ACCESS SPECIFIERS

Access specifier or access modifiers in java specifies **accessibility (scope) of a data member , method, constructor or class**. It determines whether a data or method in a class can be used or invoked by other class or subclass.

Types of Access Specifiers

There are 4 types of java access specifiers:

1. Private
2. Default (no speciifer)
3. Protected
4. Public

The details about accessibility level for access specifiers are shown in following table.

Access Modifiers	Default	Private	Protected	Public
Accessible inside the class	Yes	Yes	Yes	Yes
Accessible within the subclass inside the same package	Yes	No	Yes	Yes
Accessible outside the package	No	No	No	Yes
Accessible within the subclass outside the package	No	No	Yes	Yes

Private access modifier

Private data fields and methods are accessible only inside the class where it is declared i.e accessible only by same class members. It provides low level of accessibility. Encapsulation and data hiding can be achieved using private specifier.

Example:

Role of private specifier

```
class PrivateEx{
   private int x;                    // private data
   public int y;                     // public data
   private PrivateEx(){}             // private
   constructorpublic PrivateEx(int a,int b){      //
   public constructor
      x=a;
      y=b;
   }
}
public class Main {
   public static void main(String[] args) {
```

```
    PrivateEx obj1=new PrivateEx(); // Error: private constructor cannot be applied
    PrivateEx obj2=new PrivateEx(10,20); // public constructor can be applied to
    obj2 System.out.println(obj2.y);  // public data y is accessible by a non-
    memberSystem.out.println(obj2.x);  //Error: x has private access in PrivateEx
  }
}
```

In this example, we have created two classes PrivateEx and Main. A class contains private data member, private constructor and public method. We are accessing these private members from outside the class, so there is compile time error.

Default access modifier

If the specifier is mentioned, then it is treated as default. There is no default specifier keyword. Using default specifier we can access class, method, or field which belongs to same package, but not from outside this package.

Example:

Role of default specifier

```
class DefaultEx{
   int y=10;// default data
}
public class Main {
    public static void main(String[] args)
       {DefaultEx obj=new DefaultEx();
       System.out.println(obj.y);       // default data y is accessible outside the class
    }
}
```

Sample Output:

10

In the above example, the scope of class DefaultEx and its data *y* is default. So it can be accessible within the same package and cannot be accessed from outside the package.

Protected access modifier

Protected methods and fields are **accessible within same class**, subclass inside same package and subclass in other package (through inheritance). It cannot be applicable to class.

Example:

Role of protected specifier

```
class Base{
   protected void show(){
      System.out.println("In Base");
   }
}
public class Main extends Base{
   public static void main(String[] args)
      {Main obj=new Main();
      obj.show();
   }
}
```

Sample Output:

In Base

In this example, *show()* of class Base is declared as protected, so it can be accessed from outside the class only through inheritance. Chapter 2 explains the concept of inheritance in detail.

Public access modifier

The public access specifier has highest level of accessibility. Methods, class, and fields declared as public are **accessible by any class in the same package** or in other package.

Example:

Role of public specifier

```
class PublicEx{
   public int no=10;
}
```

```
public class Main{
   public static void main(String[] args)
      {PublicEx obj=new PublicEx();
      System.out.println(obj.no);
   }
}
```

Sample Output:

10

In this example, public data *no* is accessible both by member and non-member of the class.

STATIC KEYWORD

The static keyword indicates that the member belongs to the class instead of a specific instance. It is used **to create class variable and mainly used for memory management**. The static keyword can be used with:

- Variable (static variable or class variable)
- Method (static method or class method)
- Block (static block)
- Nested class (static class)
- import (static import)

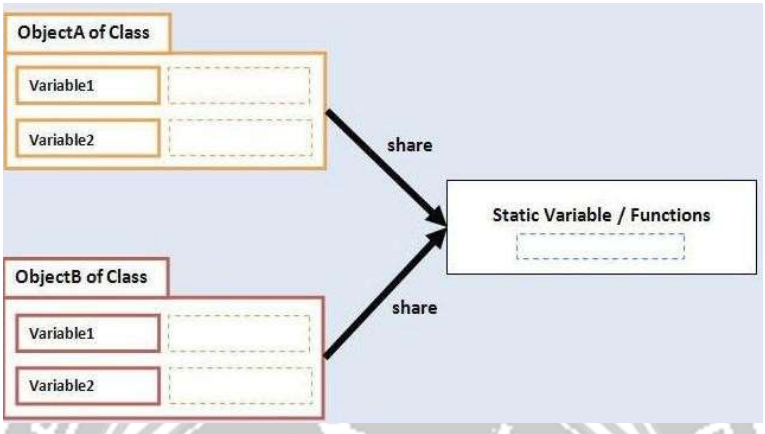
Static variable

Variable declared with keyword static is a static variable. It is a class level variable commonly shared by all objects of the class.

