# Access Modifiers

In Java, **access modifiers** are keywords used to control the visibility (or accessibility) of **classes, constructors, methods, and variables**. They help implement **encapsulation**.

## 📘 1. public

* **Access Level:** Anywhere (global)
* **Usage:** When you want something to be visible from **any class** in any package.

**Example:**

public class Animal {  
 public void makeSound() {  
 System.*out*.println("Some sound");  
 }  
}

## 📘 2. protected

* **Access Level:** Within the **same package** and **subclasses** in any package
* **Usage:** Often used for allowing subclass customization while restricting global access.

**Example:**

class Animal {  
 protected void eat() {  
 System.*out*.println("Animal is eating");  
 }  
}  
  
class Dog extends Animal {  
 void bark() {  
 eat(); // Allowed  
 }  
}

## 📘 3. (No Modifier) — *Default* (a.k.a. Package-Private)

* **Access Level:** Only within the **same package**
* **Usage:** Restrict access to package-level only.

**Example:**

class Bird {  
 void fly() {  
 System.*out*.println("Flying...");  
 }  
}

## 📘 4. private

* **Access Level:** Only within the **same class**
* **Usage:** Used for **encapsulation**—to hide data and internal logic.

**Example:**

public class BankAccount {  
 private double balance;  
  
 public void deposit(double amount) {  
 if (amount > 0) {  
 balance += amount;  
 }  
 }  
  
 private void audit() {  
 System.*out*.println("Auditing account");  
 }  
}

**📌 Key Points**

* A **top-level class** (outer class) can only be public or package-private (default).
* **Private** and **protected** top-level classes are not allowed. The .java file can have only one public top-level class or interface, and it must have the same name as the file.
* You can apply different access modifiers for **inner classes, methods, fields, and constructors**.

## 🔐 Access Modifier: protected vs *default* (no modifier)

| **Feature** | **protected** | **default *(no modifier)*** |
| --- | --- | --- |
| **Same Class** | ✅ Yes | ✅ Yes |
| **Same Package (any class)** | ✅ Yes | ✅ Yes |
| **Subclass in Same Package** | ✅ Yes | ✅ Yes |
| **Subclass in Different Package** | ✅ Yes *(via inheritance only)* | ❌ No |
| **Non-subclass in Different Package** | ❌ No | ❌ No |
| **Usage Allowed For** | Classes (methods/fields), Interfaces (methods) | Same as protected |
| **Can be applied to Class?** | ❌ Only for inner classes | ❌ Only for inner classes |

## 📌 Allowed Access Modifiers for Nested Interface

| **Nested Interface Location** | **Allowed Access Modifiers** |
| --- | --- |
| Inside Interface | public (only) and static(Since **interfaces can't have instance members**, all nested types (like classes or interfaces) are **implicitly static**, and must be **public** so that they’re usable.) |
| Inside Class | public, protected, default, private |

## Nested Class inside a Class

There are **4 types of nested classes inside a class**:

| **Type** | **Static?** | **Can have Access Modifier?** | **Can access instance members of outer class?** |
| --- | --- | --- | --- |
| Static Nested Class | ✅ Yes | ✅ public, protected, private, default | ❌ No |
| Non-static Inner Class | ❌ No | ✅ public, protected, private, default | ✅ Yes |
| Local Class (inside method) | ❌ No | ❌ No access modifiers allowed | ✅ Yes |
| Anonymous Inner Class | ❌ No | ❌ No access modifiers allowed | ✅ Yes |

**🔍 Summary Table**

| **Context** | **Can have private, protected?** | **Implicitly static?** | **Access to outer class instance?** |
| --- | --- | --- | --- |
| Interface -> Class | ❌ No (public static only) | ✅ Yes | ❌ No |
| Class -> Static Class | ✅ Yes | ✅ Yes | ❌ No |
| Class -> Inner Class | ✅ Yes | ❌ No | ✅ Yes |
| Method -> Local Class | ❌ No modifier | ❌ No | ✅ Yes |
| Method -> Anonymous | ❌ No modifier | ❌ No | ✅ Yes |

# Access modifiers with Constructor 🡪

**🔍 Summary Table**

| **Modifier** | **Same Class** | **Same Package** | **Subclass (diff pkg)** | **Outside Package** |
| --- | --- | --- | --- | --- |
| public | ✅ | ✅ | ✅ | ✅ |
| protected | ✅ | ✅ | ✅ via super() | ❌ via object |
| default | ✅ | ✅ | ❌ | ❌ |
| private | ✅ | ❌ | ❌ | ❌ |

# Protected Constructor 🡪

A protected constructor allows subclassing **outside the package**, but **not direct instantiation** using new in a non-subclass.

