Modifies In Java

Modifiers are keywords that are added to change meaning of a definition. In Java, modifiers are catagorized into two types,

1. Access control modifier
2. Non Access Modifier

1) Access control modifier

Java language has four access modifier to control access levels for classes, variable methods and constructor.

- **Default :** Default has scope only inside the same package
- **Public :** Public scope is visible everywhere
- **Protected :** Protected has scope within the package and all sub classes
- **Private :** Private has scope only within the classes

```
public class Encap_Ex1 {
        public int a = 10;
        protected int b = 20;
        private int c = 30;
        void privateDisplay() {
                c = c+100;
                System.out.println("Private Variable : "+c);
        }
}

class SubClass extends Encap_Ex1 {
        void subDisplay() {
                a = a+100;
                b = b+100;
                System.out.println("Protected Variable : "+b);
                System.out.println("Public Variable : "+a);
        }
}

class MainClass {
        public static void main(String args[]) {
                SubClass subObj = new SubClass();
                subObj.privateDisplay();
                subObj.subDisplay();
        }
}
```

Sample Output

```
Private Variable : 130
Protected Variable : 120
Public Variable : 110
```

Public Specifies

```
class Encap_Ex2 {
        public int a;
        public float b;
        public String str;
        Encap_Ex2() {
                a = 10;
                b = (float)12.34;
                str = "Public Values !!!";
        }
        public void display() {
                System.out.println("The Public Integer Value is : "+a);
                System.out.println("The Public Float Value is : "+b);
                System.out.println("The Public String Value is : "+str);
        }
}
 class MainClass {
        public static void main(String args[]) {
                Encap_Ex2 obj = new Encap_Ex2();
                obj.display();
        }
}
```

```
The Public Integer Value is : 10
The Public Float Value is : 12.34
The Public String Value is : Public Values !!!
```

Protected Specifies

```
class Encap_Ex3 {
	protected int a;
	protected float b;
	protected String str;
	Encap_Ex3() {
		a = 111;
		b = (float)123.456;
		str = "Protected Values !!!";
	}
}
 class SubClass extends Encap_Ex3 {
	public void display() {
		System.out.println("The Protected Integer Value is : "+a);
		System.out.println("The Protected Float Value is : "+b);
		System.out.println("The Protected String Value is : "+str);
	}
}
 class MainClass {
	public static void main(String args[]) {
		SubClass obj = new SubClass();
		obj.display();
	}
}
```

```
The Protected Integer Value is : 111
The Protected Float Value is : 123.456
The Protected String Value is : Protected Values !!!
```

Private Specifies

```
class Encap_Ex4 {
	private int a;
	private float b;
	private String str;
	Encap_Ex4() {
		a = 123;
		b = (float)52.38;
		str = "Private Values !!!";
	}
	private void get() {
		System.out.println("The Private Integer Value is : "+a);
		System.out.println("The Private Float Value is : "+b);
		System.out.println("The Private String Value is : "+str);
	}
	public void display() {
		get();
	}
}
 class MainClass {
	public static void main(String args[]) {
		Encap_Ex4 obj = new Encap_Ex4();
		obj.display();
	}
}
```

```
The Private Integer Value is : 123
The Private Float Value is : 52.38
The Private String Value is : Private Values !!!
```

2) Non-access Modifier

Non-access modifiers do not change the accessibility of variables and methods, but they do provide them special properties. Non-access modifiers are of 5 types,

1. Final
2. Static
3. Transient
4. Synchronized
5. Volatile

Final

Final modifier is used to declare a field as final i.e. it prevents its content from being modified. Final field must be initialized when it is declared.

```
class Cloth
{
 final int MAX_PRICE = 999;    //final variable
 final int MIN_PRICE = 699;
 final void display()      //final method
 {
  System.out.println("Maxprice is" + MAX_PRICE );
  System.out.println("Minprice is" + MIN_PRICE);
 }
}
```

A class can also be declared as final. A class declared as final cannot be inherited.Method declared as final can be inherited but you cannot override(redefine) it.

Static with Variables

Static variables are defined as a class member that can be accessed without any object of that class. Static variable has only one single storage. All the object of the class having static variable will have the same instance of static variable. Static variables are initialized only once.

Static variable are used to represent common property of a class. It saves memory. Suppose there are 100 employee in a company. All employee have its unique name and employee id but company name will be same all 100 employee. Here company name is the common property. So if you create a class to store employee detail, company_name field will be mark as static.

```
class Employee

{

int e_id;

String name;

static String company_name = "StudyTonight";

}
```

Example of static variable

```
class ST_Employee

{

   int eid;

   String name;

   static String company_name ="StudyTonight";

   public void show()

   {

      System.out.println(eid+" "+name+" "+company_name);

   }

   public static void main( String[] args )

   {

    ST_Employee se1 = new ST_Employee();

    se1.eid = 104;

    se1.name = "Abhijit";

    se1.show();

    ST_Employee se2 = new ST_Employee();
    se2.eid = 108;
    se2.name = "ankit";
    se2.show();
   }

}
```

Static variable vs Instance Variable

Static variable	**Instance Variable**
Represent common property	Represent unique property
Accessed using class name	Accessed using object
get memory only once	get new memory each time a new object is created

```
public class Test
{
  static int x = 100;
  int y = 100;
  public void increment()
  {
     x++; y++;
  }
 public static void main( String[] args )
 {
    Test t1 = new Test();
    Test t2 = new Test();
    t1.increment();
    t2.increment();
    System.out.println(t2.y);
    System.out.println(Test.x); //accessed without any instance of class.
 }
}
```

Static Method

A method can also be declared as static. Static methods do not need instance of its class for being accessed. main() method is the most common example of static method. main() method is declared as static because it is called before any object of the class is created.

```
class Test
{

 public static void square(int x)
 {
  System.out.println(x*x);
 }

 public static void main (String[] arg)
 {

  square(8)   //static method square () is called without any instance of class.
 }
}
```

```
Output: 64
```

Q. Why main() method is static in java ?

Because static methods can be called without any instance of a class and **main()** is called before any instance of a class is created.