EECS1022 (M,N,O) Winter 2021
Lab3
Simple Loops

Chen-Wei Wang

Release Date: Friday, January 29
Due Date: 23:59 EST, Friday, February 5

- 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 arrays or any library class (e.g., `ArrayList`). Violating this requirement will cause a **50% penalty** on your lab marks.

- Starting from 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.

  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
     - Simple Loops (`for`-loop vs. `while`-loop)
   - 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

- Starting from 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 arrays or 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 Lab3, you are assigned to study **Week 4 Part A to Part E** of the Java tutorial series:

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_4yEdqdvaQH4LppQvGstofS` [Week 4 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:

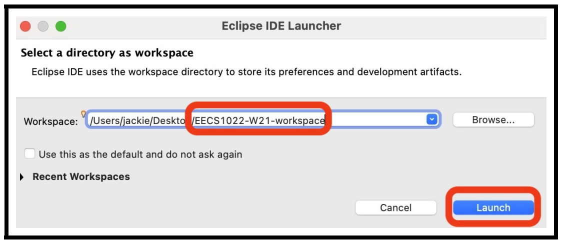
  `https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/EECS1022%20Tutorial%20on%20Java.pdf`

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_Lab3.zip`
2. Launch Eclipse and browse to `EECS1022-W21-workspace` as the `Workspace` then click on `Launch`, e.g.,



Eclipse IDE Launcher

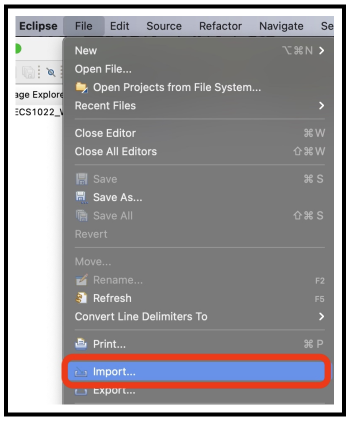
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

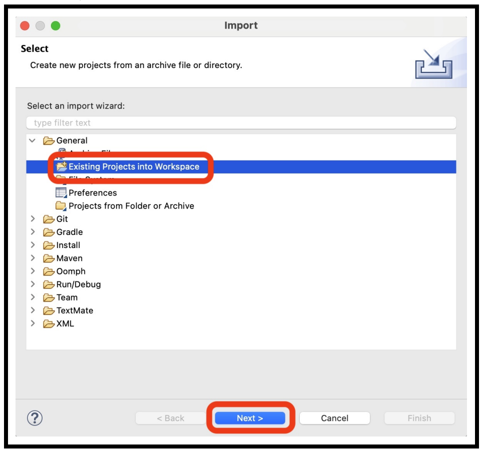
Workspace: /Users/jackie/Desktop/EECS1022-W21-workspace

3. In Eclipse:

3.1 Choose `File`, then `Import`.



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



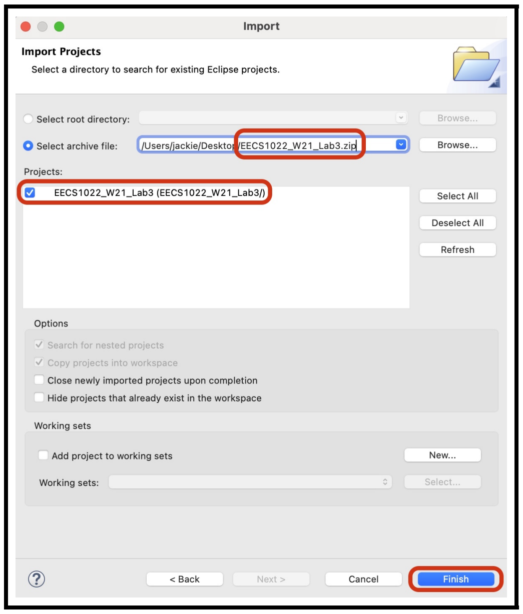
Import

Select

Create new projects from an archive file or directory.

