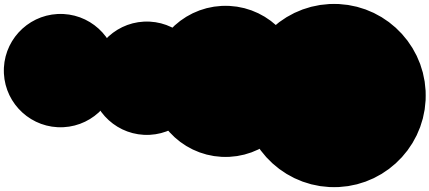
Asst. 4: Palindrome parsing



Preparation

Familiarity with the following textbook sections and lectures will help you complete the assignment. You are also expected to attempt the exercises below before beginning the assignment. This assignment is intended to be done with a partner!

Textbook sections

- 4.3, 4.4
- 5.2, 5.3, 5.6

Most relevant lectures

- L8, L9

Exercises

1. Read through the entire assignment!
2. What is a palindrome? If you don't know, look it up. Google is your friend.
3. Write an algorithm to solve the **Part A: Two-character** problem.
4. Write an algorithm to solve the **Part B: Digits only** problem.
5. Using the algorithm you just wrote as a starting point, write an algorithm (in pseudocode) to solve the **Part C: A real sentence** problem. Trace through your algorithm to verify whether "Madam, I'm Adam." is accepted as a palindrome. (It should be.) You will be required to submit this.

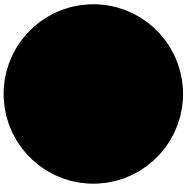
Introduction

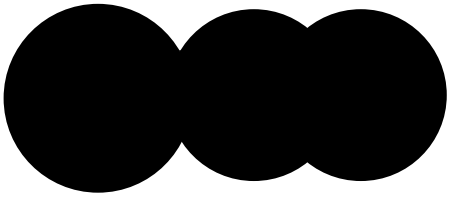
You will be writing a program to test whether or not various user-entered `String`s of text are palindromes. You will start with a simpler version of this program, and gradually upgrade it until you arrive at a full solution.

Pair programming

You are encouraged to work in pairs for this assignment, and learn the practice of *pair programming*! Introduce yourself to somebody in the class and ask to work together. (If you truly insist on working alone, you may.) Each pair should only submit one copy of the assignment (to one partner's D2L account), with both of your names and IDs in the files.

Pair programming works like this: The partners assume different roles. Once the algorithm has been developed and agreed upon, one partner will be the programmer (sitting at the keyboard, typing), and the other partner will observe what is typed. The second partner's job is to catch syntax/logical errors as they occur, and to make suggestions to the programming partner. Every so





often, the two partners should exchange roles. This technique has been proven to be very effective at producing better code.

1 Specifications and instructions

A reminder of good practices

- Declare all your variables at the top of your main method, before any other statements.
- Use in-line comments carefully to explain what you are doing. Make sure each comment is useful. There is no need to comment every line. Obvious comments like *//prints output* are fairly useless. Concise but descriptive comments like *//displays the largest number in numList* are much more useful. Especially when there is tricky logic involved, be sure to explain it clearly.
- Comment your ending braces, i.e., *//end outer while*.
- Align your ending braces with the opening `if` or `public` or `while` that begins that block.
- Code inside a block should be uniformly indented by one level.

1.1 Part A: Two-character

In this part, you will write a program called `PalChecker`, which tests whether or not a two character `String` is a palindrome. Do not proceed to Part B until Part A works properly.

Input: The user should input a two character `String`. Don't worry about improper input.

Output: Sample console if the entered `String` is a palindrome:

```
Please enter two characters:
mm
You entered a palindrome!
```

Sample console if not:

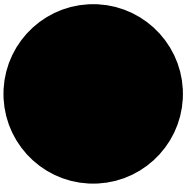
```
Please enter two characters:
3f
Sorry, that is not a palindrome.
```

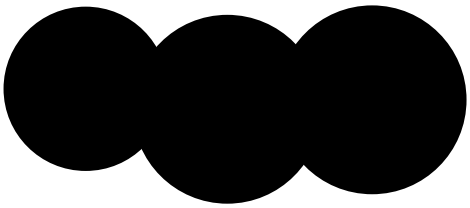
1.2 Part B: Digits only

In this part, you will modify `PalChecker` to test whether or not any integer is a palindrome. The integer may have arbitrarily many digits. You will overwrite your earlier program. Do not proceed to Part C until Part B works properly. Hint: You will need to use a loop in this part.

Input: The user should input an integer. Don't worry about improper input. (Hint: You will likely want to treat the input as a `String`.)

Output: Sample console if the entered integer is a palindrome:





```
Please enter an integer:
5489845
You entered a palindrome!
```

Sample console if not:

```
Please enter an integer:
57402
Sorry, that is not a palindrome.
```

1.3 Part C: A real sentence

In this part, you will create a new file called `RobustPalChecker`, which will test whether or not a full sentence is a palindrome. Adding punctuation and spaces should not affect the result. Letters in different cases should be considered equivalent.

1. Create a new file and save it as `RobustPalChecker.java`.
2. Copy-paste the code from Part B into this new file.
3. Modify `RobustPalChecker` to behave as explained below. You should only need to add/change 6-9 lines of code, so *plan carefully* before starting to code.

Input: The user should input any sentence (or word, or any sequence of characters). Note: Copy-pasting will not yield the proper results in `SciTE`. The user should type the entire sentence.

Output: The program should accept any palindrome, ignoring punctuation, case, and spaces, etc. Sample console if the entered sentence is a palindrome:

```
Please enter a sentence:
A man, a plan, a canal -- Panama!
You entered a palindrome!
```

Sample console if not:

```
Please enter a sentence:
On a dark, desert highway...
Sorry, that is not a palindrome.
```

Submission

Recall that submission instructions are in the **Lab Guide**. You are required to submit **one** `.zip` folder (on one partner's D2L) containing:

- the properly documented and styled source code file `PalChecker.java`
- the properly documented and styled source code file `RobustPalChecker.java`
- a .txt or .pdf file containing your algorithm for `RobustPalChecker`

Make sure both partner's names/IDs are on both files. Also, ensure that each partner saves a copy of the finished code to their personal H: drive.