# Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-004-S2024/it114-sockets-part-1-3-checkpoint/grade/msa224

IT114-004-S2024 - [IT114] Sockets Part 1-3-Checkpoint

Submissions:

Submission Selection

1 Submission [active] 2/22/2024 1:13:27 AM

Instructions

COLLAPSE

1. Create a new branch for this assignment
2. Go through the socket lessons and get each part implemented (parts 1-3)
    1. You'll probably want to put them into their own separate folders/packages (i.e., Part1, Part2, Part3) These are for your reference
    2. Part 3, below, is what's necessary for this HW
    3. https://github.com/MattToegel/IT114/tree/Module4/Module4/Part3
3. Create a new folder called Part3HW (copy of Part3)
4. Make sure you have all the necessary files from Part3 copied here and fix the package references at the top of each file
    1. Add/commit/push the branch
    2. Create a pull request to main and keep it open
5. Implement **two** of the following **server-side** activities for all connected clients (majority of the logic should be processed server-side and broadcasted/sent to all clients if/when applicable)
    1. Simple number guesser where all clients can attempt to guess while the game is active
        1. Have a /start command that activates the game allowing guesses to be interpreted
        2. Have a /stop command that deactivates the game, guesses will be treated as regular messages (i.e., guess messages are ignored)
        3. Have a guess command that include a value that is processed to see if it matches the hidden number (i.e., */guess 5*)
            1. Guess should only be considered when the game is active
            2. The response should include who guessed, what they guessed, and whether or not it was correct (i.e., Bob guessed 5 but it was not correct)
        4. No need to implement complexities like strikes
    2. Coin toss command (random heads or tails)
        1. Command should be something logical like /flip or /toss or /coin or similar
        2. The result should mention *who* did *what* and got what *result* (i.e., Bob Flipped a coin and got heads)
    3. Dice roller given a command and text format of "/roll #d#" (i.e., roll 2d6)
        1. Command should be in the format of /roll #d# (i.e., roll 1d10)
        2. The result should mention *who* did *what* and got what *result* (i.e., Bob rolled 1d10 and got 7)
    4. Math game (server outputs a basic equation, first person to guess it correctly gets congratulated and a new equation is given)
        1. Have a /start command that activates the game allowing equaiton to be answered
        2. Have a /stop command that deactivates the game, answers will be treated as regular messages (i.e., any game related commands when stopped will be ignored)
        3. Have an answer command that include a value that is processed to see if it matches the hidden number (i.e., */answer 15*)

    1. The response should include who answered, what they answered, and whether or not it was correct (i.e., Bob answered 5 but it was not correct)
5. Private message (a client can send a message targetting another client where only the two can see the messages)
    1. Command can be /pm, /dm followed by the user's name or an @ preceding the users name (clearly note which)
    2. The server should properly check the target audience and send the response to the original sender and to the receiver (no one else should get the message)
    3. Alternatively (make note if you do this and show evidence) you can add support to private message multiple people at once. Evidence should show a larger number of clients than the target list of the private message to show it works. Note to grader: if this is accomplished add 0.5 to total final grade on Canvas
6. Message shuffler (randomizes the order of the characters of the given message)
    1. Command should be /shuffle or /randomize (clearly mention what you chose) followed by the message to shuffle (i.e., /shuffle hello everybody)
    2. The message should be sent to all clients showing it's from the user but randomized
        1. Example: Bob types */command* hello and everyone recevies Bob: lleho
6. Fill in the below deliverables
7. Save the submission and generated output PDF
8. Add the PDF to the Part3HW folder (local)
9. Add/commit/push your changes
10. Merge the pull request
11. Upload the same PDF to Canvas

**Branch name:** M4-Sockets3-Homework

Tasks: 7 Points: 10.00

Baseline (2 pts.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

**Text: Demonstrate Baseline Code Working**

Details:
This can be a single screenshot if everything fits, or can be multiple screenshots