Select an import wizard:

3.3 Choose `Select archive file`. Make sure that the `EECS1022_W21_Lab3` box is checked under `Projects`. Then `Finish`.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory:

Select archive file: /Users/jackie/Desktop/EECS1022_W21_Lab3.zip

Projects:

☑ EECS1022_W21_Lab3 (EECS1022_W21_Lab3/)

Options

☑ Search for nested projects

☑ Copy projects into workspace

☐ Close newly imported projects upon completion

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

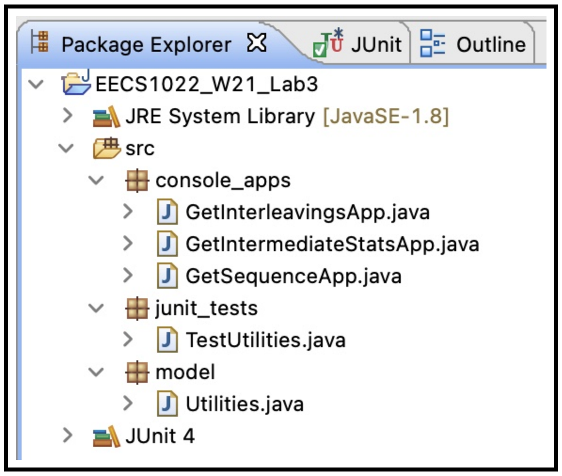
Working sets:

2.2 Step 2: Programming Tasks

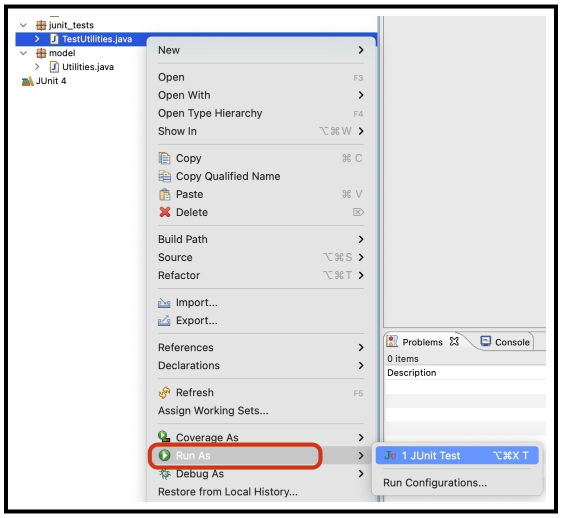
From the `Package Explorer` of Eclipse, your imported project has the following structure.

- 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 <u>**all**</u> 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.

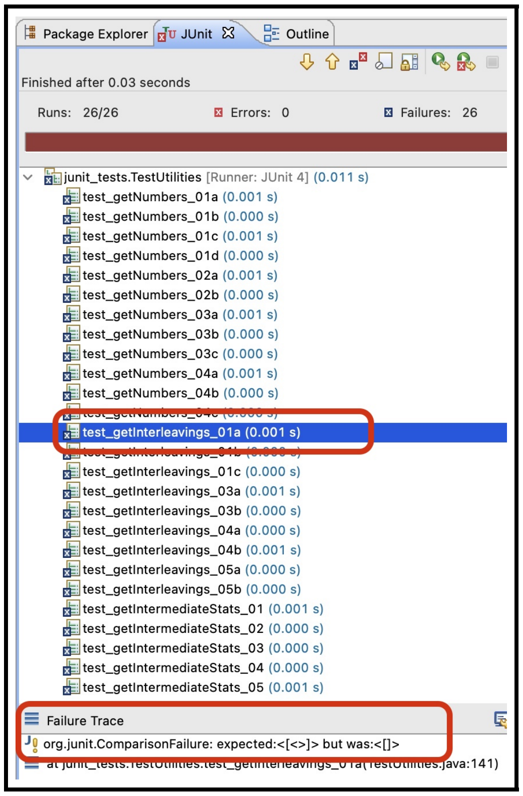
  You must <u>not</u> modify these given JUnit tests.



```
EECS1022_W21_Lab3
  JRE System Library [JavaSE-1.8]
  src
    console_apps
      GetInterleavingsApp.java
      GetIntermediateStatsApp.java
      GetSequenceApp.java
    junit_tests
      TestUtilities.java
    model
      Utilities.java
  JUnit 4
```



