PSA 8: Multithreading and Data Structures

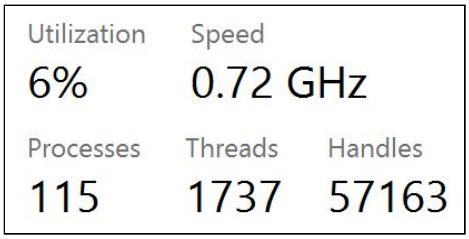
To view the Table of Contents, use [View > Show document outline]. Be aware of course policies as explained on the website and linked via past PSA documents. Read the description carefully.

Part 1 relates to multithreading to be addressed early in week 9 while Part 2 strives to give you a head start for CSE 12. **However, you already have all the tools** to finish the whole PSA!

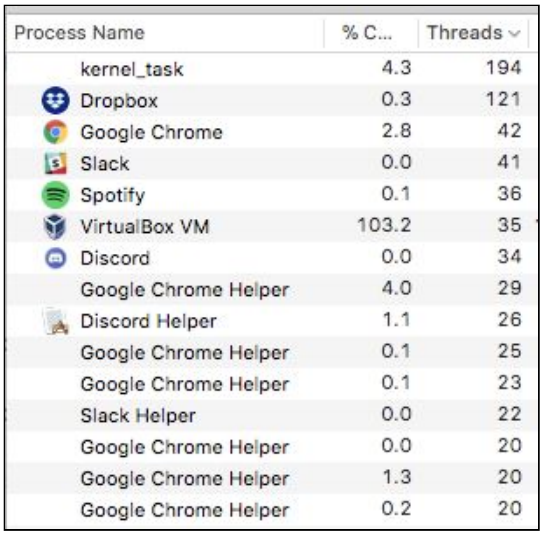
This assignment is due 11:59 PM Tuesday 03/13/2018.

Introduction

As discussed in lecture, threads help break up independent tasks into parts. Each part can then be executed sooner. The effectiveness of this approach is seen by its use in modern operating systems.



Current status as seen by Task Manager of an entire Windows 10 OS in WI 17. Note the thread count.



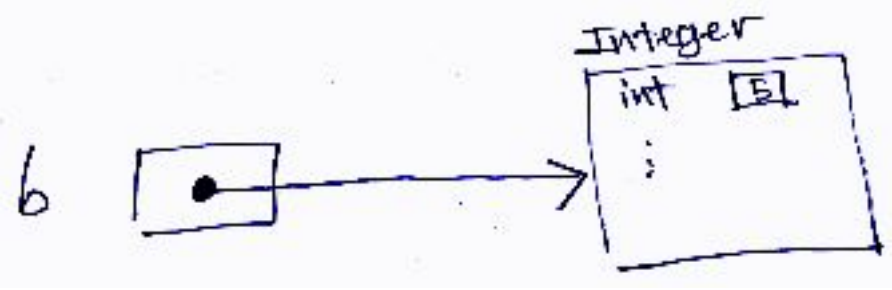
Current status as seen by Activity Monitor of the Mac OS processes in WI 18. Note the thread counts.

The second part of the assignment uses primitive types and pointers to create data structures. Why have we taught memory model diagrams? Why does Java allow us to have null pointers? While building our own data structure, the answer will become clear.

Java Pointers and References

Recall the memory model diagrams and the NullPointerException. When a variable is null, the diagram will not have an arrow. When a variable "points to an object" or "has a reference", then it points to an object.

Integer b;



Integer b = new Integer (5);

In *Java* *specifically*, "pointers" and "references" are interchangeable. It will no longer be clear enough for us to speak about "variables". In this assignment, we will use the word **pointer.** Why? **To get used to the idea of having pointers pointing to objects**.

A **null pointer** does not point to any **object**, and so when you do `null_pointer.any_variable` or `null_pointer.any_method()`, Java throws a NullPointerException. A Pointer of class Object points to an instance in memory of that Object.

