|  |
| --- |
| Fontys ICT |
| State pattern |
| Design patterns |

|  |
| --- |
| Jan-Niklas Schneider, Georgiana Manolache  10-4-2016 |

Contents

[1 Introduction 3](#_Toc463386967)

[2 Iterator pattern 3](#_Toc463386968)

[3 Implementation 4](#_Toc463386969)

[3.1 Explanation of classes 4](#_Toc463386970)

[3.2 Features 5](#_Toc463386971)

[4 Design choices 5](#_Toc463386972)

[5 Graphical User Interface 6](#_Toc463386973)

[6 Unit tests 7](#_Toc463386974)

[7 References 7](#_Toc463386975)

# Introduction

The goal of this document is to give an overview of the state pattern by giving an example implementation which features a simplified industrial 3D print process. Furthermore, reusability, extensibility, and maintainability of this pattern are elaborated. Also, the implementation, its unit test and graphical user interface (GUI) are reviewed.

# State pattern

The state pattern is a software design pattern which describes an object-oriented state machine. The pattern has two application areas. Firstly, an object which must change its behavior at run-time depending on a state. Secondly, the pattern is utilized in applications that uses a significant amount of case statements that vector flow based the applications state (SourceMaking, 2016).

The figure below depicts an UML diagram of the state pattern.

Initially, a *Context* class is created which is a single point of entry. *Context* holds a *State* and is able to either set or get a state.

Secondly, an *State* interface is defined which sets the actions of its derived classes.

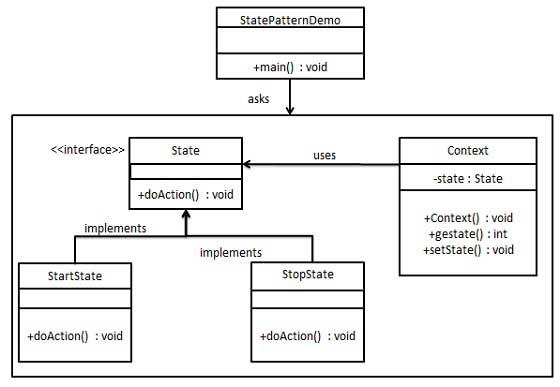


Figure 2‑1: UML diagram of state pattern (TutorialsPoint, 2016)

# Implementation

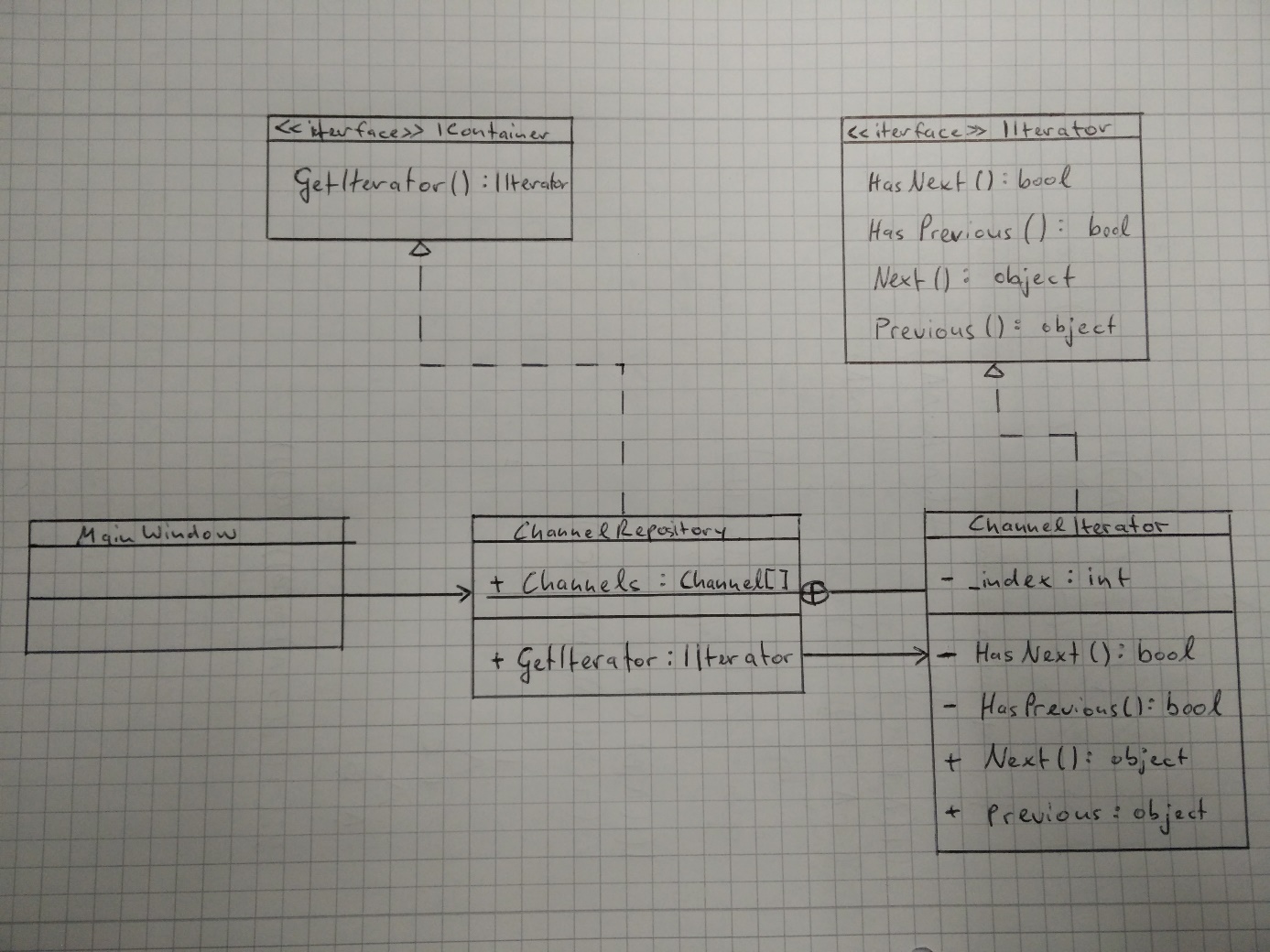
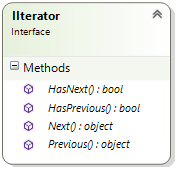