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



```
Finished after 0.03 seconds

Runs: 26/26          Errors: 0          Failures: 26

junit_tests.TestUtilities [Runner: JUnit 4] (0.011 s)
  test_getNumbers_01a (0.001 s)
  test_getNumbers_01b (0.000 s)
  test_getNumbers_01c (0.001 s)
  test_getNumbers_01d (0.000 s)
  test_getNumbers_02a (0.001 s)
  test_getNumbers_02b (0.000 s)
  test_getNumbers_03a (0.001 s)
  test_getNumbers_03b (0.000 s)
  test_getNumbers_03c (0.000 s)
  test_getNumbers_04a (0.001 s)
  test_getNumbers_04b (0.000 s)
  test_getNumbers_04c (0.000 s)
  test_getInterleavings_01a (0.001 s)
  test_getInterleavings_01b (0.000 s)
  test_getInterleavings_01c (0.000 s)
  test_getInterleavings_03a (0.001 s)
  test_getInterleavings_03b (0.000 s)
  test_getInterleavings_04a (0.000 s)
  test_getInterleavings_04b (0.001 s)
  test_getInterleavings_05a (0.000 s)
  test_getInterleavings_05b (0.000 s)
  test_getIntermediateStats_01 (0.001 s)
  test_getIntermediateStats_02 (0.000 s)
  test_getIntermediateStats_03 (0.000 s)
  test_getIntermediateStats_04 (0.000 s)
  test_getIntermediateStats_05 (0.001 s)

Failure Trace
org.junit.ComparisonFailure: expected:<[<>]> but was:<[]>
at junit_tests.TestUtilities.test_getInterleavings_01a(TestUtilities.java:141)
```

2.2.1 Method to Implement: `getNumbers`

Problem. You are asked to implement a utility method which takes two integer bounds (*lower* and *upper*) and returns a string consisting of all numbers between the two bounds, inclusively.

Requirement. It is **strictly forbidden** for you to use arrays or any library class (e.g., `ArrayList`). Violating this requirement will cause a **50% penalty** on your lab marks.

Testing. Your goal is to pass **all** tests related to this method in the JUnit test class `TestUtilities`. **These tests document the expected values on various cases: study them while developing your code.** However, use the console application class `GetSequenceApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here are two example runs:

-
  ```
  Enter an integer lower bound:
  88
  Enter an integer upper bound:
  88
  1 number between 88 and 88: <[88]>
  ```

-
  ```
  Enter an integer lower bound:
  23
  Enter an integer upper bound:
  28
  6 numbers between 23 and 28: <{23}, (24), [25], {26}, (27), [28]>
  ```

Todo. Implement the `Utilities.getNumbers` method. See the comments there for the input parameters and requirements. The `String` return value must conform to the expected format:

- There are two possible errors: **1)** when not both bounds are non-negative (≥ 0); and **2)** when the lower bound is not less than or equal to the upper bound.

  What if both error conditions hold simultaneously (e.g., lower `5` and upper `-3`, lower `-3` and upper `-5`)?

  In this case, error condition **1)** takes the priority. That is, error condition **2)** should only be checked when condition **1)** is not the case (i.e., both bounds are non-negative). See the JUnit tests.

- Notice that the second word in the output may be either singular (`number`) or plural (`numbers`, when there are more than one numbers in the sequence).

- Each number in the sequence is wrapped differently: wrapped by round parentheses if the number is a multiple of 3 (e.g., `(24)`); wrapped by square brackets if the number is some multiple of 3 plus one (e.g., `[25]`); and wrapped by a pair of curly braces if the number is some multiple of 3 plus two (e.g., `{26}`).

- All wrapped numbers are separated by commas (`,`). There is one space after each comma.

