**Exercise 1:**

**Create an abstract class Vehicle with abstract methods startEngine() and stopEngine(). Create two subclasses, Car and Motorcycle, that inherit from Vehicle. Implement the methods in each subclass to print out appropriate messages**.

Solution:

**/\***

**\* Create an abstract class Vehicle with abstract methods**

**\* startEngine() and stopEngine().**

**\***

**\* Create two subclasses, Car and Motorcycle, that inherit from Vehicle.**

**\***

**\* Implement the methods in each subclass to print out appropriate messages.**

**\***

**\* Sedans:**

**Model ID: SD-001**

**Model Name: Acclaim Deluxe**

**SUVs:**

**Model ID: SUV-002**

**Model Name: Expedition Xplorer**

**Trucks:**

**Model ID: TRK-003**

**Model Name: Titan Hauler**

**Sports Cars:**

**Model ID: SC-004**

**Model Name: Velocity Viper**

**Electric Cars:**

**Model ID: EC-005**

**Model Name: Electron Evolve**

**Motorcycles:**

**Model ID: MC-006**

**Model Name: Blaze Raptor**

**Vans:**

**Model ID: VN-007**

**Model Name: Transit Voyager**

**Crossover Vehicles:**

**Model ID: CV-008**

**Model Name: Crosswind Cruiser**

**Luxury Cars:**

**Model ID: LC-009**

**Model Name: Elegance Elite**

**Hybrid Cars:**

**Model ID: HC-010**

**Model Name: EcoHarmony Hybrid**

**Answer:**

**Vehicle.java**

**package com.exercise.bll;**

**public abstract class Vehicle {**

**private String modelId;**

**private String nameOfTheModel;**

**private String typeOfVehicle;**

**public Vehicle() {**

**}**

**public Vehicle(String modelId, String nameOfTheModel, String typeOfVehicle) {**

**super();**

**this.modelId = modelId;**

**this.nameOfTheModel = nameOfTheModel;**

**this.typeOfVehicle = typeOfVehicle;**

**}**

**public String getModelId() {**

**return modelId;**

**}**

**public void setModelId(String modelId) {**

**this.modelId = modelId;**

**}**

**public String getNameOfTheModel() {**

**return nameOfTheModel;**

**}**

**public void setNameOfTheModel(String nameOfTheModel) {**

**this.nameOfTheModel = nameOfTheModel;**

**}**

**public String getTypeOfVehicle() {**

**return typeOfVehicle;**

**}**

**public void setTypeOfVehicle(String typeOfVehicle) {**

**this.typeOfVehicle = typeOfVehicle;**

**}**

**public abstract void startEngine();**

**public abstract void stopEngine();**

**@Override**

**public String toString() {**

**return "Vehicle [modelId=" + modelId + ", nameOfTheModel=" + nameOfTheModel + ", typeOfVehicle=" + typeOfVehicle**

**+ "]";**

**}**

**}**

**Car.java**

**package** com.exercise.bll;

**public** **class** Car **extends** Vehicle {

**public** Car() {

}

**public** Car(String modelId, String nameOfTheModel, String typeOfVehicle) {

**super**(modelId, nameOfTheModel, typeOfVehicle);

}

@Override

**public** **void** startEngine() {

System.***out***.println("The model id of the car :" + getModelId());

System.***out***.println("The model name of the car: " + getNameOfTheModel());

System.***out***.println("The type of the car is: " + getTypeOfVehicle());

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("The car engine is started.\n");

System.***out***.println("===================================");

}

@Override

**public** **void** stopEngine() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("The car engine is stopped.");

}

}

**Motorcycle.java**

**package** com.exercise.bll;

**public** **class** Motorcycle **extends** Vehicle {

**public** Motorcycle() {

}

**public** Motorcycle(String modelId, String nameOfTheModel, String typeOfVehicle) {

**super**(modelId, nameOfTheModel, typeOfVehicle);

}

@Override

**public** **void** startEngine() {

System.***out***.println("The model id of the motorcycle :" + getModelId());

System.***out***.println("The model name of the motorcycle: " + getNameOfTheModel());

System.***out***.println("The type of the motorcycle is: " + getTypeOfVehicle());

