

## MPP Pretest October, 2021

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 (other than access to ProctorTrack). 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 pass the Polymorphism problem (Problem 3) – get at least 4 out of 5 points on this one.
- B. Your total score needs to be 70% or higher. (Problems 1 and 2 are worth 10 points and Problem 3 is worth 5 points. To get 70% you must get 17.5 points.)

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]** The class `prob1.Problem1.java` has a `computeAverage` method that accepts a `HashMap scores` as input. The method is intended to compute the average of the test scores that are stored as values in the map. An example of the `scores` map is the following:

Key	Value
"John William"	"77.5"
"Annette Jones"	"91.0"
"Kevin Specker"	"88.8"

Example of the `scores` `HashMap`

The output of the `computeAverage` method working on this example would be (approximately) 85.77. Note that both keys and values in the map are `Strings`.

The `computeAverage` method in your startup code has already been implemented (and this implementation should not be modified), but it relies on the *unimplemented method* `computeSum`. Your task in this problem is to implement `computeSum` using recursion.

### *Requirements for Problem 1.*

1. You must use recursion in your implementation of `computeSum` (failure to use recursion will result in a score of 0 for this problem).

2. Outputs of your `computeSum` method must be correct for any given input (note that if the input `HashMap` is either `null` or empty, the return value should be 0).
3. Your code must not have compiler errors or produce any runtime exceptions.
4. You are allowed to make use of additional data structures in your implementation of `computeSum`.

*Hint:* To extract the `double` value from a `String`, use the static method `Double.parseDouble()`. For example, to extract 2.3 from the `String` "2.3", use the following code:

```
double val = Double.parseDouble("2.3")
```

**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];
```

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, within the limits of computer memory.
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).
7. 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 `Pentagon`, `EquilateralTriange`, and `Rectangle`. Each has a `computePerimeter` method, which returns the perimeter of the figure according to the usual mathematical formulas. In the `Control` class, a `computeSum` method is provided that will compute the sum of the perimeters of the geometric objects contained in the input array `shapes`. However, this `computeSum` method has been implemented by checking the type of each element of the input array as it is encountered and downcasting to the correct type, in order to compute perimeters.

Your task for this problem is to rewrite the `computeSum` method so that polymorphism is used. In order to carry out a polymorphic computation, you will need to make use of the `Polygon` interface that has been provided for you; `Polygon` will serve as a general type for all the geometric figures in this problem.

Your code must meet the following requirements:

1. You must implement polymorphism in your implementation of `computeSum`
2. The output of the new `computeSum` method that you create should be the same as the output of `computeSum` as it is given in the startup code
3. You must adjust the type of the input passed to `computeSum` and also the type of array that is created in the `main` method so that polymorphism can be used. (In the startup code, the type in each case is `Object[]`; the occurrences of the type `Object[]` must be changed accordingly.)
4. You must use the `Polygon` interface that has been provided. (Since that interface contains no methods, you will have to place a method in that interface yourself.) You are not allowed to use some other interface or superclass for the purpose of polymorphism; you *must* use the `Polygon` interface.