2.2.2 Method to Implement: `getIntermediateStats`

Problem. You are asked to implement a utility method which takes as inputs the first term (ft), common difference (d), and size (n) of an arithmetic sequence. Based on these three input values, the corresponding arithmetic sequence (of n terms) is:

$$\langle t_1,\ t_2,\ t_3,\ \ldots, t_n \rangle \qquad \text{where } t_i = ft + (i-1) \cdot d \text{ and } 1 \le i \le n$$

The utility method should return a **string value** containing the following equal-sized sequence of statistical items:

$$\langle s_1,\ s_2,\ s_3,\ \ldots,\ s_n \rangle$$

where each statistical item s_i ($1 \le i \le n$) reports the sum and average of the sub-sequence $\langle t_1,\ \ldots,\ t_i \rangle$ (of size i). For example, the statistical item s_3 reports the sum and average of the sub-sequence $\langle t_1,\ t_2,\ t_3 \rangle$ (which, of course, is just a smaller arithmetic sequence).

Requirement. It is **strictly forbidden** for you to use arrays or any library class (e.g., `ArrayList`). Violating this requirement will cause a **50% penalty** on your lab marks.

Testing. Your goal is to pass <u>**all**</u> tests related to this method in the JUnit test class `TestUtilities`. **These tests document the expected values on various cases: study them while developing your code.** However, use the console application class `GetIntermediateStatsApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here is an example run:

```
Enter the first integer term of an arithmetic sequence:
23
Enter the common difference of the sequence:
11
Enter the size of the sequence:
2
{[sum of <23>: 23 ; avg of <23>: 23.00], [sum of <23, 34>: 57 ; avg of <23, 34>: 28.50]}
```

Todo. Implement the `Utilities.getIntermediateStats` method. See the comments there for the input parameters and requirements. The `String` return value must conform to the expected format:

- All statistical terms are wrapped within curly braces (`{}`) and separated by commas (`,`).
- Each statistical term is wrapped within square brackets (`[]`) and reports the sum and average of the corresponding sub-subsequence.
- Each sum is an integer and each average should be formatted as a floating-point number with 2 digits after the decimal point, using `String.format(%.2f, someNumber)`.
- In the above example, the arithmetic sequence implied by the three input values (`23`, `11`, and `2`) is $\langle 23,\ 34 \rangle$, and the output string contains the statistical items for the two sub-sequences: $\langle 23 \rangle$ and $\langle 23,\ 34 \rangle$.
- As a slightly longer example, consider the statistical terms that should be included in the output string by input values 23 (first term), 11 (common difference), and 5 (size):

  ```
  [sum of <23>: 23 ; avg of <23>: 23.00]
  [sum of <23, 34>: 57 ; avg of <23, 34>: 28.50]
  [sum of <23, 34, 45>: 102 ; avg of <23, 34, 45>: 34.00]
  [sum of <23, 34, 45, 56>: 158 ; avg of <23, 34, 45, 56>: 39.50]
  [sum of <23, 34, 45, 56, 67>: 225 ; avg of <23, 34, 45, 56, 67>: 45.00]
  ```

  All five statistical items above should be wrapped within curly braces (`{}`) and separated by commas (`,`).
- There is one space before and after the semicolon (`;`).
- There is one space after each comma (`,`) and colon (`:`).

2.2.3 Method to Implement: `getInterlevaings`

Problem. You are asked to implement a utility method which takes as inputs the first terms ($f1$, $f2$), common differences ($d1$, $d2$), and sizes ($n1$, and $n2$) of two arithmetic sequences. The utility method should return a **string value** containing a new sequence *interleaving* items drawn from the two arithmetic sequences. The interleaving starts from an item drawn from the first sequence, if it is not empty. For example, consider two arithmetic sequences with the same length: $\langle 3,\ 8\rangle$ and $\langle 11,\ 4\rangle$. Their interleaving is then $\langle 3,\ 11,\ 8,\ 4\rangle$, where the 1st and 3rd items are drawn from the first arithmetic sequence, and the 2nd and 4th items are drawn from the second arithmetic sequence. **In general**, your implementation of the utility method should consider when the two arithmetic sequences are of *different* lengths (in which case the last items in the interleaving should be drawn from the longer arithmetic sequence).

Requirement. It is **strictly forbidden** for you to use arrays or any library class (e.g., `ArrayList`). Violating this requirement will cause a **50% penalty** on your lab marks.

