**Exercise 1: Creating an Abstract Class**

1. Create an abstract class named **Shape**.
2. Define an abstract method **calculateArea()** in the **Shape** class.
3. Create two subclasses, **Circle** and **Rectangle**, that inherit from the **Shape** class.
4. Implement the **calculateArea()** method in both the **Circle** and **Rectangle** classes to calculate the area of a circle and a rectangle, respectively.
5. Test your classes by creating instances of both **Circle** and **Rectangle** and calling the **calculateArea()** method on each.

abstract class Shape {

public abstract double calculateArea();

}

class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

class Rectangle extends Shape {

private double width;

private double height;

public Rectangle(double width, double height) {

this.width = width;

this.height = height;

}

@Override

public double calculateArea() {

return width \* height;

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle(5.0);

Rectangle rectangle = new Rectangle(4.0, 6.0);

System.out.println("Circle Area: " + circle.calculateArea());

System.out.println("Rectangle Area: " + rectangle.calculateArea());

}

}

**Exercise 2: Abstract Class Inheritance**

1. Create an abstract class **Animal** with an abstract method **makeSound()**.
2. Create two subclasses, **Dog** and **Cat**, that inherit from the **Animal** class.
3. Implement the **makeSound()** method in both the **Dog** and **Cat** classes to make appropriate sounds for each.
4. Create instances of both **Dog** and **Cat** and call the **makeSound()** method on each.

abstract class Animal {

public abstract void makeSound();

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Woof!");

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Meow!");

}

}

public class Main {

public static void main(String[] args) {

Animal dog = new Dog();

Animal cat = new Cat();

dog.makeSound();

cat.makeSound();

}

}

**Exercise 3: Banking System with Abstract Class**

Create a simple banking system using abstract classes. The system should have the following components:

1. An abstract class **Account** with the following properties and methods:
   * Properties: **accountNumber**, **accountHolder**, and **balance**.
   * Abstract methods: **withdraw(double amount)** and **deposit(double amount)**.
   * A constructor to initialize the account number, account holder's name, and initial balance.
   * A method **getAccountInfo()** to display account information.
2. Two subclasses of **Account**: **SavingsAccount** and **CheckingAccount**. Implement the **withdraw** and **deposit** methods in both subclasses. For example, in a savings account, there might be a minimum balance requirement, and a checking account might not have such a requirement.
3. Create instances of both a **SavingsAccount** and a **CheckingAccount**. Perform some deposit and withdrawal operations and display the account information.

public abstract class Account {

private int accountNumber;

private String accountHolder;

protected double balance;

public Account(int accountNumber, String accountHolder, double initialBalance) {

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = initialBalance;

}

public abstract void withdraw(double amount);

public abstract void deposit(double amount);

public void getAccountInfo() {

System.out.println("Account Number: " + accountNumber);

System.out.println("Account Holder: " + accountHolder);

System.out.println("Balance: $" + balance);

}

}

public class SavingsAccount extends Account {

private double minimumBalance;

public SavingsAccount(int accountNumber, String accountHolder, double initialBalance, double minimumBalance) {

super(accountNumber, accountHolder, initialBalance);

this.minimumBalance = minimumBalance;

}

@Override

public void withdraw(double amount) {

if (balance - amount >= minimumBalance) {

balance -= amount;

} else {

System.out.println("Insufficient balance for withdrawal.");

}

}

@Override

public void deposit(double amount) {

balance += amount;

}

}

public class CheckingAccount extends Account {

public CheckingAccount(int accountNumber, String accountHolder, double initialBalance) {

super(accountNumber, accountHolder, initialBalance);

}

@Override

public void withdraw(double amount) {

if (balance - amount >= 0) {

balance -= amount;

} else {

System.out.println("Insufficient balance for withdrawal.");

}

}

@Override

public void deposit(double amount) {

balance += amount;

}

}

public class Main {

public static void main(String[] args) {

SavingsAccount savingsAccount = new SavingsAccount(1001, "Alice", 1000.0, 500.0);

