Rubin Peci
rp3196
N15878268
Homework #3

A.1
The code fragment will print both "I am a Rectangle!" and "I am a LocatedRect!" because the instance in question, rr, was initalized as type Rectangle but with a call to the constructor of LocatedRect, thus making it both types

A.2
a.
The List.java design contains an inner class, Node, that has all the constructor for a node, which has two varables: a integer named value, and a Node named next. The "next" variable will point to the next node in the linked list. The actual contents of the list class contain variables that point to the head and tail of the linked list, with methods to add to the list either from the front (head) or the back (tail), as well as a method to remove the head and a method to show the list in human-readable format. The main method shows the aforementioned methods by populating a list with random values.

b.
The Node-as-List.java design contains two separate classes, one Node class and one Node-as-List class, the latter simply containing a main method that tests the Node class. The Node class itself has two variables, an integer named data, and a Node named next. The constructor takes just one input, an integer, and it sets the data variable to that. There is a show method that prints a String input and the value of the current node, and then goes on to the next one. The Nodes themselves are the list, since they all point to the next node in the list. The Node-as-List class shows this in action.

c.
Pros of List.java design
- More control over the List with addHead/Tail, and removeHead method
- Easier to loop through and keep track of nodes in the list
- Easier to add nodes to the list

d.
Cons of List.java design
- More code than Node-as-List.java
- Must keep track of head and tail in order for it to work
- More variables of type Node means more chances for a NullPointerException

A.3
a.
The design of this class is very similar to the List.java class, but the main difference is presence of "sentinel" nodes. There is a tail sentinel and a head sentinel that are at their respective positions in the list. These nodes never change, and every other node that is added is put into the LinkedList by adding between these pre-existing sentinel nodes.
b.
Pros
- Easier to organize code
- Less amount of code while providing security
- Insertions are easier - every node is between an already existing node
Cons
- More Nodes must be created in memory

A.4
String - "[^a].[bc]+"
1. abc
False - there is an "a" in the string

2. 1bbbbbbbb
True - there is no "a" and the b matches the b from the set in the regex

3. abbbbbbbb
False - there is an "a" in the string

4. 1zc
True - there is no "a" in the string, and the c matches the c from the set in the regex and the dot meta-character matches the z

5. abcbcbcbc
False - There is an "a" present

6. 1c
False - Not long enough

7. asccbbbbcbcccc
False - There is an "a" present

A.5

a. Non-trivial solution - ".*[is].*"
b. False-negative - "z"
c. False-positive - ".*[p].*"
d. False-positive AND false-negative - ".*[o].*"

A.6
NO - if you have two lists, AA and BB, that have the same characters used in all Strings, or even the two lists being identical, it's impossible to find a solution