**Write code (preferably test automation example) to implement Access modifiers, Enums. Operator hierarchy**

Access modifiers: -

Access modifiers in Java helps to restrict the scope of a class, constructor, variable, method, or data member

 There are four types of access modifiers available in java:

1. **Default** – No keyword required

When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default.

The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

1. **Private**

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.

Top-level classes or interfaces cannot be declared as private because

private means “only visible within the enclosing class”

1. **Protected**

The protected access modifier is specified using the keyword **protected**.

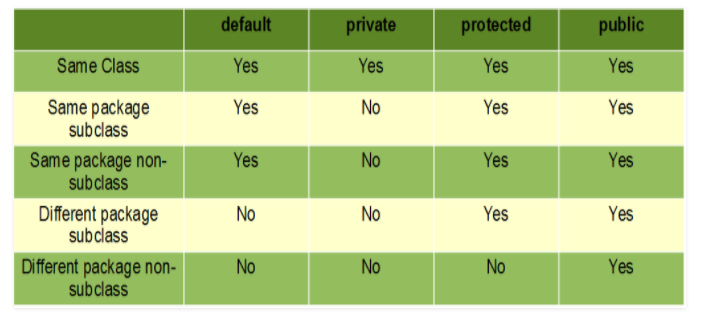
The methods or data members declared as protected are **accessible within the same package or subclasses in different packages.**

1. **Public**

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.



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 Modifier is having Default access modifier

class Modifier

{

    void display()

    {

        System.out.println("Hello World!");

    }

}

// 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 ModifierNew

{

    public static void main(String args[])

    {

        // Accessing class Modifer from package p1

        Modifier modifier = new Modifier();

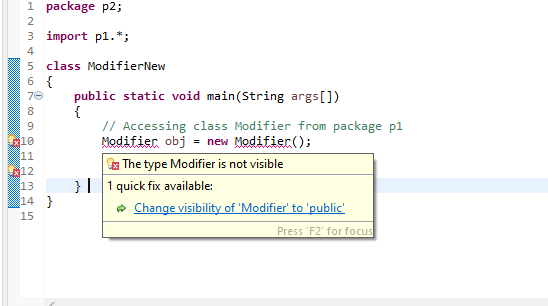
        modifier.display();

    }

}

Output:

Compile time error



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.

// Java program to illustrate

// protected modifier

package p1;

// Class A

public class A

{

protected void display()

    {

        System.out.println("Hello World");

    }

}

// Java program to illustrate

// protected modifier

package p2;

import p1.\*; // importing all classes in package p1

// Class B is subclass of A

class B extends A

{

public static void main(String args[])

{

    B obj = new B();

    obj.display();

}

}

**Output:**

Hello World

Enums in Test Automation

An enum type is a special data type that enables for a variable to be a set of predefined constants

**package** p1;

**public** **enum** MenuOption {

***FIRST\_OPTION***(0),***SECOND\_OPTION***(1);

**private** **final** **int** value;

MenuOption(**int** value){

**this**.value=value;

}

**public** **int** getValue() {

**return** **this**.value;

}

}

**package** p1;

**import** java.util.List;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.support.FindBy;

**public** **class** SomePageClass {

@FindBy(css=".menu")

**private** List<WebElement> menuItems;

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

}

**public** **void** selectMenuItem(MenuOption menuOption) {

menuItems.get(menuOption.getValue()).click();

}

}

**package** p1;

**public** **class** SomeTestClass {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**new** SomePageClass().selectMenuItem(MenuOption.***FIRST\_OPTION***);

}

}

Operator hierarchy

Operator hierarchy determines the order in which the operators in an expression are evaluated.

int myInt = 12 - 4 \* 2;

What will be the value of myInt? Will it be (12 - 4)\*2, that is, 16? Or it will be 12 - (4 \* 2), that is, 4?

When two operators share a common operand, 4 in this case, the operator with the highest precedence is operated first.

In Java, the precedence of \* is higher than that of -. Hence, the multiplication is performed before subtraction, and the value of myInt will be 4.

**Operator Precedence Table**

The table below lists the precedence of operators in Java; higher it appears in the table, the higher its precedence.

|  |  |
| --- | --- |
| Java Operator Precedence | |
| Operators | Precedence |
| postfix increment and decrement | ++ -- |
| prefix increment and decrement, and unary | ++ -- + - ~ ! |
| multiplicative | \* / % |
| additive | + - |
| shift | << >> >>> |
| relational | < > <= >= instanceof |
| equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| ternary | ? : |
| assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

### Example: Operator Precedence

class Precedence {

public static void main(String[] args) {

int a = 10, b = 5, c = 1, result;

result = a-++c-++b;

System.out.println(result);

}

}

**Output**:

2

The operator precedence of prefix ++ is higher than that of - subtraction operator. Hence,

result = a-++c-++b;

is equivalent to

result = a-(++c)-(++b);