CheckingAccount checkingAccount = new CheckingAccount(2001, "Bob", 2000.0);

savingsAccount.deposit(500.0);

checkingAccount.deposit(100.0);

savingsAccount.withdraw(200.0);

checkingAccount.withdraw(1500.0);

System.out.println("Savings Account Information:");

savingsAccount.getAccountInfo();

System.out.println("\nChecking Account Information:");

checkingAccount.getAccountInfo();

}

}

**Exercise 4: Abstract Shape Hierarchy**

Create an abstract shape hierarchy consisting of **Shape**, **Circle**, and **Square** classes. The classes should have the following properties and methods:

1. **Shape** (abstract class):
   * An abstract method **calculateArea()**.
   * An abstract method **calculatePerimeter()**.
   * A **name** property to store the shape's name.
   * A constructor to initialize the **name**.
2. **Circle** (subclass of **Shape**):
   * A **radius** property.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for a circle.
3. **Square** (subclass of **Shape**):
   * A **sideLength** property.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for a square.
4. Create instances of both a **Circle** and a **Square**. Calculate and display their areas and perimeters.

abstract class Shape {

protected String name;

public Shape(String name) {

this.name = name;

}

public abstract double calculateArea();

public abstract double calculatePerimeter();

}

class Circle extends Shape {

private double radius;

public Circle(String name, double radius) {

super(name);

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

@Override

public double calculatePerimeter() {

return 2 \* Math.PI \* radius;

}

}

class Square extends Shape {

private double sideLength;

public Square(String name, double sideLength) {

super(name);

this.sideLength = sideLength;

}

@Override

public double calculateArea() {

return sideLength \* sideLength;

}

@Override

public double calculatePerimeter() {

return 4 \* sideLength;

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle("Circle", 5.0);

Square square = new Square("Square", 4.0);

System.out.println(circle.name + " Area: " + circle.calculateArea());

System.out.println(circle.name + " Perimeter: " + circle.calculatePerimeter());

System.out.println(square.name + " Area: " + square.calculateArea());

System.out.println(square.name + " Perimeter: " + square.calculatePerimeter());

}

}

**Exercise 5: Abstract Employee Hierarchy**

Create an abstract employee hierarchy consisting of an abstract class **Employee** with two subclasses, **HourlyEmployee** and **SalariedEmployee**. The classes should have the following properties and methods:

1. **Employee** (abstract class):
   * Properties: **employeeID**, **firstName**, **lastName**.
   * An abstract method **calculateEarnings()** to calculate the employee's earnings.
   * A constructor to initialize the employee's information.
2. **HourlyEmployee** (subclass of **Employee**):
   * Additional property: **hourlyRate**.
   * Implement the **calculateEarnings()** method to calculate earnings for hourly employees.
3. **SalariedEmployee** (subclass of **Employee**):
   * Additional property: **monthlySalary**.
   * Implement the **calculateEarnings()** method to calculate earnings for salaried employees.
4. Create instances of both an **HourlyEmployee** and a **SalariedEmployee**. Calculate and display their earnings.

abstract class Employee {

protected int employeeID;

protected String firstName;

protected String lastName;

public Employee(int employeeID, String firstName, String lastName) {

this.employeeID = employeeID;

this.firstName = firstName;

this.lastName = lastName;

}

public abstract double calculateEarnings();

}

class HourlyEmployee extends Employee {

private double hourlyRate;

private int hoursWorked;

public HourlyEmployee(int employeeID, String firstName, String lastName, double hourlyRate, int hoursWorked) {

super(employeeID, firstName, lastName);

this.hourlyRate = hourlyRate;

this.hoursWorked = hoursWorked;

}

@Override

public double calculateEarnings() {

return hourlyRate \* hoursWorked;

}

}

class SalariedEmployee extends Employee {

private double monthlySalary;

public SalariedEmployee(int employeeID, String firstName, String lastName, double monthlySalary) {

super(employeeID, firstName, lastName);

this.monthlySalary = monthlySalary;

