MPP Pretest
November, 2022

The purpose of this test is to assess your level of preparation in problem-solving, data structures, basic OO, and the Java programming language. For each of the problems below, write the simplest, clearest solution you can, in the form of a short program. You will be writing your code with the help of a Java compiler and the Eclipse development environment; you will not, however, have access to the internet. Because a compiler has been provided, it is expected that the code you submit for each of the problems will be free of compilation errors and will be a fully functioning program. If a solution that you submit has compilation errors, you will not receive credit for that problem.

Initially, you will receive startup code for each problem. Your task is to add new code to the startup code to meet requirements that are specified in the instructions below. Do not change the names of the methods in the startup code (though you may add new methods if you want) and do not change their signatures or access modifiers (e.g. public)).

To get a passing grade on this Pretest (so that you may go directly to MPP rather than FPP), there are two requirements:

A. You must get full credit for the Polymorphism problem (Problem 3)

B. Your total score needs to be 70% or higher

A supplement is attached to this test to remind you about set-up procedures and procedures for submitting your code; this is the same supplement you received in your onboarding instructions.

Problem 1: [40 %] [Recursion] In this problem you will write a recursive method `merge(list1, list2)` for merging two sorted linked lists of `Integer`s.
You are given as input two sorted linked lists, `list1` and `list2`. Output for your method is another linked list that contains all the elements in `list1` and `list2`, now in sorted order, obtained from merging the two lists.

Example:
`list1` = [2, 5, 8, 11] `list2` = [1,3,6] output: [1,2,3,5,6, 8, 11]

Write your code in the `Merge` class that has been provided for you in the package `prob1`. A `main` method has been provided there that lets you test your code.

Requirements for Problem 1:
(1) Your implementation must use recursion
(2) You may not use any kind of loops (no `for` loops, no `while` loops).
(3) There should be no compiler or runtime errors. In the same spirit, if your code causes a stack overflow, or does not halt, you will get no credit for this problem.

Problem 2. **[40%] [Data Structures]** For this problem, you will implement your own `ArrayList`. The startup code for this problem is in the package `prob2`. In that package there is a class `MyArrayList` containing three unimplemented methods: `add` (which adds an element to the end of the list), `remove(k)` (which removes the element in the list at

position `k`) and `toString` (which outputs a `String` representation of the list). Your task for this problem is to implement these three methods. A `main` method is provided that will allow you to test your list implementation using tests that are very similar to the tests that will be used when your work is evaluated; note that expected outputs are shown in the comments for each of these test methods.

Your code must meet the following requirements:

1. You must not modify the first line of code that appears in the body of the `MyArrayList` class – namely the initialization of the background array `arr`:

   ```
   public class MyArrayList {
       //DO NOT MODIFY THIS LINE OF CODE
       private String[] arr = new String[8];
   ```

   Also, there is an implemented `get` method in `MyArrayList` – this must not be modified (it is used for testing your code).

2. If the `add` method is called when the background array `arr` is already full, a resize must be performed so that the background array is made to be large enough to hold more elements. Your implementation should support adding any number of elements (up to the size of the largest possible `int`). Note: To add a large number of elements, it is likely that your `resize` operation will be called many times.

3. Your `add` method must ignore `null` inputs – do not allow a `null` to be added to your list.

4. The `remove(k)` method should ignore input integers `k` that are negative or that are greater than or equal to size of the list. (Note that the size of the list is the number of elements in the list – size will not usually be equal to the length of the underlying array `arr`).

5. The `toString` method should return a `String` form of the list. If the list has one or more elements (example: the list contains "A" and "B"), the `toString` method should return a `String` that begins with '[', ends with ']', and shows elements of the list in a comma-separated format. For example: if the list contains "A" and "B", `toString` will produce the string

   ```
   [A, B]
   ```

   If the list is empty, output of `toString` should be `[ ]` (see the startup code). It should never happen that the `toString` method outputs a `String` that contains "`null`" as a substring. (For instance, the following output from `toString` would indicate an incorrect implementation: `[D, null, E, null, null]`.)

6. You are not allowed to use any of the list classes contained in the Java library (such as `ArrayList`, `LinkedList`, or any other implementation of the `List` interface).

Your code should have no compiler errors and should not produce any runtime errors.

Problem 3. [20%][Polymorphism]] In the `prob3` package of your workspace, you are given fully implemented classes `Circle` and `Rectangle`. Each has a `computeArea` method, which returns the area of the figure according to the usual mathematical formulas. In the `DataMiner` class, a raw list is provided, named `objects`, and the `main` method calls `populateList` in order to populate this list with instances of `Rectangle` and `Circle`.

There is one unimplemented method in `DataMiner`: `computeAverageArea`. This method should compute the average area of the figures in the `objects` list.

For this problem you must implement `computeAverageArea` by *polymorphically computing* the area of each figure in the list. This implies that your implementation *does not check* the runtime types of the figures in the objects list.

In order to set up the code so that you can use polymorphism, create a suitable *interface* that can be implemented by both `Rectangle` and `Circle`, and insert the appropriate type parameter in the declaration of the `objects` list. Then implement `computeAverageArea`. (If you do not use an interface, you will lose credit.)

Programming Environment Set-up Instructions for Online MPP Pretests

This document describes how to set up your home laptop in order to take the MPP Pretest online.

The MPP Pretest makes use of an embedded proctoring tool. You will receive separate instructions for how to set up the proctoring tool and work with it while you are taking one of the pretests. The proctoring tool requires that you take an *onboarding test* a few days before the actual MPP Pretest; this preliminary test is given to ensure that you have been successful in the setup procedures; your answers to the programming parts of the onboarding test will not be graded.

NOTE: In an orientation video that you may have seen, it was stated that you are free to use any development environment you want for the test---this is NO LONGER TRUE. Because the number of test-takers has become very large, we need all of you to use the same IDE – namely, Eclipse. Details for set up of Eclipse are given below. We cannot accept solutions created using other tools (such as IntelliJ or Netbeans).

