Practical Programming - Exception Handling and OOP Mastery

# Part 1: Exception Handling Code

Below is the code for the calculator program with exception handling:

import java.util.Scanner;  
  
class InvalidOperationException extends Exception {  
 public InvalidOperationException(String message) {  
 super(message);  
 }  
}  
  
public class CalculatorProgram {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 try {  
 System.out.print("Enter first number: ");  
 double num1 = scanner.nextDouble();  
  
 System.out.print("Enter second number: ");  
 double num2 = scanner.nextDouble();  
  
 System.out.print("Enter operation (+, -, \*, /): ");  
 char operation = scanner.next().charAt(0);  
  
 double result = 0;  
 switch (operation) {  
 case '+':  
 result = num1 + num2;  
 break;  
 case '-':  
 result = num1 - num2;  
 break;  
 case '\*':  
 result = num1 \* num2;  
 break;  
 case '/':  
 if (num2 == 0) {  
 throw new ArithmeticException("Division by zero is not allowed.");  
 }  
 result = num1 / num2;  
 break;  
 default:  
 throw new InvalidOperationException("Unsupported operation: " + operation);  
 }  
  
 System.out.println("Result: " + result);  
  
 } catch (InvalidOperationException | ArithmeticException e) {  
 System.out.println("Error: " + e.getMessage());  
 } catch (Exception e) {  
 System.out.println("Invalid input. Please enter numeric values.");  
 } finally {  
 scanner.close();  
 System.out.println("Calculator program finished.");  
 }  
 }  
}

## Output for Calculator Program:

Insert output screenshot here.

# Part 2: OOP Concepts Code

Below is the code for the Vehicle Management System demonstrating inheritance and constructors:

class Vehicle {  
 protected String model;  
 protected int year;  
 protected String fuelType;  
  
 public Vehicle(String model, int year, String fuelType) {  
 this.model = model;  
 this.year = year;  
 this.fuelType = fuelType;  
 }  
  
 public void start() {  
 System.out.println(model + " is starting.");  
 }  
  
 public void stop() {  
 System.out.println(model + " is stopping.");  
 }  
}  
  
class Car extends Vehicle {  
 private int numberOfDoors;  
  
 public Car(String model, int year, String fuelType, int numberOfDoors) {  
 super(model, year, fuelType);  
 this.numberOfDoors = numberOfDoors;  
 }  
  
 @Override  
 public void start() {  
 System.out.println(model + " (Car) is starting.");  
 }  
  
 public void openDoors() {  
 System.out.println(numberOfDoors + " doors opening.");  
 }  
}  
  
class Motorcycle extends Vehicle {  
 private boolean hasSidecar;  
  
 public Motorcycle(String model, int year, String fuelType, boolean hasSidecar) {  
 super(model, year, fuelType);  
 this.hasSidecar = hasSidecar;  
 }  
  
 @Override  
 public void start() {  
 System.out.println(model + " (Motorcycle) is starting.");  
 }  
  
 public void displaySidecarStatus() {  
 if (hasSidecar) {  
 System.out.println("This motorcycle has a sidecar.");  
 } else {  
 System.out.println("This motorcycle does not have a sidecar.");  
 }  
 }  
}  
  
public class VehicleManagementSystem {  
 public static void main(String[] args) {  
 Car car = new Car("Toyota Camry", 2020, "Gasoline", 4);  
 Motorcycle motorcycle = new Motorcycle("Harley Davidson", 2018, "Gasoline", true);  
  
 car.start();  
 car.openDoors();  
 car.stop();  
  
 motorcycle.start();  
 motorcycle.displaySidecarStatus();  
 motorcycle.stop();  
 }  
}

## Output for Vehicle Management System:

Insert output screenshot here.