**Usman Institute of Technology**

**Department of Computer Science**

**Course Code: SE312**

**Course Title: Software Construction and Development**

**SPRING 2024**

**Lab 06**

**Objective: To acquire knowledge of the fundamental principles of OOP such as inheritance, polymorphism, abstraction and encapsulation.**

**Student Information**

|  |  |
| --- | --- |
| Student Name | Syed Muhammad Zaid |
| Student ID | 20B-052-SE |
| Date | 4/27/2024 |

**Assessment**

|  |  |
| --- | --- |
| Marks Obtained |  |
| Remarks |  |
| Signature |  |

**LAB #06**

**PILLARS OF OBJECT-ORIENTED PROGRAMMING**

**Object-Oriented Programming (OOP)** is an approach to **software development** that uses objects, or data structures consisting of data fields and methods together with their interactions, to design applications and computer programs.

The four pillars of OOPs allow developers to better manage software complexity by organizing code into objects and allowing them to interact in a structured way.

* **Inheritance** is the ability to create a new class (child class) from an existing one (parent class). The child class typically inherits the attributes (members and methods) of the parent class, although it can also redefine them.
* **Data abstraction** is the process of hiding unnecessary details of an object’s internal structure. By abstracting an object’s data, its structure and behavior can be kept separate and more easily understood.
* **Encapsulation** is the process of wrapping data and related functions into a single unit (object). Encapsulation limits access to object data and methods, preventing their misuse and ensuring their proper functioning. This allows developers to hide the data from other parts of the program, making the code more secure and reliable.
* **Polymorphism** is the ability of an object to take on multiple forms. This allows objects of different classes to be used interchangeably, as long as they implement a certain interface (have methods of the same name).

**EXAMPLES**

Some suggestions on how **inheritance** can be implemented in real-life scenarios:

**Employee System:**

In a company's HR system, you can have a superclass Employee representing common attributes like name, ID, and salary.

Subclasses such as Manager, Engineer, and SalesPerson can inherit from Employee, each with specialized attributes and methods relevant to their role.

**Education Management System:**

Design a superclass Person with attributes like name, age, and address.

Subclasses like Student and Teacher can inherit from Person, with additional attributes such as grade, GPA, and subjects taught.

**Banking System:**

Have a superclass Account with attributes like account number, balance, and account type.

Subclasses like SavingsAccount, CheckingAccount, and LoanAccount can inherit from Account, each with specific methods for deposit, withdrawal, and interest calculation.

**E-commerce System:**

Design a superclass Product with attributes like name, price, and description.

Subclasses like Electronics, Clothing, and Books can inherit from Product, with additional attributes specific to each category, such as screen size, fabric type, and author.

Some suggestions on how **abstraction** can be implemented in real-life scenarios:

**Education Management System:**

Abstraction can be applied to manage course materials and assignments. Students access learning resources, submit assignments, and view grades through user-friendly portals without needing to comprehend the complexities of curriculum planning, grading algorithms, or academic regulations.

**Banking System:**

Abstraction can be applied to represent bank accounts. The user interacts with simplified interfaces such as deposit, withdraw, and check balance without needing to understand the intricate details of transaction processing, interest calculation, or database operations.

**E-commerce Platform:**

Abstraction can be used to manage product listings. Customers interact with a user-friendly interface to browse, search, and purchase products without needing to know the underlying complexities of inventory management, payment processing, or order fulfillment.

Some suggestions on how encapsulation can be implemented in real-life scenarios:

**Banking System:**

Encapsulation can be applied to represent bank accounts. The account balance, account holder's information, and transaction history are encapsulated within the account object. Methods such as deposit, withdraw, and check balance ensure that these attributes are accessed and modified securely.

**Healthcare System:**

Encapsulation can be used to manage patient records. Patient information such as medical history, diagnoses, and treatment plans are encapsulated within the patient object. Access to sensitive medical data is restricted to authorized healthcare professionals through secure methods.

