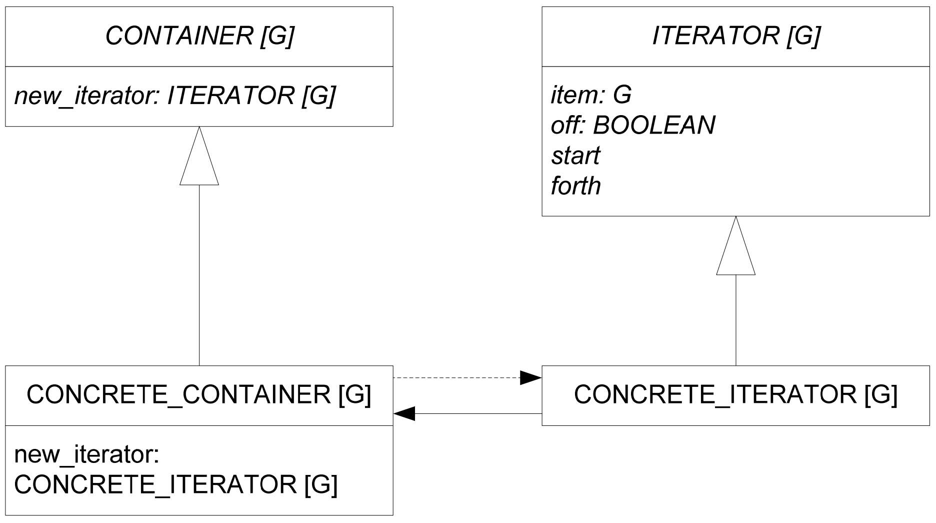
Exercise 3: Contract a pattern

The *Iterator* design pattern provides a way to access the elements of a container without exposing its underlying representation [1]. In this assignment we consider a *passive linear* iterator. "Passive" means that rather than supplying an operation to the iterator and letting it be executed over all the elements of the container, the client itself controls the iteration process, initiating its every step. "Linear" means that at each point in the iteration process there exists only one possibility for the next step until the iteration finally goes beyond the last element. The structure of the pattern can be described by the following class diagram:



An important concern regarding passive iterators is: what happens if the container contents changes while iteration is in progress? Informal specification of an iterator usually says that it should visit all elements of the container exactly once. However, in presence of contents changes, such a specification becomes ambiguous and difficult to enforce.

One possible solution is to prohibit changes in the container contents while an iteration is in progress. If you look at iterator implementations in programming languages that don't support Design-by-Contract, you will often see comments like: "If the container is changed while the iteration is in progress, then the result is unpredictable!" In this assignment you are asked to enforce this restriction in a formal way, that is, to design contracts for the Iterator pattern in such a way that container contents changes will be permitted only if no iterations are currently in process.

To Do

1. Introduce an abstract class MY_LIST [G] that represents the abstract container. The class should provide an interface to determine, whether the list is empty, to add elements to the

front and to remove elements from the front. Implement two different representations of lists (concrete containers): MY_LINKED_LIST [G] and MY_ARRAYED_LIST [G]. The first one should implements singly-linked lists and the second one - store list elements in an array.
2. Introduce an abstract class MY_ITERATOR [G] that provides an interface for traversing lists (see class diagram above). Implement concrete iterators MY_LINKED_ITERATOR [G] and MY_ARRAYED_ITERATOR [G].
3. Add features and contracts to your classes to ensure that whenever there is an active iterator attached to a list (that is, an iterator that is not "off"), nothing can be added or removed from the list.

References

1. *Gamma et. al*. Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995, ISBN 0201634988.