|  |
| --- |
| Fontys ICT |
| Iterator pattern |
| Design patterns |

|  |
| --- |
| Jan-Niklas Schneider, Georgiana Manolache  9-28-2016 |

Contents

[1 Introduction 3](#_Toc462786292)

[2 Factory pattern 3](#_Toc462786293)

[3 Implementation 5](#_Toc462786294)

[3.1 Features 6](#_Toc462786295)

[4 Design choices 6](#_Toc462786296)

[5 Graphical User Interface 7](#_Toc462786297)

[6 Unit tests 7](#_Toc462786298)

[7 References 7](#_Toc462786299)

# Introduction

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

# 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.

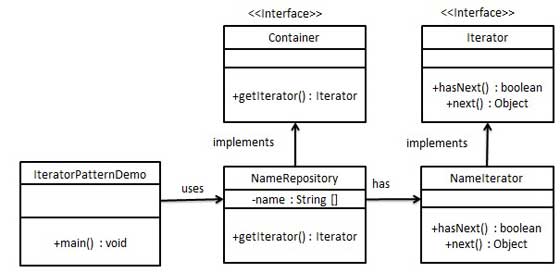


Figure 2‑1: UML diagram of iterator pattern (Tutorialspoint, 2016)

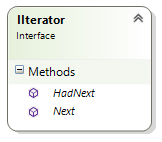
# Implementation

UML

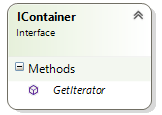
Figure 3‑1: UML diagram of the iterator pattern

## Explanation of classes

1. *IIterator* is an interface with two methods *HadNext() and Next()*. These methods define the iterator protocol for all derived classes.

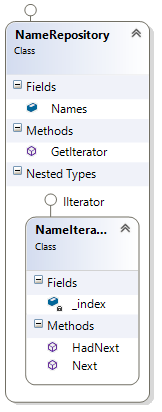


1. *IContainer* is an interface with one method *GetIterator()* which returns the implementation of the *IIterator* interface which is called *NameIterator.*



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

*NameRespository* 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. User can select a store and a car of choice. Selection can be cleared up.

# 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 *NameIterator* is independent of the type of the data structure given, hence, it can be reused in any other case.

In terms of **maintainability** the pattern is easy to maintain since the data structure inside *NameRepository* and the iterator protocol in *NameIterator* separated units which can be easily changed without affecting neither.

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.

Something about private class maybe.

# Graphical User Interface

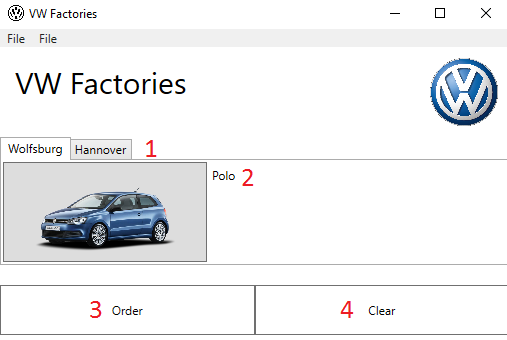


Figure 5‑1: Graphical user interface

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

1. Factories from where a car can be purchased (assembled).
2. Choose from one of the cars using car specific icon.
3. Order car.
4. Clear order.

# Unit tests

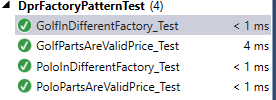
For each implemented Component unit tests have been defined to assert correct behavior. The test validate correct price of the components created in the factory, further, products have been created in different factories. Consequently, all test ran successfully.

Figure 6‑1: Unit test results

# References

*Abstract factory pattern*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Abstract\_factory\_pattern

Data & Object Factory, LLC. (n.d.). *Abstrcat factory design pattern*. Retrieved from DoFactory: http://www.dofactory.com/net/abstract-factory-design-pattern