**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.

**CODE:-**

// BuilderPatternExample.java

class Computer {

    // Required components

    private String CPU;

    private String RAM;

    // Optional components

    private String storage;

    private String graphicsCard;

    private String operatingSystem;

    // Private constructor

    private Computer (Builder builder) {

        this.CPU = builder.CPU;

        this.RAM = builder.RAM;

        this.storage = builder.storage;

        this.graphicsCard = builder.graphicsCard;

        this.operatingSystem = builder.operatingSystem;

    }

    // Static nested Builder class

    public static class Builder {

        private String CPU;

        private String RAM;

        private String storage;

        private String graphicsCard;

        private String operatingSystem;

        public Builder(String CPU, String RAM) {

            this.CPU = CPU;

            this.RAM = RAM;

        }

        public Builder setStorage(String storage) {

            this.storage = storage;

            return this;

        }

        public Builder setGraphicsCard(String graphicsCard) {

            this.graphicsCard = graphicsCard;

            return this;

        }

        public Builder setOperatingSystem(String operatingSystem) {

            this.operatingSystem = operatingSystem;

            return this;

        }

        public Computer build() {

            return new Computer(this);

        }

    }

    @Override

    public String toString() {

        return """

               Computer Configuration:

               CPU: """ + CPU + "\n" +

                "RAM: " + RAM + "\n" +

                "Storage: " + (storage != null ? storage : "Not included") + "\n" +

                "Graphics Card: " + (graphicsCard != null ? graphicsCard : "Not included") + "\n" +

                "Operating System: " + (operatingSystem != null ? operatingSystem : "Not installed");

    }

}

// Test class

public class BuilderPatternExample {

    public static void main(String[] args) {

        // Basic configuration

        Computer basicComputer = new Computer.Builder("Intel i5", "8GB")

                .build();

        // Gaming configuration

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

                .setGraphicsCard("NVIDIA RTX 4080")

                .setStorage("1TB SSD")

                .setOperatingSystem("Windows 11")

                .build();

        // Developer configuration

        Computer devComputer = new Computer.Builder("AMD Ryzen 7", "16GB")

                .setStorage("512GB SSD")

                .setOperatingSystem("Ubuntu 22.04")

                .build();

        System.out.println(basicComputer);

        System.out.println("\n---------------------------\n");

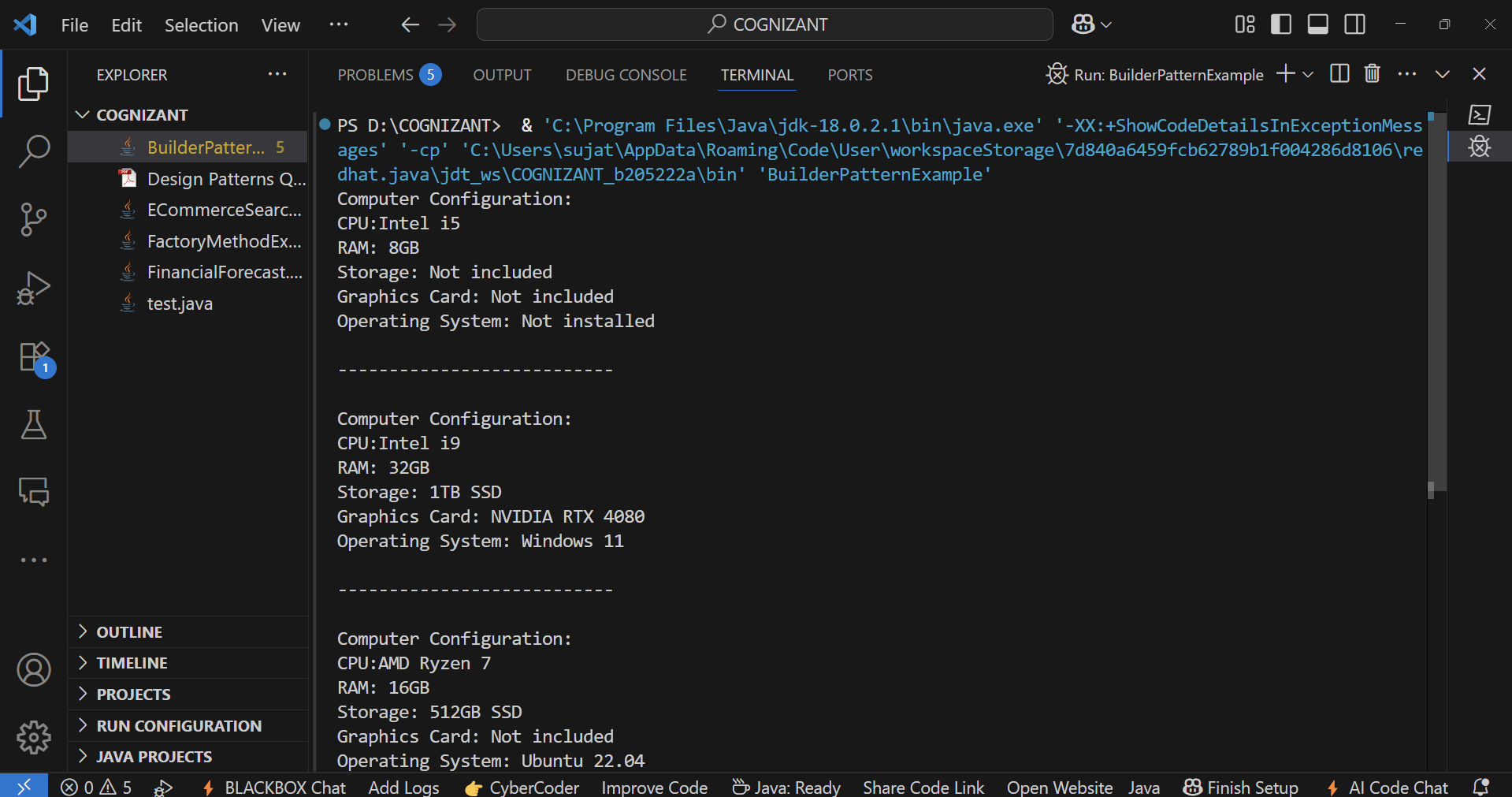
        System.out.println(gamingComputer);

        System.out.println("\n---------------------------\n");

        System.out.println(devComputer);

    }

}

**OUTPUT:-**