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("The motorcycle engine is started.");

}

@Override

**public** **void** stopEngine() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("The motorcycle engine is stopped.\n");

System.***out***.println("==================================");

}

}

**TestVehicle.java**

**package** com.exercise.pll;

**import** com.exercise.bll.Car;

**import** com.exercise.bll.Motorcycle;

**import** com.exercise.bll.Vehicle;

**public** **class** TestVehicle {

**public** **static** **void** main(String[] args) {

Vehicle []vehicles = **new** Vehicle[10];

vehicles[0] = **new** Car("SD-001", "Acclaim Deluxe", "Sedans");

vehicles[1] = **new** Car("SUV-002", "Expedition Xplorer", "SUVs");

vehicles[2] = **new** Car("TRK-003", "Titan Hauler", "Trucks");

vehicles[3] = **new** Car("SC-004", "Velocity Viper", "Sports Cars");

vehicles[4] = **new** Car("EC-005", "Electron Evolve", "Electric Cars");

vehicles[5] = **new** Motorcycle("MC-101", "Thunderbolt Titan", "Cruiser");

vehicles[6] = **new** Motorcycle("MC-202", "Vortex Viper", "Sportbike");

vehicles[7] = **new** Motorcycle("MC-303", "Shadow Sabre", "Cruiser");

vehicles[8] = **new** Motorcycle("MC-404", "Blaze Breaker", "Off-Road/Adventure");

vehicles[9] = **new** Motorcycle("MC-505", "Phantom Fury", "Touring");

**for**(**int** i =0; i<vehicles.length; i++) {

vehicles[i].startEngine();

vehicles[i].stopEngine();

}

}

}

**Exercise 2:**

**Create an abstract class Shape with an abstract method calculateArea(). Create two subclasses, Triangle and Square, that inherit from Shape. Implement the calculateArea() method to calculate the area of each shape.**

Solution:

public abstract class Shape {

abstract double calculateArea();

}

class Triangle extends Shape {

double base, height;

Triangle(double base, double height) {

this.base = base;

this.height = height;

}

@Override

double calculateArea() {

return 0.5 \* base \* height;

}

}

class Square extends Shape {

double side;

Square(double side) {

this.side = side;

}

@Override

double calculateArea() {

return side \* side;

}

}

public class Main {

public static void main(String[] args) {

Triangle triangle = new Triangle(5, 8);

Square square = new Square(4);

System.out.println("Triangle area: " + triangle.calculateArea());

System.out.println("Square area: " + square.calculateArea());

}

}

**Exercise 3:**

**Create an abstract class Person with attributes name and age, and an abstract method displayDetails(). Create two subclasses, Student and Teacher, that inherit from Person. Implement the displayDetails() method to display the details of each person.**

Solution:

abstract class Person {

String name;

int age;

abstract void displayDetails();

}

class Student extends Person {

int studentId;

Student(String name, int age, int studentId) {

this.name = name;

this.age = age;

this.studentId = studentId;

}

@Override

void displayDetails() {

System.out.println("Student: " + name + ", Age: " + age + ", Student ID: " + studentId);

}

}

class Teacher extends Person {

String subject;

Teacher(String name, int age, String subject) {

this.name = name;

this.age = age;

this.subject = subject;

}

@Override

void displayDetails() {

System.out.println("Teacher: " + name + ", Age: " + age + ", Subject: " + subject);

}

}

public class Main {

public static void main(String[] args) {

Student student = new Student("Alice", 20, 12345);

Teacher teacher = new Teacher("Mr. Smith", 35, "Mathematics");

student.displayDetails();

teacher.displayDetails();

}

}

**Exercise 4:**

**Create an abstract class BankAccount with attributes accountNumber and balance, and abstract methods deposit() and withdraw(). Create two subclasses, SavingsAccount and CheckingAccount, that inherit from BankAccount. Implement the methods in each subclass to handle deposits and withdrawals appropriately.**

Solution:

abstract class BankAccount {

int accountNumber;

double balance;

abstract void deposit(double amount);

abstract void withdraw(double amount);

