MPP Pretest
February, 2024

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 solution to the problem of building a frequency table for a linked list of `Integers`.

Given a list of `Integers`, a frequency table associates to each `Integer` in the list the number of times the `Integer` occurs. For example, given the list

[4, 8, 4, 4, 2, 8],

a frequency table for the list would be:

4	3
8	2
2	1

In your `prob1` package, you will find a class `Frequency` that contains an *unimplemented* static method with signature and return type given by:

```
HashMap<Integer, Integer> recurFreqTable(LinkedList<Integer> list)
```

For this problem, you must implement the `recurFreqTable` method by using recursion. Output of the method must be a frequency table for the input `list`, in the form of a `HashMap`. A `main` method has been provided in the class `Frequency` that allows you to test your implementation, and expected output is shown in the comments for that method.

Requirements for Problem 1:

(1) Your implementation of the method `recurFreqTable` must be based on recursion.
(2) You may not use any kind of loops (no `for` loops, no `while` loops).
(3) The output of your implementation of `recurFreqTable`, when run on any `LinkedList` consisting of `Integer`s, must be a frequency table for the list; the order in which entries in the table occur is not important.
(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.

Problem 2. **[Data Structures]** The *union* of two lists `list1` and `list2` is the list obtained by adding all elements of `list2` to `list1`. You can form the union of three or more lists in the same way. In your `prob2` package, you will find an unimplemented method having the following signature and return type:

```
List<String> extractList(List<List<String>> lists, String word1,
                         int num1, String word2, int num2)
```

The method `extractList` returns the union—in *sorted order*—of those lists L belonging to the input `lists` that meet *both* of the following requirements:

1. L has `num1` or more occurrences of the `String word1`
2. L has *fewer than* `num2` occurrences of the `String word2`.

If none of the lists in the input `lists` satisfies *both* of these requirements, the return value of `extractList` should be an empty list (*not* a null list). If *just one* of the lists in the input `lists` meets both of these requirements, then the list that does meet the requirements should be returned in sorted order. If two or more of the lists in the input `lists` meet both of these requiements, then the union of these lists must be formed, and the resulting list should be returned in sorted order.

Here are some examples that show how `extractList` should behave on various inputs:

First Example

Input: `lists`=[[cat, dog, mouse, cat], [cat, cat, horse, cat, bug], [caterpillar, horse, dog, fly], [gorilla, tiger, cat, bug, cat]];
`word1`= cat
`num1` = 2
`word2` = bug
`num2` = 1

Expected Output: [cat, cat, dog, mouse]

Explanation: The input contains 4 lists. The first list [cat, dog, mouse, cat] meets the criteria described above: it has 2 or more occurrences of word1 (which is cat) and fewer than 1 occurrence of word2 (which is bug – it has 0 occurrences of bug). The second list [cat,cat, horse,cat, bug] satisfies the first requirement since it has more than 2 occurrences of word1, but does not satisfy the second requirement since it contains 1 occurrence (and so not fewer than num2 occurrences) of word2 (which is bug). The third list [caterpillar, horse, dog, fly] fails to satisfy the first requirement since it contains no occurrence of word1 (it does satisfy the second requirement since it has 0 occurrences of word2). The fourth list [gorilla, tiger, cat, bug, cat] meets the first requirement (having 2 occurrences of cat) but fails to satisfy the second requirement since it does not have *fewer than* 1 occurrence of word2 (it has exactly 1 occurrence of word2). Since the first list [cat, dog, mouse, cat] is the only list satisfying both requirements, this list is returned in sorted order.

Second Example

Input: `lists`= [[cat, dog, mouse, cat], [cat,cat, horse, cat, bug], [caterpillar, horse, dog, fly], [gorilla, tiger, cat, bug, cat], [dog, dog, tree, horse]]
`word1`= dog
`num1` = 1
`word2` = cat
`num2` = 2

Expected Output: [caterpillar, dog, dog, dog, fly, horse, horse, tree]

Explanation: The input lists contains 5 lists. Note that the third list [caterpillar, horse, dog, fly] and the fifth list [dog,dog, tree, horse] both have at least 1 (num1) occurrence of dog (word1) and fewer than 2 (num2) occurrences of cat (word2); the other lists in lists do not meet both requirements. The return value is therefore the union of [caterpillar, horse, dog, fly] and [dog,dog, tree, horse], arranged in sorted order.

The startup code for this problem has a `main` method that provides test data that will allow you to check your work; expected solutions are provided. As in the other problems on this pretest, to get credit for this problem, your code must compile and run without runtime errors.

Problem 3. [Polymorphism] In a university library, there are two kinds of library members: faculty and student. Each type of library member can check out books and other items, but fees for late returns are computed differently for these two types of library members. From time to time, the library administration produces a report that includes the total number of books that have been checked out but that are overdue. You will implement a method in this problem that computes this total.

In the `prob3` package you will find classes `Faculty, Student,` and `CheckoutRecord.` The `Faculty` class represents a faculty library member and the `Student` class represents a student library member. Each library member class has a variable of type `CheckoutRecord`, which keeps track of the list of the books checked out by the member, and also a list of the overdue books that the member still has in his possession. Each library member class provides a public `getRecord` method that provides access to the member's `CheckoutRecord`.

For this problem, you will need to implement the two unimplemented methods in the `Admin` class:

```
List combineLists(List faculty, List students)
int countNumOverdueBooks(List combined)
```

The method `combineLists` will combine the two input lists into a single list with a common type. To produce the combined list, you will need to implement a common supertype for `Student` and `Faculty`. An abstract class `LibraryMember` has been provided for this purpose; you will need to implement this class appropriately.

The method `countNumOverdueBooks` computes the total number of overdue books for all library members in the input list. This computation is done by polymorphically reading the `CheckoutRecord` of each libarary member and then reading the size of the `overdueBooks` list. Finally, this method returns the computed total.