This document covers the follow points:

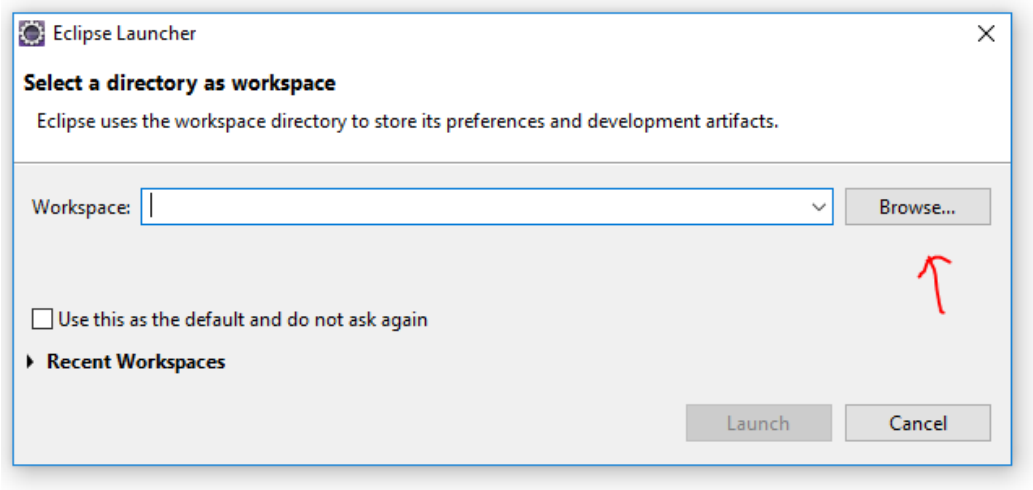
1. Software setup (not including ProctorTrack)
2. Configuring Eclipse
3. Taking the Onboarding Test and the MPP Pretest

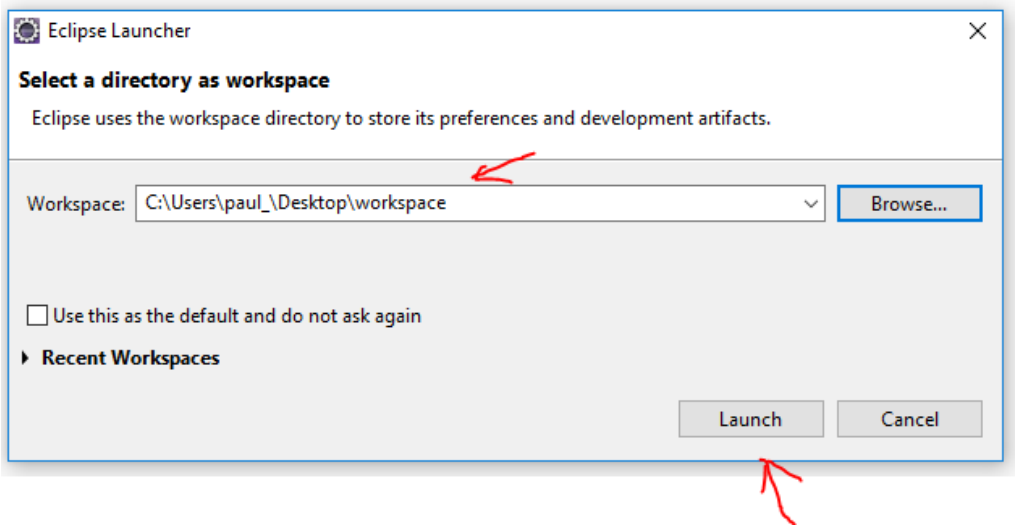
Software Setup. For convenience in assessing your work, we ask you to use the versions of Java and Eclipse that we specify here. You will download and install these as part of your software setup.

> ***Java.*** You will need to download and install Oracle jdk-17 which can be obtained by following this link for windows (use the installer) .
> https://www.oracle.com/java/technologies/downloads/#jdk17-windows
>
> For the Mac, use this link:
> https://www.oracle.com/java/technologies/downloads/#jdk17-mac
>
> ***Eclipse.*** Download the latest version of Eclipse from here:
> https://www.eclipse.org/downloads/

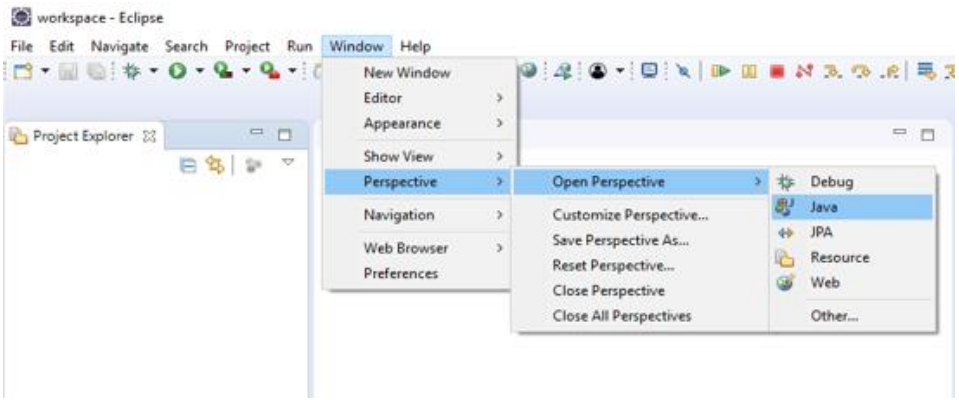
Configuring Eclipse.

1. Create a workspace. You can do this by creating a new folder on your Desktop called workspace. All the code that you write will be in this folder.
2. Launch Eclipse; at startup, it will ask for your workspace location; browse to the workspace folder that you just created and click the "Launch" button.



