EECS1022 (M,N,O) Winter 2021
Lab4
Loops and Arrays

Chen-Wei Wang

Release Date: Friday, February 5
Due Date: 23:59 EST, Friday, February 19

- You are required to **work on your own** for this lab. **No** group partners are allowed.

  Plagiarism checks will be run on all submissions, and suspiciously similar ones will be reported to Lassonde.
- To complete this lab, it is **strictly forbidden** for you to use any library class (e.g., `ArrayList`). Violating this requirement will cause a **50% penalty** on your lab marks.
- For this lab, you will be graded not only by JUnit tests given to you, **but also additional tests** covering some other input values. This is to encourage you to take more responsibility for the correctness of your code, by writing your own tests.
- Your lab assignment is **not** graded during the weekly scheduled lab sessions.
  - Follow the instructions to submit (via eClass) the required file(s) for grading.
  - There are three submission links: two of them are for late submissions only. If you already submit before the original deadline, do not make any more submissions, as the later submission (with a penalty) will be graded.
  - Emailing your solutions to the instructor or TAs will **not** be acceptable.
- Texts in blue are hyperlinks to the corresponding documents/recordings.

Policies

- **Your (submitted or un-submitted) solution to this lab exercise (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.**
- When you submit your lab, you claim that it is **solely** your work. Therefore, it is considered as **a violation of academic integrity** if you copy or share **any** parts of your Java code during **any** stages of your development.
- When assessing your submission, the instructor and TA may examine your code, and suspicious submissions will be reported to the department if necessary. **We do not tolerate academic dishonesty**, so please obey this policy strictly.
- You are entirely responsible for making your submission to the TA in time. Back up your work **periodically**, so as to minimize the damage should any sort of computer failures occur. Follow this tutorial series on setting up a **private** Github repository for your Java projects.
- The deadline is **strict** with no excuses. Refer to the course syllabus for the policy on a late submission.

Contents

Learning Outcomes

By completing the assigned exercises of this lab, you are expected to be able to:

1. Exercise a simple workflow of Github.
2. In the Eclipse IDE (Integrated Development Environment):
   - Import a starter project archive file.
   - Given a computational problem, develop a Java solution (i.e., a utility method) composed of:
     - Numerical Literals and operators
     - String Literals and operators
     - Variables and assignments
     - (Nested) Selections/Conditionals/If-Statements
     - Loops (`for`-loop vs. `while`-loop)
     - Arrays
   - Run a Java class with with the `main` method as a console Java application.
   - Use the given JUnit tests (calling the utility methods) to guide the development.
   - Use the **debugger** features (step over/into/out/return) to find defects in programs.
   - Export an existing project as an archive file.
3. Understand the separation of concerns (using packages): `model`, `console_apps`, and `junit_tests`.

Assumptions

- You have already setup a Github account and stored work in a **private** repository: `EECS1022-W21-workspace`.

  Note. You only submit your lab through eClass, not Github. Though not required, it is highly recommended that you adapt to the practice of backing your work using a versioning tool like Github.

- You are able to use Eclipse to complete the tutorial videos and this lab assignment on either your own machine or the EECS remote labs.

  Note. The starter project was created using Eclipse and an Eclipse project archive file is expected to be submitted. Therefore, you may not want to use other IDE such as IntelliJ.

Requirements of this Lab

- For this lab, you will be graded not only by JUnit tests given to you, **but also additional tests** covering some other input values. This is to encourage you to take more responsibility for the correctness of your code, by writing your own tests.
- To complete this lab, it is **strictly forbidden** for you to use any library class (e.g., `ArrayList`). Violating this requirement will cause a **50% penalty** on your lab marks.
- The grading of your lab will **start** by automatically **unzipping** the submitted Java project archive file (**.zip**) and extracting the required class(es). It is therefore crucial for you to follow **precisely** the spelling of the archive file name. **Penalty** will be taken if the grading cannot proceed due to carelessness on following the instructions in Section 2.3.
- Your lab submission will **only** be graded using JUnit (for this lab, the tests supplied to you plus some additional tests). That is, your lab submission will **not** be graded manually using the console application given to you.
- For the JUnit test class `TestUtilities.java` given to you:
  - Do **not** modify the test methods given to you.
  - You are allowed to add new nest methods.
- Do **not** modify the console application class (in package `console_apps`) given to you.
- For each utility method in the `Utilities` class that you are assigned to implement, as discussed in **Part F** of Week 1's Java tutorial videos:
  - No `System.out.println` statements should appear in each of the utility method.

