MPP Pretest
February, 2023

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 `listAreEqual(list1, list2)` that will return true when the two input lists are equal, otherwise false. In this problem, lists will contain integers.

Here is the precise criterion for equality of two lists, list1 and list 2:

list1and list2 are <u>equal</u> if one of the following is true: (A) both list1 and list2 are null (B) both list1 and list2 are not null, they have the same size, every element of list1 is also an element of list2, and the elements of list1 occur in the same order as those of list2.

Examples:

(i) list1 = [1,2] and list2 = null. These lists are *not* equal
(ii) list1 = [ ] and list2 = [ ]. These lists are equal.
(iii) list1 = [1,3] and list2 = [1,2,3]. These lists are not equal (different sizes)
(iv) list1 = [1,3] and list2 = [3,1]. These lists are not equal (same elements, different order)
(v) list1 = [1,2,3] and list2 = [1,2,3]. These lists are equal.

In the `prob1` package, you have been given a class `ListsEqual` containing an unimplemented method `listsAreEqual`. Also in the `prob1` package is a class `Main`, to be used for testing your code if you wish.

For this problem, you must provide a *recursive* implementation of the method `listsAreEqual`. If you do not use recursion as the primary means of solving the problem, you will receive no credit. Be sure to follow the precise definition of equality between lists given above. The test code provided in the `main` method of `Main` tests all the possibilities; comments to these test methods tell you the expected outputs.

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) You are not allowed to use the `equals` method of any of Java's collection classes (you *are* allowed to use the `equals` method to compare `Integers`).
(4) 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.

<u>Problem 2</u>. [40%] [Data Structures] In your `prob2` package, you will find the two classes, `Employee` and `EmployeeAdmin`. A `Main` class is also provided that will make it convenient to test your code.

The `Employee` class has been fully implemented. It has three fields: `name, salary`, and `ssn` (which stores a social security number). `Employee` provides getters and setters for each of these fields.

The `EmployeeAdmin` class is intended to provide reports about `Employee`s. For this problem, the `EmployeeAdmin` class has just one static method, `prepareReport`, which accepts a `HashMap table` and a `List socSecNums` as arguments.

The `HashMap` consists of *key/value* pairs, where *key* is a social security number and *value* is the Employee that has this social security number. The `main` method in the `Main` class shows how this table could be populated in a particular case.

The `List` that is passed in to `prepareReport` is a list of employee social security numbers, represented as `String`s. Note that the social security numbers in this `List` may not match up with the social security numbers that occur as keys in the `HashMap`. For instance, some of those keys may not be present in the `List`; likewise, some of the social security numbers in the `List` may not occur as keys in the `HashMap`.

Your method `prepareReport` must produce a list of all `Employee`s in the input `table` whose social security number is in the input list `socSecNums` and whose `salary` is greater than $80,000. In addition, this list of `Employee`s must be sorted by social security number, in ascending order (from numerically smallest to numerically largest).

The `main` method in the `Main` class provides test data that you can use to test your code.

Here is an example of how the method `prepareReport` should behave: In the input `table`, you see four entries: The first entry associates the `ssn` "`223456789`" with the `Employee` object ["Jim", 90000, "`223456789`"]. There are three additional entries in the table. The list `socSecNums`, also provided as input for `prepareReport`, contains four social security numbers.

```
table:
   "223456789"  →  ["Jim", 90000, "223456789"]
   "100456789"  →  ["Tom", 88000, "100456789"]
   "630426389"  →  ["Don", 60000, "630426389"]
   "777726389"  →  ["Obi", 60000, "777726389"]

socSecNums:
    "630426389",  "223456789"  , "929333111",  "100456789"
```

When we scan the `List socSecNums` and compare with the keys in `table,` we find only three `Employee`s with matching social security numbers: Jim, Tom, Don. We also notice that, among those three, only Jim and Tom have salaries greater than $80000, so only these two `Employee`s will be returned in our final list. We then sort this list of two `Employee`s (Jim and Tom) according to the order of their social security numbers. The final output should be :

```
["Tom", 88000, "100456789"],  ["Jim", 90000, "223456789"]
```

Requirements for this problem.

(1) Your return list of `Employee`s must not contain any `Employee` object whose social security number is not in the input list `socSecNums`.
(2) Your return list of `Employees` must not contain any `null`s.
(3) All `Employees` in your return list must have salary > 80,000.
(4) Your return list of `Employees` must not contain duplicates (in other words, the same `Employee` must not be listed twice).
(5) You may not modify the `Employee` class in any way.
(6) There must not be any compilation errors or runtime errors in the solution that you submit.

<u>Problem 3</u>. [20%][Polymorphism]] In a particular video game, a player attempts to travel the greatest possible number of miles within a specified timeframe. The player will encounter various terrains as he travels. He may make use of a variety of vehicles in order to travel. If he encounters a lake, he can choose to drive a boat across the lake. If he needs to climb paths through hills, he may choose to use a bicycle. If he has an open road, he may use an automobile. As the game proceeds, in the background a record is kept of the number of miles he travels with each vehicle; this number is stored in an object representing this type of vehicle. For example, if the player uses a bicycle, a new instance of a `Bicycle` class is created, and the number of miles traveled until the next stop is stored in the `milesUsedToday` field of the `Bicycle` object.

In the `prob3` package of your workspace, you will find implementations of three types of vehicle classes: `Bicycle`, `Automobile`, and `Boat`. The game code (simulated by the `main` method) will insert the number of miles traveled in instances of these classes and arrange these instances into a `vehicles` array. This `vehicles` array may contain any combination of instances of `Bicycle`, `Automobile`, and `Boat`, according to how a given player plays the game.

For instance, a player may first ride a bicycle for a while, then switch to an automobile, then drive a boat, then drive a different automobile, and after that, may use another bicycle. In that case, the `vehicles` array would look something like this:

```
            [bike1, auto1, boat1, auto2, bike2].
```

The objective of this problem is to compute the total number of miles traveled during one play of the game. For this purpose, a class `MilesCounter` has been provided for you in your `prob3` package. `MilesCounter` has two static methods, which you need to implement.

```
public static List convertArray(Object[] vehicleArray)
public static int computeMiles(List vehicleList)
```

The `convertArray` method converts the `vehicles` array that is passed to it by the game code to a `List` of the proper type. The `computeMiles` method uses this list to polymorphically compute the total miles traveled by all the vehicles in the list.

