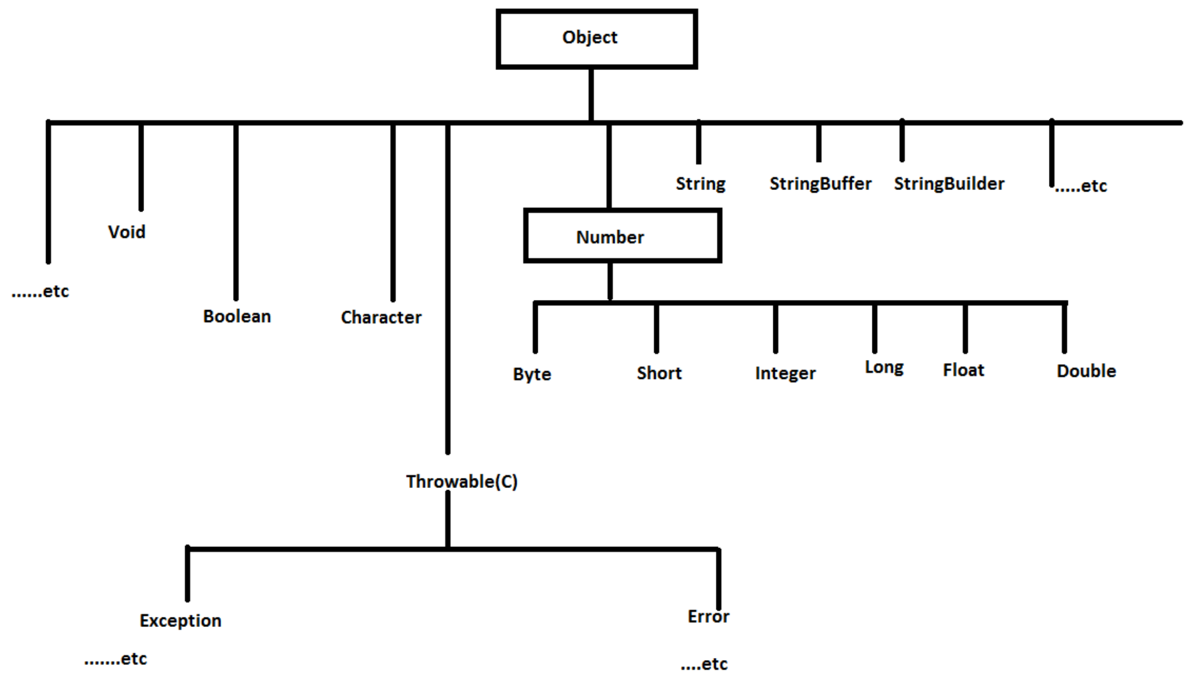
## Java Lang package

Introduction

- The Basic understanding  about the java language package is fundamental to write any simple or complex program.In simple without java lang package we can not write a small program of java.
- The most important weightage package for java is java lang package.
- There are multiple classes which need to be understood on a high level.

Java lang hierarchy



## Object(c)

- Object class is a super class for every class in java either directly or indirectly.
- Object class has 11 methods other than registerNatives() method and there is no variables.
- Since the Object class is a super class of all java classes ,then all those 11 methods are applicable to each and every class in java by default.

**Methods**

```
private static native void registerNatives();

public final native Class getClass();

public native int hashCode();
```

public boolean equals(Object obj);

protected native Object clone() throws CloneNotSupportedException;

public String toString();

public final native void notify();
public final native void notifyAll();

public final native void wait(long timeout) throws InterruptedException;
public final void wait(long timeout, int nanos) throws InterruptedException;
public final void wait() throws InterruptedException

protected void finalize() throws Throwable

## toString()

- In general the toString() method returns a String (textually represents) of any object.

**Example : 1**

```
 public class Test {
      public static void main(String[] args){
      String s = new String("Ganapati Bappa Morya");
            System.out.println(s.toString());
      }
}
```

**Output:**

javac : Test.java
java Test

```
Ganapati Bappa Morya
```

**Example : 2**

```
public class Test {
      String wish;
      Test(String wish){
            this.wish = wish;
      }
public static void main(String[] args){
      Test t = new Test("Ganapati Bappa Morya");
      System.out.println(t.toString());
      }
}
```

**Output:**

javac : Test.java
java Test
```
Test@5acf9800
```

**Focus : String class in java toString() method is implemented to represent meaningful representation of Object.Where as our Test class does not have toString() method so by default Parent class Object toString() method has been called.**

**Example : 3**

```
public class Test {
        String wish;

        Test(String wish){
                this.wish = wish;
        }

        public static void main(String[] args){
        Test t = new Test("Ganapati Bappa Morya");
        System.out.println(t.toString());
        }

        public String toString() {
                return wish;
        }
}
```