Checklist

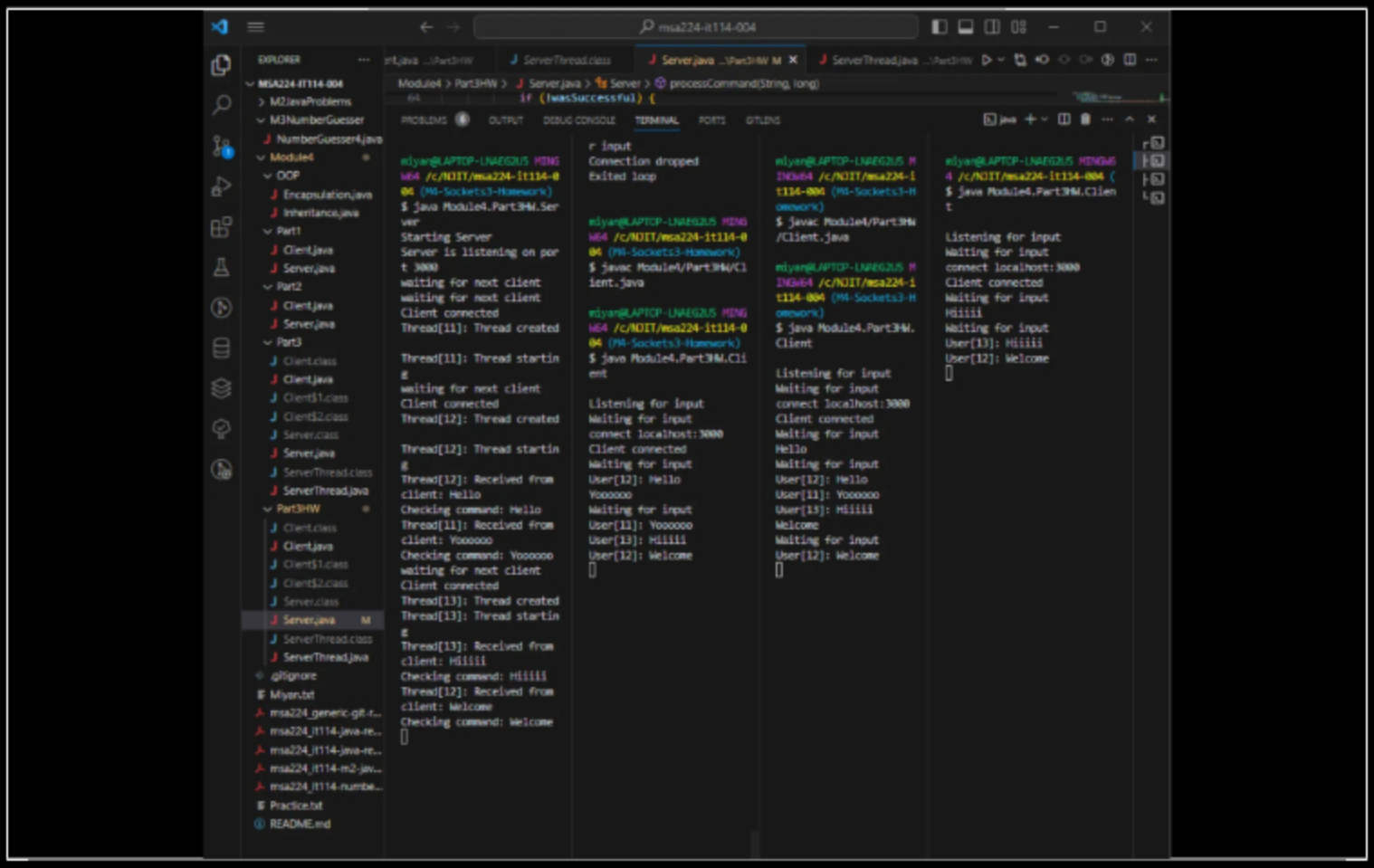
*The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Server terminal/instance is clearly shown/noted |
| #2 | 1 | At least 3 client terminals should be visible and noted |
| #3 | 1 | Each client should correctly receive all broadcasted/shared messages |
| #4 | 1 | Captions clearly explain what each screenshot is showing |
| #5 | 1 | Include a screenshot showing you grabbed Parts 1-3 correctly and have them in your repository alongside Part3HW |

Task Screenshots:

Gallery Style: Large View

Small Medium Large




This screenshot shows everything asked for, it shows the files from part 1-3 and part 3 hw are in my folders correctly and in my repository. it shows the clear terminal and baseline code working with 3 clients minimum that are also receiving all messages.

Checklist Items (5)

#1 Server terminal/instance is clearly shown/noted

#2 At least 3 client terminals should be visible and noted

#3 Each client should correctly receive all broadcasted/shared messages

#4 Captions clearly explain what each screenshot is showing

#5 Include a screenshot showing you grabbed Parts 1-3 correctly and have them in your repository alongside Part3HW

Feature 1 (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: What feature did you pick? Briefly explain how you implemented it

Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Feature is clearly stated (best to copy/paste it from above) |
| #2 | 1 | Explanation sufficiently and concisely describes implementation (should be aligned with code snippets in related task) |

Response:

The first feature I chose was,

1. Coin toss command (random heads or tails)
    1. Command should be something logical like /flip or /toss or /coin or similar
    2. The result should mention *who* did *what* and got what *result* (i.e., Bob Flipped a coin and got heads)

To achieve this feature what I had to do was make a process command in the code, luckily there was one already setup so it wasnt as hard since I essentially in a way didnt have to type it from scratch and I had the code existing to use as a baseline. So using that I made mine for the coin flip. All I had to add that was new was the broadcast message of course. At first I made it seperate but I got an error which then I realized I cant have duplicates so I combined the process commands with if else statements.