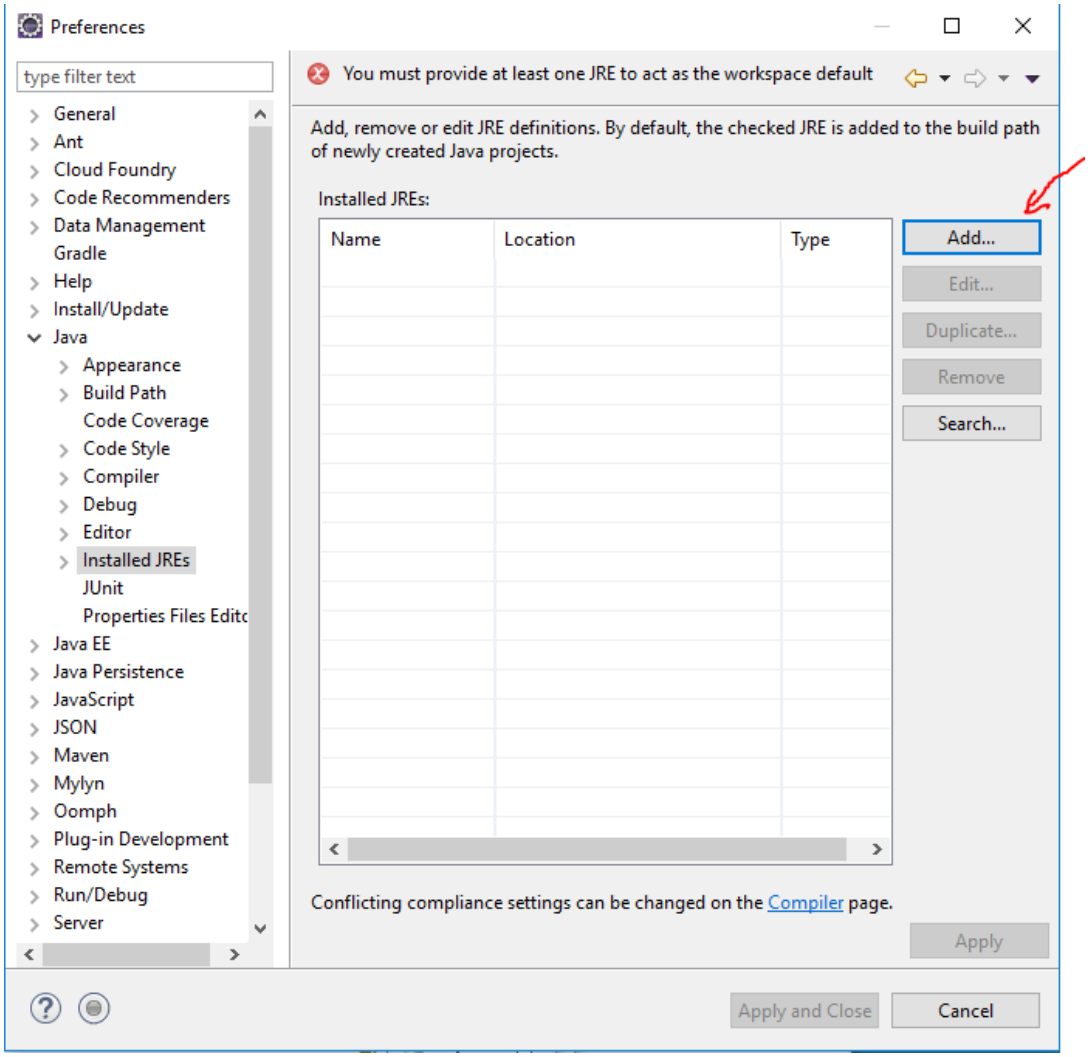


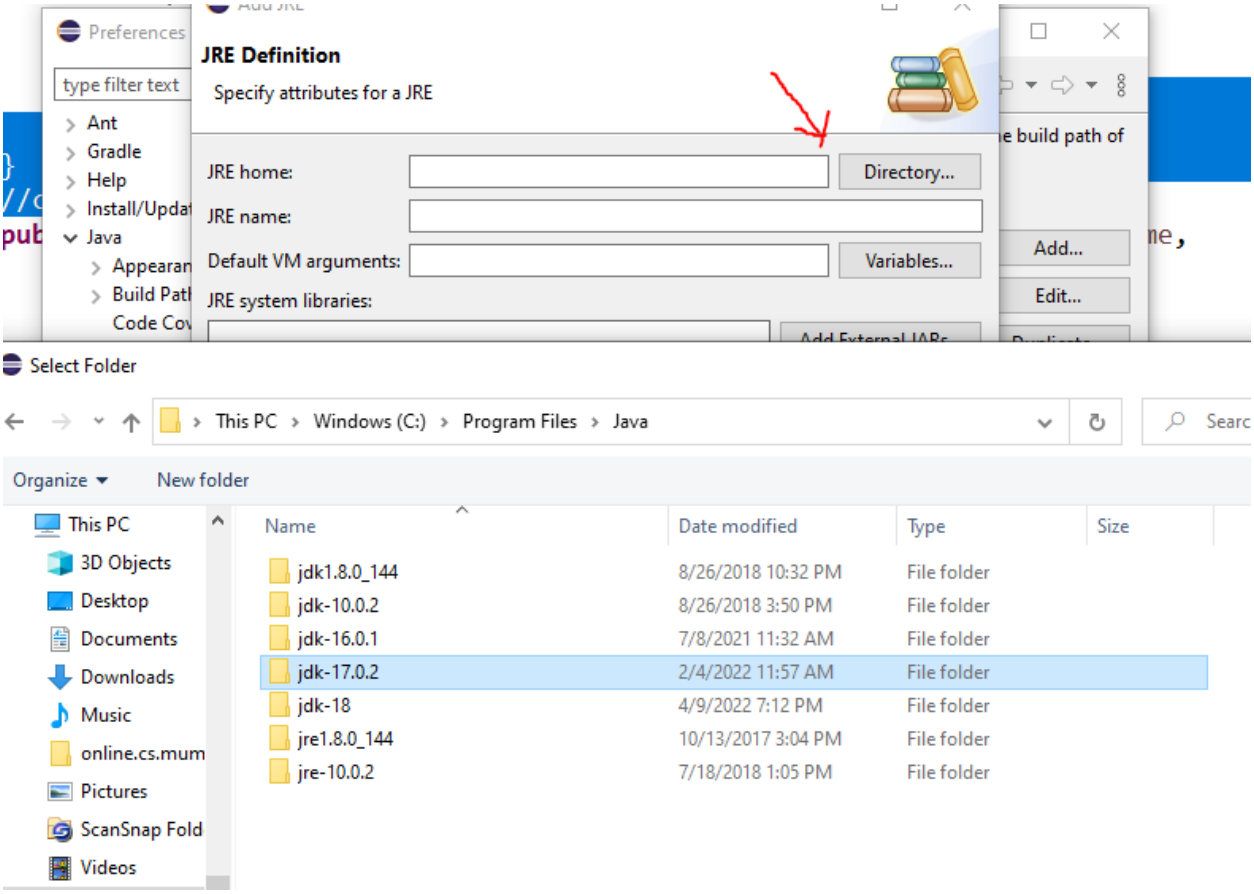
3. Close the Welcome tab and find Window along the top menu bar. Click Window > Perspective > Open Perspective > Java.



