EECS1022 (M,N,O) Winter 2021
Lab2
(Nested) Selections/Conditions

Chen-Wei Wang

Release Date: Friday, January 22
Due Date: 23:59 EST, Friday, January 29

- 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.
- 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
   - 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

- 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, only the tests supplied to you). That is, your lab submission will **not** be graded manually using the console application given to you.
- Do **not** modify the JUnit test class `TestUtilities.java` given to you.
- 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.
- To complete this lab, it is not necessary to use loops or methods from library classes (e.g., `Math`).
- 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 3'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 Lab2, you are assigned to study **Week 3 Part A to Part G** of the Java tutorial series:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_6UHGyyTZTNorPaBgcXpauf [Week 3 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.

     Parts A to G are directly relevant to this lab. Parts **F and G** from Week 2 are also relevant to your Lab2.

  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

- This week's Java tutorial shows some example of nested if-statement. Optionally, if you wish to study more about nested if-statement, refer to these two videos:

  – A simple bank application: Nested If-Statements Version 1:

    https://www.youtube.com/watch?v=w3YojTfDTeA&list=PL5dxAmCmjv_5NRNPG3OiWZWAqmvCjiLfG&index=19

  – A simple bank application: Nested If-Statements Version 2:

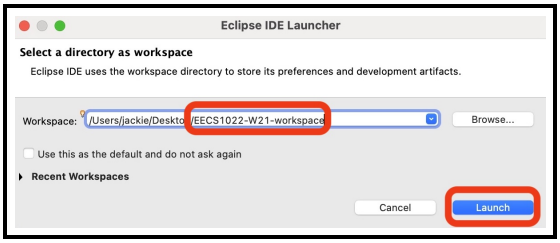
    https://www.youtube.com/watch?v=FI0d8XKBymg&list=PL5dxAmCmjv_5NRNPG3OiWZWAqmvCjiLfG&index=20

2 Task 2: Complete Programming Exercises

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

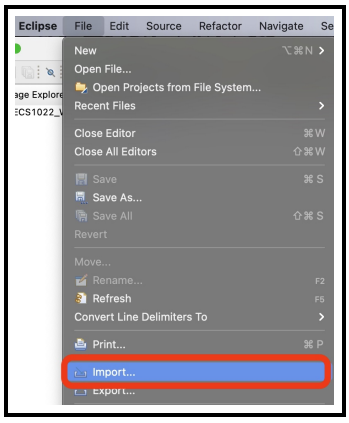
2.1 Step 1: Download and Import the Starter Project

1. Download the Eclipse Java project archive file from eClass: `EECS1022_W21_Lab2.zip`
2. Launch Eclipse and browse to `EECS1022-W21-workspace` as the `Workspace` then click on `Launch`, e.g.,

