**Exercise 3: Implementing the Builder Pattern**

**Computer.java**

**package** mypackage;

**public** **class** Computer {

// Required parameters

**private** String CPU;

**private** String RAM;

// Optional parameters

**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 storage(String storage) {

**this**.storage = storage;

**return** **this**;

}

**public** Builder graphicsCard(String graphicsCard) {

**this**.graphicsCard = graphicsCard;

**return** **this**;

}

**public** Builder operatingSystem(String operatingSystem) {

**this**.operatingSystem = operatingSystem;

**return** **this**;

}

**public** Computer build() {

**return** **new** Computer(**this**);

}

}

@Override

**public** String toString() {

**return** "Computer Configuration:\n" +

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

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

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

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

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

}

}

**TestBuilderPattern.java**

**package** mypackage;

**public** **class** TestBuilderPattern {

**public** **static** **void** main(String[] args) {

// Creating computer with minimal configuration

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

System.***out***.println(basicComputer);

System.***out***.println();

// Creating computer with full configuration

Computer gamingComputer = **new** Computer.Builder("AMD Ryzen 9", "32GB")

.storage("1TB SSD")

.graphicsCard("NVIDIA RTX 4090")

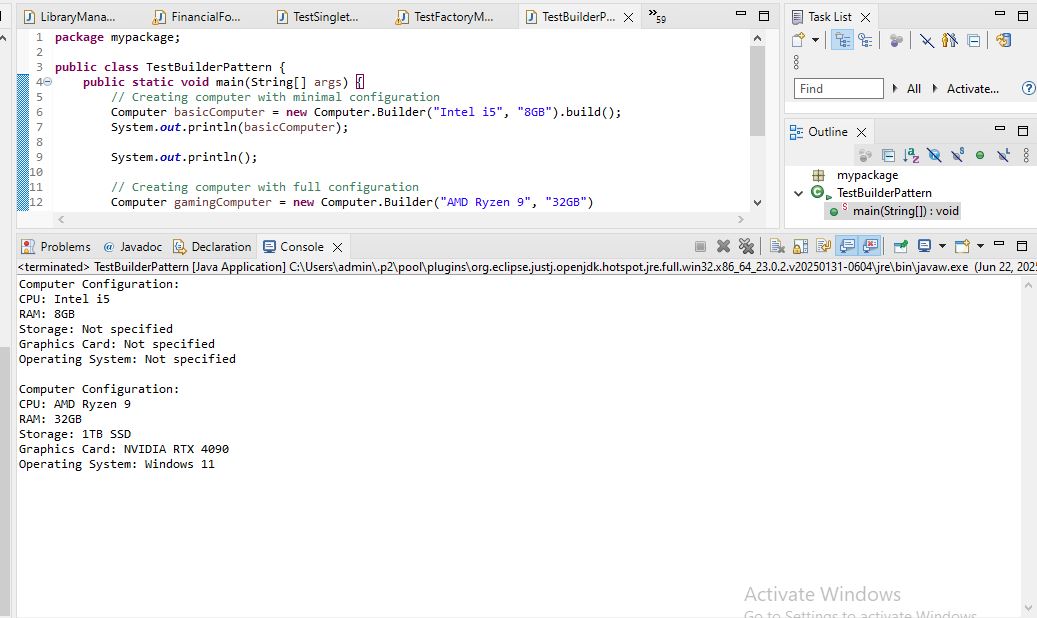
.operatingSystem("Windows 11")

.build();

System.***out***.println(gamingComputer);

}

}

  
We are developing a system to create complex objects such as a Computer with multiple optional parts.

**1. Product Class:**  
The Computer class represents the product with attributes: CPU, RAM, Storage, Graphics Card, and Operating System.  
Some attributes are mandatory (CPU, RAM), while others are optional.

**2. Builder Class:**  
A static nested class Builder is created inside the Computer class to allow step-by-step construction.  
The Builder class has methods to set optional attributes.  
The build() method returns the fully constructed Computer object.

**3. Advantages of Builder Pattern:**

* Handles construction of complex objects.
* Supports optional parameters flexibly.
* Improves code readability and maintainability.
* Avoids multiple constructors with many parameters (constructor telescoping problem).
* Produces immutable final object after building.

**4. Time Complexity:**  
Building object: O(1) (constant time for object creation).

**5. Real-life Examples:**

* GUI builders
* Query builders
* Configuration builders in frameworks