EECS2030 (B & E) Fall 2021
Lab4
Programming with Inheritance

Chen-Wei Wang

Release Date: Monday, November 8
Due Date: 14:00 EST, Monday, November 22

- 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`). Instead, use **primitive arrays** (e.g., `String[]`) to implement collections/lists for the classes and methods. 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.
  - 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: late submissions will **not** be accepted.

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 composed of:
     - Numerical Literals and operators
     - String Literals and operators
     - Variables and assignments
     - (Nested) Selections/Conditionals/If-Statements
     - OOP Basics: Classes, Attributes, Constructors, Accessor and Mutator Methods, Method Invocations, Context Objects, Dot Notation
     - Declaring and manipulating (single-valued vs. multi-valued) reference-typed attributes
     - Programming with Inheritance (`extends`, `super`, polymorphism, dynamic binding)
     - Inferring Java Classes from JUnit Tests
   - Use the given JUnit tests 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.

Assumptions

- You have already setup a Github account and stored work in a **private** repository: e.g., `EECS2030-F21-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 this lab 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

- To complete this lab, it is **strictly forbidden** for you to use any library class. Violating this requirement will cause a **50% penalty** on your lab marks.
  - Here are some examples of ***forbidden*** classes/methods: `Arrays` class (e.g., `Arrays.copyOf`), `System` class (e.g., `System.arrayCopy`), `ArrayList` class, `String` class (e.g., `substring`).
  - The use of some library classes does not require an `import` statement, but these classes are ***also forbidden*** to be used.
  - Here are the exceptions (library methods which you ***are allowed*** to use if needed):
    - `String` class (`equals`, `format`)
- 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 2.5.
- 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.
- For the JUnit test class `StarterTests.java` given to you:
  - Do **not** modify the test methods given to you.
  - You are allowed to add new test methods.
- Derived from the given JUnit test methods:
  - Each class you introduce and implement must be placed under the `model` package.
  - Do **not** add any class that is not required by the starter tests.
  - All attributes you declare must be **private** or **protected**. Use public accessors/mutators if needed.
  - Besides the required methods required by the starter tests, you are **free** to implement additional public/private/protected helper methods as you see fit.
  - For each method you implement:
    - No `System.out.println` statements should appear in it.
    - No Scanner operations (e.g., `input.nextInt()`) should appear in it.
      Instead, declare input parameters of the method as indicated by the JUnit tests.
- You are welcome to ask questions related to this lab 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).
    - The Review Tutorial (Part 1 and Part 2) addresses how to use **debugger in Eclipse**. You are advised 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 the Background Studies

1. This lab requires knowledge and skills covered in exceptions and JUnit lectures:
   - Lecture 5 (Weeks 7, 8, 9): Inheritance [Pdf]
2. This lab still covers topics discussed in the Review Tutorial Series (Part 1 and Part 2):

   https://www.eecs.yorku.ca/~jackie/teaching/tutorials/index.html#refurbished_store

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

   https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/Building%20an%20Apple%20Refurbished%20Store%20App%20in%20Java.pdf
3. It is also required that you review these two written notes supplementing the above Review Tutorial Series:
   - How to manipulate objects with reference-typed, multi-valued attributes:

     https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/F/EECS2030/notes/EECS2030_F21_Tracing_PointCollectorTester.pdf
   - See this notes on how to infer classes and methods from given JUnit tests:

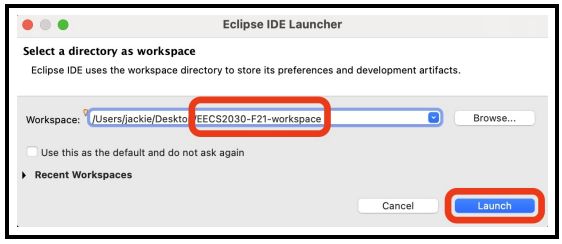
     https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/F/EECS2030/notes/EECS2030_F21_Inferring_Classes_from_JUnit.pdf
   - You can find here the example covered in the notes for practice:
     - Starter: https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/F/EECS2030/notes/EECS2030_F21_Inferring_Classes_from_JUnit.zip
     - Solution: https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/F/EECS2030/notes/EECS2030_F21_Inferring_Classes_from_JUnit_Solution.zip

2 Task 2: Complete Programming Exercises

Starting Task 2 should mean that you have ***already completed*** the background studies as outline in Section 1.

