Assignment 1: Java finger exercises

Due dates:

- **Examplar submissions: Tuesday, Sept 19 at 8:59pm**
- **Implementation: Tuesday, Sept 19 at 8:59pm**
- **Self-evaluation: Wednesday, Sept 20 at 11:59pm**

Starter files: code.zip

1 Purpose

The primary goal of this assignment is to get you writing Java (again or for the first time). As a secondary goal, completing this assignment will help you ensure that you have a properly configured and working Java dev[illegible]at you can use the submission system.

2 [illegible]

Bef[illegible]nsure that you have all the necessary tools to complete an assi[illegible]

- [illegible]
- [illegible]
- [illegible] server to submit an assignment
- [illegible]st and get answers to common questions

Please review the directions on the course page if you need help setting up any of this, or talk to the course staff.

3 Representing simple e-books

In this assignment, we'll think about e-books and some possible data representations for them. For now, we will ignore punctuation marks completely (i.e. will not check for them in arguments, and pretend they don't exist in examples), and expect only two behaviors:

- Count the number of words in an e-book (where words are separated by whitespace)
- Search for a given word in the e-book

The starter code contains two classes that implement e-books: a *section*, and a *paragraph*. The Javadoc in those classes should explain what they are intended to represent.

4 Examplar: Creating examples to guide implementations

A key task in understanding the meaning of a data definition is creating sufficient examples to illustrate its intended behavior in a representative variety of settings. In the next few assignments, you will be creating and submitting examples *without* having a working implementation to start from. In this assignment, you have been given starter code that contains *mostly working* implementations. There are several bugs that have been deliberately left in the starter code.

Your initial task is to write a collection of test methods in the `ExamplarEBooks` class (in the *default* package, not in the same package as the EBooks starter code) that demonstrate how these two classes are *supposed* to work. This means that when you write your examples and run them on the starter code, they should not yet all pass.

For this part of the assignment, you will submit *only* the `ExamplarEBooks.java` file:

- Either as a single standalone file
- Or as a properly-structured zip file. When your Examplar class is in the default package, you do not *need* the zip, but it's good practice for future assignments

Regardless of which way you submit the code, *do not* submit the starter code we give you.

You should not proceed to the next portion of this assignment until you have (nearly) 100% Correctness and Thoroughness score on your Examplar submission. (We obviously cannot prevent you from jumping ahead, but you will get more benefit from this portion of the assignment if you work through it before the remaining pieces.)

Hints for this assignment:

- There are two methods in this interface, and they both seem very simple: so how might each of them go wrong?
- One method produces an `int` from text, or from lists of content: are there other (incorrect!) ways to generate numbers from such information?
- Read the Javadoc for the interfaces given to you: what cases seem to be indicated by the Javadoc that are ignored in the code given to you?
- All of the chaffs come from bugs in the starter code. There are many other potential chaffs we could devise, but the purpose of this assignment is to read, understand, and diagnose code that was given to you.
- Each example you write should be short and simple: it should construct an e-book and measure some property of it. You do not need to create large examples in order to find the chaffs here.

5 Correct the bugs

Once you've completed Examplar above, repair the starter code we give you and correct the bugs in it. You should now be able to run your Examplar tests against your own code, and they should all pass.

6 Adding a class

In lecture, we discussed designing an interface for a data defintition to allow us to *add new implementations* of that interface that made different design tradeoffs. You will repeat this process here for a different data definition.

As we did in class, you will create a new class `Hyperlink` (in package `cs3500.hw01.ebooks`) that implements the `EBookFlow` interface. A hyperlink consists first of a flow of content to show, and second a link destination (which in a "real" e-book reader would be opened in a browser when the person clicked on the link). It must define a public constructor with two arguments as described above.

Write a JUnit 4 test class, `EBookTests`, that include the *new* tests you add for this method — keep your existing Examplar tests in the existing `ExamplarEBook` class.