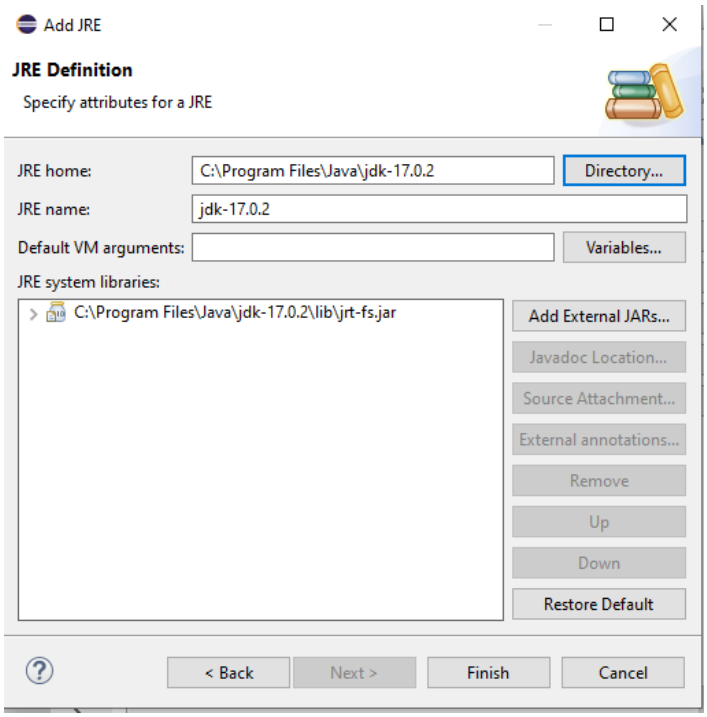
4. Point Eclipse to the jdk-17 distribution by doing the following.
   a. Go to Window > Preferences > Java > Installed JREs

   b. If jdk-17 already appears in this window, be sure the box next to this java version is checked. Otherwise, if a jdk-17 version does not appear in this window, then click Add and navigate to this jdk distribution in your file system. Select Standard VM and click Next. Click the Directory button beside the JRE home field, and then navigate to your jdk distribution.





c. When you have highlighted the folder jdk-17.0.2, click OK and you will see the following window. Click Finish.

