

RegNo: 23MCA

Name: Vinayak Kumar Singh

Java Programming Lab (PMCA502P)

Ex 5.h Abstract class and Final Class

1: Abstract Shape Class

Define an abstract class named Shape. Inside the Shape class, declare an abstract method named calculateArea(). Create two concrete subclasses of Shape named Circle and Rectangle. Implement the calculateArea() method in each subclass to calculate the area of the circle and rectangle respectively.

In the main class, create instances of Circle and Rectangle, and invoke the calculateArea() method for each.

Code:

```
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 length;
    private double width;
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    @Override
    public double calculateArea() {
        return length * width;
    }
}
```

```

    }
}
public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        Rectangle rectangle = new Rectangle(4, 3);
        double circleArea = circle.calculateArea();
        double rectangleArea = rectangle.calculateArea();
        System.out.println("23MCA1030 Vinayak Kumar Singh");
        System.out.println("Area of the circle: " + circleArea);
        System.out.println("Area of the rectangle: " + rectangleArea);
    }
}

```

Output:

```

PROBLEMS  OUTPUT  PORTS  TERMINAL  COMMENTS

d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\1>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\1\" && javac Main.java &&
java Main
23MCA1030 Vinayak Kumar Singh
Area of the circle: 78.53981633974483
Area of the rectangle: 12.0

```

2: Abstract Animal Class

Create an abstract class called Animal. Define an abstract method named makeSound(). Create concrete subclasses of Animal named Dog, Cat, and Cow. Implement the makeSound() method in each subclass to make appropriate sounds for each animal.

In the main class, create instances of Dog, Cat, and Cow, and invoke the makeSound() method for each.

Code:

```

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!");
    }
}
class Cow extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Moo!");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        Animal cow = new Cow();
        dog.makeSound();
        cat.makeSound();
        cow.makeSound();
    }
}

```

Output:

PROBLEMS OUTPUT PORTS TERMINAL COMMENTS

 Code      

```

d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\2>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\2\" && javac Main.java && java Main
Woof!
Meow!
Moo!

```

3: Abstract Vehicle Class

Design an abstract class called Vehicle with instance variables make and model. Include a constructor to initialize these variables. Declare an abstract method named drive().

Create two concrete subclasses of Vehicle named Car and Motorcycle. Implement the drive() method in each subclass to print a message indicating the action of driving the respective vehicle.

In the main class, create instances of Car and Motorcycle, and invoke the drive() method for each.

Code:

```
abstract class Vehicle {
    private String make;
    private String model;
    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }
    public abstract void drive();
    public String getMake() {
        return make;
    }
    public String getModel() {
        return model;
    }
}
class Car extends Vehicle {
    public Car(String make, String model) {
        super(make, model); // Call parent class constructor to initialize make and
model
    }
    @Override
    public void drive() {
```

```

        System.out.println("Driving the " + getMake() + " " + getModel());
    }
}
class Motorcycle extends Vehicle {
    public Motorcycle(String make, String model) {
        super(make, model); // Call parent class constructor to initialize make and
model
    }
    @Override
    public void drive() {
        System.out.println("Riding the " + getMake() + " " + getModel());
    }
}
public class Main {
    public static void main(String[] args) {
        Car car = new Car("Telsa", "Model X");
        Motorcycle motorcycle = new Motorcycle("Yamaha", "R15");
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        car.drive();
        motorcycle.drive();
    }
}

```

Output:

```

PROBLEMS  OUTPUT  PORTS  TERMINAL  COMMENTS
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

D:\MCA\MCA Semester 2\1. Java Programming + Lab\Code>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\3\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
Driving the Telsa Model X
Riding the Yamaha R15

```

4: Abstract Bank Account Class

Create an abstract class named BankAccount with instance variables accountNumber and balance. Include a constructor to initialize these variables and abstract methods deposit(double amount) and withdraw(double amount).

Create two subclasses of BankAccount named SavingsAccount and CheckingAccount. Implement the deposit() and withdraw() methods in each subclass to handle transactions specific to savings and checking accounts respectively.

In the main class, create instances of SavingsAccount and CheckingAccount, and perform deposit and withdrawal transactions for each.