Testing. Your goal is to pass **all** tests related to this method in the JUnit test class `TestUtilities`. **These tests document the expected values on various cases: study them while developing your code.** However, use the console application class `GetInterleavingsApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here are two example runs:

-
  ```
  Enter the first integer term of arithmetic sequence 1:
  3
  Enter the common difference of the sequence:
  5
  Enter the size of the sequence:
  2
  Enter the first integer term of arithmetic sequence 2:
  11
  Enter the common difference of the sequence:
  -7
  Enter the size of the sequence:
  2
  <(3), [11], (8), [4]>
  ```

-
  ```
  Enter the first integer term of arithmetic sequence 1:
  3
  Enter the common difference of the sequence:
  5
  Enter the size of the sequence:
  1
  Enter the first integer term of arithmetic sequence 2:
  11
  Enter the common difference of the sequence:
  -7
  Enter the size of the sequence:
  3
  <(3), [11], [4], [-3]>
  ```

Todo. Implement the `Utilities.getInterlevaings` method. See the comments there for the input parameters and requirements. The `String` return value must conform to the expected format:

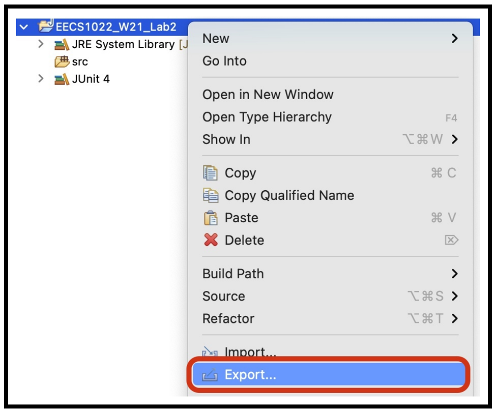
- All interleaved items should be wrapped within angle brackets (`<>`) and separated by commas (`,`). There is one space after each comma.
- Items drawing from the first arithmetic sequence should be wrapped within round parentheses (e.g., `(3)`).
- Items drawing from the second arithmetic sequence should be wrapped within square brackets (e.g., `[11]`).

2.3 Step 3: Exporting the Completed Project

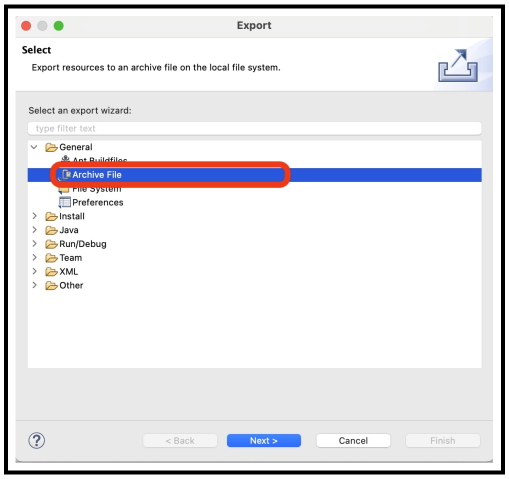
You are required to submit a Java project archive file (.zip) consisting all subfolders.

In Eclipse:

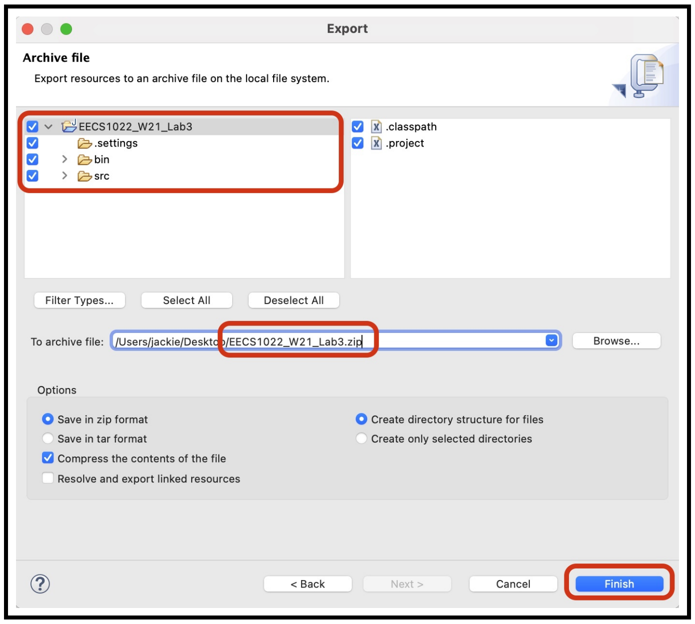
1. Right click on project `EECS1022_W21_Lab3`. Then click `Export`



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



3. Check the top-level `EECS1022_W21_Lab3`
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_Lab3.zip` (case-sensitive)
Then `Finish`.



