**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:**
   * Create a new Java project named **BuilderPatternExample**.
2. **Define a Product Class:**
   * Create a class **Computer** with attributes like **CPU**, **RAM**, **Storage**, etc.
3. **Implement the Builder Class:**
   * Create a static nested Builder class inside Computer with methods to set each attribute.
   * Provide a **build()** method in the Builder class that 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.
5. **Test the Builder Implementation:**
   * Create a test class to demonstrate the creation of different configurations of Computer using the Builder pattern.

**SOLUTION:**

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

Create a new Java project named BuilderPatternExample.

**Step 2:** Define a Product Class:

public class Computer {

    private String cpu;

    private String ram;

    private String storage;

    private String operatingsystem;

    public Computer(Builder build) {

        this.cpu=build.cpu;

        this.ram=build.ram;

        this.storage=build.storage;

        this.operatingsystem=build.operatingsystem;

    }

    public String setCPU() {

        return cpu;

    }

    public String setram() {

        return ram;

    }

    public String setStorage() {

        return storage;

    }

    public String setOS() {

        return operatingsystem;

    }

    public String toString() {

        return "Computer [CPU=" + cpu + ", RAM=" + ram + ", storage=" + storage +",OS="+operatingsystem+"]";

    }}

**Step 3:** Implement the Builder Class

public static class Builder {

    private String cpu;

    private String ram;

    private String storage;

    private String operatingsystem;

    public Builder() {

    }

    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 setOS(String os) {

        this.operatingsystem=os;

        return this;

    }

    public Computer build() {

        return new Computer(this);

    }

}

}

**Step 4**: Implement the Builder Pattern:

    public Computer(Builder build) {

        this.cpu=build.cpu;

        this.ram=build.ram;

        this.storage=build.storage;

        this.operatingsystem=build.operatingsystem;

    }

**Step 5:** Test the Builder Implementation

package Builder\_example;

public class Test\_builder {

    public static void main(String[] args) {

        Computer gamingCp = new Computer.Builder()

                .setCPU("Intel Core i9")

                .setRAM("32GB")

                .setStorage("1TB SSD")

                .setOS("Windows 11")

                .build();

        Computer officeCp = new Computer.Builder()

                .setCPU("Intel Core i5")

                .setRAM("16GB")

                .setStorage("512GB SSD")

                .setOS("Windows 10")

                .build();

        Computer budgetCp = new Computer.Builder()

                .setCPU("AMD Ryzen 3")

                .setRAM("8GB")

                .setStorage("256GB SSD")

                .setOS("Linux")

                .build();

        // Print the computer built

        System.out.println("Gaming Computer: " + gamingCp);

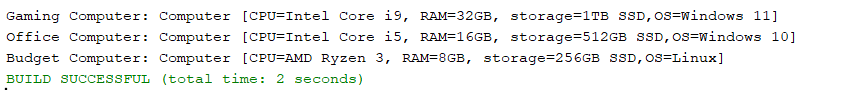
        System.out.println("Office Computer: " + officeCp);

        System.out.println("Budget Computer: " + budgetCp);

    }

}

**SAMPLE OUTPUT:**

****