

Programming Assignment 1:
CPU Process Scheduling System

Chelsea Jaculina

March 30, 2018

Professor Wu

CS 146

Table of Contents

System Overview	2
Implementation Design	3
Explanations of Functions	4
How to Run the Application	16
Problems Encountered	45
Lessons Learned	46

System Overview

The system that has been implemented replicates a dynamic scheduling software application. It has been developed to simulate a CPU process scheduling system by using the heap priority queue. The CPU scheduler will allow a user to dynamically keep track of process scheduling for the CPU by depending on the priority index. The system is built so that a process that has a bigger priority index will preempt (obtain the spot of another process) even if that process has been waiting a larger amount of time in the CPU.

Implementation Design

In order to separate all of the functions and keep track of where all the functions are implemented at, the CPU dynamic process scheduling system has been divided up into 3 different classes. The 3 classes include: Heap.java, Process.java, and MainMethod.java. All 3 classes and their functions are discussed in the next section. Most of each functions are static so that an additional Object didn't have to be created. Outputs of each implemented functions are also provided.

The initial design of the CPU process scheduling system was implemented by using arrays, but after several test cases the scheduling system became more efficient to implement with arraylists. For instance in the Heap.java class one of the functions called heapExtractMax, was easier to extract the maximum value from the list by using an arraylist rather than an array. Using an arraylist allowed to implement the code much simpler by just calling the arraylist method **remove()**. In addition, since the scheduling system allowed the user to insert a process, the size of the arraylist was much easier to increase.

Explanations of Functions

Heap.java

The Heap.java class extends the Process.java class so that the user can use the functions and retrieve the instance variables in the Process.java class.

maxheapify(ArrayList<Process> a, int i)

The maxHeapify() function is implemented so that it maintains the max-heap property. That is, the max-heap property assures that every node in the heap, other than the root the value of the its parents is largest. The largest element in the max-heap should be stored at the root. In Java, the root is stored at index 0. Calling this function by passing an arraylist of type Process and an integer i takes $O(\log n)$ running time.

```

Option Choice: 5
Enter an index number to heapify:
4
Process ID: P1          Priority Index of P1: 2080687002
Process ID: P2          Priority Index of P2: 1878219492
Process ID: P3          Priority Index of P3: 545032872
Process ID: P4          Priority Index of P4: 501262562
Process ID: P11         Priority Index of P11: 1824985829
Process ID: P6          Priority Index of P6: 306635355
Process ID: P7          Priority Index of P7: 682219367
Process ID: P8          Priority Index of P8: 1025217555
Process ID: P9          Priority Index of P9: 1349170960
Process ID: P10         Priority Index of P10: 1709802425
Process ID: P5          Priority Index of P5: 460548838
Process ID: P12         Priority Index of P12: 193404369
Process ID: P13         Priority Index of P13: 1319675311
Process ID: P14         Priority Index of P14: 1415158549
Process ID: P15         Priority Index of P15: 872999683
Process ID: P16         Priority Index of P16: 1940595691
Process ID: P17         Priority Index of P17: 165834107
Process ID: P18         Priority Index of P18: 1692885058
Process ID: P19         Priority Index of P19: 287494569
Process ID: P20         Priority Index of P20: 44845439

```

Option 5: MaxHeapify on Index 4 using buildMaxHeap() function

buildMaxHeap(ArrayList<Process> a)

The buildMaxHeap() function is implemented to produce a max heap from an unsorted input arraylist of type Process. It starts at the highest index of the arraylist size and decrements. The buildMaxHeap calls the maxHeapify function to help maintain the max heap property. Calling this function by passing an arraylist of type Process runs in linear time.

```

-----
Option Choice: 3
Build Max Heap
Process ID: P11      Priority Index of P11: 1930744616
Process ID: P10      Priority Index of P10: 1916950568
Process ID: P17      Priority Index of P17: 1785209605
Process ID: P9       Priority Index of P9: 1632960769
Process ID: P2       Priority Index of P2: 1315698744
Process ID: P15      Priority Index of P15: 1267092010
Process ID: P4       Priority Index of P4: 1457077086
Process ID: P14      Priority Index of P14: 1446509497
Process ID: P21      Priority Index of P21: 1527058378
Process ID: P5       Priority Index of P5: 988784029
Process ID: P6       Priority Index of P6: 203379529
Process ID: P13      Priority Index of P13: 163987768
Process ID: P7       Priority Index of P7: 279389092
Process ID: P1       Priority Index of P1: 1243116844
Process ID: P16      Priority Index of P16: 271475187
Process ID: P8       Priority Index of P8: 913025011
Process ID: P18      Priority Index of P18: 406993646
Process ID: P19      Priority Index of P19: 112969118
Process ID: P20      Priority Index of P20: 131567118
Process ID: P12      Priority Index of P12: 766991146
-----

```

Option 3: Build Max Heap by calling buildMaxHeap function

heapSort(ArrayList<Process> a)

The heapSort() function is implemented to sort an unsorted arraylist of type

Process by swapping the smaller elements with the largest element. It first starts by using the buildMaxHeap function to build a max-heap on the arraylist. The maximum element of the arraylist is stored at index 0 and exchanges with every index. Since we need to discard the index we decrement the heap size and then apply max heapify to apply the max heap property. This function sorts it by the priority index. The highest priority index is displayed at the last element of the arraylist. Calling this function sorts the arraylist in place and runs in $O(n \lg n)$ time.

```
-----
Option Choice: 4
Sorted List of Processes
Process ID: P19      Priority Index of P19: 35225104
Process ID: P5       Priority Index of P5: 90379877
Process ID: P2       Priority Index of P2: 387703544
Process ID: P1       Priority Index of P1: 391863106
Process ID: P3       Priority Index of P3: 442643609
Process ID: P14      Priority Index of P14: 740308680
Process ID: P4       Priority Index of P4: 807268900
Process ID: P10      Priority Index of P10: 809209526
Process ID: P20      Priority Index of P20: 858724531
Process ID: P13      Priority Index of P13: 1096525623
Process ID: P12      Priority Index of P12: 1166815590
Process ID: P9       Priority Index of P9: 1178176987
Process ID: P11      Priority Index of P11: 1535739520
Process ID: P18      Priority Index of P18: 1587346468
Process ID: P15      Priority Index of P15: 1639141965
Process ID: P6       Priority Index of P6: 1758448311
Process ID: P17      Priority Index of P17: 1876733374
Process ID: P16      Priority Index of P16: 1936787629
Process ID: P8       Priority Index of P8: 1965007460
Process ID: P7       Priority Index of P7: 2054082568
-----
MENU      Enter an Option Number
```

Option 4: Sorted List of Processes using heapSort() function

Priority Queue

The following functions are used to support a max-priority queue.

maxHeapInsert(ArrayList<Process> a, Process key)

The maxHeapInsert function inserts a key at the end of the arraylist. It uses the heapIncreaseKey function to maintain the max-heap property. In order to increase the size of the arraylist, we create a new Process object and stored the lowest integer value (Integer.MIN_VALUE). The running time of this function takes O(1).


```
-----
Option Choice: 1
Process Inserted - Process ID: P21      Priority Index of P21: 1527058378

----Updated List After Inserting Process---
Process ID: P1      Priority Index of P1: 1243116844
Process ID: P2      Priority Index of P2: 1315698744
Process ID: P3      Priority Index of P3: 1996725954
Process ID: P4      Priority Index of P4: 1457077086
Process ID: P5      Priority Index of P5: 988784029
Process ID: P6      Priority Index of P6: 203379529
Process ID: P7      Priority Index of P7: 279389092
Process ID: P8      Priority Index of P8: 913025011
Process ID: P9      Priority Index of P9: 1632960769
Process ID: P10     Priority Index of P10: 1916950568
Process ID: P11     Priority Index of P11: 1930744616
Process ID: P12     Priority Index of P12: 766991146
Process ID: P13     Priority Index of P13: 163987768
Process ID: P14     Priority Index of P14: 1446509497
Process ID: P15     Priority Index of P15: 1267092010
Process ID: P16     Priority Index of P16: 271475187
Process ID: P17     Priority Index of P17: 1785209605
Process ID: P18     Priority Index of P18: 406993646
Process ID: P19     Priority Index of P19: 112969118
Process ID: P20     Priority Index of P20: 131567118
Process ID: P21     Priority Index of P21: 1527058378
-----
```

Option 1: Insert a Process using maxHeapInsert() function

heapExtractMax(ArrayList<Process> a)

The heapExtractMax function extracts the maximum after it has been built to be a max heap. It first checks that there is atleast 1 element inside the arraylist. If there is at least 1 element in the arraylist, buildMaxHeap is called to create a max heap on the input Process type arraylist “a”. Swapping is involved to swap the highest value located at index 0 with the last element in the array. To extract the highest value, the arraylist uses the method **remove()**. The

function then calls maxHeapify to maintain the max heap property and returns the max value.

Performing this function takes $O(\lg n)$ time.

```
-----
Option Choice: 2
Extract Max Process
Extracted Process - Process ID: P7      Priority Index of P7: 2054082568

----Updated List After Extraction----
Process ID: P8      Priority Index of P8: 1965007460
Process ID: P15     Priority Index of P15: 1639141965
Process ID: P16     Priority Index of P16: 1936787629
Process ID: P12     Priority Index of P12: 1166815590
Process ID: P11     Priority Index of P11: 1535739520
Process ID: P18     Priority Index of P18: 1587346468
Process ID: P17     Priority Index of P17: 1876733374
Process ID: P20     Priority Index of P20: 858724531
Process ID: P13     Priority Index of P13: 1096525623
Process ID: P5      Priority Index of P5: 90379877
Process ID: P9      Priority Index of P9: 1178176987
Process ID: P14     Priority Index of P14: 740308680
Process ID: P2      Priority Index of P2: 387703544
Process ID: P4      Priority Index of P4: 807268900
Process ID: P6      Priority Index of P6: 1758448311
Process ID: P10     Priority Index of P10: 809209526
Process ID: P1      Priority Index of P1: 391863106
Process ID: P19     Priority Index of P19: 35225104
Process ID: P3      Priority Index of P3: 442643609
-----
```

Option 2: Extracted Max Process using heapExtractMax() function

heapIncreaseKey(ArrayList<Process> a, int i, Process key)

The heapIncreaseKey function increase the priority of an element by a certain key value. The index i in in the arraylist checks the priority queue key that we want to increase. The procedure checks the key of priority index i in the arraylist and updates it to its new value (key). The key must be greater than the priority index that is originally there otherwise an error message will occur. The new key value might violate the max-heap property, so swapping must

be involved. Invoking this function takes $O(\lg n)$ running time, since the worst case is going the distance of the length of the root.

heapMaximum(ArrayList <Process> a)

The heapMaximum functions returns the largest element of the input arraylist with the largest key value. In a max-heap, the largest value is at the root at index 0. Calling this function takes $O(1)$ running time.