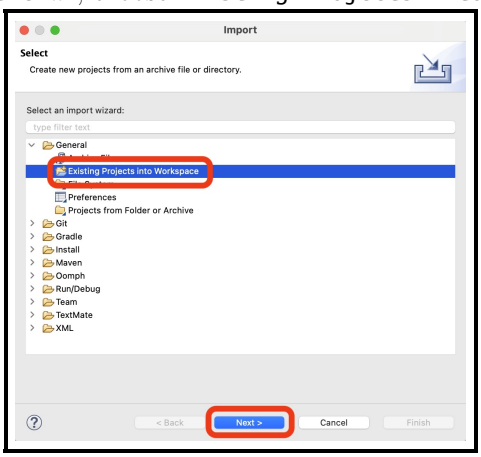


3. In Eclipse:

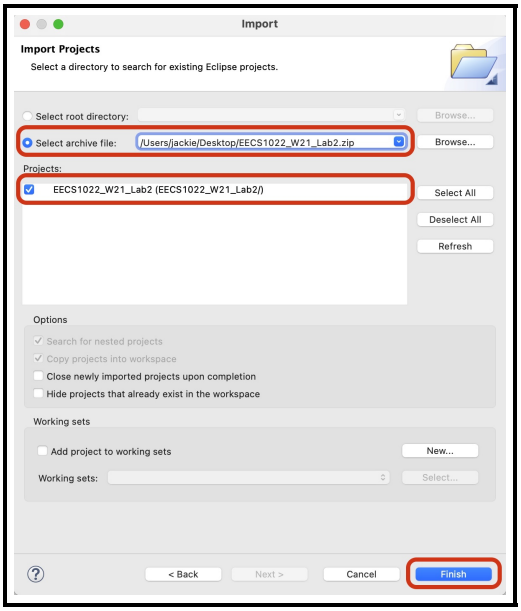
3.1 Choose `File`, then `Import`.



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



3.3 Choose `Select archive file`. Make sure that the `EECS1022_W21_Lab2` box is checked under `Projects`. Then `Finish`.

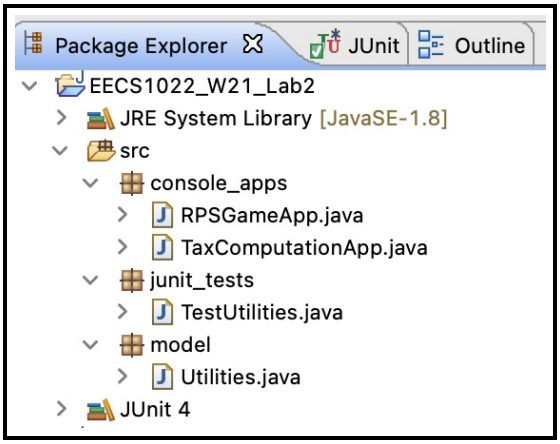


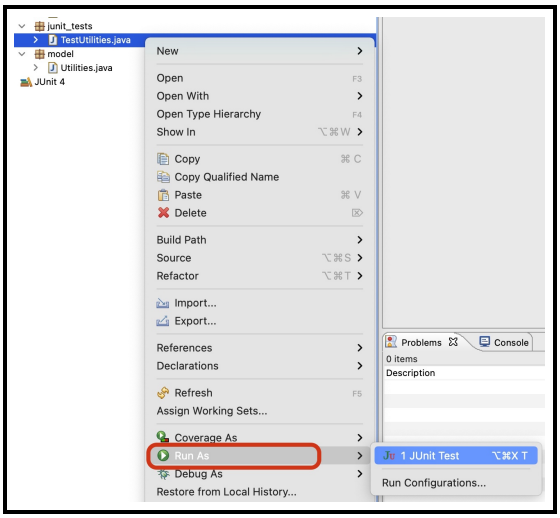
2.2 Step 2: Programming Tasks

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

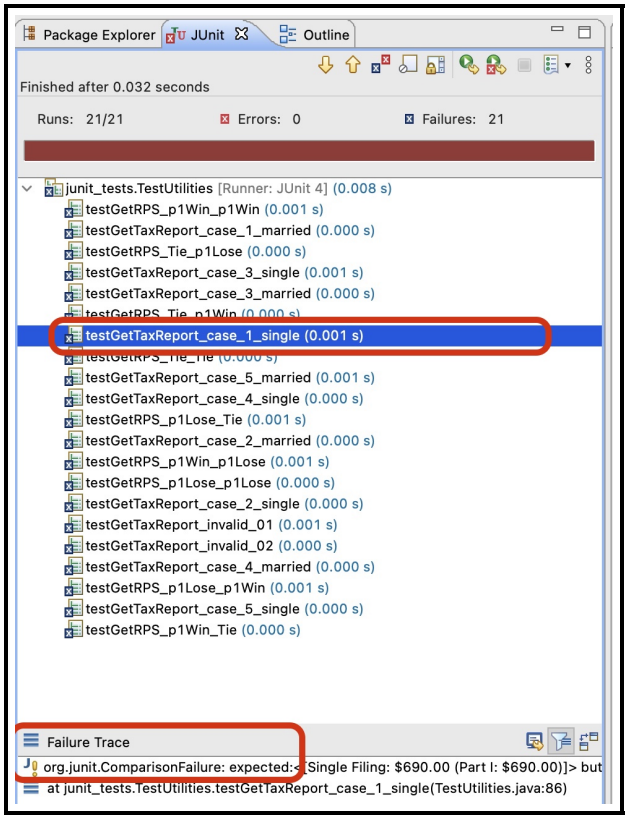
- You can manually test the assigned methods using the corresponding console application classes in package `console_apps`. These classes are completed and given to you. See below for more descriptions.
- Your goal is to pass **all** JUnit tests given to you (i.e., a **green bar**). To run them, as shown 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: `getRPSGameReport`