    Instead, an explicit, final `return` statement is placed for you.
  - No Scanner operations (e.g., `input.nextInt()`) should appear in each of the utility method.

    Instead, refer to the input parameters of the method.
- You are welcome to ask questions related to this lab or the assigned tutorial videos on the forum. However, please be **cautious**:
  - You can help your fellow students understand the requirements of tasks.
  - **Do not share the code you developed to ask, or to answer, questions.**
    - Questions specific to the code you write would be best and most effectively addressed by TAs (during scheduled labs) or your instructor (during office hours or appointments).
    - Week 2's tutorial videos (Parts C to E) introduce to you **debugger in Eclipse**. You are encouraged to **set breakpoints and launch the debugger** when you are stuck at your own program.
  - **Hints** on how the solution should look like are **left only to the instructors** who moderate the forum.

1 Task 1: Complete Weekly Java Tutorial Videos

- For Lab4, you are assigned to study **Week 5 Part A to Part D** of the Java tutorial series:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_7Ri55ckAVTx0IbtcJEqzz8 [Week 5 only]

  To reference tutorial videos from the previous weeks, see:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_6wy2m0yq2wObIWPz4tAxW6 [All Weeks]

  These Java tutorial videos assigned to you are meant for you to:

  1. Obtain extra hands-on programming experience on Java, supplementing your weekly lectures.
  2. Complete the lab assignment with the necessary skills and background.

  Though we do **not** require the submission of the weekly Java tutorial project (like in Lab0), **examples and insights discussed in these tutorials will be covered in your (written and programming) tests and exam**: should you decide to skip the weekly tutorial videos, it would be your choice.

  As you study through the example Java classes in the tutorial videos, you are advised to **type them out** (but absolutely feel free to add new Java classes to experiment) on Eclipse.

- You can find the iPad notes of illustrations from the tutorial videos here:

  https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/EECS1022%20Tutorial%20on%20Java.pdf

Getting Prepared for Programming Test 2

- Your Programming Test 2 (shortly after the reading week) will cover loops and arrays.
- Three sets of practice test questions are available on the M,N,O eClass site (under the *Practice Programming Tests* section).

  Here is a tutorial series on the solution to each question:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_4UZNiLzeFPAgDDv2vLCGb4

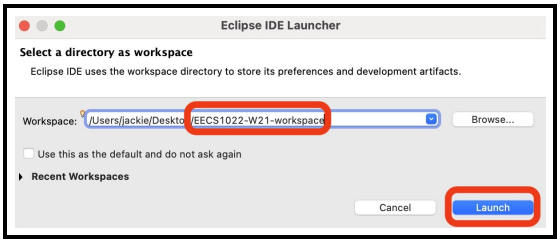
- You may consider tackling some of the practice tests, which will help you complete the lab exercises.

2 Task 2: Complete Programming Exercises

Starting Task 2 should mean that you have already completed the weekly Java tutorial videos (Section 1).

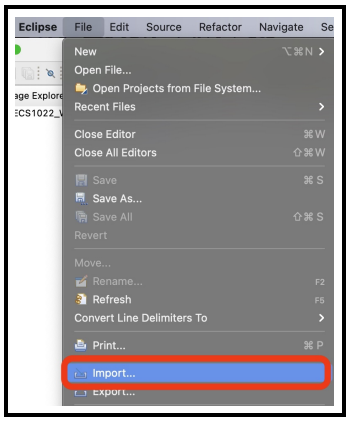
2.1 Step 1: Download and Import the Starter Project

1. Download the Eclipse Java project archive file from eClass: `EECS1022_W21_Lab4.zip`
2. Launch Eclipse and browse to `EECS1022-W21-workspace` as the `Workspace` then click on `Launch`, e.g.,