Option Choice: 7	
Heap Maximum	
Process ID: P20	Priority Index of P20: 44845439
Process ID: P17	Priority Index of P17: 165834107
Process ID: P12	Priority Index of P12: 193404369
Process ID: P19	Priority Index of P19: 287494569
Process ID: P6	Priority Index of P6: 306635355
Process ID: P5	Priority Index of P5: 460548838
Process ID: P4	Priority Index of P4: 501262562
Process ID: P3	Priority Index of P3: 545032872
Process ID: P7	Priority Index of P7: 682219367
Process ID: P15	Priority Index of P15: 872999683
Process ID: P8	Priority Index of P8: 1025217555
Process ID: P13	Priority Index of P13: 1319675311
Process ID: P9	Priority Index of P9: 1349170960
Process ID: P14	Priority Index of P14: 1415158549
Process ID: P18	Priority Index of P18: 1692885058
Process ID: P10	Priority Index of P10: 1709802425
Process ID: P11	Priority Index of P11: 1824985829
Process ID: P2	Priority Index of P2: 1878219492
Process ID: P16	Priority Index of P16: 1940595691
Process ID: P1	Priority Index of P1: 2080687002

Option 7: Heap Maxium. Gets the Roots

Helper Functions

The following functions are helper functions that were implemented to simplify the CPU scheduler code.

leftChild(int i)

The function returns the left child node. Though the pseudocode states the left child = $2 * i$, we need to change the equation to $2 * i + 1$ since Java is 0 based index.

rightChild(int i)

The function returns the right child node. Though the pseudocode states the left child = $2 * i + 1$, we need to change the equation to $2 * i + 2$ since Java is 0 based index.

swap(ArrayList<Process> data, int i, int j)

The swap function swaps two integers in an arraylist. Though implemented and easy to use the swap function was not used in any of the functions so that more practice could be done by swapping indices on arraylists.

getHeapSize(int h)

This functions gets the heap size of the input arraylist and sets it.

setHeapSize(ArrayList<Process> a)

This functions gets the heap size of the input arraylist and sets it to the instance variable.

printArray()

This functions prints out the arraylist with the process id and priority index by using the getter methods `getProcessId()` and `getPriorityIndex()` in `Process.java`

```
/*
 * PRINTS THE ARRAYLIST
 * Public function that prints out the arraylist
 * To print out the process id and priority index
 */
public static void printArray(ArrayList<Process> arr)
{
    for(Process n : arr)
    {
        System.out.println("Process ID: " + n.getProcessId() +
"\t\t" + "Priority Index of " + n.getProcessId() + ": " +
n.getPriorityIndex());
    }
}
```

Process.java

Process()

This is a no-argument constructor to construct a process in the CPU scheduling system.

Process(String processName, int pIndex)

This is a 2-argument constructor to construct a process. It passes in a process name and priority index.

getProcessId()

This is a getter method to get the process id number. The process id number is concatenated with the character "P."

getPriorityIndex()

This is a getter method to get the priority index number. This priority index number is randomly generated and distinct from others. To get the numbers distinct, a random generator was created and planted a seed of the largest maximum integer value (Integer.MAX_VALUE).

MainMethod.java

main()

The function is the main method in which it executes the all of the functions that are implemented in Process.java and Heap.java. It creates an empty arraylist and fills the arraylist with at least 20 processes. It then calls the displayMenu() function to display the menu so that the user can interact with the system.

displayMenu(ArrayList<Process> list)

The function displayMenu() includes a menu list consisting of several options using print statements. The user is allowed to choose one of the options listed on the menu interface. The options are displayed using the sop() function and include terminating the program, inserting a process, extract max process, build max heap, sort processes, max heapify, increase key, heap maximum, and viewing the current list of processes in sorted order.

In addition, a do-while loop is used to display the options to the user until the user enters “0.” order to display the menu to user.


```
-----  
                        MENU - Enter an Option Number  
-----  
0 - Terminate  
1 - Insert a Process  
2 - Extract Max Process  
3 - Build Max Heap  
4 - Sort Processes  
5 - Max Heapify an Index  
6 - Increase Key  
7 - Heap Maximum  
8 - View Current List of Processes  
-----  
Option Choice:
```

sop(Object x)

This shortcut function implements the input/output print statement

System.out.println. It is used to make the main method have a cleaner feel. Whatever is passed as the Object “x” will be printed out.

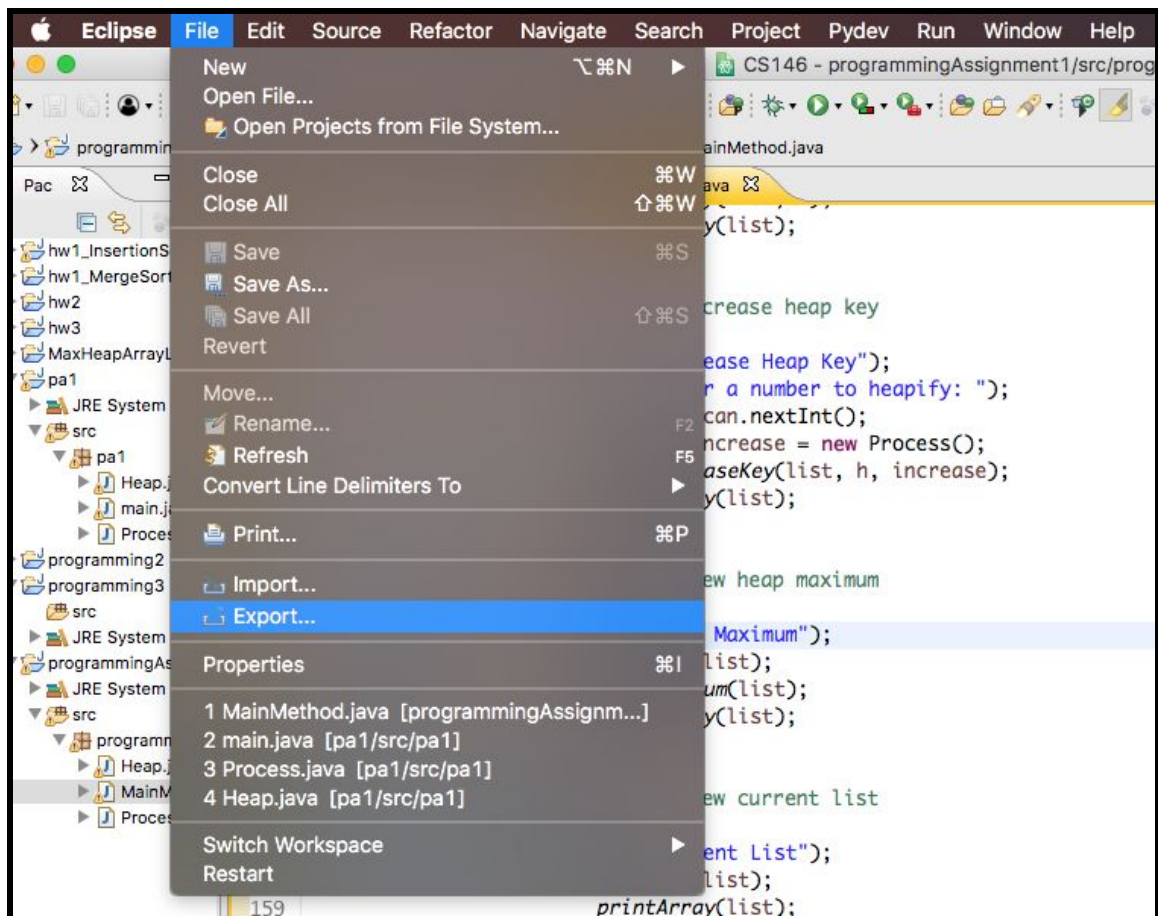
```
/*  
 * Private shortcut function that implements the input/output  
System.out.println  
 */  
private static void sop(Object x)  
{  
    System.out.println(x);  
}
```