}

@Override

public double calculateEarnings() {

return monthlySalary;

}

}

public class Main {

public static void main(String[] args) {

HourlyEmployee hourlyEmployee = new HourlyEmployee(101, "John", "Doe", 15.0, 40);

SalariedEmployee salariedEmployee = new SalariedEmployee(102, "Jane", "Smith", 3000.0);

System.out.println("Hourly Employee Earnings: $" + hourlyEmployee.calculateEarnings());

System.out.println("Salaried Employee Earnings: $" + salariedEmployee.calculateEarnings());

}

}

**Exercise 6: Abstract Geometric Shapes**

Create an abstract class **Shape** with two subclasses, **Circle** and **Triangle**. The classes should have the following properties and methods:

1. **Shape** (abstract class):
   * An abstract method **calculateArea()** to calculate the area of the shape.
   * An abstract method **calculatePerimeter()** to calculate the perimeter of the shape.
   * A constructor to initialize the name of the shape.
2. **Circle** (subclass of **Shape**):
   * Additional property: **radius**.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for circles.
3. **Triangle** (subclass of **Shape**):
   * Additional properties: **base** and **height**.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for triangles.
4. Create instances of both a **Circle** and a **Triangle**. Calculate and display their areas and perimeters.

abstract class Shape {

protected String name;

public Shape(String name) {

this.name = name;

}

public abstract double calculateArea();

public abstract double calculatePerimeter();

}

class Circle extends Shape {

private double radius;

public Circle(String name, double radius) {

super(name);

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

@Override

public double calculatePerimeter() {

return 2 \* Math.PI \* radius;

}

}

class Triangle extends Shape {

private double base;

private double height;

private double sideA;

private double sideB;

private double sideC;

public Triangle(String name, double base, double height, double sideA, double sideB, double sideC) {

super(name);

this.base = base;

this.height = height;

this.sideA = sideA;

this.sideB = sideB;

this.sideC = sideC;

}

@Override

public double calculateArea() {

return 0.5 \* base \* height;

}

@Override

public double calculatePerimeter() {

return sideA + sideB + sideC;

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle("Circle", 5.0);

Triangle triangle = new Triangle("Triangle", 4.0, 6.0, 3.0, 4.0, 5.0);

System.out.println(circle.name + " Area: " + circle.calculateArea());

System.out.println(circle.name + " Perimeter: " + circle.calculatePerimeter());

System.out.println(triangle.name + " Area: " + triangle.calculateArea());

System.out.println(triangle.name + " Perimeter: " + triangle.calculatePerimeter());

}

}

**Exercise 7: Abstract Media Players**

Create an abstract class **MediaPlayer** with two subclasses, **AudioPlayer** and **VideoPlayer**. The classes should have the following properties and methods:

1. **MediaPlayer** (abstract class):
   * Properties: **mediaName** and **volume**.
   * An abstract method **play()** to play media.
   * An abstract method **stop()** to stop media playback.
   * A constructor to initialize the media name and volume.
2. **AudioPlayer** (subclass of **MediaPlayer**):
   * Implement the **play()** method to play audio.
   * Implement the **stop()** method to stop audio playback.
3. **VideoPlayer** (subclass of **MediaPlayer**):
   * Implement the **play()** method to play video.
   * Implement the **stop()** method to stop video playback.
4. Create instances of both an **AudioPlayer** and a **VideoPlayer**. Play and stop media, and adjust the volume.

abstract class MediaPlayer {

protected String mediaName;

protected int volume;

public MediaPlayer(String mediaName, int volume) {

this.mediaName = mediaName;

this.volume = volume;

}

public abstract void play();

public abstract void stop();

public void setVolume(int volume) {

this.volume = volume;

}

}

class AudioPlayer extends MediaPlayer {

public AudioPlayer(String mediaName, int volume) {

super(mediaName, volume);