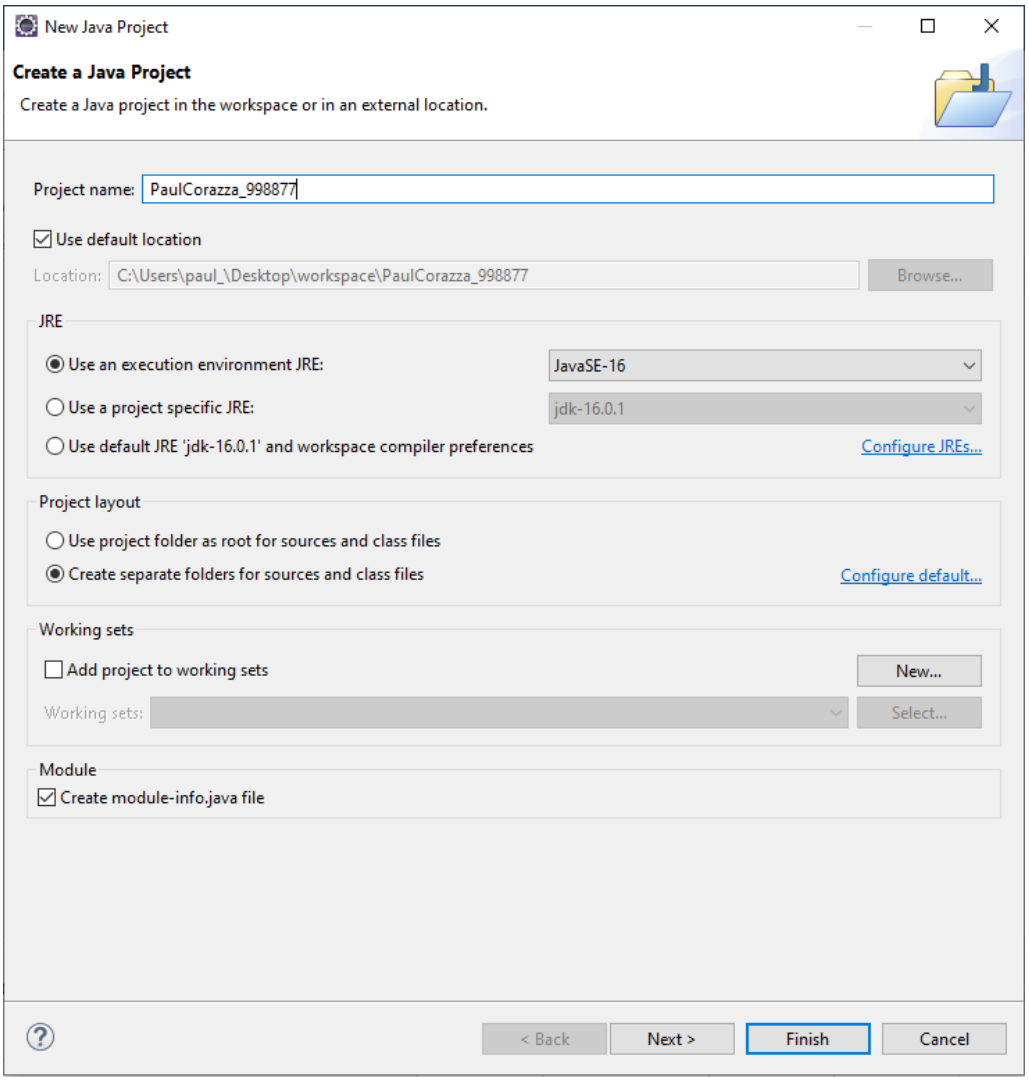


On the next window that comes up, you must check the box on the next window that asks you to specify jdk 17.0.2 as your default JRE. Then click the Apply and Close button.

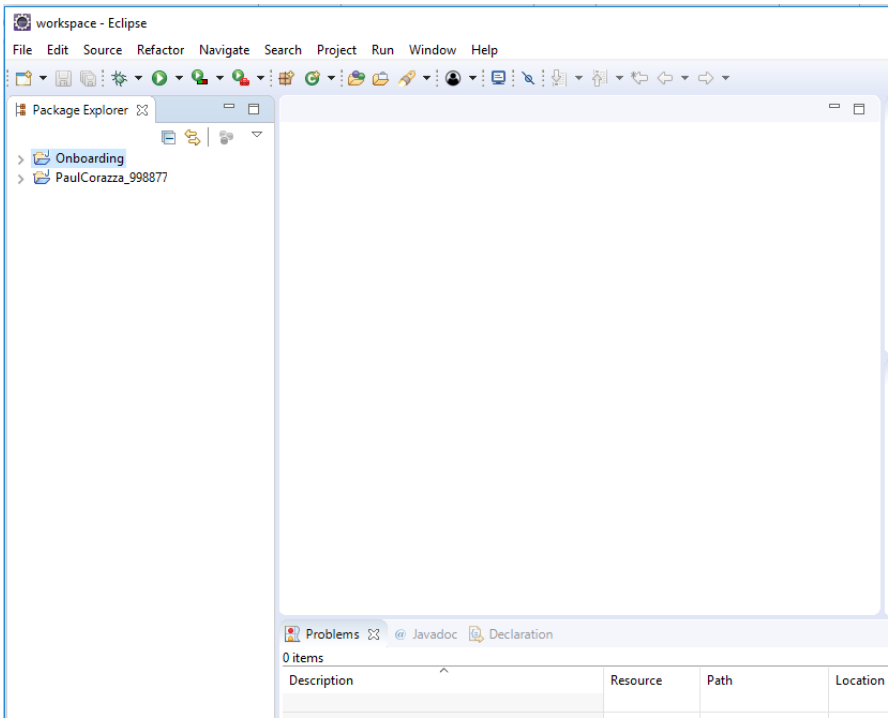
5. The next step is to upload the startup code that you have downloaded from Sakai. From Sakai, you will get zip files containing startup code for the Onboarding test and for the MPP Pretest. To upload these into Eclipse, you first create two Eclipse projects, and then add packages to those projects. All of this is described in detail below.

   NOTE: You will upload the Onboarding test first and take/submit that test a couple of days before you upload and take the actual MPP Pretest.

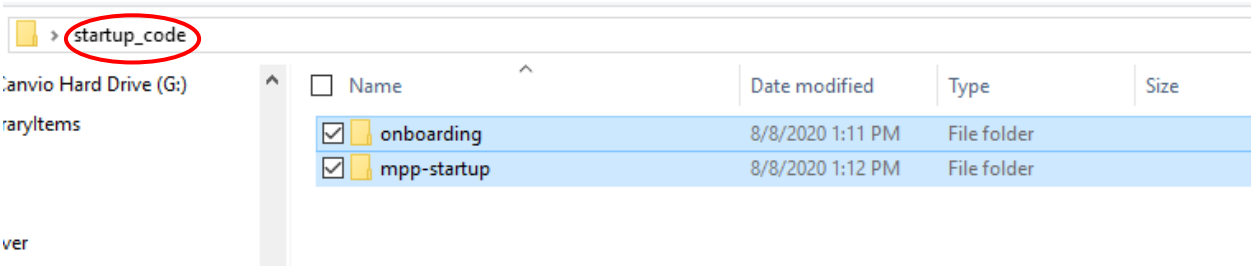
a. *MPP PreTest.* In the Package Explorer panel, right click and select New > Java Project. You will see the following window. In the Project Name field type your first name and last name (do not include more than two names) followed by an underscore `_', followed by your student ID. This is the project name for your MPP Pretest code. Click the Finish button at the bottom