A `main` method (in the `Main` class) has been provided, which simulates a play of the game. In that method, several vehicles are initialized and placed in an array; then a call is made to the `convertArray` method to convert the array to a list of vehicles; then the list of vehicles is passed to `computeMiles` to compute total miles; and finally the total miles are printed to the console.

In order to do your polymorphic computation of total miles, in the `convertArray` method you will need to make use of a common type for all the vehicles in the input array; the abstract class `Vehicle` (unimplemented) has been provided in your `prob3` package for this purpose. With this common type, you will be able to do the necessary polymorphic computation in `computeMiles`.

Requirements for this problem.

(1) You must compute total miles using polymorphism. (For instance, if you obtain the total miles by first computing miles from each of the vehicles and then summing these, without using polymorphism, you will receive no credit.)

(2) Your implementation of `computeMiles` may not check types (using `instanceof` or `getClass()`) in order to read mileage from any of the vehicles in the input list.

(3) You must use parametrized lists, not "raw" lists. (Example: This is a parametrized list: `List<Duck> list`. This is a "raw" list: `List list`.) This means that all `Lists` that appear in the code (in the `Main` class and in the `MilesCounter` class) must be given proper type parameters.

(4) You must implement both the methods `convertArray` and `computeMiles` in the `MilesCounter` class.

(5) You must implement and use the provided abstract class `Vehicle` in carrying out a polymorphic computation of total miles in `computeMiles`.

(6) Your computation of total number of miles traveled must be correct.

(7) There must not be any compilation errors or runtime errors in the solution that you submit.

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

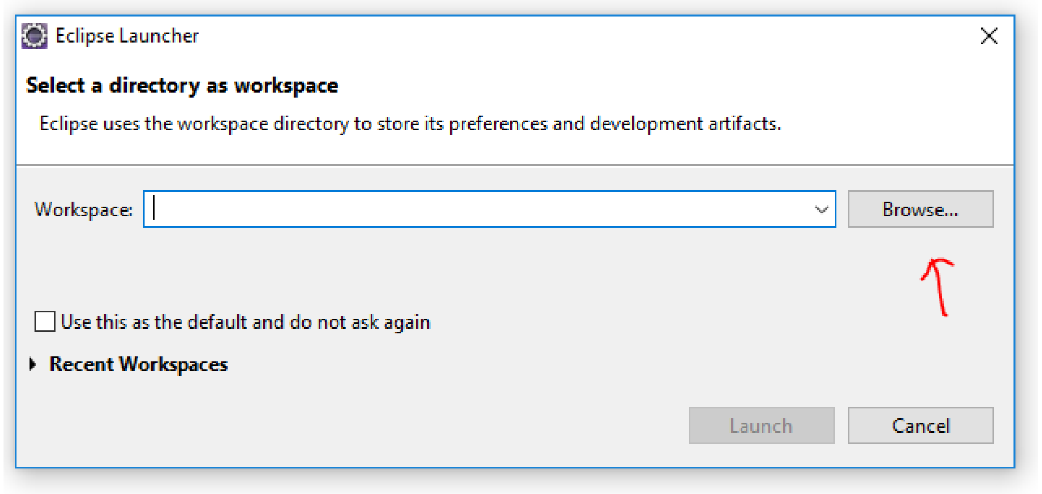
For the Mac, use this link:
https://www.oracle.com/java/technologies/downloads/#jdk17-mac

Eclipse. Download the latest version of Eclipse from here:
https://www.eclipse.org/downloads/

NOTE: The rest of the setup instructions talk about jdk 17.0.2 – however the latest Java 17 version (as of February 2023) is jdk 17.0.6. The instructions below will work just as well for jdk 17.0.6.

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.

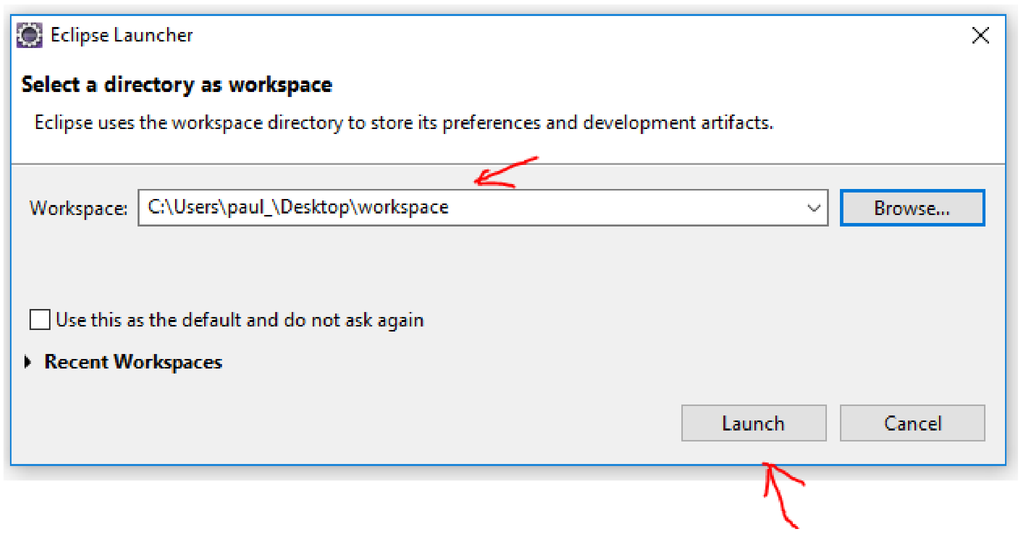


Eclipse Launcher

Select a directory as workspace

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

Workspace:



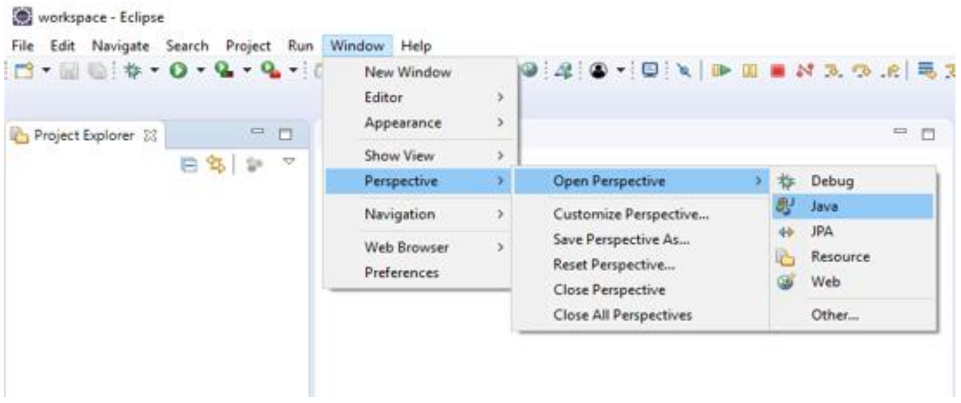
Eclipse Launcher

Select a directory as workspace

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

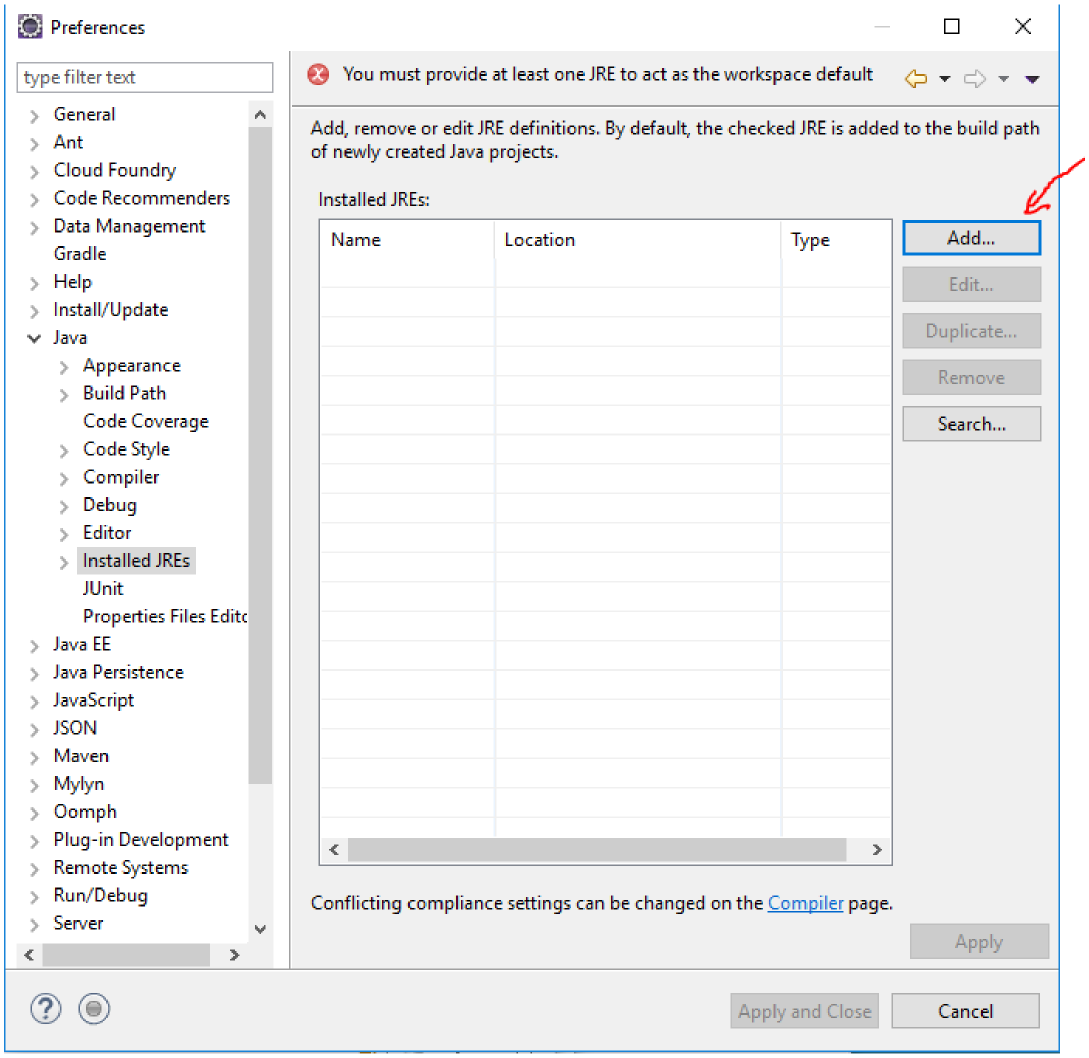
Workspace: C:\Users\paul_\Desktop\workspace

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.