A `Main` class, with a `main` method, has been provided to permit you to test your code. Test data has been provided in the `main` method; at the bottom of the `main` method, the test data is used to run a test on the two methods in `Admin` that you will implement. Expected output is displayed in the comments.

Requirements for this problem.

(1) You must read `CheckoutRecord`s from the list of library members (in the method `countNumOverdueBooks`) *using polymorphism*. You may not use `instanceof` or `getClass()` to check types when you read `CheckoutRecords` from the library member objects.

(2) You must use parametrized lists, not "raw" lists. (Example: This is a parametrized list: `List<Duck> list`. This is a "raw" list: `List list`.) You will need to replace every occurrence of a raw list with an appropriately typed parametrized list.

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

(4) You may not code new classes, enums, or interfaces for this problem; you must use only the classes provided in the `prob3` package.

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

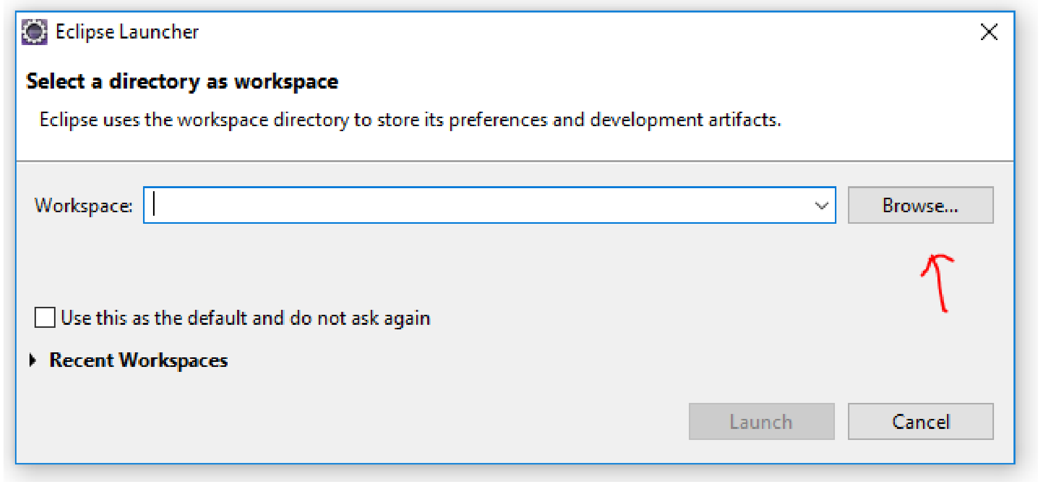
For the Mac, use this link:
https://www.oracle.com/java/technologies/downloads/#jdk17-mac

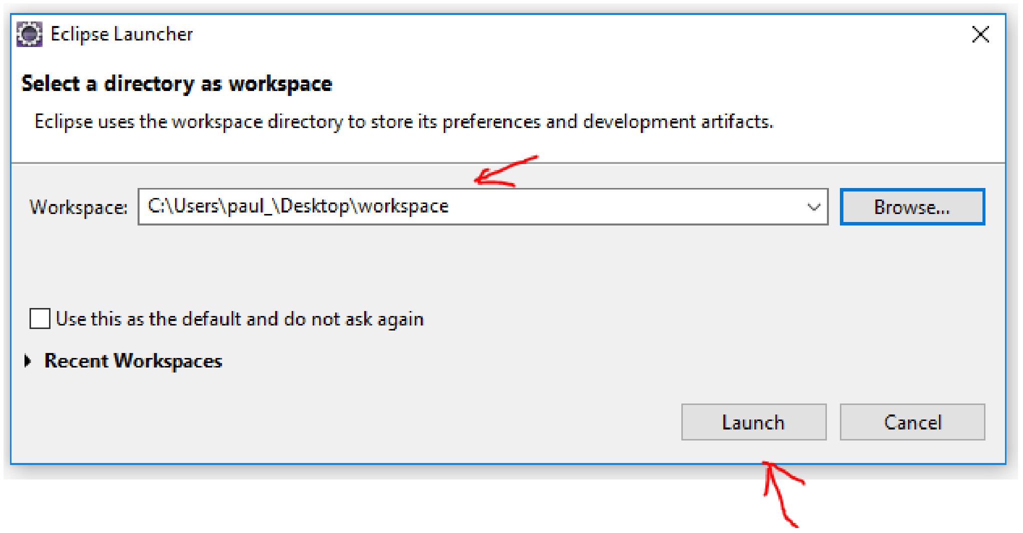
Eclipse. Download the latest version of Eclipse from here:
https://www.eclipse.org/downloads/

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.

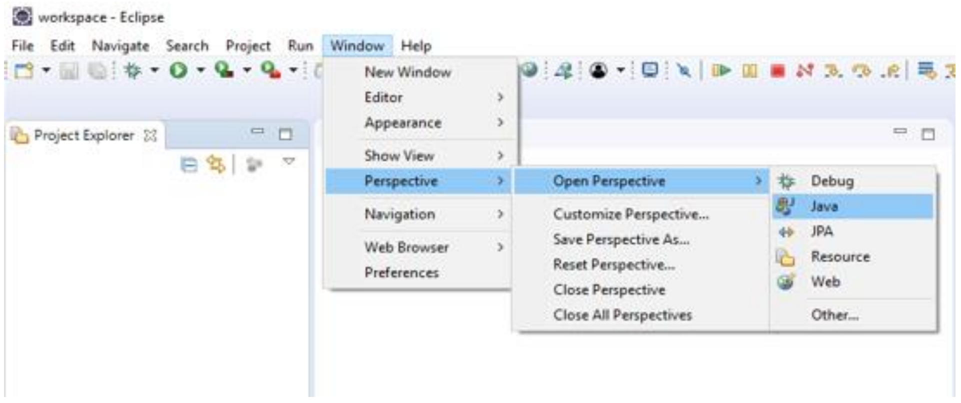
Configuring Eclipse.