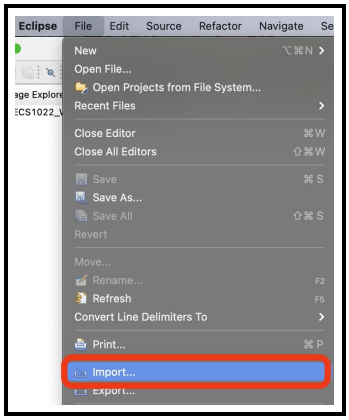
2.1 Step 1: Download and Import the Starter Project

1. Download the Eclipse Java project archive file from eClass: `EECS2030_F21_Lab4.zip`
2. Launch Eclipse and browse to, e.g., `EECS2030-F21-workspace`, as the `Workspace` then click on `Launch`, e.g.,

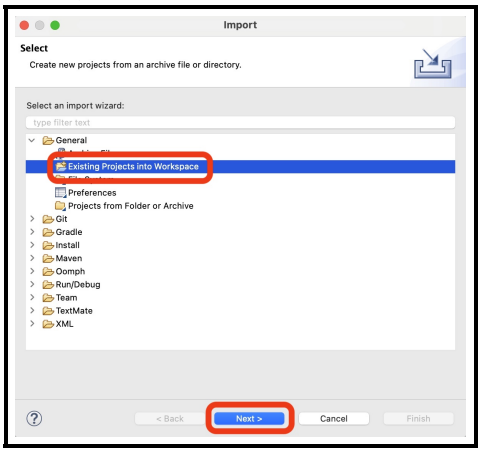


3. In Eclipse:

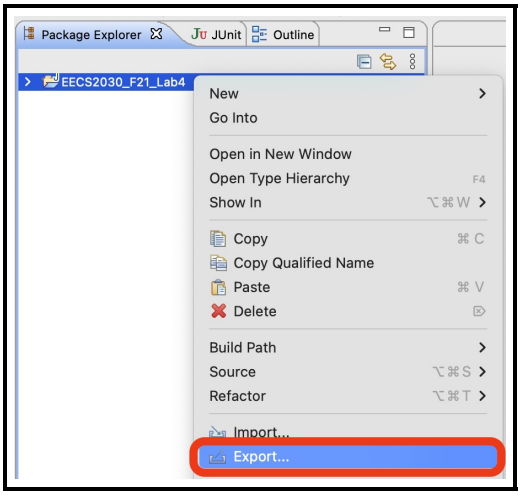
3.1 Choose `File`, then `Import`.



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



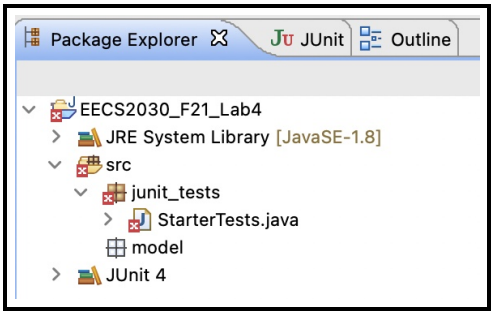
3.3 Choose `Select archive file`. Make sure that the `EECS2030_F21_Lab4` box is checked under `Projects`. Then `Finish`.



2.2 Step 2: Programming Tasks

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

- The `console_apps` package is empty. You may add new console application classes here to test the implemented methods if you wish. However, these added console application classes will **not** be graded.
- It is **expected** that the `StarterTests` JUnit class contains **compilation errors**. This is because that declarations and definitions of the required class(es) and method(s) it references are missing.



