Introduction to Computer Science, Homework 6

Background

In this homework, you’ll build a Stack class that represents a last-in-first-out collection of objects. You may not use any class in java.util for steps 1 and 2. You should turn in two Java files, submit/hw6/Stack.java and submit/hw6/Matcher.java.

Step 1

Implement a class Stack that represents a collection of objects with the following methods:

```java
public Object top();
public void pop();
public void push(Object obj);
```

top returns the top object on the stack, or null if it is empty. pop removes the top object from the stack. It should throw an IllegalStateException if the stack is empty. push adds its argument to the top of the stack. You will also need a no-argument constructor to initialize the state of your stack. For example:

```java
Stack stack = new Stack(); // create empty stack
System.out.println(stack.top()); // should print "null"
stack.push(Integer.valueOf(5)); // stack now contains (5)
stack.push(Character.valueOf('x')); // stack now contains (5,x)
System.out.println(stack.top()); // should print "x"
stack.pop(); // stack now contains (5)
System.out.println(stack.top()); // should print "5"
```

I suggest implementing the internal state with an array of Object and an index of the top element of the stack. If an object is pushed and there is no space, push should allocate a bigger array, copy the data, and store the new element. It is a good idea to grow the array by a fixed factor (50%?) rather than a fixed number of elements so that you don’t need to copy the array many times.
Step 2

Add `toString` and `equals` methods, overriding the methods of the same name in `java.lang.Object`. Changing `equals` without changing `hashCode` is a bad idea, so override `hashCode` too so that it throws `UnsupportedOperationException`. Here's an example for `hashCode`.

```java
@Override public int hashCode() {
    throw new UnsupportedOperationException(
        "hashCode not implemented");
}
```

`toString` should return "(foo,bar)" if the `Stack` was populated with:

```java
Stack stack = new Stack();
stack.push("foo");
stack.push("bar");
```

equals should consider a `Stack` equal to another `Stack` if they have the same number of elements and corresponding elements compare equal with their own `equals` methods.

Step 3

Change your class to extend `AbstractList`. It should override/implement:

```java
public boolean add(Object obj);
public Object get(int index);
public int size();
```

Read the `AbstractList` documentation for the precise contracts your implementations should obey. Remember `@Override`.
Step 4

Use your stack class to match balanced parentheses and brackets. Create a class Matcher with a method:

```java
public static boolean matchParentheses(String s);
```

It takes a String containing (), [], and {} and returns true if the parentheses are balanced. For example:

- Matcher.matchParentheses("()") // returns true
- Matcher.matchParentheses("([)]") // returns true
- Matcher.matchParentheses("([)]())") // returns true
- Matcher.matchParentheses("([)])") // returns false
- Matcher.matchParentheses("()") // returns false
- Matcher.matchParentheses("[]") // returns false

I suggest implementing this by creating a Stack, pushing a ( or [ onto the stack whenever one of those characters is seen, and popping it off when the corresponding closing character is seen. If the wrong character is at the top of the stack, the parentheses do not match, and the method should return false. Note that your Stack class can hold Character objects, but not char values, since the latter is a primitive type.