**E-commerce Platform:**

Encapsulation can be applied to manage user accounts and orders. User information, order details, and payment information are encapsulated within user and order objects. Secure methods ensure that only authenticated users can access and modify their own data.

Some suggestions on how **polymorphism** can be implemented in real-life scenarios:

**Banking System Transactions:**

In a banking system, different types of transactions (e.g., deposit, withdrawal, transfer) can be represented by subclasses of a Transaction superclass.

Polymorphism allows methods such as processTransaction() to be implemented differently for each transaction type, considering factors such as account balances and transaction limits.

**Employee Payroll System:**

In an employee payroll system, different types of employees (e.g., full-time, part-time, contractor) can be represented by subclasses of an Employee superclass.

Polymorphism allows payroll calculation methods to be implemented differently for each employee type, accommodating diverse compensation structures and benefit packages.

**Personnel Management:**

In a School Management System, various personnel such as teachers, administrators, and support staff can be represented by subclasses of a common Person superclass.

Polymorphism allows methods such as calculateSalary() to be implemented differently for each personnel type, considering factors such as role, experience, and qualifications.

**CODE EXAMPLES**

* EXAMPLE FOR POLYMORPHISM

class Polygon {

public void render() {

System.out.println("Rendering Polygon...");

}

}

class Square extends Polygon {

public void render() {

System.out.println("Rendering Square...");

}

}

class Circle extends Polygon {

public void render() {

System.out.println("Rendering Circle...");

}

}

class Main {

public static void main(String[] args) {

Square s1 = new Square();

s1.render();

// create an object of Circle

Circle c1 = new Circle();

c1.render();

}

}

* EXAMPLE FOR ENCAPSULATION

class Area {

int length;

int breadth;

// constructor to initialize values

Area(int length, int breadth) {

this.length = length;

this.breadth = breadth;

}

public void getArea() {

int area = length \* breadth;

System.out.println("Area: " + area);

}

}

class Main {

public static void main(String[] args) {

// create object of Area

// pass value of length and breadth

Area rectangle = new Area(5, 6);

rectangle.getArea();

}

}

* EXAMPLE FOR INHERITANCE

class Employee{

 float salary=40000;

}

class Programmer extends Employee{

 int bonus=10000;

 public static void main(String args[]){

   Programmer p=new Programmer();

   System.out.println("Programmer salary is:"+p.salary);

   System.out.println("Bonus of Programmer is:"+p.bonus);

}

}

* EXAMPLE FOR ABSTRACTION

abstract class Animal {

// Abstract method (does not have a body)

public abstract void animalSound();

// Regular method

public void sleep() {

System.out.println("Zzz");

}

}

class Cat extends Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The cat says: meu meu");

}

}

class Main {

public static void main(String[] args) {

Cat myCat = new Cat(); // Create a Cat object

myCat.animalSound();

myCat.sleep();

}

}

**TASKS**

1. Define a BankAccount class with private attributes such as account number, balance, and owner's name. Implement public methods like deposit() and withdraw() to ensure proper encapsulation of account data. Create an AccountHolder class with private attributes for username, password, and email. Implement methods for user authentication and password reset while keeping sensitive data encapsulated.
2. Write a Java program to create a class called "Employee" with a name, job title, and salary attributes, and methods to calculate and update salary.
3. Write a Java program to create a class called "Movie" with attributes for title, director, actors, and reviews, and methods for adding and retrieving reviews.

**Home Task**

Write a Java program to create class called “TrafficLight” with attributes for color and duration, and methods to change the color and check for red or green.

**How to Submit**

* Submit lab work in a single pdf/docx on MS Team.
* Submit the work as per format given in this manual (No other format will be accepted).
* Lab work (Exercises) file name should be saved with your roll number and course code (e.g. 21B-001-SE\_SExxx\_LWxx.pd f where SExxx is course code and LWxx is Lab number).