Day 3

Constructor

```
- It is method of a class which is used to initialzie instance.
- Due to following reasons it is considered as special method of a class:
    1. Its name is always same as class name.
    2. It doesn't have any return type
    3. It is designed to call implictly.
    4. It is designed to call once per instance.
- We can not call constructor on instance explicitly. It is designed to
call implictly.
```

```
    Complex c1 = new Complex();    //OK
    c1.Complex();   //Not OK
```

Types of constructor

```
1. Parameterless constructor / Default constructor( User defined )
2. Parameterized contructor
3. Default constructor( Compiler generated )
```

Parameterless constructor

```
- Constructor which do not take any parameter is called parameterless
constructor.
- If we create instance, without passing argument then parameterless
constructor gets called.
```

> It is also called as zero argument constructor / user defined default constructor.

Parameterized constructor

```
- We can write constructor by declaring parameter(s). It is called
parameterized constructor.
- If we create instance, by passing argument then paramerized constructor
gets called.
```

> Process of writing mulitple constructor inside class is called constructor overloading(actually method overloading)

Default constructor

```
- If we do not define constructor inside class, then compiler generate one
constructor for the class by default. It is called default construtor.
- Compiler generates only default parameterless constructor. In other
word, if we want to create instance by passing argument then we must
define parameterized constructor inside class.
```

> Java do not support constructor's member initializer list and default argument.

Constructor Chaining.

```
- To reduce developer efforts, we can call constructor from another
constructor. It is called "Constructor Chaining".
- For constructor chaining we must use "this() statement".
- this statement must be first statement inside constructor body.
```

```
class Complex
{
    private int real;
    private int imag;
    public Complex(  )
    {
        this( 50,60 );  //Constructor Chaining
    }
    public Complex( int real, int imag )
    {
        this.real = real;
        this.imag = imag;
    }
}
```

Static

```
- Non static local variable get space once per method call.
```

```
class Program
{
    public void print( )
```

```
    {
        int count = 0;
        ++ count;
        System.out.println("Count   :   "+count);
    }
    public static void main(String[] args)
    {
        Program p = new Program();
        p.print();  //1
        p.print();  //1
        p.print();  //1
    }
}
```

```
-  In java, we can not declare local variable static.
```

> Why we can not declare local variable static?

Static Field

```
- If we want to share value of any field in all the instances of same
class then we should declare field static.
- Static field is also called as "Class level variable". Class   variable
get space once per class.
- To access class variable we should use classname and dot operator.
- Static field get space during class loading on method area.
- If we want to initialize static field then we should use static
initializer block.
- Inside static initializer block, we can initialize only static fields.
```

```
class Test
{
    private static int num3;
    static  //static initializer block
    {
        //num3 = 500; //OK
        Test.num3 = 500; //OK
    }
}
```

```
- We can write multiple static initializer block inside class. JVM execute
it sequentially.
- Since static field do not get space inside instance, we should not
initialize it inside constructor.
```

Static Method

```
- In class, we can declare method static as well as non static.
- To access non  static members[ field + method ] of the class,  we should
define non static method inside class and to access static members of the
class, we should define static method inside class.
- Non static method is also called as instance method and it is designed
to call on instance.
- Static method is called class level method and class level methods are
designed to call on classname.
- Static method do not get this reference.
```

> Why static method do not get this reference. - Since static method do not get this reference, we can not access non static members inside static method(using this). - Using instance, we can access non static members inside static method.

```
class Program
{
    private int num1 = 10;
    private static int num2 = 20;
    public static void main(String[] args)
    {
        //System.out.println("Num1  :   "+num1);  //Not OK
        Program p = new Program();
        System.out.println("Num1  :   "+p.num1);
        System.out.println("Num2  :   "+num2);  //OK : 20
    }
}
```

```
- If we dont want to use this reference inside method then method should
be static. In other word static methid may or may not access static
members.
```

> Write a program to count number of instances created from class.

```
class InstanceCounter
{
    private static int count;
    public InstanceCounter( )
    {
        ++ InstanceCounter.count;
    }
```

```
    public static int getCount( )
    {
        return InstanceCounter.count;
    }
}
class Program
{
    public static void main(String[] args)
    {
        InstanceCounter c1 = new InstanceCounter();
        InstanceCounter c2 = new InstanceCounter();
        InstanceCounter c3 = new InstanceCounter();

        System.out.println("Instance Count  :
"+InstanceCounter.getCount());
    }
}
```

