

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

- Private access modifier
- Role of private constructor
- Default access modifier
- Protected access modifier
- Public access modifier
- Access Modifier with Method Overriding

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

## Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

| Access Modifier | within class | within package | outside package by subclass only | outside package |
|-----------------|--------------|----------------|----------------------------------|-----------------|
| Private         | Y            | N              | N                                | N               |
| Default         | Y            | Y              | N                                | N               |
| Protected       | Y            | Y              | Y                                | N               |
| Public          | Y            | Y              | Y                                | Y               |

### 1) Private

The private access modifier is accessible only within the class.

#### Simple example of private access modifier

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

```
class A{
    private int data=40;
    private void msg(){System.out.println("Hello java");}
}

public class Simple{
    public static void main(String args[]){
        A obj=new A();
        System.out.println(obj.data);//Compile Time Error
        obj.msg();//Compile Time Error
    }
}
```

#### Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{
    private A(){}//private constructor
    void msg(){System.out.println("Hello java");}
}

public class Simple{
    public static void main(String args[]){
        A obj=new A();//Compile Time Error
    }
}
```

 Note: A class cannot be private or protected except nested class.

### 2) Default

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

#### Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
    void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
class B{
    public static void main(String args[]){
        A obj = new A();//Compile Time Error
        obj.msg();//Compile Time Error
    }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

### 3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifier.

#### Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A{
    protected void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;

class B extends A{
    public static void main(String args[]){
        B obj = new B();
        obj.msg();
    }
}

Output:Hello
```

### 4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

#### Example of public access modifier

```
//save by A.java

package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
import pack.*;

class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}

Output:Hello
```

## Java Access Modifiers with Method Overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
class A{
    protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
    void msg(){System.out.println("Hello java");};//C.T.Error
    public static void main(String args[]){
        Simple obj=new Simple();
        obj.msg();
    }
}
```

The default modifier is more restrictive than protected. That is why, there is a compile-time error.