**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:**

class Computer

{

public string CPU { get; set; }

public int RAM { get; set; }

public int Storage { get; set; }

private Computer() { }

public class Builder

{

private Computer computer = new Computer();

public Builder WithCPU(string cpu) { computer.CPU = cpu; return this; }

public Builder WithRAM(int ram) { computer.RAM = ram; return this; }

public Builder WithStorage(int storage) { computer.Storage = storage; return this; }

public Computer Build() => computer;

}

public void Display() => Console.WriteLine($"CPU: {CPU}, RAM: {RAM}GB, Storage: {Storage}GB");

}

class Program

{

static void Main()

{

Computer gamingPC = new Computer.Builder()

.WithCPU("Intel i9")

.WithRAM(32)

.WithStorage(1000)

.Build();

gamingPC.Display();

}

}

**OUTPUT:**