5. Your code should have no compiler errors and should not produce any runtime errors.

# 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-16 which can be obtained by following this link:

<https://www.oracle.com/java/technologies/javase-jdk16-downloads.html>

Use the Installer:

Windows x64 Installer	150.58 MB	 <a href="#">jdk-16.0.2_windows-x64_bin.exe</a>
-----------------------	-----------	--

**Eclipse.** Download Eclipse 2021-06; make use of the Installer. Go here:

<https://www.eclipse.org/downloads/>

**Troubleshooting.** When you attempt to run Eclipse, you may get an error message "Incompatible JVM". This can be solved by opening the text file eclipse.ini (in the folder where eclipse.exe is installed) and adding two lines:

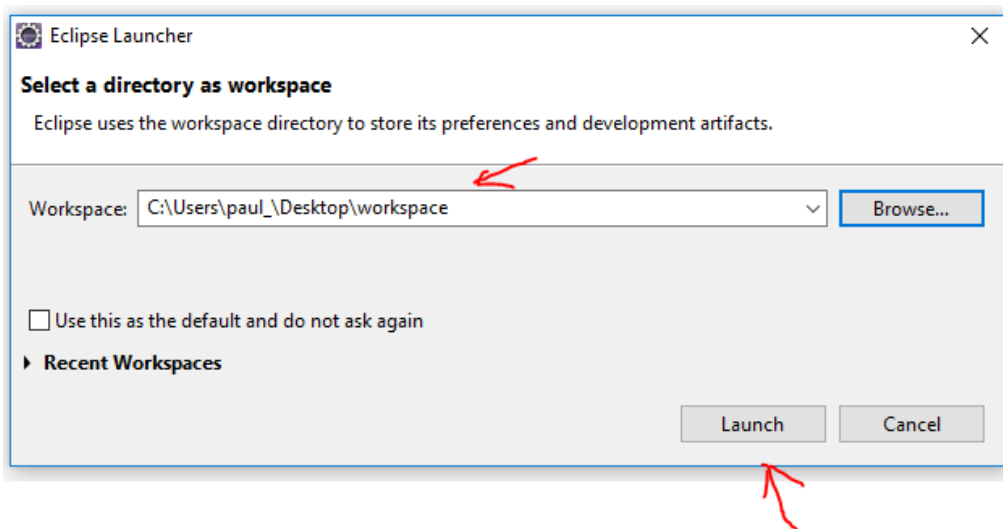
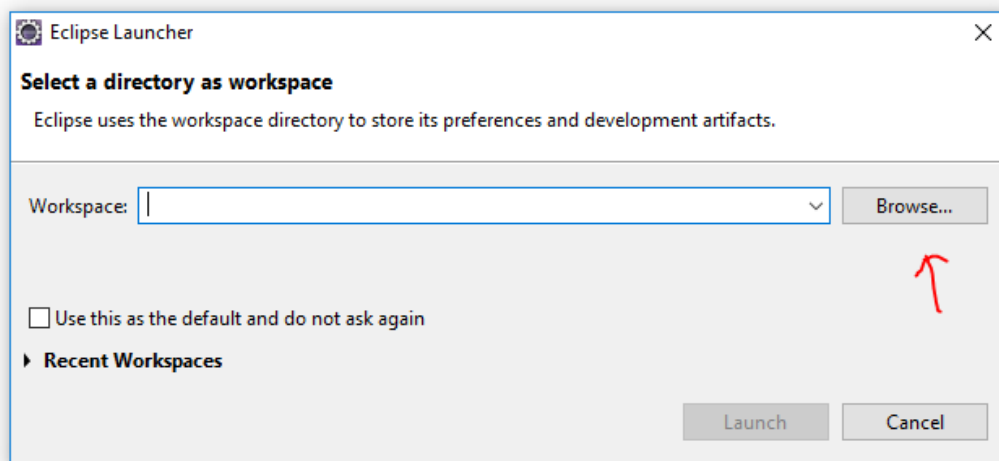
-vm

C:\Program Files\Java\jdk-16.0.1\bin\java.exe

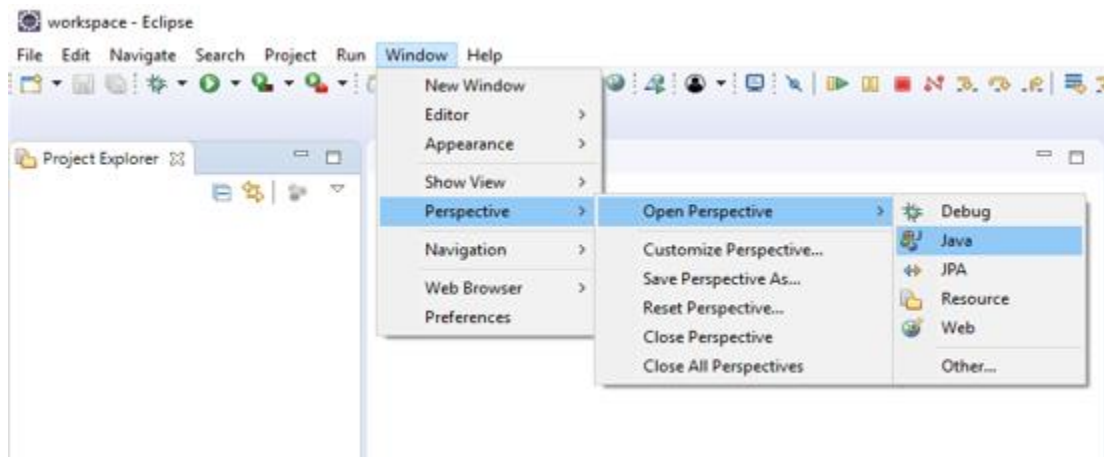
For more details about this, look at [https://wiki.eclipse.org/Eclipse.ini#Specifying\\_the\\_JVM](https://wiki.eclipse.org/Eclipse.ini#Specifying_the_JVM)