Makefile

I have included a makefile to streamline compiling and running your code. To compile the Part 1 code, type **make part1a** or **make part1b**. Notice these two commands will automatically run the program for you as well. That was for convenience.

To compile the Part 2 code, type **make part2**. Notice this command will not run the program for you. To run all three of the above rules, type **make all**. To remove all local .class files, run **make clean**. To understand why these commands work, open the **Makefile** in the starter code. This will work only for UNIX-based operating systems or simulated UNIX consoles on Windows operating systems.

Part 1: Multithreaded Caesar Cipher Turtle GUI [30 pts]

Threads help our programs use time wisely. Consider a scenario where you are cooking and you have three tasks. (1) Boil soup, (2) slice your ingredients, and (3) put those ingredients inside the soup. One way to perform these tasks is to start boiling the soup and wait around, idling. Once the soup is boiling, then start to slice the ingredients. This is not efficient! It is possible start task 2 before task 1 finishes.

The result of Part 1 is two programs that help people visualize the difference between single threading and multi threading. As we had done in PSA 1, the program will take a string and encrypt it. Then, CSE 8A Turtles take that string, open a new World, and writes the encrypted string. The below screenshot is run on the String “ENCRYPTION”. We will develop one program with a single Turtle drawing all the letters out. Then, we will develop a separate program with one Turtle per letter. The visual contrast will show the desirability of multithreading.

Note: Multithreading allows us to run code **concurrently**. With multiple threads, we are able to switch between threads and move work forward if the processor was idling. But we still had a single processor which means lines of code are still run one at a time. To achieve **parallelism**, we would need multiple cores and code that assigns work to each processor. We get conceptually similar performance increase from both. But in this assignment, we are using **multithreading**.

Part 1 Policies:

The World is set with a width of 800 and a height of 120. Don’t change it. Your encryption must support at least 9 characters inclusively. We will test edge cases of primitive types and Object classes such as the empty string and null pointers. Legal strings for the encryption are strings of length 1 to length 9 inclusive, all capital letters and no symbols. Your programs must finish drawing/writing the encrypted string on the World within 10 seconds. Whether you remove the Turtles after they finish drawing does not matter. The letters must have uniform padding between each character and the world must comfortably display at least 9 characters. The letters must show the correctly encrypted result. The delay must be set to 75 before submitting.

Before reading Part 1A and 1B, you may want to take a look at all of the Turtle files first, or have them open while reading the next sections. Note the provided fields.

Part 1A: Single-Threaded Turtle

Public Class EncryptionTurtle

EncryptionTurtle.java has a main method. You can add your test cases there. To run the encryption, you will create a new Turtle for each case.

To encrypt “POTATO” with a rotation of ten, add a line of code to main as follows:

```
new EncryptionTurtle (“POTATO”, 10); // Example how we will call your code
```

Notice that we did NOT save the pointer of this newly allocated Turtle. That’s because we won’t need to use the Turtle instance for the remainder of the program. Use the Makefile to streamline compiling and running the code (see above). If you would like, you may add your tester code in a separate Java file that only contains the main method. The following will work:

```
java -cp ./turtleClasses.jar YourTesterFile
```

Take a look at DrawingTurtle.java. All it has are the abilities to draw out the letters. Make sure to use these functions to draw out your letters rather than reimplementing them in EncryptionTurtle.java.

EncryptionTurtle subclasses DrawingTurtle, which means EncryptionTurtle can also draw. Use inheritance properly to avoid busy work. EncryptionTurtle has one and only one constructor that takes in (String, int). Since we are constructing the classic CSE 8A Turtle, we must [1] call the super(). Look at the parameters for DrawingTurtle. We must [2] create a new World(int width, int height).

Remember to **`setPenWidth (int width)`** and **`setPenColor (Color.YOUR_FAVORITE_COLOR_HERE)`**. They belong to the Turtle class. Feel free to make these fancy. Here comes the fun part.