1. Create a workspace. You can do this by creating a new folder on your Desktop called <u>workspace</u>. 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:
Browse...
Use this as the default and do not ask again
Recent Workspaces
Launch
Cancel

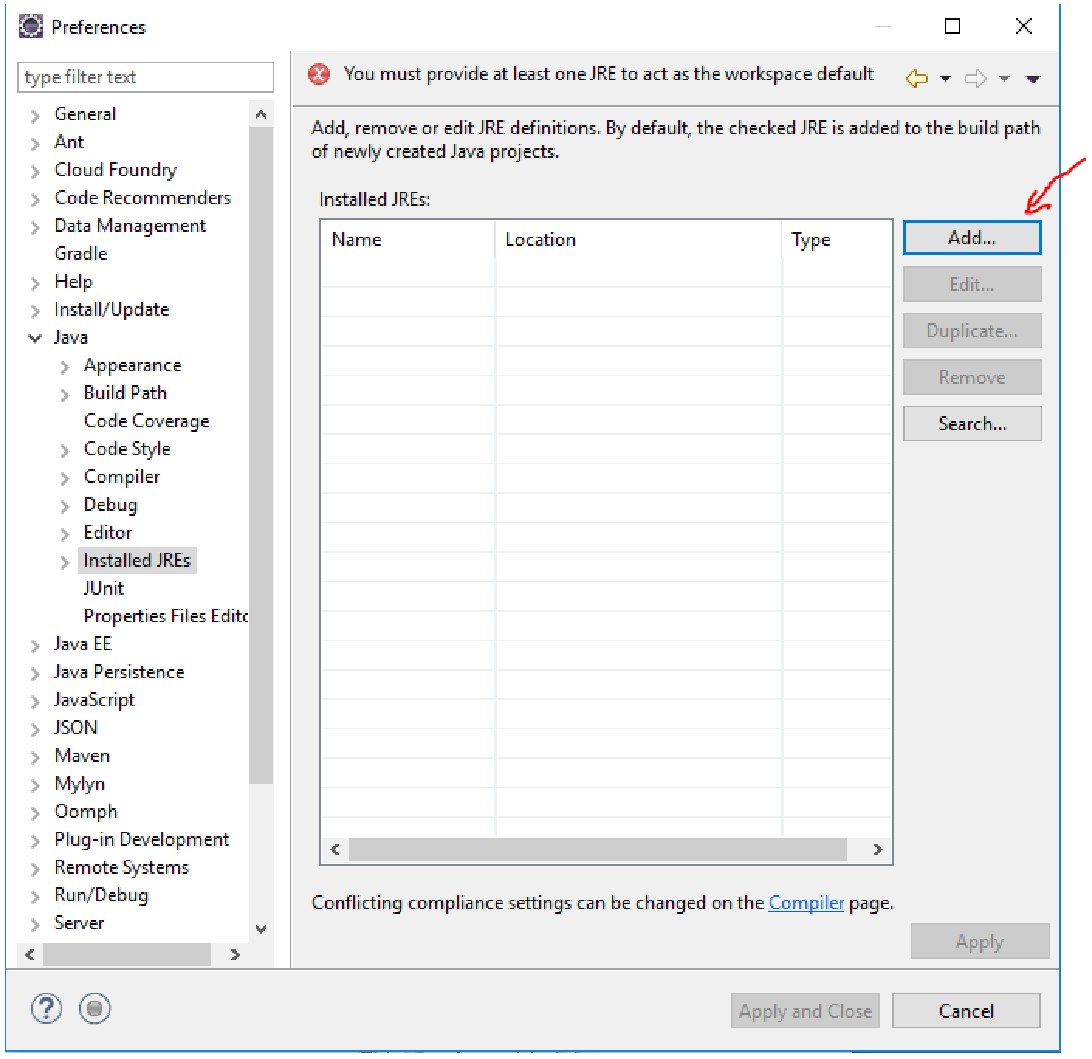

Eclipse Launcher
Select a directory as workspace
Eclipse uses the workspace directory to store its preferences and development artifacts.
Workspace:
C:\Users\paul_\Desktop\workspace
Browse...
Use this as the default and do not ask again
Recent Workspaces
Launch
Cancel