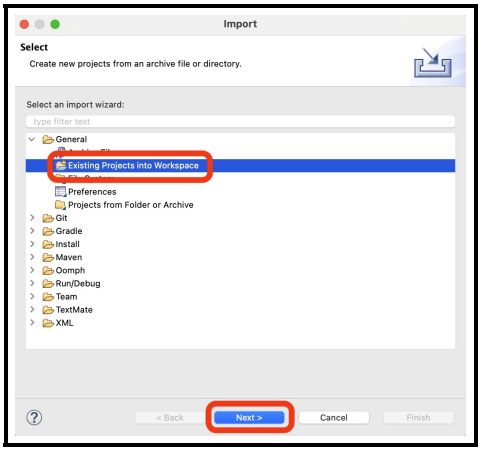


3. In Eclipse:

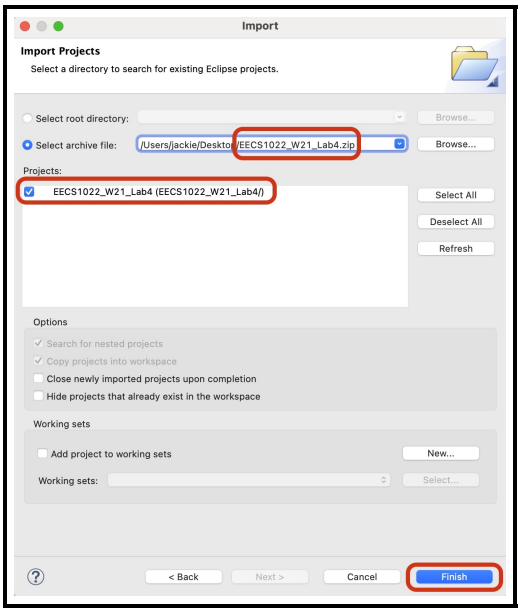
3.1 Choose `File`, then `Import`.



3.2 Under `General`, choose `Existing Projects into Workspace`.



3.3 Choose `Select archive file`. Make sure that the `EECS1022_W21_Lab4` box is checked under `Projects`. Then `Finish`.

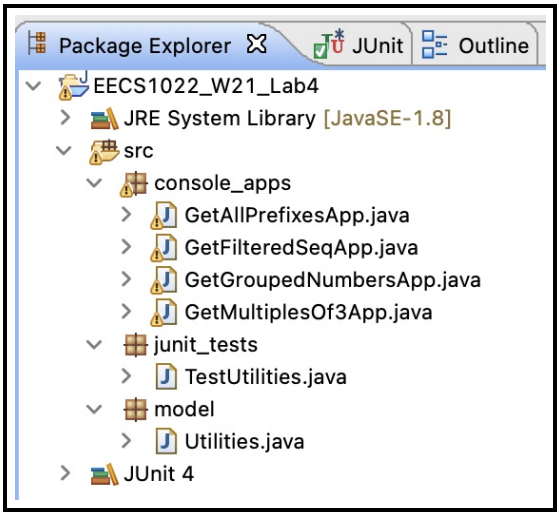


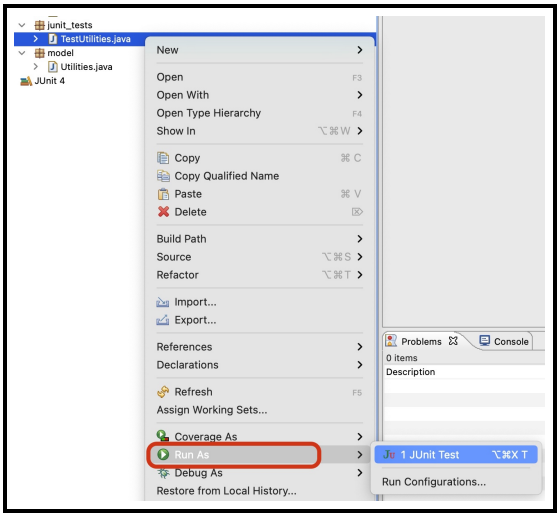
2.2 Step 2: Programming Tasks

From the `Package Explorer` of Eclipse, your imported project has the following structure.

