Fontys ICT

ITERATOR PATTERN

Design patterns

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

Contents

1 Introduction

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

2 Iterator pattern

The iterator pattern is a software design pattern which gives access to data structures without exposing its internal structure. The pattern takes responsibility for access and traversal of the aggregate object and defines an iterator object that holds a standard traversal protocol (SourceMaking, 2016).

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

Firstly, an interface *Iterator* interface is defined which describes the iterator protocol. In the example, the *NameIterator* implements this interface.

Secondly, a *Container* interface provides the iterator to derived classes. *NameRepository* is the data structure and implements the *Container* interface.

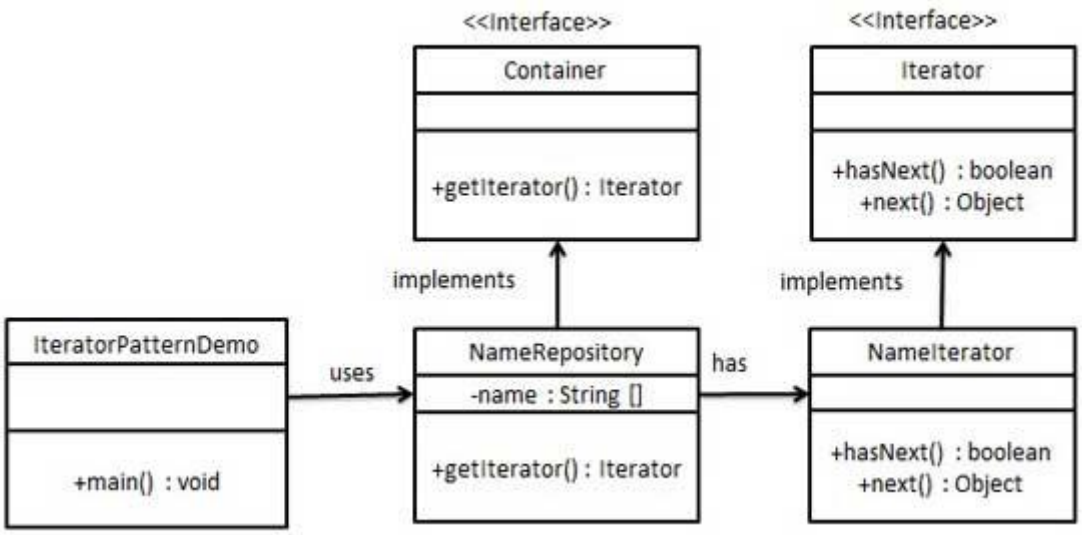


Figure 2-1: UML diagram of iterator pattern (Tutorialspoint, 2016)