}

@Override

public void play() {

System.out.println("Playing audio: " + mediaName);

}

@Override

public void stop() {

System.out.println("Stopping audio: " + mediaName);

}

}

class VideoPlayer extends MediaPlayer {

public VideoPlayer(String mediaName, int volume) {

super(mediaName, volume);

}

@Override

public void play() {

System.out.println("Playing video: " + mediaName);

}

@Override

public void stop() {

System.out.println("Stopping video: " + mediaName);

}

}

public class Main {

public static void main(String[] args) {

AudioPlayer audioPlayer = new AudioPlayer("Song.mp3", 50);

VideoPlayer videoPlayer = new VideoPlayer("Movie.mp4", 70);

audioPlayer.play();

videoPlayer.play();

audioPlayer.setVolume(60);

videoPlayer.setVolume(80);

audioPlayer.stop();

videoPlayer.stop();

}

}

**Exercise 8: Abstract School Courses**

Create an abstract class **Course** with two subclasses, **MathCourse** and **ScienceCourse**. The classes should have the following properties and methods:

1. **Course** (abstract class):
   * Properties: **courseName**, **courseCode**, and **instructorName**.
   * An abstract method **calculateGrade()** to calculate a student's grade.
   * A constructor to initialize the course name, course code, and instructor name.
2. **MathCourse** (subclass of **Course**):
   * Additional property: **numberOfAssignments**.
   * Implement the **calculateGrade()** method for math courses.
3. **ScienceCourse** (subclass of **Course**):
   * Additional property: **numberOfExperiments**.
   * Implement the **calculateGrade()** method for science courses.
4. Create instances of both a **MathCourse** and a **ScienceCourse**. Calculate and display a student's grade.

abstract class Course {

protected String courseName;

protected String courseCode;

protected String instructorName;

public Course(String courseName, String courseCode, String instructorName) {

this.courseName = courseName;

this.courseCode = courseCode;

this.instructorName = instructorName;

}

public abstract double calculateGrade();

}

class MathCourse extends Course {

private int numberOfAssignments;

public MathCourse(String courseName, String courseCode, String instructorName, int numberOfAssignments) {

super(courseName, courseCode, instructorName);

this.numberOfAssignments = numberOfAssignments;

}

@Override

public double calculateGrade() {

// Implement the grading calculation for a math course.

// This is just a placeholder, you can replace it with your own logic.

return 90.0;

}

}

class ScienceCourse extends Course {

private int numberOfExperiments;

public ScienceCourse(String courseName, String courseCode, String instructorName, int numberOfExperiments) {

super(courseName, courseCode, instructorName);

this.numberOfExperiments = numberOfExperiments;

}

@Override

public double calculateGrade() {

// Implement the grading calculation for a science course.

// This is just a placeholder, you can replace it with your own logic.

return 85.0;

}

}

public class Main {

public static void main(String[] args) {

MathCourse mathCourse = new MathCourse("Math 101", "MATH101", "Dr. Smith", 5);

ScienceCourse scienceCourse = new ScienceCourse("Physics 101", "PHYS101", "Prof. Johnson", 3);

double mathGrade = mathCourse.calculateGrade();

double scienceGrade = scienceCourse.calculateGrade();

System.out.println("Math Course Grade: " + mathGrade);

System.out.println("Science Course Grade: " + scienceGrade);

}

}

**Exercise 9: Abstract Vehicle Hierarchy**

Create an abstract class **Vehicle** with two subclasses, **Car** and **Motorcycle**. The classes should have the following properties and methods:

1. **Vehicle** (abstract class):
   * Properties: **make**, **model**, and **year**.
   * An abstract method **start()**.
   * An abstract method **stop()**.
   * A constructor to initialize the make, model, and year.
2. **Car** (subclass of **Vehicle**):
   * Implement the **start()** method to start the car's engine.
   * Implement the **stop()** method to stop the car's engine.
3. **Motorcycle** (subclass of **Vehicle**):
   * Implement the **start()** method to start the motorcycle's engine.
   * Implement the **stop()** method to stop the motorcycle's engine.
4. Create instances of both a **Car** and a **Motorcycle**. Start and stop their engines.

