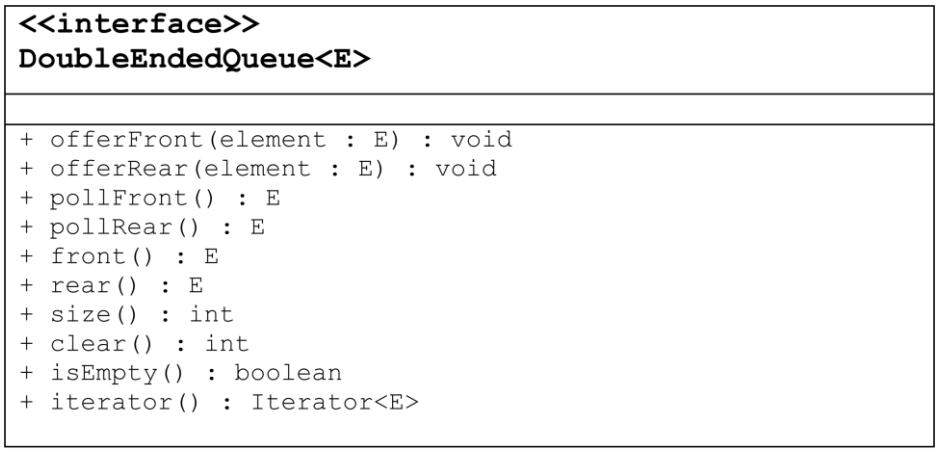
– Double ended Queue implementations

A Double-Ended Queue (often also called a Deque), is a linear collection that allows insertion and deletion at both ends. The interface above called *DoubleEndedQueue<E>* available from Blackboard, is an abstract data type which describes operations that the data structure should have.



- It has *offer* and *poll* methods for adding or removing an element from the front and rear of the double ended queue.
- *front* and *rear* for obtaining the elements (but not removing) at the front and rear of the double ended queue.
- *isEmpty* for checking whether any elements are in the double ended queue.
- *clear* for clearing out all elements in double ended queue.
- *size* for returning the number of elements currently in the double ended queue.
- *iterator* which can be used to contiguously iterate over the elements in the from front to back.

Implement the *DoubleEndedQueue* abstract data type using **two** different underlying data structures. One called *LinkedDoubleEndedQueue<E>* which uses a suitable linking structure, and one called *ArrayDoubleEndedQueue<E>* which uses a suitable underlying array structure. Implement the data structure yourself, **do not** use extra Java *Collection* classes.

For efficiency all double ended queue operations should be *O(1).* Add a suitable *toString()* to your data structure implementations which returns a string representation of each implementation for testing.

Further enhance your classes to make sure all methods are safe from unexpected events using appropriate exception handling (example, if user code tries to remove something from the double ended queue when empty, it should throw an appropriate exception) . Create a test class with a suitable *main* method which effectively tests all operations on one or both *double ended queue* implementations.

Example Output:

Offer Rear: A, B, C	*Queue: [A, B, C]*	*Offer Rear: D*	*Queue: [A, B, C, D]*
Offer Front: E	*Queue: [E, A, B, C, D]*	*Poll Rear: - returns D*	*Queue: [E, A, B, C]*
Poll Front: - returns E	*Queue: [A, B, C]*	*Front: - returns A*	*Queue: [A, B, C]*
Rear: - returns C	*Queue: [A, B, C]*	*Poll Rear: - returns C*	*Queue: [A,B]*