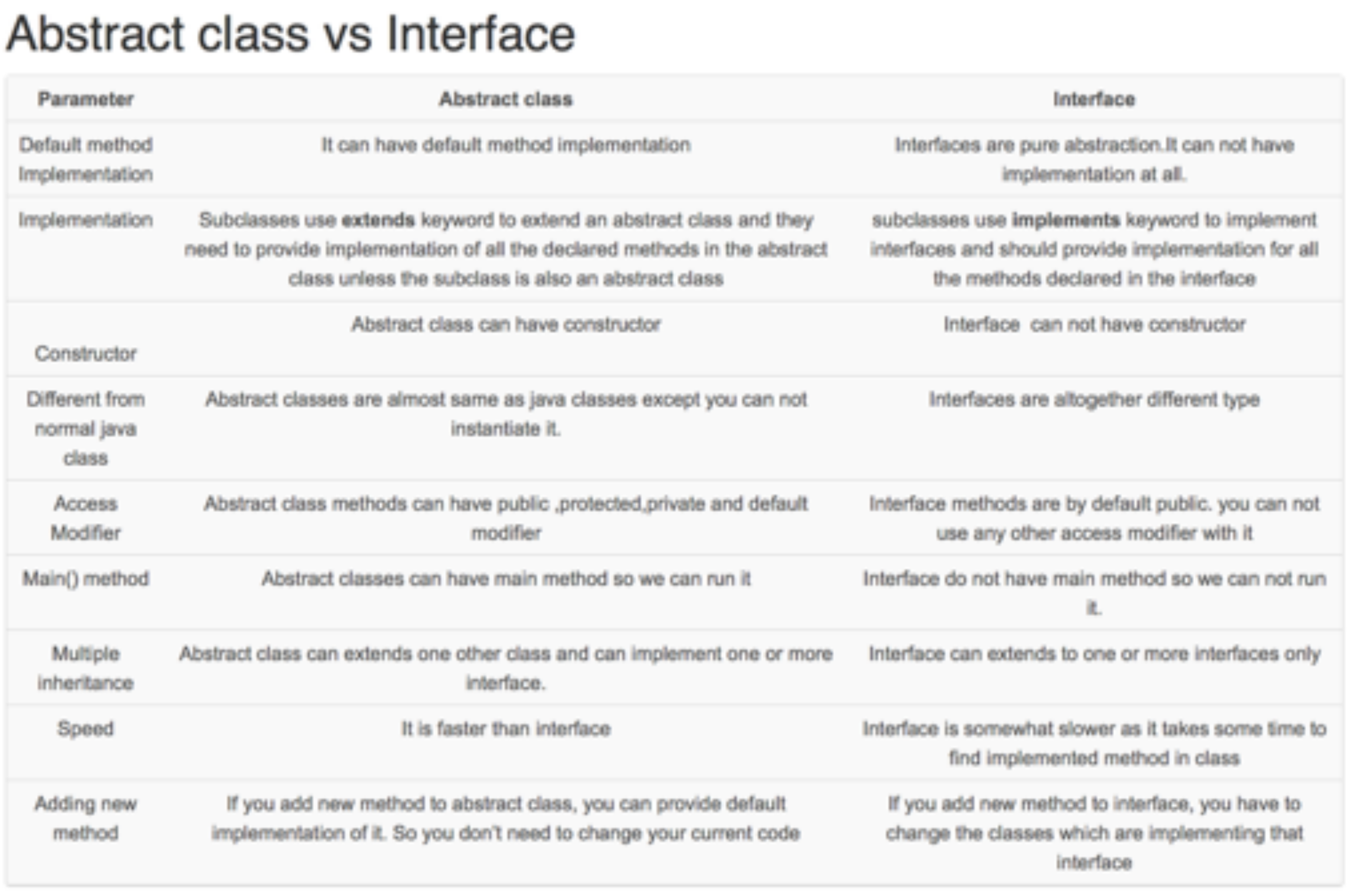
However, Java has a feature called "named breaks" in which you can name your loops and then specify which one to break out of. For example:
```
A: while (true) {
    B: while (true) {
        break A;
    }
}
```
This code will not loop forever, because the `break` explicitly leaves loop `A`.