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

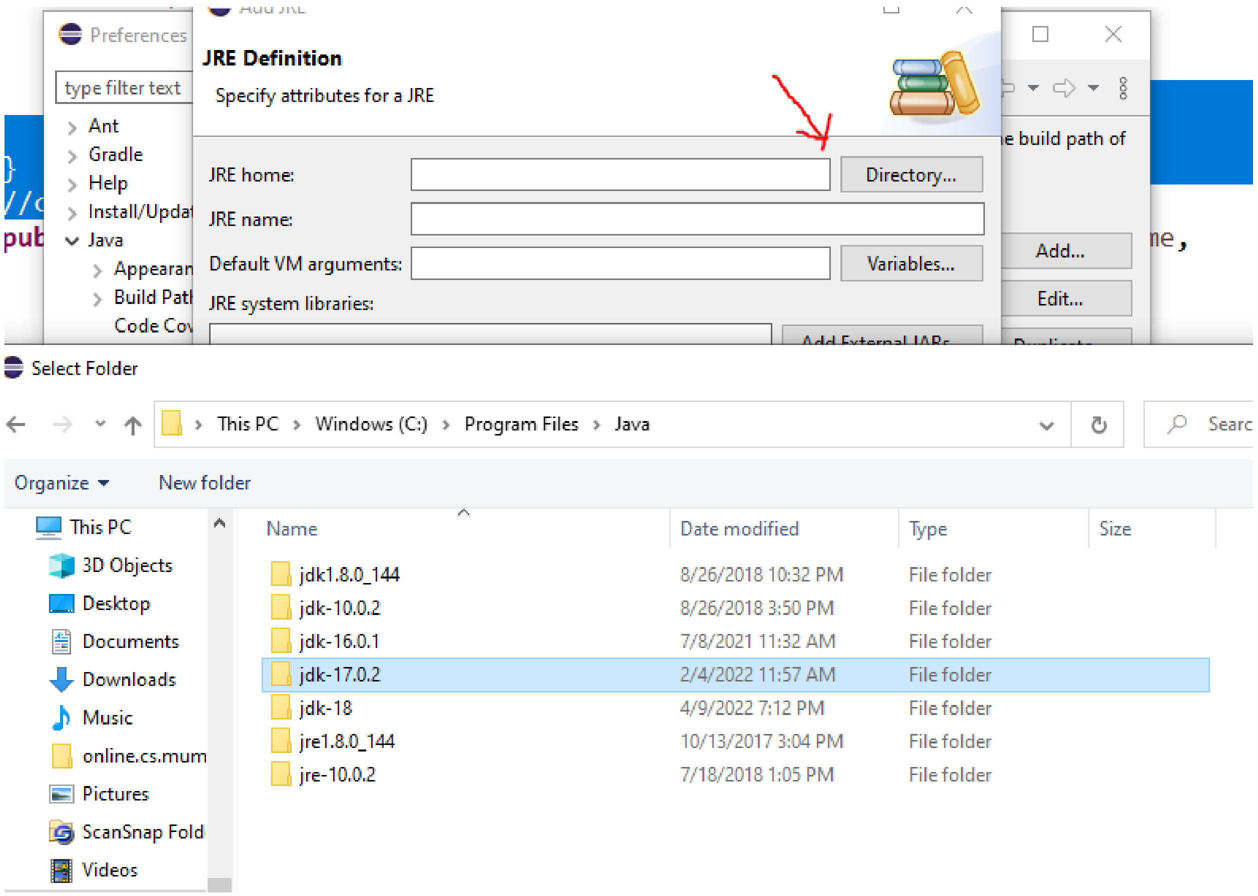
   b. If jdk-17 already appears in this window, be sure the box next to this java version is checked. Otherwise, if a jdk-17 version does not appear in this window, then click Add and navigate to this jdk distribution in your file system. Select Standard VM and click Next. Click the Directory button beside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

Conflicting compliance settings can be changed on the Compiler page.

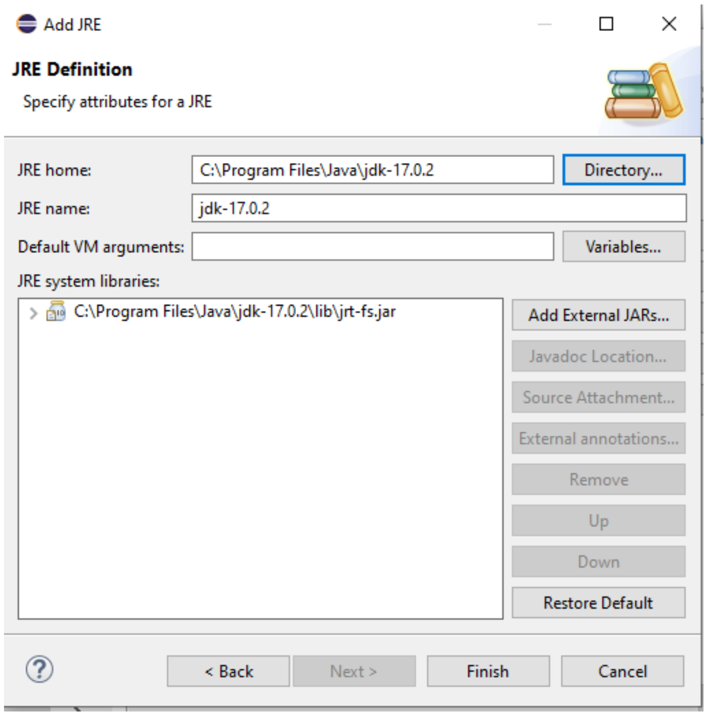


JRE home:
JRE name:
Default VM arguments:
JRE system libraries:

This PC › Windows (C:) › Program Files › Java

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.



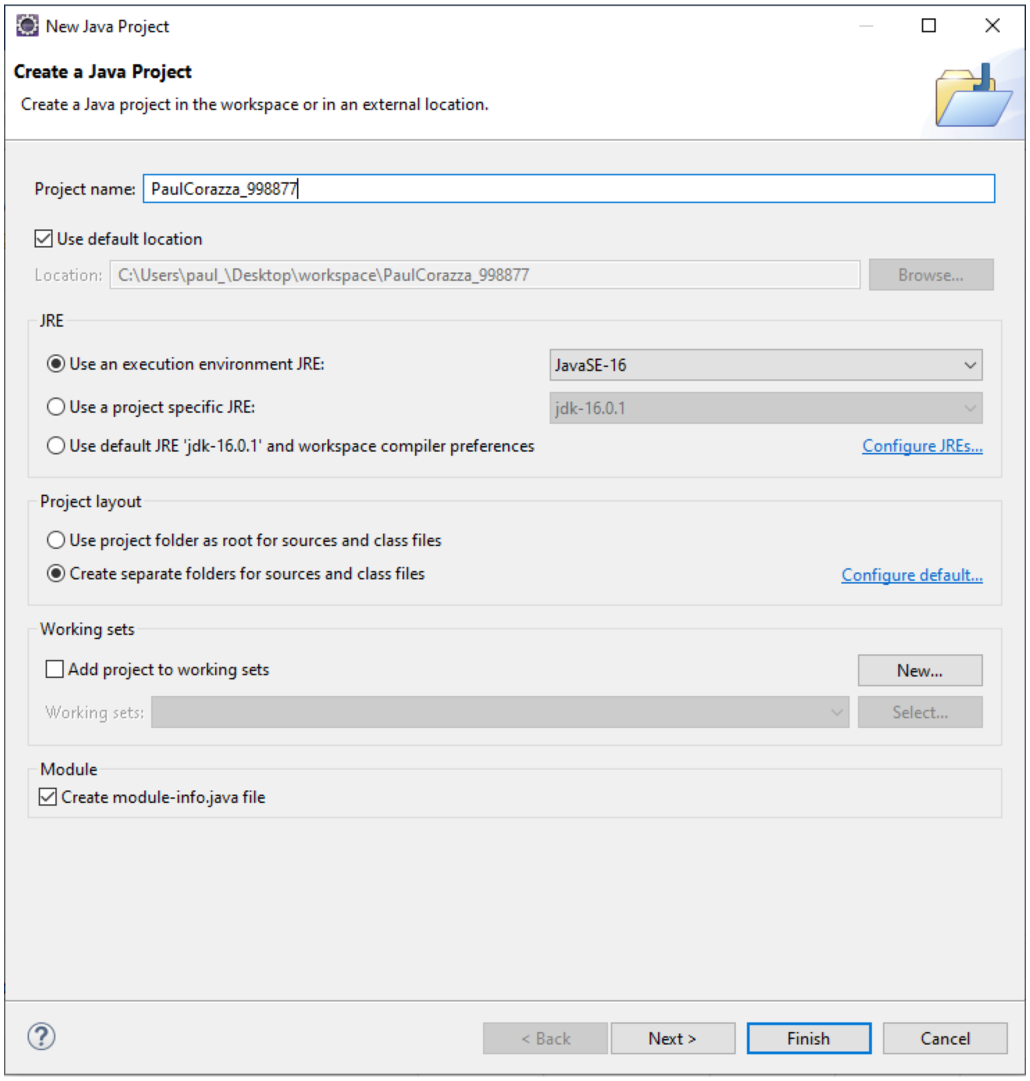
JRE home: C:\Program Files\Java\jdk-17.0.2
JRE name: jdk-17.0.2
Default VM arguments:
JRE system libraries:
C:\Program Files\Java\jdk-17.0.2\lib\jrt-fs.jar

