Iteration (Optional)

Design challenge: Support iteration over stack items by client, without revealing the internal representation of the stack.

- **Java solution**. Make stack implement the **java.lang.Iterable** interface.

Q. What is an `Iterable` ?
A. Has a method that returns an `Iterator`.

Iterable interface

```
public interface Iterable<Item> {
   Iterator<Item> iterator();
}
```

Q. What is an `Iterator` ?
A. Has methods `hasNext()` and `next()`.
Q. Why make data structures `Iterable` ?
A. Java supports elegant client code.

Iterator interface

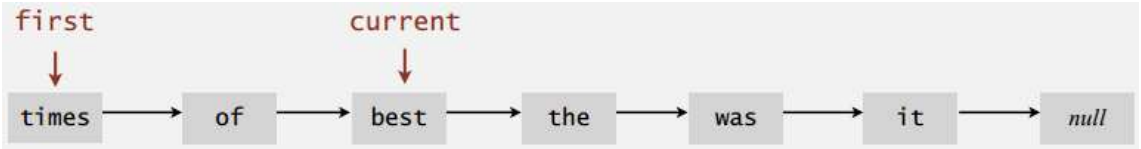
```
public interface Iterator<Item> {
   boolean hasNext();
   Item next();
   void remove();   <-- optional; use at your own risk
}
```

```
import java.util.Iterator;
public class LinkedStack<T extends Comparable<T>> implements Iterable<T> {
   :
   public Iterator<T> iterator(){
      return new ListIterator();
   }

   private class ListIterator implements Iterator<T>{
      private Node<T> curr = topNode;
      public boolean hasNext(){return curr!=null;}
      public void remove(){}
      public T next(){
         T t = curr.data;
         curr = curr.next;
         return t;
      }
   }
}
```

```
Iterator<String> itt = ls.iterator();
while (itt.hasNext())
   System.out.println(itt.next());
```

```
for(String s: ls)
   System.out.println(s);
```



STUDENTS-HUB.com Uploaded By: Jibreel Bornat