- The `model` package is empty. Class(es) and method(s) derived from the given JUnit class **must** be added to this package. Class(es) added to a package other than `model` will **not** be graded.

  Therefore, your tasks are:

  1. Inferring from the given JUnit tests, add the missing class(es) and method(s) into the `model` package. For example, if you add class `Foo` in the model package, make sure that you write a line in the beginning of the `StarterTests` class (after the line `package junit_tests;`):

     ```
     import model.Foo;
     ```

  2. Pass **all** JUnit tests given to you (i.e., a **green bar**).

     To run them, as shown in the Review Tutorial Series, right click on `StarterTests.java` and run it as JUnit tests. Of course, none of the given tests would pass to begin with.

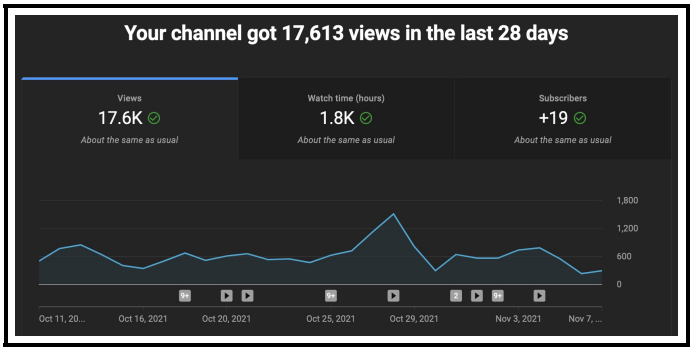
  You must not modify these given JUnit tests, as they suggest how the intended class(es) and method(s) should be declared.

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 implemented method. Furthermore, when needed, you should a **breakpoint** at the line of the failing assertion, then launch the **debugger** to pinpoint where the error came from.

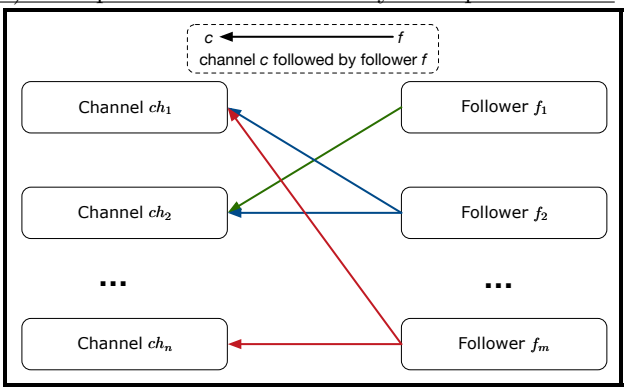
2.3 The Video Channel Platform Problem

You are required to develop an object-oriented program solving aspects of a video channel platform (e.g., YouTube), where each video channel has a number of followers, each of which being either one subscribed to the channel and watching videos as new ones are released, or one monitoring the channel's statistis (e.g., average watch time).

(a) Example Statistics of a Video Channel



(b) Multiple Channels Followed by Multiple Followers



Here are the relevant entities involved in this problem:

- Each video *channel* is characterized by its name (e.g., `Cafe Music BGM channel`), its list of followers, and its list of released videos. New followers may be added, and existing followers may be removed. When a channel is first created, it is specified with both the **maximum** number of followers allowed and the **maximum** number of released videos allowed. Periodically, a channel may update itself by releasing a video (e.g., `MONDAY MORNING JAZZ: Sweet November Jazz Music`).

  The above Figure (a) shows some example statistics of a video channel (e.g., views, watch time). Figure (b) shows that a channel may be linked to multiple followers (e.g., channel ch_2 followed by f_1 and f_2), whereas each follower may be linked to multiple channels (e.g., f_2 follows ch_1 and ch_2).

