Iterator Report

In this report I will cover the inner workings of the iterator. The primary name of this behavioral pattern is iterator and it also known as cursor. It belongs in the Behavioral patterns section of design. Its primary intent is to provide the program where it’s implemented a way of getting elements from an aggregate object without showing off the inner workings of the data itself

The motivation behind the iterator is as follows. In order for the iterator to be useful we must consider how an object like a list will be traversed and it’s internal workings no exposed. This is the purpose of the iterator. You can use an iterator to traverse a list in a normal way (1-20 elements for example) or even in different ways (Such as starting from the 5th element and going only until the 10th.) By using an iterator you also don’t bloat the lists’ interface thus cleaning up the code and efficiency in it as well as taking over the actions of access and traversal away from the list object.

There are 3 primary cases of where you would use it. The first case is to access an object’s contents without exposing its internal representation. The second case is to support multiple traversals of the object. Finally it’s to provide a “All-in-one” solution for traversing all kinds of structures.

In terms of real life examples for an iterator there are plenty. I will be using small talk to give you an example of a real life usage for it. Iterators are object-oriented thus they can only be used in such languages that support it. Smalltalk utilizes iterators in all it’s major collection classes (Such as bag, set, etc) and how it works is simple. Each element is saved inside the block of an object in a local variable and then the block is executed. Small talk also has a ReadStream object which acts essentially as an iterator as well as an external one for all the collection classes mentioned above (Since it outputs what was in the objects.) However in small talks’ example there is no standard external iterator for all non- sequential collection, such as set or dictionary.

The final thing I will cover is related patterns. There are 3 patterns that are related to the iterator pattern. The first one is composite. The composite pattern uses iterators to traverse itself. The second pattern is the factory method. In order to streamline the creation of the appropriate subclass polymorphic iterators must be used. Finally memento uses iterators as well. A iterator can use a memento to save the state of the iteration itself which can make things easier later down the road in terms of access and traversal.