Class Activity4

CTU – Training Solutions
UID – User Interface Development
Software Development
SWD412
Week 4

Complete the following Class activity Java Exercises:

1. Polymorphism

- Exercise: Create a base class Shape with a method area().
- Then create two subclasses Rectangle and Circle that override the area() method to calculate the area for each shape.
- Write a main method to demonstrate polymorphism by calling the area() method on an array of Shape references.

```
public class Polymorphism {

    // Base Shape class
    static class Shape {
        // Method to calculate area (to be overridden by subclasses)
        public double area() {
            return 0; // Default implementation (could also be abstract)
        }
    }

    // Subclass Rectangle that extends Shape
    static class Rectangle extends Shape {
        private double length;
        private double width;

        // Constructor
        public Rectangle(double length, double width) {
            this.length = length;
            this.width = width;
        }

        // Override the area() method to calculate the area of a rectangle
```

```java
    @Override
    public double area() {
        return Math.PI * radius * radius;
    }
}

// Main method to demonstrate polymorphism
public static void main(String[] args) {
    // Creates an array of Shape references
    Shape[] shapes = new Shape[2];

    // Initialize the array with Rectangle and Circle objects
    shapes[0] = new Rectangle( length: 5,  width: 7);
    shapes[1] = new Circle( radius: 3);

    // Loop through the array and call the area() method on each shape in the program
    for (Shape shape : shapes) {
        System.out.println("Area: " + shape.area());
    }
}
}
```

```
Run:
Area: 35.0
Area: 28.274333882308138
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Inheritance

- Exercise: Create a base class Vehicle with properties like make and model, and a method displayInfo().
- Then create a subclass Car that inherits from Vehicle and adds a property numberOfDoors. Override the displayInfo() method in Car to include the number of doors.
- Write a main method to demonstrate inheritance.

```java
/*
 * @author Dian
 */
public class Vehicle {
    // Properties of Vehicle
    String make;
    String model;

    // Constructor for Vehicle
    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }

    // Method to display Vehicle information
    public void displayInfo() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
    }
}
```

```java
 * @author Dian
 */
// Subclass Car that inherits from Vehicle
public class Car extends Vehicle {
    // Property specific to Car
    int numberOfDoors;

    // Constructor for Car
    public Car(String make, String model, int numberOfDoors) {
        // Call the constructor of the superclass Vehicle
        super(make, model);
        this.numberOfDoors = numberOfDoors;
    }

    // Override the displayInfo() method to include numberOfDoors
    @Override
    public void displayInfo() {
        // Call the superclass's displayInfo() method
        super.displayInfo();
        System.out.println("Number of Doors: " + numberOfDoors);
    }
}
// Main class to demonstrate inheritance
public class Inheritance {
    public static void main(String[] args) {
        // Create an instance of Vehicle
        Vehicle vehicle = new Vehicle( make: "Toyota",  model: "Camry");
        System.out.println( x: "Vehicle Information:");
        vehicle.displayInfo();

        // Create an instance of Car
        Car car = new Car( make: "Honda",  model: "Civic",  numberOfDoors: 4);
        System.out.println( x: "\nCar Information:");
        car.displayInfo();
    }
}
```

```
Vehicle Information:
Make: Toyota
Model: Camry

Car Information:
Make: Honda
Model: Civic
Number of Doors: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Encapsulation

- Exercise: Create a class BankAccount with private fields for accountNumber and balance.
- Provide public methods to deposit and withdraw funds, as well as a method to check the balance. Write a main method to demonstrate encapsulation.

```java
public class Bankaccount {
    // Private fields
    private String accountNumber;
    private double balance;

    // Constructor to initialize accountNumber and balance
    public Bankaccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    // Public method to deposit funds
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println( x: "Deposit amount must be positive.");
        }
    }
```

```java
    // Public method to withdraw funds
    public void withdraw(double amount) {
        if (amount > 0) {
            if (amount <= balance) {
                balance -= amount;
                System.out.println("Withdrew: $" + amount);
            } else {
                System.out.println( x: "Insufficient funds.");
            }
        } else {
            System.out.println( x: "Withdrawal amount must be positive.");
        }
    }

    // Public method to check the balance
    public double getBalance() {
        return balance;
    }

    // Public method to get the account number (if needed)
    public String getAccountNumber() {
        return accountNumber;
    }
```

```
    // Main method to demonstrate encapsulation
    public static void main(String[] args) {
        // Create a BankAccount object
        Bankaccount account = new Bankaccount( accountNumber: "123456789",  initialBalance: 1000.00);

        // Display initial balance
        System.out.println("Initial Balance: $" + account.getBalance());

        // Deposit funds
        account.deposit( amount: 500.00);

        // Withdraw funds
        account.withdraw( amount: 200.00);

        // Try to withdraw more than the balance
        account.withdraw( amount: 2000.00);

        // Check final balance
        System.out.println("Final Balance: $" + account.getBalance());
    }
}
```

4. Constructors and Methods

- Exercise: Create a class Book with properties for title, author, and price.
- Include a constructor to initialize these properties and methods to display book details and apply a discount.
- Write a main method to demonstrate the usage of constructors and methods.

```
public class Books {
    private String title;
    private String author;
    private double price;

    // Constructor to initialize properties
    public Books(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // Method to display book details
    public void displayDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
    }
```

```java
// Method to apply a discount
public void applyDiscount(double discountPercentage) {
    if (discountPercentage > 0 && discountPercentage <= 100) {
        double discountAmount = price * (discountPercentage / 100);
        price -= discountAmount;
        System.out.println("Discount of " + discountPercentage + "% applied.");
    } else {
        System.out.println( x: "Invalid discount percentage.");
    }
}

// Main method to demonstrate usage
public static void main(String[] args) {
    // Create a Book object
    Books book = new Books( title: "The Great Gatsby",  author: "F. Scott Fitzgerald",  price: 15.99);

    // Display book details
    System.out.println( x: "Book Details:");
    book.displayDetails();

    // Apply a discount
    book.applyDiscount( discountPercentage: 10);

    // Display book details after discount
    System.out.println( x: "\nBook Details After Discount:");
    book.displayDetails();
}
```