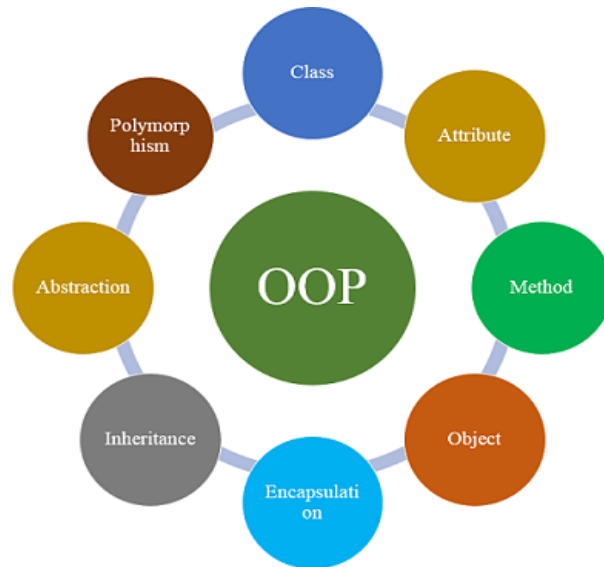


Chapter Four

Abstraction and Encapsulation



Mulugeta G. and Sufian K.

Abstraction

- **Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.**
- **In other words, the user will have the information on what the object does instead of how it does it.**
- **Abstraction is achieved using**
Abstract classes and interfaces.

Abstraction

Abstract Class

- A class which contains the **abstract** keyword in its declaration
- may or may not contain **abstract methods**, i.e., methods without body (`public void get();`)
- But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it **cannot be instantiated**.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstraction

Abstract Method

- abstract keyword is used to declare the method as abstract.
- You have to place the **abstract** keyword before the method name in the method declaration.
- An abstract method **contains a method signature**, but **no method body**.
- Instead of curly braces, an abstract method will have a **semoi colon (;)** at the end.
- **Example:** `public abstract double computePay();`

Abstraction

Example

```
public abstract class Employee {  
    private String name;  
    private String address;  
    private int number;  
    public abstract double computePay();  
    // Remainder of class definition  
}
```

Abstraction

Example

```
abstract class Animal {  
    // Abstract method (no implementation)  
    public abstract void makeSound();  
    // Concrete method  
    public void eat() {  
        System.out.println("This animal eats food.");  
    }  
}
```

Abstraction

```
class Dog extends Animal {  
    public void makeSound() {  
        System.out.println("Bark!");  
    }  
}  
  
class Cat extends Animal {  
    public void makeSound() {  
        System.out.println("Meow!");  
    }  
}
```

Abstraction

```
public static void main(String[] args) {  
    Animal dog = new Dog();  
    dog.makeSound(); // Bark!  
    dog.eat();       // This animal eats food.  
  
    Animal cat = new Cat();  
    cat.makeSound(); // Meow!  
}  
}
```


Abstraction

Interface

- **Interfaces are another method of implementing abstraction in Java.**
- **The key difference is that, by using interfaces, we can achieve 100% abstraction in Java classes.**
- **In Java or any other language, interfaces include both methods and variables but lack a method body.**
- **Apart from abstraction, interfaces can also be used to implement inheritance in Java.**

Abstraction

Example

```
interface Shape {  
    void draw();  
}
```

```
class Circle implements Shape {  
    public void draw() {  
        System.out.println("Drawing a Circle.");  
    }  
}
```

Abstraction

```
class Rectangle implements Shape {  
    public void draw() {  
        System.out.println("Drawing a Rectangle.");  
    }  
}  
  
public static void main(String[] args) {  
    Shape s1 = new Circle();  
    Shape s2 = new Rectangle();  
    s1.draw(); // Drawing a Circle.  
    s2.draw(); // Drawing a Rectangle.  
}
```

Abstraction

Advantages of Abstraction

- Abstraction makes **complex systems easier to understand** by hiding the implementation details.
- Abstraction keeps different part of the system separated.
- Abstraction **maintains code more efficiently**.
- Abstraction **increases the security** by only showing the necessary details to the user

Abstraction

Disadvantages of Abstraction

- **It can add unnecessary complexity if overused.**
- **May reduce flexibility in implementation.**
- **Makes debugging and understanding the system harder for unfamiliar users.**
- **Overhead from abstraction layers can affect performance.**

Encapsulation

- **Encapsulation**

- **is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.**
- **In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.**
- **it is also known as data hiding.**

Encapsulation

- **To achieve encapsulation in Java**
 - **Declare the variables of a class as private.**
 - **Provide public setter and getter methods to modify and view the variables values**
 - The public `setXXX()` and `getXXX()` methods are the access points of the instance variables

Encapsulation: Example

```
public class Example {  
    private String name;  
    private String idNum;  
    private int age;  
  
    public int getAge() {  
        return age;  
    }  
    public String getName() {  
        return name;  
    }  
    public String getIdNum() {  
        return idNum;  
    }  
}
```


Encapsulation

```
public void setAge( int newAge) {  
    age = newAge;  
}
```

```
public void setName(String newName) {  
    name = newName;  
}
```

```
public void setIdNum( String newId) {  
    idNum = newId;  
}
```

Encapsulation

```
public static void main(String args[]) {  
    Example e = new Example();  
    e.setName("James");  
    e.setAge(20);  
    e.setIdNum("12343ms");  
  
    System.out.print("Name : " + e.getName() );  
    System.out.print("Age : " + e.getAge());  
    System.out.print("IdNum : " + e.getIdNum());  
}  
}
```

Encapsulation

- **Benefits of Encapsulation**
 - **The fields of a class can be made read-only or write-only.**
 - **A class can have total control over what is stored in its fields**

Thank You