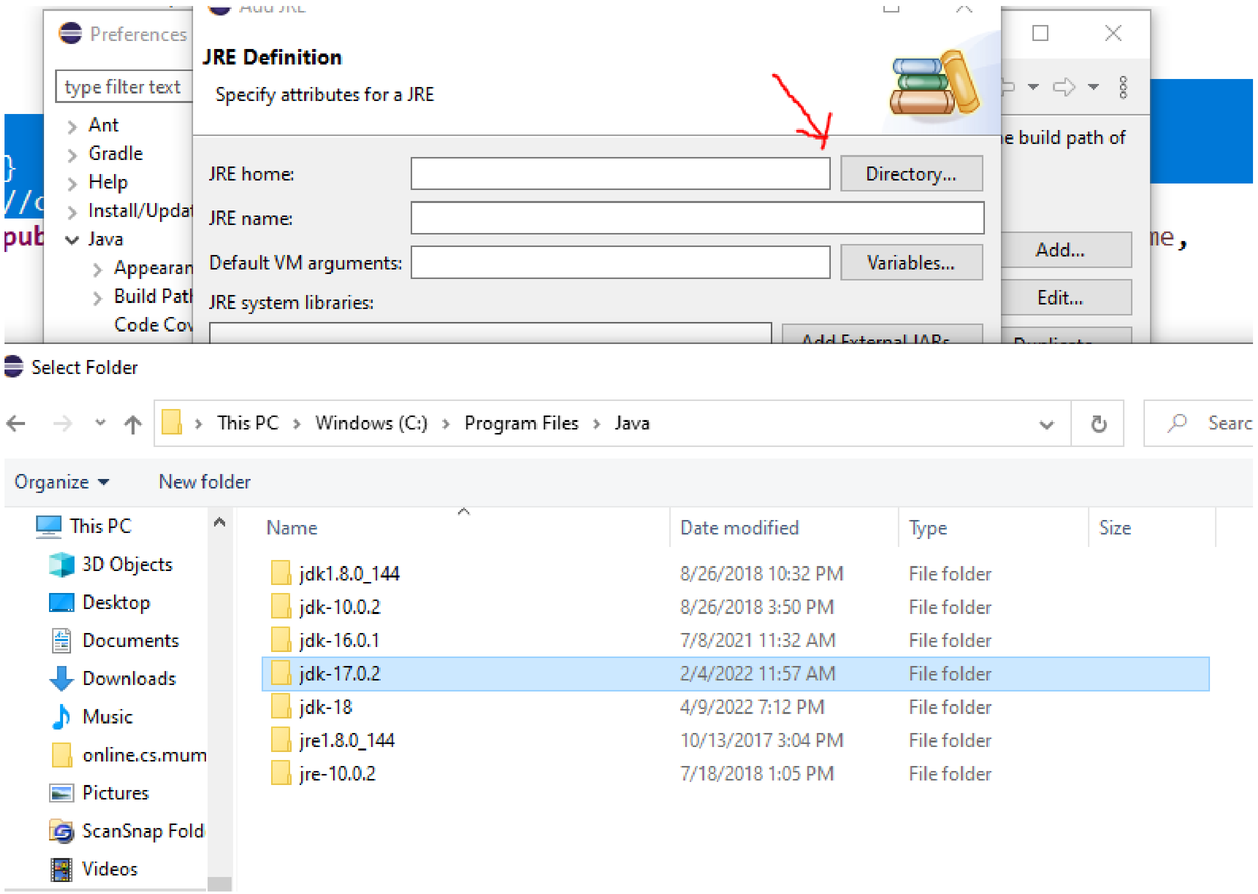
You must provide at least one JRE to act as the workspace default

Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

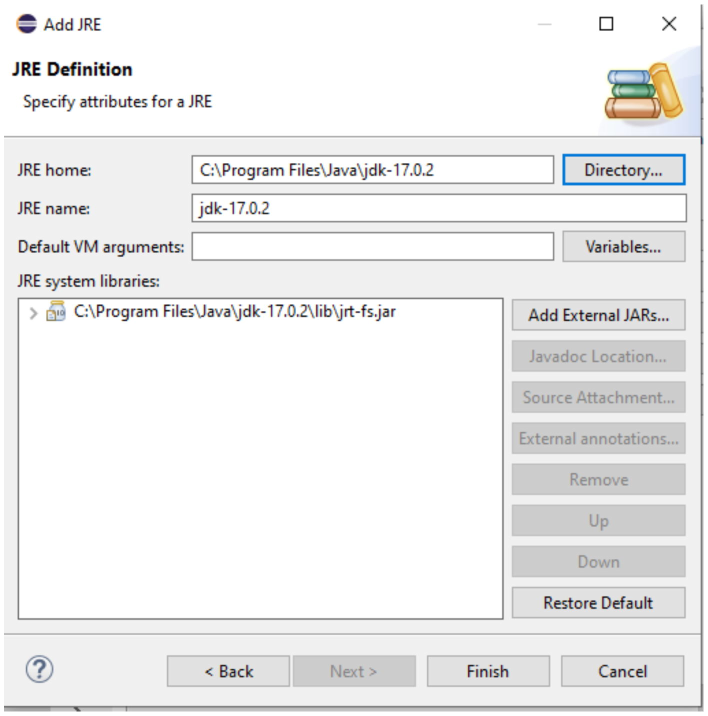
Name	Location	Type

Conflicting compliance settings can be changed on the Compiler page.



Name	Date modified	Type	Size
jdk1.8.0_144	8/26/2018 10:32 PM	File folder	
jdk-10.0.2	8/26/2018 3:50 PM	File folder	
jdk-16.0.1	7/8/2021 11:32 AM	File folder	
jdk-17.0.2	2/4/2022 11:57 AM	File folder	
jdk-18	4/9/2022 7:12 PM	File folder	
jre1.8.0_144	10/13/2017 3:04 PM	File folder	
jre-10.0.2	7/18/2018 1:05 PM	File folder	