Recall from PSA 1 where the encrypt method takes in a String and a rotation, and you used a helper method called letterOperation to rotate each letter of the String one at a time. Do the same thing here. The EncryptionTurtle constructor’s parameters gives you everything you need. Helpful pieces of code might be **`toCharArray()`** and the for-each loop **`for (T t: t_array)`**. where `t_array` is of `type t[]` or `ArrayList<T>.` Read the for-each loop as

“**for each element t of type T in t_array…**”

Pass each encrypted letter to the draw method as defined in DrawingTurtle. Use inheritance. Don’t overengineer your solution. For each new EncryptionTurtle created, the program will open a new World and write the encrypted string in the World. If there are 5 new EncryptionTurtles, then there will be 5 different Worlds each with their own respective Strings. Don’t forget the Part 1 policies as mentioned above.

In the constructor, check that the parameter String is legal as defined in the policies. Should an illegal String be passed in, use **`System.err.println()`** and **not `System.out.println()`** saying an illegal String was passed in. Do not throw an exception -- it will cause the entire program to fail which is undesirable. Imagine a case where we have multiple calls to the EncryptionTurtle’s constructor within the same main method. As the program designer, we shall make the decision that it’s okay for some strings to fail (and simply print to stderr a complaint) and others to succeed within the whole program.

Part 1B: Multi-Threaded Turtle

Public Class EncryptionTurtleMT

Recall the steps in Part 1A: Create a Turtle which will create a World, then encrypt the string, then draw the string. Which step wastes the most time? (Whereby "waste", I mean the program is running yet not doing anything.) Creating a new World is likely not idling -- we may assume that the constructor was well implemented. Encrypting the String is likely not idling -- unless you put sleep code in the encryption, it will not waste time. Note, this is different from inefficient algorithms. Idling means that the running program literally does nothing.

Recall that the constructor for DrawingTurtle has a delay where the Turtle will wait [delay] time before drawing the next line. The Turtle was the culprit! If you set the DELAY to 0, notice that drawing is done instantly. Set the DELAY back to 75. We don't want to overwork the single Turtle. Instead, for each letter encrypted, another new Turtle will take care of drawing that letter.

Since we decided that constructing a new world does not waste time, we will not multithread it. The encryption also likely does not waste time, but if you like, you can multithread the encryption. From this point on, we will assume that we are just multithreading the Turtles.

The order of this algorithm is different. We will call your code through a ***static*** function called encryptMT that takes as parameters a String and an int:

```
EncryptionTurtleMT.encryptMT("POTATO", 10); // Example static call for your code
```

This is different from Part 1A; we begin our algorithm using a static call rather than through a constructor. The general approach is the following. (1) Create a new World. (2) It is up to you to decide how and when to encrypt each character. (3) For each letter, create a new EncryptionTurtleMT. The parameters of EncryptTurtleMT's constructors are up to you to decide, but remember that you must call the super() constructor in the first line. (4) After configuring your Turtle's settings, make it "run". (5) In the run method, call the turtle's draw method. Arguments cannot be passed into run() so these arguments must be passed as instance variables.

You might loop through each letter and for each of those letters, create a new EncryptionTurtleMT. The World will be populated by numerous Turtles. Although *technically* each Turtle was made to "run" at different times, the animation will *feel* like all the Turtles start at the same time.

Remember to use DrawingTurtle via inheritance. Think about your solution before attempting it. Don't overengineer the program. Draw memory model diagrams to help think through the algorithm. Remember to look at the lecture slides and textbook. Recall the Part 1 policies as mentioned above. Don't call `run()` directly.

1A and 1B Debrief

What will be the performance gain? When we perform a letterOperationMT() on each character, we will do it **as many as possible concurrently**, which will have a runtime of NOT (the time it takes to encrypt all chars), but rather only (the time it takes to encrypt all chars) divided by (the number of parallel threads). However, if we keep increasing the number of threads, we will find that eventually the program is using its time "wisely" - while the program is running, there is always a Turtle drawing. For example, if the delay allows us to have 500 Turtles draw a line, then having 1000 threads (1000 Turtles) draw would not result in a performance increase because the time was already being used wisely by the first 500 Turtles.

