PA1: Debugging Arrays and Working with Strings (100 Points)

Due: 11:59pm, Thursday, April 9th

Overview

This assignment consists of two parts. This first part involves debugging a Java file which has both static and dynamic errors. The second part involves writing a basic Caesar Cipher program to encrypt/decrypt text.

In all of the following, the > is a generic command line prompt (you do not type that).

Setup

If you do not know your own CS8B login id (cs8bXX), go to the ACMS lookup tool here: https://sdacs.ucsd.edu/~icc/index.php

Open a new Linux terminal window. In your home directory, create/make a new directory called pa1 using mkdir (in Unix, the tilde ~ expands to your home directory):

   > mkdir ~/pa1

Copy the provided files from the public directory by typing in:

   > cp ~../public/pa1/* ~/pa1/

Then change directory (cd) into your pa1 directory

   > cd ~/pa1

Part 1: Debugging Buggy Array Code (30 points)

There are two files you will need for this part of the assignment:

FunWithIntArrays: This class is designed to have some useful features for int arrays but is marred with errors (both compile time and runtime). There are a number of methods in the class whose expected behavior is commented.

ArraysTester: This code tests the class FunWithIntArrays. When we test your fixed FunWithIntArrays, we will use the original copy of this file. But you may want to modify this while fixing the FunWithIntArrays class (e.g., by commenting out some method calls while debugging).

To compile at the command line:

   > javac FunWithIntArrays.java

To edit files, we will be using the vim editor.

   > vim FunWithIntArrays.java
Part of this assignment is to start learning the vim editor. Several resources on vim are under the Useful Links section of the CSE 8B homepage.

After debugging and fixing the FunWithIntArrays class, compile the ArrayTester.java test driver.

> javac ArraysTester.java

Run the ArrayTester program, the output should be the following:

> java ArraysTester
Creating Initial Array:
2, 7, 1, 22, 5, 53, 21, 9,

Creating Array Copy:
2, 7, 1, 22, 5, 53, 21, 9,

Min element is: 1

Max element is: 53

Testing Reversed Array
9, 21, 53, 5, 22, 1, 7, 2,

Part 2: Caesar Cipher (50 points)

A *cipher* is an algorithm that takes a message in the form of a sequence of characters called the "plaintext" and returns another sequence of characters called the "ciphertext" which is (a) apparently hard to read, but (b) can be converted back into the original plaintext message. Turning the plaintext into the ciphertext is called encryption; reversing the process is called decryption.

A *substitutional cipher* is an algorithm that creates a ciphertext by simply substituting, for every occurrence of a particular letter of the alphabet in the plaintext, another particular letter of the alphabet; recovering the plaintext then just involves doing the reverse substitution. In this part, you will implement a kind of substitutional cipher called a *rotational cipher* or *Caesar cipher*.

Substitutional ciphers such as the Caesar cipher have been used historically to encode secret diplomatic and military messages, but these days they are not considered very good ways to hide information that you really want to keep secret. In fact, you could fairly trivially write a method that can break one using just character frequencies. More sophisticated modern techniques exist and should be used if you want strong security. But still, substitutional ciphers are useful in some contexts; for example, the "rot 13" rotational cipher is sometimes used to obscure (usually controversial or potentially offensive) text in email or newsgroups.

The rotational cipher you will implement in this assignment will affects only *letters* in the plaintext; numbers, punctuation, etc. are not changed. Here's why it is called a rotational cipher: Imagine the letters A,B,C,...,X,Y,Z written in order around the rims of two wheels on the same axle, and the wheels lined up so that A on one wheel is next to A on the other, B is next to B, etc. If you peeled the letters off the rims of the wheels and laid them out side-by-side, it would look like:
Now rotate one of the wheels by one position, and you have the correspondences for the "rot 1" cipher, that substitutes B for A, C for B, ... , and A for Z. Again, laid out side-by-side, it would look like:

BCDEFGHIJKLMNOPQRSTUVWXYZA
ABCDEFGHIJKLMNOPQRSTUVWXYZ

If instead you rotated one position in the other direction, you have the "rot -1" cipher, which is equivalent to "rot 25". Laid out, it would look like:

ZABCDEFGHIJKLMNOPQRSTUVWXY
ABCDEFGHIJKLMNOPQRSTUVWXYZ

"Rot N" ciphers are possible for any integer value of the offset N, but many of them are equivalent; there are only 26 distinct rotational ciphers (including the "rot 0" 'identity' cipher shown first above that substitutes each letter with itself, changing nothing). In particular, note that "rot 13" is equivalent to "rot -13", and so it decodes itself:

NOPQRSTUVWXYZABCDEFGHIJKLM
ABCDEFGHIJKLMNOPQRSTUVWXYZ

An important detail: In the rotational cipher you will implement, upper-case letters are always substituted for upper-case letters, and lower-case ones for lower-case ones. The amount of rotation is the same for each. So, you could think of a rotational cipher as really having two sets of wheels, one pair for upper-case and the other for lower-case letters, with each set of wheels rotated the same amount; for example for "rot 13" you would have the correspondences:

NOPQRSTUVWXYZABCDEFGHIJKLM
ABCDEFGHIJKLMNOPQRSTUVWXYZ

nopqrstuvwxyzabcdefghijklm
abcdefghijklmnopqrstuvwxyz

To implement the cipher, add the following methods to your Caesar class (each of which is described in more detail below):

- public String encrypt(String s, int rotation)
- public String decrypt(String s, int rotation)
Now here are the details for each method (See the section below for some HINTS!)

public String encrypt(String s, int rotation)

This method takes a String and an int and encrypts the String by rotating each character which is an upper or lower case letter by the amount rotation. It should not change non-alphabetic characters.

public String decrypt(String s, int rotation)

This method will decrypt a string by applying the rotation in the opposite direction. So a string encrypted with one rotation should be recovered by calling decrypt with the same value for rotation.

**General Hints and Other Remarks**

1. **START EARLY!** Go to discussion sections and tutoring hours for help.

2. Before you write any Java code, be sure you clearly understand the concept of a rotational cipher. Here are some examples of input-output behavior of the encrypt and decrypt methods: The rotational offset can be any int (positive, negative or 0); but any rotational offset will be equivalent to one in the range 0 through 25. For example, it can be seen that a rotation of 39 is equivalent to one of 13, and a rotation of 26 is equivalent to one of 0.

3. Your class must have the certain public methods as shown above. Some of these public methods should use other public methods to do what they need to do (for example, your encryptFile method should make good use of the encrypt method). However, you may also define additional private "helper" methods if you wish. This is often a good idea; 'procedural abstraction' can help you organize your thinking about how to solve a problem. For example, in thinking about how to define your String encrypt(String, int) method, you might notice that in building up the resulting String to return, you repeatedly need to 'rotate' a char by a certain amount. Your job would be easier if there was a method char rot(char, int) that would do that. So, you can write that 'helper' method, and use it! These helper methods should be private. Spend some time thinking about the breakdown of methods that you will need as you are coding your solution.

4. The standard library Java class Character contains some static methods that may be useful to you in this assignment:

   public static boolean isLetter(char c)
   public static boolean isUpperCase(char c)
   public static boolean isLowerCase(char c)

The String class defines many instance methods, and so these are instance methods that every String object has. Some important ones are described in Chapter 9 of the text. For this assignment, the most useful ones are probably these:

   public int length()
   public char charAt(int index)
   public char[] toCharArray()

5. Note that in Java, Strings are immutable, that is, no String instance method can change anything about the contents of a String object once it is created. In particular, the String object passed in as argument to your encrypt(String, int) and decrypt(String, int) methods cannot itself be modified by those methods. Instead, you will loop through the chars in that String, translating them one-by-one according to the
required rotational cipher, and building up new Strings by string concatenation, until you get the resulting String you want, and then return that String. An alternative approach is to translate the String into a character array, modify the characters in that array, and then to turn the array back into a String (see examples in the book and from class).

6. One thing that can help in correctly computing the rotational substitution for a letter is understanding how to deal with char values as numerical values. Given the numerical value of a char, you can add a number to it (an offset, given by the rotation), and so get the char for the corresponding ciphertext letter.

7. To operate on the numerical value of a char, it can be cast to int, or assigned it to an int variable, or in some contexts it will be automatically converted to int for you. The value so obtained is the Unicode code for the char, which for us is the same as the ASCII code. ASCII code charts are available many places online; looking at such a chart, you can see that all the uppercase letters are in order, and have int values in the range 65 (which is the int value of the char 'A') through 90 (which is the int value of 'Z'). Similarly, the lowercase letters have the 26 int values in order in the range 97 through 122, which are the int values of 'a' and 'z' respectively.

8. When computing a letter substitution, the resulting letter has to have an int value in the appropriate range: upper-case letters go only to upper-case letters, and lower-case to lower-case. So, when applying an offset to a letter, you have to make sure you "wrap back around" so you do not exceed the limits of the range. This "wrapping around" makes sure the transformation is a rotation, and not just a simple offset. As an example, suppose you want to compute the substitution for char 'd', with a rotation of $-40$. This is equivalent to a positive rotation of 12. (Make sure you handle any rotation amount, including negative rotations, correctly!) Adding this offset to the int value of 'd', we get

$$\text{(int)'d' + 12}$$

as the numerical value of the letter we want. We can convert back to char with a cast:

$$\text{(char) ((int)'d' + 12)}$$

which is the char 'p'. Now suppose you want to apply the same rotation of 12 to the char 'Q'. The expression

$$\text{(char) ((int)'Q' + 12)}$$

has as its value the char ']'; but it should be 'C'. The problem is that adding 12 to 'Q' gives an int greater than 90, so it goes past the end of the range of the uppercase chars. If you take this approach, you need to figure out how to "wrap back around" in all cases like this.