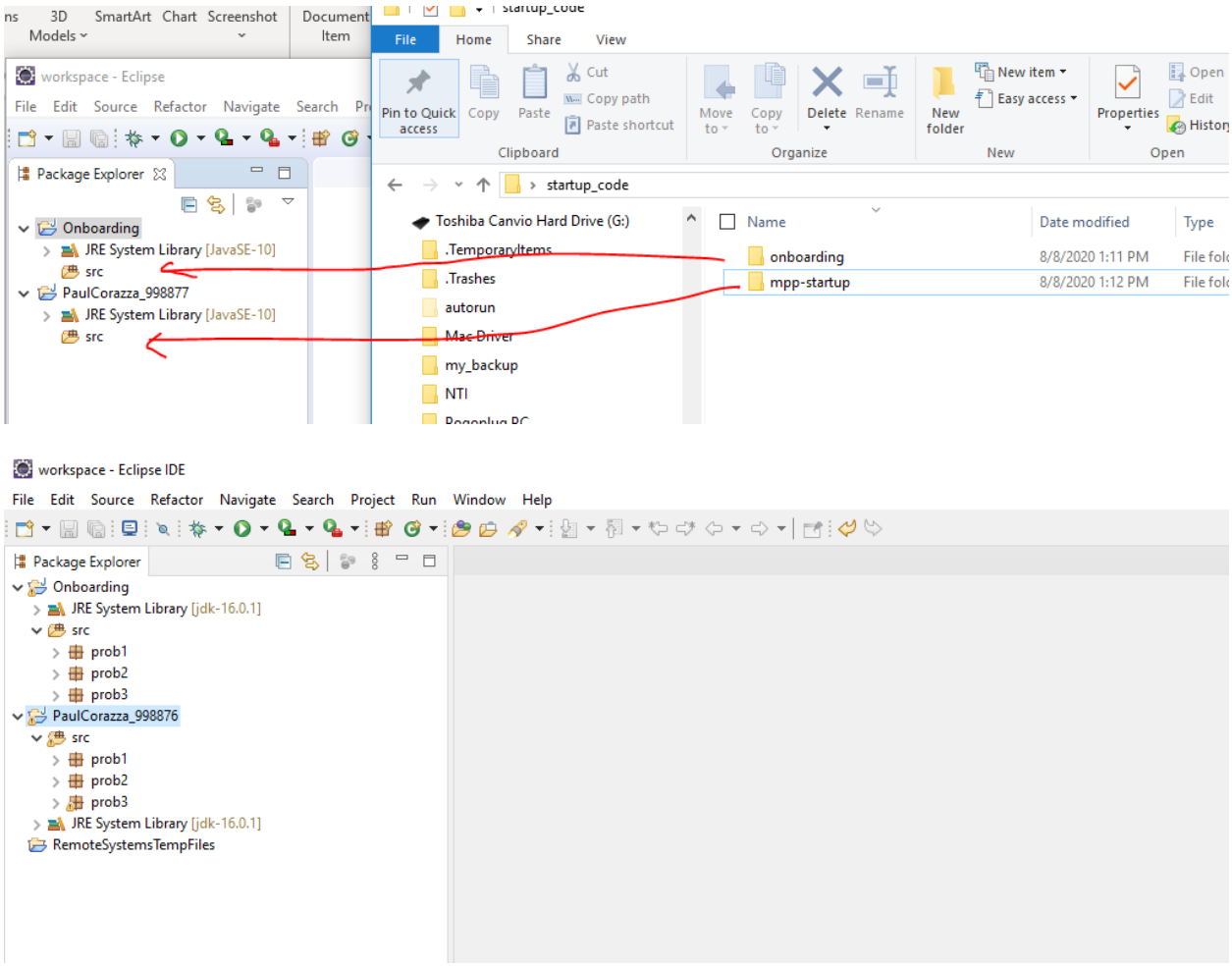
b. *Onboarding Test.* Create a second project (as in the previous step). This time, name the project Onboarding (you do not need to include you ID for this one). REMEMBER: You will create the Onboarding project and work on it a couple of days before starting work on the actual MPP Pretest. Below is a picture of these two projects as they appear in your workspace.



c. Create a folder startup_code in which to place the startup code that you retrieve from Sakai – you will find this code in a zipped folder as an attachment to the tests shown in Sakai. Unzip them and place them in start_up code folder. There will be startup code for the Onboarding test and also, later, for the MPP Pretest.



d. Each of these two folders contains three java packages, named prob1, prob2, and prob3. Copy these three from the onboarding folder into the src folder of the Onboarding project as shown below. When the MPP Pretest startupcode becomes available, do the same thing: copy the packages prob1, prob2, prob3 from the mpp-startup folder to the src folder of the mpp project (that uses your name as the project name).