How To Run the Application

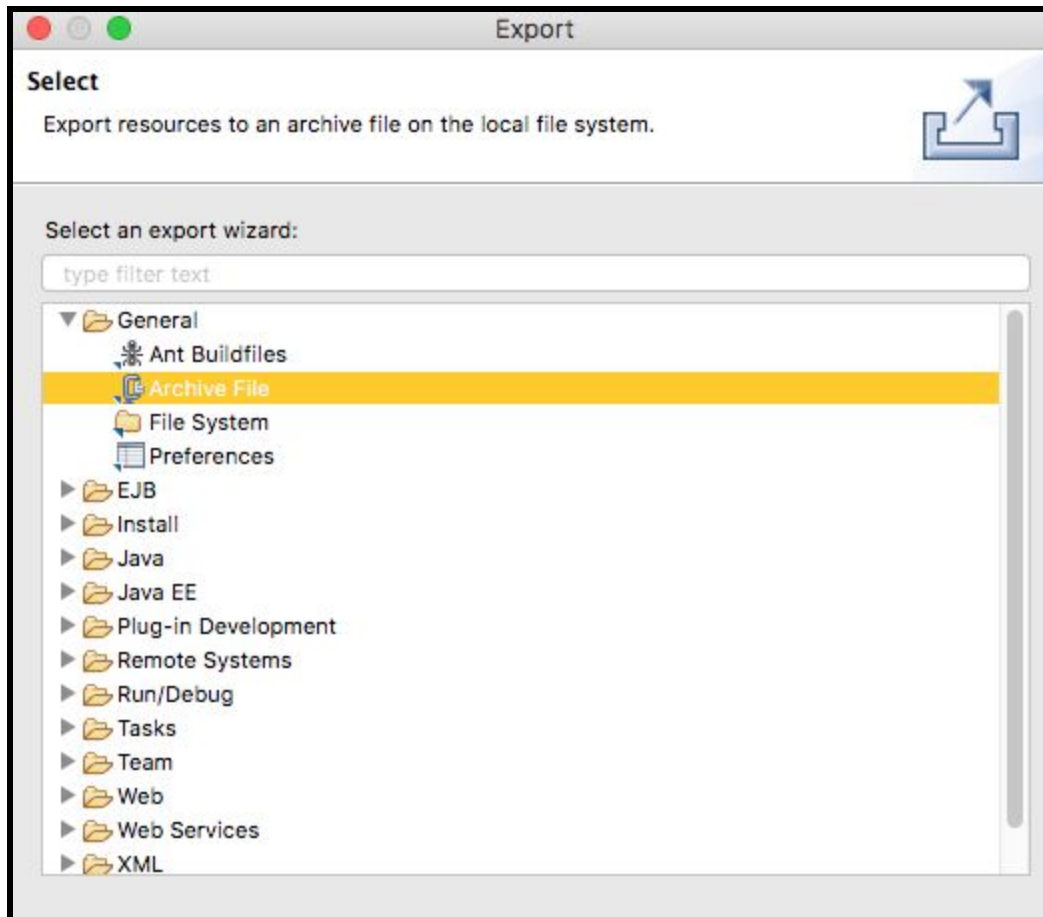
Step 1 Export Files

Zip File

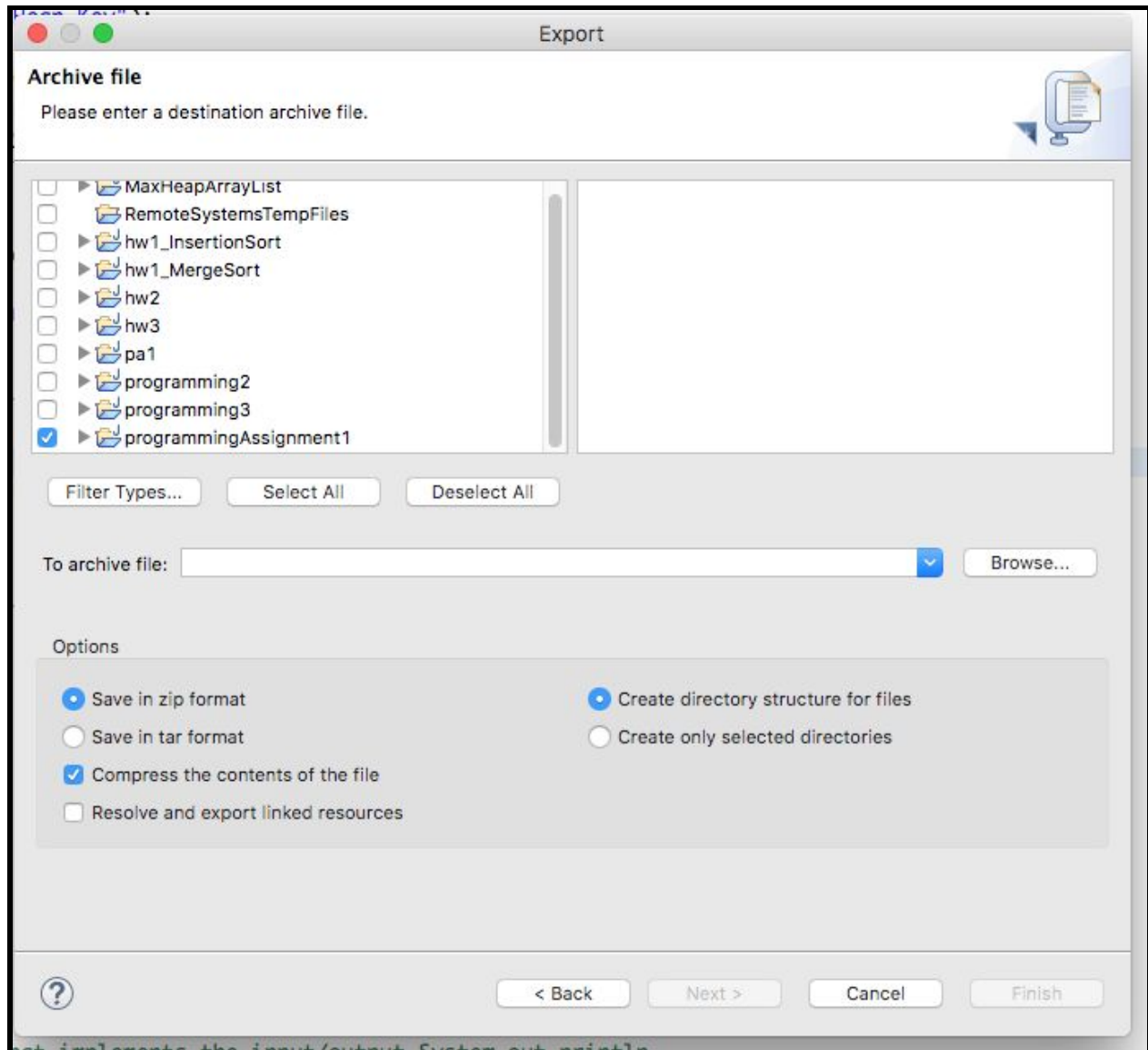
1. File → Export



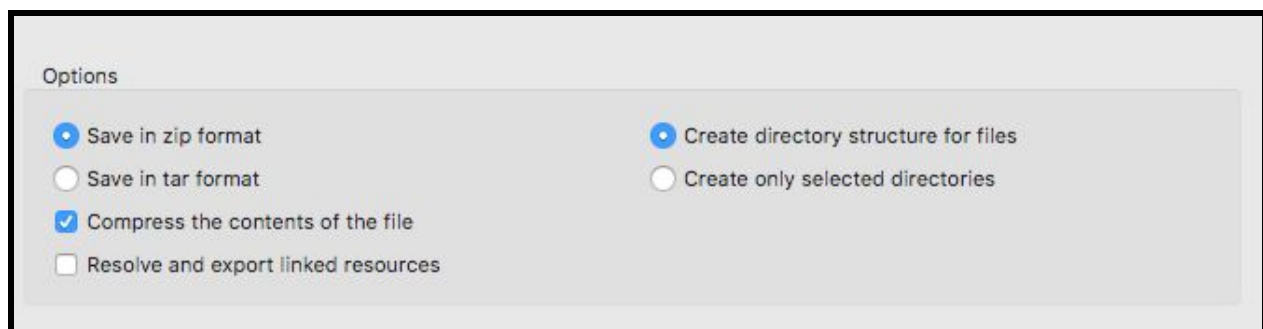
Select the General → Archive File export Wizard



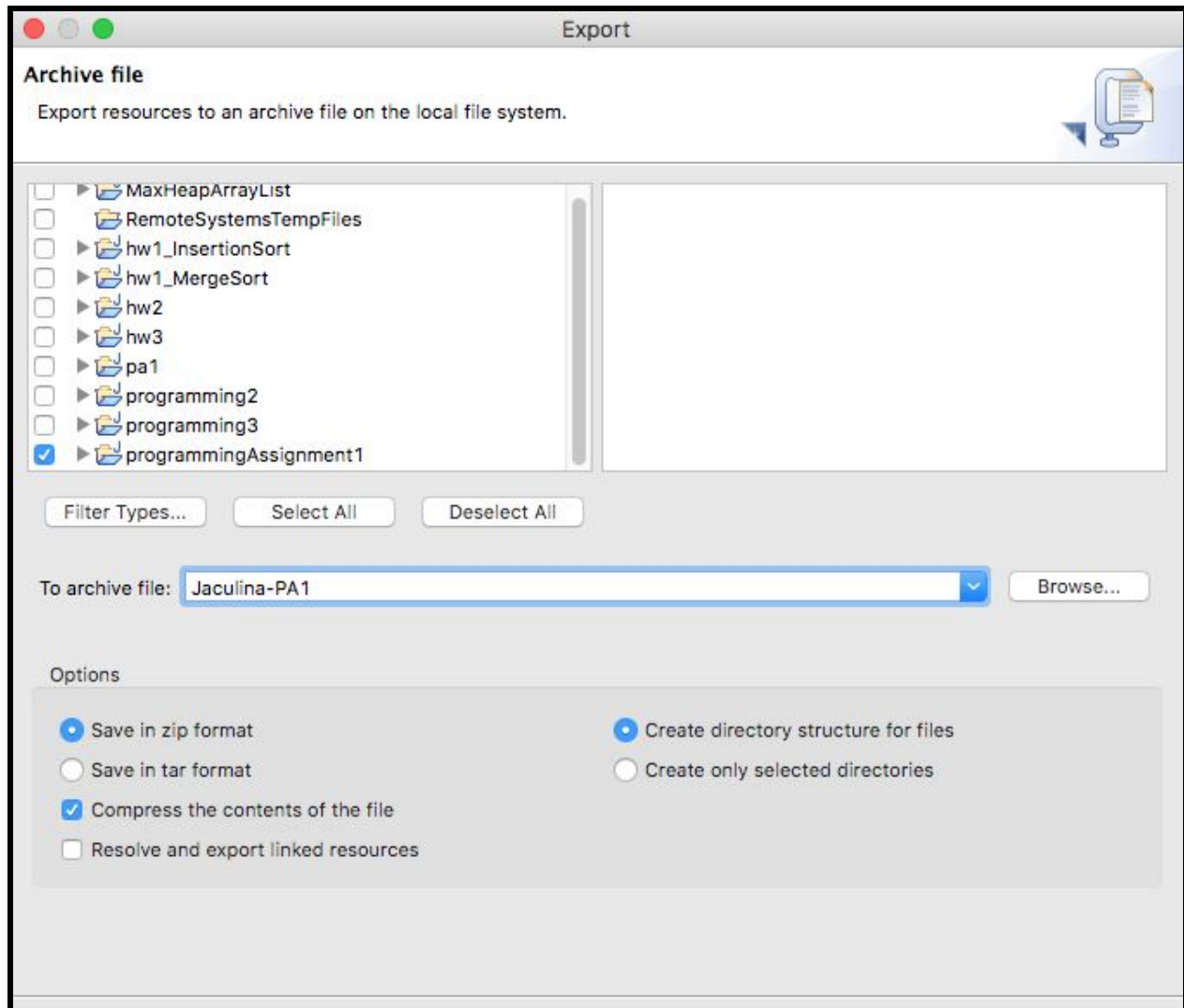
Select the Project to be exported. In this case the CPU scheduling system is under “programmingAssignment1.”



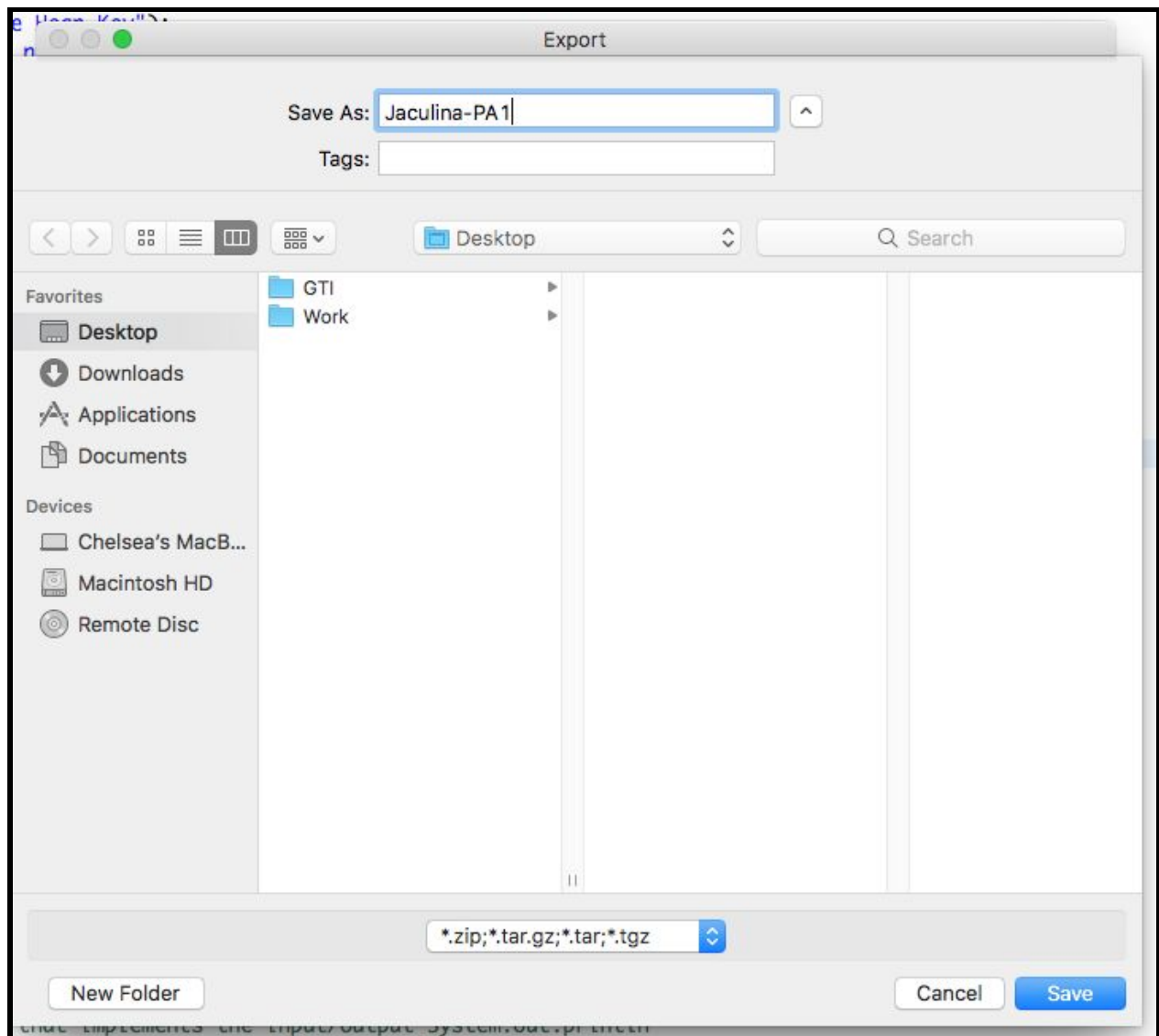
Choose the archive file type **ZIP**. To create a zip file.