### 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.

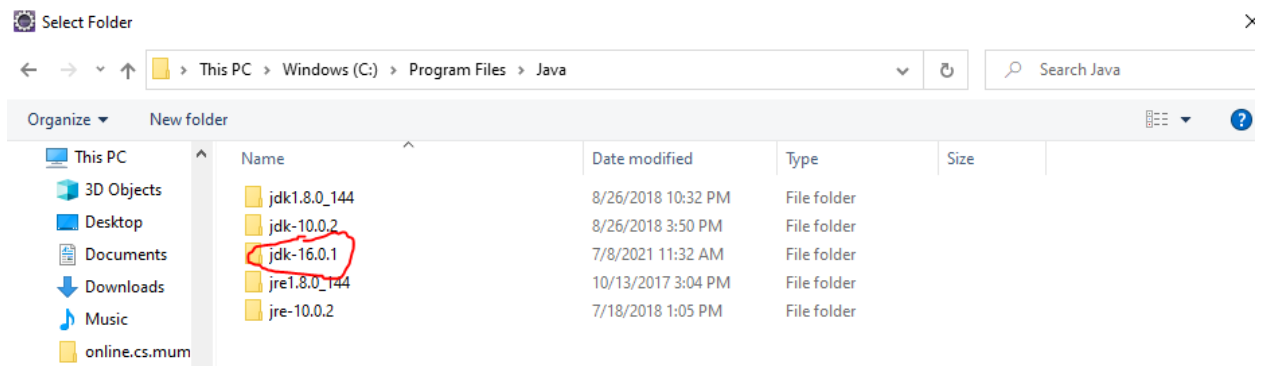
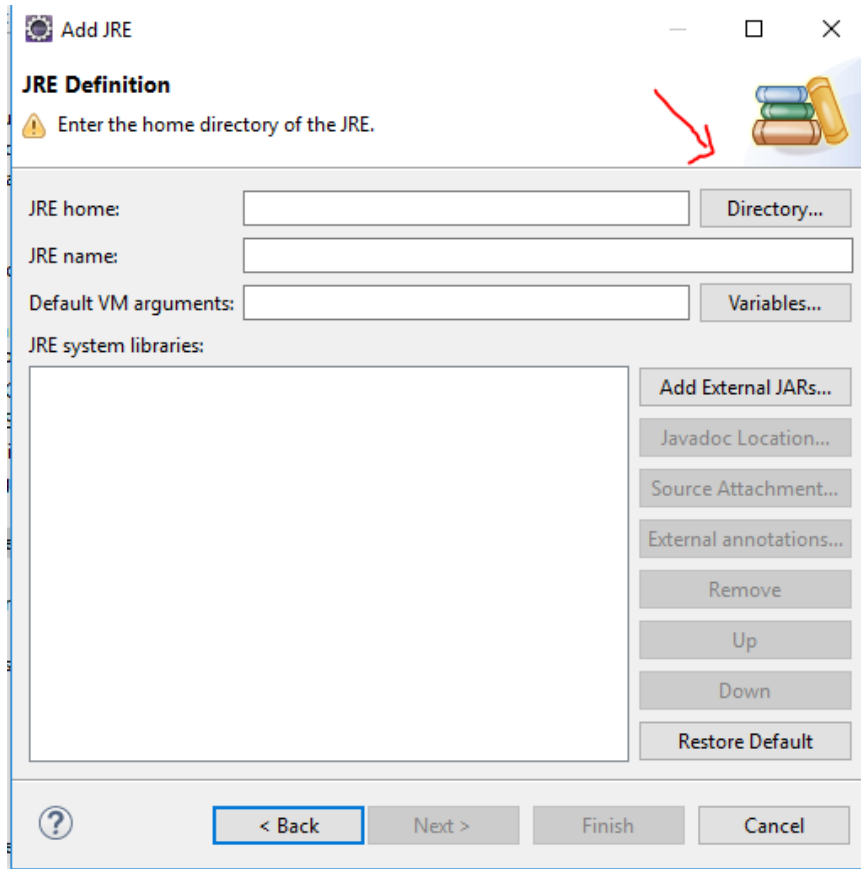