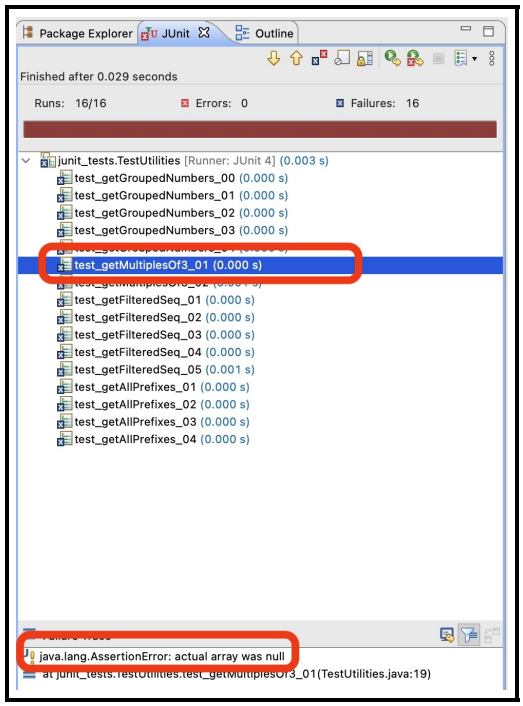
- You can manually test the assigned methods using the corresponding console application classes in package `console_apps`. These classes are completed and given to you. See below for more descriptions.
- Your goal is to pass **all** JUnit tests given to you (i.e., a **green bar**). To run them, as shown in the Java tutorials on Week 1, right click on `TestUtilities.java` and run it as JUnit tests. Of course, none of the given tests would pass to begin with.

  You must not modify these given JUnit tests.





How to Deal with a Failed JUnit Test? From the JUnit panel from Eclipse, click on the failed test, then **double click** on the first line underneath `Failure Trace`, then you can see the **expected value** versus the **return value** from your utility method.



2.2.1 Method to Implement: `getMultiplesOf3`

Problem. You are asked to implement a utility method which takes as input an array of integers and returns another array containing all those elements that are multiples of 3. For example, if the input array is:

```
<2, 1, 6, 5, 4, 3>
```

Then the output or returned array is:

```
<6, 3>
```

Note that the length of the output array corresponds the number of elements in the input array that are multiples of 3, meaning that if the input array is empty, or if there are no multiples of 3 in the input array, then the output array should be empty.

Also, the output array should preserve the order in which these multiples, if any, appear in the input array. For example, in the above example, multiples of 3 in the input array appear in the order of `6` followed by `3`, which is reflected in the output array.

Testing. Your goal is to first pass **all** tests related to this method in the JUnit test class `TestUtilities`. You are encouraged to write additional JUnit tests. **These tests document the expected values on various cases: study them while developing your code.**

However, use the console application class `GetMultiplesOf3App` if you wish (e.g., use the input and expected values from the JUnit tests). Here is an example run:

```
How many numbers do you want to input?
10
Enter input 1:
89
Enter input 2:
23
Enter input 3:
42
Enter input 4:
87
Enter input 5:
18
Enter input 6:
36
Enter input 7:
23
Enter input 8:
97
Enter input 9:
102
Enter input 10:
987
<42, 87, 18, 36, 102, 987>
```

Todo. Implement the `Utilities.getMultiplesOf3` method. See the comments there for the input parameters and requirements.

2.2.2 Method to Implement: `getFilteredSeq`

Problem. You are asked to implement a utility method which takes as input an array (say *numbers*) of integers and an integer (say n). Note that n may or may not exist in *numbers*. The utility method returns another array containing all elements in *numbers* that are not equal to n. For example, if the input n is 3 and the input array *numbers* is:

```
<3, 1, 2, 3, 4>
```

Then the output or returned array is:

```
<1, 2, 4>
```

Note that the length of the output array corresponds the number of elements in the input array that are not equal to n, meaning that if the input array is empty, or if n does not exist in the input array, then the output array should be just the same as the input array.

Also, the output array should preserve the order in which elements appear in the input array. For example, in the above example, elements in the output array appear in the order of `1` followed by `2` and then by `4`, which is reflected in the input array.

Testing. Your goal is to first pass **all** tests related to this method in the JUnit test class `TestUtilities`. You are encouraged to write additional JUnit tests. **These tests document the expected values on various cases: study them while developing your code.**

However, use the console application class `GetFilteredSeqApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here is an example run:

```
How many numbers do you want to input?
5
Enter input 1:
3
Enter input 2:
1
Enter input 3:
2
Enter input 4:
3
Enter input 5:
4
Which number do you wish to filter from the input numbers?
3
<1, 2, 4>
```

Todo. Implement the `Utilities.getFilteredSeq` method. See the comments there for the input parameters and requirements.

