Program ? – Stacks

100 points

IMPORTANT
You must write your code while carefully following the "IT 179 Program Grading Guidelines" described in the file **posted with Program 1**.

Set-Up
Create a new Java project named: **`P6`**. Next, inside the created Java project **`P6`**, create a Java package.

Objective: Practice with stacks and linked lists.

You are to develop a Java program that checks Java code fragments for balanced symbols—(), [], {}. The program will ask the user for a file name. It will read the file and print **one** of 4 messages to the console:

Case 1: *`Symbol1`* `on line` *`num`* `does not match` *`symbol2`*`.`

Example: `'}' found on line 12 does not match '('.`

Case 2: `End of file reached with unmatched` *`symbol`*`.`

Example: `End of file reached with unmatched '{'.`

Case 3: *`Symbol1`* `on line` *`num`* `has no matching opening symbol.`

Example: `')' on line 1 has no matching opening symbol.`

Case 4: `All symbols correctly balanced.`

As soon as a symbol balance issue is found, your program will stop looking (doing this with a boolean flag controlling your loop, not a break or return out of the loop).

You will provide two JUnit classes: `StackTest` to test your Stack class, and a `SymbolCheckerTest` to test the individual functions of your program.

The `Stack` Class

You will begin by writing a `Stack` class that stores `chars` and uses a singly linked list representation. The `Stack` class will have a `Node` inner class with data that is a single `char`. The `Stack` will have one instance variable of type `Node` which will reference the head of the list (or top of the stack).

The `Stack` class must have exactly the following public methods.

- `isEmpty()` which will return a `boolean`
- `push(char item)`
- `pop()`
- `top()` which will return a `char`.
- a correct override of the `equals` method which will be true if and only if the parameter is a `Stack` with exactly the same contents as the calling object.
- `clear()` which will pop all items off the stack.

The `pop` and `top` methods may assume that the stack does have data (which means that your program will **have** to check for an empty stack before calling them).

The SymbolChecker Class

`SymbolChecker` will manage the process of checking a file. It will have the following instance variables:

- A `Stack`
- A `Scanner` that will be attached to the file being checked
- An `int` that will be used to keep track of the current line number

You must have **at least** the following methods:

- a default constructor which will create the `Stack` object
- a `setup` method which will accept a `String` parameter and open the corresponding file.
- a `runChecker` method that will manage the checking process (reading through the file and calling other methods)
- a `cleanup` method that closes the `Scanner`.
- additional methods that handle pieces of the checking process. This is your opportunity to think about some program design: what are some individual tasks that could be placed in methods?

- a `main` method that will create a SymbolChecker object and call the `setup`, `runChecker`, and `cleanup` methods in turn.

Sample files have been provided for you that demonstrate each of the cases. We are putting the code we are checking in .txt files for convenience (and reduced confusion). Remember that your program must work correctly on the provided sample data to get a B or better (as well as meeting other requirements for an A or B).

JUnit Testing:
Create a JUnit test class StackTest that thoroughly tests each of your Stack methods.

Also create a SymbolCheckTest class that tests the methods that handle pieces of the checking process.

Submission
Zip your .java files and submit the .zip file to the **Program 6** assignment on ReggieNet.

Grading will be in accordance with the Program Grading Criteria provided on the Important Resources page on ReggieNet.