3. Close the Welcome tab and find Window along the top menu bar. Click Window > Perspective > Open Perspective > Java.



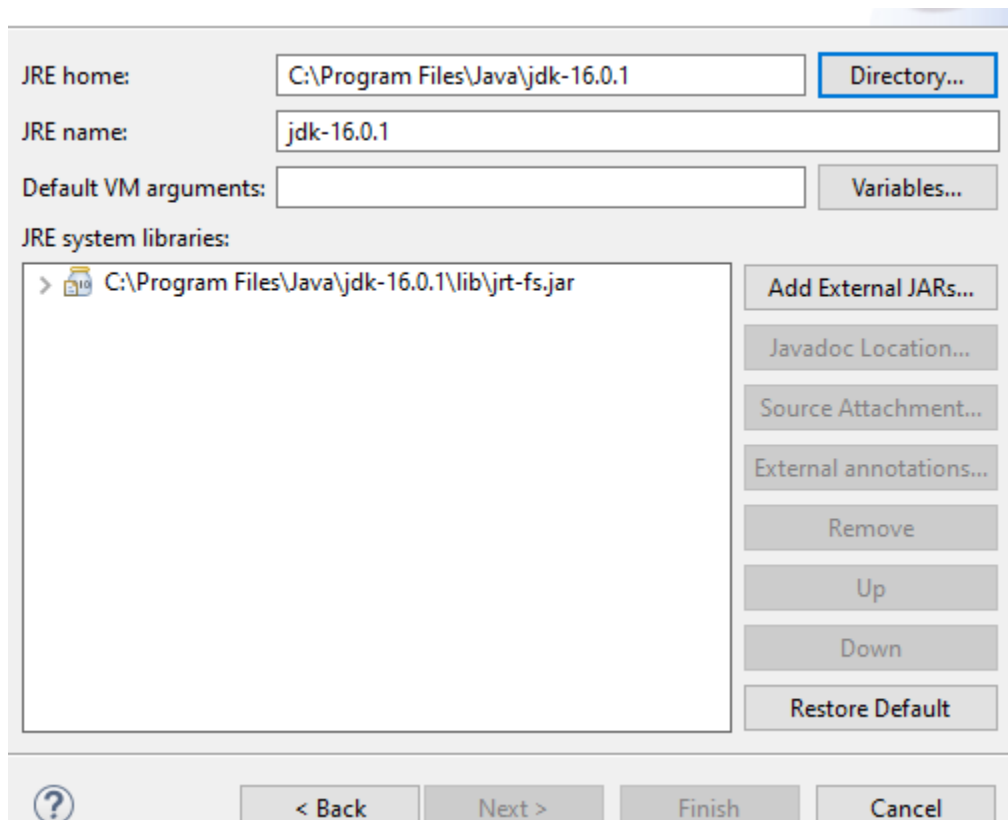
4. Point Eclipse to the jdk-16 distribution by doing the following.
  - a. Go to Window > Preferences > Java > Installed JREs
  - b. If jdk-16 already appears in this window, be sure the box next to this java version is checked. Otherwise, if the jdk-16 version has not been installed, then click Add 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-16.0.1, click OK and you will see the following window:





