```java
package util;

import java.util.ArrayList;
import java.util.List;

/**
 * Represents a LIFO stack of objects where each item has a reference to the next one.
 *
 * @author Kevin Farine, Timothee Van Hove
 */
public class Stack {
    /**
     * The item that is currently on top of the stack.
     */
    private Item topItem;

    /**
     * Returns an iterator pointing to the top item.
     *
     * @return an iterator pointing to the top item.
     */
    public StackIterator getIterator() {
        return new StackIterator(topItem);
    }

    /**
     * Returns an array that represents the current state of this stack starting with the top item.
     *
     * @return An array that represents the state of this stack.
     */
    public Object[] getCurrentState() {
        StackIterator iterator = getIterator();
        List<Object> result = new ArrayList<>();

        while (iterator.hasNext()){
            result.add(iterator.next());
        }
        return result.toArray();
    }

    /**
     * Pushes an item onto the top of this stack.
     *
     * @param object The object to push on top.
     */
    public void push(Object object) {
        topItem = new Item(object, topItem);
    }

    /**
     * Removes the object at the top of this stack and returns it.
     *
     * @return The object at the top of this stack.
     * @throws RuntimeException If the stack is empty.
     */
    public Object pop() {
        if (topItem == null){
            throw new RuntimeException("The stack is empty");
        }
        Object currentTopItem = topItem.value;
        topItem = topItem.next;
        return currentTopItem;
    }

    /**
     * Returns a string representation of this stack.
     *
     * @return The string representation of this stack.
     */
    @Override
    public String toString() {
        StackIterator iterator = getIterator();
```

```
        StringBuilder builder = new StringBuilder("[ ");

        while (iterator.hasNext()){
            builder.append("<").append(iterator.next()).append("> ");
        }
        return builder.append("]").toString();
    }

    /**
     * represents an item in a stack with a value and a reference to the next one.
     *
     * @author Kevin Farine, Timothee Van Hove
     */
    private static class Item{
        /**
         * The value of this item.
         */
        private final Object value;

        /**
         * The next item linked to this one.
         */
        private final Item next;

        /**
         * Item constructor
         *
         * @param value The value of the item.
         * @param next the next item in the stack.
         */
        private Item(Object value, Item next) {
            this.value = value;
            this.next = next;
        }
    }

    /**
     * Iterator which can be used to iterate on the Stack (e.g. in a while loop using hasNext()).
     *
     * @author Kevin Farine, Timothee Van Hove
     */
    public static class StackIterator {

        /**
         * The item on which the iterator is currently pointing.
         */
        private Item item;

        /**
         * Iterator constructor with a reference on the item to point. This constructor is
         * private because the user cannot instantiate an Item outside this class.
         *
         * @param item The item to point on.
         */
        private StackIterator(Item item) {
            this.item = item;
        }

        /**
         * Checks if the next item exists.
         *
         * @return true if the next item exists, else returns false.
         */
        public boolean hasNext() {
            return item != null;
        }

        /**
         * Makes the iterator referencing the next item. The top Item value is returned
         *
         * @return the current top item value
         * @throws RuntimeException if the next item does not exist
```

```
         */
        public Object next() {
            if (!hasNext()){
                throw new RuntimeException("The next item doesn't exists");
            }

            Object currentValue = item.value;
            item = item.next;
            return currentValue;
        }
    }
}
```