EECS1022 (M,N,O) Winter 2021
Lab 1
Elementary Programming (with no conditionals or loops)

Chen-Wei Wang

Release Date: Friday, January 15
Due Date: 23:59 EST, Friday, January 22

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

- Do not modify the JUnit test class 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 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 Lab1, you are assigned to study **Week 2 Part A to Part G** of the Java tutorial series:

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_6wy2m0yq2wObIWPz4tAxW6`

  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 E are directly relevant to this lab. Parts F and G are 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:

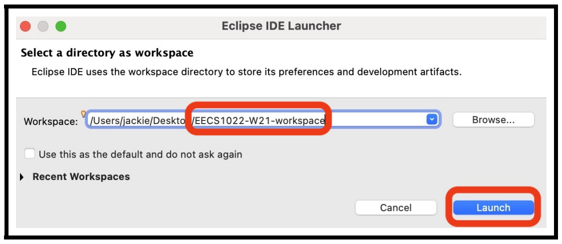
  `https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/EECS1022%20Tutorial%20on%20Java.pdf`

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_Lab1.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

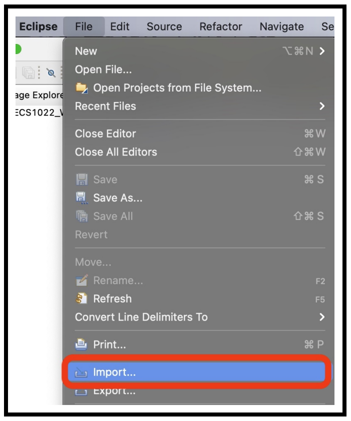
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

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

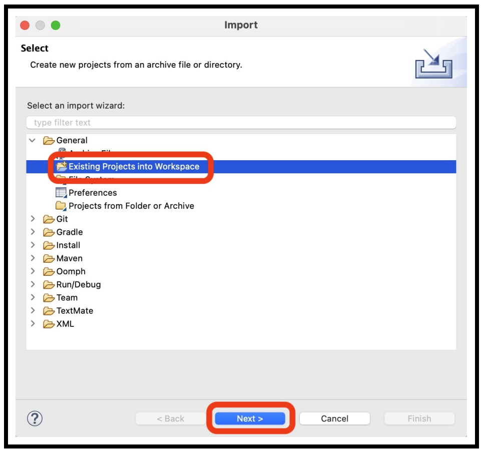
Use this as the default and do not ask again

3. In Eclipse:

3.1 Choose `File`, then `Import`.



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



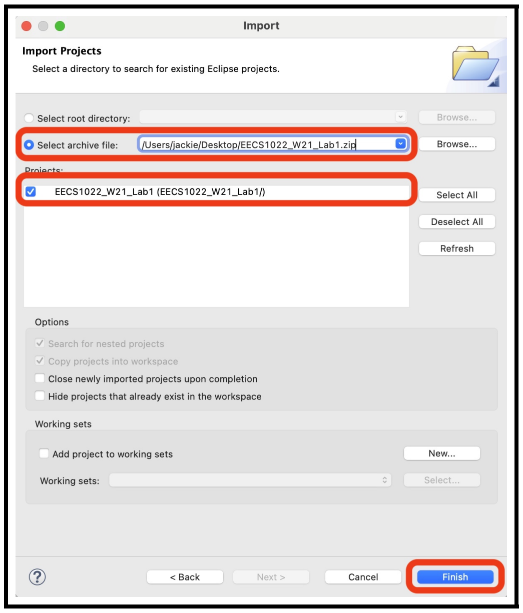
Import

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_Lab1` 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_Lab1.zip

Projects:

☑ EECS1022_W21_Lab1 (EECS1022_W21_Lab1/)