Problem. You are asked to consider the rock-paper-scissors (RPS) game (for which it is assumed that you know how to play it). Rock is denoted by character `'R'`, paper by `'P'`, and scissors by `'S'`. As a reminder: rock wins scissors, paper wins rock, and scissors wins paper.

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 `RPSGameApp` if you wish (e.g., use the input and expected values from the JUnit tests).

Let's consider two example runs (where the same two players swap the order from one game to another):

- ```
  Enter the name of player 1:
  Doraemon
  Enter the name of player 2:
  Edward Scissorhands
  What does Doraemon play at round 1 (R, P, S)?
  R
  What does Edward Scissorhands play at round 1 (R, P, S)?
  S
  What does Doraemon play at round 2 (R, P, S)?
  R
  What does Edward Scissorhands play at round 2 (R, P, S)?
  S
  Game over: Doraemon wins! [Round 1: Doraemon wins (R vs. S) ; Round 2: Doraemon wins (R vs. S)]
  ```

- ```
  Enter the name of player 1:
  Edward Scissorhands
  Enter the name of player 2:
  Doraemon
  What does Edward Scissorhands play at round 1 (R, P, S)?
  S
  What does Doraemon play at round 1 (R, P, S)?
  R
  What does Edward Scissorhands play at round 2 (R, P, S)?
  S
  What does Doraemon play at round 2 (R, P, S)?
  R
  Game over: Doraemon wins! [Round 1: Doraemon wins (R vs. S) ; Round 2: Doraemon wins (R vs. S)]
  ```

  In this 2nd example run:

  - The Round 1 result shows: `Round 1: Doraemon wins (R vs. S)`.
  - More precisely, `R vs. S` shows that rock wins scissors, despite the order of the players.
  - That is, **the left-hand side of `vs.` always shows the play that wins, not what player 1 (in this case Edward Scissorhands) plays.**
  - A similar argument also applies to the Round 2 result.

Todo. Implement the `Utilities.getRPSGameReport` method. See the comments there for the input parameters and requirements. The `String` return value must conform to the expected format (otherwise your code will fail some JUnit tests). Here are some hints for you:

- Read carefully the comments in the `Utilitye` class about what each input of method `getRPSGameReport` means. For example, invoking

  ```
  Utilities.getRPSGameReport("Suyeon", "Yuna", 'R', 'P', 'S', 'R')
  ```

means: Suyeon is player 1, Yuna is player 2, player 1 plays rock and player 2 plays paper in Round 1, and player 1 plays scissors and player 2 plays rock in Round 2.

- Do **not** modify the console application class `RPSGameApp` given to you: the string return value of the utility method `getRPSGameReport` must be a string like:

  ```
  Game over: Doraemon wins! [Round 1: Doraemon wins (R vs. S) ; Round 2: Doraemon wins (R vs. S)]
  ```

- As illustrated in the above 2nd example run, **the left-hand side of `vs.` always shows the play that wins, not what player 1 plays.**

- Be mindful about the spellings (e.g., `Game over`, `wins`): they are case-sensitive.

- Be mindful about the **single** spaces between words and after symbols such as `:`, `!`, `.`, and `:`.

