# Access Control, Modifiers

## Java Modifiers

The public keyword that appears in almost all of our examples:

```
public class stud
```

The public keyword is an **access modifier**, meaning that it is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

- **Access Modifiers** - controls the access level
- **Non-Access Modifiers** - do not control access level, but provides other functionality

Access Modifiers

For **classes**, you can use either public or *default*:

| Modifier | Description |
|---|---|
| public | The class is accessible by any other class |
| *default* | The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter |

For **attributes, methods and constructors**, you can use the one of the following:

| Modifier | Description |
|---|---|
| public | The code is accessible for all classes |
| private | The code is only accessible within the declared class |

| | |
|---|---|
| *default* | The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter |
| protected | The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter |

Non-Access Modifiers ( class, method, members)

For **classes**, you can use either final or abstract:

| **Modifier** | **Description (chapter no 3)** |
|---|---|
| Final class | The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter) |
| Abstract class | The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters 3) |

For **attributes and methods**, you can use the one of the following:

| **Modifier** | **Description** |
|---|---|
| final | Attributes and methods cannot be overridden/modified |

| | |
|---|---|
| static | Attributes and methods belongs to the class, rather than an **object** |
| abstract | Can only be used in an **abstract class**, and can only be used on methods. The **method does not have a body**, for example **abstract void add();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters |

Final

If you don't want the ability to override existing attribute values, declare attributes as final

```
public class finalModifiers
{
  final int x = 10;
  final double PI = 3.14;

  public static void main(String[] args)
{
    finalModifiers myObj = new finalModifiers();
   myObj.x = 50; // will generate an error: cannot assign a value to a final variable
   myObj.PI = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

Static

A static method means that it can be accessed without creating an object of the class, unlike public:

**Use of static :**

1. The static keyword is primarily used for memory management.

2. Java actually doesn't have the concept of Global variable. To define a Global variable in java, the keyword static is used.

3. A class can be declared static only if it is a **nested class**. It does not require any reference of the outer class. The property of the static class is that it does not allows us to access the non-static members of the outer class.

Static Members:

The class level members which have static keyword in their definition are called static members.

Types of Static Members:

- Java supports four types of static members

1. Static Variables
2. Static Blocks
3. Static Methods
4. Main Method (static method)

- All static members are identified and get memory location at the time of class loading by default by JVM in Method area.
- Only static variables get memory location, methods will not have separate memory location like variables.
- Static Methods are just identified and can be accessed directly without object creation.

```
public class StaticModifiers
{
// Static method

  static void showStaticMethod()
 {
   System.out.println("Static methods can be called without creating objects");
 }

 // Public method

 public void showMethod()
```

```
  {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method

  public static void main(String[ ] args)
  {
    showStaticMethod(); // Call the static method


    StaticModifiers myObj = new StaticModifiers ();
    myObj.showMethod(); // Call the public method
  }
}
```

1.Static Variable:

- A class level variable which has static keyword in its creation statement is called static variable.

```
class Demo{

static int a=10;

static int b=20;

}
```

- We can not declare local variables as static it leads to compile time error "illegal start of expression".
- Because being static variable it must get memory at the time of class loading, which is not possible to provide memory to local variable at the time of class loading.

```
class Demo{
```

```
static int a=10;

static int b=20;

public static void main(String [] args){

   //local variables should not be static

static int a=10;// compile time error: illegal start of expression

}
```

- All static variables are executed by JVM in the order of they defined from top to bottom.
- JVM provides individual memory location to each static variable in method area only once in a class life time.

Life time and scope:

- Static variable get life as soon as class is loaded into JVM and is available till class is removed from JVM or JVM is shutdown.
- And its scope is class scope means it is accessible throughout the class.

```
class staticmember
{
static int a=10;
static int b=20;

public static void show()
{

  System.out.println("a="+a);
  System.out.println("a="+b);
}

 public static void main(String [] args)
{
   System.out.println("a="+a);
  System.out.println("a="+b);
  show();
```

}
}

Abstract

An abstract method belongs to an abstract class, and it **does not have a body.** The body is provided by the subclass:

A class which contains the **abstract** keyword in its declaration is known as abstract class.

- Abstract classes may or may not contain abstract methods, i.e., methods without body ( public void get(); )
- But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be represent or make an instant
- To use an abstract class, you have to **inherit** it from another class, provide implementations of the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

**Acess modi return type methodnm(args);**

**Public void add(int a,int b)**

**{**