- Each video channel *follower* is characterized by its name and can be linked to a list of channels. Each follower should be kept in synch with each of the channel it follows, whenever a channel update occurs. Specifically, there are two kinds of channel followers:

  - A channel *monitor* follows a list of channels. When a monitor is first created, it is specified with the **maximum** number of channels allowed to follow.

    For each of the channel it follows, the channel maintains a list of statistical data (updated as soon as a subscriber of the same channel watched one of its released videos): number of views, maximum watch time, and average watch time.

  - A channel *subscriber* follows a list of channels and is recommended videos by these channels (whenever there is a new release). When a subscriber is first created, it is specified with both the **maximum** number of channels allowed to follow and the **maximum** number of recommended videos allowed.

    A subscriber may watch each of the recommended videos for as many times as they like: each time after they watch a video of a channel, all monitors following that channel will be updated for their statistics (e.g., number of views, average watch time) accordingly.

  Notes:

  - **Other intended functionalities of the above kinds of entities/objects can be inferred from the: 1) given JUnit test method; and 2) comments in class `StarterTests`.**
  - **Error handling (via exceptions) is only required if it is either explicitly mentioned in the above problem description or test comments, or indicated by the JUnit tests.**
  - For each of the above-mentioned **maximum capacities**, **no** error handling is needed when the limit is exceeded. For example, where the maximum capacity is 200, given that the private (or protected), primitive array is created with the corresponding size, attempting to add an 201st object is expected to result in an `ArrayIndexOutOfBoundsException`.

2.4 Hints and Requirements

- See this notes on how to declare and manipulate reference-typed, multi-valued attributes:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/F/EECS2030/notes/EECS2030_F21_Tracing_PointCollectorTester.pdf

- See this notes on how to infer classes and methods from given JUnit tests:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/F/EECS2030/notes/EECS2030_F21_Inferring_Classes_from_JUnit.pdf

  Programming IDEs such as Eclipse are able to fix such compilation errors for you. **However, you are advised to follow the guidance as specified in the notes to fix these compilation errors manually, because: 1) it helps you better understand how the intended classes and methods work together; and 2) you may be tested in a written test or exam without the assistance of IDEs.**

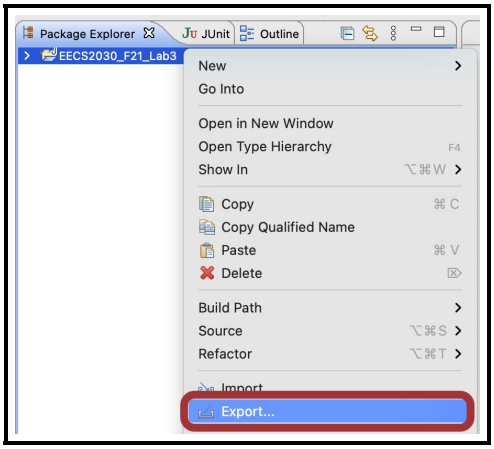
- Any new class(es) you add must reside in the `model` package.
  - All attributes you declare must be **private**. Use public accessors/mutators to retrieve/change their values from other classes.
  - Once the necessary class(es) and method(s) are declared, you can add as many attributes as necessary to implement the body of each method.
  - Study carefully example manipulations of the relevant objects as specified in `StarterTests.java`: they suggest the how the intended class(es) and method(s) should be declared and implemented.
  - Focus on *gradually* passing one test at a time.
  - You **cannot** use any Java library classes (e.g., `ArrayList`) or methods for implementation. That is, there must **not** be any `import` statement in the class(es) you add to the `model` package.

