198,199:

Object class is the parent of all classes.

For every Object created a unique hash code will be assigned by compiler.

Every class directly or in-directly in-herited from Object class.

When we print Object, it will make call to toString() method.

```
Object o1=new Object();
System.out.println(o1); ///o1.toString()

import java.lang.*;
public class LangDemo {

public static void main(String[] args) {

Object o1=new Object();
System.out.println(o1);
Object o2=new Object();
System.out.println(o1.equals(o2));//equals method checks whether both o1
                                  //and o2 are referring to the same Object/not

Object o3=o1;
System.out.println(o1.equals(o3));//true -- As both the o1 and o3 are
                                  //referring to same Object

System.out.println(o1.hashCode());
```

```
}

}
```

Ex 2:

```
class MyObject
{
public String toString()
{
return "MyObject ";

}
@Override
public int hashCode()
{
return 100;
}
public boolean equals(Object o)
{

return this.hashCode()==o.hashCode();
}
}
public class LangPackageDemo {
```

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

MyObject o1=new MyObject();
MyObject o2=new MyObject();
System.out.println(o1.equals(o2));
}

}
```

We can't Override

wait()

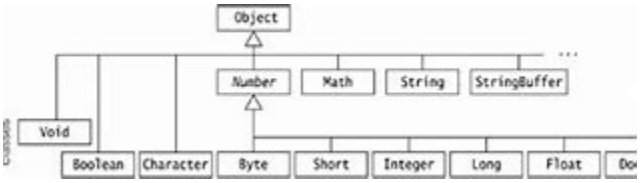
notify() as there are final.

OBJECT CLASS

- Object class is the parent class for all the classes in java.
- It can also be said as mother of all classes in the java.
- Even the user-defined classes are inherited from the object class.
- There are important methods in java:
  - **clone( ):** creates a clone of itself and give the same object.
  - **equals(object obj):** it will compare two objects, and will return true if both the references are holding the same objects.
  - **finalize( ):** this method is called by a garbage collector when ever a object of a class is being taken away by the garbage collector.

    It is same as destructor method in c++.
  - **Class( ):** returns the runtime class of the object.
  - **hashCode( ):** returns the hash code value of the object, there are no two programs created by java program which have same hashcodes.
  - **notify( ):** wakes up a single thread.
  - **notifyAll( ):** wakesup all the threads.
  - **toString( ):** returns the string representation of the object, when we print any object it calls this method by itself, in any class if you want the object to be printed by s.o.p then over ride toString( ) method.
  - **wait( ):** causes the current thread to wait until the thread invokes the notify.
  - **wait(long timeout):** causes the current thread to wait until either thread invokes the notify.
  - **wait(long timeout, int nanos):** causes the current thread to wait until the thread invokes the notify certain amount of real time has elapsed.

Wrapper classes :



Byte,Short,Integer,Long,Float,Double are child classes of Number class

WRAPPER CLASSES

- Java provides wrapper classes around primitives so they can be used as classes and their objects can be created.
- Wrapper classes are available for every data type.
- Wrapping is also known as boxing.
- All these classes are present inside the java.lang package.
- Number, Character and Boolean classes are child classes of object class.
- **Number class** have methods like:
    - byteValue( ): returns the value as byte.
    - doubleValue( ): returns the value as double.
    - floatValue( ): returns the value as float.
    - intValue( ): returns the value as int.
    - longValue( ): returns the value as long.
    - shortValue( ): returns the value as short.
- **Integer class:**
    - It is the child class of number.
    - The Integer class wraps a value of the primitive type int in an object. An object of type Integer contains a single field whose type is int.
    - It inherits the methods from the number class as it is the parent class.
- **Byte Class:**
    - The Byte class wraps a value of primitive type byte in an object. An object of type Byte contains a single field whose type is byte.

➢ **Long Class:**

- The Long class wraps a value of the primitive type long in an object. An object of type Long contains a single field whose type is long.

➢ **Short Class:**

- The Short class wraps a value of primitive type short in an object. An object of type Short contains a single field whose type is short.

➢ **Float Class:**

- The Float class wraps a value of primitive type float in an object. An object of type Float contains a single field whose type is float.

➢ **Double Class:**

- The Float class wraps a value of primitive type float in an object. An object of type Float contains a single field whose type is float.

➢ **Character Class:**

- The Character class wraps a value of the primitive type char in an object. An object of class Character contains a single field whose type is char.

➢ **Boolean Class:**

- The Boolean class wraps a value of the primitive type boolean in an object. An object of type Boolean contains a single field whose type is boolean.

single field whose type is boolean.

➢ **AutoBoxing and AutoUnBoxing:**

- *Autoboxing* is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called *unboxing*.