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



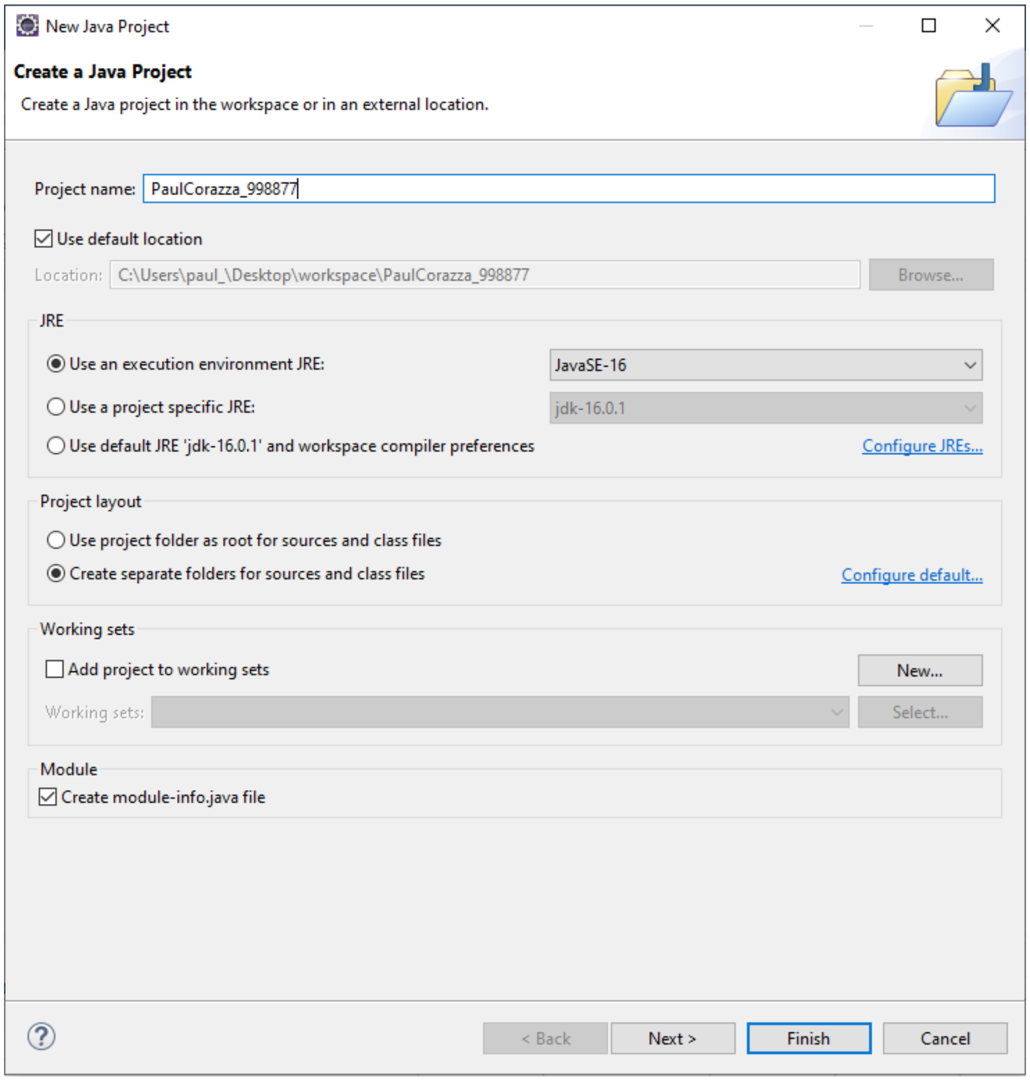
JRE home: C:\Program Files\Java\jdk-17.0.2
JRE name: jdk-17.0.2
Default VM arguments:
JRE system libraries:
C:\Program Files\Java\jdk-17.0.2\lib\jrt-fs.jar

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


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: PaulCorazza_998877

☑ Use default location

Location: C:\Users\paul_\Desktop\workspace\PaulCorazza_998877

JRE

● Use an execution environment JRE: JavaSE-16

○ Use a project specific JRE: jdk-16.0.1

○ Use default JRE 'jdk-16.0.1' and workspace compiler preferences

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

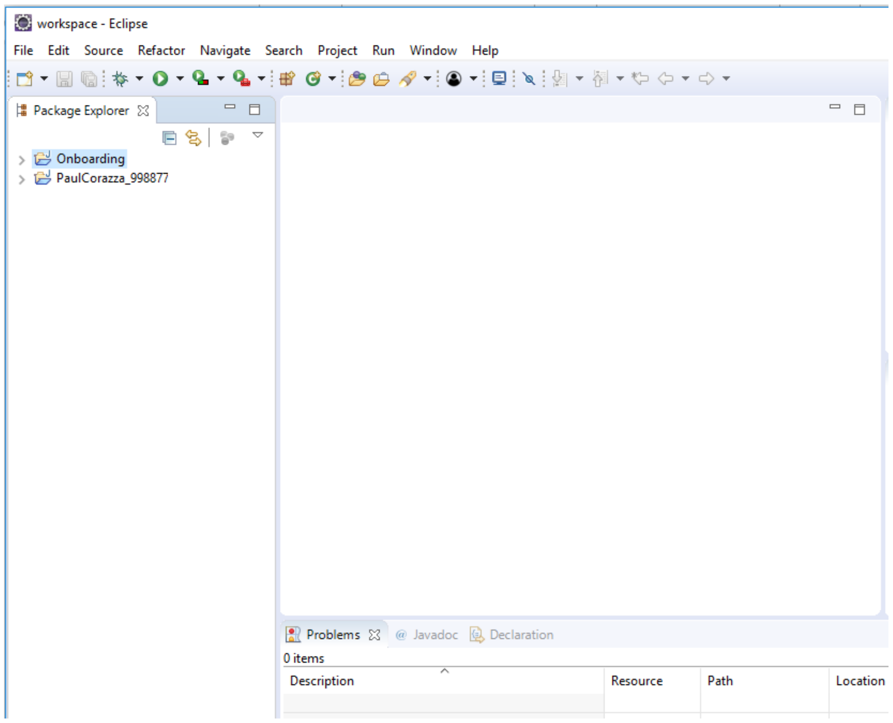
☐ Add project to working sets

Working sets:

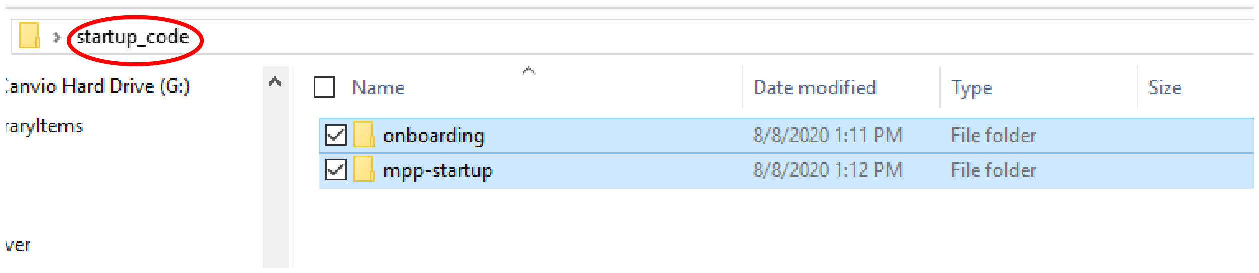
Module

☑ Create module-info.java file

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.