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

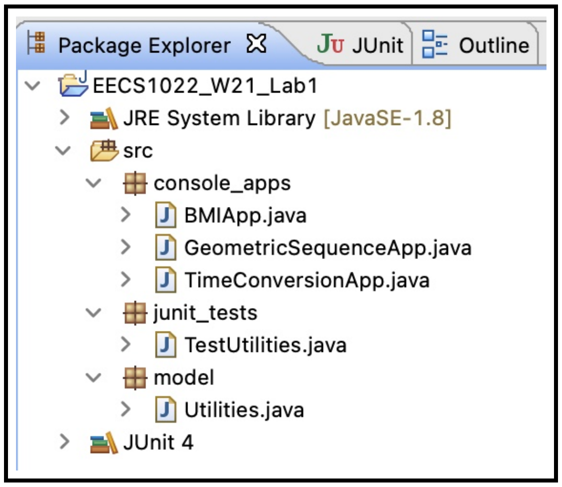
Working sets:

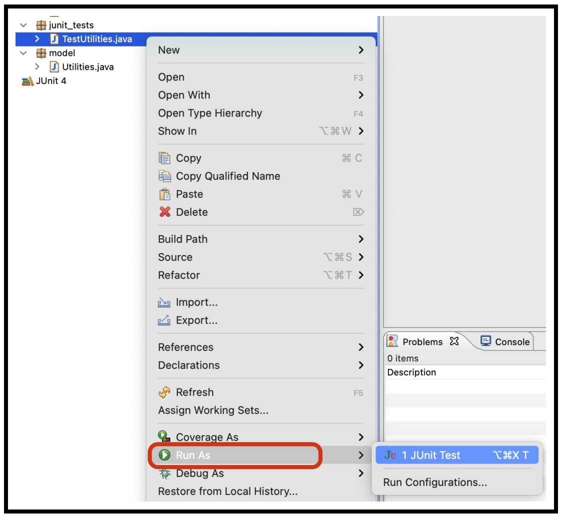
2.2 Step 2: Programming Tasks

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 **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.





2.2.1 Method to Implement: `getGeometricSequence`

Problem. You are asked to consider a *geometric* sequence (of integer values) of length 5.

$$\langle a_1,\ a_2,\ a_3,\ a_4,\ a_5 \rangle$$

where a_1, the start of the sequence, is called the *first term*. In a geometric sequence, there is a common ratio r between every two adjacent terms:

$$\frac{a_i}{a_{i-1}} = d \qquad\qquad 2 \leq i \leq 5$$

For example, $\langle 6,\ 12,\ 24,\ 48,\ 96 \rangle$ is a geometric sequence with 6 being the first term, 2 being the common ratio, and 5 being the length.

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

```
Enter the first term of a geometric sequence (of size 5):
3
Enter the common ratio of the geometric sequence:
2
[3][6][12][24][48] has average 18.6
```

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

2.2.2 Method to Implement: `getBMI`

Problem. You are asked to compute the Body Mass Index (BMI), which is a measure of health based on height and weight specified in metric units (i.e., meters and kilograms, respectively). Given a user weighting k kilograms and who is m meter tall, their BMI is calculated as: $BMI = \frac{k}{m^2}$ (i.e., weight divided by the square of height). Once the BMI is calculated, one may consult with the following table to interpret the result:

BMI Range	Interpretation
$BMI < 18.5$	Underweight
$18.5 \leq BMI < 25.0$	Normal
$25.0 \leq BMI < 30.0$	Overweight
$30.0 \leq BMI$	Obese

For this exercise, you are only asked to compute the BMI without interpreting it as specified above. **Users of your program, however, will specify a weight in pounds and a height in inches**, meaning that you need to consider the following conversions:

- 1 pound is equivalent to 0.4536 kilograms
- 1 inch is equivalent to 0.0254 meters

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

```
Enter your name:
Jim
Jim, enter a non-negative floating-point number of your weight (in pounds):
223.5
Jim, enter a non-negative floating-point number of your height (in inches):
68.77
Jim, your BMI is: 33.22653789251047
```

Todo. Implement the `Utilities.getBMI` method. See the comments there for the input parameters and requirements. The `double` return value needs **no** formatting, but it must be within a tolerance range of 0.1 against the expected value. You can also play with calculations using this web-based BMI calculator here: `https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmicalc.htm`

2.2.3 Method to Implement: `getTimeConversion`

Problem. You are asked to convert a given (non-negative) integer number of seconds and convert it into its equivalent formatted in terms of days, hours, minutes, and seconds.