```
27
28          Float f1=Float.valueOf("12.3");
29          float f2=f1.floatValue();
30          float f3=f1;
31
32
```

Un-Boxing
Converting Object type data to Primitive type data.

```
float f3=f1; //Here internally it will call
               floatValue()method.
```

```
28          Float f1=Float.valueOf("12.3"); //Boxing
29          float f2=f1.floatValue(); //un-boxing
30          float f3=f1; //un-boxing
31
```

```
32          int val=10;
33          Integer i1=val; //Auto-Boxing
34          int i2=i1;    //Auto-unBoxing
35
```

Boxing :

Converting Primitive type of data to Object.

Unboxing:

Converting Object date to Primitive type .

Auto boxing:

Automatic Conversion of Primitive type of data to Object.

Auto un-Boxing:

Automatic Conversion of Object type of data to Primitive.

Example :

```
package com.chinmay.wrapper;


public class WrapperEx1 {


public static void main(String[] args) {


Integer i=new Integer(10);

Integer a=Integer.valueOf(10);

Integer b=10;


Byte c=15;

Byte d=Byte.valueOf("15");

//Byte e=Byte.valueOf(15); //we cannot directly pass number as by default any number will be

                //Considered as int.

//correct Approach

byte bb=15;

Byte e=Byte.valueOf(bb);

Short s=Short.valueOf("123");

Float f=12.3f;

Double d1=Double.valueOf("12.3");

Double d2=Double.valueOf(12.3);

Character ch=Character.valueOf('A');

Boolean l=Boolean.valueOf(false);
```

Float f1=Float.valueOf("12.3"); //Boxing

float f2=f1.floatValue(); //un-boxing

float f3=f1; //un-boxing

int val=10;

Integer i1=val; //Auto-Boxing

int i2=i1; //Auto-unBoxing

}

}

202: String vs String Buffer vs String Builder

Refer the ineuron content.

203: Math.class

Math.class will in turn call StrictMath.

```
public class MatchClass
{
public static void main(String[] args)
{
System.out.print("Absolute :");
System.out.println(Math.abs(-123));
System.out.print("Absolute :");
System.out.println(StrictMath.abs(-123));
System.out.print("Cube Root :");
System.out.println(Math.cbrt(27));
System.out.print("Exact Decrement :");
// System.out.println(Math.decrementExact(Integer.MIN_VALUE));
int i=Integer.MIN_VALUE;
i--;
System.out.println(i);
System.out.print("Exponent value in Floating Point Rep. :");
System.out.println(Math.getExponent(123.45));
System.out.print("Round Division :");
System.out.println(Math.floorDiv(50, 9));
System.out.print("e power x :");
```

```
System.out.println(Math.exp(1));
System.out.print("e power x :");
System.out.println(StrictMath.exp(1));

System.out.print("Log base 10 :");
System.out.println(Math.log10(100));
System.out.print("Maximum :");
System.out.println(Math.max(100, 50));
System.out.print("Tan :");
System.out.println(Math.tan(45*Math.PI/180));
System.out.print("Convert to Radians :");
System.out.println(Math.toRadians(90));
System.out.print("Convert to Degree :");
System.out.println(Math.toDegrees(Math.atan(1)));

System.out.print("Convert To Degree :");
System.out.println(StrictMath.toDegrees(StrictMath.tanh(1)));

System.out.print("Random :");
System.out.println(Math.random()*1000);
System.out.print("Power :");
System.out.println(Math.pow(2, 3));
System.out.print("Excact Product :");
System.out.println(Math.multiplyExact(100, 200));
System.out.print("Next Float Value :");
System.out.println(Math.nextAfter(12.5, 11));
 System.out.println("Absolute : "+Math.abs(-123));
 System.out.println("Absolute : "+StrictMath.abs(-12376));
 System.out.print("Cube Root : ");
 System.out.println(Math.cbrt(27));
 System.out.println(Math.decrementExact(-7));
//System.out.println(Math.decrementExact(Integer.MIN_VALUE));//Will result in
Exception
//as decrement of this will reult in underflow
System.out.print("Exponenet value in Floating Point Representation : ");
System.out.println(Math.getExponent(123.45));
System.out.print("Round Division : ");
System.out.println(Math.floorDiv(50,9));

}
}
```

.

204 : Enum:

Enum is used to define User Defined datatype.

Each identifier in enum is public static final.