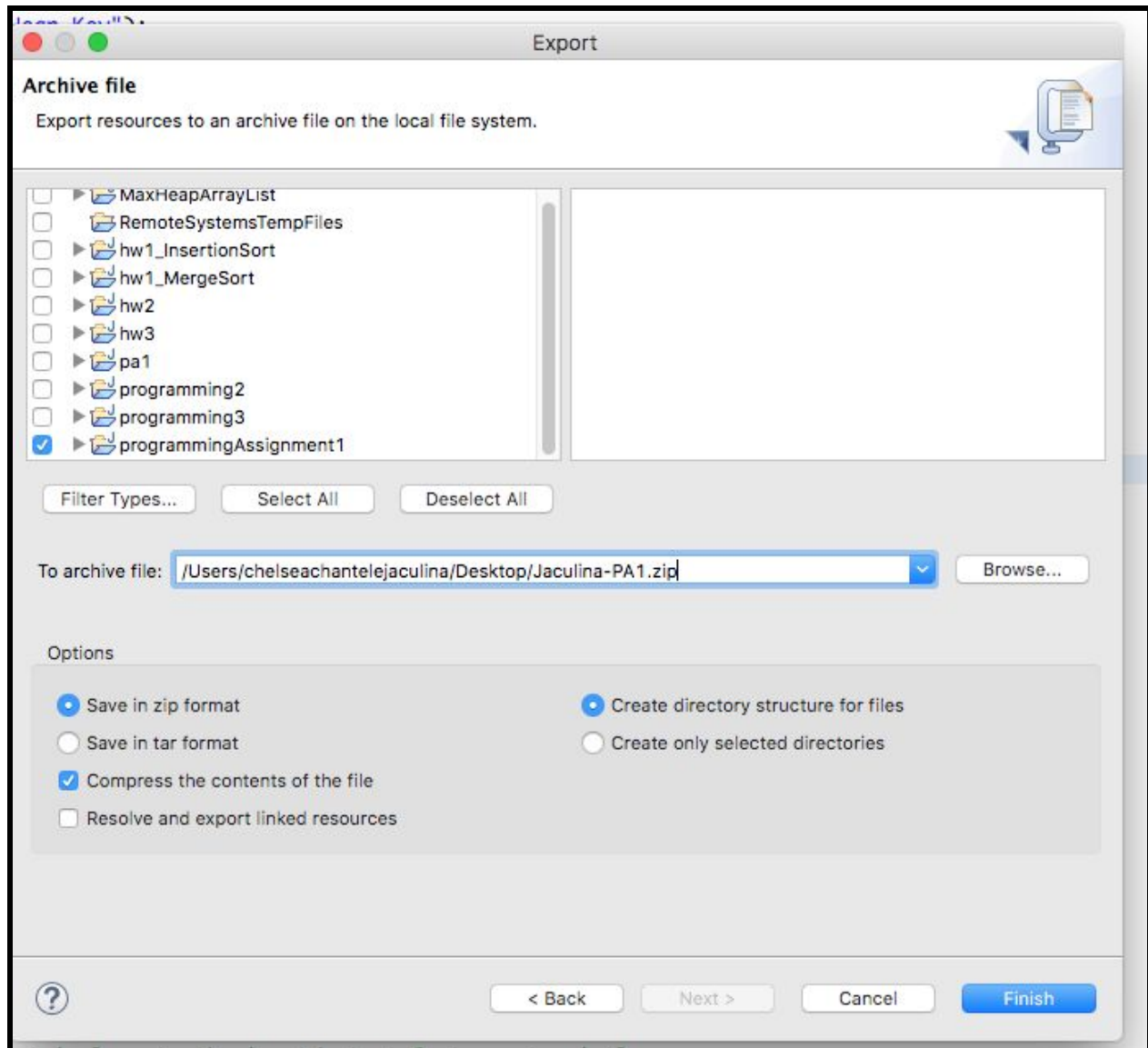
Save archive file name as Jaculina-PA1.zip (Your-Last-Name)-PA1.zip



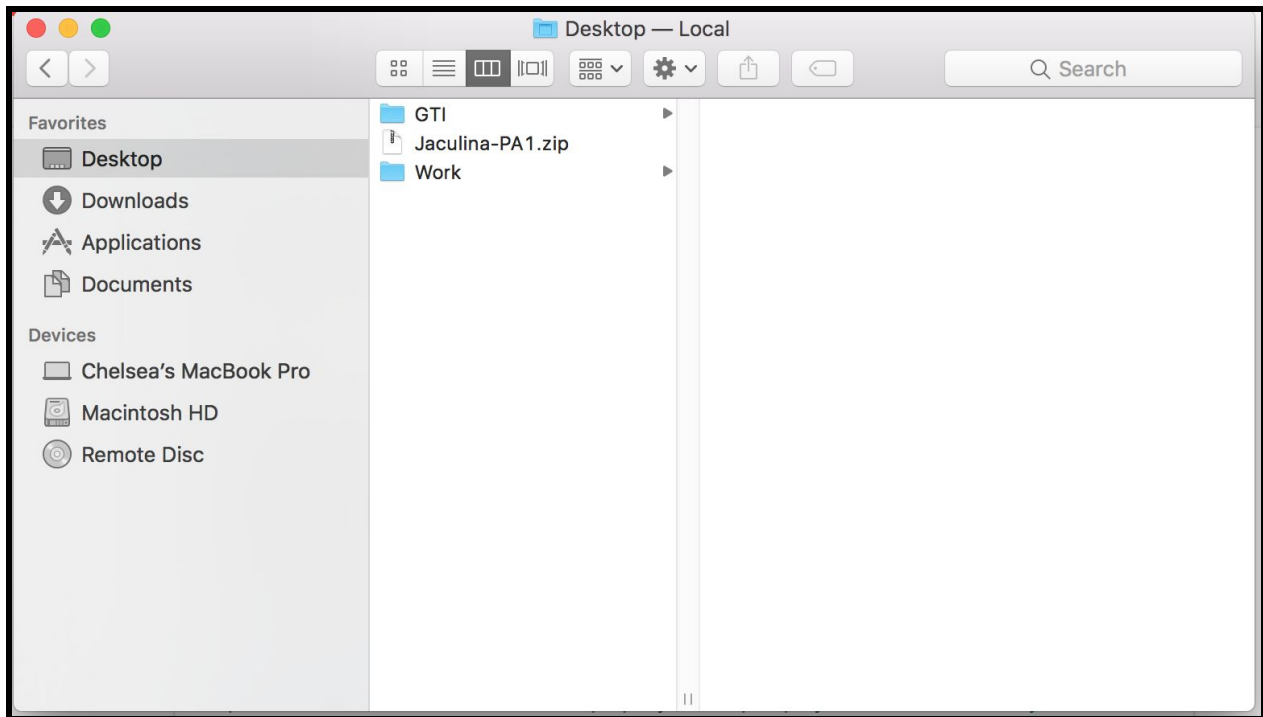
Click on Browse to save to the Desktop. Make sure it is saved as zip file.



Click **Save** to save zip file on Desktop. Then click on Finish.



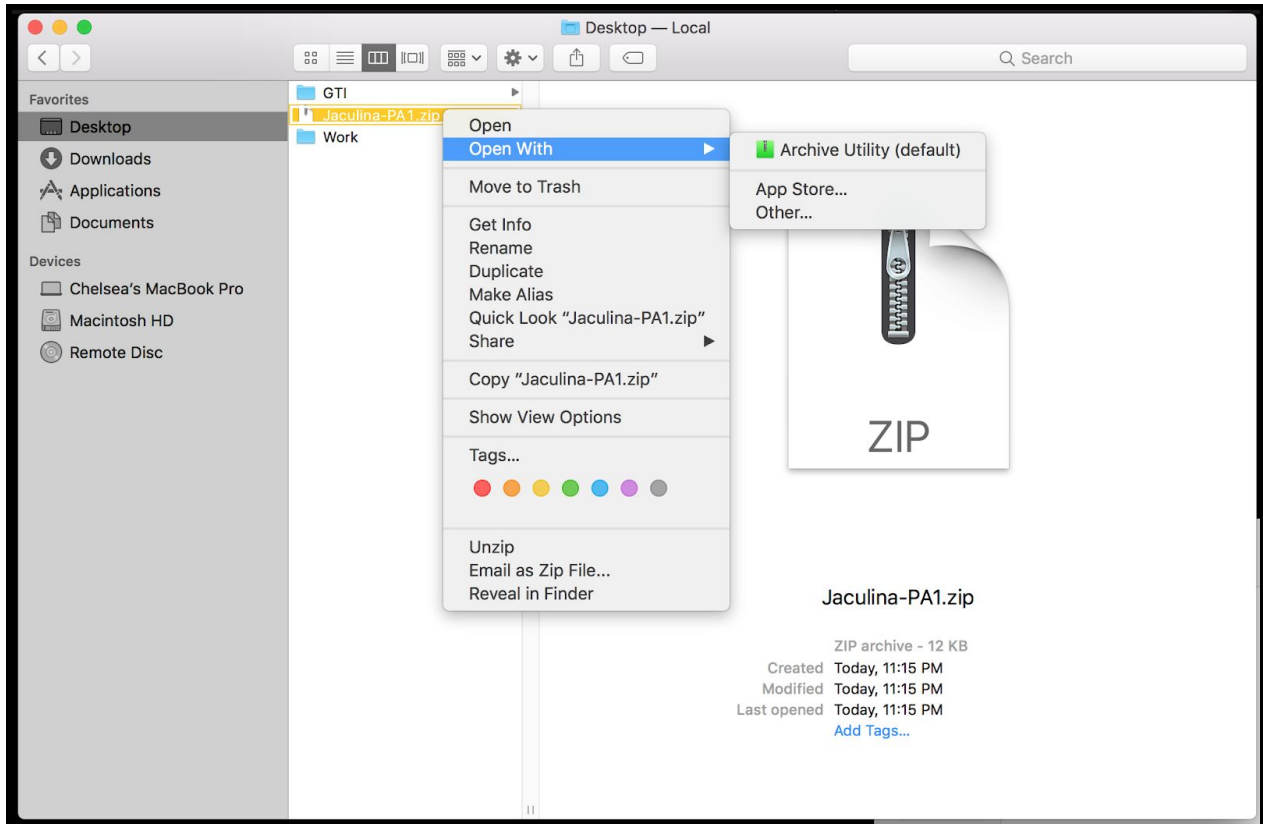
Zip File should be saved on Desktop as “**Jaculina-PA1.zip**”.