If we keep increasing the thread count to, say, a million, we actually find the performance will decrease. That's because the thread controller (Java, in our case) has trouble keeping up with managing all the threads. Thus, to have a performance gain, multithreading must be used wisely.

You may wish to skip to Part 3 now, which has questions related to Part 1.

Part 1C: For Curiosity Only [Not for credit]

If you feel like experimenting, you can go back to your PSA 1 code and apply multithreading on your code. Then run the encrypt on the full Odyssey file. Recall that we provided a shortened version of the Odyssey and asked you to run the PSA1 encrypt methods on that file. See if there are any performance gains when multithreading is applied and see the performance gains on the full file. Note the number of characters in the file. To see a contrast, keeping your original PSA1 code while writing a new copy of the encrypt method that integrated threading is recommended.

Example multithreading cases: run letterOperation using multithreading (one thread per some subset of letters to encrypt). Encrypt multiple files concurrently within a single program and to see whether doing that is faster than encrypting files in series. If you attempted this and want to share, write your findings in a file called `Part1C.md`

Part 2: MyStringBuilder [60 pts]

Strings have an implementation detail in programming languages. In the Java programming language, Strings are an actual class/type. In the C programming language, Strings are not a class and are really char arrays. Furthermore, in Java, we have multiple classes that implement sequences of characters in varying ways (char arrays, ArrayLists of Characters, Strings, StringBuilder). These all have similar capabilities with different implementations internally. Here, we now explore one internal implementation by writing a simple string data structure. While it is based on the StringBuilder interface, the internal workings of our StringBuilder will be one we have not seen before.

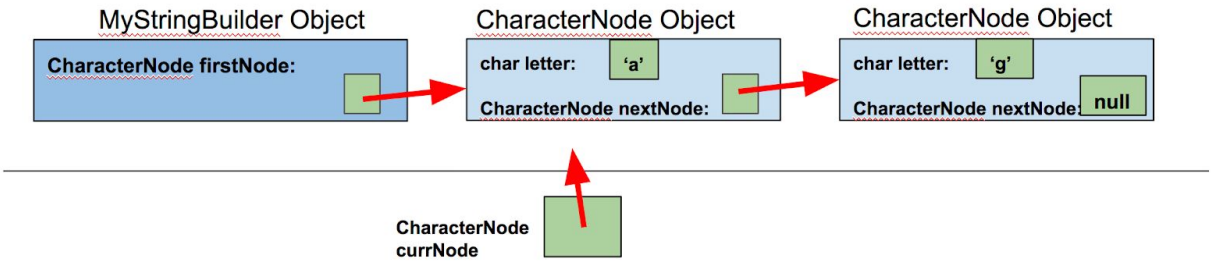
Let's discard the notion that each element in the data structure is next to each other. In fact, we will not care at all where objects are stored in memory. Instead, we will make the objects themselves remember the memory location of the next object through pointers.

Why do this? Suppose we misspelled "CSE" and meant to spell "CASE". An array would have {'C', 'S', 'E'}. We would have to create a new array, move 'S' and 'E' back, and then assign to position 1 the letter 'A'. Instead, we will make each letter **know** what is the next letter. To insert a letter is to **edit** those pointers to point at the new letters.

C → S → E puts A between C and S. Get: C → A → S → E.

But we are working with a primitive type which is not an Object subclassed type. There are a few ways to add functionality to a type, but what we will do is compose the char inside a class whose sole purpose is to hold chars and pointers to the next char in the sequence. This class is called a node.

Below is a memory model diagram to provide a visual context for the following methods. The light blue and darker blue boxes represent objects while the green boxes represent the actual value. The red arrows represent pointers. Remember that pointers are simply memory addresses, and in order to access elements, Java **automatically** follows the pointer to the actual object. As you work through the following methods, you may find that drawing your own memory model diagrams may help you think through the algorithms.