- After the phrase **`''Game over:''`** comes the game result: it only shows who wins.

- Within the pair of square brackets `[]` comes the results of each round: it **only** declares who wins.

  Hint. Study the 9 test cases in `TestUtilites`: they are arranged in a systematic (yet not complete) way.

  Question. In order to ***exhaustively*** test this game, considering just how two players say `Suyeon` and `Yuna` may play in the two rounds, how many tests do we need? (Optionally, you may write extra test yourself as an exercise.)

2.2.2 Method to Implement: `getTaxReport`

After reading the problem description below, here is an illustration on the two example filers Jim and Jonathan: https://www.youtube.com/watch?v=q2NT5x77hdg&list=PL5dxAmCmjv_7YgI2LgcwjWTHiNZSR-aQX&index=1.

Problem. You are asked to consider an imaginary income tax scheme, where the calculation is based on:

1. filing status (single filing or married filing)
2. taxable income

Once the above information is given by the user, the tax is calculated based on the following scheme:

Tax Rate	Single Filing Status	Married Filing Status
10%	$0 – $8,350	$0 – $16,700
15%	$8,351 – $33,950	$16,701 – $67,900
25%	$33,950+	$67,900+

That is, a tax payer's income is split into **up to** three parts, depending on how high their income is. Once split, each part of the income is taxed with the corresponding rate. For a single filer, the cutoff points are $8,350 and $33,950. For a married filer, the cutoff points are $16,700 and $67,900.

As an example, consider Jim who is a single filer and whose income is $186,476. Since his income is higher than the second cutoff point for a single filer (i.e., $33,950), his income will be split into 3 parts:

- The first $8,350 is taxed with a rate of 10%.
- The subsequent $(33,950 - 8,350) is taxed with a rate of 15%.
- The final $(186,476 - 33,950) is taxed with a rate of 25%.

Important Exercise Before You Proceed: Try the tax calculation for Jonathan, who is a married filer and who has a lower income $33,500. How would his income be split with this lower income? How would his income tax then be calculated?

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 `TaxComputationApp` if you wish (e.g., use the input and expected values from the JUnit tests). Here are two example runs:

1.
```
Enter your name:
Alan
Alan, enter your filing status (1 -- Single Filing; 2 -- Married Filing):
1
Alan, enter your taxable income (an integer value):
186476
Single Filing: $42806.50 (Part I: $835.00, Part II: $3840.00, Part III: $38131.50)
```

2.
```
Enter your name:
Mark
Mark, enter your filing status (1 -- Single Filing; 2 -- Married Filing):
2
Mark, enter your taxable income (an integer value):
33500
Married Filing: $4190.00 (Part I: $1670.00, Part II: $2520.00)
```

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

- Do **not** modify the console application class `TaxComputationApp` given to you: the string return value of the utility method `getTaxReport` must be a string like:

  ```
  Married Filing:  $4190.00 (Part I: $1670.00, Part II: $2520.00)
  ```

- If an income is split into less than 3 parts, the missing part(s) will not be included in the string report.
- You should first calculate the part(s) of tax of a given income, then calculate its sum, and then you must format each part and the sum as floating-point numbers with exactly two digits following the decimal point (e.g., `800.00`, `789.10`).

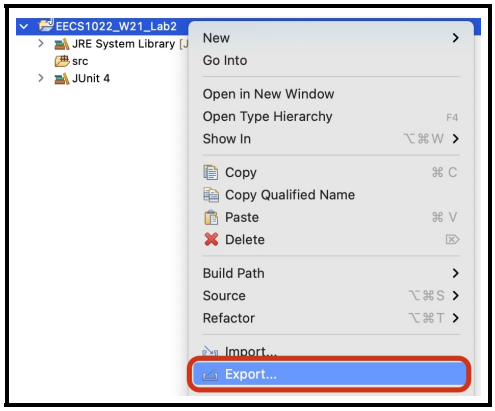
  Hint. Use `String.format(%.2f, someNumber)` as shown in Week 1's Java tutorials for this purpose.