Other than that, I had to make the string to execute the actual coin toss, it took some refreshing on java rules and math and what not to figure it out, but in the end I got it and it wasnt as hard as I thought.

^COLLAPSE^

Task #2 - Points: 1

**Text: Add screenshot(s) showing the implemented feature working (code and output)**

Details:
Add screenshots of the relevant code changes AND relevant output during runtime

Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Output is clearly shown and captioned |
| #2 | 1 | Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code |

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
} else if (message.equalsIgnoreCase(anotherString:"/cointoss")) {  //msa224 2/22/24
    String result = executeCoinToss();
    broadcast(String.format(format:"User[%d] initiated a coin toss and got %s", clientId, result), clientId);
    return true;
```

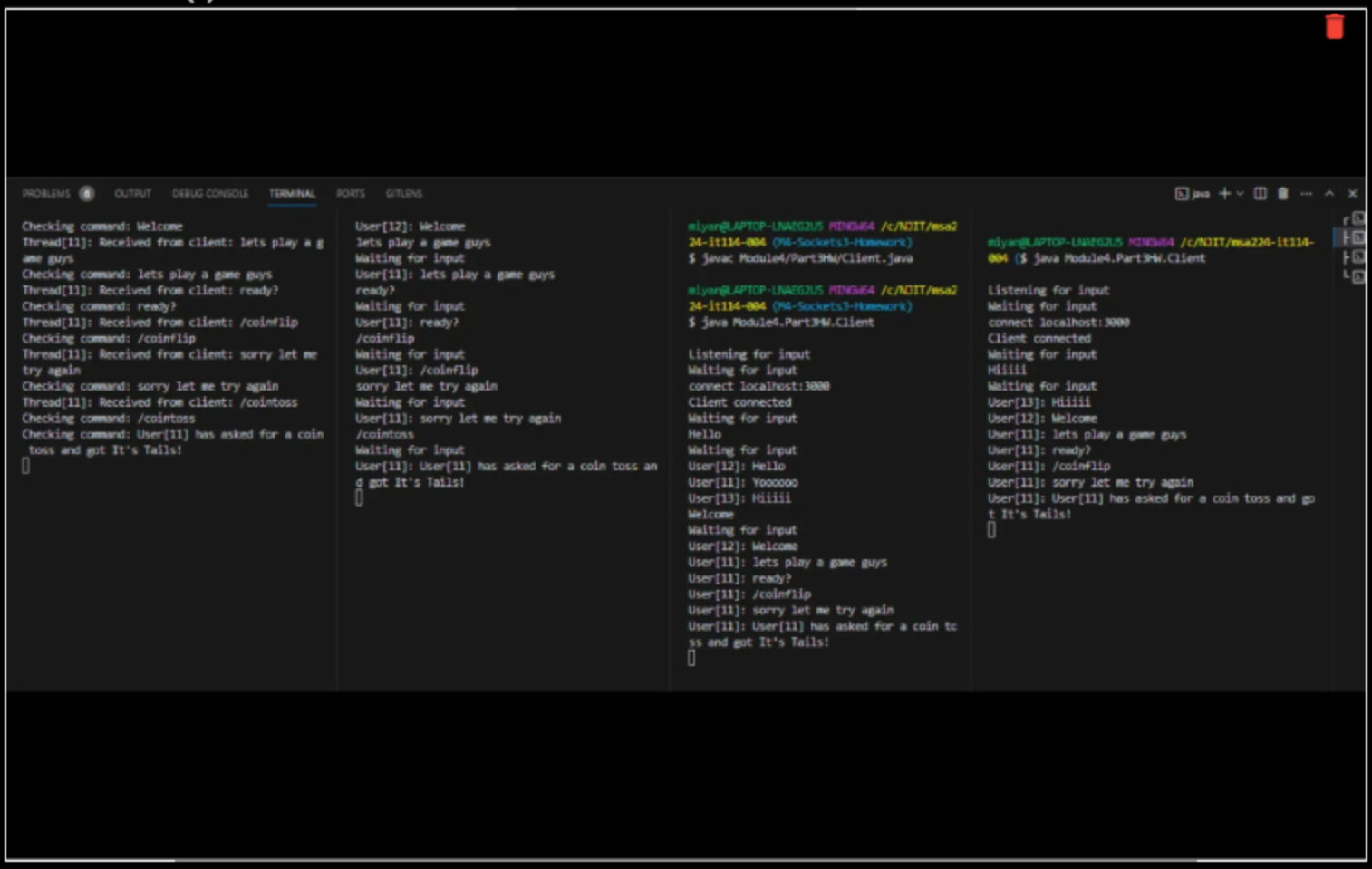
This shows the if else statement I made for it

Checklist Items (0)