3 Implementation

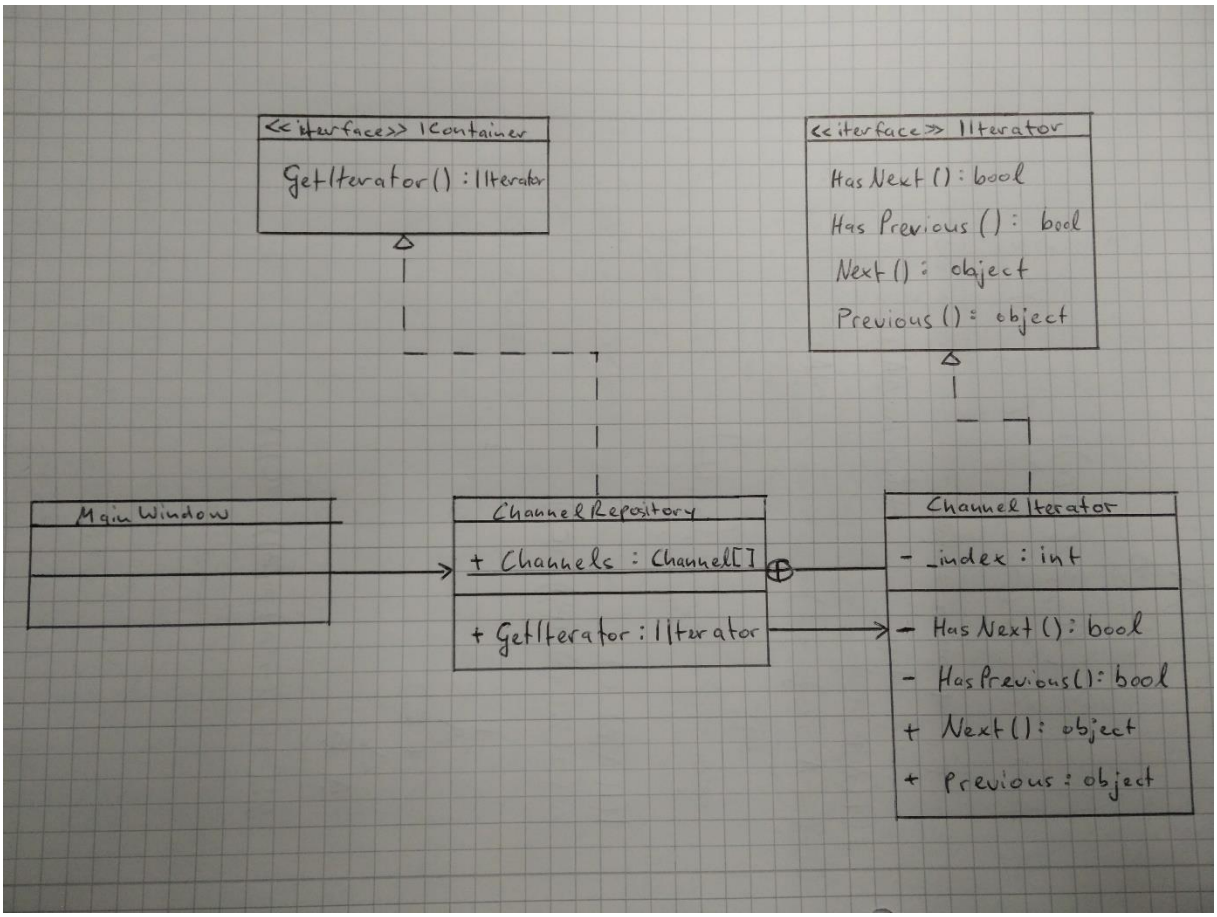
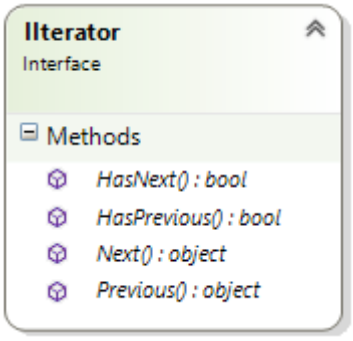


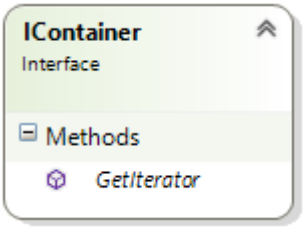
Figure 3-1: UML diagram of the iterator pattern

3.1 Explanation of classes

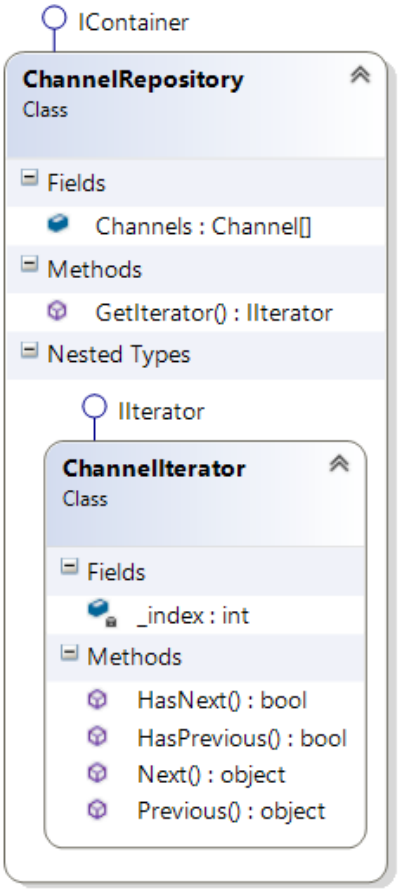
1. *IIterator* is an interface with four methods *HasNext(), HasPrevious(), Previous()* and *Next()*. These methods define the iterator protocol for all derived classes.



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



3. *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.



3.2 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.

4 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.

5 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.

6 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.

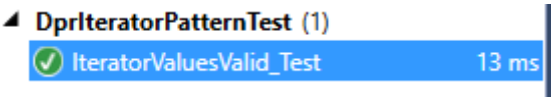


Figure 6-1: Unit test result

7 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