# Access Modifiers in Java

1. [Private access modifier](https://www.javatpoint.com/access-modifiers" \l "accessprivate)
2. [Role of private constructor](https://www.javatpoint.com/access-modifiers" \l "accessprivatecons)
3. [Default access modifier](https://www.javatpoint.com/access-modifiers" \l "accessdefault)
4. [Protected access modifier](https://www.javatpoint.com/access-modifiers" \l "accessprotected)
5. [Public access modifier](https://www.javatpoint.com/access-modifiers" \l "accesspublic)
6. [Access Modifier with Method Overriding](https://www.javatpoint.com/access-modifiers" \l "accessoverriding)

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.

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 |

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

 }

}

### 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

}

}

### 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.