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

```
Enter a non-negative integer number of seconds:
10000
0 days 2 hours 46 minutes 40 seconds
```

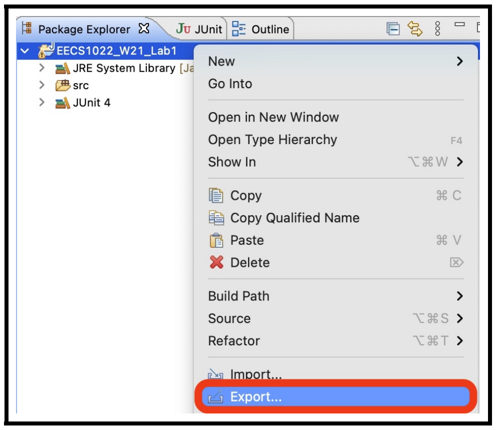
Todo. Implement the `Utilities.getTimeConversion` method. See the comments there for the input parameters and requirements. The `String` return value must conform to the expected format (e.g., words `days`, `hours`, `minutes`, and `seconds` are always in their plural form). Of course, in the output, the maximum values that can appear for hours, minutes, and seconds are, respectively, 23, 59, and 59.

2.3 Step 3: Exporting the Completed Project

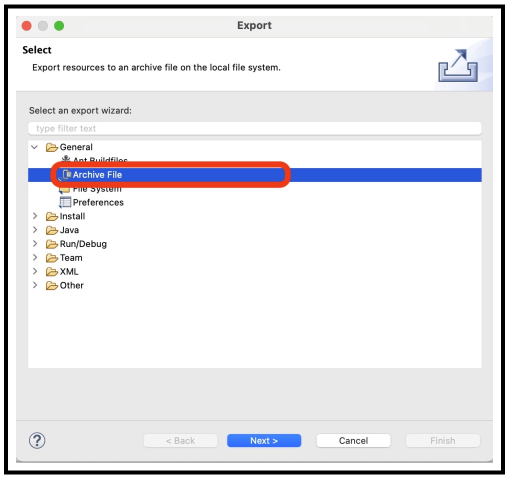
You are required to submit a Java project archive file (.zip) consisting all subfolders.

1. In Eclipse:

3.1 Right click on project `EECS1022_W21_Lab1`. Then click `Export`



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



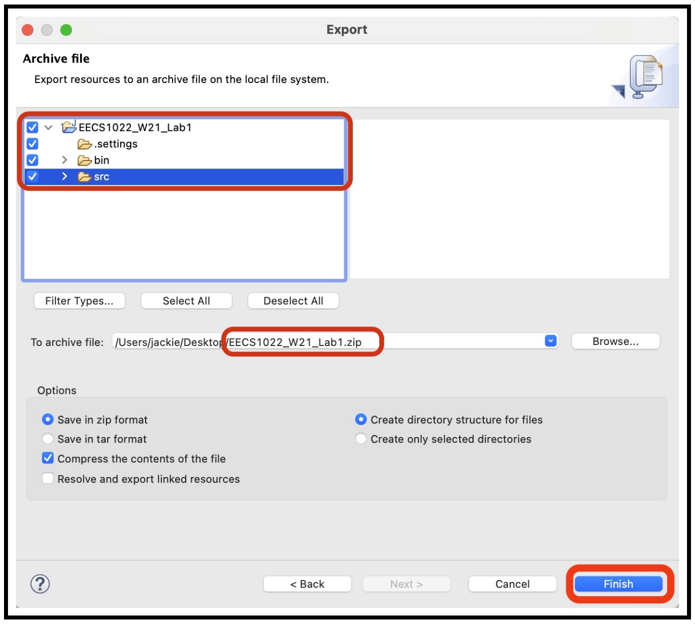
Export resources to an archive file on the local file system.