This MyStringBuilder has "ag". Notice that the chars are not inside MyStringBuilder.

Upon finishing the implementation of MyStringBuilder, the data structure will be able to store infinitely many characters (theoretically).

Java Language Details

To “iterate” through a MyStringBuilder, you need a way to keep track of where you currently are. You will use an additional CharacterNode pointer for that (e.g. the currNode pointer in the figure above). Notice two pointer pointing to the first CharacterNode.

The Java garbage collector will delete all Objects that no longer have any pointer pointing to it. If you unintentionally lose the only pointer of an CharacterNode, you will lose that CharacterNode and ALL subsequent nodes to the Java garbage collector! (On the flip side, the way to intentionally delete a Node is to do some type of pointer rerouting such that a certain Object has zero pointer pointing to it…)

Part 2 Policies:

We WILL check primitive type and Object class edge cases. All instance variables must be named correctly, else the script will fail. There is no starter code. Use of libraries and other data structures of any kind will result in zero.

The way we ask you to implement this data structure is not the most efficient. That’s fine; we deal with efficiency in later courses. However, your program must work within two minutes for long sequences of characters of length, say, 1 000 000. If it takes more than a few seconds to run the program, there may be something wrong with the algorithm.

Consider using recursive helper methods. Read Part2A through Part2G before programming to understand why a helper is desirable. Loops are also fine. Develop slowly. Test heavily. Required variable and function names must match exactly, else the scripts will fail. No extra instance variables or constructors are allowed, else the implementation will lose points. No use of external libraries are allowed. We will check. There is no starter code.

Part 2A: Define our two Classes

public class CharacterNode

This class contains a private final char type named letter and a public CharacterNode pointer named nextNode. Next, define the one and only public constructor. This takes in one parameter: a char, initializing the instance char letter. Leave the nextNode pointer as null. **We will not allow a no-argument constructor to exist**.

We now see one reason why Java allows null pointers. nextNode is null if and only if it is the last character, and nextNode cannot be null if it is not the last node. Refer to the diagram again.

Implement a getter method with no parameters to return the letter. It is called `getLetter`. Do not write a setter for nextNode.

public class MyStringBuilder

Our custom MyStringBuilder class will have no constructor and only one field variable: a pointer to our first node.

Part 2B: Append a Letter

MyStringBuilder: public void append (char addingChar)

Append a char to the end of the MyStringBuilder. If MyStringBuilder is empty, then firstNode must point to a new CharacterNode containing this char. Otherwise, traverse through all nodes until you arrive at the last node. Recall that the last node's nextNode pointer is always null. The previously last node will start pointing at a newly created CharacterNode containing addingChar.

HINT: You must "remember" where you are as you traverse through the nodes.

Part 2C: Output as a String

MyStringBuilder: public String toString ( )

Recall the goal of this data structure: to implement a class that ultimately results in a String. This function accomplishes that final goal. Yet, what we currently have is not a char array or a String, but a sequence of nodes of chars. This function takes this sequence of CharacterNodes and turns the sequence of chars into a String.

Part 2D: Output the Length

MyStringBuilder: public int length ()

It is useful to know the data structure's size. When the user executes this method, traverse through the nodes counting each one until the last node is reached. Return the number of nodes counted.

Test Your Code

At the point, we already have a partially implemented data structure! Test the above three methods **extensively** to prove that [1] the CharacterNode works, [2] appending chars to MyStringBuilder works, [3] outputting the data structure as a String works, and [4] returning the lengths work. This is not necessarily an exhaustive list. Ensure correctness to avoid cascading logic errors. As you work on the next few functions, you may want to use the previous few functions.

Part 2E: Insert a Letter, Take Care of Exceptional Input

MyStringBuilder: public void insert (int offset, char insertChar) throws IndexOutOfBoundsException

The offset is the position in our sequence of nodes where we want to insert a char. Once finished, insertChar will have position offset. The previous node at position offset will be at position offset + 1. In fact, all nodes whose positions were [offset] or higher has their positions incremented.

