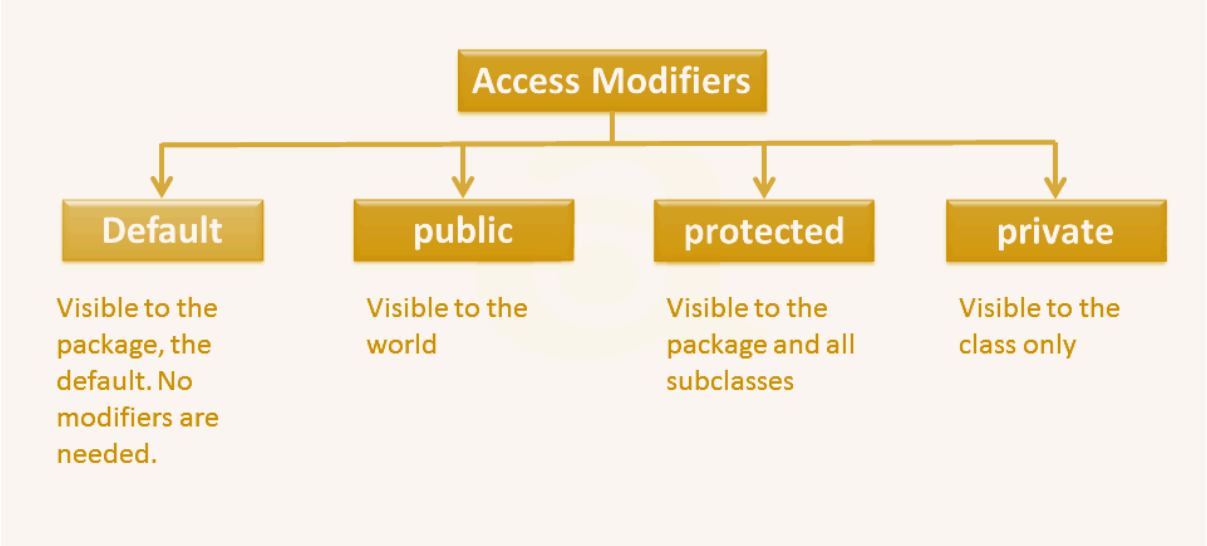
# 8.3-Access Modifiers

Access modifiers in Java are keywords that define the accessibility or scope of variables, methods, constructors, and classes. They play a crucial role in encapsulating data, ensuring security, and maintaining control over how different parts of your code interact with each other. By using access modifiers, developers can specify which parts of a program can access and modify certain data, thereby protecting sensitive information and preventing unintended interactions.

1. **Private:** It is only accessible within the class in which it is declared.
2. **Default:** The default setting is accessible within the package and is applied when no other modifier is specified.
3. **Protected**: accessible within the package and by subclasses outside the package.
4. **Public:** The public access modifier is accessible from anywhere within the program.




**Examples of Each Access Modifier**

Let's illustrate each access modifier using simple examples in a scenario where we have a package named **"other"** and a few classes outside of it.

Let's go through the examples for each access modifier and display the expected output or error messages.

## Private Access Modifier Example

**Class A in other package:**

```
package other;


public class A {
   private int marks = 6;


   private void displayMarks() {
      System.out.println("Marks: " + marks);
   }


   public void show() {
      displayMarks();
   }
}
```

**Class Demo outside 'other' package:**

```
public class Demo {
   public static void main(String[] args) {
      other.A obj = new other.A();


      // System.out.println(obj.marks);
      // Error: marks has private access in A


      // obj.displayMarks();
      // Error: displayMarks() has private access in A


      obj.show(); // This will work
   }}
```

**Output**:

Marks: 6

**Explanation**:

The private variable marks and the private method displayMarks are not accessible directly outside the class A. However, within class A, the public method show() allows access to them, resulting in the output being displayed.

