|  |  |  |
| --- | --- | --- |
| Design Patterns | March 20  2016 | |
| Monica Stoica  Rosen Danev  House building  Application | | Builder pattern |

Table of Contents

[About the pattern 2](#_Toc446260195)

[Introduction 3](#_Toc446260196)

[UML Diagram 3](#_Toc446260197)

[User interface 3](#_Toc446260198)

[Reusability 3](#_Toc446260199)

[Maintainability 4](#_Toc446260200)

[Extensibility 4](#_Toc446260201)

# About the pattern

The pattern used to develop this software application is the builder pattern. The purpose of the builder patterns is to separate the construction of a complex objects from its representations so that the same construction process can create different representations. The Builder constructs the product step by step. In our example, the pattern is used to create houses. A house typically consists of floors, bathrooms, bedrooms and a specific feature (e.g the vacation house has a pool while the cabin has a fireplace). The construction of the item is the same while the final product is different. The pattern consists of:

-***Builder*** (e.g. IHouseBuiler) interface which is in charge of creating pars of a Product object

***- ConcreteBuiler***(e.g.VacationHouseBuilder) class constructs and puts together parts by implementing the interface.

***- Director*** (e.g. ArchitectDirector) class constructs the complex object using the Builder interface

**- Product** (e.g. House) represents the complex object that is being built.

***Advantages:***

When using the builder pattern, the objects can have different internal representation. For example, the extra property of our house is different for all our concrete builders. It is also possible to control the steps of the construction process – first build the foundation and in the end the roof. If the number of objects is very big, it is easier to maintain the code and only the final product. The pattern provides encapsulation; therefore the fully constructed object will be available to the client.

***Disadvantages:***

The main disadvantage of the builder pattern is that when a new type of product is added, a new concrete builder has to be added. Moreover, if the product is very complex then code duplication will be high.

***Related patterns:***

The builder pattern is related to the abstract factory pattern because both of them separate system from product creation and representation. However, in the abstract factory the application calls the factory methods directly. The factory pattern can be used to create different implementations of the same interface but the Builder is tied up with the class that implement it and returns objects of outer class.

# Introduction

The following document analyses the three main characteristics of the pattern used to develop an application that allows the user to build and buy a house. The user can choose from 4 different types of houses: vacation house, family house, cabin and country-side house. He/she selects the house and buys it. The software will display information about the purchased house such as price, number of bedrooms and specific features. A pattern is general reusable solution to a commonly occurring problem within a giving context. The purpose of using patterns is to speed up the developing process and helps preventing issues that can cause major problems.

# UML Diagram

# D:\Documents\2015-2016\Block7\DPR\DPR\Builder\HouseBuilder\Builder pattern UML.jpg

# 

# User interface

The user can choose from 4 different types of houses: vacation house, family house, cabin and country-side house. He/she selects the house and buys it. The software will display information about the purchased house such as price, number of bedrooms and specific features.

# C:\Users\MonicaS\Desktop\housebuilder.png

# 

# Reusability

One of the main reasons why patterns are useful is because they can be easily reused without changing the code. Considering our above described application, we can attest that the pattern is reusable. The factory pattern can be reused for different types of builders to create different products.

# Maintainability

Because the classes are not tightly coupled and the pattern is quite small, the system is easily maintained. When a system is easy to maintain it means that new features can be added. However, if the user decides to extend the pattern and add more concrete builders, it will become difficult to maintain. Also, if the main product is very complex, it will take quite some time to maintain all the classes.

# Extensibility

New functionality can be provided by adding new code without changing the initial one. Therefore, there is no need to worry about bugs or causing problems. The current pattern can be easily extended but it does depend on one interface meaning that the new concrete product has to have a similar behavior. There can be as many concrete products as wanted but the code redundancy will be very high.