Assignment 3

Team members:
1. Clayton Mottley
2. Xavier Gitiaux
3. Anowarul Kabir

Team works:
1. As usual, we first came up with our own solution where everyone's solutions is used to construct the final answer Then we discussed over some additional points.
1.1 Do we need to handle pushing 'null' object in push()?
Ans: No. As that is not in the previous contract.
1.2 Do we need to use custom EmptyStackException instead of IllegalStateException in top and pop?
Ans: No, that will also break the contract. Moreover, ISE is serving the context, as we are calling top and pop before reaching the expected state of the object.
1.3 Do we need to implement clone, toString, equals?
Ans: No, that is not asked in the assignment. It only asked immutable stack implementation given the behavior.
2. Finally, when we merge three solutions, we take Clayton's implementation for the code, Xavier's unit test cases and Anowarul's javadoc.

Here, we have attached the original solutions developed by each.

Clayton’s original solution

```
import java.util.*;
public class Stack {
    private Object[] elements;
    private int size = 0;
    public Stack() {
        this.elements = new Object[0];
    }
    private Stack(Object[] elements) {
        this.elements = elements;
        this.size = elements.length;
    }
    public Stack push(Object e) {
        Object[] newElements = new Object[this.elements.length + 1];
        System.arraycopy(this.elements, 0, newElements, 0, this.elements.length);
        newElements[newElements.length - 1] = e;
        return new Stack(newElements);
    }
    public Stack pop() {
        if (this.size == 0) throw new IllegalStateException("Stack is empty!");
        Object[] newElements = new Object[this.elements.length - 1];
        System.arraycopy(this.elements, 0, newElements, 0, newElements.length);
        return new Stack(newElements);
    }
    public Object top() {
        if (this.size == 0) throw new IllegalStateException("Stack is empty!");
        return this.elements[this.elements.length - 1];
    }
}
```

Xavier’s original solution

```
public class ImmutableStack {
    /*
    Overview: implements a immutable stack
     */
    private Object[] elements;
    private int size = 0;

    private ImmutableStack(int n) {
        this.elements = new Object[n];
    }
    public ImmutableStack(){
        this.elements = new Object[size];
    }

    public ImmutableStack push(Object e){
        /*
        @param element to add
        @return a new stack with all the elements in this.elements plus e at the end
         */
        ImmutableStack newStack = new ImmutableStack(size + 1);
        System.arraycopy(elements, 0, newStack.elements, 0, size);
        newStack.elements[size] = e;
        newStack.size = size + 1;
        return newStack;
    }

    public ImmutableStack pop(){
        /*
        @return a new stack with all elements in this.elements but the last element
        @throw IllegalStateException if this.size == 0
         */
        if (size == 0)
            throw new IllegalStateException("ImmutableStack.pop");
        ImmutableStack newStack = new ImmutableStack(size -1);
        System.arraycopy(elements, 0, newStack.elements, 0, size - 1);
        newStack.size = size -1;
        return newStack;
    }

    public Object gethead(){
        /*
        @return the last element in this.element
        @throw IllegalStateException if this.size == 0
         */
        if (size == 0)
            throw new IllegalStateException("ImmutableStack.pop");

        return elements[--size];
    }
}
```

Anowarul's original solution

```
/**
 * Stack is an unbounded, immutable implementation of "Bloch's Stack example
 * page 60, 3rd edition"
 */
public class Stack {
	private Object[] elements;
	private int size = 0;
	private static final int DEFAULT_INITIAL_CAPACITY = 0;
	/**
	 * Initializes Stack to be empty.
	 */
	public Stack() {
		// EFFECTS: Initializes Stack to be empty.
		this.elements = new Object[DEFAULT_INITIAL_CAPACITY];
	}
	private Stack(Object[] elements) {
		this.elements = elements;
		this.size = elements.length;
	}
	/**
	 * Push an Object at the top of the Stack.
	 *
	 * @param e The Object to be pushed.
	 * @return Stack after e being pushed.
	 * @throws IllegalArgumentException if e is null.
	 */
	public Stack push(Object e) {
		if (e == null)
			throw new IllegalArgumentException("Stack.push: cannot push null
object.");
		Object[] oldElements = new Object[this.elements.length + 1];
		System.arraycopy(this.elements, 0, oldElements, 0, this.elements.length);
		oldElements[this.elements.length] = e;
		return new Stack(oldElements);
	}
	/**
	 * Returns Stack after the top element being popped up.
	 *
	 * @return Stack after top element being popped up.
	 * @throws EmptyStackException if Stack is empty.
	 */
	public Stack pop() {
		// EFFECTS: If Stack is empty throws EmptyStackException, else return Stack
		// after top element being popped up.
		if (size == 0)
			throw new EmptyStackException("Stack.pop: Stack is empty.");
		Object[] newElements = new Object[this.elements.length-1];
		System.arraycopy(this.elements, 0, newElements, 0, newElements.length);
		return new Stack(newElements);
	}
	/**
	 * Returns the top element of the Stack.
	 *
	 * @return Top element of the Stack.
	 * @throws EmptyStackException if no element at the top.
	 */
	public Object top() {
		// EFFECTS: If Stack is empty throws EmptyStackException, else return top
Object
		// of the Stack.
		if (size == 0)
			throw new EmptyStackException("Stack.top: No element at the top, Stack
is empty.");
```

```
		return this.elements[this.elements.length - 1];
	}
	/**
	 * Returns the current size of the Stack.
	 *
	 * @return The size of the Stack.
	 */
	public int getSize() {
		return size;
	}
}

class EmptyStackException extends RuntimeException {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	public EmptyStackException() {
		super();
	}
	public EmptyStackException(String msg) {
		super(msg);
	}
}
```