For example, if a string was “rest”, and I called insert (3, ‘e’). Before insert was called, at position 3 was the letter ‘t’. Now, ‘e’ is at position 3 while ‘t’ is at position 4, resulting in the string “reset”. Note that the node containing ‘s’ must point to a different node now.

The offset must be non-negative (0 is not negative) and less than or equal to the length of the sequence. If the parameter offset is invalid, then use throw new IndexOutOfBoundsException(String yourMessage), where yourMessage is your own helpful message. Do not throw any other types of exceptions.

Part 2F: Delete a Range of Chars, Take Care of Exceptional Input

MyStringBuilder: public void delete (int start, int end) throws StringIndexOutOfBoundsException

This method deletes a subsequence of characters from MyStringBuilder. (Deleting the entire sequence is also considered deleting a subsequence.) See Java Language Details on how to delete Java objects.The nodes for deletion starts at position start and deletes up to, but not including, the node at position end. If start == end, then nothing is deleted. For example, consider the string “abc”. delete (1,1) does nothing and delete (1, 2) will delete ‘b’ and cause the node containing ‘a’ to point at the node containing ‘c’. Assuming string ‘abc’ again, delete (0, length()) will delete the whole sequence.

The parameter start must be nonnegative, and less than or equal to end. If start is negative, or greater than end, or greater than the length of the string, then throw new StringIndexOutOfBoundsException (String yourMessage). Notice this is a different exception. Do not throw any other kinds of exceptions. The parameter end can be greater than length().

Part 2G: Compare MyStringBuilder with MyStringBuilder and String

[1] MyStringBuilder: public boolean equals (Object other)

[2] MyStringBuilder: public boolean equals (String other)

Both Strings and MyStringBuilders are sequences of characters. Do a character-by-character check between this (the MyStringBuilder calling object) and other (the parameter which will be either String or MyStringBuilder; we will consider any other types not comparable and thus not equal).

If this and other have the same sequence of characters, then return true. Otherwise return false. Ensure you consider all possible checks. If you prefer, you can check the type is either MyStringBuilder or String using just the first signature [1] to avoid overloading .equals() . We explicitly ban the use of any function that conveniently returns a String or character iterable (lists and arrays) and compares between the objects here. Your implementation shall be in your own words.

Part 3: README.md [10 pts]

No summary questions. If interested, lookup “Markdown Language” and how it relates to READMEs.

Makefile

If you had edited the Makefile, note how to run your rule setups.

Conceptual Question

In the context of **`encryptMT()`**, given a legal 8-character String, two threads in use simultaneously, and every character takes the same amount of time to encrypt and draw, **how long will the encryption take to finish the drawing?** The unit of time is **`charTime`**, the amount of time it takes to encrypt and draw one character. You should consider overhead as in lecture. Include all possible reasons in your answer. Draw out a diagram if it helps. For example, here’s a small one to start with.

Single Thread View:

```
---------------------------
| g | r | e | e | n | e | g | g |
---------------------------

----------- time -------->
```

Pointer Questions

In your own words, what are the benefits of the way we used pointers to build MyStringBuilder?

Suppose I wanted to **reuse** the nodes that have the same characters. Assuming I only have capital letters as characters, that would mean I have at most 26 nodes at once. How would you modify the way the SBNodes used pointers to allow this to happen? Consider the case where I have a word like CHEESE. There would be one node with the letter E in it. How would you know to go from E to E, then from E to S, then stop at E?

Design Decision

You either used loops or recursion or a mix for MyStringBuilder. Which one did you choose and why?

Despite your choice, what are the benefits of each of those techniques? What are the consequences of each? As a starting point, consider the existence of parameters.