Fortunately, this same logic works for `continue`. By default, `continue` executes the next iteration of the innermost loop containing the `continue` statement, but it can also be used to jump to outer loop iterations as well by specifying a label of a loop to continue executing.

In languages other than Java, for example, C and C++, this "labeled break" statement does not exist and it's not easy to break out of a multiply nested loop. It can be done using the `goto` statement, though this is usually frowned upon. For example, here's what a nested break might look like in C, assuming you're willing to ignore Dijkstra's advice and use `goto`:
```
while (true) {
    while (true) {
        goto done;
    }
}
done:
   // Rest of the code here.
```
Hope this helps!

12. Abstract Class and Interface

Abstract classes and Interfaces both are used to give an abstract definition of something. But Abstract classes are nothing but Java classes, which cannot be instantiated. We can have default implementation of a method with any access modifiers unlike Interfaces.



Abstract class vs Interface

Parameter	Abstract class	Interface
Default method Implementation	It can have default method implementation	Interfaces are pure abstraction.It can not have implementation at all.
Implementation	Subclasses use **extends** keyword to extend an abstract class and they need to provide implementation of all the declared methods in the abstract class unless the subclass is also an abstract class	subclasses use **implements** keyword to implement interfaces and should provide implementation for all the methods declared in the interface
Constructor	Abstract class can have constructor	Interface can not have constructor
Different from normal java class	Abstract classes are almost same as java classes except you can not instantiate it.	Interfaces are altogether different type
Access Modifier	Abstract class methods can have public ,protected,private and default modifier	Interface methods are by default public. you can not use any other access modifier with it
Main() method	Abstract classes can have main method so we can run it	Interface do not have main method so we can not run it.
Multiple inheritance	Abstract class can extends one other class and can implement one or more interface.	Interface can extends to one or more interfaces only
Speed	It is faster than interface	Interface is somewhat slower as it takes some time to find implemented method in class
Adding new method	If you add new method to abstract class, you can provide default implementation of it. So you don't need to change your current code	If you add new method to interface, you have to change the classes which are implementing that interface

Abstract classes are always extended by its child classes, whereas Interfaces are only implemented. Interfaces can extend only other Interfaces, and not classes. Abstract classes can have constructors, where as interfaces cant.