}

class SavingsAccount extends BankAccount {

SavingsAccount(int accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

@Override

void deposit(double amount) {

balance += amount;

}

@Override

void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

} else {

System.out.println("Insufficient funds");

}

}

}

class CheckingAccount extends BankAccount {

CheckingAccount(int accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

@Override

void deposit(double amount) {

balance += amount;

}

@Override

void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

} else {

System.out.println("Insufficient funds");

}

}

}

public class Main {

public static void main(String[] args) {

SavingsAccount savingsAccount = new SavingsAccount(12345, 1000);

CheckingAccount checkingAccount = new CheckingAccount(67890, 500);

savingsAccount.deposit(200);

savingsAccount.withdraw(150);

checkingAccount.deposit(300);

checkingAccount.withdraw(700);

System.out.println("Savings account balance: " + savingsAccount.balance);

System.out.println("Checking account balance: " + checkingAccount.balance);

}

}

**Exercise 5:**

**Create an abstract class Shape with attributes color and filled, and an abstract method draw(). Create two subclasses, Circle and Rectangle, that inherit from Shape. Implement the draw() method to print out drawing instructions for each shape.**

Solution:

abstract class Shape {

String color;

boolean filled;

Shape(String color, boolean filled) {

this.color = color;

this.filled = filled;

}

abstract void draw();

}

class Circle extends Shape {

double radius;

Circle(String color, boolean filled, double radius) {

super(color, filled);

this.radius = radius;

}

@Override

void draw() {

System.out.println("Drawing a " + (filled ? "filled " : "unfilled ") + "circle with color " + color + " and radius " + radius);

}

}

class Rectangle extends Shape {

double width, height;

Rectangle(String color, boolean filled, double width, double height) {

super(color, filled);

this.width = width;

this.height = height;

}

@Override

void draw() {

System.out.println("Drawing a " + (filled ? "filled " : "unfilled ") + "rectangle with color " + color + " and dimensions " + width + "x" + height);

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle("Red", true, 5);

Rectangle rectangle = new Rectangle("Blue", false, 4, 6);

circle.draw();

rectangle.draw();

}

}

**Exercise 6:**

**Create an abstract class Employee with attributes name, salary, and a constructor to initialize them. Create two subclasses, Manager and Worker, that inherit from Employee. Implement a method displayInfo() in each subclass to display the details of each employee**.

Solution:

abstract class Employee {

String name;

double salary;

Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

abstract void displayInfo();

}

class Manager extends Employee {

String department;

Manager(String name, double salary, String department) {

super(name, salary);

this.department = department;

}

@Override

void displayInfo() {

System.out.println("Manager: " + name + ", Salary: $" + salary + ", Department: " + department);

}

}

class Worker extends Employee {

int hoursWorked;

Worker(String name, double salary, int hoursWorked) {

super(name, salary);

this.hoursWorked = hoursWorked;

}

@Override

void displayInfo() {

System.out.println("Worker: " + name + ", Salary: $" + salary + ", Hours Worked: " + hoursWorked);

}

}

public class Main {

public static void main(String[] args) {

Manager manager = new Manager("John Doe", 60000, "Sales");

Worker worker = new Worker("Alice Smith", 30000, 40);

manager.displayInfo();

worker.displayInfo();

}

}

**Exercise 7:**

**Create an abstract class Animal with attributes name and sound, and a constructor to initialize them. Create two subclasses, Dog and Cat, that inherit from Animal. Implement a method makeSound() in each subclass to print the sound of the animal.**

Solution:

abstract class Animal {

String name;

String sound;

Animal(String name, String sound) {

this.name = name;

this.sound = sound;

}

abstract void makeSound();

}

class Dog extends Animal {

Dog(String name) {

super(name, "Bark");

}

@Override

void makeSound() {

System.out.println(name + " makes a " + sound + " sound");

}

}

class Cat extends Animal {

Cat(String name) {

super(name, "Meow");

}

@Override

void makeSound() {

System.out.println(name + " makes a " + sound + " sound");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog("Buddy");

Cat cat = new Cat("Whiskers");

dog.makeSound();

cat.makeSound();