Code:

```
abstract class BankAccount {
    protected int accountNumber;
    protected double balance;
    public BankAccount(int accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }
    public abstract void deposit(double amount);
    public abstract void withdraw(double amount);
    public double getBalance() {
        return balance;
    }
}

class SavingsAccount extends BankAccount {
    public SavingsAccount(int accountNumber, double balance) {
        super(accountNumber, balance);
    }
    @Override
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
}
```

```

        System.out.println("Deposited $" + amount + " to savings account " +
accountNumber + ". New balance: $" + balance);
    } else {
        System.out.println("Deposit amount must be positive.");
    }
}

@Override
public void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        System.out.println("Withdrew $" + amount + " from savings account " +
accountNumber + ". New balance: $" + balance);
    } else {
        System.out.println("Insufficient funds in savings account " +
accountNumber);
    }
}

class CheckingAccount extends BankAccount {
    private double overdraftLimit;
    public CheckingAccount(int accountNumber, double balance, double
overdraftLimit) {
        super(accountNumber, balance);
        this.overdraftLimit = overdraftLimit;
    }
    @Override
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;

```

```
        System.out.println("Deposited $" + amount + " to checking account " +
accountNumber + ". New balance: $" + balance);
    } else {
        System.out.println("Deposit amount must be positive.");
    }
}
@Override
public void withdraw(double amount) {
    if (amount > 0 && amount <= balance + overdraftLimit) {
        balance -= amount;
        System.out.println("Withdrew $" + amount + " from checking account " +
accountNumber + ". New balance: $" + balance);
    } else {
        System.out.println("Insufficient funds in checking account " +
accountNumber);
    }
}
}
public class Main {
    public static void main(String[] args) {
        SavingsAccount savings = new SavingsAccount(12345, 1000);
        CheckingAccount checking = new CheckingAccount(54321, 500, 100);
        savings.deposit(500);
        savings.withdraw(200);
        checking.deposit(200);
        checking.withdraw(600);
    }
}
```


Output:

```
PROBLEMS OUTPUT PORTS TERMINAL COMMENTS
d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\3>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\4\" && javac Main.java && java Main
Deposited $500.0 to savings account 12345. New balance: $1500.0
Withdrew $200.0 from savings account 12345. New balance: $1300.0
Deposited $200.0 to checking account 54321. New balance: $700.0
Withdrew $600.0 from checking account 54321. New balance: $100.0
```

5: Abstract Employee Class

Define an abstract class called Employee with instance variables name and salary. Include a constructor to initialize these variables and an abstract method named calculateSalary().

Create concrete subclasses of Employee named HourlyEmployee and SalariedEmployee. Implement the calculateSalary() method in each subclass to calculate the total salary based on hours worked for hourly employees and fixed salary for salaried employees.

In the main class, create instances of HourlyEmployee and SalariedEmployee, and invoke the calculateSalary() method for each.

Code:

```
abstract class Employee {
    protected String name;
    protected double salary;
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    public abstract double calculateSalary();
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
}
```

```
class HourlyEmployee extends Employee {
    private int hoursWorked;
    private double hourlyRate;
    public HourlyEmployee(String name, double salary, int hoursWorked, double
hourlyRate) {
        super(name, salary); // Call superclass constructor to initialize name and
salary
        this.hoursWorked = hoursWorked;
        this.hourlyRate = hourlyRate;
    }
    @Override
    public double calculateSalary() {
        return hoursWorked * hourlyRate;
    }
}

class SalariedEmployee extends Employee {
    public SalariedEmployee(String name, double salary) {
        super(name, salary); // Call superclass constructor to initialize name and
salary
    }
    @Override
    public double calculateSalary() {
        return salary;
    }
}

public class Main {
    public static void main(String[] args) {
        HourlyEmployee hourlyEmployee = new HourlyEmployee("Vinayak Singh",
0, 40, 55.0);
```

```

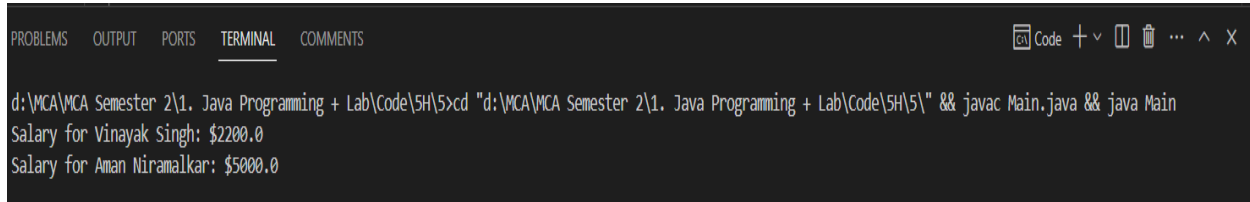
        SalariedEmployee salariedEmployee = new SalariedEmployee("Aman
Niramalkar", 5000.0);

        System.out.println("Salary for " + hourlyEmployee.getName() + ": $" +
hourlyEmployee.calculateSalary());

        System.out.println("Salary for " + salariedEmployee.getName() + ": $" +
salariedEmployee.calculateSalary());
    }
}