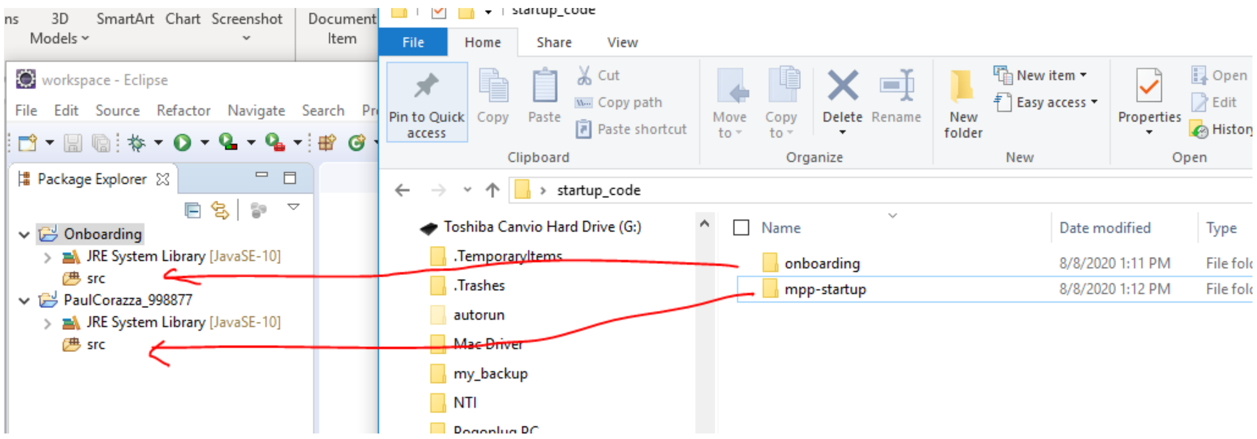


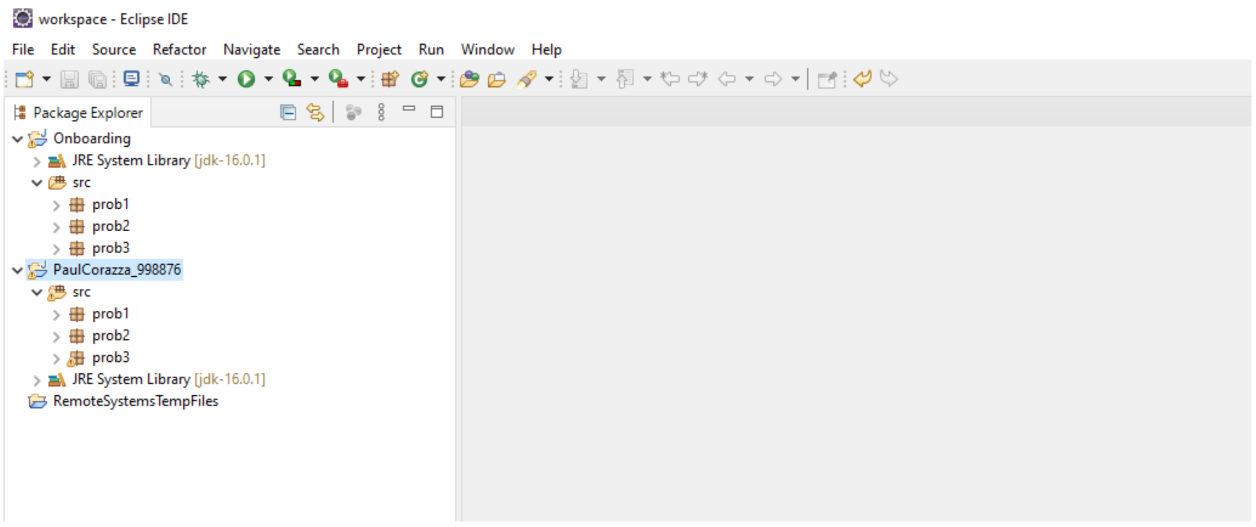
c. Create a folder startup_code in which to place the startup code that you retrieve from Sakai – you will find this code in a zipped folder as an attachment to the tests shown in Sakai. Unzip them and place them in start_up code folder. There will be startup code for the Onboarding test and also, later,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



Name	Date modified	Type	Size
onboarding	8/8/2020 1:11 PM	File folder	
mpp-startup	8/8/2020 1:12 PM	File folder	

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).



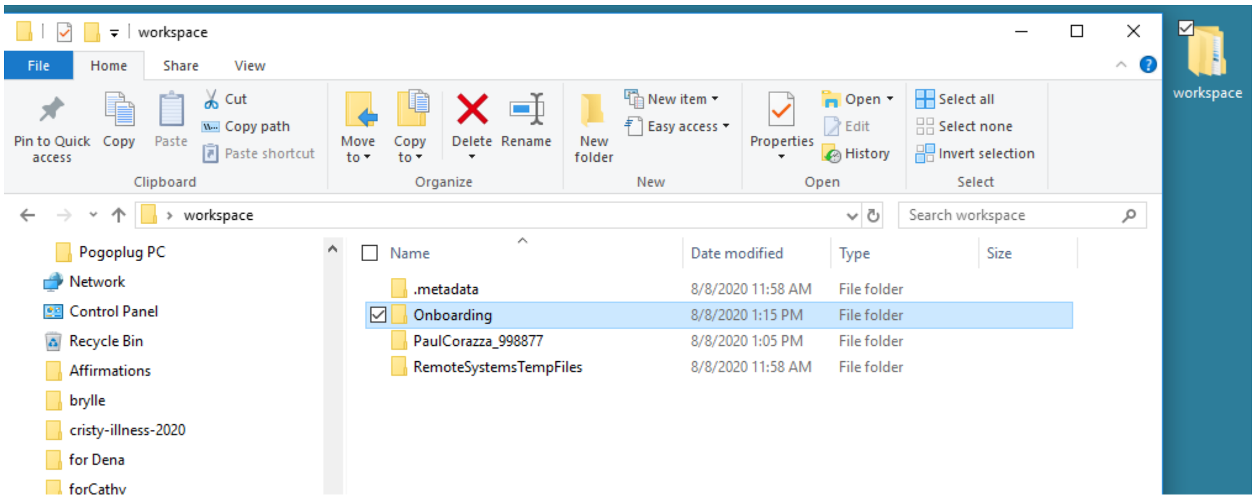


Taking the Onboarding Test and the MPP Pretest. You will receive instructions about accessing the exam instructions and startup code within the proctoring tool in a separate document. For both the onboarding and MPP pretest, you will take some steps to ensure that the proctoring tool is set up properly.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