Singleton class

```
- A class from which we can create only one instance is called singleton
class.
- It is a creational design pattern.
```

> Write a program for singleton class.

```
class Singleton
{
    private Singleton( )
    {   }
    private static Singleton instance;
    public static Singleton getInstance( )
    {
        if( Singleton.instance == null )
            Singleton.instance = new Singleton();
        return Singleton.instance;
    }
}
class Program
{
    public static void main(String[] args)
    {
        Singleton s1 = Singleton.getInstance();
    }
}
```

```
- We can use any access modifier on constructor.
- If constructor is public then we can create instance anywhere in the
program.
- If constructor is private then we can create instance inside method of
same class.
- Static method of the class, which hides instantiation from end user is
called factory method.
```

Final

```
- If we dont want to modify value of the variable then we should declare
that variable final.
- In java, we can not declare local variable static but we can declare it
final.
```

```
    public static void main(String[] args)
    {
        final int x = 10;   //Ok
        System.out.println("X   :   "+x);   //OK
    }
```

```
    public static void main(String[] args)
    {
        final int x;   //Ok
        x = 10; //Ok
        System.out.println("X   :   "+x);   //OK
    }
```

```
- Once value is stored, if we dont want to modify value/state then we
should declare variable final.
```

```
    public static void main(String[] args)
    {
        final int x = 10;   //Ok
        x = 20; //Not Ok
        System.out.println("X   :   "+x);   //OK
    }
```

```
- If we dont want to modify state of field inside any method of the class
then we should declare field final.
- Note : If we want declare any field final then we should declare it
static also.
- We can not declare instance final but we can declate reference final.
```

```
class Complex
{
    private int real;
    private int imag;
    public Complex( int real, int imag )
    {
        this.real = real;
        this.imag = imag;
    }
    public void setReal(int real)
    {
        this.real = real;
    }
    public void setImag(int imag)
    {
        this.imag = imag;
    }
    public void printRecord( )
    {
        System.out.println("Real Number :   "+this.real);
        System.out.println("Imag Number :   "+this.imag);
    }
}
class Program
{
    public static void main(String[] args)
    {
        final Complex c1 = new Complex( 10,20 );
        c1.setReal(11);
        c1.setImag(22);
        //c1 = new Complex( 50,60 );  //Not OK
        c1.printRecord();   //OK : 11,22
    }
}
```

Object class

```
- It is a non final and concrete class declared in java.lang package.
- It is ultimate base class / super cosmic base class / root of java class
hierarchy. In other word, all the class( not interface ) are directly or
indirectly extended from java.lang.Object class.
```

```
- Members of object class
    1. It does'nt contain nested type
    2. It does'nt contain any field
    3. It contains only parameterless constructor
    4. It contains 11 methods( 5 non final + 6 final methods ).
```

Methods of java.lang.Object class

```
- Following are the non final methods of Object class
1. public String toString()
2. public boolean equals(Object obj)
3. public native int hashCode()
4. protected native Object clone()
            throws CloneNotSupportedException
5. protected void finalize()throws Throwable
- Following are the final methods of Object class
6. public final native Class<?> getClass()
7. public final void wait()
            throws InterruptedException
8. public final native void wait(long timeout)
            throws InterruptedException
9. public final void wait(long timeout,int nanos)
        throws InterruptedException
10. notify public final native void notify()
11. public final void native notifyAll()
```

toString() method

```
- It is non final method of java,lang.Object class
- Synatx:
    public String toString( );
- If we want to return/represent state of instance of reference type in
String format then we should use toString() method,
- If we do not define "toString()" method inside class then it's super
class's toString() method will call.
- toString() method of java.lang.Object class returns String in following
format:
    F.Q.ClassName@HexadecimalHashCode
- If implementation of super class method is logically incomplete then we
should override method in sub class.
- Result in toString method should be concise/short but     informative.
```

```
    public String toString()
    {
```

```
        return String.format("%-15s%-5d%-10.2f", this.name, this.empid,
this.salary);
    }
```