**ACCESS SPECIFIERS OR MODIFIERS**

* There are **2 types** of access of modifiers in java.
* Access modifiers
* Non-Access modifiers
* An Access modifier controls the access of class members.
* 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.

**TYPES OF ACCESS MODIFIERS:**

There are four types of Java access modifiers:

* **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
* **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.
* **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.
* **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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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** |

**Example:**

// Class in package1

package package1;

// Public class accessible from everywhere

public class Person {

// Public variable accessible from anywhere

public String name;

// Protected variable accessible within the same package and subclasses

protected int age;

// Default variable (no specifier), accessible only within the package

String address;

// Private variable accessible only within this class

private String phoneNumber;

// Public constructor accessible from anywhere

public Person(String name, int age, String address, String phoneNumber) {

this.name = name;

this.age = age;

this.address = address;

this.phoneNumber = phoneNumber;

}

// Public method accessible from anywhere

public void showDetails() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

System.out.println("Address: " + address);

// Can access private variable within the same class

System.out.println("Phone Number: " + phoneNumber);

}

}

**Accessing from a Different Class in the Same Package**

// Another class in package1

package package1;

public class Main {

public static void main(String[] args) {

// Creating an object of Person class

Person person = new Person("John", 30, "123 Street", "555-1234");

// Accessing public field

System.out.println("Name: " + person.name); // Works, public

// Accessing protected field (works within the same package)

System.out.println("Age: " + person.age); // Works, protected

// Accessing default field (works within the same package)

System.out.println("Address: " + person.address); // Works, default

// Accessing private field (not allowed)

// System.out.println(person.phoneNumber); // Error, private

// Accessing public method

person.showDetails(); // Works, public

}

}

**Accessing from a Different Package (Inheritance Example)**

// Class in package2 that inherits from Person in package1

package package2;

import package1.Person;

public class Employee extends Person {

// Constructor

public Employee(String name, int age, String address, String phoneNumber) {

super(name, age, address, phoneNumber);

}

// Method to display employee details

public void displayEmployeeDetails() {

// Accessing public field

System.out.println("Name: " + name); // Works, public

// Accessing protected field (works in subclass)

System.out.println("Age: " + age); // Works, protected

// Accessing default field (not allowed across packages)

// System.out.println(address); // Error, default

// Accessing private field (not allowed)

// System.out.println(phoneNumber); // Error, private

}

}

**Main Class in package2**

package package2;

public class Main {

public static void main(String[] args) {

Employee emp = new Employee("Alice", 25, "456 Avenue", "555-5678");

// Accessing public method

emp.displayEmployeeDetails(); // Works, public and protected

}

}

**TYPES OF NON-ACCESS MODIFIERS IN JAVA**

* static
* final
* abstract
* synchronized
* volatile
* transient
* native
* strictfp

**static modifier**

**Purpose:** Belongs to the class rather than instances (objects) of the class.

**Use with:** Variables, methods, blocks, inner classes.

**Example:**

public class Main {

// Static variable

static int counter = 0;

// Static method

static void incrementCounter() {

counter++;

}

public static void main(String[] args) {

incrementCounter();

System.out.println("Counter: " + counter); // Output: Counter: 1

}

}

**Explanation:** The static variable and method belong to the class itself, and they are shared among all objects of that class.

**final modifier**

**Purpose:** Prevents modification.

**Use with:** Classes, methods, and variables.

**Example:**

public class Main {

// Final variable (constant)

final int MAX\_VALUE = 100;

// Final method

final void show() {

System.out.println("This is a final method.");

}

public static void main(String[] args) {

Main obj = new Main();

// obj.MAX\_VALUE = 200; // Error: Cannot modify final variable

obj.show();

}

}

// Final class

final class FinalClass {

// This class cannot be extended (no inheritance allowed)

}

**Explanation:**

* **Final class:** Cannot be inherited.
* **Final method:** Cannot be overridden.
* **Final variable:** Its value cannot be changed (constant).

**abstract modifier**

**Purpose:** Used for declaring abstract classes and methods that must be implemented by subclasses.

**Use with:** Classes, methods.

**Example:**

abstract class Animal {

// Abstract method (must be implemented by subclasses)

abstract void sound();

// Concrete method

void sleep() {

System.out.println("Sleeping...");

}

}

// Dog class extends abstract class Animal

class Dog extends Animal {

@Override

void sound() {

System.out.println("Barks");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

myDog.sound(); // Output: Barks

myDog.sleep(); // Output: Sleeping...

}

}

**Explanation:**

* Abstract methods do not have a body and must be implemented in subclasses. An abstract class cannot be instantiated directly.

**volatile modifier**

**Purpose:** Ensures that changes to a variable are visible to all threads immediately.

**Use with:** Ensures that changes to a variable are visible to all threads immediately.

**Example:**

public class Main {

// Volatile variable

private volatile boolean flag = true;

public void toggleFlag() {

flag = !flag; // The change is visible to all threads

}

}

**Explanation:**

* The volatile keyword tells the JVM to read the variable's value from main memory, ensuring visibility of changes across threads.

**native modifier**

**Purpose:** Specifies that a method is implemented in native code using another programming language like C or C++.

**Use with:** Methods.

**Example:**

public class Main {

// Native method (implementation provided in C/C++)

public native void print();

static {

// Load the native library

System.loadLibrary("nativeLib");

}

}

**Explanation:**

* Native methods are implemented in other languages (like C/C++) and invoked in Java.

**strictfp modifier**

**Purpose:** Ensures that floating-point calculations follow strict IEEE 754 standards for consistency across platforms.

**Use with:** Classes, methods.

**Example:**

strictfp class Calculator {

// Method with strict floating-point precision

strictfp double computeSum(double a, double b) {

return a + b;

}

}

**Explanation:**

* The strictfp modifier ensures that floating-point operations yield consistent results across different platforms.