Builder Pattern Implementation for Computer Configuration

public class BuilderPatternExample {

public static void main(String[] args) {

Computer gamingPC = new Computer.Builder("Intel i9")

.setRAM(32)

.setStorage(1000)

.setGPU("NVIDIA RTX 3080")

.setHasBluetooth(true)

.setHasWiFi(true)

.build();

Computer officePC = new Computer.Builder("AMD Ryzen 5")

.setRAM(16)

.setStorage(500)

.build();

Computer budgetPC = new Computer.Builder("Intel i3")

.setRAM(8)

.setStorage(256)

.setHasWiFi(true)

.build();

System.out.println("Gaming PC Configuration:");

System.out.println(gamingPC);

System.out.println("\nOffice PC Configuration:");

System.out.println(officePC);

System.out.println("\nBudget PC Configuration:");

System.out.println(budgetPC);

System.out.println("\nBuilding a custom PC step by step:");

Computer.Builder customBuilder = new Computer.Builder("AMD Ryzen 7");

customBuilder.setRAM(64);

customBuilder.setStorage(2000);

customBuilder.setGPU("NVIDIA RTX 3090");

customBuilder.setHasBluetooth(true);

Computer customPC = customBuilder.build();

System.out.println(customPC);

}

static class Computer {

private final String CPU;

private final int RAM;

private final int storage;

private final String GPU;

private final boolean hasBluetooth;

private final boolean hasWiFi;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.GPU = builder.GPU;

this.hasBluetooth = builder.hasBluetooth;

this.hasWiFi = builder.hasWiFi;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append("CPU: ").append(CPU).append("\n");

sb.append("RAM: ").append(RAM).append("GB\n");

sb.append("Storage: ").append(storage).append("GB SSD\n");

if (GPU != null) sb.append("GPU: ").append(GPU).append("\n");

sb.append("Bluetooth: ").append(hasBluetooth ? "Yes" : "No").append("\n");

sb.append("WiFi: ").append(hasWiFi ? "Yes" : "No");

return sb.toString();

}

public static class Builder {

private final String CPU;

private int RAM = 8;

private int storage = 256;

private String GPU = null;

private boolean hasBluetooth = false;

private boolean hasWiFi = false;

public Builder(String CPU) {

this.CPU = CPU;

}

public Builder setRAM(int RAM) {

this.RAM = RAM;

return this;

}

public Builder setStorage(int storage) {

this.storage = storage;

return this;

}

public Builder setGPU(String GPU) {

this.GPU = GPU;

return this;

}

public Builder setHasBluetooth(boolean hasBluetooth) {

this.hasBluetooth = hasBluetooth;

return this;

}

public Builder setHasWiFi(boolean hasWiFi) {

this.hasWiFi = hasWiFi;

return this;

}

public Computer build() {

return new Computer(this);

}

}

}

}

Key Features of This Implementation:

1. **Complex Object Construction**: Handles creation of Computer objects with many optional components
2. **Fluent Interface**: Method chaining makes the client code readable
3. **Immutable Object**: The Computer object is immutable once created
4. **Parameter Validation**: Can be added in the build() method (not shown for simplicity)
5. **Step-by-Step Construction**: Shows how to build an object incrementally

Design Pattern Benefits:

1. **Separation of Concerns**: Construction logic is separate from the object representation
2. **Flexible Construction**: Can create different representations using the same construction process
3. **Readable Code**: Fluent interface makes client code more readable
4. **Control Over Construction**: Can enforce invariants during object creation
5. **Immutable Objects**: The built objects are immutable by design

This implementation demonstrates how to use the Builder Pattern to create complex objects with many optional parameters in a clean and maintainable way.

Output

