**Exercise 3: Implementing the Builder Pattern**

**Scenario:**

You are developing a system to create complex objects such as a computer with multiple optional parts. Use the Builder Pattern to manage the construction process.

1. Create a new Java Project:

* Create a new Java Project named BuilderPatternExample.
* A Java project named BuilderPatternExample is created in eclipse IDE.

2. Define a product class:

* Create a class **Computer** with attributes like **CPU**, **RAM**, **Storage**, etc.

public class Computer {

private String CPU;

private String RAM;

private String storage;

private String GPU;

private String motherboard;

private String powerSupply;

* The Computer class has attributes like CPU, RAM, storage, GPU, motherboard, and powerSupply.

3. Implement the Builder Class:

* Create a static nested Builder class inside Computer with methods to set each attribute.

public static class Builder {

private String CPU;

private String RAM;

private String storage;

private String GPU;

private String motherboard;

private String powerSupply;

public Builder setCPU(String CPU) {

this.CPU = CPU;

return this;

}

public Builder setRAM(String RAM) {

this.RAM = RAM;

return this;

}

public Builder setStorage(String storage) {

this.storage = storage;

return this;

}

public Builder setGPU(String GPU) {

this.GPU = GPU;

return this;

}

public Builder setMotherboard(String motherboard) {

this.motherboard = motherboard;

return this;

}

public Builder setPowerSupply(String powerSupply) {

this.powerSupply = powerSupply;

return this;

}

* The Builder class is a static nested class inside the Computer class.
* It has methods to set each attribute (setCPU(), setRAM(), setStorage(), setGPU(), setMotherboard(), setPowerSupply()).
* Provide a **build()** method in the Builder class that returns an instance of Computer.

public Computer build() {

return new Computer(this);

}

* The build() method returns an instance of Computer.

4. Implement the Builder Pattern:

* Ensure that the **Computer** class has a private constructor that takes the **Builder** as a parameter.

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.GPU = builder.GPU;

this.motherboard = builder.motherboard;

this.powerSupply = builder.powerSupply;

}

* The constructor is private and takes a Builder object as a parameter.

5. Test the Builder Implementation:

* Create a test class to demonstrate the creation of different configurations of Computer using the Builder pattern.

import java.util.Scanner;

public class TestComputer {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Computer.Builder builder = new Computer.Builder();

System.out.println("Enter CPU: ");

builder.setCPU(scanner.nextLine());

System.out.println("Enter RAM: ");

builder.setRAM(scanner.nextLine());

System.out.println("Enter Storage: ");

builder.setStorage(scanner.nextLine());

System.out.println("Enter GPU: ");

builder.setGPU(scanner.nextLine());

System.out.println("Enter Motherboard: ");

builder.setMotherboard(scanner.nextLine());

System.out.println("Enter Power Supply: ");

builder.setPowerSupply(scanner.nextLine());

Computer computer = builder.build();

System.out.println("Created Computer: " + computer);

scanner.close();

}}

* In the TestComputer class, user input is taken for each attribute of the Computer.
* The Builder object sets each attribute based on user input.
* The build() method creates the Computer object, which is then printed to demonstrate the configured computer.
* Output:

