Chapter 1

Introduction to design patterns

1.1 Safe choice in presence of polymorphism

The visitor design pattern is a way of separating an algorithm from an object structure on which it operates. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures. It is one way to follow the open/closed principle.

1.1.1 Exercise 1

Design

In this assignment you will apply the visitor design pattern to print the concrete type of a number.

- **INumber** interface: contains the method visit, which returns a void and takes an INumberVisitor as input.
- **INumberVisitor** interface: contains two methods with almost identical signature, namely onMyInt and onMyFloat, both returning void. The former takes a MyInt, the latter takes a MyFloat as an argument.
- **MyFloat** implements INumber. Its visit calls onMyFloat on the given visitor-object, with 'this' as input.
- **MyInt** implements INumber. Its visit calls onMyInt on the given visitor-object, with 'this' as input.
- **NumberVisitor** implements INumberVisitor. It prints "Found float" on onMyFloat and "Found int" on onMyInt.

Test

In the main program: define a NumberVisitor and use it to visit an instance of a MyInt-object.

1.1.2 Exercise 2

Design

In this assignment you will make another implementation of the visitor pattern. This time you will focus on music.

- **ISong**, an interface with a visit-method, which takes an `IMusicLibraryVisitor`.
- **Jazz** implements ISong. Its visit calls onJazz on the given visitor-object, wiht 'this' as input.
- **HeavyMetal** implements ISong. Its visit calls onHeavyMetal on the given visitor-object, wiht 'this' as input.
- **IMusicLibraryVisitor** interface: contains two methods with almost identical signature, namely onHeavyMetal and onJazz, both returning void. The former takes a HeavyMetal song, the latter takes a Jazz song as an argument.
- **MusicLibraryVisitor** implements IMusicLibraryVisitor and is composed of two Lists, one for jazz and one for heavy metal. It adds the heavy metal song to the corresponding list on onHeavyMetal, and the jazz song to the corresponding list on onJazz.

Test

Write a program that adds some jazz songs and some heavy metal songs to a list and visit each song in the list, using a MusicLibraryVisitor. Eventually, print the amount of heavy metal song and jazz songs in the MusicLibraryVisitor.

1.1.3 Exercise 3

In this assignment you will make another implementation of the visitor pattern. This time you will focus on options.

Design

- **IOption** ⟨**T** ⟩ a parametic interface: contains the method visit (parametric with type U), which returns an object of type U and takes an IOptionVisitor as input.
- **IOptionVisitor** ⟨**T, U** ⟩ a parametric interface: contains two methods with almost identical signature, namely onSome and onNone, both returning an object of type U. The former takes a T-value, the latter takes nothing as an argument.
- **Some** implements IOption. Its visit calls onSome on the given visitor-object, with this.value as input, where value is a field of such class, initialized properly.
- **None** implements IOption. Its visit calls onNone on the given visitor-object.
- **IntPrettyPrinterIOptionVisitor** implements IOptionVisitor, where T equals Integer and U equals String. It returns "I am nothing ..." on onNone and the value converted to string on onSome.

Test

In the main program: define a IntPrettyPrinterIOptionVisitor and use it to print the value of a Some containing number 5.

1.1.4 Exercise 4

In this assignment you will make another implementation of the visitor pattern. This time you will focus on options combined with only lambda's.

Design

- **IOption** ⟨ **T** ⟩ a parametic interface: contains the method visit (parametric with type U), which returns an object of type U and takes two lambda's: the first one called onNone with type void to U, the second called onSome with type T to U.
- **Some** implements IOption. Its visit calls the onSome function provided in the arguments, with this.value as input, where value is a field of such class, initialized properly.
- **None** implements IOption. Its visit calls the onNone function provided in the arguments.

Test

In the main program: define a IntPrettyPrinterIOptionVisitor and use it to print the value of a Some containing number 5. Define a Some ccontaining number 5, and visit it with the following functions as input:

- A function that takes nothing and returns the string "I am nothing".
- A function that takes an integer and returns its string representation.

1.1.5 Exercise 5 - Combining exercises 3, 4

Recycle all declarations from exercise 3, and extend them with a new visitor that accepts two functions (see exercise 4) each specifying the behaviours for objects of type IOption: our Some and None. Use them properly.

Chapter 2

Iterating collections

According to wikipedia[1]: An iterator is an object that enables a programmer to traverse a container, particularly lists. Various types of iterators are often provided via a container's interface. Though the interface and semantics of a given iterator are fixed, iterators are often implemented in terms of the structures underlying a container implementation and are often tightly coupled to the container to enable the operational semantics of the iterator. Note that an iterator performs traversal and also gives access to data elements in a container, but does not perform iteration (i.e., not without some significant liberty taken with that concept or with trivial use of the terminology). An iterator is behaviorally similar to a database cursor. Iterators date to the CLU programming language in 1974.

2.1 The iterator interface

In this exercise you will study the interator interface. Between languages, a clear difference is seen in the naming of methods. However the behaviour is very much the same. You can experiment with different interfaces, so implement one or more of the following iterator interfaces. With that interface you can implement different concrete implementations.

2.1.1 Gang of four-iterator

Start with the one defined by the industry-standard book of the gang of four[2]. Define a **GofIterator** $\langle \mathbf{T} \rangle$ interface: contains the methods:

[1] Wikipedia: Iterator

[2] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design patterns: Abstraction and reuse of object-oriented design. Springer.

1. first(), which returns void
2. next(), which returns void
3. isDone(), which returns boolean
4. currentItem(), which returns the element (of type T)

2.1.2 Idiomatic C-sharp

For C-sharp there are two idiomatic iterators: Iterator and TraditionalIterator.

Iterator ⟨**T**⟩ interface: contains the methods:

1. GetNext(), returns Option⟨T⟩
2. Reset(), returns void

TraditionalIterator ⟨**T**⟩ interface: contains the methods:

1. MoveNext(), returns boolean
2. Current(), returns T

2.1.3 Idiomatic Java

For idiomatic Java define: **Iterator** ⟨**E**⟩ interface: contains the methods:

1. hasNext(), returns boolean
2. next(), returns E

2.2 Implementations

After you have defined one or more interfaces, you will now implement deferent concrete implementations.

2.2.1 Exercise 1

Design

Define a **NaturalNumbers** class that implements one of the iterator interfaces: contains the attributes:

1. **FIRST**, an integer set to -1.

2. **current**, an integer.

Implement the methods of the interface, in such a way that all natural numbers can be traversed.

Test

Write a test program that iterates all natural numbers, printing each one when visited.

2.2.2 Exercise 2

Design

1. **InfiniteLoopListIterator** ⟨**T**⟩ class that implements one of the iterator interfaces: contains the attributes:

   (a) list, of type List¡T¿

   (b) index an int

   Add a constructor which receives a List. Now implement the iterator, in such a way that the list is iterated. When the end is reached, iteration will continue from the front again, thus iterating infinitely.

Test

Write a test program that iterates a list (that you may build), printing each element when visited.

2.2.3 Exercise 3

Design

1. **Map** ⟨**T, U**⟩ class that implements one of the iterator interfaces: contains the attributes:

   (a) decoratedCollection of type iterator ⟨T⟩

   (b) f of type **Function** ⟨**T, U**⟩

   Add a constructor which receives an Iterator ⟨T⟩ and a **Function** ⟨**T, U**⟩. Now implement the iterator, in such a way that f is applied when the current item is called.

Test

Write a test program that iterates an infiniteLoopList which is mapped with a plus +1 function, printing each element when visited.