# Homework 3 – Pirate Fudd Talk

## Problem Description

This assignment will test your knowledge of String parsing, JavaDoc, and object-oriented principles.

For this assignment, you will be creating a translator that takes in an English sentence and translates it into Pirate and Elmer Fudd sentences. It is up to you how to achieve this functionality. Feel free to create all the helper methods that you need. Your program must utilize good **design principles**, **JavaDocs**, as well as **Checkstyle**.

The first variant you need to incorporate is Pirate. If Pirate is selected, perform the following word replacements:

**and → ’n** **my → me** **is → be** **--ing → --in’** **are → be** **you-- → ye—** **am → be**

Assume that all words are separated by a single space. Thus “is”, “are”, and “am” should be replaced with “be”. Words **ending** with “—ing” should be changed to “—in”. Words **beginning** with “you—” should be replaced with “ye—“. Additionally, you should always add “Yarr!” to the front of pirate sentences.
For example:

- Are you ready kids? → Yarr! Be ye ready kids?

The second variant you need to incorporate is Elmer Fudd. If Elmer Fudd is selected, perform the following text replacements:

**“th” → “d”** **“l” → “w”** **“ith” → “if”** **“r” → “w”**

You should replace all substrings on the left with the substring on the right. If the rules conflict, (for example “ith” and “th”), the longer substring should be replaced first.
For example:

- That is very cool! → Dat is vewy coow!

Make sure that your capitalization matches the capitalization of the input text. (The first letter of the replaced substring should match the first letter of the substring you are replacing.)

## Solution Description

You are free to design this program as you want. However, your program must have a driver class named PirateFudd that includes the `main()` method. Your main method must do the following:

When running the program, you should first prompt the user for their input sentence. Then prompt whether they would like to convert to Pirate with a (Y/N) input. Then prompt for Elmer Fudd variant with a (Y/N) input. If you receive a “Y” answer for either part, immediately print out the appropriate translation. Keep prompting for user input if neither a “Y” nor a “N” is provided. If both Pirate and Elmer Fudd are "Y”, then first print the Pirate translation, and then perform the Elmer Fudd translation on the Pirate sentence (see example output).

You should then print out the past 3 translations (not including those from the current sentence). If there haven’t been three prior translations yet, then print out the 0, 1, or 2 prior translations. You must utilize a new class named History. You may design and incorporate History however you wish, but the past translations should display both the English sentence and the translated sentence. If the past translation had both Pirate and Elmer Fudd applied, then displayed translated sentence should be the Elmer Fudd applied onto Pirate.

Finally, prompt the user whether they would like to translate a sentence again. Keep prompting for user input if neither a “Y” nor a “N” is provided.

Example Output: (User input in bold)
> Input English Sentence: **How are you?**
> Pirate (Y/N): **Y**
> Yarr! How be ye?
> Elmer Fudd (Y/N): Y
> Yaww! How be ye?
> Past Translations:
> That is very cool! -> Dat is vewy coow!
> Are you ready kids? -> Yarr! Be ye ready kids?
>
> Would you like to translate again (Y/N)? **N**

Even though you are free to implement your own program style, you should think carefully about the design of your program and its classes. To achieve the best score, you should follow the principles of modularity and encapsulation in your design. Try not to overload any one class or method with too much to do.

## Submitting

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `PirateFudd.java`
- `History.java`
- `Additional classes that you created`

Make sure you see the message stating, "Homework 3 submitted successfully." We will only grade your last submission be sure to **submit *every file* each time you resubmit.**

## Import Restrictions

You may only import Scanner.

## Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

## Checkstyle and Javadoc

You must run Checkstyle on your submission. (To learn more about Checkstyle, examine our course's external website Java Resources page under "CS 1331 Style Guide".) The Checkstyle cap for this assignment is **5 points**. If you don't have Checkstyle yet, download it from Canvas → files → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.
Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

## Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due. Only post code on Piazza in a **private** post.

## Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs.

The Gradescope autograder visible to you is **NOT** comprehensive. A few test cases will be hidden when you submit. This is done intentionally so that you learn to write your own test cases for your assignments.