Documentation for the Builder Pattern

Introduction

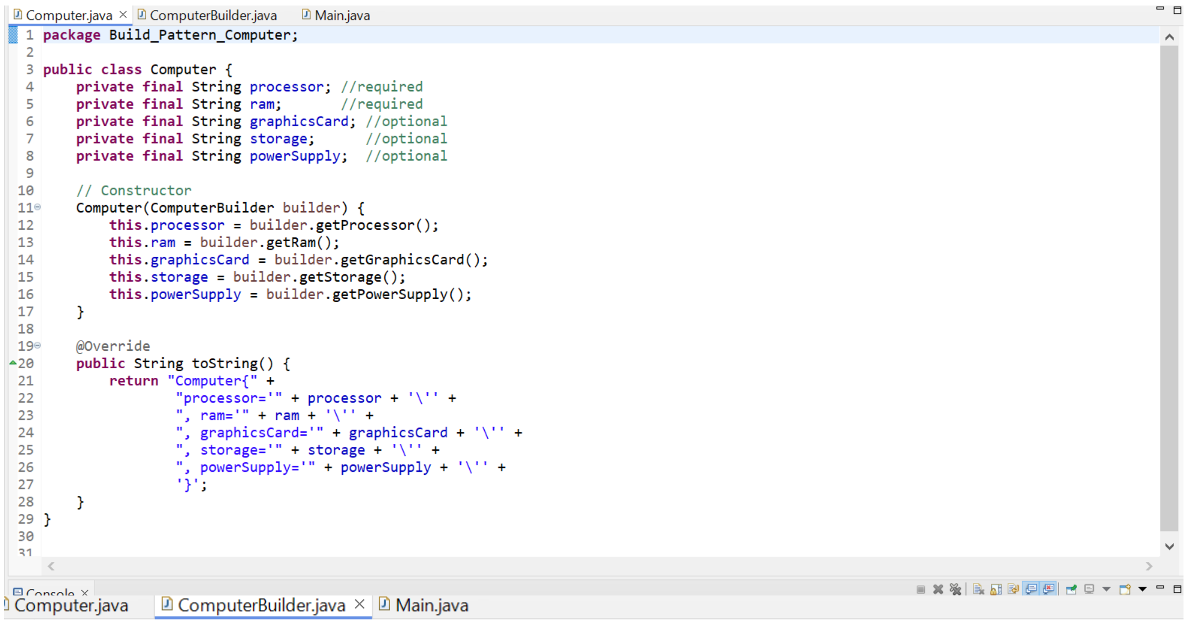
The Builder Pattern is a creational design pattern that allows for the construction of complex objects step by step. It provides a flexible solution for object creation by allowing optional parameters without having to define multiple constructors. The Builder Pattern is particularly useful when an object can be created with various configurations or optional attributes.

Code Overview

In this example, we have implemented a Builder Pattern to create a Computer object. The Computer class has required attributes such as processor and ram, as well as optional attributes like graphicsCard, storage, and powerSupply.

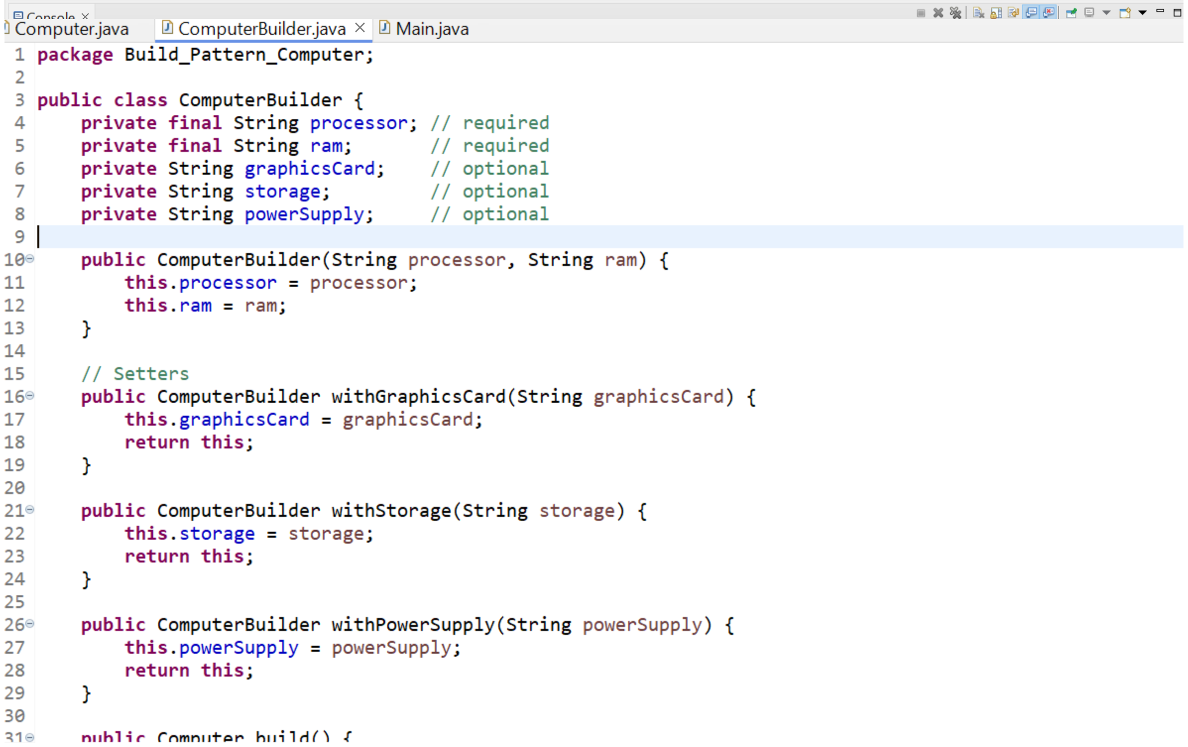
The pattern consists of two main classes:

1. **Computer.java**: Represents the product (the object being built).
2. **ComputerBuilder.java**: Implements the Builder Pattern, allowing the creation of Computer objects with different configurations.

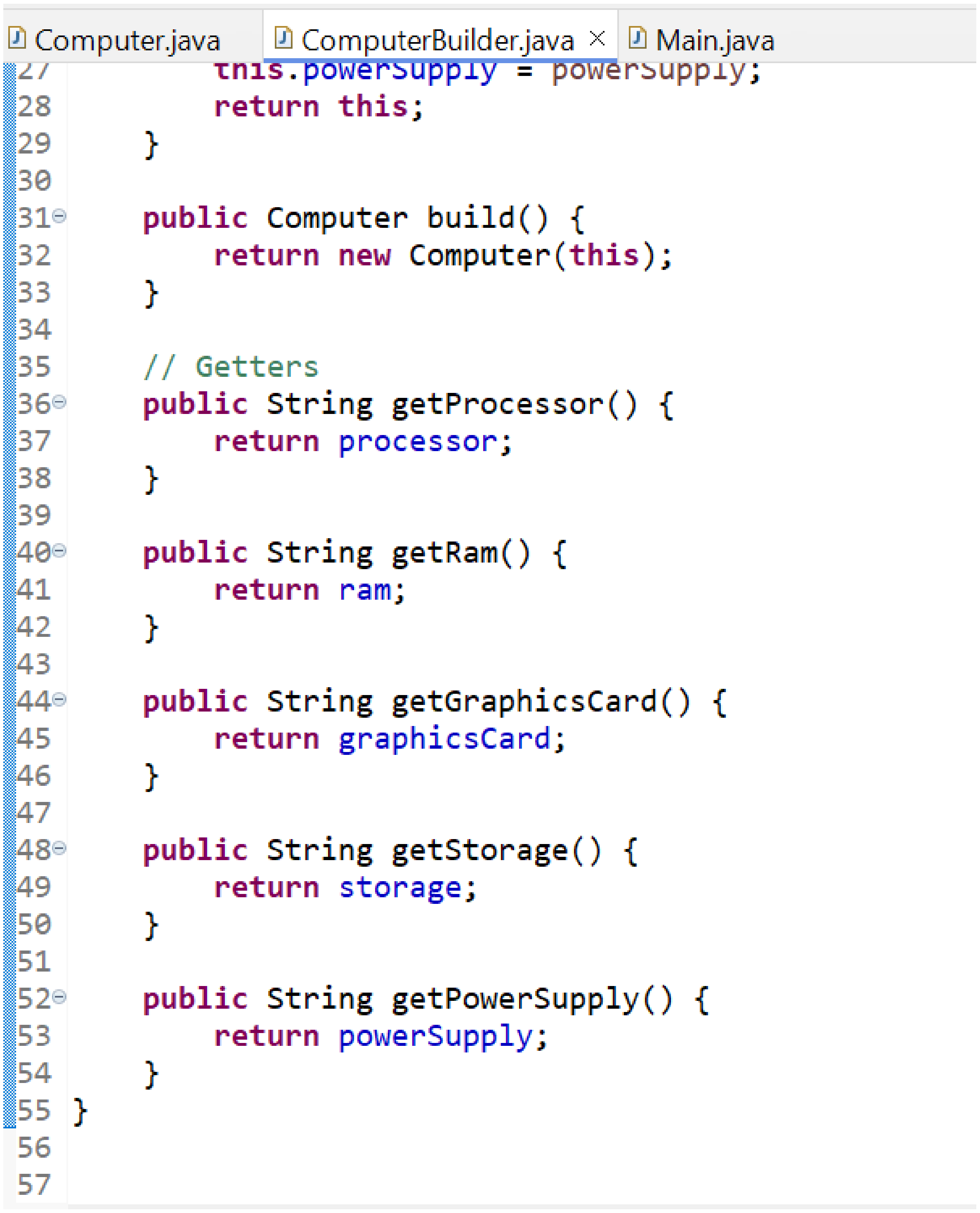

Computer.java × ComputerBuilder.java Main.java

```java
package Build_Pattern_Computer;

public class Computer {
    private final String processor; //required
    private final String ram;       //required
    private final String graphicsCard; //optional
    private final String storage;      //optional
    private final String powerSupply;  //optional

    // Constructor
    Computer(ComputerBuilder builder) {
        this.processor = builder.getProcessor();
        this.ram = builder.getRam();
        this.graphicsCard = builder.getGraphicsCard();
        this.storage = builder.getStorage();
        this.powerSupply = builder.getPowerSupply();
    }

    @Override
    public String toString() {
        return "Computer{" +
                "processor='" + processor + '\'' +
                ", ram='" + ram + '\'' +
                ", graphicsCard='" + graphicsCard + '\'' +
