

Access Modifiers and Non Access Modifiers

-K.L.MADHAVI


Access Modifiers in java

Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member. It provides security, accessibility, etc to the user depending upon the access modifier used with the element.

Best Practices for Using Access Modifiers

1. Data Security
2. Code Organization
3. Inheritance

Types of Access Modifiers in Java

- 1.Default-No keyword required
 - 2.Private
 - 3.Protected
 - 4.Public
- 

Continue...

1.Default Access Modifier: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.

In this example, we will create two packages and the classes in the packages will be having the default access modifiers and we will try to access a class from one package from a class of the second package.

```
// Java program to illustrate default modifier  
package p1;  
// Class D is having Default access modifier  
class D  
{  
    void display()  
    {  
        System.out.println("Hello World!");  
    }  
}
```

Continue...

```
// Java program to illustrate error while using class from different package with default modifier
package p2;
import p1.*;
// This class is having default access modifier
class DisplayNew
{
    public static void main(String args[])
    {
        // Accessing class D from package p1
        D obj = new D();
        obj.display();//compile time error
    }
}
```

Continue...

2.Private Access Modifier:The private access modifier is specified using the keyword `private`. The methods or data members declared as private **are accessible only within the class in which they are declared.**

In this example, we will create two classes A and B within the same package p1. We will declare a method in class A as private and try to access this method from class B and see the result.

```
// Java program to illustrate error while Using class from different package with Private Modifier
package p1;
class A
{
    private void display()
    {
        System.out.println("display method");
    }
}
class B
{
    public static void main(String args[]) {
        A obj = new A();
        obj.display(); // Trying to access private method of another class
    }
}
```

Continue...

3.Protected Access Modifier:The protected access modifier is specified using the keyword protected.

The methods or data members declared as protected are **accessible within the same package or subclasses in different packages.**

In this example, we will create two packages p1 and p2. Class A in p1 is made public, to access it in p2. The method display in class A is protected and class B is inherited from class A and this protected method is then accessed by creating an object of class B.

```
package p1;
public class A
{
    protected void display()
    {
        System.out.println("Hello");
    }
}
```

Continue...

```
package p2;  
// importing all classes of package p1  
import p1.*;  
// Class B is subclass of A  
class B extends A  
{  
    public static void main(String args[])  
    {  
        B obj = new B();  
        obj.display();  
    }  
}
```

Continue...

4.Public Access modifier:The public access modifier is specified using the keyword `public`.

- The public access modifier has the widest scope among all other access modifiers.
- Classes, methods, or data members that are declared as public are accessible from everywhere in the program. There is no restriction on the scope of public data members.

```
package p1;  
public class A  
{  
    public void display()  
    {  
        System.out.println("display method");  
    }  
}
```


Continue...

```
package p2;  
import p1.*;  
class B  
{  
    public static void main(String args[])  
    {  
        A obj = new A();  
        obj.display();  
    }  
}
```

Non-Access Modifiers in java

Non-access modifiers provide information about the characteristics of a class, method, or variable to the JVM. Seven types of Non-Access modifiers are present in Java. They are –

1.static

2.final

3.abstract

4.synchronized

5.volatile

Continue...

1. static: The static keyword means that the entity to which it is applied is available outside any particular instance of the class. That means the static methods or the attributes are a part of the class and not an object. The memory is allocated to such an attribute or method at the time of class loading. This field exists across all the class instances, and without creating an object of the class, they can be called.

```
import java.io.*;
// static variable
class StaticClass {
    static String s = "Static variable";
}
class Main {
    public static void main(String[] args)
    {
        // No object required
        System.out.println(StaticClass.s);
    }
}
```

Continue...

2. final: The final keyword indicates that the specific class cannot be extended or a method cannot be overridden.

```
import java.io.*;
final class FinalClass{
    String s1 = "finalclass";
}
class ExtendedClass extends FinalClass {

    String s2 = "subclass";
}
class Main {
    public static void main(String[] args)
    {
        // creating object
        ExtendedClass obj = new ExtendedClass();

        System.out.println(obj.s1);
        System.out.println(obj.s2);
    }
}
```

Continue...

3. abstract: abstract keyword is used to declare a class as partially implemented means an object cannot be created directly from that class. Any subclass needs to be either implement all the methods of the abstract class, or it should also need to be an abstract class. The abstract keyword cannot be used with static, final, or private keywords.

```
abstract class AC{
    abstract void myMethod();
}
class MyClass extends AC{
    void myMethod(){
        System.out.println("body of Abstract method");
    }
}
class Main{
    public static void main(String[] args) {
        MyClass obj=new MyClass();
        obj.myMethod();
    }
}
```

Continue...

4. synchronized:synchronized keyword prevents a block of code from executing by multiple threads at once. It is very important for some critical operations.

5. volatile:volatile keyword only and only ensures that changes that are made to variables are reflected across all threads that are accessing the same variable/thread. The usage of volatile keywords does not guarantee thread safety for class but it does not ensure thread safety and concurrency issues. Hence if we want our class to be thread-safe then we need to invoke it via the below technologies:

- Using synchronized keyword
- Using monitors
- using concurrency techniques (language dependent)

Note:The volatile keyword is only applicable to a variable.