- [✓] EECS1022_W21_Lab3
  - [✓] .settings
  - [✓] bin
  - [✓] src
- [✓] .classpath
- [✓] .project

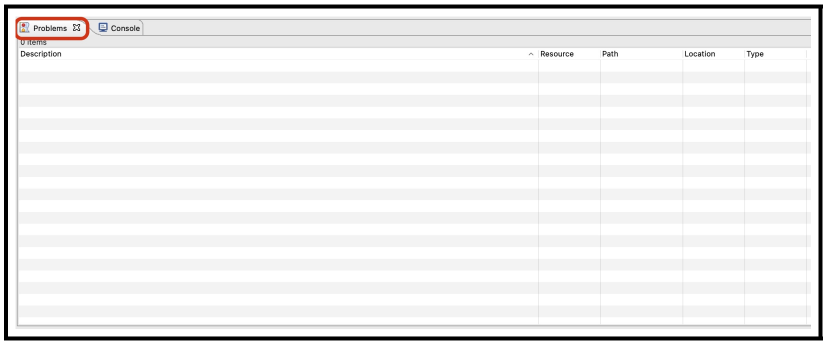
To archive file: /Users/jackie/Desktop/EECS1022_W21_Lab3.zip

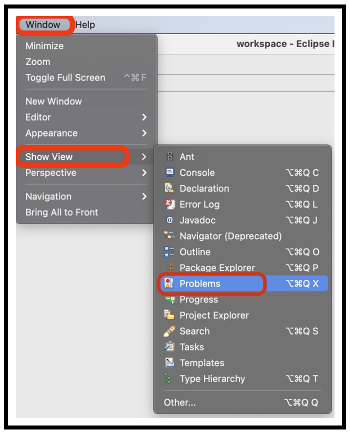
Options:
- (●) Save in zip format
- ( ) Save in tar format
- [✓] Compress the contents of the file
- [ ] Resolve and export linked resources
- (●) Create directory structure for files
- ( ) Create only selected directories

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 **<u>are</u>** acceptable). In case you do not see the `Problems` panel: click on `Window`, then `Show View`, then `Problems`.



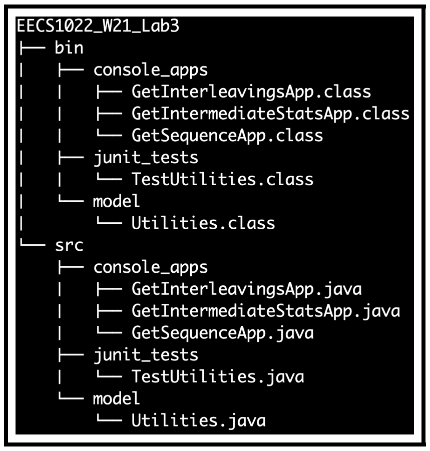


Submitting programs with errors (meaning that it cannot be run for grading) will result in possible partial, but low, marks.

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

`EECS1022_W21_Lab3.zip`

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



```
EECS1022_W21_Lab3
├── bin
|   ├── console_apps
|   |   ├── GetInterleavingsApp.class
|   |   ├── GetIntermediateStatsApp.class
|   |   └── GetSequenceApp.class
|   ├── junit_tests
|   |   └── TestUtilities.class
|   └── model
|       └── Utilities.class
└── src
    ├── console_apps
    |   ├── GetInterleavingsApp.java
    |   ├── GetIntermediateStatsApp.java
    |   └── GetSequenceApp.java
    ├── junit_tests
    |   └── TestUtilities.java
    └── model
        └── Utilities.java
```

Figure 1: Lab3 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 `Lab3` to submit the Java archive file: `EECS1022_W21_Lab3.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.

-