abstract class Vehicle {

protected String make;

protected String model;

protected int year;

public Vehicle(String make, String model, int year) {

this.make = make;

this.model = model;

this.year = year;

}

public abstract void start();

public abstract void stop();

}

class Car extends Vehicle {

public Car(String make, String model, int year) {

super(make, model, year);

}

@Override

public void start() {

System.out.println("Starting the car's engine.");

}

@Override

public void stop() {

System.out.println("Stopping the car's engine.");

}

}

class Motorcycle extends Vehicle {

public Motorcycle(String make, String model, int year) {

super(make, model, year);

}

@Override

public void start() {

System.out.println("Starting the motorcycle's engine.");

}

@Override

public void stop() {

System.out.println("Stopping the motorcycle's engine.");

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car("Toyota", "Camry", 2022);

Motorcycle motorcycle = new Motorcycle("Harley-Davidson", "Sportster", 2023);

car.start();

car.stop();

motorcycle.start();

motorcycle.stop();

}

}

**Exercise 10: Abstract Payment System**

Create an abstract payment system with an abstract class **Payment** and two subclasses, **CreditCardPayment** and **PaypalPayment**. The classes should have the following properties and methods:

1. **Payment** (abstract class):
   * Properties: **amount**, **recipient**, and **paymentDate**.
   * An abstract method **processPayment()** to process the payment.
   * A constructor to initialize the payment details.
2. **CreditCardPayment** (subclass of **Payment**):
   * Additional properties: **cardNumber**, **cardHolder**, and **expirationDate**.
   * Implement the **processPayment()** method to process credit card payments.
3. **PaypalPayment** (subclass of **Payment**):
   * Additional properties: **email** and **password**.
   * Implement the **processPayment()** method to process PayPal payments.
4. Create instances of both a **CreditCardPayment** and a **PaypalPayment**. Process payments using both methods.

import java.text.SimpleDateFormat;

import java.util.Date;

abstract class Payment {

private double amount;

private String recipient;

private Date paymentDate;

public Payment(double amount, String recipient) {

this.amount = amount;

this.recipient = recipient;

this.paymentDate = new Date();

}

public abstract void processPayment();

public double getAmount() {

return amount;

}

public String getRecipient() {

return recipient;

}

public String getPaymentDate() {

SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");

return dateFormat.format(paymentDate);

}

}

class CreditCardPayment extends Payment {

private String cardNumber;

private String cardHolder;

private String expirationDate;

public CreditCardPayment(double amount, String recipient, String cardNumber, String cardHolder, String expirationDate) {

super(amount, recipient);

this.cardNumber = cardNumber;

this.cardHolder = cardHolder;

this.expirationDate = expirationDate;

}

@Override

public void processPayment() {

System.out.println("Processing credit card payment of $" + getAmount() + " to " + getRecipient() +

" on " + getPaymentDate() + " using card ending in " + cardNumber.substring(cardNumber.length() - 4));

}

}

class PaypalPayment extends Payment {

private String email;

private String password;

public PaypalPayment(double amount, String recipient, String email, String password) {

super(amount, recipient);

this.email = email;

this.password = password;

}

@Override

public void processPayment() {

System.out.println("Processing PayPal payment of $" + getAmount() + " to " + getRecipient() +

" on " + getPaymentDate() + " using email: " + email);

}

}

public class PaymentSystem {

public static void main(String[] args) {

CreditCardPayment creditCardPayment = new CreditCardPayment(100.0, "Online Store", "1234-5678-9012-3456", "John Doe", "12/25");

PaypalPayment paypalPayment = new PaypalPayment(50.0, "Subscription Service", "johndoe@example.com", "password123");

creditCardPayment.processPayment();

paypalPayment.processPayment();

}

}