|  |
| --- |
| Fontys ICT |
| Decorator pattern |
| Design patterns |

|  |
| --- |
| Jan-Niklas Schneider, Georgiana Manolache  9-20-2016 |

# Introduction

The goal of this document is to give an overview of the decorator pattern by giving an example implementation which displays a simplified pizza delivery service called *PizzaMe*. Furthermore, reusability, extensibility, and maintainability of this pattern are elaborated. Also, the implementation, its unit test and graphical user interface (GUI) are reviewed.

# Decorator pattern

The decorator pattern is a software design pattern which allows an object’s functionality to be extended during runtime. This allows the object to be changed when it’s being used compared to inheritance where functionality can be changed only statically, during compile time. Hence, the main goal of this pattern is to add additional responsibilities to an object dynamically.

The figure below depicts an UML diagram of the decorator pattern. The pattern consists of four classes.

Firstly, A *Component* interface which objects implement that can have responsibilities to them dynamically.

Secondly, A *ConcreteComponent* that represents a derived object of the *Component* interface that can add responsibilities dynamically.

Third, the *Decorator* holds a reference to the *Component* and implements its interface.

Lastly, the *ConcreteDecorator* which adds a certain behavior.

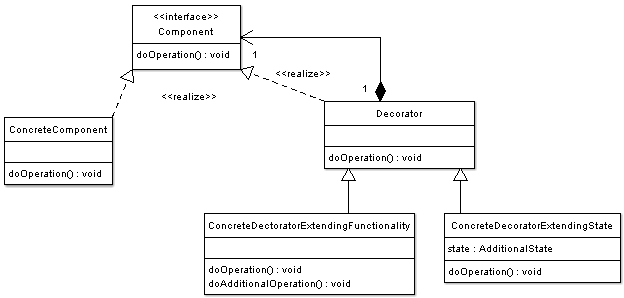


Figure 2‑1: UML diagram of decorator pattern

# Implementation

## Explanation of classes

## Features

# Design choices

# Graphical User Interface

# Unit tests

# References

OODesign. (September 2016). *Decorator Pattern*. Von OODesign.com: http://www.oodesign.com/decorator-pattern.html abgerufen

SourceMaking. (September 2016). *Decorator*. Von SourceMaking.com: https://sourcemaking.com/design\_patterns/decorator abgerufen