|  |  |  |
| --- | --- | --- |
| Design Patterns | March 29  2016 | |
| Monica Stoica  Rosen Danev | | Adapter 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 adapter pattern. The purpose of the adapter patterns is to help two incompatible interfaces work together. Therefore, an interface of an existing class can be used from another interface. The pattern consists of:

-***ITarget –*** interface that the client wants to use

-I***Adaptee***- defines an existing interface that needs adapting

- ***Adapter-*** adapts the interface IAdaptee to the ITarget interface

­

***Advantages:***

When using the adapter pattern, the design of the software program is more flexible. Also, it is very useful when using APIs with different interfaces without breaking existing implementations.

***Disadvantages:***

The main disadvantage of the adapter pattern is in case the adaptee and the target do not have much in common. Then, there are required many adaptions to get the desired result. To continue with, if the adaptee and the target are very complex then the duplication of code is inevitable.

***Related patterns:***

The adapter pattern is related to the decorator pattern. Using the decorator pattern, you can add more behaviours to an object without modifying the existing cod. However, a decorator works with recursive composition which is not possible with the adaptor pattern.

# Introduction

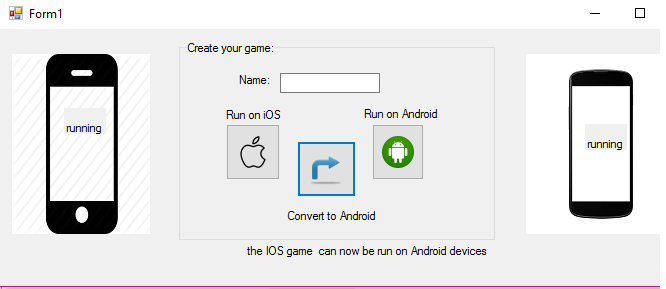
The following document analyses the three main characteristics of the pattern used to develop an application that allows the user to convert an plication for ios to an application for android. To simplify the process, we used strings instead of real applications. The user can enter the name of the application and can run it on both iPhone and Samsung. The ‘application’ (string) can also be adapted to Android phones. 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



# 

# 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.