**✅ A protected constructor can be accessed:**

| **Access Location** | **Can access protected constructor?** |
| --- | --- |
| Same class | ✅ Yes |
| Same package (any class) | ✅ Yes |
| Subclass in **same package** | ✅ Yes |
| Subclass in **different package** | ✅ Yes (but only via super()) |
| Non-subclass in different package | ❌ No |

📌 Example 1: Subclass in Same Package – ✅ Works

package pack1;  
  
public class Animal {  
 protected Animal() {  
 System.*out*.println("Animal constructor");  
 }  
}  
  
class Dog extends Animal {  
 Dog() {  
 super(); // ✅ allowed  
 }  
}

📌 Example 2: Subclass in Different Package – ✅ Works

package pack1;  
  
public class Animal {  
 protected Animal() {  
 System.*out*.println("Animal constructor");  
 }  
}

package pack2;  
import pack1.Animal;  
  
public class Dog extends Animal {  
 Dog() {  
 super(); // ✅ allowed (subclass can call protected constructor)  
 }  
}

📌 Example 3: Different Package, No Inheritance – ❌ Compile Error

package pack1;  
  
public class Animal {  
 protected Animal() {  
 System.*out*.println("Animal constructor");  
 }  
}

package pack2;  
import pack1.Animal;  
  
public class Test {  
 public static void main(String[] args) {  
 Animal a = new Animal(); // ❌ ERROR: constructor is protected  
 }  
}

# Interview Questions 🡪

1. Programs

### Code (pack1 → Animal.java):

public class Animal {  
 protected void sound() {  
 System.*out*.println("Animal sound");  
 }  
}

### Code (pack2 → Dog.java):

import pack1.Animal;  
  
public class Dog extends Animal {  
 void makeSound() {  
 sound(); // Line A ✅ (Allowed via inheritance)  
 }  
  
 public static void main(String[] args) {  
 Dog d = new Dog();  
 d.makeSound(); // Line B ✅  
  
 Animal a = new Animal();  
 // a.sound(); // Line C ❌ (Will cause compile error if uncommented)  
 }  
}

✔️ **Line A** & **Line B** ✅ work fine — protected is accessible via inheritance.  
❌ **Only Line C**, if uncommented, causes **compile-time error**.

📤 **Output if Line C is commented:** Animal sound

1. Program

package com.example;  
  
public class Person {  
 private Person() {  
 System.*out*.println("Private constructor");  
 }  
  
 public static Person create() {  
 return new Person(); // ✅ OK (same class)  
 }  
}

package com.test;  
import com.example.Person;  
  
public class Test {  
 public static void main(String[] args) {  
 Person p = new Person(); // ❌ Compile-time Error  
 }  
}

package com.test;  
import com.example.Person;  
  
public class Test {  
 public static void main(String[] args) {  
 Person p = Person.create(); // ✅ Works via public static factory method  
 }  
}

1. Program

// File: pack1/Parent.java  
package pack1;  
  
public class Parent {  
 private void show() {  
 System.*out*.println("Parent show()");  
 }  
  
 public void display() {  
 show();  
 }  
}

// File: pack2/Child.java  
package pack2;  
  
import pack1.Parent;  
  
public class Child extends Parent {  
 private void show() {  
 System.*out*.println("Child show()");  
 }  
  
 public static void main(String[] args) {  
 Child c = new Child();  
 c.display(); // Line X  
 }  
}

✅ **Parent show()**

**private methods are not inherited** — so even if a subclass defines a method with the same signature, it's just a **new method**, not an override.