- Memory allocation for such variables only happens once when the class is loaded in the memory.
- scope of the static variable is class scope (accessible only inside the class)
- lifetime **is global** (memory is assigned till the class is removed by JVM).
- Automatically initialized to 0.
- It is accessible using ***ClassName.variablename***
- Static variables can be accessed directly in static and non-static methods.

Example :

Without static	With static
class StaticEx{ **int no=10;** **StaticEx(){** **System.out.println(no);** **no++;** **}** **}** **public class Main{** **public static void main(String[] args)** **{** **StaticEx obj1=new StaticEx();** **StaticEx obj2=new StaticEx();** **StaticEx obj3=new StaticEx();** **}** **}** *Sample Output:* **10** **10** **10**	**class StaticEx{** **static int no=10;** **StaticEx(){** **System.out.println(no);** **no++;** **}** **}** **public class Main{** **public static void main(String[] args)** **{** **StaticEx obj1=new StaticEx();** **StaticEx obj2=new StaticEx();** **StaticEx obj3=new StaticEx();** **}** **}** *Sample Output:* **10** **11** **12**



Static Method

The method declared with static keyword is known as static method. *main()* is most common static method.

- **It belongs to the class and not to object** of a class.
- A static method can directly **access only static variables** of class and directly invokeonly static methods of the class.
- Static methods cannot access non-static members(instance variables or instance methods) of the class
- Static method cannot access this and super references
- It can be called through the name of class without creating any instance of that class. For example, ***ClassName.methodName()***

Example:

```
class StaticEx{
   static int x;
   int y=10;
   static void display(){
      System.out.println("Static Method "+x);  // static method accessing static variable
   }
   public void show(){
      System.out.println("Non static method "+y);
      System.out.println("Non static method "+x); // non-static method can access static variable
```

```
    }
}
public class Main
{
    public static void main(String[] args)
            {StaticEx obj=new StaticEx();
            StaticEx.display();      // static method invoked without using
            objectobj.show();
    }
}
```

Sample Output:

Static Method 0

Non static method 10

Non static method 0

In this example, class StaticEx consists of a static variable x and static method display(). The static method cannot access a non-static variable. If you try to access y inside static method display(), it will result in compilation error.

```
static void display(){                    /*non-static variable y cannot be referred from a
                                           static context*/
  System.out.println("Static Method "+x+y);
}
```

Static Block

A static block is a **block of code enclosed in braces**, preceded by the keyword *static*.

- The statements within the static block are first executed automatically before main when the class is loaded into JVM.
- A class can have any number of static blocks.
- JVM combines all the static blocks in a class as single block and executes them.
- Static methods can be invoked from the static block and they will be executed as and when the static block gets executed.

Syntax:

```
static{
```

```
                ……..
        }
```

Example:

```
class StaticBlockEx{
    StaticBlockEx (){
        System.out.println("Constructor");
    }
    static {
        System.out.println("First static block");
    }
    static void show(){
        System.out.println("Inside method");

    }
    static{
        System.out.println("Second static
        block");show();
    }

    public static void main(String[] args) {
                StaticBlockEx obj=new StaticBlockEx
                ();
      }
       static{
          System.out.println("Static in main");
       }
  }
```

Sample Output:

First static block

Second static block

Inside method

Static in main

Constructor

Nested class (static class)

Nested class is a class declared inside another class. The inner class must be a static class declared using keyword static. The static nested class can refer directly to static members of the enclosing classes, even if those members are private.

Syntax:

```
class OuterClass{
        ……..
        static class InnerClass{
            ……..
        }
    }
```

We can create object for static nested class directly without creating object for outer class. For example:

OuterClassName.InnerClassName=new OuterClassName.InnerClassName();

Example:

```
class Outer{
   static int
   x=10;
   static class
      Inner{int
      y=20;
      public void show(){
         System.out.println(x+y);          // nested class accessing its own data &
outerclass static data
      }
   }
}
class Main{
   public static void main(String args[]){
      Outer.Inner obj=new Outer.Inner();  // Creating object for static nested
      classobj.show();
   }
}
```

Sample Output:

30

Static Import

The static import allows the programmer to access any static members of imported class directly. There is no need to qualify it by its name.

Syntax:

Import static package_name;

Advantage:

- Less coding is required if you have access any static member of a class oftenly.

Disadvantage:

- Overuse of static import makes program unreadable and unmaintable.

Example:

```
import static java.lang.System.*;
class StaticImportEx{
  public static void main(String args[]){
    out.println("Static Import Example");        //Now no need of System.out
}
}
```

Sample Output:

Static Import Example