2.2.3 Method to Implement: `getAllPrefixes`

Problem. You are asked to implement a utility method which takes as input an array of integers and returns another array of strings, each of which denoting a non-empty prefix of the input array. For example, if the input array is:

```
<3, 1, 4>
```

Then the output or returned array of string values is:

```
<"[3]", "[3, 1]", "[3, 1, 4]">
```

Note that the length of the output array is equal to the length of the input array. Also, elements in the output array are "sorted" by the lengths of the prefixes (e.g,. the first element is the prefix of length 1, the second element is the prefix of length 2).

Testing. Your goal is to first pass **all** tests related to this method in the JUnit test class `TestUtilities`. You are encouraged to write additional JUnit tests. **These tests document the expected values on various cases: study them while developing your code.**

However, use the console application class `GetAllPrefixesApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here is an example run:

```
How many numbers do you want to input?
5
Enter input 1:
3
Enter input 2:
1
Enter input 3:
4
Enter input 4:
2
Enter input 5:
5
<[3], [3, 1], [3, 1, 4], [3, 1, 4, 2], [3, 1, 4, 2, 5]>
```

Todo. Implement the `Utilities.getAllPrefixes` method. See the comments there for the input parameters and requirements.

2.2.4 Method to Implement: `getGroupedNumbers`

Problem. You are asked to implement a utility method which takes as input an array of integers and returns another array grouping **all** elements from the input array as follows, from left to right: **1)** elements divisible by 3; **2)** elements divided by 3 with the remainder 1; and **3)** elements divided by 3 with the remainder 2. For example, if the input array is:

<2, 4, 6, 5, 1, 3>

Then the output or returned array is:

⟨ 6, 3 , 4, 1 , 2, 5 ⟩
Group 1 Group 2 Group 3

Note that the input and output arrays are equally long, meaning that if the input array is empty, then the output array should be empty.

Also, each group of the output array should preserve the order in which its elements appear in the input array. For example, in the above example, Group 2 of the output array (i.e., those divided by 3 with the remainder 1) contains `4` followed by `1`, reflecting their order in the input array.

Testing. Your goal is to first pass **all** tests related to this method in the JUnit test class `TestUtilities`. You are encouraged to write additional JUnit tests. **These tests document the expected values on various cases: study them while developing your code.**

However, use the console application class `GetGroupedNumbersApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here is an example run:

```
How many numbers do you want to input?
10
Enter input 1:
89
Enter input 2:
22
Enter input 3:
42
Enter input 4:
87
Enter input 5:
19
Enter input 6:
36
Enter input 7:
23
Enter input 8:
97
Enter input 9:
102
Enter input 10:
987
<42, 87, 36, 102, 987, 22, 19, 97, 89, 23>
```

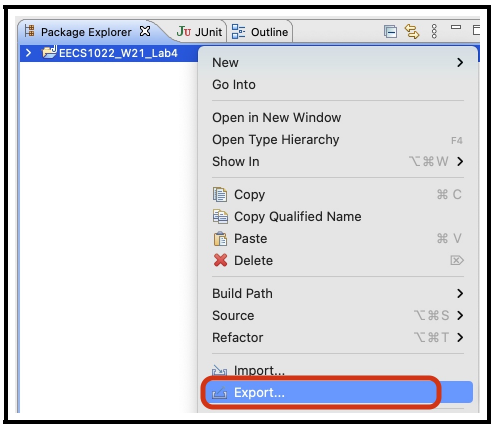
Todo. Implement the `Utilities.getGroupedNumbers` method. See the comments there for the input parameters and requirements.

2.3 Step 3: Exporting the Completed Project

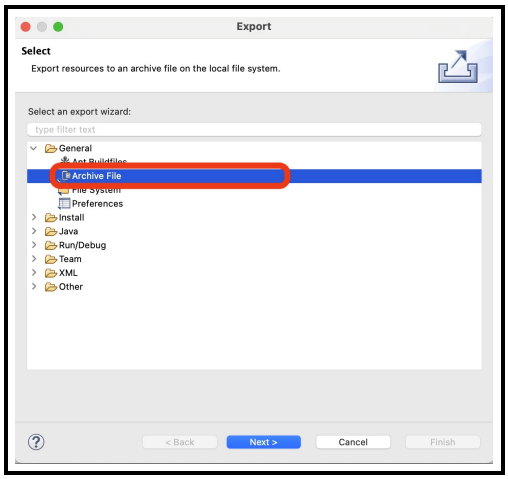
You are required to submit a Java project archive file (.zip) consisting all subfolders.