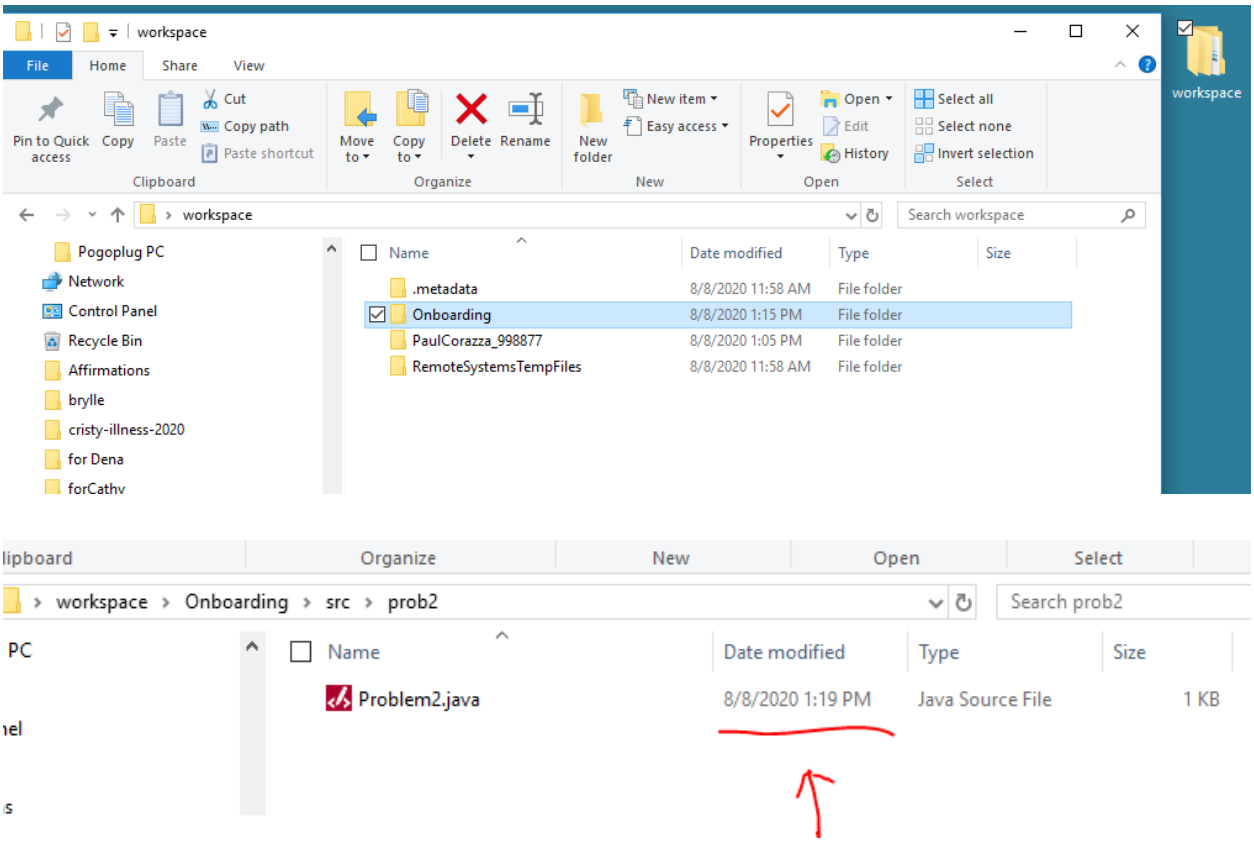


Taking the Onboarding Test and the MPP Pretest. You will receive instructions about accessing the exam instructions and startup code within the proctoring tool in a separate document. For both the onboarding and MPP pretest, you will take some steps to ensure that the proctoring tool is set up properly. Then when you are ready, you will begin working on one of the tests.

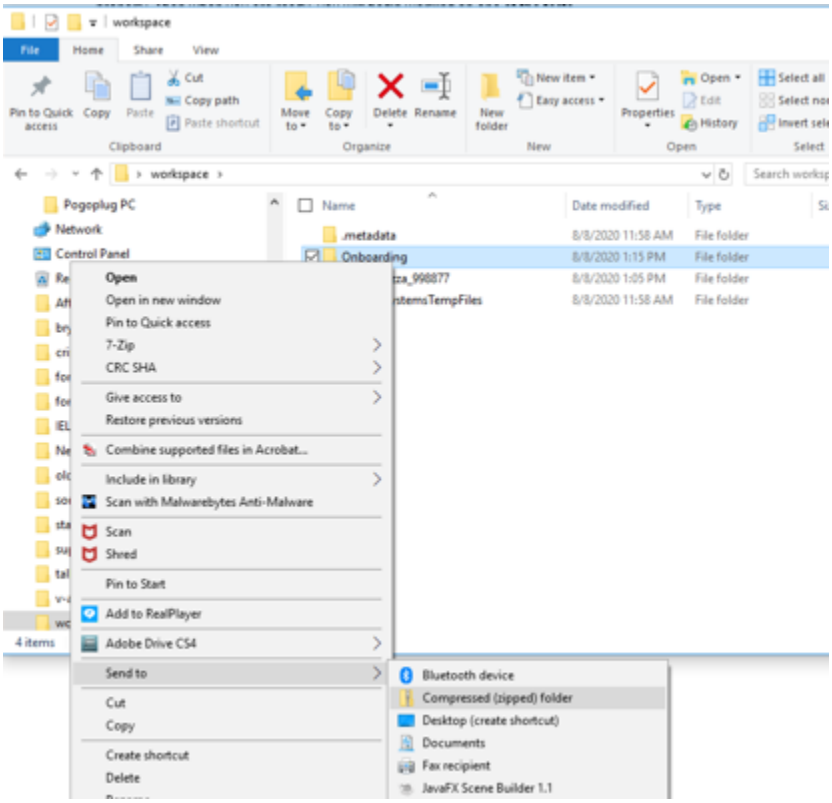
For each of the tests (onboarding and MPP pretests), you will follow the exam instrutions and write your code using the startup code as the starting point.

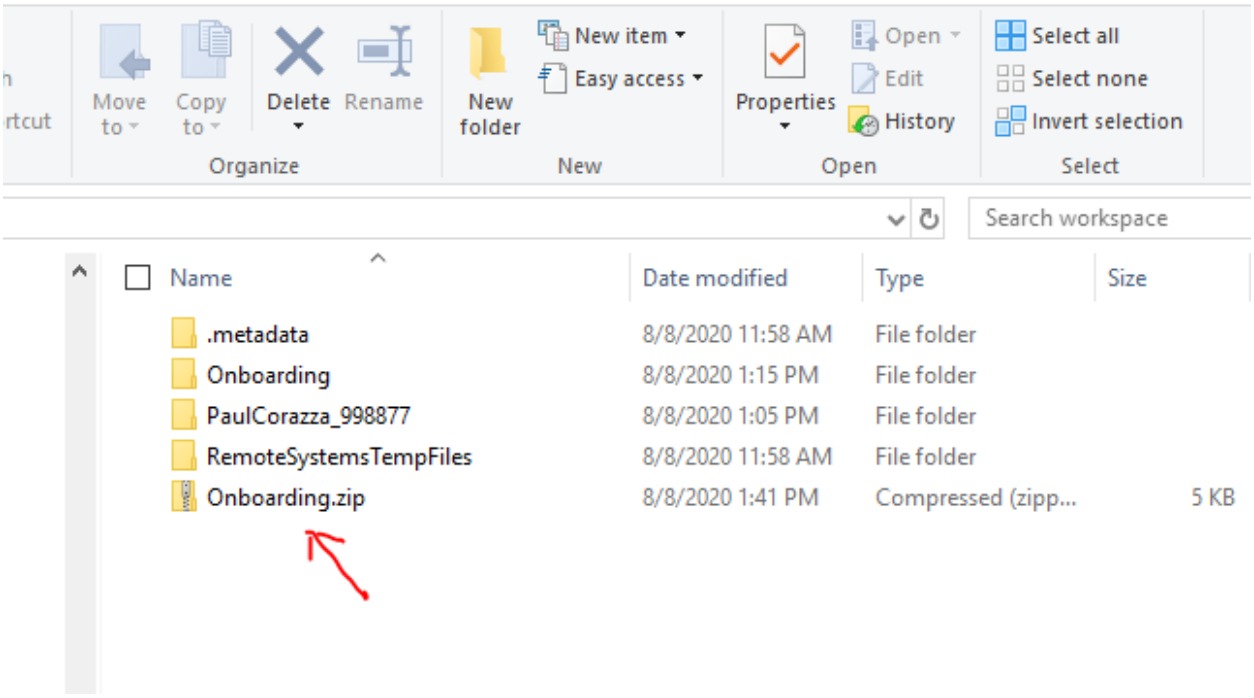
When you have finished writing your code, you will do the following. The procedure will be the same for each test.

