Day 5

- Non static field declared inside class is called instance variable.
- Instance variable get space once per instance.
- Intializing instance variable is a job of constructor.
- Constructor gets called once per instance.
- Non static method is also called as instance method.
- Instance method is designed to call on instance. Hence it gets this reference.
- Since instance method contains body, it is also called as concrete method.

Static Field

- If we want to share, value of any field inside all the instances of same class then we should declare such field static.
- static is modifier in java.
- Static field do not get space inside instance. Rather all the instances of same class share single copy of it. Hence size of instance depends on size of non static fields declared inside class.
- Static field declared inside class is called class level variable.
- To access instance variable, we should use object reference(t1/t2/t3/this) and to access class level variable we should use class name and dot operator.
- Instance variable get space once per instance during instantiation. But class level variable get space once per class during class loading on method area.

```
class A{
   private int a;
   private int b;
   private static int z;
}
A a1 = new A( );
A a2 = new A( );
A a3 = new A( );


class B{
    private int p;
    private int q;
    private static int z;
}
B b1 = new B( );
B b2 = new B( );
B b3 = new B( );

class C{
    private int x;
    private int y;
    private static int z;
}
C c1 = new C( );
```

```
C c2 = new C( );
C c3 = new C( );
```

- To intialize instance variable, we should use constructor but to initialize class level variable, we should use static initialization block.

```
//Static Initialization Block
static {
    //num3 = 500;   //OK
    Test.num3 = 500;    //OK
}
```

Static Method

- If we call non static method on instance then methpod get this reference. Using this reference, we can use non static members inside method.
- To access non static members 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 methods are designed to call on instance and static methods are designed to call on classname.

Why static method do not get this reference?

```
1. If we call non static method on instance then methpod get this
reference.
2. static methods are designed to call on class name.
3. Since static method is not designed to call on instance, it doesn't get
this reference.
```

- this reference is a link between non static fields and method. Since static method do not get this reference, we can not access non static members inside static method directly.
- Inside instance method, we can use static as well as non static members(field/method).
- Inside static method, we can access only static members of the class directly.
- Using instance, we can access non static members inside static method.

```
public class Program {
    public int num1 = 10;
    public 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);  //OK : 10
        System.out.println("Num2    :   "+num2);    //OK : 20
    }
}
```

- Inside method, if there is no need to use this reference then we should declare method static.
- Consider following code:

```
class Calculator{
    public static double power( double base, int index ) {
        double result = 1;
        for( int count = 1; count <= index; ++ count )
            result = result * base;
        return result;
    }
}

public class Program {
    public static void main(String[] args) {
        double result = Calculator.power( 10.5d, 2);
        System.out.println("Result  :   "+result);
    }
}
```

- Instance Counter:

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

public class Program {
    public static void main(String[] args) {
        InstanceCounter c1 = new InstanceCounter();
        InstanceCounter c2 = new InstanceCounter();
        InstanceCounter c3 = new InstanceCounter();
        System.out.println("Instance Counter    :
"+InstanceCounter.getCount());
    }
}
```

Method Local Static Variable

- Non static members(non static field + non static method) declared inside class is called instance members.
- Static members(static field + static method) declared inside class is called class level members.
- Scope of instance members and class level members is class scope.

```
public class Program {
    public static void print( ) {
        static int number = 0;  //NOT ALLOWED
        number = number + 1;
        System.out.println("Number  :   "+number);
    }
    public static void main(String[] args) {
        Program.print();    //1
        Program.print();    //2
        Program.print();    //3
    }
}
```

- In Java, static variable is also called as class level variable. class level variable should be in class scope hence we can not declare local variable static.
- But we can declare field static.

```
public class Program {
    private static int number;  //OK
    public static void print( ) {
        number = number + 1;
        System.out.println("Number  :   "+number);
    }
    public static void main(String[] args) {
        Program.print();    //1
        Program.print();    //2
        Program.print();    //3
    }
}
```

Singleton class

- A class from which, we can create single instance is called singleton class.
- A method which hides instantiation from end user is called factory method.

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

```
        Singleton s1 =  Singleton.getInstance();
        Singleton s2 =  Singleton.getInstance();
        Singleton s3 =  Singleton.getInstance();
    }
}
```