                ", storage='" + storage + '\'' +
                ", powerSupply='" + powerSupply + '\'' +
                '}';
    }
}
```

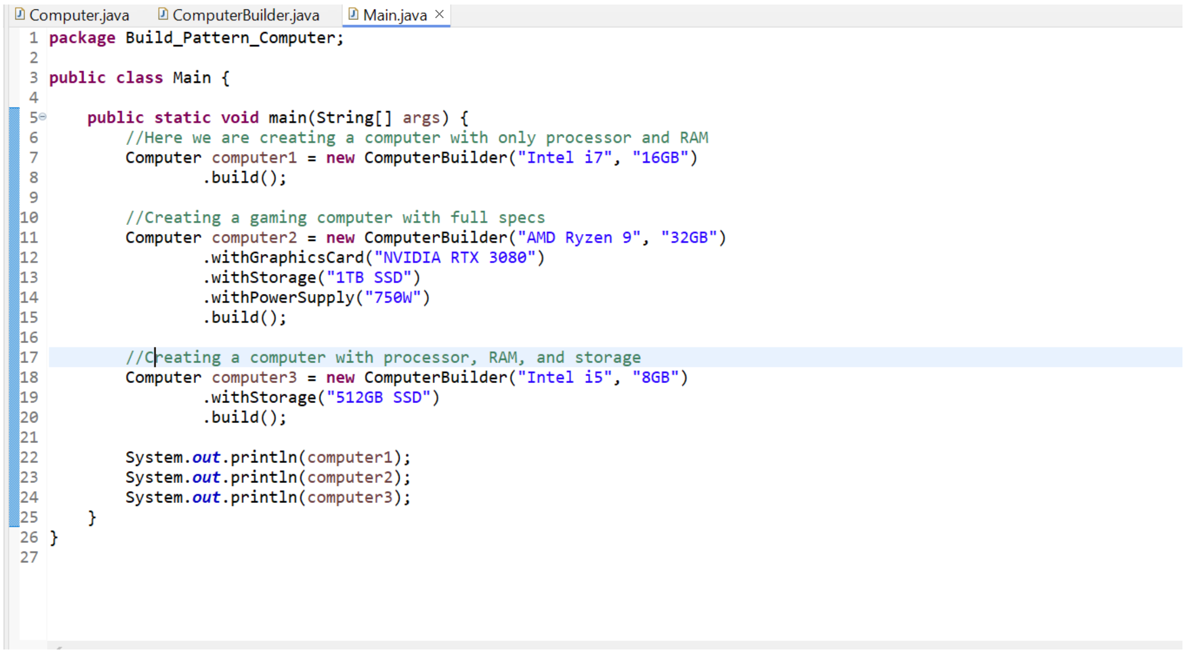
Console ×


Computer.java ComputerBuilder.java × Main.java

```java
package Build_Pattern_Computer;

public class ComputerBuilder {
    private final String processor; // required
    private final String ram;       // required
    private String graphicsCard;    // optional
    private String storage;         // optional
    private String powerSupply;     // optional

    public ComputerBuilder(String processor, String ram) {
        this.processor = processor;
        this.ram = ram;
    }

    // Setters
    public ComputerBuilder withGraphicsCard(String graphicsCard) {
        this.graphicsCard = graphicsCard;
        return this;
    }

    public ComputerBuilder withStorage(String storage) {
        this.storage = storage;
        return this;
    }

    public ComputerBuilder withPowerSupply(String powerSupply) {
        this.powerSupply = powerSupply;
        return this;
    }

    public Computer build() {
```

