CS310 – Summer 2024

Project 1: The Prerequisites

DUE: Sunday, June 2nd at 11:59pm
LATE SUBMISSION without penalty until Sunday June 30th

Submission Instructions

- Make a backup copy of your project folder!
- Remove all test files, `jar` files, `class` files, etc.
- You should just submit your java files.
- Do not Zip your files, just submit the `java` files required in one submission (on Gradescope)
- For instance, for this project, the submitted files should be:
  `AppleOrange.java`
  `SingleItemBox.java`
- Submit to Gradescope. DOWNLOAD AND VERIFY WHAT YOU HAVE UPLOADED THE RIGHT THING. Submitting the wrong files will result in a 0 on the assignment!

Basic Procedures

You must:

- Have code that compiles with the command: **`javac *.java`** in your project directory
- Have code that runs with the following command for part 1: **`java BoxUsageDemo`**
- Have code that runs with the following command for part 2: **`java AppleOrange`** ***`[number]`***

You may:

- Add additional methods and variables not specifically mentioned in the project description, however these methods and variables **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional public methods or variables.
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no **`ArrayList`**, **`LinkedList`**, **`HashSet`**, etc.).
- Use any arrays anywhere in your program (except **`args`** in **`main()`**).
- Add any additional import statements (or use the "fully qualified name" to get around adding **`import`** statements).
- Add any additional libraries/packages which require downloading from the internet.

Grading Rubric

For assignment with greater level of complexity, an accompanying grading rubric pdf will be provided. For this first assignment, you just need to make sure you pass all the tests and do not violate any constraints. This is an opportunity for you to review and apply the prerequisites skills. You can make multiple attempts until you pass all the tests and get full credit for this initial assignment.

You get full credit and pass the prerequisite test if:

1) Correct submission format: You submit the 2 java files requested (not zipped, correct extension, etc.)
2) Correct Submission Format:
    a. Code passes all checks for `cs310code.xml`
    b. and Code has a set indentation and formatting style which is kept consistent throughout and code looks "well laid out"
    c. and Code has good, meaningful variable, method, and class names.
3) Appropriate JavaDocs:
    a. Code passes all checks for `cs310comments.xml`
    b. and The entire code base is well documented with meaningful comments in JavaDoc format. Each class, method, and field has a comment describing its purpose. Occasional in-method comments used for clarity.

This project should be very straight forward and take only an hour or so to implement if you have all the prerequisite knowledge from the prior courses. However, if it's been a while since you took CS211, or if you took CS211 at another university, or even if you just didn't get the best grade you could have in that class, you may have additional "catching up" to do.

Requirements

An overview of the requirements is below:

- Implementing the classes - You will need to implement required classes/methods.
- Style – You must follow the coding and conventions specified by the style checker.
- JavaDocs - You are required to write JavaDoc comments for all the required classes and methods.

JavaDocs?? Style Checker?? Yes, as you may often be asked to do this in a professional setting. You'll need to document your code correctly and conform to a given code style (many companies have their own style requirements, but there are also some standard ones like Sun and Google). We are using a subset of the Google standard for this class.

Part 1: Make a box that can hold *anything*

This first part checks that you know how to use generics, write JavaDocs, and use the basic command line tools you'll use throughout the semester.

You need to write a class called **SingleItemBox** (in a file called **SingleItemBox.java** that you create). This class has a constructor that takes a single item (of any type) and puts it in the box. You also need a method called **getItem()** which provides the item back to the user but does not remove it from the box (this is an "accessor", if you remember, sometimes called a "getter"). Make sure to comment your code as you go in proper JavaDoc style.

To check that you've got the right idea, we've provided a class called **BoxUsageDemo.java** which shows creating three different boxes that store different things (an apple, a banana, and a cat). You should not alter this class to "make it work", but rather alter your box to allow the provided code to work. You can use the command "**java BoxUsageDemo**" to run the testing code defined in **main()**. You could also edit this **main()** to perform additional testing (we won't be using **BoxUsageDemo** for testing, it's just a demo for you).

Note that JUnit test cases will not be provided for **SingleItemBox**, but feel free to create JUnit tests for yourself. A part of your grade will be based on automatic grading using test cases made from the specifications provided.

Checking That You've Got Style

Every professional developer needs to adhere to a coding style (indentation, variable naming, etc.). This is non-optional in 99.99999% of professional settings (aka. "jobs"). Normally, in school, a grader checks your style manually, but this is very inefficient since you only get feedback after your project is completed (not while you're writing), and it's very hard for someone to manually check these things.

We're going to help move you along the path to "coding with style" this semester. We have provided you with a command line tool that checks your style for you: **checkstyle** (https://checkstyle.org). This tool has plugins for Eclipse, NetBeans, jGRASP, and many others (see their website), but there is also a command line interface (CLI) which has been provided with this project (**checkstyle.jar**). This automatic checker is similar to ones used at large companies (like Google, Oracle, and Facebook). If you're curious, we're using a subset of Google's style requirements for this class.