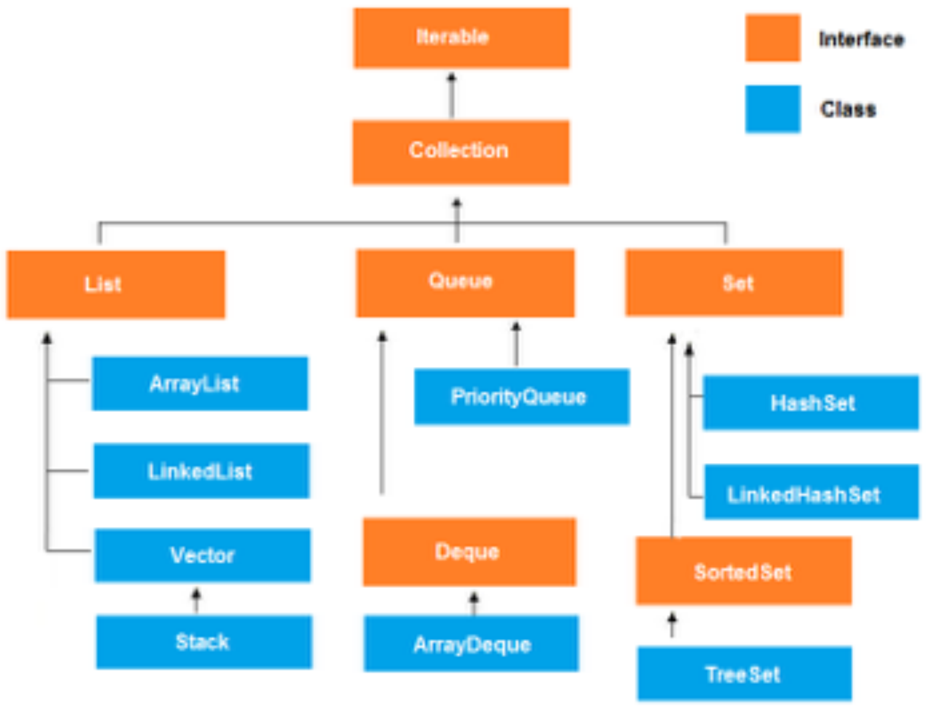
Why do we need abstract classes?
-

13. Comparator and Comparable

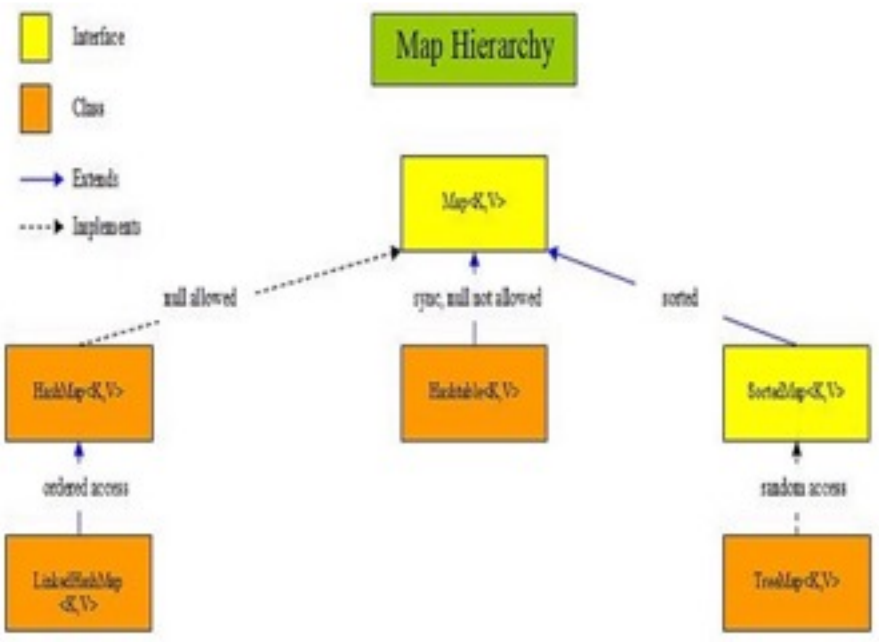
Comparator = compare(obj1,obj2);
Comparable = compareTo(obj1);

http://www.javacodegeeks.com/2013/03/difference-between-comparator-and-comparable-in-java.html

14. Collections



Maps:



WHY MAIN METHOD IS STATIC:

http://stackoverflow.com/questions/146576/why-is-the-java-main-method-static

15. Installing Maven on Mac

First install Maven as mentioned here:

http://www.tutorialspoint.com/maven/maven_environment_setup.htm

One command above is wrong: export MAVEN_OPTS="-Xms256m -Xmx512m" <quotes>

copy the tar file in usr/local/apache-maven-<version> . cp -r source destination wont work, since its a root folder. So, you need to do: sudo <above command>

after that: follow these links for problems.

http://stackoverflow.com/questions/21028872/mvn-command-not-found-in-osx-mavrerick/21030998#21030998 http://stackoverflow.com/questions/21012019/how-to-set-commands-for-mvn-in-bash-in-osx

http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/

15. Exceptions

Unchecked
ArrayIndexOutOfBoundsException
ClassCastException
IllegalArgumentException
IllegalStateException
NullPointerException
NumberFormatException
AssertionError
ExceptionInInitializerError
StackOverflowError
NoClassDefFoundError