}

}

**Exercise 8:**

**Create an abstract class Employee with attributes name and salary, and abstract methods calculatePay() and displayInfo(). Create two subclasses, Manager and Worker, that inherit from Employee. Implement the methods in each subclass to calculate pay and display employee information.**

Solution:

public abstract class Employee {

String name;

double salary;

Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

abstract double calculatePay();

abstract void displayInfo();

}

class Manager extends Employee {

Manager(String name, double salary) {

super(name, salary);

}

@Override

double calculatePay() {

return salary;

}

@Override

void displayInfo() {

System.out.println("Manager: " + name + ", Salary: $" + salary);

}

}

class Worker extends Employee {

int hoursWorked;

double hourlyRate;

Worker(String name, int hoursWorked, double hourlyRate) {

super(name, 0);

this.hoursWorked = hoursWorked;

this.hourlyRate = hourlyRate;

}

@Override

double calculatePay() {

return hoursWorked \* hourlyRate;

}

@Override

void displayInfo() {

System.out.println("Worker: " + name + ", Hours Worked: " + hoursWorked + ", Hourly Rate: $" + hourlyRate);

}

}

public class Main {

public static void main(String[] args) {

Manager manager = new Manager("John Doe", 60000);

Worker worker = new Worker("Alice Smith", 40, 15);

manager.displayInfo();

System.out.println("Manager's Pay: $" + manager.calculatePay());

worker.displayInfo();

System.out.println("Worker's Pay: $" + worker.calculatePay());

}

}

**Exercise 9:**

**Create an abstract class BankAccount with attributes accountNumber and balance, and abstract methods deposit() and withdraw(). Create two subclasses, SavingsAccount and CheckingAccount, that inherit from BankAccount. Implement the methods in each subclass to handle deposits and withdrawals, with a minimum balance requirement for the checking account.**

Solution:

public abstract class BankAccount {

int accountNumber;

double balance;

BankAccount(int accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

abstract void deposit(double amount);

abstract void withdraw(double amount);

}

class SavingsAccount extends BankAccount {

SavingsAccount(int accountNumber, double initialBalance) {

super(accountNumber, initialBalance);

}

@Override

void deposit(double amount) {

balance += amount;

}

@Override

void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

} else {

System.out.println("Insufficient funds");

}

}

}

class CheckingAccount extends BankAccount {

static final double MINIMUM\_BALANCE = 100;

CheckingAccount(int accountNumber, double initialBalance) {

super(accountNumber, initialBalance);

}

@Override

void deposit(double amount) {

balance += amount;

}

@Override

void withdraw(double amount) {

if (balance - amount >= MINIMUM\_BALANCE) {

balance -= amount;

} else {

System.out.println("Insufficient funds or below minimum balance");

}

}

}

public class Main {

public static void main(String[] args) {

SavingsAccount savingsAccount = new SavingsAccount(12345, 1000);

CheckingAccount checkingAccount = new CheckingAccount(67890, 500);

savingsAccount.deposit(200);

savingsAccount.withdraw(150);

checkingAccount.deposit(300);

checkingAccount.withdraw(250);

System.out.println("Savings account balance: " + savingsAccount.balance);

System.out.println("Checking account balance: " + checkingAccount.balance);

}

}

**Exercise 10:**

**Create an abstract class Shape with an abstract method calculateArea() and a concrete method displayInfo() to display the shape's name. Create two subclasses, Rectangle and Circle, that inherit from Shape. Implement the calculateArea() method in each subclass to calculate the area of each shape.**

Solution:

public abstract class Shape {

abstract double calculateArea();

void displayInfo() {

System.out.println("This is a " + getClass().getSimpleName());

}

}

class Rectangle extends Shape {

double width, height;

Rectangle(double width, double height) {

this.width = width;

this.height = height;

}

@Override

double calculateArea() {

return width \* height;

}

}

class Circle extends Shape {

double radius;

Circle(double radius) {

this.radius = radius;

}

@Override

double calculateArea() {

return Math.PI \* radius \* radius;

}

}

public class Main {

public static void main(String[] args) {

Rectangle rectangle = new Rectangle(4, 6);

Circle circle = new Circle(5);

rectangle.displayInfo();

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

circle.displayInfo();

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

}

}