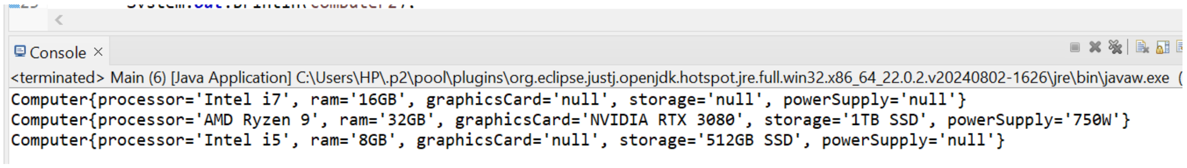

Computer.java | ComputerBuilder.java | Main.java

```java
        this.powerSupply = powerSupply;
        return this;
    }

    public Computer build() {
        return new Computer(this);
    }

    // Getters
    public String getProcessor() {
        return processor;
    }

    public String getRam() {
        return ram;
    }

    public String getGraphicsCard() {
        return graphicsCard;
    }

    public String getStorage() {
        return storage;
    }

    public String getPowerSupply() {
        return powerSupply;
    }
}
```



```java
package Build_Pattern_Computer;

public class Main {

    public static void main(String[] args) {
        //Here we are creating a computer with only processor and RAM
        Computer computer1 = new ComputerBuilder("Intel i7", "16GB")
                .build();

        //Creating a gaming computer with full specs
        Computer computer2 = new ComputerBuilder("AMD Ryzen 9", "32GB")
                .withGraphicsCard("NVIDIA RTX 3080")
                .withStorage("1TB SSD")
                .withPowerSupply("750W")
                .build();

        //Creating a computer with processor, RAM, and storage
        Computer computer3 = new ComputerBuilder("Intel i5", "8GB")
                .withStorage("512GB SSD")
                .build();

        System.out.println(computer1);
        System.out.println(computer2);
        System.out.println(computer3);
    }
}
```

The client (user) interacts with the ComputerBuilder to construct a Computer object with desired specifications.



```
<terminated> Main (6) [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.2.v20240802-1626\jre\bin\javaw.exe (
Computer{processor='Intel i7', ram='16GB', graphicsCard='null', storage='null', powerSupply='null'}
Computer{processor='AMD Ryzen 9', ram='32GB', graphicsCard='NVIDIA RTX 3080', storage='1TB SSD', powerSupply='750W'}
Computer{processor='Intel i5', ram='8GB', graphicsCard='null', storage='512GB SSD', powerSupply='null'}
```

Methods and Key Points

- **Computer.java**:
    - **Constructor**: The Computer constructor is private and only accessible by the ComputerBuilder. It takes a ComputerBuilder object as an argument to initialize the attributes of the Computer.
    - **toString() Method**: This method is overridden to provide a string representation of the Computer object, listing all its attributes.
- **ComputerBuilder.java**:
    - **Constructor**: The ComputerBuilder constructor requires the mandatory attributes (processor and ram) to create a valid Computer. These are passed when creating an instance of the builder.
    - **withGraphicsCard(String graphicsCard)**, **withStorage(String storage)**, **withPowerSupply(String powerSupply)**: These are the optional setters that allow the client to add extra features to the Computer. Each method returns the ComputerBuilder instance, enabling method chaining.
    - **build() Method**: This method constructs the Computer object using the attributes set in the builder. It returns a fully initialized Computer instance.

Key Points

- **Mandatory and Optional Parameters**: The Builder Pattern separates mandatory parameters (passed through the builder's constructor) from optional parameters (set through method chaining).
- **Immutable Object**: The Computer object is immutable once it’s built, meaning that its state cannot be modified after creation.
- **Method Chaining**: The builder methods return the builder instance itself, allowing for a fluent interface style, where methods can be chained together.
- **Flexible Object Creation**: The Builder Pattern makes it easy to create objects with varying configurations, without requiring numerous constructors or setter methods.