2.5 Step 3: Exporting the Completed Project

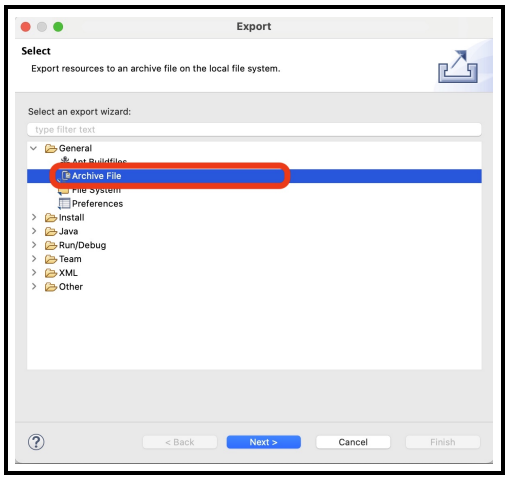
You are required to submit a Java project archive file (.zip) consisting all subfolders.

In Eclipse:

1. Right click on project `EECS2030_F21_Lab4`. Then click `Export`



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

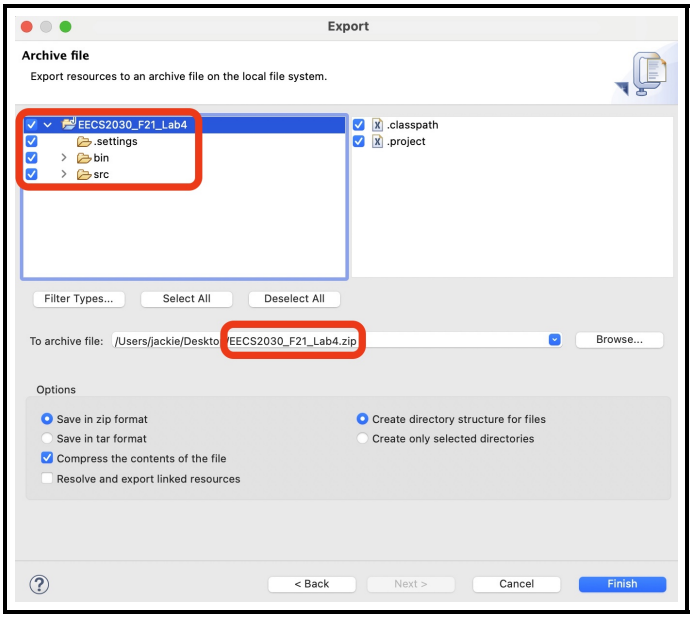


3. Check the top-level `EECS2030_F21_Lab4`

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

Under `To archive file:` browse to, e.g., desktop, and save it as `EECS2030_F21_Lab4.zip` (**case-sensitive**)

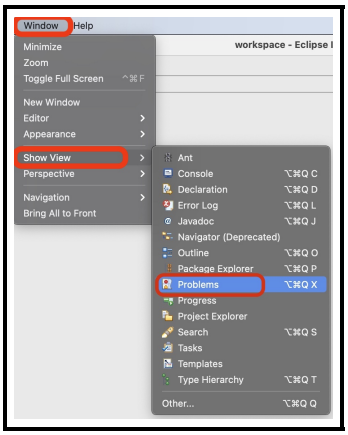
Then `Finish`.



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.

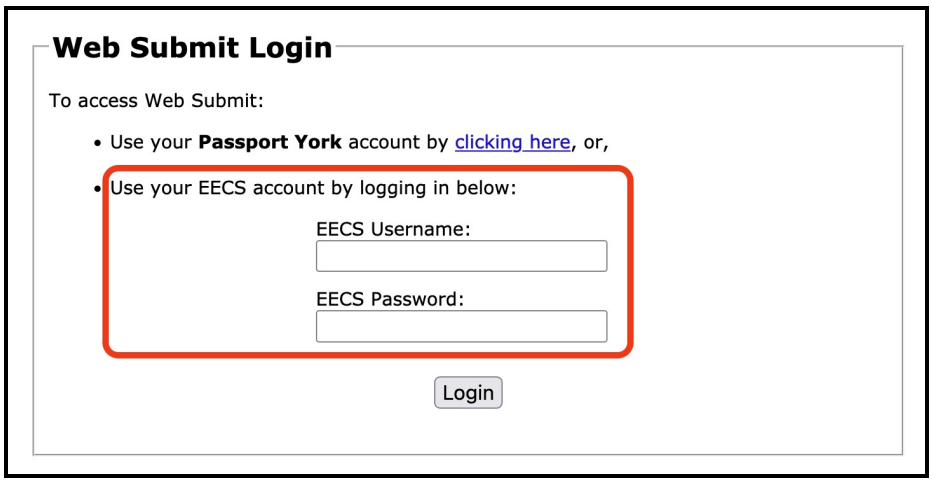
2. Section 2.5 asks you to **export** the Java project as an archive file:

`EECS2030_F21_Lab4.zip`

Click on the following link (for which you will be prompted to enter your EECS account login credentials):

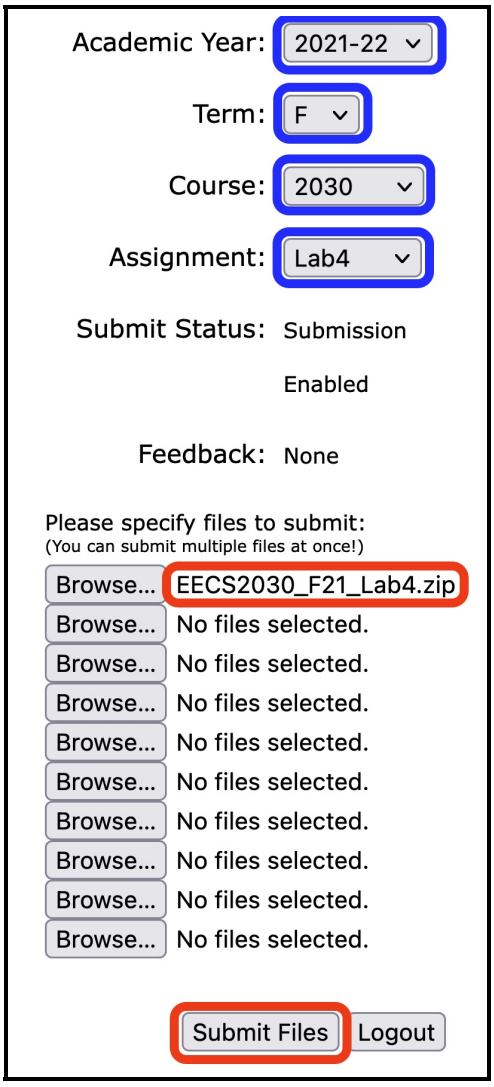
https://webapp.eecs.yorku.ca/submit/?acadyear=2021-22&term=F&course=2030&assignment=Lab4

- You **must** login into the web submit page using your EECS login credentials (otherwise, your submitted folder on the EECS server may not be identified properly):

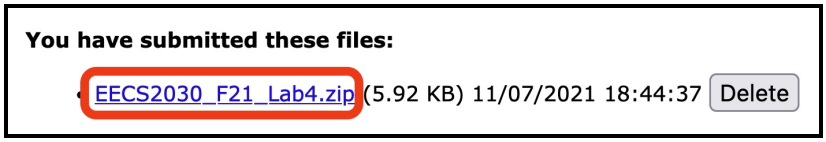


Note. If you are prompted for your PPY login instead, then it might be due to an earlier login session. In this case, login first with your PPY account credentials, then log out. Then, clicking on the above submission link should lead you to the login page for EECS account credentials.

- Ensure that the correct academic year, term, course, and assignment are chosen. Then, browse to the archive file `EECS2030_F21_Lab4.zip` and click on `Submit Files`.



- You may upload as many draft versions as you like before the deadline – only the **latest** submitted version of your work before the deadline will be graded.
- It is your **sole responsibility** to **download and ensure** that:
  - The submitted zip file is the one you intend to be graded (e.g,. non-empty, not the starter project).



4 Amendments

Clarifications or corrections will be added to this section.

- Nov 9:
  - Attributes declared should be either `private` or `protected`.