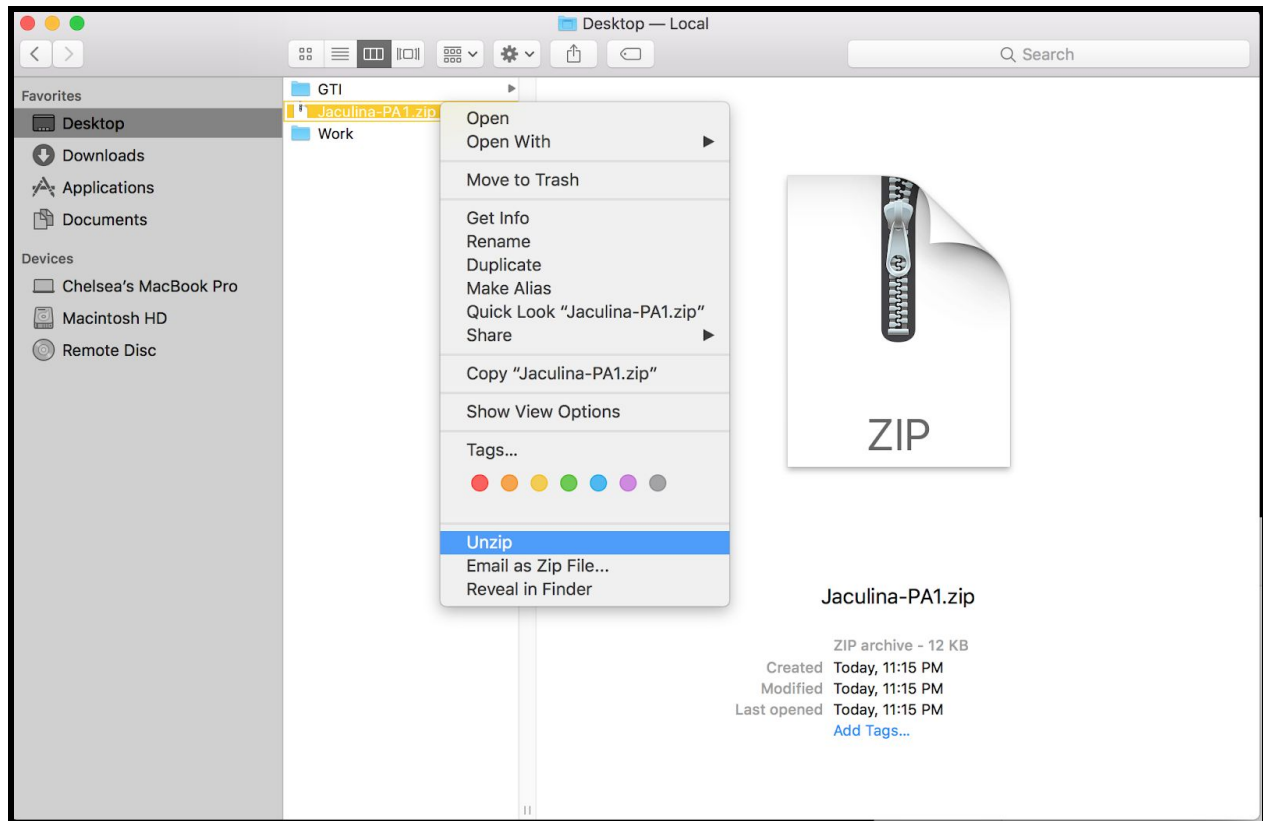
How to Unzip File

There are two ways to download the zip file and unzip it.

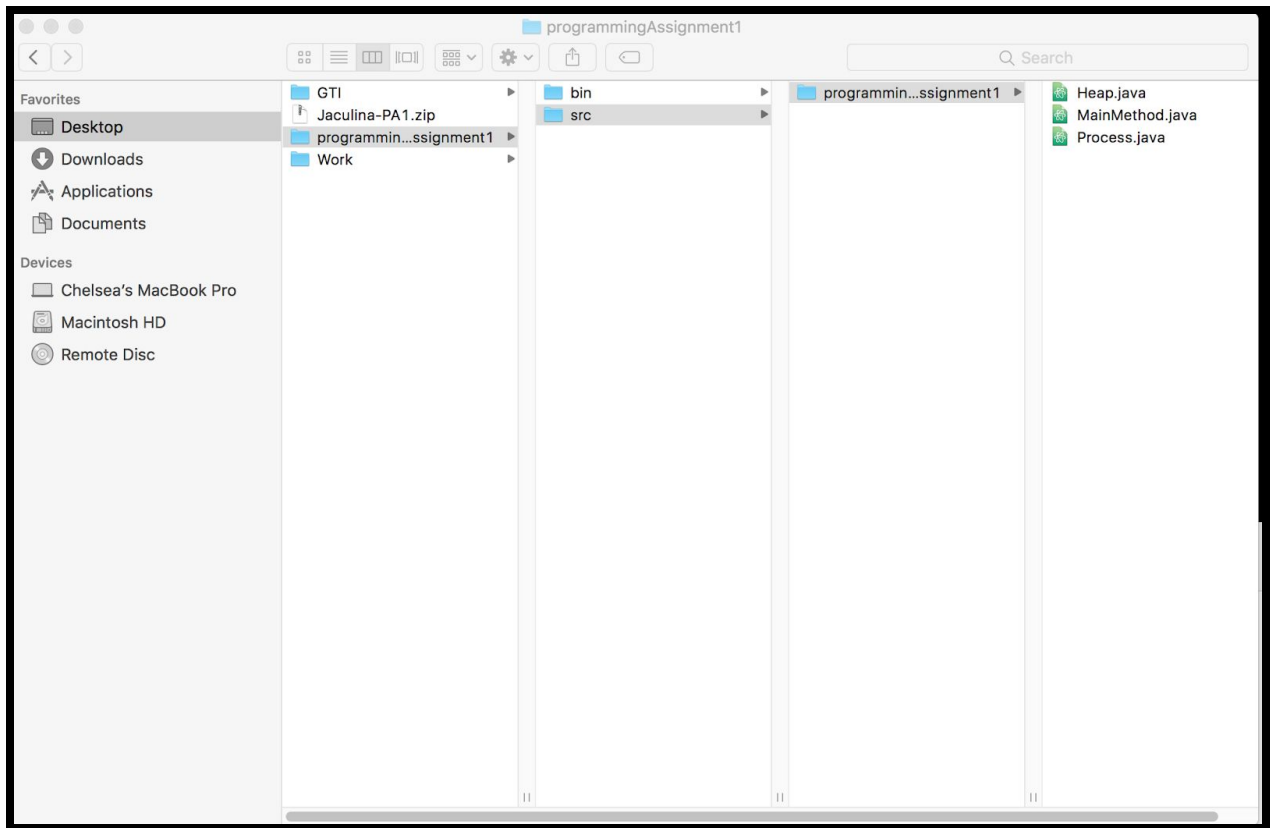
1. One way is to right click on the zip file and select the option **“Archive Utility (default).”**



2. The second way is to right click on the zip file and select the option **“Unzip.”**

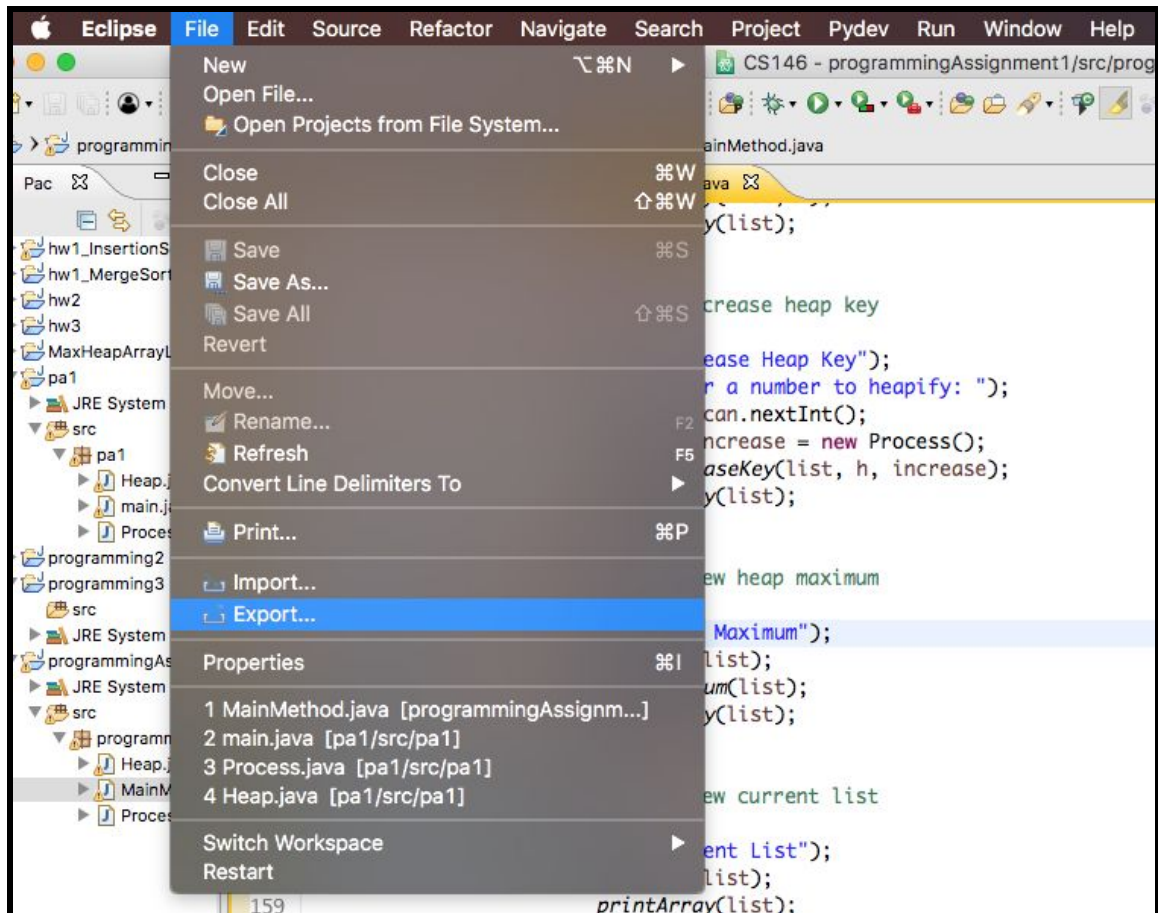


The result will be of unzipping the file will contain a folder that has a subfolder called **src** which has a subfolder called **programmingAssignment1**. In **programmingAssignment1** will contain the 3 java source code files (Heap, MainMethod, and Process).