You must then check the box on the next window that asks you to specify jdk 16.0.1 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:

☒ Use default location

Location:

**JRE**

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE 'jdk-16.0.1' and workspace compiler preferences [Configure JREs...](#)

**Project layout**

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

**Working sets**

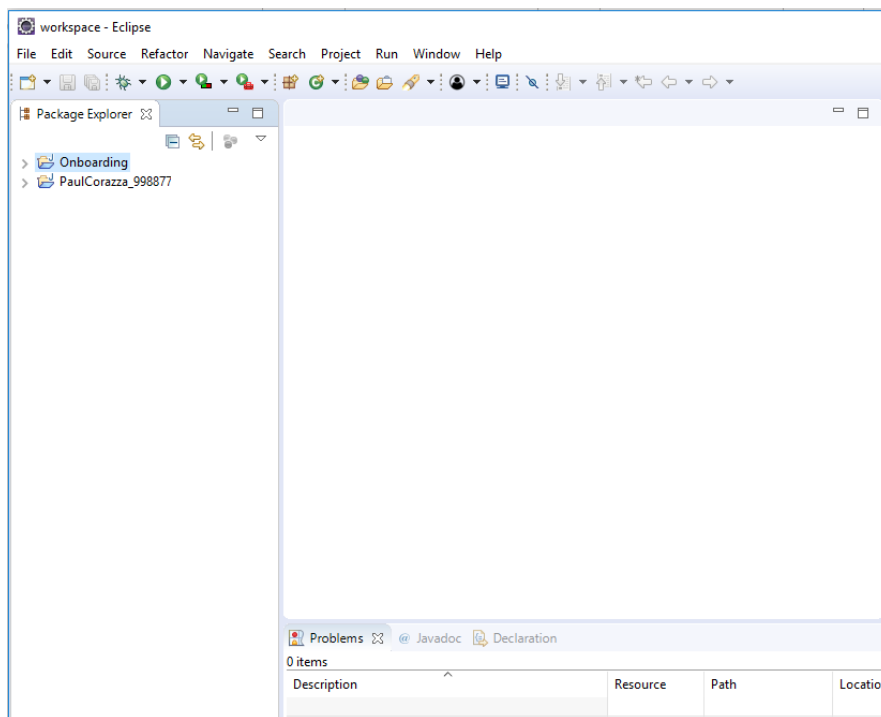
