Pair up with anyone who is agreeable to pairing up with you, pick the first driver, and let’s get to work!

PART I: Matching Brackets

In mathematical expressions and Java programs, brackets of various forms are used to group elements in ways that clarify intent. The Java compiler needs to make sure that the brackets in your programs are appropriately paired together, and, if brackets appear inside of pairs of brackets, all of the brackets are correctly ‘nested’. Pairing up matching brackets is a job for ... a stack!

1. On the class web page is the shell of Brackets.java. Load it into DrJava.

2. Also on the class web page is a completed stack class, StackSection7.java, that uses a StringBuilder object as the representation of a stack of characters. Load it into DrJava, too.

3. Spend a few minutes looking at StackSection7.java to familiarize yourselves with the methods and how they work. You’ll need to use a StackSection7 object in this checkpoint. Use the Java API to familiarize yourselves with StringBuilder’s methods, if necessary.

4. This program accepts a string of characters on the command line and checks that any pairs of brackets it contains pair up correctly. The program includes some example situations that the program needs to handle, and provides the expected output. It also includes some internal comments to help you along, but that’s all the help you get. (We’re confident that you can work out the necessary details with your partner.)

Complete the program! Be sure to test it on the situations given in the top comment block to make sure the program does what it needs to do in those situations.

✔ CHECKPOINT 1 Raise your hand. Your SL will come over and verify that your program is correctly matching brackets.

PART II: A StringBuilder Queue

In Part I, you used a stack class that stacked characters using a StringBuilder object as the stack’s representation. In Part III of this activity, you will have need of that stack class, and of a queue class that also holds characters. In this part you’ll create that queue class, which also uses a StringBuilder object as its data structure.

1. Open DrJava, and create a class named QueueSection7 that uses a StringBuilder object to hold characters in a queue. Here are the method signatures of the methods that your class needs to provide:

(Continued ...)
- **QueueSection7()** — the constructor; creates an empty queue
- *boolean isEmpty()* — true if no characters in the queue; false otherwise
- *int getOccupancy()* — quantity of characters in the queue
- *void enqueue(char)* — add char to the rear of the queue
- *int dequeue()* — removes the front character, returns -1 if empty
- *int peek()* — shows the character at the front of the queue, returns -1 if empty

Make it happen! (Hint: Looking at the code of the **StackSection7** class used in Part I might help.)

2. Copy and paste the main program file, **QueueSection7Main.txt**, from the class web page into your **QueueSection7** class.

3. Compile and run **QueueSection7**. If necessary, debug your methods until you’re getting the output specified in **QueueSection7Main.txt**’s comments.

![CHECKPOINT 2](image)

Raise your hand. Your SL will come over and verify that your class is working.

**PART III: Detecting Palindromes with a Stack and a Queue**

*A palindrome, as your SL explained at the start of section today, is a word or phrase whose letters appear in the same order left-to-right and right-to-left. For example, I’m a lasagna hog, go hang a salami.* is a palindrome (we ignore spaces and punctuation symbols).

Here’s one way to test a string to see if it is a palindrome: Take each letter in the string and both push it onto a stack and enqueue it into a queue. Then, repeatedly pop the stack and dequeue from the queue. If the pairs of characters removed do not match, then the string is not a palindrome. But, if you empty both data structures and all of the pairs of characters have matched, the string is a palindrome.

1. Using a pen/pencil and paper, use the algorithm described at the top of this part, and drawings of a stack and a queue, to test each of these four small strings:

   (a) *kayak* (a palindrome)
   (b) *abcb* (one ‘a’ short of being a palindrome)
   (c) *bcba* (same, but on the opposite end)
   (d) *abcdeba* (not a palindrome; ‘c’ doesn’t pair with ‘e’)

   (The point is to get a feel for how the algorithm works, which should make it easier to write a program that carries out that algorithm.)

(Continued …)
2. Write a Java program named `PalPal.java` (for ‘Palindrome Pal’) that takes a string from the command line and tests it to see if it is a palindrome using the stack/queue algorithm. There’s no shell for this; you need to write it from scratch. The following execution examples show the behaviors your SL will expect your program to demonstrate:

```java
> java PalPal
Usage example: java PalPal "Yo, banana boy!"

> java PalPal ""
"" is a palindrome!

> java PalPal "Sit on a potato pan, Otis!"
"Sit on a potato pan, Otis!" is a palindrome!

> java PalPal "abcb"
"abcb" is not a palindrome; the problem is that 'a' doesn't match 'b'.

> java PalPal "bcba"
"bcba" is not a palindrome; the problem is that 'b' doesn't match 'a'.

> java PalPal "abcdcba"
"abcdcba" is not a palindrome; the problem is that 'c' doesn't match 'e'.
```

Hint: Remember that the `Character` wrapper class has static methods that can test characters to see if they have various properties.

![CHECKPOINT 3] Raise your hand. Your SL will come over and check that your program is working (and is following the given algorithm).

**PART IV: Clean Up!**

1. Log out of your computer.
2. Pick up your papers, writing implements, cell phones, trash, etc.
3. Push in your chair(s).

![CHECKPOINT 4] Raise your hand. Your SL will come over and see if your workspace is so clean that its jokes would be funny only to pre-schoolers.

You’re free to go! But, if you have time, we recommend that you use it to work some more on the next programming assignment.