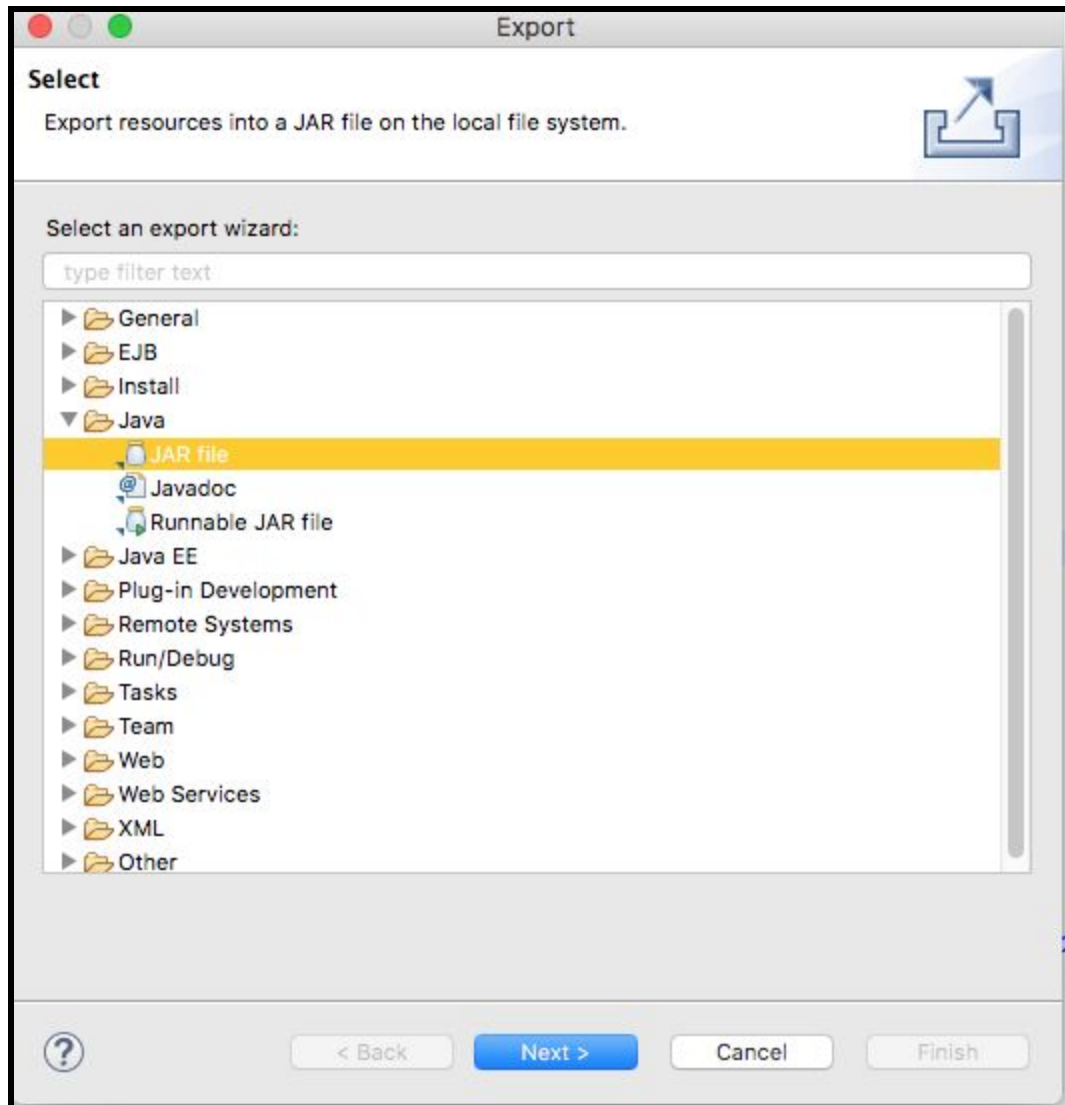


Jar File

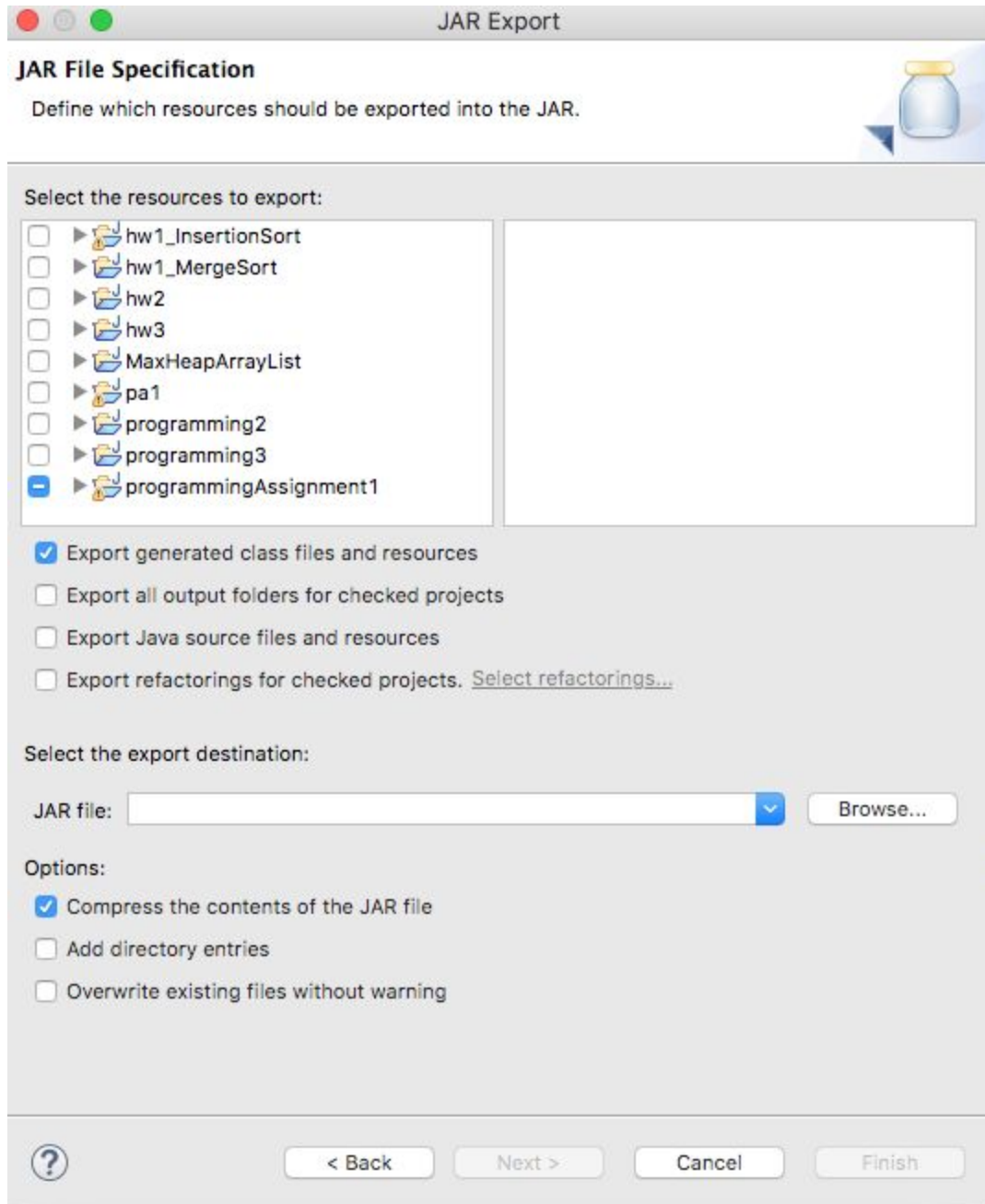
1. File → Export



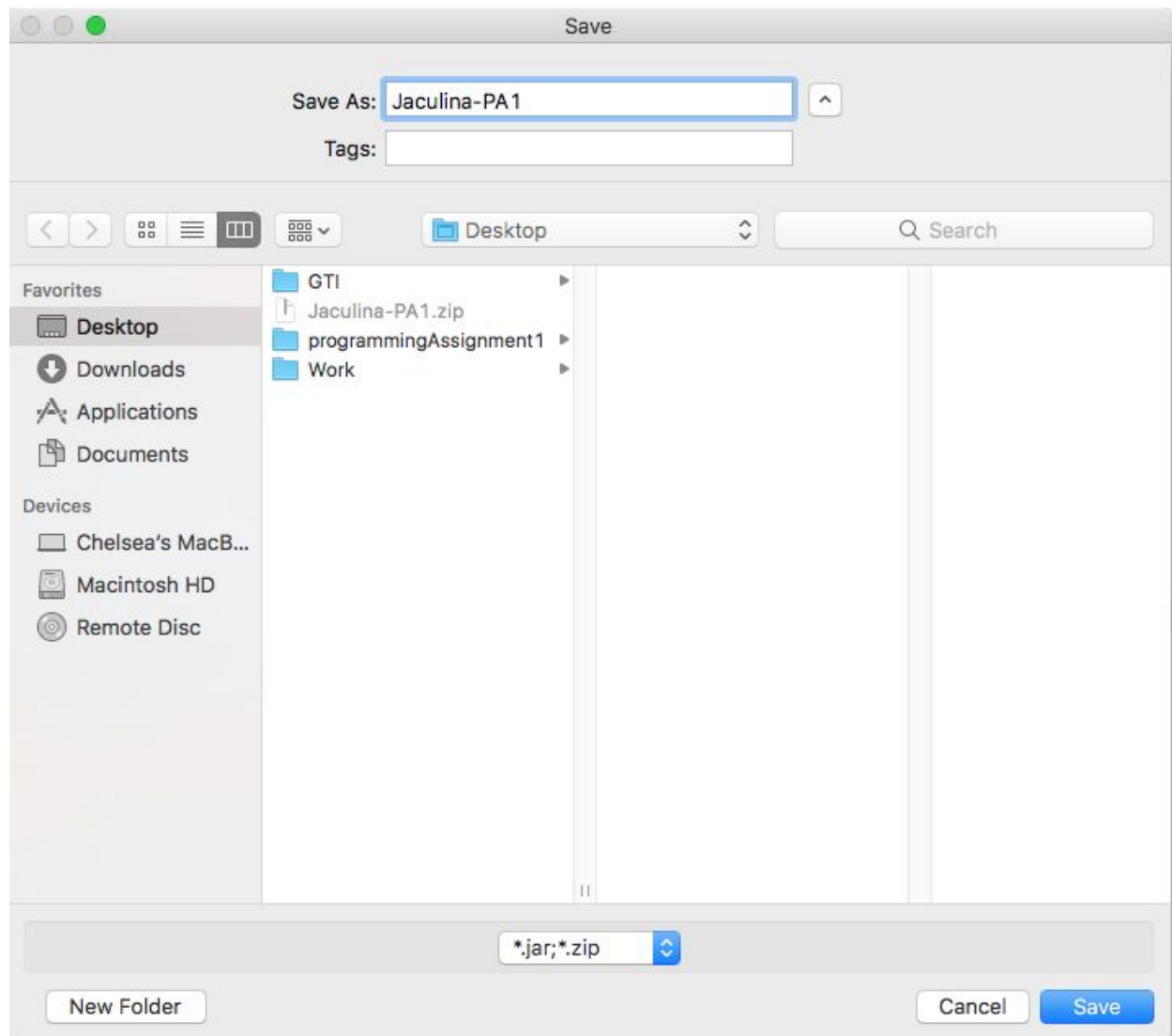
2. Selection the Java node and choose JAR file