7 Adding a method

Add a new method to `EBook`,

```
String format(int lineWidth);
```

The intent of this method is to "render" the e-book such that it fits on a screen of the given width (measured in characters), without any horizontal overflow. This method should compute the rendered e-book as a single string as follows:

- All chunks should be formatted, and concatenated together with a blank line (i.e., `"\n\n"`). Section titles should be formatted as if they were paragraphs, and separated from their content by a single `"\n"`.
- All flows should be line-wrapped to fit as many words as possible on the current line, with line-breaks inserted only between words (i.e. replace the space between words with a `"\n"`). Words within a flow should be separated by a single space.

Your method may need to throw exceptions: if so, they must be documented in your Javadoc for the method.

You may choose to change the implementation of the starter code given to you, if you think the existing represenation is inconvenient for implementing this method. If you do so, leave a comment within the Javadoc of the class, as follows:

```
/**
 * ... existing Javadoc ...
 * @implNote ...explain what you changed about the representation, and why...
 */
```

8 Grading standards

For this assignment, you will be graded on

- whether your code compiles,
- whether your code implements the specification (functional correctness),
- whether you thoroughly test every method that you write
- how well you follow the style guide.
- whether your program is suitably commented

8.1 The style guide

Coding style is important. For this class we follow Google's Java style guide. It's comprehesive but not very long, so I suggest reading the whole thing and then referring to it as needed.

While it can't yet take on full responsibility for formatting code—much less for programming style more broadly—your IDE may be able to help you follow the code formatting portion of the style guide:

- For IntelliJ IDEA, download `intellij-java-google-style.xml` and save it in the `config/codestyles/` subdirectory of your IntelliJ configuration folder. Then from within IntelliJ, go to *File > Other Settings > Default Settings…*, then in the dialog that pops up, go to *Editor / Code Style*, and select it in the *Scheme* dropdown, where it should have appeared as an option. If you've already created a project, then you'll additionally need to set this as the project style, using *File > Settings*.
- Check here for more info.

9 Submission

Ready to submit? Look at the Design Principles Master List first! (Not all items there may apply to this assignment)

9.1 Deliverables

- For Examplar: submit
  - *only* your `ExamplarEBooks.java` file
  - or a properly-structured zip containing *only* your `ExamplarEBooks.java` file
- For your implementation: submit a zip file containing
  - the code files defining your classes in the `cs3500.hw01.ebooks` package
  - any test files
  - any files you had to modify or create in order to implement and test your `format(String)` method.

  Please ensure that your submission is a zip file. This zip file should contain your `src/` and `test/` folders from your project, and *only* those folders. These folders should mimic the folder structure required for your packages. Please do not put these folders within another folder before submitting, as the grader will not find your files:

  A **properly-formatted** zip file:

  ```
  my-submission.zip
  +-src/
  | +-cs3500/
  |   +-hw01/
  |     +-ebooks/
  |       +-Java files for ebooks
  +-test/
    +-ExamplarEBooks.java
    +-whatever other tests you wrote...
      possibly in packages...
  ```

  A **improperly-formatted** zip file:

  ```
  my-submission.zip
  +-My Awesome Homework 1/
    +-src/
    | +-cs3500/
    |   +-hw01/
    |     +-ebooks/
    |       +-Java files for ebooks
    +-test/
      +-ExamplarEBooks.java
      +-whatever other tests you wrote...
        possibly in packages...
  ```

9.2 Instructions

You will submit the assignment on the Handins server. Follow these instructions:

1. Log in to the server using the link above, using your Khoury account (**not your Northeastern account**). Follow directions on the course page if you do not have a Khoury account or have forgotten your password.
2. You must be registered as a student to CS 3500 for Fall 2023 on the handin server. Follow directions on the course page if you have not applied for registration.
3. Submit each of the zip files you made above to the relevant assignments ("Hw 1 – Examplar" and "Hw 1") on the submission server. Note that for the autograded portions, it may take some time to see feedback. This time increases as we get closer to the deadline, because many more students tend to submit. Please be patient!
4. You have three submissions for this assignment: the Examplar submission, the implementation submission, and the self-eval. Make sure to submit to each of them by their respective deadlines.