```
private String executeCoinToss() { //msa224 2/22/24
    int randomNum = (int) (Math.random() * 2);
    return (randomNum == 0) ? "You got Heads!" : "You got Tails!";
}
```

This shows the string i made to actually execute the flip.

Checklist Items (0)




So this is the game being played and all users being able to see the result as well as see who initiated.

Checklist Items (1)

#1 Output is clearly shown and captioned

## Feature 2 (3 pts.)

^COLLAPSE^

^COLLAPSE^

### Task #1 - Points: 1

**Text: What feature did you pick? Briefly explain how you implemented it**

Checklist

*The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Feature is clearly stated (best to copy/paste it from above) |
| #2 | 1 | Explanation sufficiently and concisely describes implementation (should be aligned with code snippets in related task) |

Response:

The second feature I chose was,

1. Message shuffler (randomizes the order of the characters of the given message)
    1. Command should be /shuffle or /randomize (clearly mention what you chose) followed by the message to shuffle (i.e., /shuffle hello everybody)
    2. The message should be sent to all clients showing it's from the user but randomized
        1. Example: Bob types /*command* hello and everyone recevies Bob: lleho

To achieve this feature what I had to do was make a process command again in the code. Once again I kind of had a baseline to use looking at my other commands so using that I modeled my new one for shuffle.

COLLAPSE

Task #2 - Points: 1

**Text: Add screenshot(s) showing the implemented feature working (code and output)**

Details:
Add screenshots of the relevant code changes AND relevant output during runtime

Checklist

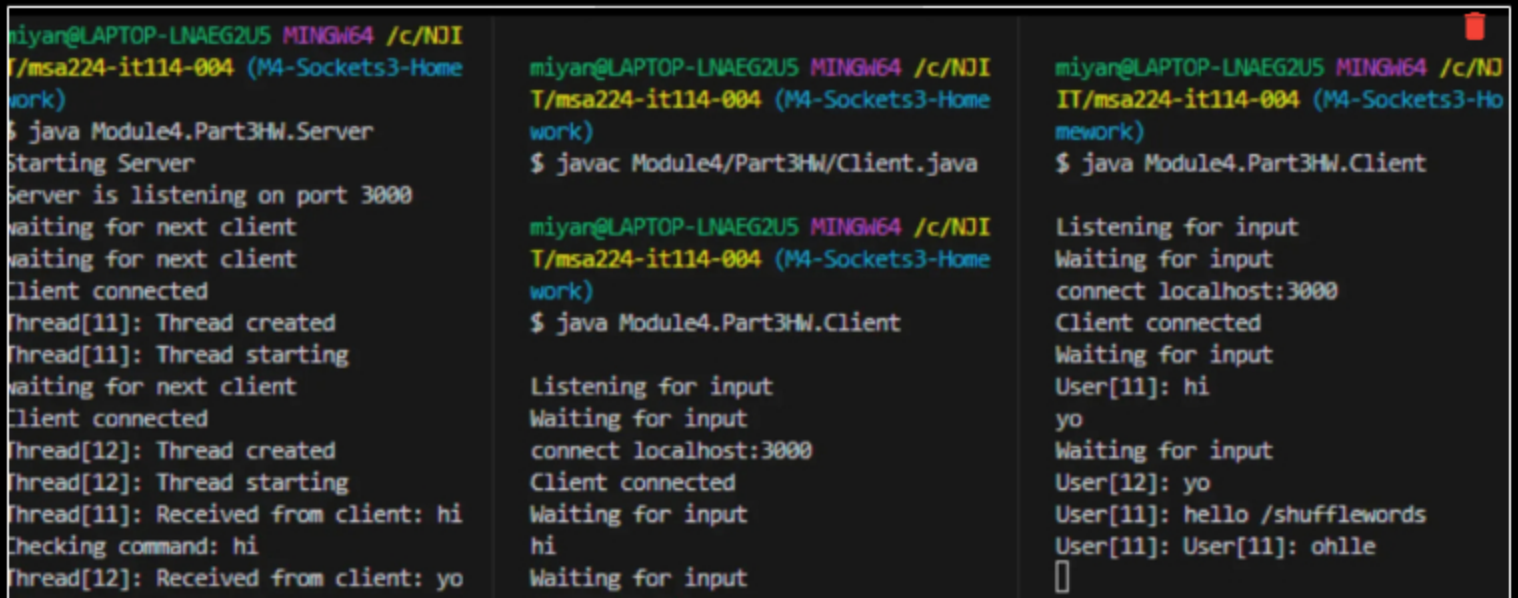
*The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Output is clearly shown and captioned |
| #2 | 1 | Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code. |

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
Checking command: yo
Thread[11]: Received from client: he
llo /shufflewords
Checking command: hello /shuffleword
s
Thread[11]: Received from client: /s
hufflewords hello
Checking command: /shufflewords hell
o
Checking command: User[11]: ohlle
```

```
User[11]: hi
User[12]: yo
hello /shufflewords
Waiting for input
User[11]: hello /shufflewords
/shufflewords hello
Waiting for input
User[11]: User[11]: ohlle
```