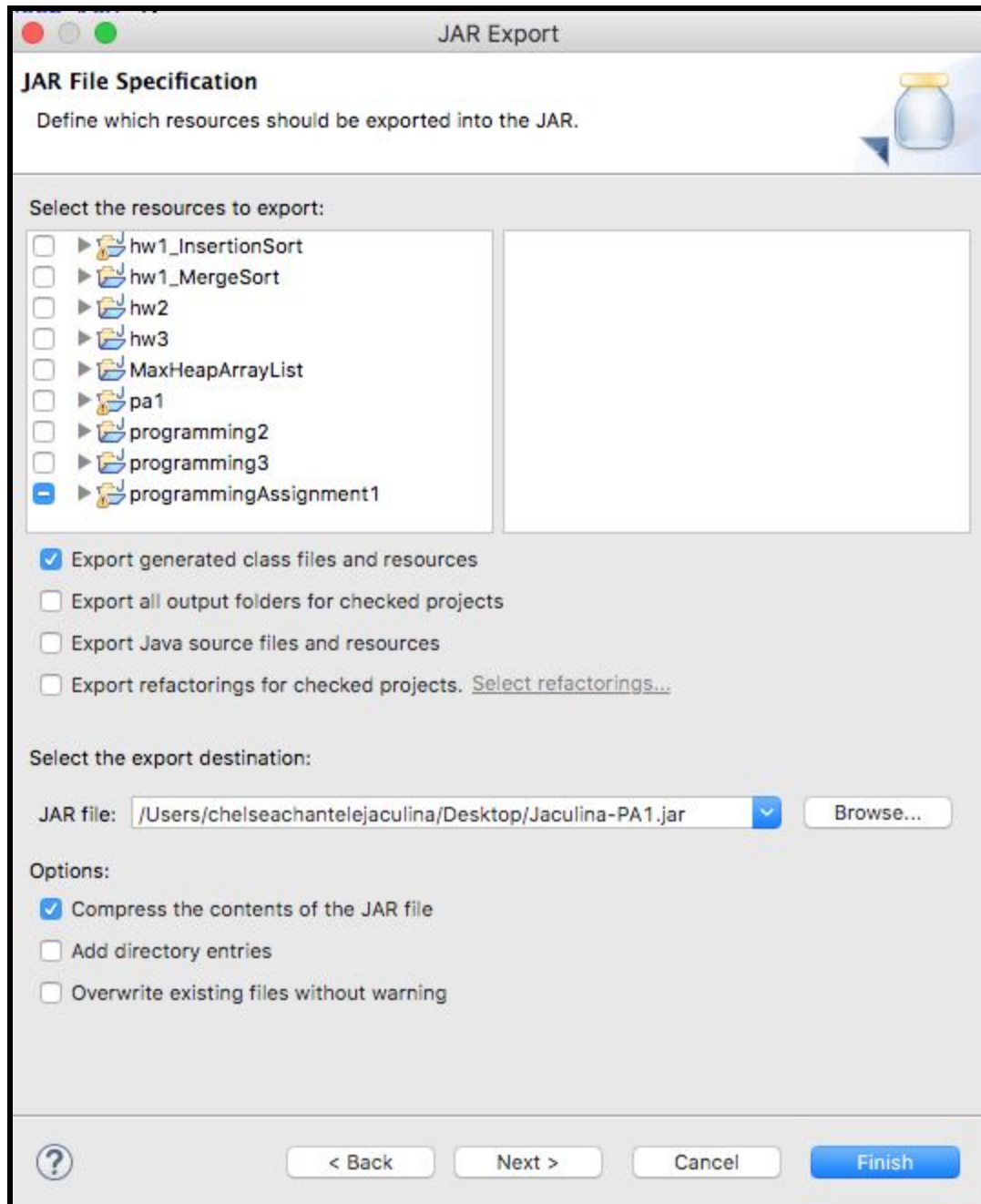
3. Select the Project file **“programmingAssignment1”**



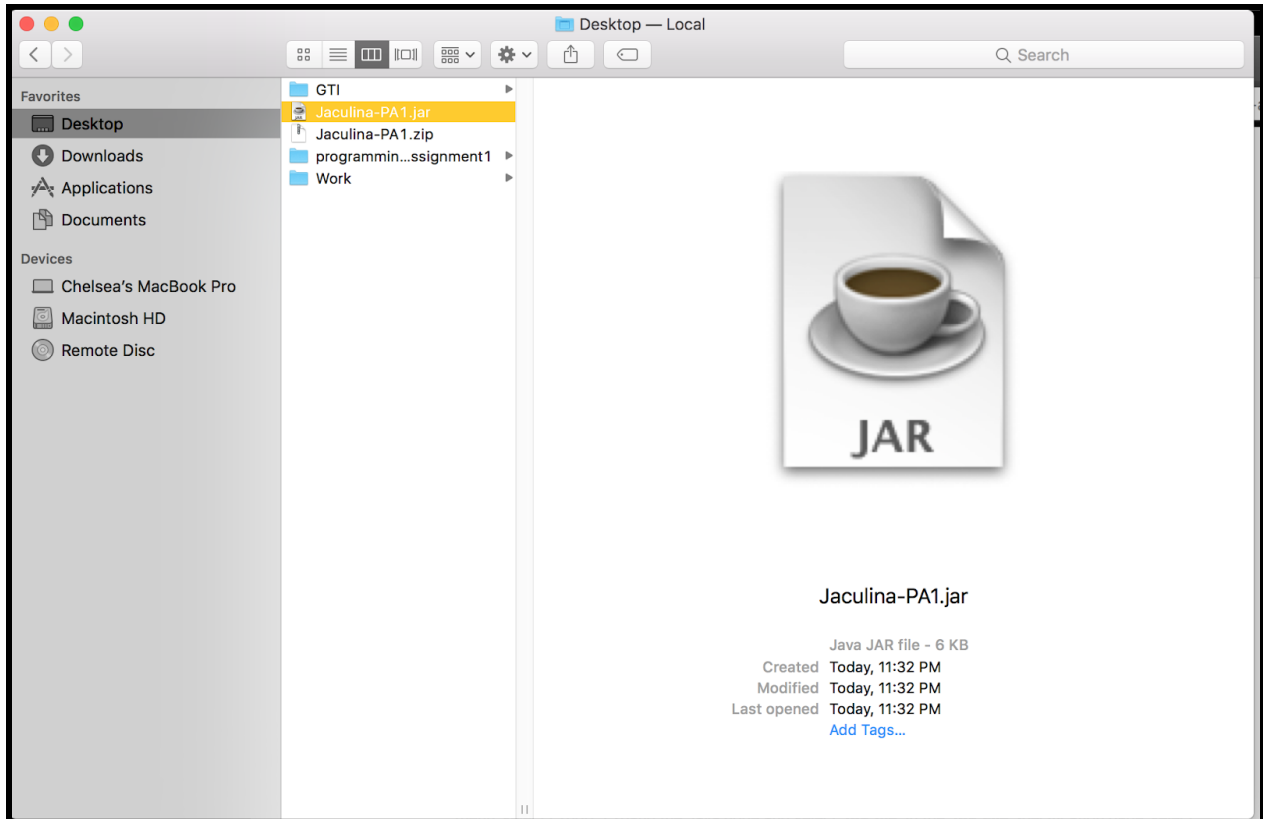
4. Select Browse and choose Desktop to save your Jar file. Name it with the extension lastname-PA1. Click save.