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

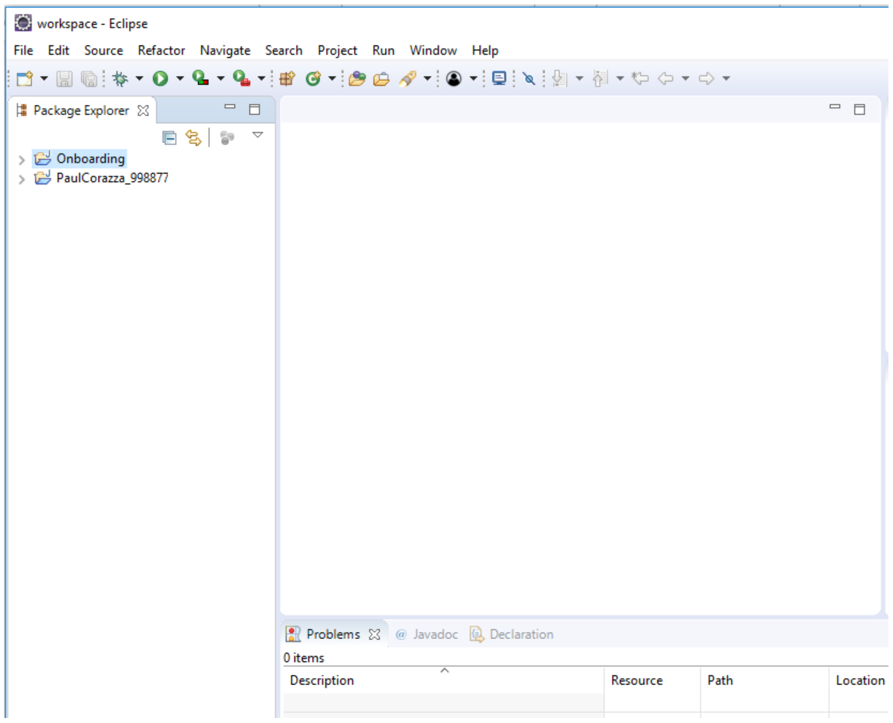
☐ Add project to working sets

Working sets:

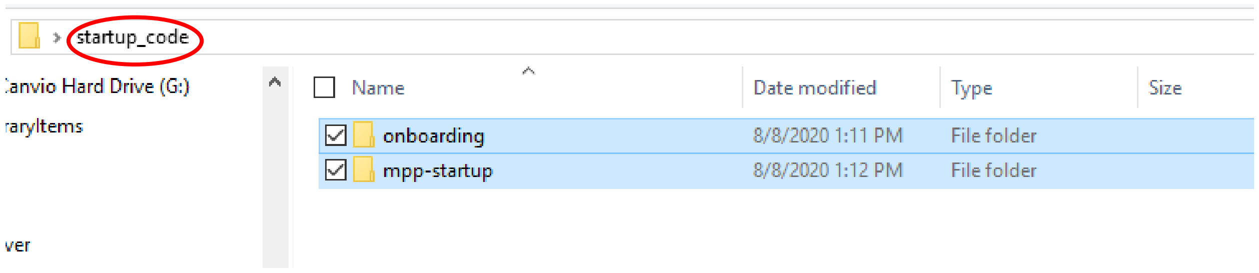
Module

☑ 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.