Part 4A: Multithreaded Drawing [+4 EC, all/none]

In a file called **`DesignerTurtle.java`**, write a program that will use multiple Turtles with different pen colors to draw a picture. DesignerTurtle.java must be able to compile and run by itself with just the turtleClass.jar file alone, similarly to the EncryptionTurtle and EncryptionTurtleMT classes. Set the DELAY to a reasonably fast enough speed for it to draw in a couple of seconds. You may use multithreading, but it is not strictly necessary. We will run your design by executing:

```
java -cp ./turtleClasses.jar;. DesignerTurtle
```

Refer to DrawingTurtle on how to draw. Think of something complicated, use multiple colors, and think about how you would draw the design. Feel free to set any value for angles, lengths, pen width, pen color, etc. Feel free to use recursion if desired, but beware of exponential growth*. This section is totally up to you.

Designs that are seen as too simple or inappropriate will not be awarded and the grader has full jurisdiction to decide what is “designed enough”. We won’t accept the identically basic recursive tree made from lines as from PSA 5, BUT that does not mean we will not accept trees and you may draw a more complicated tree using recursion. (Use a different principle to draw it.) Examples of past designs include Pikachu, the Portal logo, and anime. Examples of previous unaccepted designs include stars, happy faces, and shapes (we already did that). We will record and showcase the most creative designs!

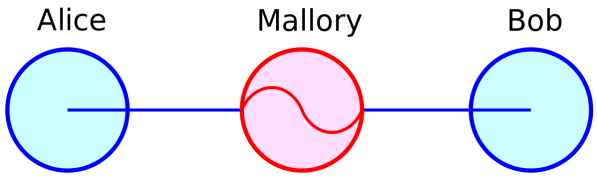
* In PSA 5, the recursion used to draw the tree was a method that would call itself twice. This is exponential growth. Assuming the use of a variable n that counts layers of stacks, there would be 2^n stack frames. Assuming (very roughly) that each frame is 1 byte and we used n=50. (2^50) will use 1.25 petabytes of memory.

Part 4B: Decryption in MyCaesarCipher [+6 EC]

absurdly known as the Possibly Multithreaded Very Secure CaesarCipher Turtle GUI StringBuilder Decrypter 9000

This section requires your EncryptionTurtleMT and MyStringBuilder classes to be correct.

Scenario: Someone is trying to send you a message, but you don't know who for certain. The sender believes you possess the ***Caesar Cipher***, an efficient, advanced, and completely secure algorithm written in Java that will make sure nobody but you can ever decode the secret messages. They are trying to send you about 4-6 words that are encrypted to the standards we used in PSA 8 Part 1 (capital letters, no other symbols). Sadly, the message will not arrive completely -- an attacker will have managed to intercept the message before it reached you and had removed the rotations!



From Wikipedia

The Caesar Cipher is no longer enough. Your mission now is to build the ***PMVSCCTGUISBD 9000***, a more powerful and absolutely secure data structure to help you quickly decrypt messages. Although there are only five strings this time, you anticipate you may receive hundreds of these secret messages in the future.

To receive the message, run **`~/../public/psa8-message/ReceiveMessage [username]`**. Make sure you are in your own directory when running that command. Don't try to copy this file out or opening it; it is a binary. Note that this program will work only on ieng6. ReceiveMessage will receive your message from the sender. The message is a Java file called **`SecretMessage.java`** to be used as a tester for MyCaesarCipher. Please see https://www.youtube.com/watch?v=-RkCMOA0s5c for demonstration.

Do not share the decrypted message nor the rotations you used with any other students -- you'll never know whether the message was an authentic message or a forged one.

Technical Info

MyCaesarCipher must have the following instance variables: an int[] called rotations and a MyStringBuilder[] called words.

MyCaesarCipher must have one constructor whose parameter is a String[]. Each String in the array is one of the words in the message from the tester file. Instantiate rotations and words using the length of the String[] parameter.

