Iterator ID

Student:

Izabela Kuźniar

Teacher:

Andrea Corradini

Course:

Software Design Patterns

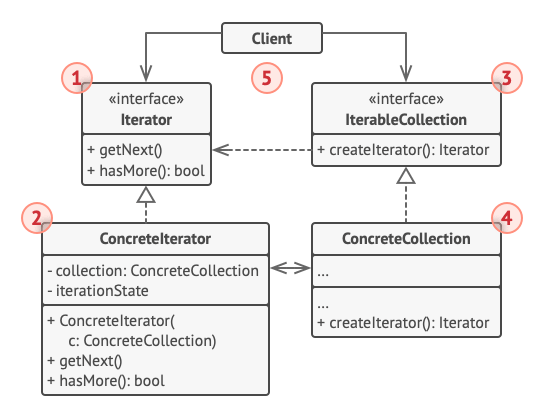
# Name and category

Iterator is a behavioral design pattern.

# Intent:

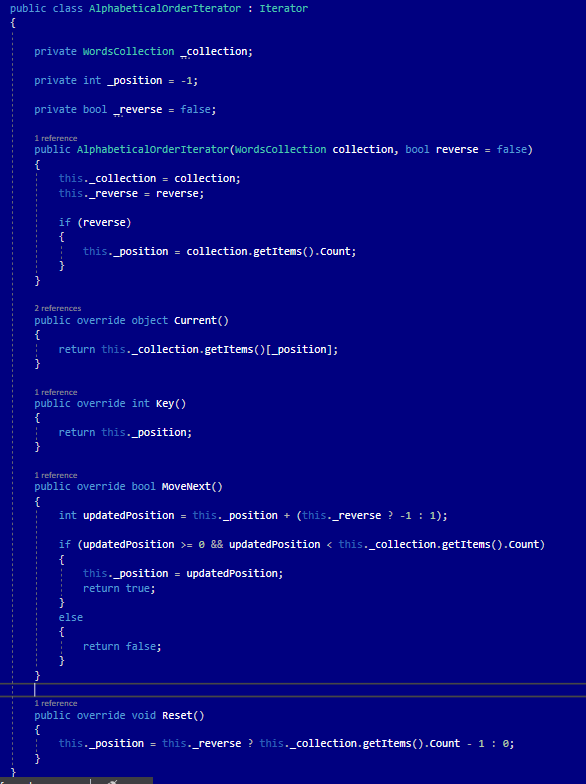
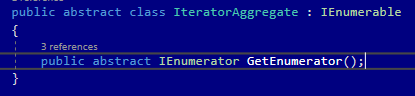
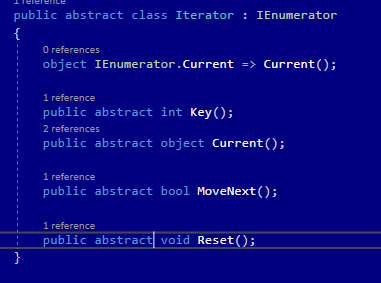
Iterator gives ability to traverse elements from a collection without exposing its underlying representation (list, stack, tree, etc.)

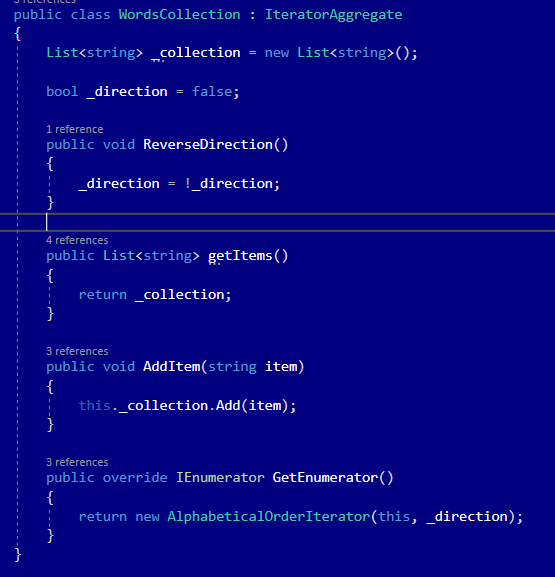
# Structure as a UML class diagram

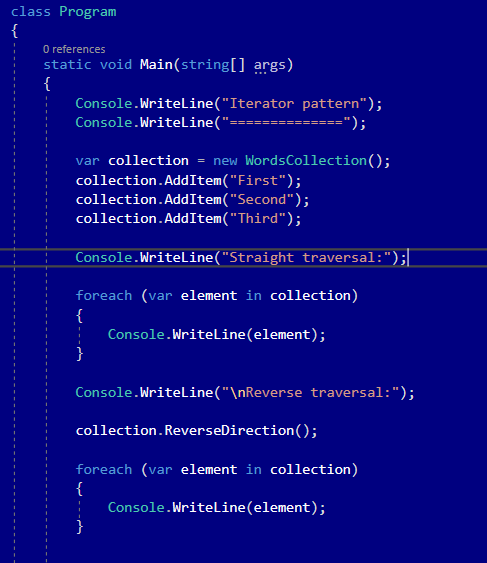


1. The Iterator interface declares the operations required for traversing a collection: fetching the next element, retrieving the current position, restarting iteration, etc.
2. Concrete Iterators implement specific algorithms for traversing a collection, The iterator object should track the traversal progress on its own. This allows several iterators to traverse the same independently of each other.
3. The Collection interface declares one or multiple methods for getting iterators compatible with collection. Note that the return type of the methods must be declared as the iterator interface so that the concrete collections can return various kinds of iterators.
4. Concrete Collections return new instances of a particular concrete iterator class each time the client requests one.
5. The Client works with both collections and iterators via their interfaces. This way the client isn’t coupled to concrete classes, allowing to use various collections and iterators with the same client code.

# Implementation:







# Consequences:

Benefits:

* Single Responsibility Principle. Ability to clean up the client code and the collections by extracting bulky traversal algorithms into separate classes.
* Opened/Closed Principle. Ability to implement new types of collections and iterators and pass them to existing code without breaking anything.
* Ability to iterate over the same collection in parallel because its own iteration style is held by each iterator object.
* Similarly because of it we can delay an iteration and continue when needed.

Drawbacks:

* Can be an overkill while applying to application that only uses simple collections.
* Using an Iterator might be less efficient than going through elements of some specialized collections directly.

# Related patterns

1. Ability to use Iterators to travers Composite trees.
2. Ability to use Factory Method along with Iterator to let collection subclasses return different types of iterators that are compatible with the collections.
3. Ability to use Memento along with Iterator to capture the current iteration state and roll back if necessary.
4. Ability to use Visitor along with Iterator to traverse a complex data structure and execute some operation over its elements, even if they all have different classes.