2.3 Step 3: Exporting the Completed Project

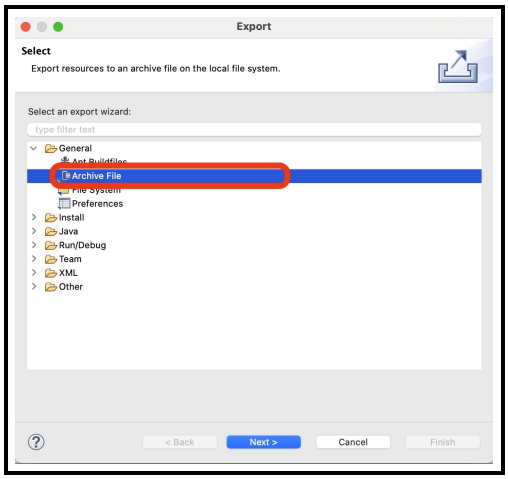
You are required to submit a Java project archive file (.zip) consisting all subfolders.

In Eclipse:

1. Right click on project `EECS1022_W21_Lab2`. Then click `Export`



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

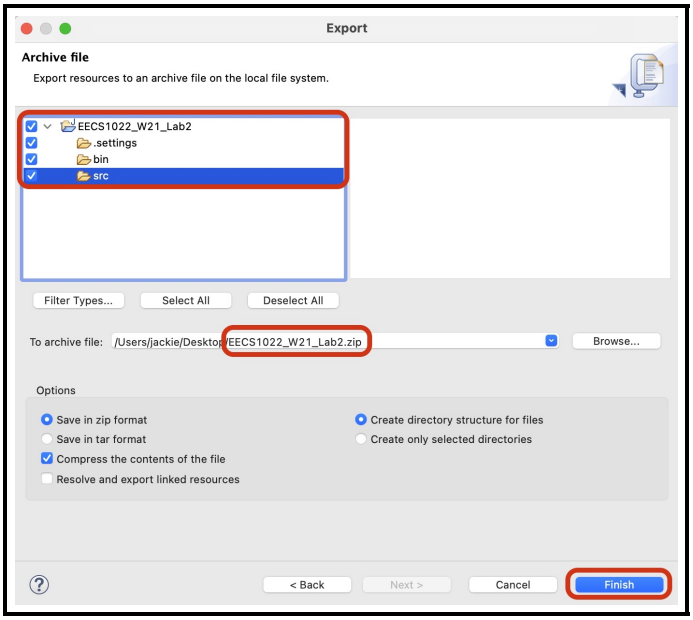


3. Check the top-level `EECS1022_W21_Lab2`

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_Lab2.zip` (**case-sensitive**)

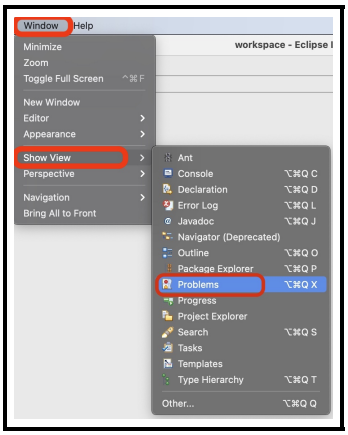
Then `Finish`.



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_Lab2.zip`

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

```
EECS1022_W21_Lab2
├── bin
|   ├── console_apps
|   |   ├── RPSGameApp.class
|   |   └── TaxComputationApp.class
|   ├── junit_tests
|   |   └── TestUtilities.class
|   └── model
|       └── Utilities.class
└── src
    ├── console_apps
    |   ├── RPSGameApp.java
    |   └── TaxComputationApp.java
    ├── junit_tests
    |   └── TestUtilities.java
    └── model
        └── Utilities.java
```

Figure 1: Lab2 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 `Lab2` to submit the Java archive file: `EECS1022_W21_Lab2.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.

-