3.3 Check the top-level `EECS1022_W21_Lab1`

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_Lab1.zip` (case-sensitive)

Then `Finish`.



Export resources to an archive file on the local file system.

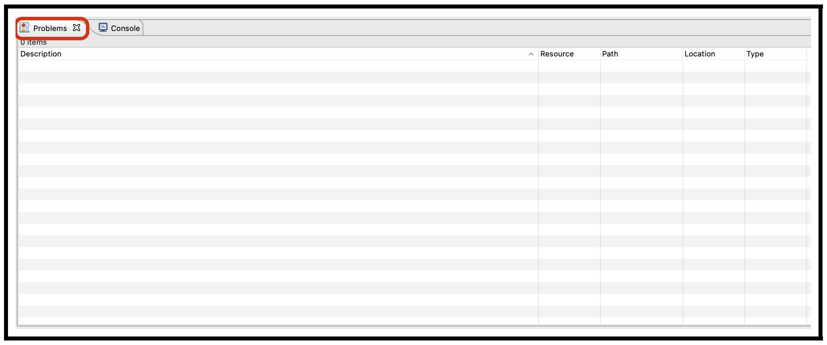
To archive file: /Users/jackie/Desktop/EECS1022_W21_Lab1.zip

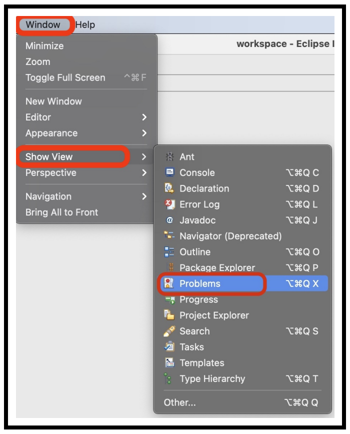
[✓ Save in zip format] Save in tar format
[✓ Compress the contents of the file]
Resolve and export linked resources
[✓ Create directory structure for files] Create only selected directories

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`.



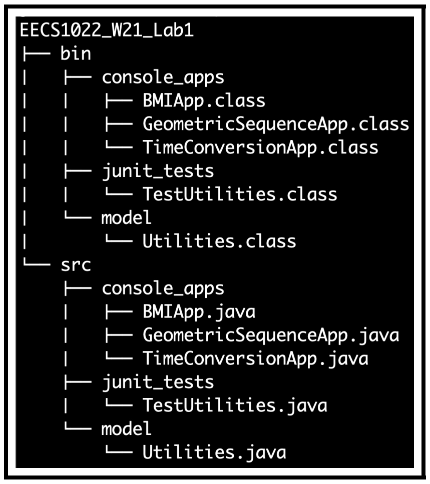


Submitting programs with errors (meaning that it cannot be run for grading) will result in possible partial, but low, marks.

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

`EECS1022_W21_Lab1.zip`

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



```
EECS1022_W21_Lab1
├── bin
|   ├── console_apps
|   |   ├── BMIApp.class
|   |   ├── GeometricSequenceApp.class
|   |   └── TimeConversionApp.class
|   ├── junit_tests
|   |   └── TestUtilities.class
|   └── model
|       └── Utilities.class
└── src
    ├── console_apps
    |   ├── BMIApp.java
    |   ├── GeometricSequenceApp.java
    |   └── TimeConversionApp.java
    ├── junit_tests
    |   └── TestUtilities.java
    └── model
        └── Utilities.java
```

Figure 1: Lab1 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 `Lab1` to submit the Java archive file: `EECS1022_W21_Lab1.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.

-