**Output:**

javac : Test.java
java Test
```
Ganapati Bappa Morya
```

**Focus : It is recommended that all subclasses override the toString() method to get the meaningful representation of objects instead of the default toString method which is present in Object class.**

**Default implementation of toString()**

```
class Object {
  public String toString() {
       return getClass().getName() + "@" + Integer.toHexString(hashCode());
    }
}
```

**Example : 4**

```
import java.util.ArrayList;
public class Student {
      int rollNo;
      String name;

      Student(int rollNo, String name) {
            this.rollNo = rollNo;
            this.name = name;
      }

      public static void main(String[] args) {
            Integer i = new Integer(100);
            System.out.println(i);

            Character ch = new Character('S');
            System.out.println(ch);

            ArrayList al = new ArrayList();
            al.add(100);
            al.add("Jhon");
            System.out.println(al);

            Student student = new Student(100,"Jhon");
            System.out.println(student);

      }
}
```

- It's Suggested to override the toString() method in our user defined classes also.
- add this method in the above Student class for meaningful representation of Student objects.

```
public String toString() {
            return rollNo+"#"+name;
      }
```

**Focus : All the wrapper classes,collection classes,String ,StringBuffer and StringBuilder classes are already overridden toString() method for meaningful object representation.**

**hashCode()**

- In java jvm will generate a unique number for every object is nothing but hasCode.
- hashCode will not relate to any address of object , infact in java we can not identify the address of object.
- hashCode will not relate to any size of object, infact in java we can not have terminology called size of object.

**Why is hashCode required for a java object ?**

Linear search alg = o(n)
Binary search alg = o(long power n base 2)
hashing search alg = o(1)

- JVM will use hashcode while saving objects into hashing related data structures.
- The main advantages of saving objects based on hashcode is that search operation will become easy.
- The most powerful searching alg is Hashing.
- If our class doesn't have hashcode then object class hashcode method will execute and Object class hashcode method is implemented in native language as based on address by applying some algorithm it yields int value. This doesn't mean hashcode represents address.
- Based on our requirement we can override hashcode and to generate our own hashcode.But while overriding hashcode we should take care of uniqueness of generation.

```
class Student {

    public int hashCode(){
        return 100;
    }
}
```

The above way generating hashcode is not looks proper way, Because for all student objects same number as hashcode

```
class Student {
    public int hashCode(){
        return rollNo;
    }
}
```

The above way of generating a hashcode looks proper, Because for all student objects a different number is a hashcode.

**Relationship between toString() and hashCode()**

- If the programmer does not override the  toString() method then the Object class toString() method calls by default and the Object class toString() method internally calls hashcode() method.
- If a programmer overrides the toString() method then may or may not be required to call hashcode() method.

## Example : 1

```
public class Test {
             int i;
      Test(int i){
        this.i = i;
      }
      public static void main(String[] args){
            Test t1 = new Test(10);
            System.out.println(t1);//Test@us32dsd
            Test t2 = new Test(100);
            System.out.println(t2);//Test@yub34gf
      }
}
```

Object class toString() ===> hashCode() of Object class itself

## Example : 2

```
public class Test {
      int i;
      Test(int i) {
            this.i = i;
      }

      public static void main(String[] args){
            Test t1 = new Test(10);
            System.out.println(t1);
            Test t2 = new Test(100);
            System.out.println(t2);
      }

      @Override
      public int hashCode() {
            return i;
      }
}
```

Object class toString() ===> hashCode() of Test class

## Example : 3

```
public class Test {
      int i;

      Test(int i) {
            this.i = i;
      }

      public static void main(String[] args) {
            Test t1 = new Test(10);
            System.out.println(t1);
            Test t2 = new Test(100);
            System.out.println(t2);
      }

```

```java
        @Override
        public int hashCode() {
              return i;
        }

        @Override
        public String toString() {
              return i+"";
        }
}
```

Test class toString() method called and hashCode() method may or may not called by Test class

**equals()**

- equals method used to compare two objects.
- if the programmer will not override the equals method then the Object class equals method will be called by default.

**Example 1:**

```java
class Test5
{
        int i;
        Test5(int i){
          this.i = i;
        }
        public static void main(String[] args)
        {
                Test5 t1=new Test5(100);
                Test5 t2=new Test5(100);
```

```
               System.out.println(t1.equals(t2));
       }
}
```

**Example 2:**

```
public class Adhaar {
       long adhaarNo;
       String address;

public Adhaar(long adhaarNo,String address) {
             this.adhaarNo = adhaarNo;
             this.address = address;
       }

       public static void main(String[] args) {

             Adhaar a1 = new Adhaar(32424242,"NGO STREET");
             Adhaar a2 = new Adhaar(34242423,"RND BANGLA");
             Adhaar a3 = new Adhaar(32424242,"NGO STREET");
             Adhaar a4 = a2;

             System.out.println(a1.equals(a2));
             System.out.println(a1.equals(a3));
             System.out.println(a3.equals(a4));
             System.out.println(a2.equals(a4));

       }
}
```