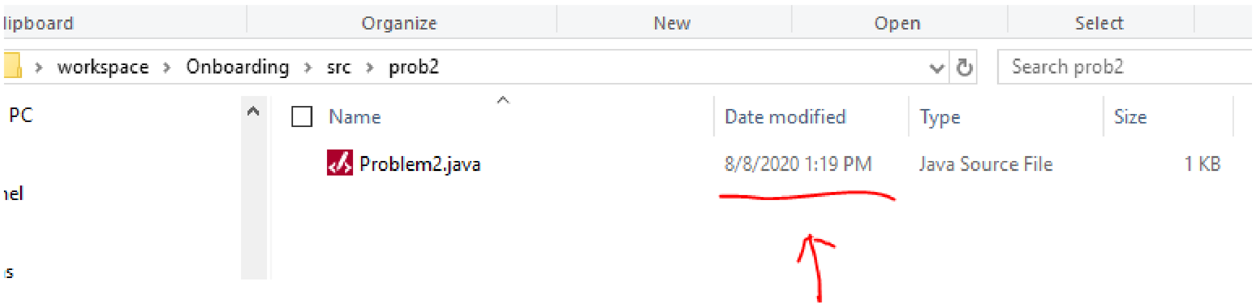
For each of the tests (onboarding and MPP pretests), you will follow the exam instrutions and write your code using the startup code as the starting point.

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.

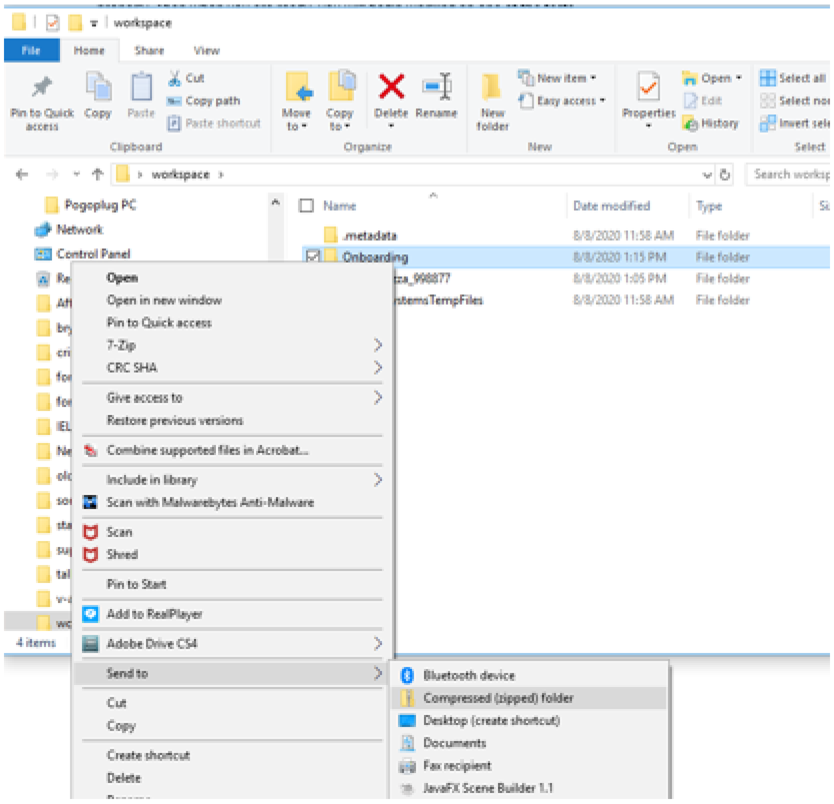


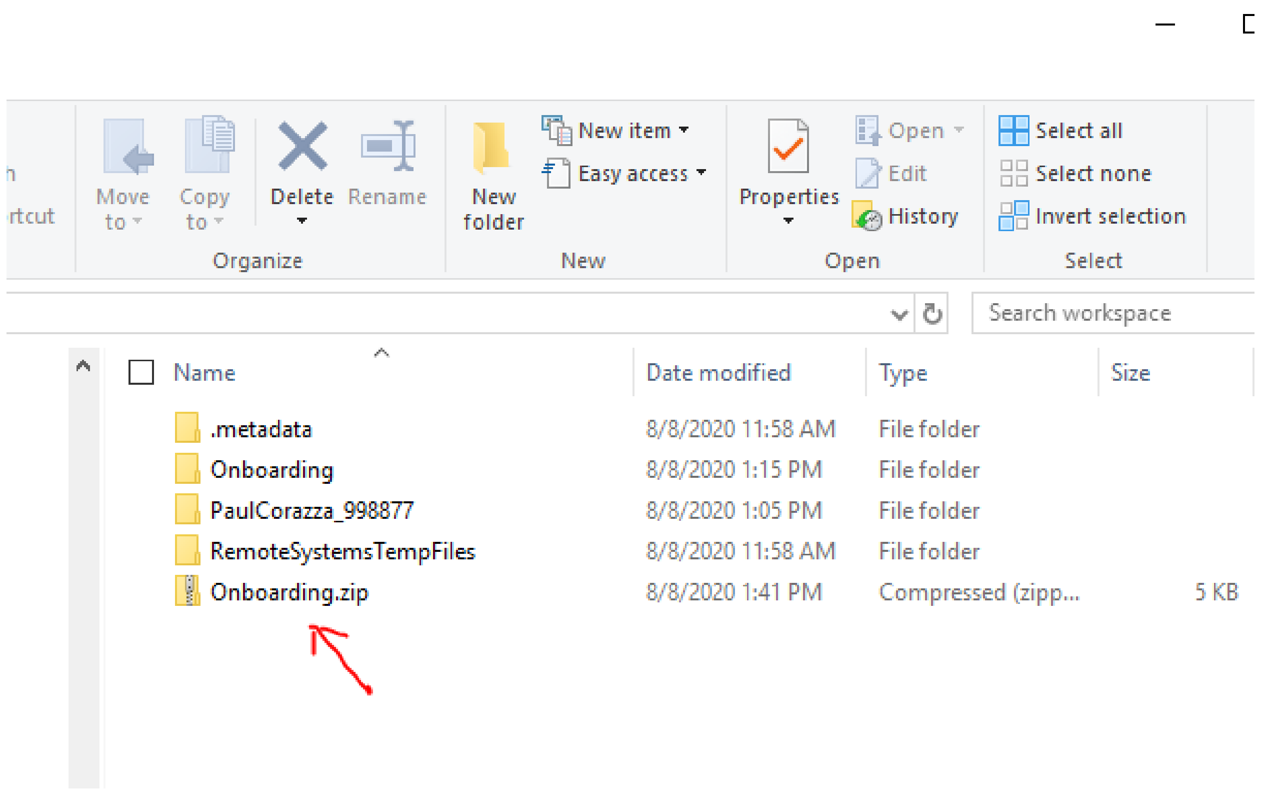
Name	Date modified	Type	Size
.metadata	8/8/2020 11:58 AM	File folder	
Onboarding	8/8/2020 1:15 PM	File folder	
PaulCorazza_998877	8/8/2020 1:05 PM	File folder	
RemoteSystemsTempFiles	8/8/2020 11:58 AM	File folder	