```
enum Dept
{
    CS,IT,CIVIL,ECE;
}
```

Every enum which we will writing will be inheriting from Enum class.

Enum must be of Capital letter.

Enums will be mostly used in Switch cases.

```
enum Dept
{
CS,IT,CIVIL,ECE;
}
public class EnumExample {

public static void main(String[] args) {
Dept dt=Dept.IT;
System.out.println(dt.ordinal());
System.out.println(dt.valueOf("CS"));
Dept []dep=Dept.values();
 for(Dept d:dep)
 {
      System.out.println(d);
 }
 switch(dt)
 {
 case CS:System.out.println("Head:Chinmay \nBlock 1");
 break;
 case IT:System.out.println("Head:Sai \nBlock 2");
break;
 case CIVIL:System.out.println("Head:George \nBlock 3");
break;
 case ECE:System.out.println("Head:ECE Head \nBlock 4");
break;
 }
}
}
```

Enum can have only private and default constructors

```
enum Dept
{
CS,IT,CIVIL,ECE;
private Dept()
{

}
public class EnumExample2 {

public static void main(String[] args) {
Dept dt=Dept.IT;
}
}
}
```

Enum's Constructor will be called for the identifiers during the enum loading:

```
enum NewDept
{
CS,IT,CIVIL,ECE;
private NewDept()
{
System.out.println(this.name());
}
}
public class EnumExample2 {

public static void main(String[] args) {
NewDept dt=NewDept.IT;
}
}
```

```
CS
IT
CIVIL
ECE
```

Example2

```
enum NewDept
```

```
{
CS,IT,CIVIL,ECE;
private NewDept()
{
System.out.println(this.name());
}
public void display()
{
System.out.println(this.name()+"->"+this.ordinal());
}
}
public class EnumExample2 {

public static void main(String[] args) {
NewDept dt=NewDept.IT;
dt.display();
}
}
```

Enum can have values

Ex:

```
enum NewDept
{
    CS("Chinmay","Block A"),IT("Sai","Block B"),CIVIL("George","Block C"),ECE("Chinmay Sai","Block D");

    private String head;
    private String location;
    NewDept(String head, String location) {

        this.head=head;
        this.location=location;
    }
    public void display()
    {
        System.out.println(this.name()+"->"+this.ordinal());
    }
}
```

For each identifier we have 2 properties .

If the identifier has any properties then the constructor should be parametrized.

Each identifier with properties can be considered as Object.

```
CS("Chinmay","Block A") --- This can be Considered as one Object with 2
                        properties.
```

```
enum NewDept
{
CS("Chinmay","Block A"),IT("Sai","Block B"),CIVIL("George","Block C"),ECE("Chinmay
Sai","Block D");

private String head;
private String location;
NewDept(String head, String location) {

this.head=head;
this.location=location;
}
public void display()
{
System.out.println(this.name()+"->"+this.ordinal());
}
public String getHead() {
return head;
}
public String getLocation() {
return location;
}
}
public class EnumExample2 {

public static void main(String[] args) {
NewDept dt=NewDept.IT;
System.out.println(dt.getHead()+" "+dt.getLocation());
}
}
```

205:Introduction to Reflection Package

```
Class c=My.class;//We can Use Class for getting the definition of class.
//For Every class .class file will be generated we are using that and getting
//the Description of the class
```

Java has java.lang.reflect.*; using which we can get the Fields,Methods and parameters .

We can also get class definition in the below way.

```
My m1=new My();
```

```
Class c1=m1.getClass();
```

```
package com.example.reflection;

import java.lang.reflect.*;

class My
{
private int a;
protected int b;
public int c;
int d;
public My() {}
public My(int x,int y) {}
public void display(String s1,String s2) {}
public int show(int x,int y) {return 0;}
}

public class ReflectDemo
{
public static void main(String[] args)
{
Class c=My.class;//We can Use Class for getting the definition of class.
//For Every class .class file will be generated we are using that and getting
//the Description of the class
My m1=new My();
Class c1=m1.getClass();
System.out.println(c.getName()); //Returns Class Name
System.out.println("\nFields");
Field field[]=c.getDeclaredFields();
for(Field f:field)
{
System.out.println(f);
}
System.out.println("\nConstructor");
Constructor con[]=c.getConstructors();
for(Constructor co:con)
{
System.out.println(co);
}
System.out.println("\nMethods");
Method meth[]=c.getMethods();
for(Method m:meth)
System.out.println(m);
System.out.println("\nParameters");
Parameter param[]=meth[0].getParameters();
```

```
for(Parameter p:param)
{
System.out.println(p);
}
}
}
```