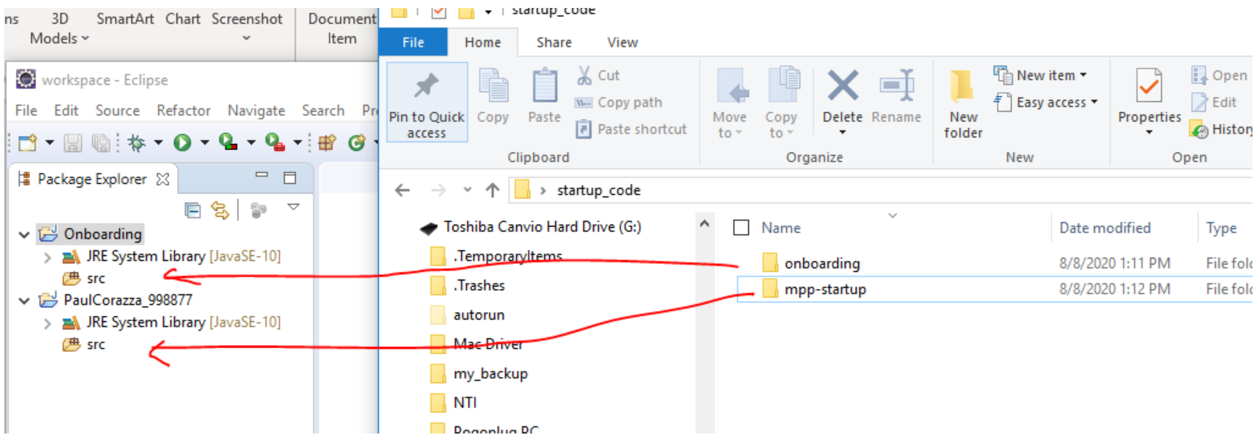


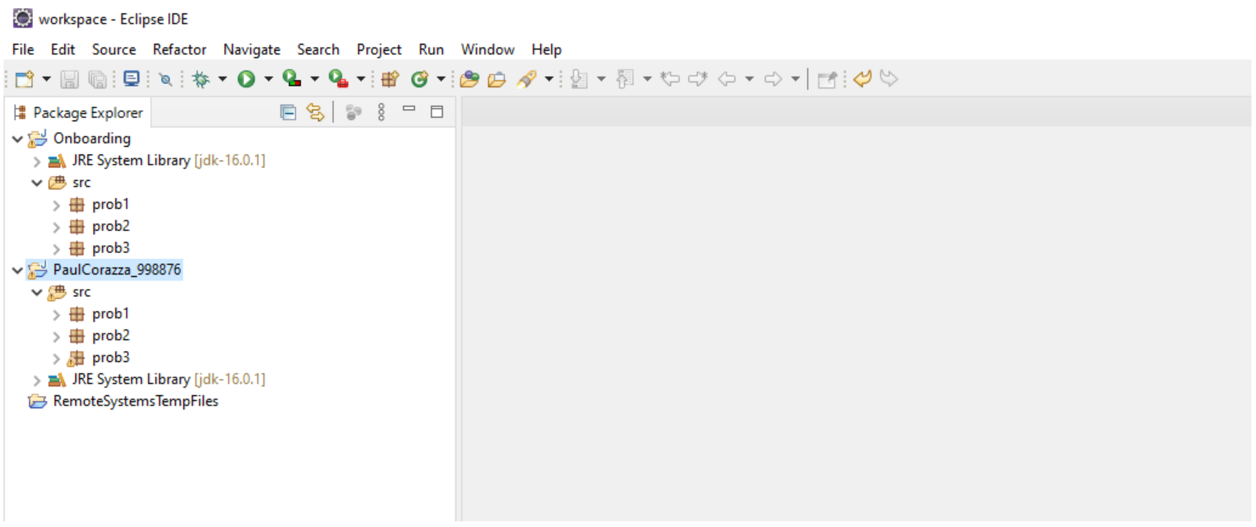
c. Create a folder startup_code in which to place the startup code that you retrieve from Sakai – you will find this code in a zipped folder as an attachment to the tests shown in Sakai. Unzip them and place them in start_up code folder. There will be startup code for the Onboarding test and also, later,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



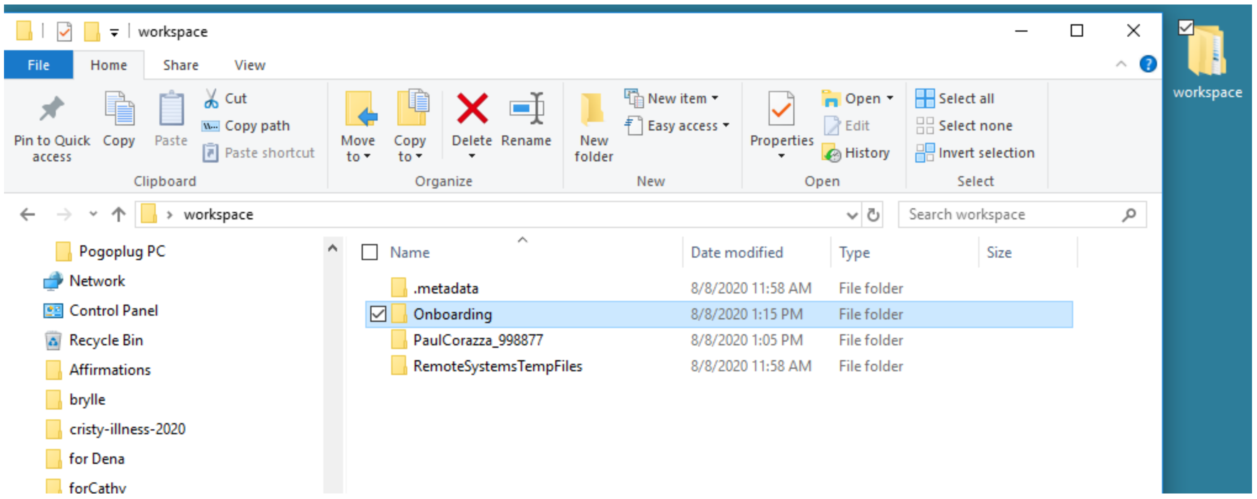

workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Onboarding
JRE System Library [jdk-16.0.1]
src
prob1
prob2
prob3
PaulCorazza_998876
src
prob1
prob2
prob3
JRE System Library [jdk-16.0.1]
RemoteSystemsTempFiles

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.