The provided **cs310code.xml** checks for common coding convention mistakes (such as indentation and naming issues). You can use the following to check your style:

```
java -jar checkstyle.jar -c [style.xml file] [projectDirectory]/*.java
```

For example, if code is in folder **project1** checking for JavaDoc mistakes one would use:

```
java -jar checkstyle.jar -c cs310code.xml project1/*.java
```

Note: if you have a lot of messages from **checkstyle**, you can add the following to the end of the command to output it to a file called `out.txt`: **> out.txt**

If **checkstyle** finds nothing wrong you'll see "**Starting audit... Audit done.**" and nothing else.

Checking Your JavaDoc Comments

You should verify that your **JavaDoc** comments adhere to the proper format (especially if you're new to writing **JavaDocs**). We've provided a second checkstyle file (**cs310comments.xml**) that looks for JavaDoc issues and in-line comment indentation issues. You can run it the same way as cs310code.xml, for example:

```
java -jar checkstyle.jar -c cs310comments.xml project1/*.java
```

Part 2: Fruit and Numbers

This second part of the project checks that you know how to use command line arguments and write basic Java code (including exception handling), and it gives you more practice writing JavaDocs and using checkstyle. You'll also be able to see and use JUnit tests (such as the ones we use when grading).

Specification: Write a program (`AppleOrange`, that lives in `AppleOrange.java`). This program should print the numbers from 1 to X (inclusive), space separated, but for multiples of 3 print "`apple`" instead of the number, for multiples of 7 print "`orange`" instead of the number, and for the multiples of both 3 and 7 print "`appleorange`" (no space) instead of the number. X is a command line argument to the program. Don't forget to document as you go. Example program run (coloring added for readability):

```
> java AppleOrange 10

1 2 apple 4 5 apple orange 8 apple 10

> java AppleOrange 25

1 2 apple 4 5 apple orange 8 apple 10 11 apple 13 orange apple 16 17
apple 19 20 appleorange 22 23 apple 25

> java AppleOrange 50

1 2 apple 4 5 apple orange 8 apple 10 11 apple 13 orange apple 16 17
apple 19 20 appleorange 22 23 apple 25 26 apple orange 29 apple 31 32
apple 34 orange apple 37 38 apple 40 41 appleorange 43 44 apple 46 47
apple orange 50
```

Checking for Invalid Input

Your program should print the following (exactly) if the command line argument is missing, if the argument is not a positive number, or if too many arguments are given. This message should be sent to **System.err** not **System.out**.

Example commands which generate this error:

```
> java AppleOrange

> java AppleOrange 0
```

```
> java AppleOrange 1 1
```

```
> java AppleOrange apple
```

Error message:

```
One positive number required as a command line argument.
```

```
Example Usage: java AppleOrange [number]
```

Exactly Matching Output

Seven unit tests have been provided to automate checking that you are exactly matching the output required (**`AppleOrangeTest.java`**) since it is hard to eyeball differences in text. To run these tests, navigate to the directory above your project directory (from your project directory type **`cd ..`** to go up one directory) and do the following...

Compile and run the tests with the following in Windows:

```
javac -cp .;junit-4.11.jar;[projectDirectory] AppleOrangeTest.java
```

```
java -cp .;junit-4.11.jar;[projectDirectory] AppleOrangeTest
```

Or for Linux/Mac, replace all the semicolons wit colons in the classpath:

```
javac -cp .:junit-4.11.jar:[projectDirectory] AppleOrangeTest.java
```

```
java -cp .:junit-4.11.jar:[projectDirectory] AppleOrangeTest
```

Check Your Style and JavaDocs

Run both `checkstyle` files again on this new program to verify you got those basics down.

Command Reference

All commands are from inside your project folder and assume you left the style items and dictionaries in an outer folder (since you unzipped project0.zip into a single place as suggested and didn't move around any files).

From in your project directory:

Compile: **`javac *.java`**

Run Part 1: **`java BoxUsageDemo`**

Run Part 2: **`java AppleOrange [number]`**

Compile JavaDocs: **`javadoc -private -d ../docs *.java`**

From above your project directory:

Style Checker:

```
java -jar checkstyle.jar -c cs310code.xml [projectDirectory]/*.java
```

Comments Checker:

```
java -jar checkstyle.jar -c cs310comments.xml [projectDirectory]/*.java
```

Compile Unit Tests:

```
javac -cp .;junit-4.11.jar;[projectDirectory] AppleOrangeTest.java
```

Run Unit Tests:

```
java -cp .;junit-4.11.jar;[projectDirectory] AppleOrangeTest
```

Or for Linux/Mac users, make sure that you use `:` instead of `;` for the classpath (the argument to **`-cp`**).

Have fun programming !