1. Make sure your work has been saved – do this by checking your workspace folder (on the Desktop) and checking the timestamp on the files inside the folder you are ready to submit.



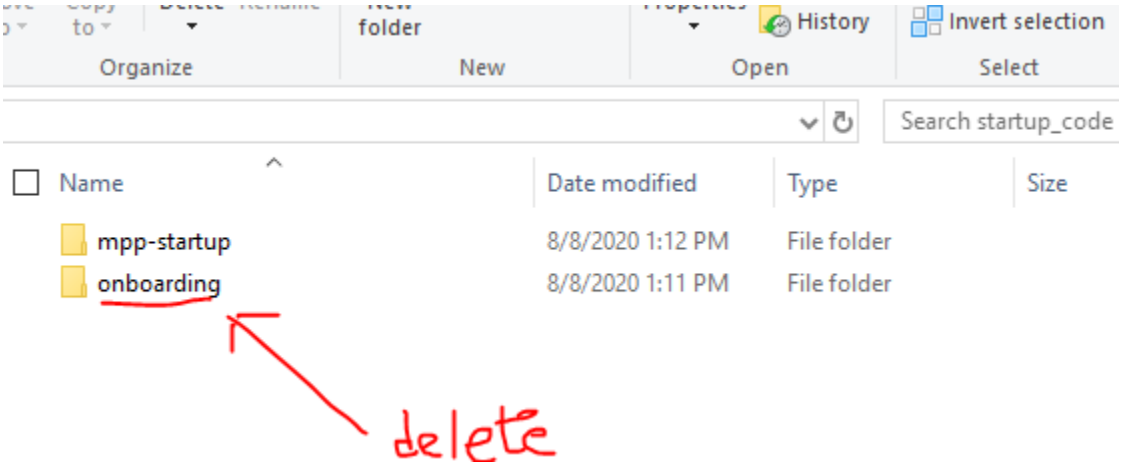
2. Then, within your workspace folder (on the Desktop), zip up the project



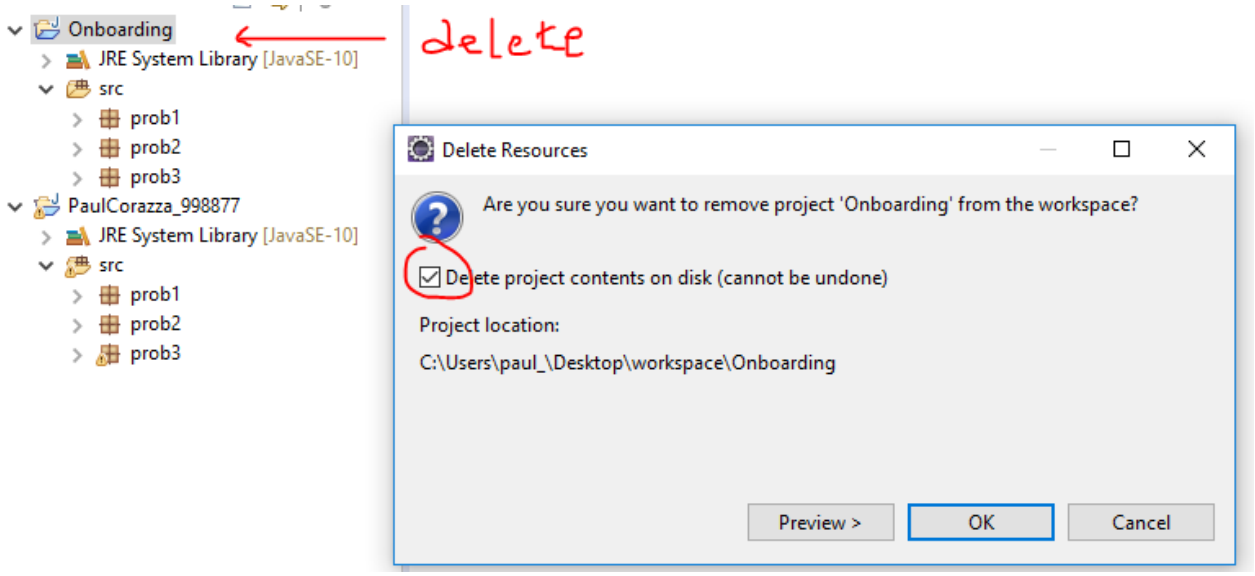


3. Submit your work by attaching this zip file in the Submit area in Sakai for this particular test. Once you have attached the file, remember to click the Submit button in Sakai.

4. *Cleanup.* Once you have submitted, go back to the startup_code folder and delete the code that you used for this test



Also, from within Eclipse, delete the work that you did on this test by right clicking the Java project, selecting delete, and clicking the option "delete files from disc".


Onboarding
JRE System Library [JavaSE-10]
src
prob1
prob2
prob3
PaulCorazza_998877
JRE System Library [JavaSE-10]
src
prob1
prob2
prob3
delete
Delete Resources
Are you sure you want to remove project 'Onboarding' from the workspace?
Delete project contents on disk (cannot be undone)
Project location:
C:\Users\paul_\Desktop\workspace\Onboarding
Preview >
OK
Cancel