Name	Date modified	Type	Size
Problem2.java	8/8/2020 1:19 PM	Java Source File	1 KB

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

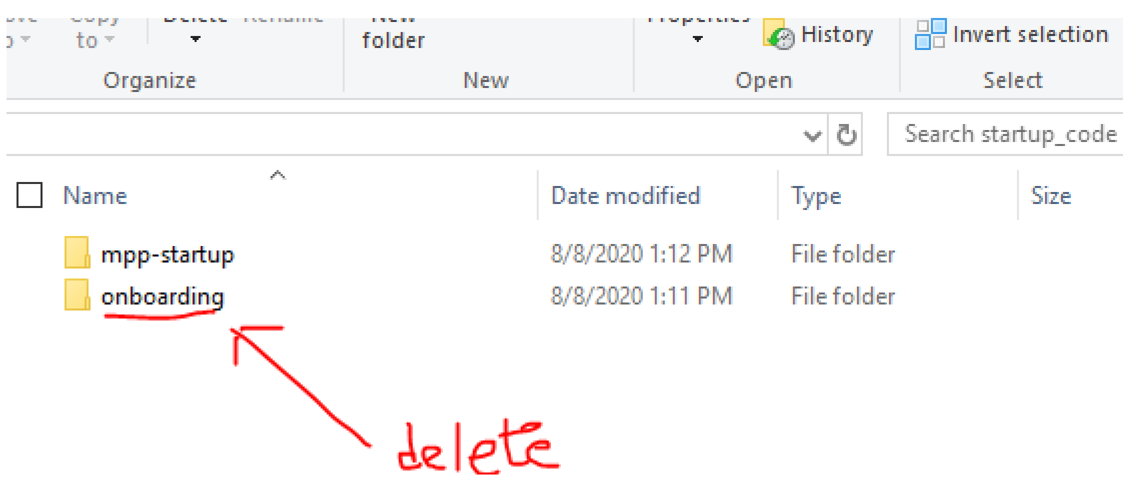




Name	Date modified	Type	Size
.metadata	8/8/2020 11:58 AM	File folder	
Onboarding	8/8/2020 1:15 PM	File folder	
PaulCorazza_998877	8/8/2020 1:05 PM	File folder	
RemoteSystemsTempFiles	8/8/2020 11:58 AM	File folder	
Onboarding.zip	8/8/2020 1:41 PM	Compressed (zipp...	5 KB

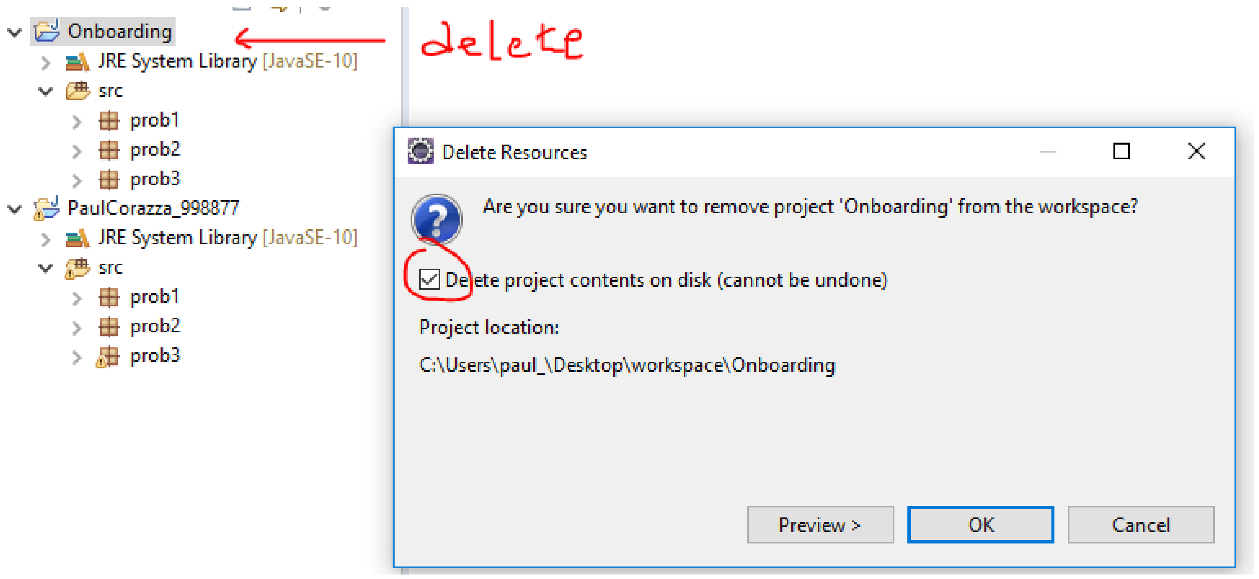
3. Submit your work by attaching this zip file in the Submit area in Sakai for this particular test. Once you have attached the file, remember to click the Submit button in Sakai.

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



Name	Date modified	Type	Size
mpp-startup	8/8/2020 1:12 PM	File folder	
onboarding	8/8/2020 1:11 PM	File folder	

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".



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