## Key principles of OOP:

Encapsulation: It combines data(attributes) and methods into a single unit, restricting direct access to the data.
WHY? - Protects data integrity by allowing controlled access using getter and setter.

What are access modifiers?

- Public Access Modifier
  - Scope: The widest visibility. Members declared as public can be accessed from anywhere in the program.

```
package package1;

public class PublicClass {
    public String publicVariable= "I am public";

    public void display(){
        System.out.println(publicVariable);
    }
}
```

```
package package2;

import package1.PublicClass;

public class TestPublic {
    public static void main(String[] args) {
        PublicClass obj = new PublicClass();
        obj.display(); // Accessible here because it is public
    }
}
```

## What is Private Access Modifier?

Scope: Most restrictive, If we are declaring members as private then they are accessible only within the same class.

```
package privateexample;

public class PrivateClass {
    private String privateVariable = "I am Private";

    private void display(){
        System.out.println(privateVariable);
    }

    public void accessPrivate(){
```

```
        display(); // Allowed within the same class only
    }
}
```

```
package privateexample;

public class TestPrivate {
    public static void main(String[] args) {
        PrivateClass obj= new PrivateClass();
        // obj.display(); // Error: Not accessible
        obj.accessPrivate(); // Indirect access is allowed
    }
}
```

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 package3;

public class ProtectedClass {
    protected String protectedVariable = "I am protected";

    protected void display(){
        System.out.println(protectedVariable);
    }
}
```

```
package package4;

import package3.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 known as “package-private” access.

| Modifiers | Same Class | Same Package | Subclass(Different Package) | Other packages |
|---|---|---|---|---|
| Public | ✓ | ✓ | ✓ | ✓ |
| Protected | ✓ | ✓ | ✓ | ✗ |
| Default | ✓ | ✓ | ✗ | ✗ |
| Private | ✓ | ✗ | ✗ | ✗ |

## Abstraction:

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

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

## Inheritance:

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

```
package iheritanceExample;

public class Vehicle {
    String brand = "Ford";
}
```

```
package iheritanceExample;

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

```java
package iheritanceExample;

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

```java
package polymorphismExample;

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

    double add(double a, double b){
        return a+b;
    }
}
```

2.Runtime Polymorphism(Method Overriding)

```java
package polymorphismExample.runtimepoly;

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

```java
package polymorphismExample.runtimepoly;

public class Dog extends Animal{
    void sound(){
        System.out.println("Dog barks");
    }
}
```

Benefits of OOPs:

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