☐ Add project to working sets

Working sets:

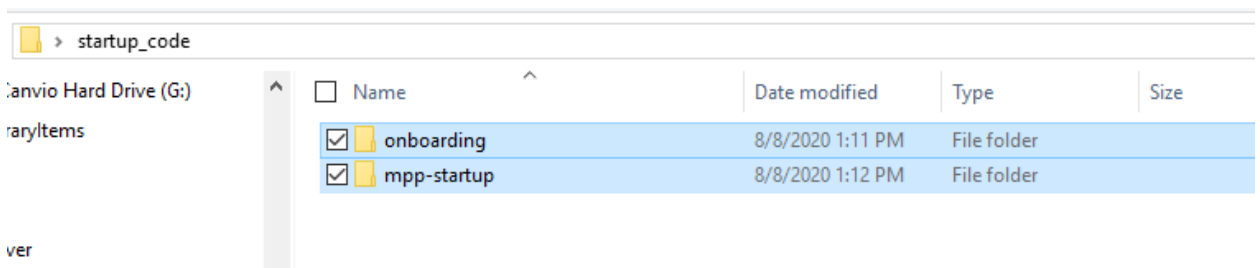
**Module**

☒ Create module-info.java file

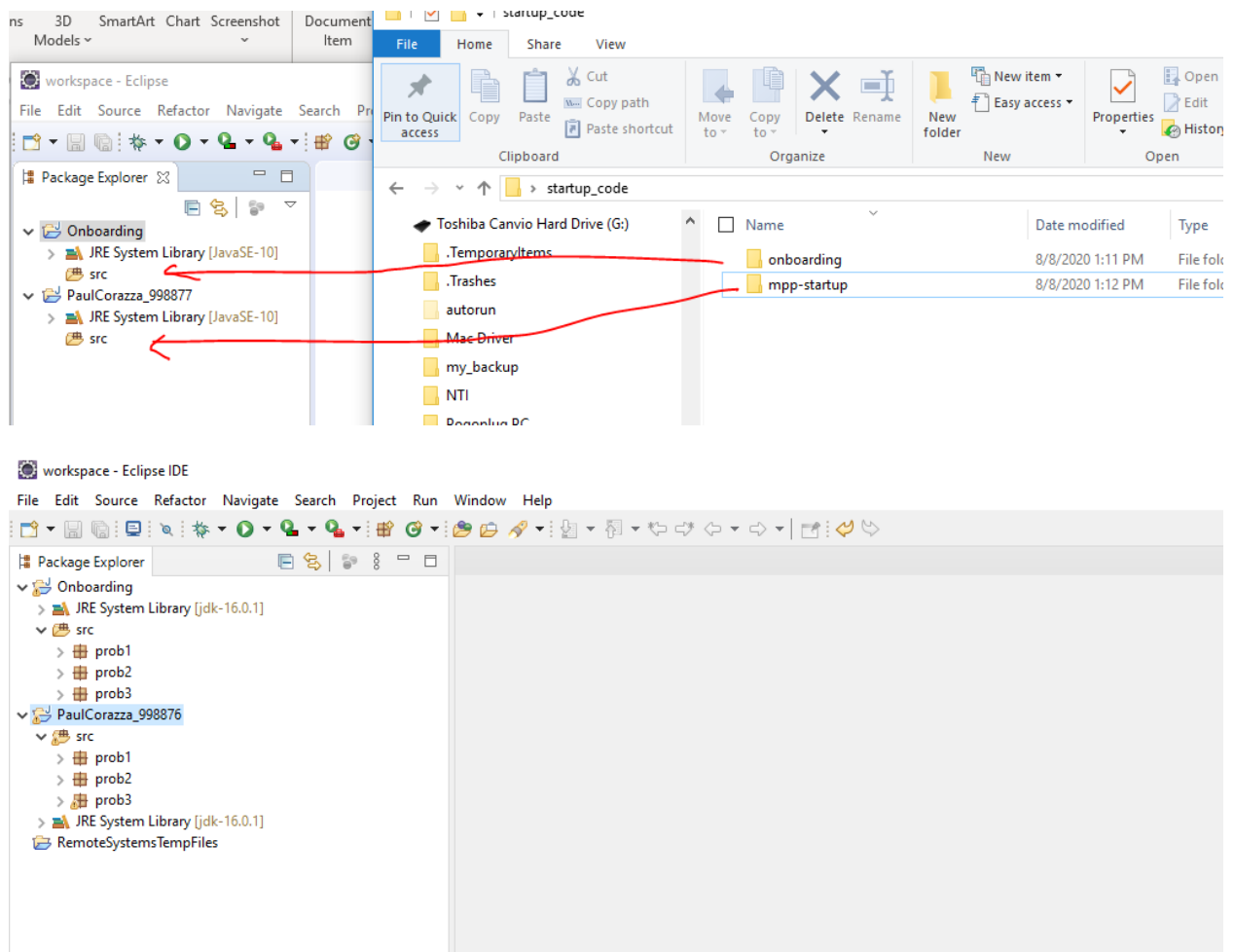
- b. *Onboarding Test.* Create a second project (as in the previous step). This time, name the project Onboarding (you do not need to include your ID for this one). REMEMBER: You will create the Onboarding project and work on it a couple of days before starting work on the actual MPP Pretest. Below is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