```

Output:



```

PROBLEMS  OUTPUT  PORTS  TERMINAL  COMMENTS
d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\5>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\5\" && javac Main.java && java Main
Salary for Vinayak Singh: $2200.0
Salary for Aman Niramalkar: $5000.0

```

6: Abstract Animal Class with Dynamic Method Dispatch (Revised)

Create an abstract class called `Animal` with an abstract method `makeSound()`. The `makeSound()` method should simply print "Animal makes a sound".

Create two concrete subclasses of `Animal` named `Dog` and `Cat`. In each subclass, implement the `makeSound()` method to print a sound specific to that animal (e.g., "Dog barks" for `Dog` and "Cat meows" for `Cat`).

In the main class, create instances of both `Dog` and `Cat`. Call the `makeSound()` method for each animal. Observe dynamic method dispatch in action as the correct `makeSound()` method is invoked based on the actual object type.

Code:

```

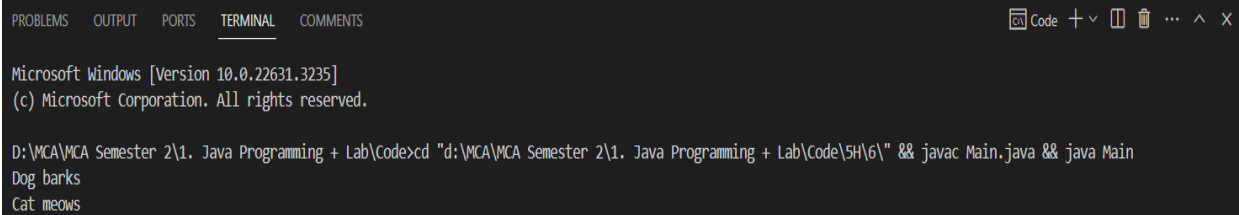
abstract class Animal {
    public abstract void makeSound();
}

class Dog extends Animal {
    @Override
    public void makeSound() {

```

```
        System.out.println("Dog barks");
    }
}
class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat meows");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        dog.makeSound(); // Output: Dog barks (due to dynamic method dispatch)
        cat.makeSound(); // Output: Cat meows (due to dynamic method dispatch)
    }
}
```

Output:



Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

D:\MCA\MCA Semester 2\1. Java Programming + Lab\Code>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\SH\6\" && javac Main.java && java Main
Dog barks
Cat meows

7: Final Class

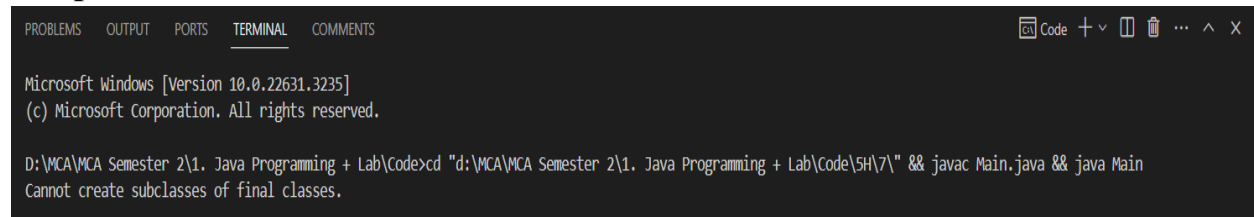
Define a class called FinalClass. Make it a final class. Inside the class, declare a final variable named CONSTANT and initialize it to some value. Create a method named display() that prints the value of the CONSTANT.

In the main class, attempt to create a subclass of FinalClass. Explain why it's not possible.

Code:

```
final class FinalClass {  
    private final int CONSTANT = 10;  
    public void display() {  
        System.out.println("Constant value: " + CONSTANT);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        // Attempted to create a subclass of FinalClass (compile-time error)  
        // FinalClass subClass = new SubClass();  
        System.out.println("Cannot create subclasses of final classes.");  
    }  
}
```

Output:



```
PROBLEMS OUTPUT PORTS TERMINAL COMMENTS Code + - [icon] [icon] [icon] [icon] X  
Microsoft Windows [Version 10.0.22631.3235]  
(c) Microsoft Corporation. All rights reserved.  
  
D:\MCA\MCA Semester 2\1. Java Programming + Lab\Code>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\7\" && javac Main.java && java Main  
Cannot create subclasses of final classes.
```

8: Final Method

Create a class called Parent. Inside Parent, define a method named display() as final, which prints "Parent display() method".