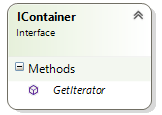
Figure 3‑1: UML diagram of the state pattern

## Explanation of classes

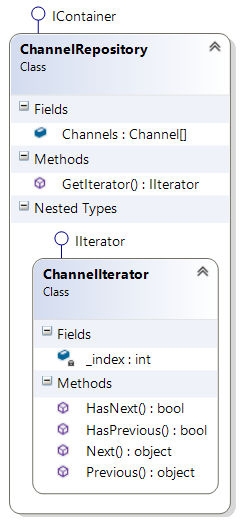
1. *IIterator* is an interface with four methods *HasNext(), HasPrevious(), Previous()* and *Next()*. These methods define the iterator protocol for all derived classes.



1. *IContainer* is an interface with one method *GetIterator()* which returns the implementation of the *IIterator* interface which is called *ChannelIterator.*



1. *ChannelIterator* implements the *IIterator* interface. It is defined as a nested private class inside *ChannelRepository. ChannelRepository* is the concrete implementation and sets the iterator protocol.

*ChannelRespository* has one method *GetIterator()* which returns the iterator protocol. Further, it holds a data structure.

## Features

The application has a simple and straightforward user interface. Users are presented a simplified TV which demonstrates the iterator pattern by switching through a TV’s channels.

# Design choices

The implementation of the iterator pattern has been done with regard to reusability, extensibility, and maintainability.

The **reusability** of the pattern is very high. The implementation of *ChannelIterator* is independent of the type of the data structure given, hence, it can be reused in any other case. The given interface *IIterator* defines a general approach excellent in terms of reusability, for instance, return types are objects which allows easy reuse of the interface as well concrete class.

In terms of **maintainability** the pattern is easy to maintain since the data structure inside *ChannelRepository* and the iterator protocol in *ChannelIterator* are separated units which can be easily changed without affecting neither. Meaning that the data structure is independent of the iterator protocol or vice versa. Further, this eases unit tests.

The iterator pattern shows decent **extensibility**. While the iterator protocol can be easily extended and modified, they can be also added by implementing *IIterator.* Additionally, *IContainer* can offer multiple methods which describe different iterator protocols.

# Graphical User Interface



Figure 5‑1: Graphical user interface

The figure above depicts the user interface where red numbers indicate functionality or controls. More precisely these are:

1. A TextBlock which displays the name of the current TV channel.
2. Two buttons which allow changing channels. “+” is channel up, “-” is channel down.

# Unit tests

To assert correct behavior of the iteration protocol one unit test has been created, namely *IteratorValuesValid\_Test.* The test cycles through all channels from first to last and back to the first.Subsequently, the test ran successful.



Figure ‑: Unit test result

# References

SourceMaking. (2016, September). *Iterator*. Retrieved from SourceMaking.com: https://sourcemaking.com/design\_patterns/iterator

Tutorialspoint. (2016, September). *Design Patterns - Iterator Pattern*. Retrieved from Tutorialspoint.com: https://www.tutorialspoint.com/design\_pattern/iterator\_pattern.htm