Checked
Exception
IOException
FileNotFoundException
ParseException
ClassNotFoundException
CloneNotSupportedException
InstantiationException
InterruptedException
NoSuchMethodException
NoSuchFieldException

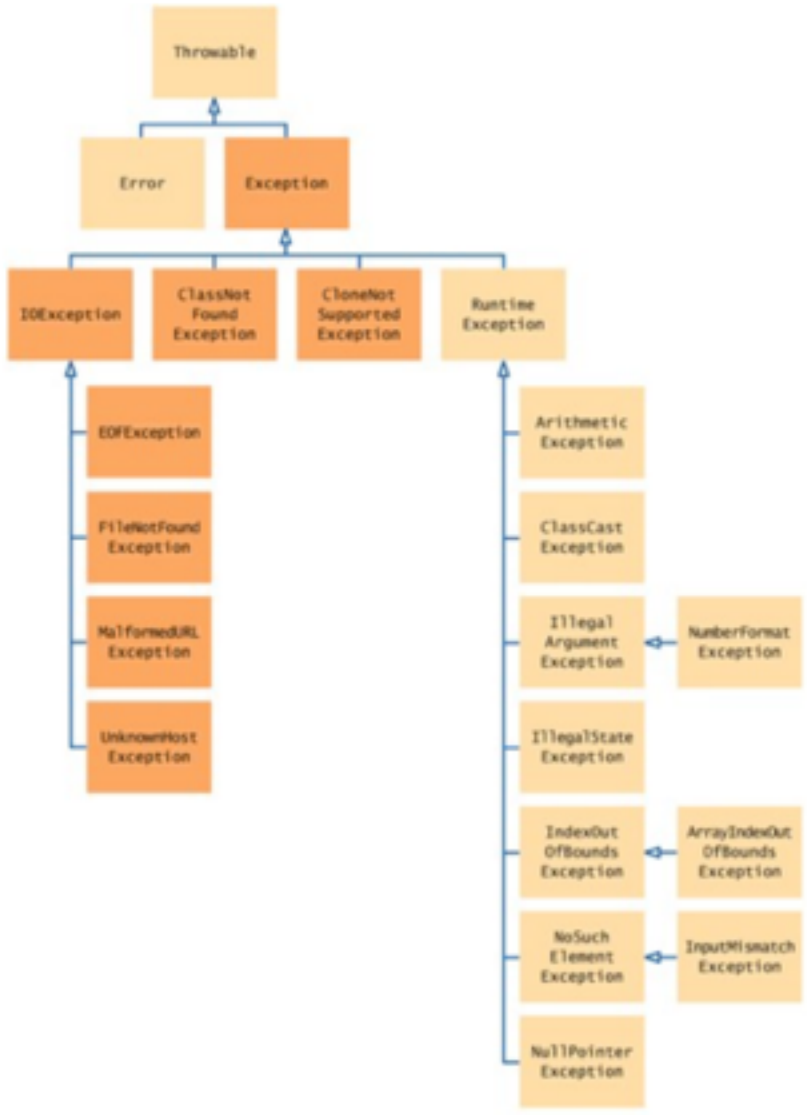


Figure 1 The Hierarchy of Exception Classes

16. == And .equals()

a == b always return true if both are pointing to the same memory location. Hence in case of primitive data types ,they always return true if the values of a and b are equal.

.equals() can actually check the value of the two strings. But behaves same as == to other objects. because Object.equals simply returns this==object; But String class actually overrides Object's .equals() method. Hence it can compare the value of two strings.

```
String str1 = “aditya"; //prim
String str2 = “aditya"; //prim

String str3 = new String(“aditya"); //nonprim
String str4 = new String(“aditya"); //nonprim
System.out.println(str1==str2); //T
System.out.println(str1.equals(str2)); //T

System.out.println(str3==str4); //F
System.out.println(str3.equals(str4)); //T
```