## Default Access Modifier Example

**Class A in other package:**

```
package other;


public class A {
    int marks = 6;


    void displayMarks() {
        System.out.println("Marks: " + marks);
    }
}
```

**Class B in the same other package:**

```
package other;
public class B {
    public static void main(String[] args) {
        A obj = new A();
        System.out.println(obj.marks);
        obj.displayMarks();
    }
}
```

**Class Demo outside other package:**

```
public class Demo {
    public static void main(String[] args) {
        other.A obj = new other.A();


        // System.out.println(obj.marks);
        // Error: marks is not public in A; cannot be accessed from outside package


        // obj.displayMarks();
        // Error: displayMarks() is not public in A; cannot be accessed from outside package
    }
}
```

**Output** (for B class in the same package):

Marks: 6

**Explanation**:

Class A and Class B are both in the same package (`other`), so `marks` and `displayMarks()` in `Class A` can be accessed directly in `Class B` because they have default access.

Class Demo is in a different package. The code shows that when trying to access `marks` and `displayMarks()` from `Class A` in `Class Demo`, it results in a compilation error because default members are not accessible outside their package.

## Protected Access Modifier Example

**Class A in other package:**

```
package other;
public class A {
    protected int marks = 6;
    protected void displayMarks() {
        System.out.println("Marks: " + marks);
    }}
```

**Class C in otherPackage (different package but subclass):**

```
package otherPackage;
import other.A;
public class C extends A {
    public void showMarks() {
        System.out.println(marks); // Accessible in subclass outside the package
        displayMarks(); // Accessible in subclass outside the package
    }
}
```

**Class Demo outside other package:**

```
public class Demo {
    public static void main(String[] args) {
        otherPackage.C obj = new otherPackage.C();
        obj.showMarks(); // This will work

        A obj2 = new A();
        // System.out.println(obj2.marks);
        // Error: marks has protected access in A

        // obj2.displayMarks();
        // Error: displayMarks() has protected access in A
    }
}
```

**Output**:

Marks: 6

Marks: 6

**Explanation**:

The protected members (marks and displayMarks) are accessible within the same package and by subclasses outside the package. In this case, C extends A and can access protected members. However, Demo cannot access them directly.

**Public Access Modifier Example**

**Class A in other package:**

```
package other;
public class A {
    public int marks = 6;
    public void displayMarks() {
        System.out.println("Marks: " + marks);
    }
}
```

**Class Demo outside other package:**

```
public class Demo {
    public static void main(String[] args) {
        other.A obj = new other.A();
        System.out.println(obj.marks); // Accessible everywhere
        obj.displayMarks(); // Accessible everywhere
    }
}
```

**Output**:

Marks: 6

Marks: 6

**Explanation**:

The public members (marks and displayMarks) are accessible from anywhere in the program, regardless of the package.

---

**Key Points**

- **Private**: Only accessible within the same class. Not accessible from any other class.
- **Default (Package-Private)**: accessible within the same package. Not accessible from other packages.
- **Protected**: accessible within the same package and by subclasses outside the package.
- **Public**: accessible everywhere, across all packages.

## Access Modifiers Table:

| Modifier | Same Class | Same Package Subclass | Same Package Non-Subclass | Different Package Subclass | Different Package Non-Subclass |
|---|---|---|---|---|---|
| Private | Yes | No | No | No | No |
| Default | Yes | Yes | Yes | No | No |
| Protected | Yes | Yes | Yes | Yes | No |
| Public | Yes | Yes | Yes | Yes | Yes |

## FAQs on Access Modifiers

1. What are access modifiers in Java?
    - Access modifiers are keywords that control the visibility and accessibility of classes, methods, and variables in Java.
2. Can a private method be overridden?
    - No, private methods cannot be overridden as they are not accessible outside their class.
3. What is the difference between protected and default access?
    - Protected access allows visibility within the package and to subclasses, while default access restricts visibility only within the package.
4. Can we have a private constructor in Java?
    - Yes, a private constructor can be used to prevent the instantiation of a class from outside.