### **Access Specifiers and Modifiers in Java**

In Java, access specifiers and modifiers control the visibility and accessibility of classes, methods, and variables. This ensures that the encapsulation principle of object-oriented programming is maintained. Let's dive into the theory first and then look at some detailed code examples.

#### **Access Specifiers**

There are four types of access specifiers in Java:

1. **Public**: The members are accessible from any other class.
2. **Protected**: The members are accessible within the same package and by subclasses.
3. **Default (no specifier)**: The members are accessible only within the same package.
4. **Private**: The members are accessible only within the same class.

#### **Modifiers**

1. **Static**: Belongs to the class rather than any instance of the class.
2. **Final**: Cannot be modified (variables), overridden (methods), or inherited (classes).
3. **Abstract**: A class that cannot be instantiated or a method that must be implemented by subclasses.
4. **Synchronized**: Used in multi-threading to prevent thread interference.
5. **Transient**: Prevents serialization of the field.
6. **Volatile**: The value of this field will never be cached thread-locally; all reads and writes will go straight to "main memory".

### **Examples**

#### **1. Public Access Specifier**

| // File: PublicExample.java package mypackage;  public class PublicExample {  public int publicVar = 10;   public void display() {  System.out.println("Public variable: " + publicVar);  } }  // File: TestPublicExample.java package mypackage;  public class TestPublicExample {  public static void main(String[] args) {  PublicExample example = new PublicExample();  example.display();  System.out.println("Accessing public variable directly: " + example.publicVar);  } } |
| --- |

#### **2. Protected Access Specifier**

| // File: ProtectedExample.java package mypackage;  public class ProtectedExample {  protected int protectedVar = 20;   protected void display() {  System.out.println("Protected variable: " + protectedVar);  } }  // File: SubClass.java package mypackage;  public class SubClass extends ProtectedExample {  public void show() {  display(); // Accessible in subclass  System.out.println("Accessing protected variable in subclass: " + protectedVar);  } }  // File: TestProtectedExample.java package mypackage;  public class TestProtectedExample {  public static void main(String[] args) {  SubClass sub = new SubClass();  sub.show();  } } |
| --- |

#### **3. Default Access Specifier**

| // File: DefaultExample.java package mypackage;  class DefaultExample {  int defaultVar = 30;   void display() {  System.out.println("Default variable: " + defaultVar);  } }  // File: TestDefaultExample.java package mypackage;  public class TestDefaultExample {  public static void main(String[] args) {  DefaultExample example = new DefaultExample();  example.display();  System.out.println("Accessing default variable directly: " + example.defaultVar);  } } |
| --- |

#### **4. Private Access Specifier**

| // File: PrivateExample.java package mypackage;  public class PrivateExample {  private int privateVar = 40;   private void display() {  System.out.println("Private variable: " + privateVar);  }   public void accessPrivateMethod() {  display(); // Accessible within the same class  } }  // File: TestPrivateExample.java package mypackage;  public class TestPrivateExample {  public static void main(String[] args) {  PrivateExample example = new PrivateExample();  example.accessPrivateMethod();  // example.display(); // Error: display() has private access  // System.out.println(example.privateVar); // Error: privateVar has private access  } } |
| --- |

#### **5. Static Modifier**

| // File: StaticExample.java package mypackage;  public class StaticExample {  static int staticVar = 50;   static void display() {  System.out.println("Static variable: " + staticVar);  } }  // File: TestStaticExample.java package mypackage;  public class TestStaticExample {  public static void main(String[] args) {  StaticExample.display();  System.out.println("Accessing static variable directly: " + StaticExample.staticVar);  } } |
| --- |

#### **6. Final Modifier**

| // File: FinalExample.java package mypackage;  public final class FinalExample {  final int finalVar = 60;   final void display() {  System.out.println("Final variable: " + finalVar);  } }  // File: TestFinalExample.java package mypackage;  public class TestFinalExample {  public static void main(String[] args) {  FinalExample example = new FinalExample();  example.display();  // example.finalVar = 70; // Error: Cannot assign a value to final variable  } } |
| --- |

#### **7. Abstract Modifier**

| // File: AbstractExample.java package mypackage;  abstract class AbstractExample {  abstract void display(); }  class ConcreteExample extends AbstractExample {  @Override  void display() {  System.out.println("Abstract method implemented in ConcreteExample");  } }  // File: TestAbstractExample.java package mypackage;  public class TestAbstractExample {  public static void main(String[] args) {  ConcreteExample example = new ConcreteExample();  example.display();  } } |
| --- |

#### **8. Synchronized Modifier**

| // File: SynchronizedExample.java package mypackage;  public class SynchronizedExample {  private int count = 0;   public synchronized void increment() {  count++;  }   public int getCount() {  return count;  } }  // File: TestSynchronizedExample.java package mypackage;  public class TestSynchronizedExample {  public static void main(String[] args) {  SynchronizedExample example = new SynchronizedExample();   Thread t1 = new Thread(() -> {  for (int i = 0; i < 1000; i++) {  example.increment();  }  });   Thread t2 = new Thread(() -> {  for (int i = 0; i < 1000; i++) {  example.increment();  }  });   t1.start();  t2.start();   try {  t1.join();  t2.join();  } catch (InterruptedException e) {  e.printStackTrace();  }   System.out.println("Final count: " + example.getCo |
| --- |