What is protected access modifier?

Scope: Visible within the same package and subclasses in other packages.

Commonly used in inheritance to provide controlled access to child classes.

package package1;  
  
public class ProtectedClass {  
 protected String protectedVariable = "I am protected";  
  
 protected void display(){  
 System.*out*.println(protectedVariable);  
 }  
}

package package2;  
  
import package1.ProtectedClass;  
  
public class TestProtected extends ProtectedClass {  
 public static void main(String[] args) {  
 TestProtected obj = new TestProtected();  
 obj.display(); // Accessible through inheritance  
 }  
}

What is Default Access Modifier(No keyword)

Scope: Restricted to the same package. Also “package-private” access.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modifiers | Same Class | Same Package | SubClass(Different Package) | Other packages |
| Public | ✔ | ✔ | ✔ | ✔ |
| Protected | ✔ | ✔ | ✔ | ❌ |
| Default | ✔ | ✔ | ❌ | ❌ |
| Private | ✔ | ❌ | ❌ | ❌ |

Abstraction:

Hides the implementation details from user and exposes only the essential functionalities.

Example: A car’s interface for driving hides the internal structure/ mechanism of its engine.

Inheritance:

It enables a child class to inherit properties and methods from parent class.

package inheritanceExample;  
  
public class Vehicle {  
 String brand = "Ford";  
}

package inheritanceExample;  
  
public class Car extends Vehicle{  
 String model = "Mustang";  
}

package inheritanceExample;  
  
public class InheritanceEx {  
 public static void main(String[] args) {  
 Car myCar = new Car();  
 System.*out*.println(myCar.brand + " " + myCar.model);  
 }  
}

Polymorphism:

Allows methods or objects to behave differently based on the context.

Types:

1. Compile-Time Polymorphism (Method Overloading)

Multiple methods with the same name but different parameters.

package polymorphismExample;  
  
public class Calculator {  
   
 int add(int a, int b){  
 return a+b;  
 }  
   
 double add(double a, double b){  
 return a+b;  
 }  
   
}

1. Runtime Polymorphism (Method Overriding)

package runtimeExample;  
  
public class Animal {  
 void sound(){  
 System.*out*.println("Animal makes a sound");  
 }  
}

package runtimeExample;  
  
public class Dog extends Animal{  
 void sound(){  
 System.*out*.println("Dog Barks");  
 }  
}

Benefits of OOPs:

1. Reusability
2. Modularity
3. Maintainability
4. Scalability