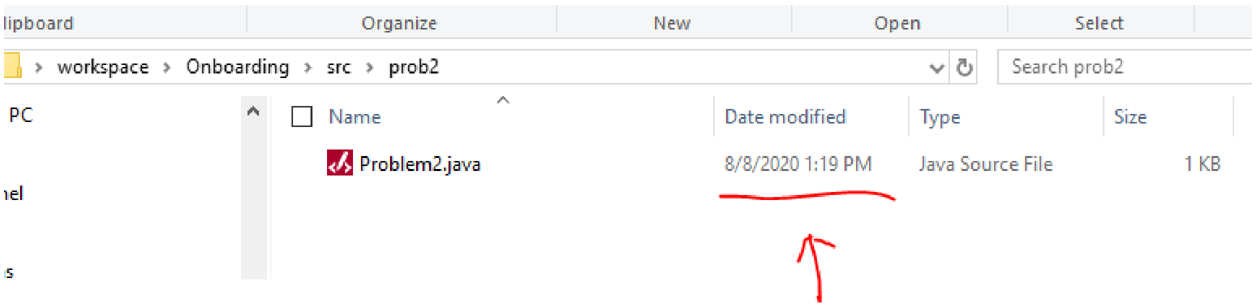
For each of the tests (onboarding and MPP pretests), you will follow the exam instrutions and write your code using the startup code as the starting point.

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.

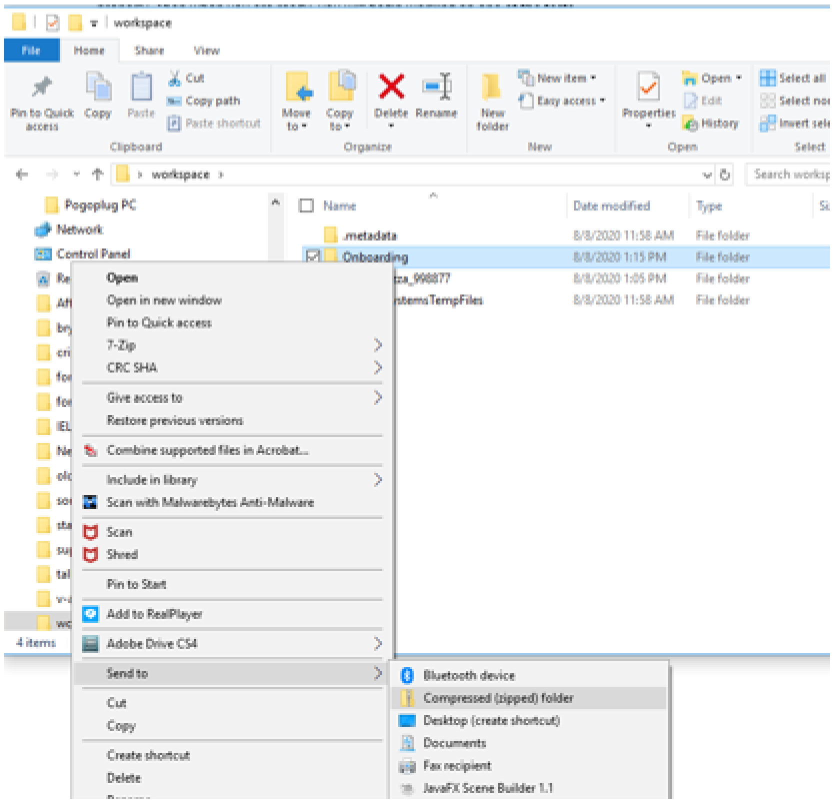


Name	Date modified	Type	Size
.metadata	8/8/2020 11:58 AM	File folder	
Onboarding	8/8/2020 1:15 PM	File folder	
PaulCorazza_998877	8/8/2020 1:05 PM	File folder	
RemoteSystemsTempFiles	8/8/2020 11:58 AM	File folder	

