## Exercise 3: Implementing the Builder Pattern

Scenario:  
Developing a system to create complex objects such as a Computer with multiple optional parts.  
Use the Builder Pattern to manage the construction process.

## Step-by-step Implementation:

### Step 1: Create a New Java Project

Create a new Java project named BuilderPatternExample.

### Step 2: Define a Product Class

Create a class Computer with attributes like CPU, RAM, Storage, etc.

public class Computer {  
 private String CPU;  
 private String RAM;  
 private String storage;  
  
 private Computer(Builder builder) {  
 this.CPU = builder.CPU;  
 this.RAM = builder.RAM;  
 this.storage = builder.storage;  
 }  
  
 public static class Builder {  
 private String CPU;  
 private String RAM;  
 private String storage;  
  
 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 Computer build() {  
 return new Computer(this);  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage + "]";  
 }  
}

### Step 5: Test the Builder Implementation

public class TestBuilderPattern {  
 public static void main(String[] args) {  
 Computer computer1 = new Computer.Builder()  
 .setCPU("Intel i7")  
 .setRAM("16GB")  
 .setStorage("512GB SSD")  
 .build();  
  
 Computer computer2 = new Computer.Builder()  
 .setCPU("AMD Ryzen 5")  
 .setRAM("8GB")  
 .build();  
  
 System.out.println(computer1);  
 System.out.println(computer2);  
 }  
}

OUTPUT:

