Title: Design Patterns: Builder & Observer

Subject: Padrões e Desenho de Software

Author: João Miguel Abrantes Dias

Date: 06/05/2020

**Index**

[Introductory Note: 2](#_Toc486947527)

[1. Builder PatterN 3](#_Builder_Pattern)

[1.1 Introduction 3](#_Introduction_1)

[1.2 Problem 3](#_Problem_1)

[1.3 Solution 4](#_Solution_1)

[1.4 Advantages 5](#_Advantages_1)

[1.5 Disadvantages 5](#_Disadvantages_1)

[1.6 References 5](#_References_1)

[2. Observer PatterN 6](#_Observer_Pattern)

[2.1 Introduction 6](#_Introduction)

[2.2 Problem 6](#_Problem)

[2.3 Solution x](#_Solution)

[2.4 Advantages x](#_Advantages)

[2.5 Disadvantages x](#_Disadvantages)

[2.6 References x](#_References)

**Introductory Note:**

#TODO

This document (by way of its own structure) presents a recommended template for a work-in-progress document, a report or an essay. It can also be used for the main body of a dissertation or thesis.

The first lines of each section also present a possible narrative style to be used in a similar section in a work-in-progress document, a report or an essay.

# Builder Pattern

## Introduction

Let’s start by understanding what this pattern does, the **builder pattern** belongs to the **creational patterns** family, meaning that it deals with object creation mechanisms. The builder pattern allows for the separation of a complex object from its representation by using a step by step approach so that the same construction process can create different representations.

In the following example, we are going to use the builder pattern to help us solve a problem about figurines. Figurines fall under several categories, in this example we are going to focus on 3 of them: *Chibi* (*Figure 3*), *Monster* (*Figure 1*) and *Humanoid* (*Figure 2*) figurines. All these types of figurines have shared components, they all have legs, arms, head, ...  but these change between them, for example, *Chibi* figures are characterised by having a big head relatively to their body, whereas *Monster* figurines are the total opposite.



Figure 1 - Monster figurine example

Figure 2 - Humanoid figurine example

Figure 3 - Chibi figurine example

## Problem

Knowing this, our figurine object is going to be very complex with lots of fields and nested objects. This type of initialization would end up being inside a gigantic constructor with lots of parameters or scattered all over the client code, which by itself, it’s an even worse implementation.

## Solution

To solve this problem, we implement the **builder pattern**, meaning that we extract the construction code out of the figurine class and move it to separate objects. These new objects are going to be our builders, responsible for each figurine type. Let’s start by building our *UML* diagram:

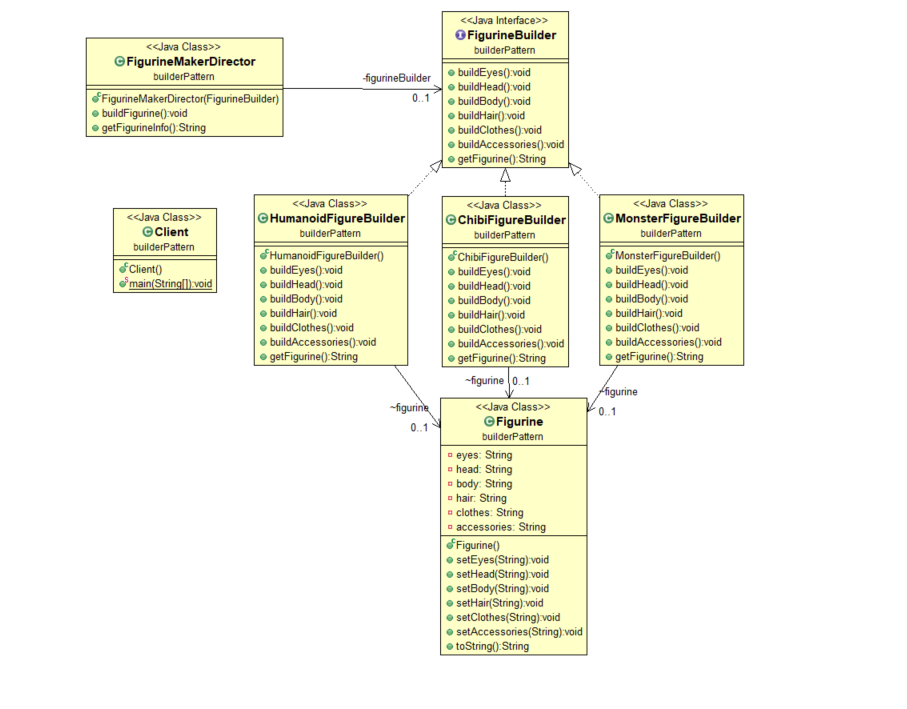


Figure 4 - Figurine *UML* solution

Since our builders implement the same methods, we create a “general” **Figurine Builder** interface where we store all methods. Like explained above, we create the figurine class, but all its construction code is implemented in the builders. Our **FigurineMakerDirector** is the going to be the one responsible for executing the building steps whereas each builder (**HumanoidFigureBuilder**, **ChibiFigureBuilder** and **MonsterFigureBuilder**) tells him how to do so. All of this is then used by our **Client**.

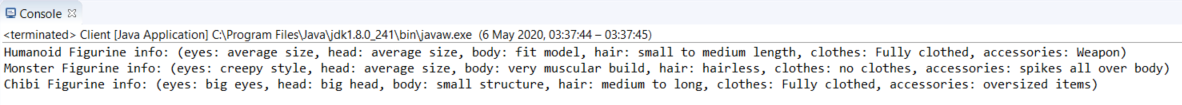


Figure 5 - Output Example

## Advantages

* Immutable objects can be built without using a lot of complex logic in the building process.
* Helps minimizing the number of parameters in the constructor so there is no need to pass a null as an optional parameter in the constructor.
* Complex construction code is isolated from the business logic.
* Allows for control oversteps of construction process.

## Disadvantages

* We need to create a separate builder for each type of figurine, so if we were to make a builder for every type of figurines that exist, it might not be the best option.
* Dependency injection can be less supported.
* Builder classes need to be mutable.

## References

* <https://www.youtube.com/watch?v=9XnsOpjclUg>
* <https://refactoring.guru/design-patterns/builder>
* <https://en.wikipedia.org/wiki/Builder_pattern>
* <https://www.tutorialspoint.com/design_pattern/builder_pattern.htm>
* <https://sourcemaking.com/design_patterns/builder>
* <https://www.geeksforgeeks.org/builder-design-pattern/>
* <https://medium.com/@andreaspoyias/design-patterns-a-quick-guide-to-builder-pattern-a834d7cacead>

# Observer Pattern

## Introduction

## Problem

## Solution

## Advantages

## Disadvantages

## References