Create a subclass of Parent named Child. Attempt to override the display() method in Child with a method of the same name and signature. Explain why it's not possible.

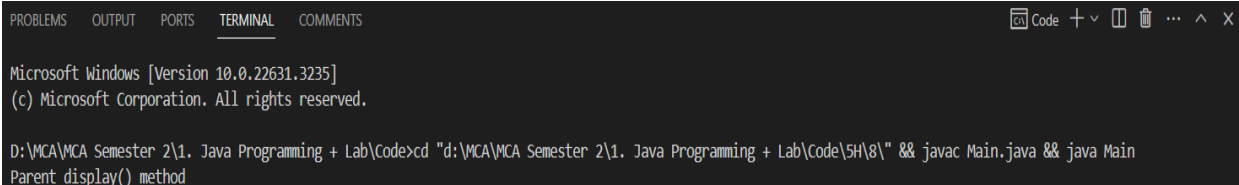
Code:

```
class Parent {
    public final void display() {
        System.out.println("Parent display() method");
    }
}

// public class Child extends Parent {
//     // Attempt to override display() (compile-time error)
//     // @Override
//     // public void display() {
//     //     System.out.println("Child display() method");
//     // }
// }

public class Main {
    public static void main(String[] args) {
        // Create an instance of Parent class
        Parent parent = new Parent();
        // Call the display method
        parent.display();
    }
}
```

Output:



The screenshot shows a code editor with a terminal window. The terminal output displays the Windows version and copyright information, followed by the command to compile and run the Java program. The output of the program is "Parent display() method".

```
PROBLEMS OUTPUT PORTS TERMINAL COMMENTS
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

D:\MCA\MCA Semester 2\1. Java Programming + Lab\Code>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\SH\8\" && javac Main.java && java Main
Parent display() method
```

9: Final Parameters

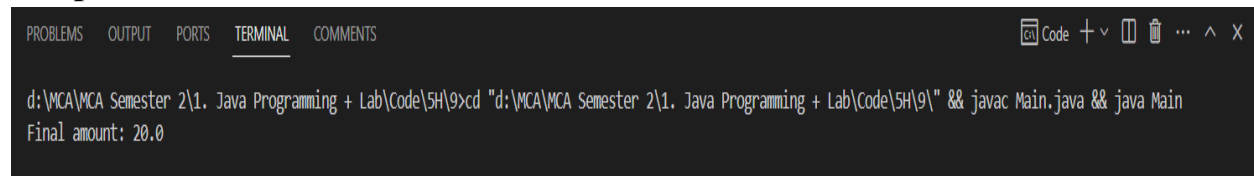
Define a class called Calculator. Inside Calculator, define a method named calculateFinalAmount(). This method should take a final parameter named finalAmount. Within the method, attempt to modify the value of finalAmount. Explain why it's not possible.

Code:

```
class Calculator {
    public void calculateFinalAmount(final double finalAmount) {
        // Attempt to modify finalAmount (compile-time error)
        // finalAmount = finalAmount + 10;
        System.out.println("Final amount: " + finalAmount);
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        calculator.calculateFinalAmount(20); // Output: Final amount: 20 (final
parameter cannot be modified)
    }
}
```

Output:



The screenshot shows a code editor with tabs for PROBLEMS, OUTPUT, PORTS, TERMINAL, and COMMENTS. The TERMINAL tab is active, displaying the command to compile and run the program: `d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\9>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\9\" && javac Main.java && java Main`. The output of the program is `Final amount: 20.0`.

10: Final Local Variable

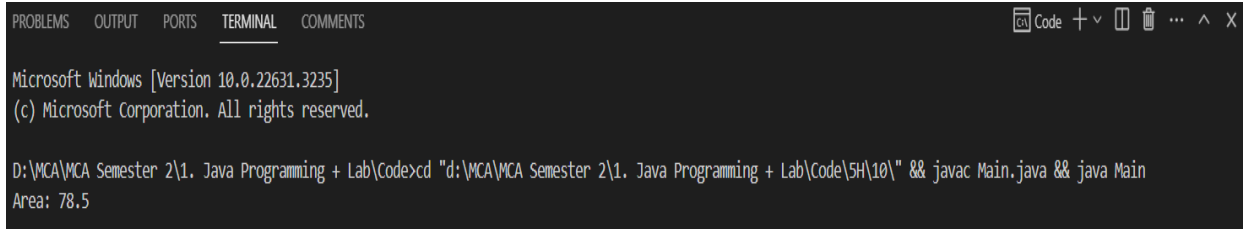
Create a method named `calculateArea()` inside a class called `ShapeCalculator`. Inside this method, define a final local variable named `PI` and assign the value of 3.14 to it. Attempt to modify the value of `PI` inside the method. Explain why it's not possible.

Code:

```
public class Main {
    public static void main(String[] args) {
        // Create an instance of ShapeCalculator class
        ShapeCalculator calculator = new ShapeCalculator();
        // Called calculateArea method and print result
        System.out.println("Area: " + calculator.calculateArea());
    }
}

class ShapeCalculator {
    public double calculateArea() {
        final double PI = 3.14;
        // Attempt to modify PI (compile-time error)
        // PI = 3.15;
        return PI * Math.pow(5, 2); // Replace radius with the actual value
    }
}
```

Output:



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, PORTS, TERMINAL, and COMMENTS. The TERMINAL tab is active, displaying the following text:

```
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

D:\MCA\MCA Semester 2\1. Java Programming + Lab\Code>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\5H\10\" && javac Main.java && java Main
Area: 78.5
```


11: Final Instance Variables

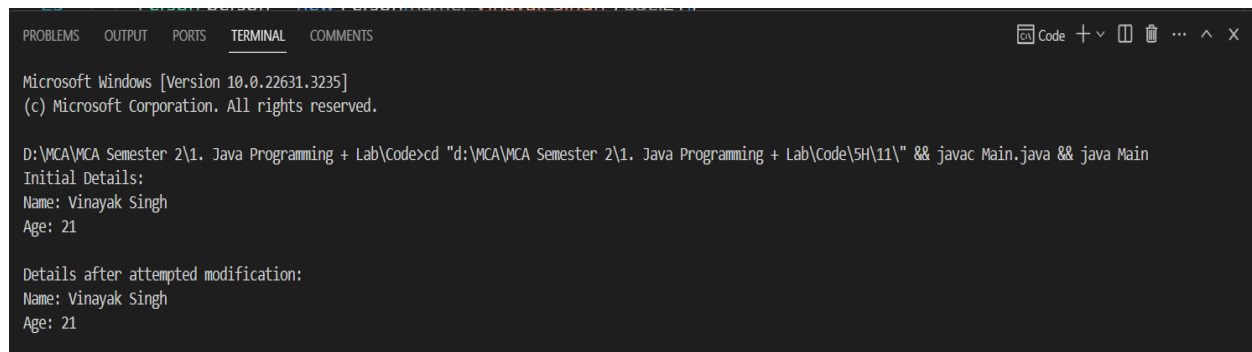
Define a class called Person with instance variables name and age, both marked as final. Create a constructor to initialize these variables. Attempt to modify the values of name and age after initialization. Explain why it's not possible.

Code:

```
class Person {  
  
    private final String name;  
  
    private final int age;  
  
    public Person(String name, int age) {  
  
        this.name = name;  
  
        this.age = age;  
  
    }  
  
    // Attempt to modify name and age after initialization (compile-time error)  
  
    // public void modifyDetails(String newName, int newAge) {  
  
    //     this.name = newName;  
  
    //     this.age = newAge;  
  
    // }  
  
    public String getName() {  
  
        return name;  
  
    }  
  
    public int getAge() {  
  
        return age;  
  
    }  
}
```

```
}  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        // Create an instance of the Person class  
  
        Person person = new Person("Vinayak Singh", 21);  
  
        // Display the initial details  
  
        System.out.println("Initial Details:");  
  
        displayPersonDetails(person);  
  
        // Attempt to modify name and age after initialization (compile-time error)  
  
        // person.modifyDetails("Vinayak Singh", 30);  
  
        // Display the details again  
  
        System.out.println("\nDetails after attempted modification:");  
  
        displayPersonDetails(person);  
  
    }  
  
    private static void displayPersonDetails(Person person) {  
  
        System.out.println("Name: " + person.getName());  
  
        System.out.println("Age: " + person.getAge());  
  
    }  
  
}
```

Output:



```
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

D:\MCA\MCA Semester 2\1. Java Programming + Lab\Code>cd "d:\MCA\MCA Semester 2\1. Java Programming + Lab\Code\SH\11\" && javac Main.java && java Main
Initial Details:
Name: Vinayak Singh
Age: 21

Details after attempted modification:
Name: Vinayak Singh
Age: 21
```