cs2220 Notes: Class 7

What are the advantages and disadvantages of using abstract data types?

Components of Data Abstractions

Ways to create new objects of the type
  Creators: create new objects of the ADT from parameters of other types
  Producers: create new objects of the ADT from parameters of the ADT type (and other types)

  In Java, operations that produce new objects of the datatype are known as **constructors**.
    Unlike methods, they are declared with no return type, and their name must match the name of the datatype.

Ways to observe properties: **observers**
Ways to change properties: **mutators**

What are the minimal operations a (useful) data abstraction must provide?

Specification of the StringStack Data Abstraction

```
public class StringStack
  OVERVIEW: A StringStack represents a mutable last-in-first-out stack where all
       elements are Strings.
       A typical stack is [ e_n-1, e_n-2, ..., e_1, e_0 ] where e_n-1 is the top of the stack.
  public StringStack()
     EFFECTS: Initializes this as an empty stack.
  public void push(String s)
    MODIFIES: this
    EFFECTS: Pushes s on the top of this.
      For example, if this_pre = [ e_n-1, e_n-2, ..., e_1, e_0 ],
       this_post = [ s, e_n-1, e_n-2, ..., e_1, e_0 ]
  public String pop() throws EmptyStackException
    MODIFIES: this
    EFFECTS: If this is empty, throws EmptyStackException.  Otherwise,
      returns the element on top of this and removes that element from this.
      For example, if this_pre = [ e_n-1, e_n-2, ..., e_1, e_0 ],
        this_post = [ e_n-2, ..., e_1, e_0 ] and the result is e_n-1.
  public String toString()
    EFFECTS: Returns a string representation of this.
```

```java
import java.util.ArrayList;
import java.util.EmptyStackException;
import java.util.List;

/**
 * OVERVIEW: A StringStack represents a last-in-first-out stack where all elements are Strings.
 *     A typical stack is [ e_n-1, e_n-2, ..., e_1, e_0 ] where e_n-1 is the top of the stack.
 */
public class StringStack {
  // Rep:
  private List<String> rep;

  /**
   * EFFECTS: Initializes this as an empty stack.
   */
  public StringStack() {
    rep = new ArrayList<String>();
  }

  /**
   * MODIFIES: this
   * EFFECTS: Pushes s on the top of this.
   *      For example, if this_pre = [ e_n-1, e_n-2, ..., e_1, e_0 ],
   *        this_post = [ s, e_n-1, e_n-2, ..., e_1, e_0 ]
   */
  public void push(String s) {
    rep.add(s);
  }

  /**
   * MODIFIES: this
   * EFFECTS: If this is empty, throws EmptyStackException.  Otherwise,
   *    returns the element on top of this and removes that element from this.
   *    For example, if this_pre = [ e_n-1, e_n-2, ..., e_1, e_0 ],
   *        this_post = [ e_n-2, ..., e_1, e_0 ] and the result is e_n-1.
   */
  public String pop() throws EmptyStackException {
    try {
      return rep.remove(rep.size() - 1);
    } catch (IndexOutOfBoundsException e) {
      assert rep.size() == 0;
      throw new EmptyStackException();
    }
  }

 ... // toString not shown
}
```