**Focus : The above program is an Object class equals method that is executed and it represents reference comparison.If two references are the same Object then equals method will return true.**

**Example 3:**

- Based on application requirements ,programmers can override Object class equals method for content comparison.

```
public class Adhaar {

       long adhaarNo;
       String address;
```

```
	public Adhaar(long adhaarNo, String address) {
		this.adhaarNo = adhaarNo;
		this.address = address;
	}

	public static void main(String[] args) {

		Adhaar a1 = new Adhaar(32424242, "NGO STREET");
		Adhaar a2 = new Adhaar(34242423, "RND BANGLA");
		Adhaar a3 = new Adhaar(32424242, "NGO STREET");
		Adhaar a4 = a2;

		System.out.println(a1.equals(a2));
		System.out.println(a1.equals(a3));
		System.out.println(a3.equals(a4));
		System.out.println(a2.equals(a4));

	}

	public boolean equals(Object obj) {
		long adhaarNo1 = this.adhaarNo;
		String address1 = this.address;

		Adhaar adhaar = (Adhaar) obj;
		long adhaarNo2 = adhaar.adhaarNo;
		String address2 = adhaar.address;

		if (address1.equals(address2) && adhaarNo1 == adhaarNo2) {
			return true;
		} else {
			return false;
		}

	}
}
```

**Example 4: Compare Heterogeneous and null objects with equals**

- Object class equals() method will return false for the comparison of Heterogenous Objects and null as shown in the below program.

```java
public class Adhaar {

      long adhaarNo;
      String address;

      public Adhaar(long adhaarNo,String address) {
            this.adhaarNo = adhaarNo;
            this.address = address;
      }

      public static void main(String[] args) {

            Adhaar a1 = new Adhaar(32424242,"NGO STREET");
            Adhaar a2 = new Adhaar(34242423,"RND BANGLA");
            Adhaar a3 = new Adhaar(32424242,"NGO STREET");
            Adhaar a4 = a2;

            System.out.println(a1.equals(null)); // comparing with null
            System.out.println(a1.equals("adadas")); //compare
heterogeneous elements like a1 is adhaar obj and adadas is string
            System.out.println(a3.equals(a4));
            System.out.println(a2.equals(a4));
      }
}
```

- If a programmer wants to make the same behavior of the Object class equals method then try to add Exception handling to return false instead of exception to avoid CCE and NPE.

```java
public class Adhaar {

      long adhaarNo;
      String address;

      public Adhaar(long adhaarNo,String address) {
            this.adhaarNo = adhaarNo;
            this.address = address;
      }

      public static void main(String[] args) {

            Adhaar a1 = new Adhaar(32424242,"NGO STREET");
            Adhaar a2 = new Adhaar(34242423,"RND BANGLA");
            Adhaar a3 = new Adhaar(32424242,"NGO STREET");
            Adhaar a4 = a2;

            System.out.println(a1.equals(null));
            System.out.println(a1.equals("adadas"));
            System.out.println(a3.equals(a4));
            System.out.println(a2.equals(a4));

      }

      public boolean equals(Object obj) {

            try{
            long adhaarNo1 = this.adhaarNo;
            String address1 = this.address;

            Adhaar adhaar = (Adhaar) obj;
            long adhaarNo2 = adhaar.adhaarNo;
            String address2 = adhaar.address;

            if (address1.equals(address2) && adhaarNo1 == adhaarNo2) {
                  return true;
            } else {
                  return false;
            }

            }catch(NullPointerException  | ClassCastException exception){
                  return false;
            }

      }
}
```

**Simplified version of equals method**

```
public boolean equals(Object obj) {
      try{
            Adhaar adhaar = (Adhaar) obj;
            if (address.equals(adhaar.address) && adhaarNo ==
adhaar.adhaarNo) {
                  return true;
            } else {
                  return false;
            }
     }catch(ClassCastException | NullPointerException exception){
        return false;
    }
}
```

**More simplified version of equals method**

```
public boolean equals(Object obj) {
  if(obj instanceof Adhaar){
       Adhaar adhaar = (Adhaar) obj;
      if (address.equals(adhaar.address) && adhaarNo == adhaar.adhaarNo) {
         return true;
        } else {
            return false;
        }
  }else{
    return false;
  }
}
```

**Most simplified version of equals method**

```
public boolean equals(Object obj) {
  if(obj == this)
    return true;

  if(obj instanceof Adhaar){
       Adhaar adhaar = (Adhaar) obj;
      if (address.equals(adhaar.address) && adhaarNo == adhaar.adhaarNo) {
        return true;
      } else {
            return false;
      }
  }else{
    return false;
  }
}
```