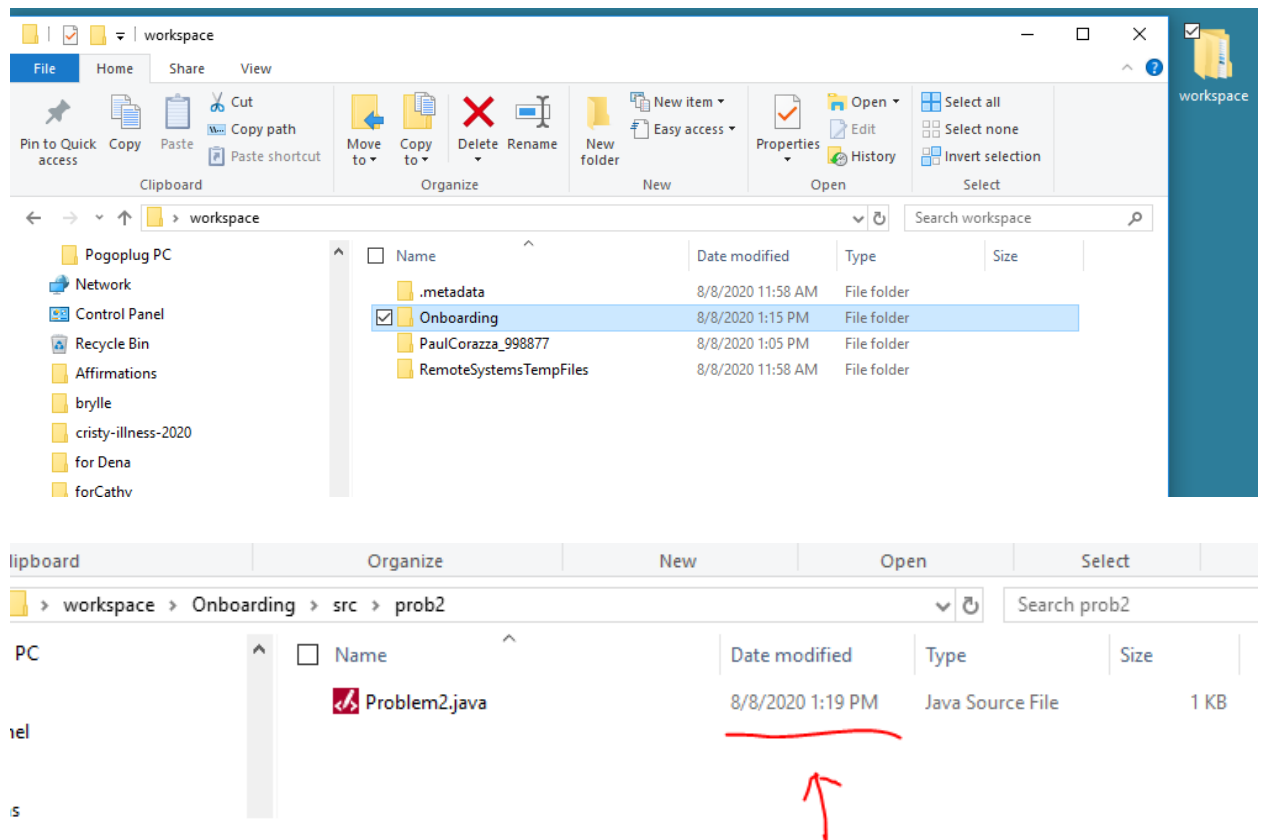


**Taking the Onboarding Test and the MPP Pretest.** You will receive instructions about accessing the exam instructions and startup code within the proctoring tool in a separate document. For both the onboarding and MPP pretest, you will take some steps to ensure that the proctoring tool is set up properly.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

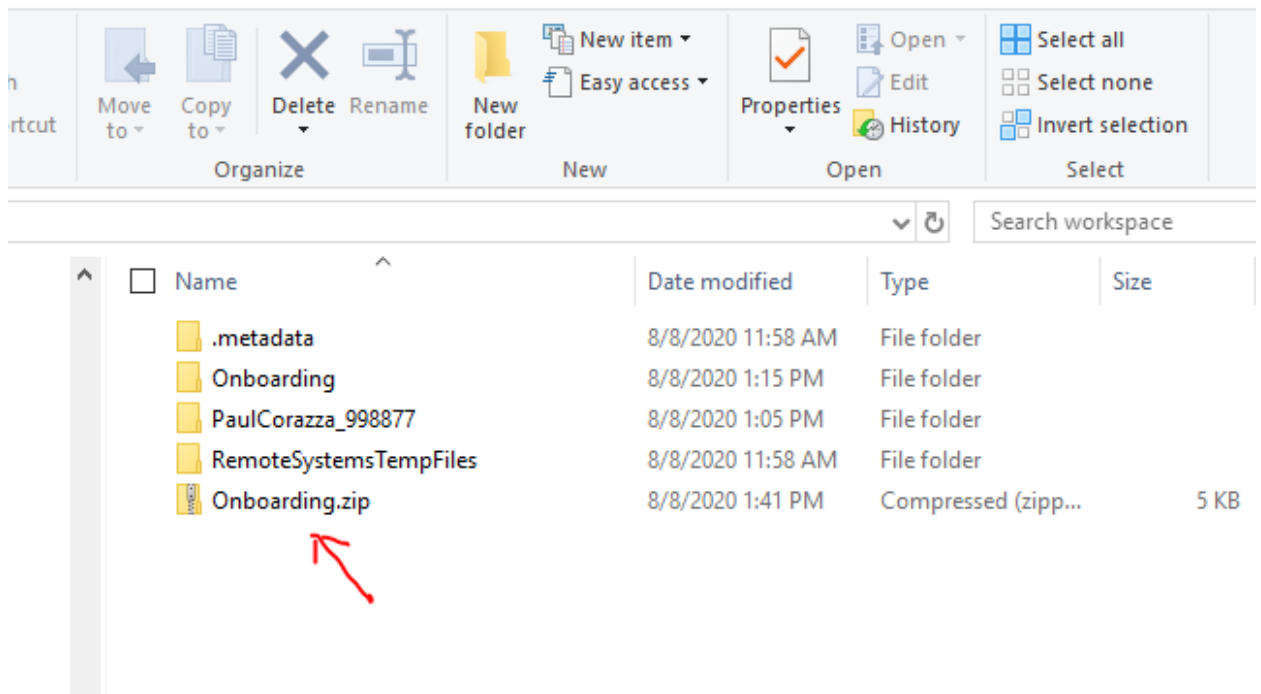
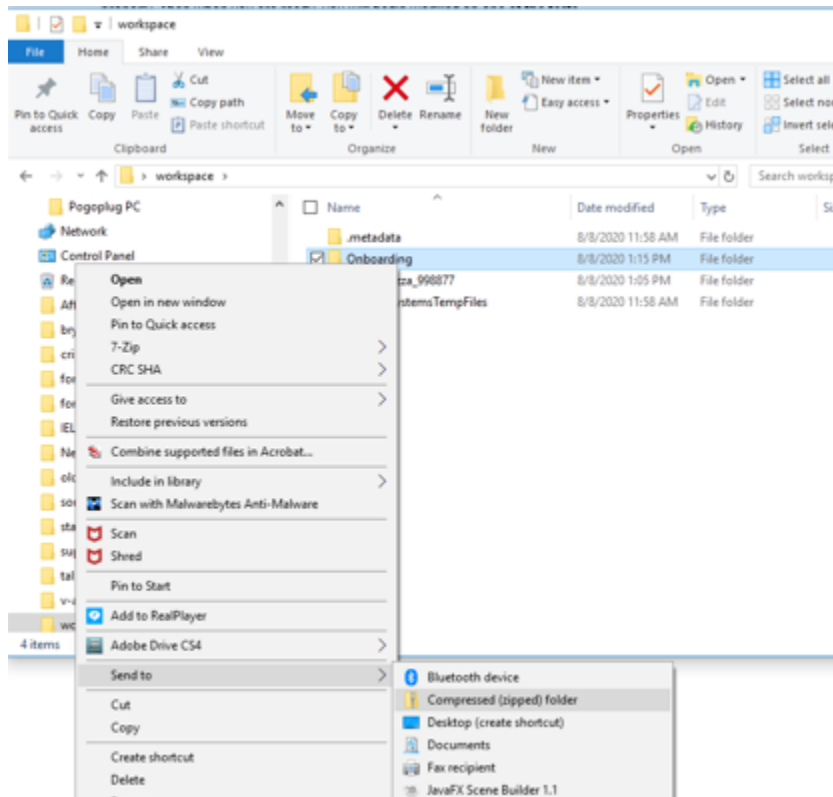