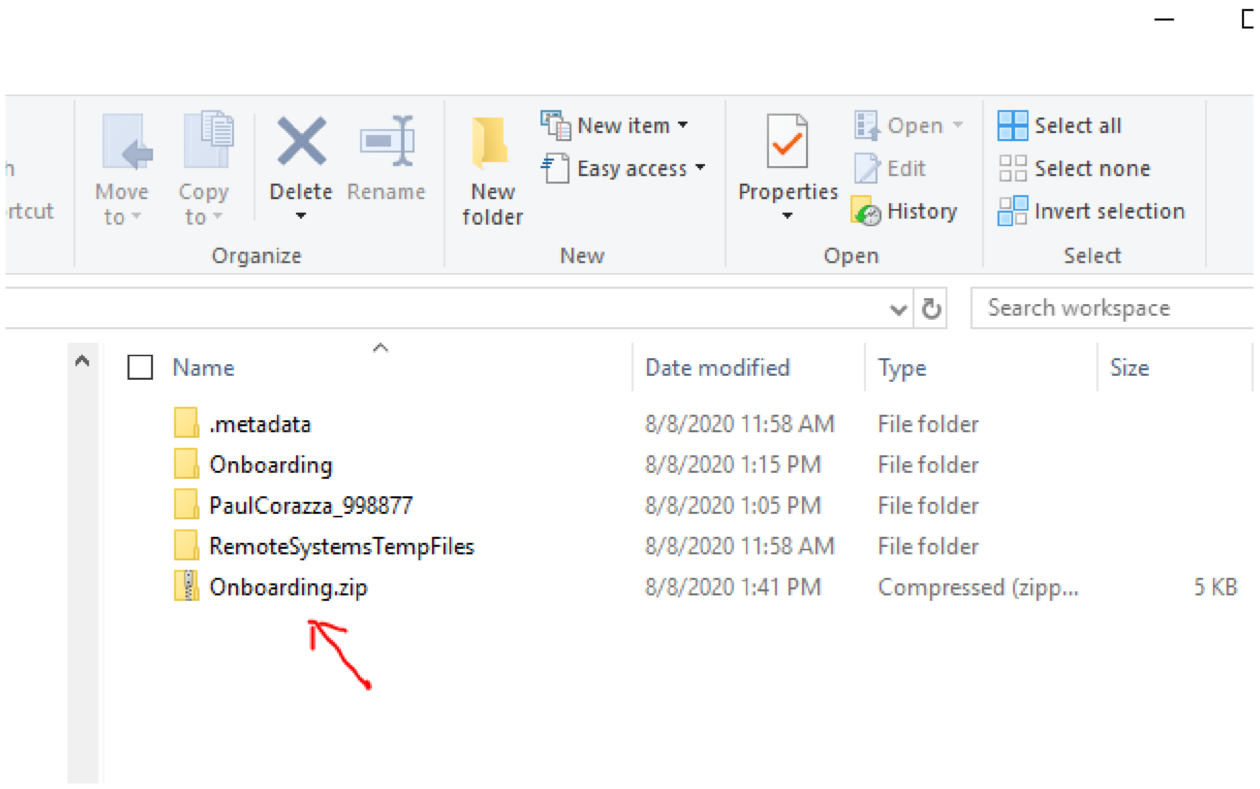


Name	Date modified	Type	Size
Problem2.java	8/8/2020 1:19 PM	Java Source File	1 KB

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



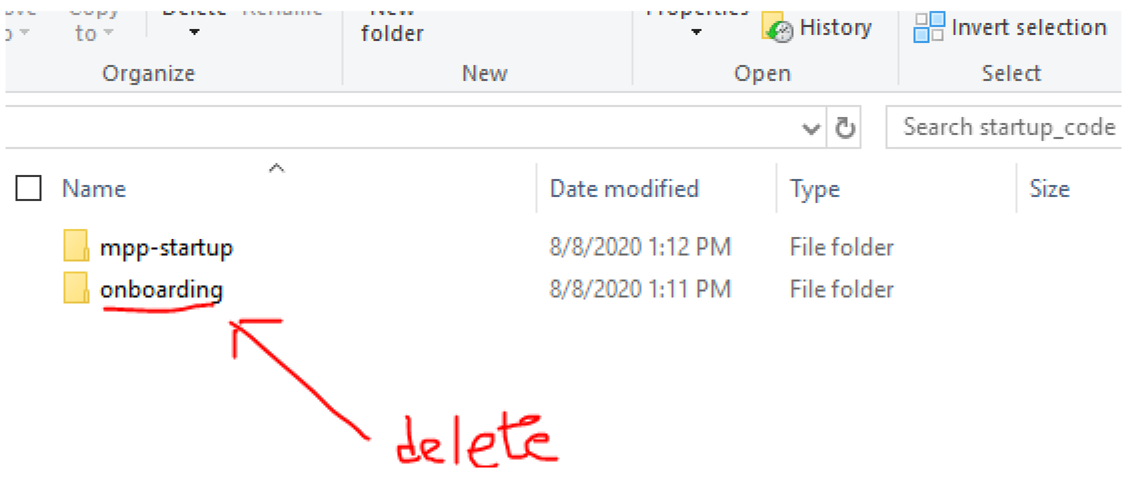
Name	Date modified	Type
.metadata	8/8/2020 11:58 AM	File folder
Onboarding	8/8/2020 1:15 PM	File folder
za_998877	8/8/2020 1:05 PM	File folder
stemsTempFiles	8/8/2020 11:58 AM	File folder



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

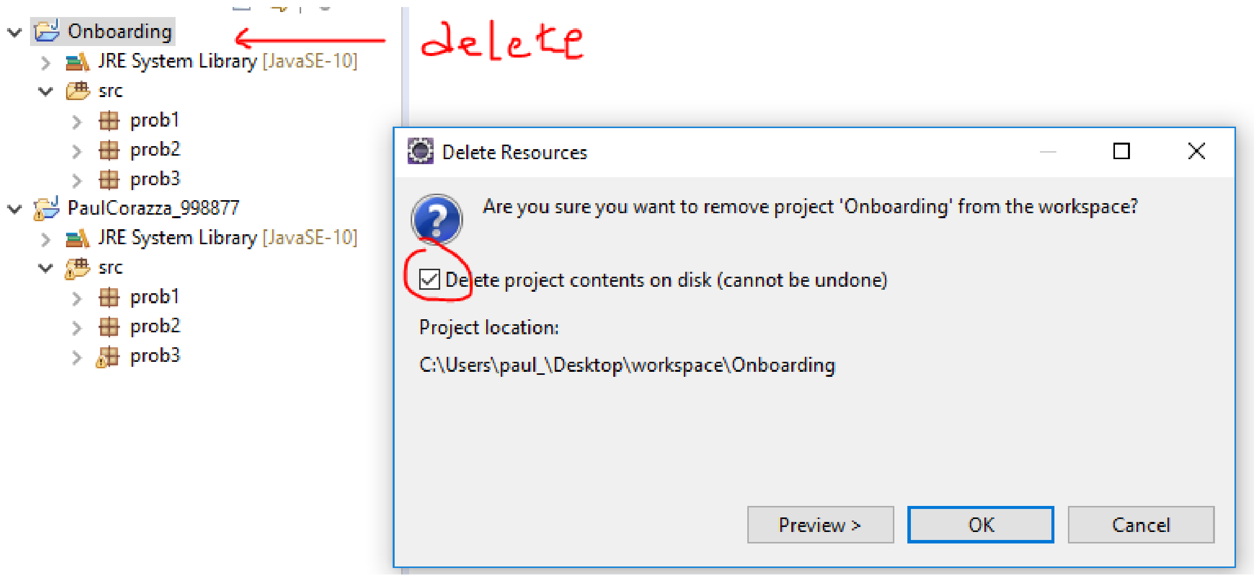
3. Submit your work by attaching this zip file in the Submit area in Sakai for this particular test. Once you have attached the file, remember to click the Submit button in Sakai.

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".



- Onboarding
  - JRE System Library [JavaSE-10]
  - src
    - prob1
    - prob2
    - prob3
- PaulCorazza_998877
  - JRE System Library [JavaSE-10]
  - src
    - prob1
    - prob2
    - prob3

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