MyCaesarCipher has a void method called play() with no parameters. Print out each word and rotation pair, and then use Scanner and System.in to ask the user which word she wants to rotate. The user will input an int designating which word was chosen. The program will print out the chosen word and

rotation, and ask if it is a word. The user will enter ‘y’ or ‘n’. If ‘n’, then increment the rotation by 1. You may want to consider also supporting decrementing the rotation by 1. If ‘y’, then store the string and rotation back into the instance variables. The program will return back to the prompt asking which word the user wants to rotate. When all is finished, the user will input ‘**d**’, which goes to the next step.

EncryptionTurtleMT.encryptMT (word, 0); will be called on each word. Remember that each stored word is already rotated, which is why we call encryptMT with rotation 0. Each word will be displayed in separate windows. Drag and drop to reorder the rotated words into a sentence. When you close a World, the program should terminate but it’s okay if it doesn’t. Once you have found the message, you solved the decryption.

In a new file called Decrypted.txt, in the first line write all the rotations separated by spaces. Mod all your rotations by 26. In the second line write the decrypted message with each word separated by spaces. Both the rotations and the words must be in the correct order. Submit both files. In the README.md, provide a description for how to use your MyCaesarCipher.java. What are the commands used to compile it? What user inputs are you supporting?

Debrief

Why all the absurdity with using MyStringBuilder instead of String, and why not just guess the order of the words from the print statements? Because in order to make MyCaesarCipher work with those two other classes, you must match the types. One way to do expert programming is to think in types.

As if that was not enough, for fun draw a memory model diagram for MyCaesarCipher. (Diagram not for credit)

Motivation

This part is based on the ideas of the Man-In-The-Middle Attack and encrypting personal data to make sure it is seen only by authorized people. HTTPS, which encrypts your bytes between your web browser and the server, is one defense mechanism against someone trying to wiretap your internet connection. SSH is another connection that encrypts your data between you and the server. Learn more about cryptography in CSE 107, and computer security in CSE 127.

Why will Google Chrome mark HTTP unsafe?

https://www.wikiwand.com/en/Man-in-the-middle_attack

https://www.wikiwand.com/en/HTTPS

Notes and Warnings

- **Do not attempt to brute-force through the assignment if you are lost.** If you find that you are not understanding something in the writeup, ask a question on Piazza or go to tutor hours.
- **Follow directions.** There are many ways to implement a program for the same result. That is not what programming assignments are for. Not only must the behavior be correct, but also you must implement the code as given in the directions.
- **Test heavily to ensure maximum possibility of points.**
- **Back up your code.** Save multiple versions of your code as you progress through your assignment. For now, you could do this by copying your files over to different folders or uploading to your private Google Drive or Dropbox. The act of posting your code publically, including but not limited to Github's public repositories, is a violation of academic integrity.
- **As always, style guidelines must be followed**. Up to 10 points can be lost. <Link to Style Guidelines>
- **The total number of points possible is 110/100**.

Submit on Vocareum

***READ THE SUBMISSION SCRIPT*, WHICH TELLS YOU IF YOUR FILES ARE MISSING, NAMED INCORRECTLY, OR DO NOT COMPILE !!**

- `Makefile`
- `EncryptionTurtle.java` `// Part 1A, remember to set the DELAY to 75`
- `EncryptionTurtleMT.java` `// Part 1B, remember to set the DELAY to 75`
- `CharacterNode.java` `// Part 2A`
- `MyStringBuilder.java` `// Part 2B-G`
- `README.md` `// Part 3`
- `DesignerTurtle.java` `// Part 4A, Extra Credit File`
- `MyCaesarCipher.java` `// Part 4B, Extra Credit File`

In the CSE 8 series, we explored introductory programming in Java, used command statements (if, else, for, while), enhanced pictures, generated music, and developed simulations. We wrote classes, tested code, implemented a game, integrated object oriented programming practices, strategized algorithms, stylized programs with GUI, glanced at runtime comparisons, sped up a program through multithreading, and defined a simple data structure.

With that, we have wrapped up the last assignment in *Intro to Computer Science: Java I & II*.

Congratulations on completing these courses' assignments and good luck in your future ventures!