University of New Brunswick
CS1083 – Open Access Version

Module 7 Assignment

The purpose of this assignment is to give you practice:

- with recursive methods.

For all assignments:

- Follow the instructions in the document "Java Coding Guidelines.pdf" available within the "Start Here" module, under the "Assignments" topic.
- For each .java file you create, include a javadoc comment (one that begins with /**) at the beginning of the file that describes that Java class. This comment block should include a line that begins with @author followed by your name and student number on the same line. (Note: inclusion of this comment is part of the instructions given in "Java Coding Guidelines.pdf")
- Include comments throughout your programs to explain any non-obvious portions of your code.
- It is recommended that you create a separate folder on your computer for each assignment. You might even wish to create separate sub-folders within an assignment folder if the assignment has multiple parts. Keeping your work organized makes it easier to find things later when you want to review what you have done.
- Few things in life are more frustrating than losing your work while working on an assignment. Get in the habit of saving frequently when working on an assignment. Also, regularly make backup copies of any files you create as part of your work for this course.
- Each module has a link to the discussion forum where you can ask questions and seek help in completing your assignments.
- Submitting your assignment involves creating a pdf file and uploading that single file to the assignment drop box on D2L. In general, that file will tend to include all Java code you wrote for the assignment, plus any output from running your programs that the assignment instructions specify you should capture. Specific submission instructions are included at the end of each assignment.
- To create the pdf file for each assignment, begin by opening a new document using the word processing program of your choice. Save the document using the name “Your Name CS1083 Module x Assignment Submission.docx” Replace x with the correct module number. “docx” may be different depending on which word processing software you choose to use.

- At the beginning of your submission document enter your name, your student number, CS1083, the assignment name (this one is “Module 2 Assignment”), and the date. It doesn’t matter if the date is when you started working on the assignment or when you submit it – either will do.
- You can add content to your submission document as you work on the various questions in the assignment. Clearly label each part of the assignment (“Part A” etc.) in your document. When your document is complete, save / export as a pdf file. Always make sure your pdf file opens properly before uploading it.
- To include Java code in your submission document, copy all the text in your .java file and then paste that text into your submission document. Use a monospaced font (e.g.: Consolas, Courier) for your code to maintain proper indentation.
- To include output from running your program in your submission document, you have two choices. You can either (a) copy and paste the text from your command prompt window, or (b) capture a screen shot of the output and include that as a picture / image in your submission document. Either approach is fine.
- To copy text from your command prompt window, try selecting the desired text and then pressing either command-c (Mac) or control-c (Windows or Linux). If you have issues, you can always use Google to see how to do this on your specific type of computer.
- To capture a screen shot of a selected portion of your command prompt window, try command-shift-4 (Mac), WindowsKey-shift-s (Windows), or shift-PrtScrn (Linux).
- Once you have pasted your output text into your submission document, make sure it is in a monospaced font (e.g.: Consolas, Courier) so you retain the alignment of output.

Part A: Greatest Common Divisor

The greatest common divisor of integers x and y is the largest integer that evenly divides into both x and y. Write a recursive method gcd() that returns the greatest common divisor of x and y. The gcd of x and y is defined recursively as follows:

Base Case: if (y == 0) then gcd(x,y) is x.

General Case: Otherwise, gcd(x,y) is gcd(y, x%y)

Your gcd method should have the following signature:

```
public static int gcd (int x, int y)
```

In the same class, write a main method to test your gcd method. Loop to generate ten pseudo-random pairs of values for x and y, where each value is between 5 and 20 inclusive. For each randomly generated pair, display the values for x, y, and the result you get back from gcd(x,y). Your output should look like the following (with different randomly generated numbers, of course). Use tab characters (`"\t"`) to separate the columns of output.

```
x      y    gcd(x,y)
==    ==    ========
7     7     7
13    5     1
20    7     1
9     7     1
9     10    1
13    18    1
10    15    5
9     12    3
5     14    1
20    14    2
```

Next, overload your gcd method by including in the same class a method with the following signature:

```
public static int gcd (int x, int y, String indent)
```

This method uses the same recursive algorithm for calculating the greatest common divisor, but this method also does some printing. The idea is to help visualize how the recursive calls work.

Inside gcd, just before the recursive call, add a line to print the values for which gcd is about to be called.

Also, just before returning, gcd prints the value it is about to return, along with the values for which it was called (that is, the values of x and y).

The `indent` string is included at the beginning of each of those prints so the output from each call is indented two spaces further to the right. To accomplish this, include

`indent+"  "` as an argument for the recursive call to gcd. In other words, this `indent` string becomes longer each time gcd is called.

In the main method, loop to generate test values for this version of gcd in a similar way as for the first gcd method. Again, loop to generate pseudo-random pairs of values for x and y, where each value is between 5 and 20 inclusive, but this time only loop twice.

Your output should look like the following (with different randomly generated numbers, of course). Print this output below the output shown above (on the preceding page). The lines of output that include `*****` are printed by the main method. The lines in between are printed by the gcd method.

```
***** gcd(12,20) *****
Calling: gcd(20,12)
  Calling: gcd(12,8)
    Calling: gcd(8,4)
      Calling: gcd(4,0)
        Returning: 4 from gcd(4,0)
      Returning: 4 from gcd(8,4)
    Returning: 4 from gcd(12,8)
  Returning: 4 from gcd(20,12)
Returning: 4 from gcd(12,20)
***** RESULT: 4 *****

***** gcd(6,9) *****
Calling: gcd(9,6)
  Calling: gcd(6,3)
    Calling: gcd(3,0)
      Returning: 3 from gcd(3,0)
    Returning: 3 from gcd(6,3)
  Returning: 3 from gcd(9,6)
Returning: 3 from gcd(6,9)
***** RESULT: 3 *****
```

Just for the sake of interest, notice that the last call always matches the base case of the recursive algorithm: where y==0.

Part B: Palindromes

A palindrome is a String value that is spelled the same way forward and backward. Some values of palindromes are:

- `"radar"`
- `"a"` (or any String of length 1)
- `""` (a String of length 0)
- `"Able was I ere I saw Elba"` (if capitalization is ignored)
- `"A man, a plan, a canal, Panama!"` (if only the letters are considered)

Write a method with the signature:

```
public static boolean isPalindrome(String s)
```

This method returns true if `s` is a palindrome, false otherwise. This method:

1. Makes a cleaned-up copy of `s` such that only the letters are retained and all the letters are lowercase (that is, no capitalization, spaces, or punctuation remain in the string);
2. Passes this cleaned-up string to a recursive method (which you also are to write) of the same name and same return type; and then
3. Returns whatever it gets back from the recursive method.

Getting rid of capitalization in a String is easy; the String class includes a toLowerCase() method.

To retain only letters, loop through `s` to extract each character with the charAt() method and build another string that only includes the letters. You can determine if a char value is a letter in a variety of ways. For instance, you might use the isLetter() method provided by the Character class. Or you can ask if a character is greater than or equal to `'a'` and also less than or equal to `'z'`. (Remember: The Java comparison operators like ==, <=, and >= are valid with char values.)

The second isPalindrome() method (the recursive one) is private instead of public, and has the following signature:

```
private static boolean isPalindrome(String s, int index)
```

The `index` parameter is used to designate a smaller and smaller portion of the string each time the recursive method is called. Initially, your public isPalindrome() method calls the recursive method with a string and an index value of 0 (zero).

As an example, consider the string `"abccba"` where index is 0.

Using the charAt() method from the String class, we can extract the character at index 0, which is:

```
'a'
```

For this string to be a palindrome, that first character must be equal to the last character in the string. Since the last character is also `'a'`, then this string might indeed be a palindrome … but it depends on checking the rest of the string. We can do that by a recursive call to isPalindrome() with `index+1`.

When `index` is 1, we want to compare the two blue characters shown here:

`"abccba"`

Similarly, when we get to the recursive call where `index` is 2, we want to compare the two blue characters shown here:

`"abccba"`

If at any point the recursive method finds two corresponding characters that are not equal, then the method immediately returns false.

The base case is when `index` gets close enough to the middle of the string that there are no more pairs to check. At this point the method returns true. (The only way the value of `index` can make it to that point is if all the corresponding pairs of characters are equal along the way to getting there, so the string must be a palindrome.)

Hint: It might help to use a couple of examples like `"abccba"` (an even number of characters) and `"radar"` (an odd number of characters) to manually trace (with pen and paper) the value of `index` and figure out how to test for the base case.

Provide in the same class a main() method that tests your public isPalindrome method.

Include your test values in an array of Strings. Loop through this array to produce output like the following:

```
Yes!  "radar"
Yes!  "a"
Yes!  ""
Yes!  "Able was I ere I saw Elba"
Yes!  "A man, a plan, a canal, Panama!"
No... "Hello"
No... "abcdba"
```

Include the first five strings shown above, and also the following strings of your own choosing:

- Two other strings that are palindromes (these don't have to be actual words); and
- Four other strings that are not palindromes. Do not include `"Hello"` or `"abcdba"`.

Hint: Use the two-character sequence `\"` to include double quotation marks in your output as shown above.

Submission Instructions

Include the following in your submission document:

- Your name, student number, CS1083, Module 7 Assignment, and the date
- Part A:
  - Complete code for the class that includes your gcd() methods
  - Sample output from executing this class twice (that is, output from two different runs)
- Part B:
  - Complete code for the class that includes your isPalindrome() methods
  - Sample output from executing this class

D2L DROPBOX SUBMISSION INSTRUCTIONS

Remember to save / export your submission document as a pdf file, and then upload that one pdf file to D2L as follows:

1. In the top-navigation bar on the course screen, select 'Assessments' and then 'Assignments'.
2. Select the assignment title and follow the instructions to upload your submission document.