For each of the tests (onboarding and MPP pretests), you will follow the exam instructions and write your code using the startup code as the starting point.

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

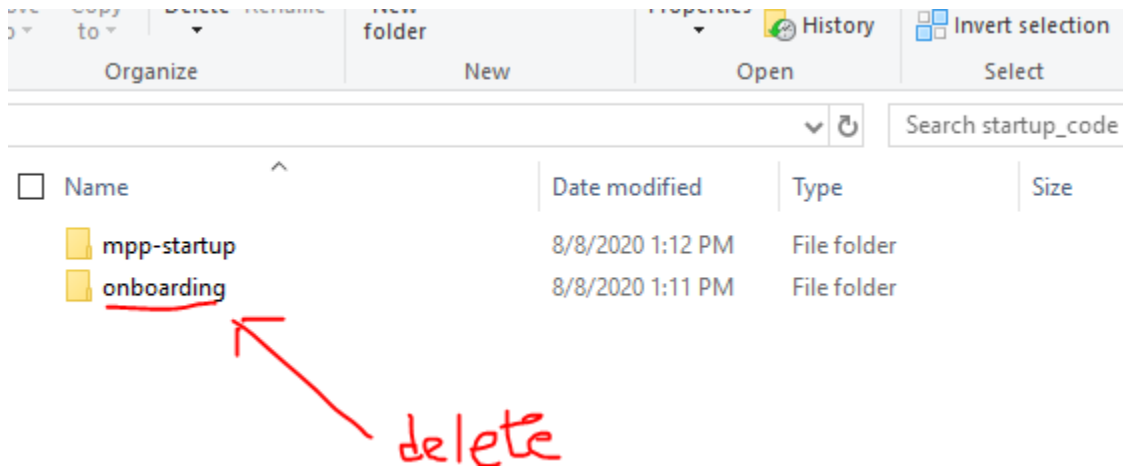
1. Make sure your work has been saved – do this by checking your workspace folder (on the Desktop) and checking the timestamp on the files inside the folder you are ready to submit.



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".

