**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.

**Steps:**

**1. Create a New Java Project:**

* **Project Name**: BuilderPatternExample
* Use your preferred IDE (like IntelliJ IDEA, Eclipse, or NetBeans) to create a new Java project.

**2. Define a Product Class:**

**Computer Class**:

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

public class Computer {

// Required parameters

private String CPU;

private String RAM;

// Optional parameters

private String storage;

private String graphicsCard;

private String powerSupply;

private String motherboard;

// Private constructor

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

this.powerSupply = builder.powerSupply;

this.motherboard = builder.motherboard;

}

// Getters for the attributes

public String getCPU() {

return CPU;

}

public String getRAM() {

return RAM;

}

public String getStorage() {

return storage;

}

public String getGraphicsCard() {

return graphicsCard;

}

public String getPowerSupply() {

return powerSupply;

}

public String getMotherboard() {

return motherboard;

}

@Override

public String toString() {

return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", storage=" + storage

+ ", graphicsCard=" + graphicsCard + ", powerSupply=" + powerSupply

+ ", motherboard=" + motherboard + "]";

}

}

**3. Implement the Builder Class:**

**Builder Class**:

* Implement a static nested Builder class inside the Computer class with methods to set each attribute.
* Provide a build() method that returns an instance of Computer.

public static class Builder {

// Required parameters

private String CPU;

private String RAM;

// Optional parameters

private String storage;

private String graphicsCard;

private String powerSupply;

private String motherboard;

// Constructor for required parameters

public Builder(String CPU, String RAM) {

this.CPU = CPU;

this.RAM = RAM;

}

// Methods to set optional parameters

public Builder setStorage(String storage) {

this.storage = storage;

return this;

}

public Builder setGraphicsCard(String graphicsCard) {

this.graphicsCard = graphicsCard;

return this;

}

public Builder setPowerSupply(String powerSupply) {

this.powerSupply = powerSupply;

return this;

}

public Builder setMotherboard(String motherboard) {

this.motherboard = motherboard;

return this;

}

// Method to build the Computer object

public Computer build() {

return new Computer(this);

}

}

**4. Implement the Builder Pattern:**

* The Computer class has a private constructor that takes the Builder as a parameter, enforcing the use of the Builder to create instances of Computer.

**5. Test the Builder Implementation:**

**TestBuilderPattern Class**:

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

public class TestBuilderPattern {

public static void main(String[] args) {

// Creating a computer with all components

Computer gamingComputer = new Computer.Builder("Intel i9", "32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 3080")

.setPowerSupply("750W")

.setMotherboard("ASUS ROG Strix")

.build();

System.out.println(gamingComputer);

// Creating a computer with only required components

Computer officeComputer = new Computer.Builder("Intel i5", "16GB")

.build();

System.out.println(officeComputer);

}

}