**Compiling and Testing your Ceasar Cipher**

In each case, doing a part of the assignment requires that your program compiles and runs correctly. If a part of your code does not run, you will not receive correctness points for it. Note that you should also try to use good style on this assignment.

**Compiling**

To compile your code, use the following:
javac CaesarTester.java

Testing

Similar to Part 1, you have been provided with a CaesarTester.java file. We will be using this file (along with additional tests) during grading. You should use our tests as a baseline and add additional tests to verify to yourself that it is working correctly.

java CaesarTester
Test 1 Passed!
Test 2 Passed!
All Tests Passed!

README (10 points)

You are required to provide a text file named README, NOT Readme.txt, README.pdf, or README.docx, etc. with your assignment in your pa1t directory. There should be no file extension after the file name "README". Your README should include the following sections:

Program Description (2 points):
Describe what each program does as if it was intended for a 5 year old or your grandmother. Do not assume your reader is a computer science major.

Short Response (8 points):
List the bugs found in FunWithIntArrays.java and include a brief description of why the bugs were wrong.

DO NOT just list bugs you encountered in the assignment. They must be specific to FunWithIntArrays.java, and they must be bugs that we gave to you.

For example, if you fixed the for loop below:

```java
for (int i = array.length; i >= 0; i--)
```

to avoid going out of bounds, your entry would look like:

Runtime Error - Array out of Bounds
Incorrect: for (int i = array.length; i >= 0; i--)
Fix: for (int i = array.length-1; i >= 0; i--)
Explanation: Correction starts at the first element which is at index array.length-1 rather than array.length.

Put a line of white space between each error for readability.

Style (10 points)

Don't forget in-line comments. They count towards your grade! These are comments that describe what your code is doing so that if you looked at your code a year later, you would understand it. In-line comments shouldn’t explain what is actually happening in the code like “incrementing variable count” but rather what that code means, so something like "keeping track of number of iterations of the while loop using the count variable". Also it should be used to explain complicated chunks of code or logic that you
struggled with understanding or writing. While we won't grade you on style beyond in-line comments (yet), graders will be giving you feedback on your coding style this week in preparation for future assignments.

Although you will not be graded on style this week, you should follow these basic style guidelines nonetheless. You will be graded on this in weeks to come, so start to practice now.

- Use proper indentation: indent each block of code (e.g., method body loop body). Line up the lines in the block so that they are all indented to the same degree. See examples of this in the book and in the code included in the WordFun.java document.
  - Every time you open a new block of code (use a '{'), indent farther. Go back to the previous level of indenting when you close the block (use a '}').
- Use descriptive variable names: the names of your variables should describe the data they hold. Almost always, your variable names should be words (or abbreviations), not single letters.
- Write short methods: break your methods up into submethods if they are getting too complicated or long.
- Write short lines: each line of code should be no longer than 80 characters, so it can fit in a reasonable size window. There's a column number in the lower-right corner when using vim (row:col). If you don't see this you may have to turn it on by running the following vim command:
  ```
  :set ruler
  ```
- Use blank spaces around logical chunks of code. It makes your code much easier to read and debug.

We'll start with these, as these are the most important. We may add to this list later in the term, but if you do all of the above you're in good shape.

**Turnin**

To turnin your code, navigate to your home directory and run the following command:

```
> cse8bturnin pal
```

You may turn in your programming assignment as many times as you like. The last submission you turn in before the deadline is the one that we will collect. **Always recompile and run your program right before turning it in, just in case you commented out some code by mistake.**

**Verify**

To verify a previously turned in assignment,

```
> cse8bverify pal
```

If you are unsure your program has been turned in, use the verify command. We will not take any late files you forgot to turn in. Verify will help you check which files you have successfully submitted. It is your responsibility to make sure you properly turned in your assignment.

**Files to be collected:**

- README
- FunWithIntArrays.java
- Caesar.java
- CaesarTester.java
- ArraysTester.java
The files that you turn in must be EXACTLY the same name as those above.

NO LATE ASSIGNMENTS ACCEPTED.
DO NOT EMAIL US YOUR ASSIGNMENT!
Start Early and Often!