Clearly shows output and the shuffle words working in the terminals

Checklist Items (1)

#1 Output is clearly shown and captioned

```
    System.out.println("Checking command: " + message);
    if (message.toLowerCase().startsWith(prefix:"/shufflewords ")) { //msa224 2/22/24
        String[] parts = message.split(regex:" ", limit:2);
        if (parts.length == 2) {
            String originalText = parts[1];
            String shuffledNewMessage = shuffleMessage(originalText);
            broadcast(String.format(format:"User[%d]: %s", clientId, shuffledNewMessage), clientI
        } <- #76-80 if (parts.length == 2)
        return true;
    } else if (message.equalsIgnoreCase(anotherString:"/cointoss")) {  //msa224 2/22/24
        String result = executeCoinToss();
        broadcast(String.format(format:"User[%d] initiated a coin toss and got %s", clientId, resu
        return true;
    } else if (message.equalsIgnoreCase(anotherString:"disconnect")) {
        Iterator<ServerThread> it = clients.iterator();
        while (it.hasNext()) {
            ServerThread client = it.next();
            if (client.getId() == clientId) {
                it.remove();
                disconnect(client);
                break;
            } <- #90-94 if (client.getId() == clientId)
        } <- #88-95 while (it.hasNext())
        return true;
    } <- #86-97 else if (message.equalsIgnoreCase("disconnect"))
    return false;
} <- #72-99 private boolean processCommand(String message, long clientId)

private String shuffleMessage(String message) { //msa224 2/22/24      You, 11 minutes ago • Uncom
    char[] characters = message.toCharArray();
    for (int x = 0; x < characters.length; x++) {
        int y = random.nextInt(characters.length);
        char temp = characters[x];
        characters[x] = characters[y];
        characters[y] = temp;
    } <- #103-108 for (int x = 0; x < characters.length; x++)
    return new String(characters);
} <- #101-110 private String shuffleMessage(String message)

private String executeCoinToss() { //msa224 2/22/24
    int randomNum = (int) (Math.random() * 2);
    return (randomNum == 0) ? "You got Heads!" : "You got Tails!";
}
```

Shows updated and relevant code with specifics requested like ucid and date, regarding the feature at hand

Checklist Items (1)

#2 Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code.

Misc (2 pts.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

**Text: Reflection: Did you have an issues and how did you resolve them? If no issues, what did you learn during this assignment that you found interesting?**

Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | An issue or learning is clearly stated |
| #2 | 1 | Response is a few reasonable sentences |

Response:

I did have some issues during this homework. It was a bit challenging making the implementations as coding isnt my strong suit but using w3schools and using google to brush up my skills I was able to get a working code. The hardest part for me was making the string to actually shuffle the message. The coin toss was a bit easier but challenging nonetheless. Having that baseline to refer back to when making my processes and things like that helped alot cause most of it was simply re typing that same thing out and adding it to the process but tweaking small things to make it specific. The hardest part for me was the math behind the shuffle words. I had to use my resources to figure out how to make it work and knock the rust off. I used the w3 schools articles and even went back to some CS113 notes I had to relearn it.

COLLAPSE

Task #2 - Points: 1

**Text: Pull request link**

Details:

URL should end with /pull/# and be related to this assignment

URL #1

https://github.com/msa224/msa224-it114-004/pull/8

End of Assignment