In Eclipse:

1. Right click on project `EECS1022_W21_Lab4`. Then click `Export`



2. Under `General`, choose `Archive File`.

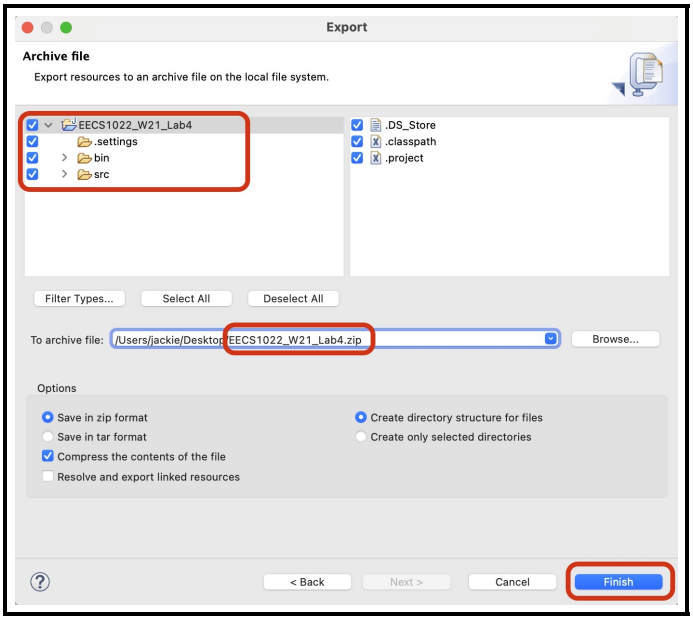


3. Check the top-level `EECS1022_W21_Lab4`

Make sure that all subfolders are checked: `.settings`, `bin`, and `src`.

Under `To archive file:` browse to, e.g., desktop, and save it as `EECS1022_W21_Lab4.zip` (**case-sensitive**)

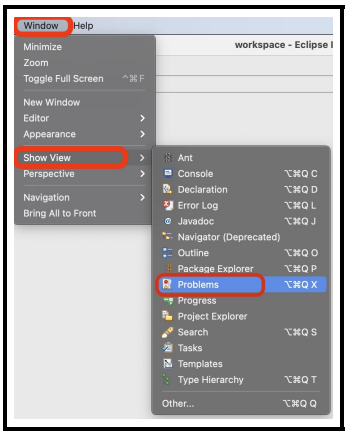
Then `Finish`.



Note. In case you have concerns about exporting and submitting the **.setting** subfolder: it will be kept confidential and access-protected on eClass.

3 Submission

1. Before you submit, you must make sure that the `Problems` panel on your Eclipse shows **no errors** (warnings **are** acceptable). In case you do not see the `Problems` panel: click on `Window`, then `Show View`, then `Problems`.



   Submitting programs with errors (meaning that it cannot be run for grading) will result in possible partial, but low, marks.

2. Section 2.3 asks you to **export** the Java project as an archive file:

   `EECS1022_W21_Lab4.zip`

   Before you submit, verify that its unzipped version has the following structure:

```
EECS1022_W21_Lab4
├── bin
|   ├── console_apps
|   |   ├── GetAllPrefixesApp.class
|   |   ├── GetFilteredSeqApp.class
|   |   ├── GetGroupedNumbersApp.class
|   |   └── GetMultiplesOf3App.class
|   ├── junit_tests
|   |   └── TestUtilities.class
|   └── model
|       └── Utilities.class
└── src
    ├── console_apps
    |   ├── GetAllPrefixesApp.java
    |   ├── GetFilteredSeqApp.java
    |   ├── GetGroupedNumbersApp.java
    |   └── GetMultiplesOf3App.java
    ├── junit_tests
    |   └── TestUtilities.java
    └── model
        └── Utilities.java
```

Figure 1: Lab4 Expected Project Structure

3. Go to the eClass site for Sections M,N,O: https://eclass.yorku.ca/eclass/course/view.php?id=6214
4. Under the `Lab Submissions` section, click on `Lab4` to submit the Java archive file: `EECS1022_W21_Lab4.zip`
   - You may **upload** as many draft versions as you like before the deadline.
   - You must explicitly **submit** the draft version for grading before the deadline.
   - **Once you click on the `submit` button, you can no longer upload another draft version.**

4 Amendments

Clarifications or corrections will be added to this section.

-