5. Select Finish



6. Jar file should be saved on the Desktop

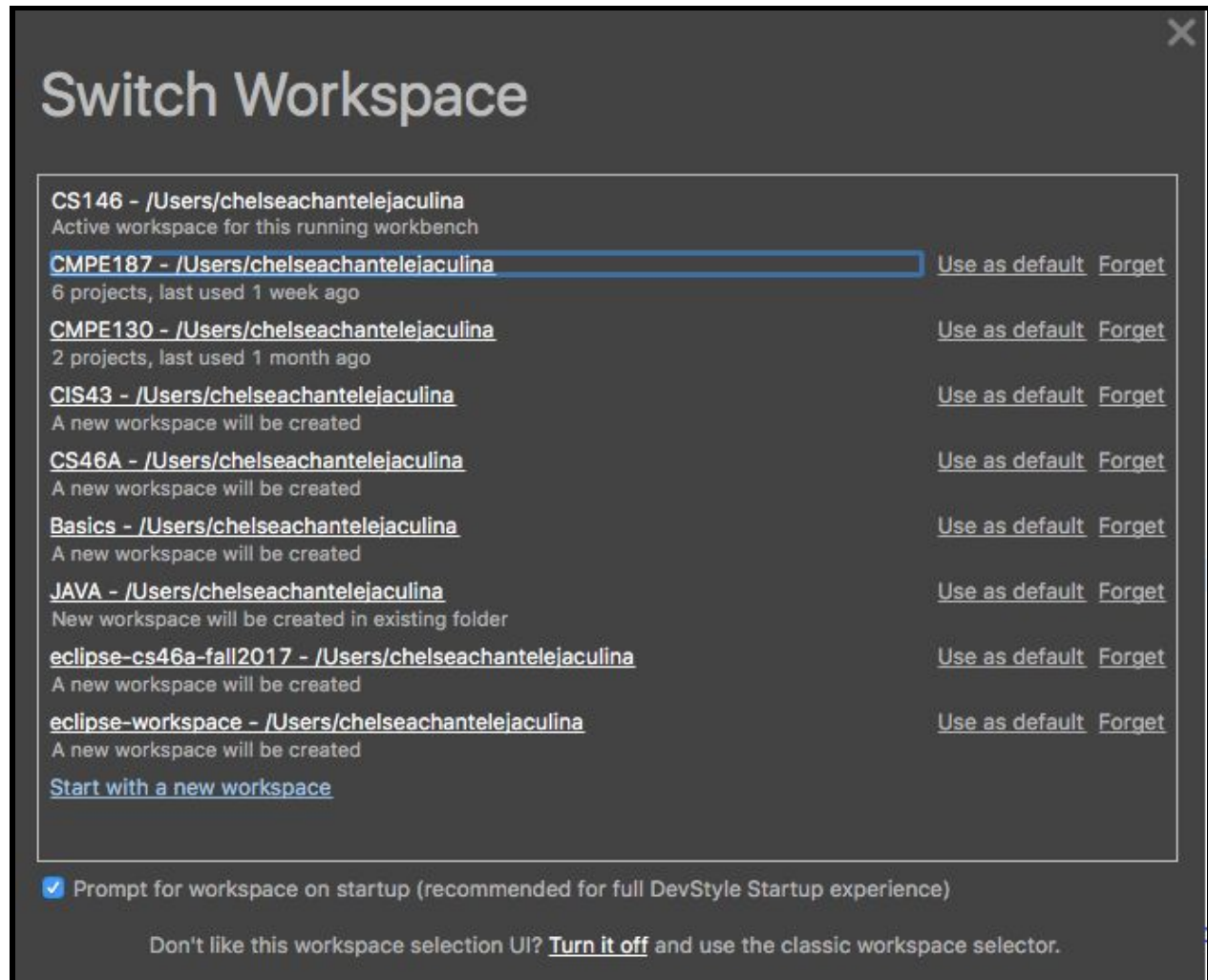


Step 3 How to Run the Application

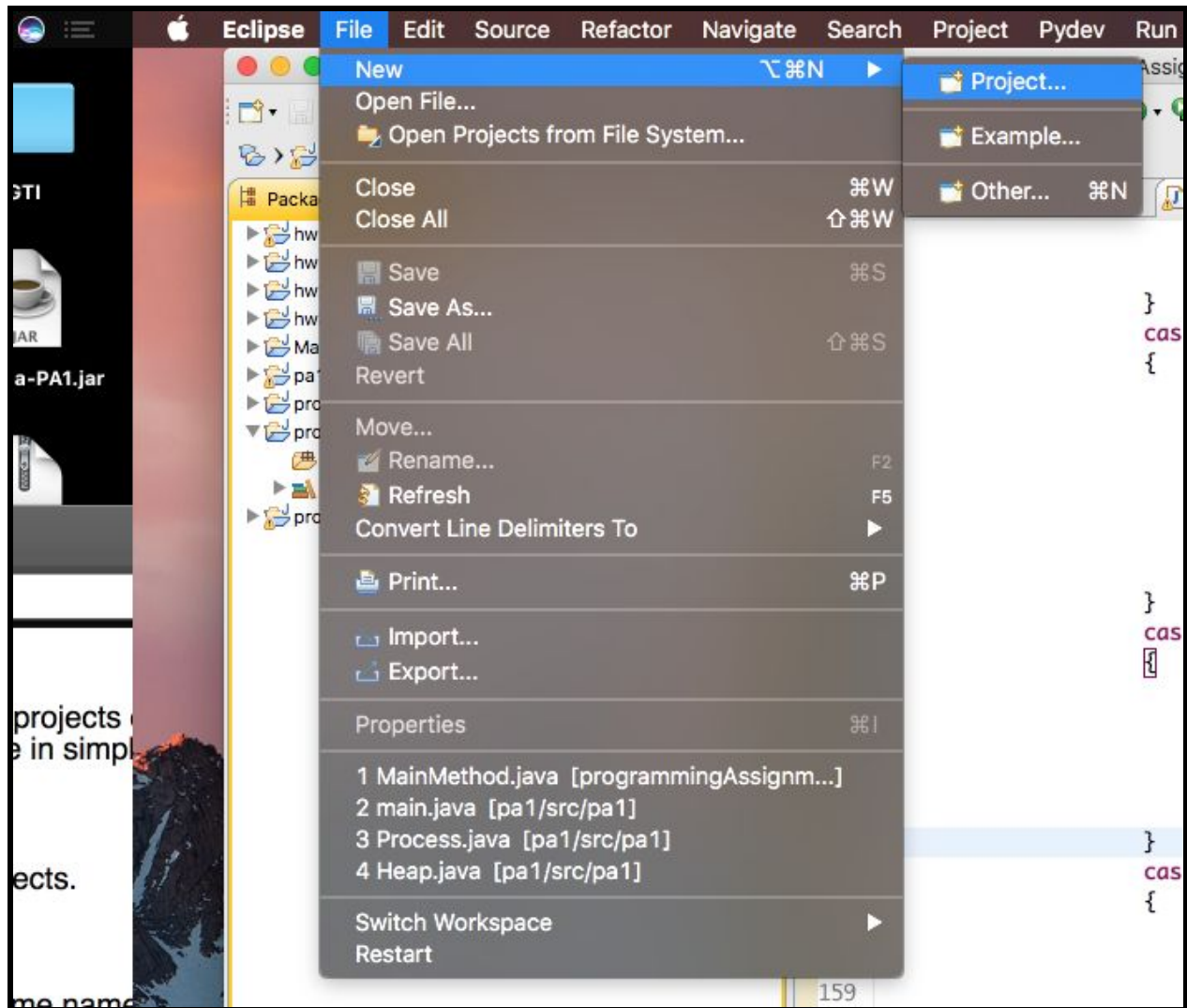
To run this CPU application it will be easier to run it through the zip file.

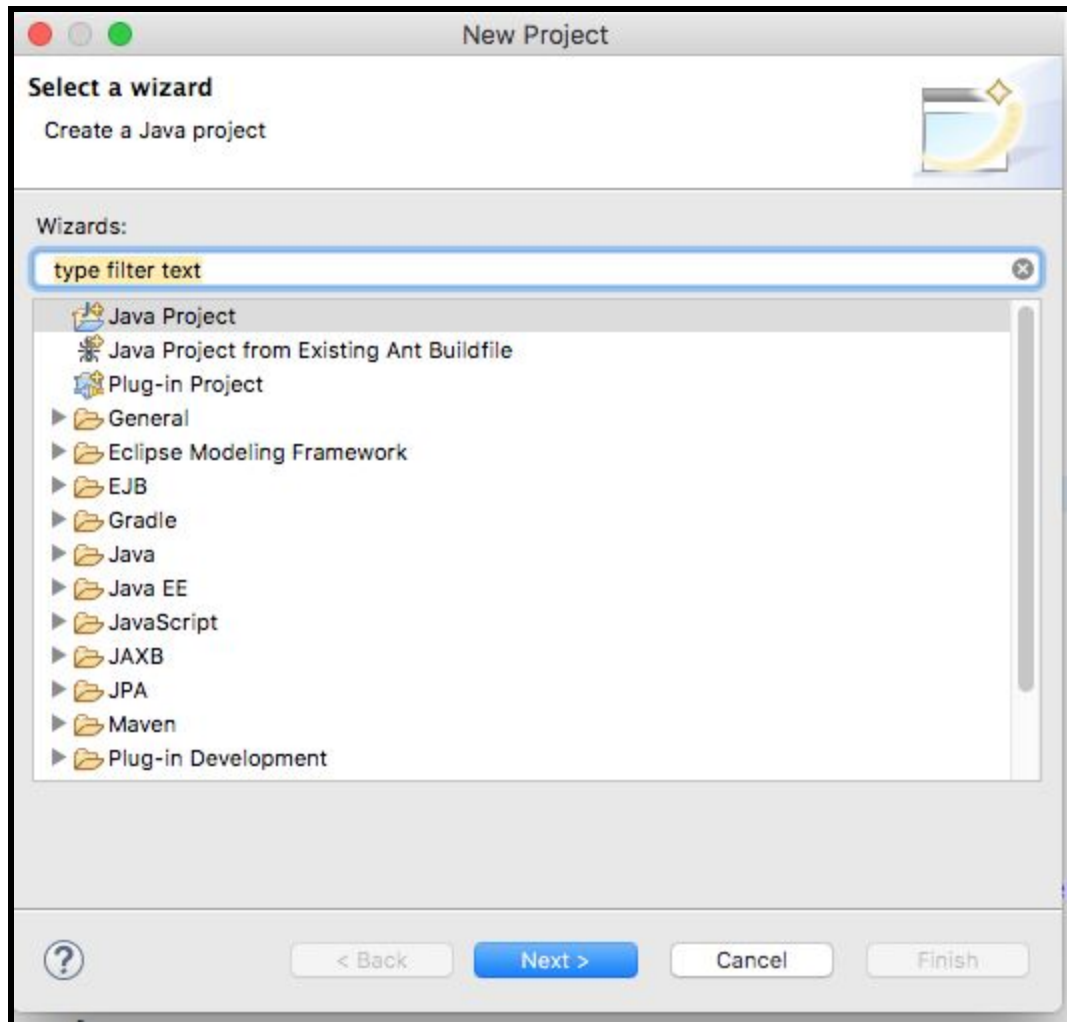
Zip File

1. Open Eclipse and choose a workspace



2. Create a Project. Select **Java Project**. Name it “CS146PA1.” Select Finish and a new Project folder should be displayed.





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: [Browse...](#)

JRE

☒ Use an execution environment JRE: [Configure JREs...](#)

☐ Use a project specific JRE:

☐ Use default JRE (currently 'Java SE 9 [9]')

Project layout

☐ Use project folder as root for sources and class files

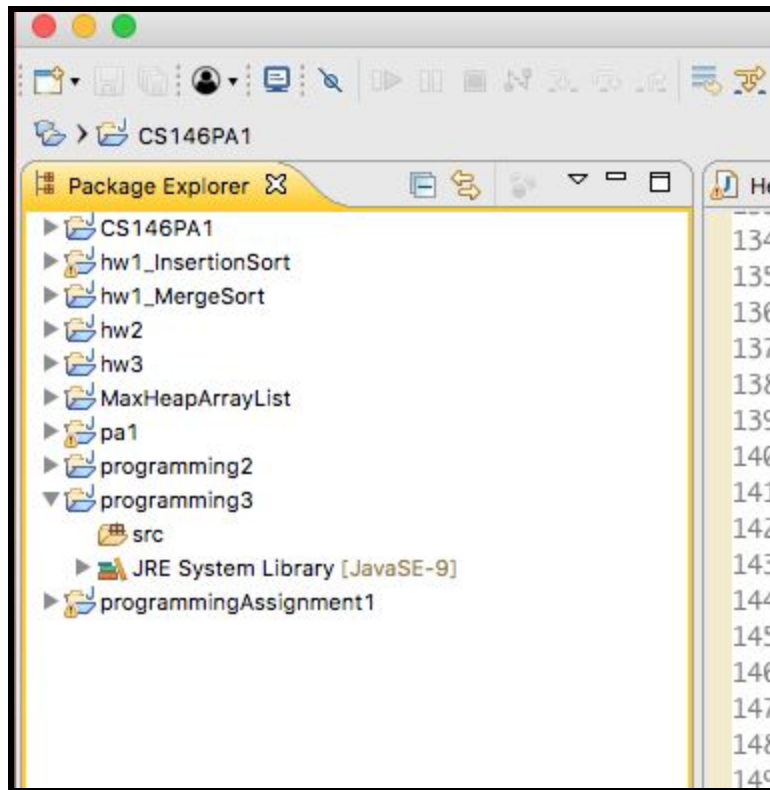
☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

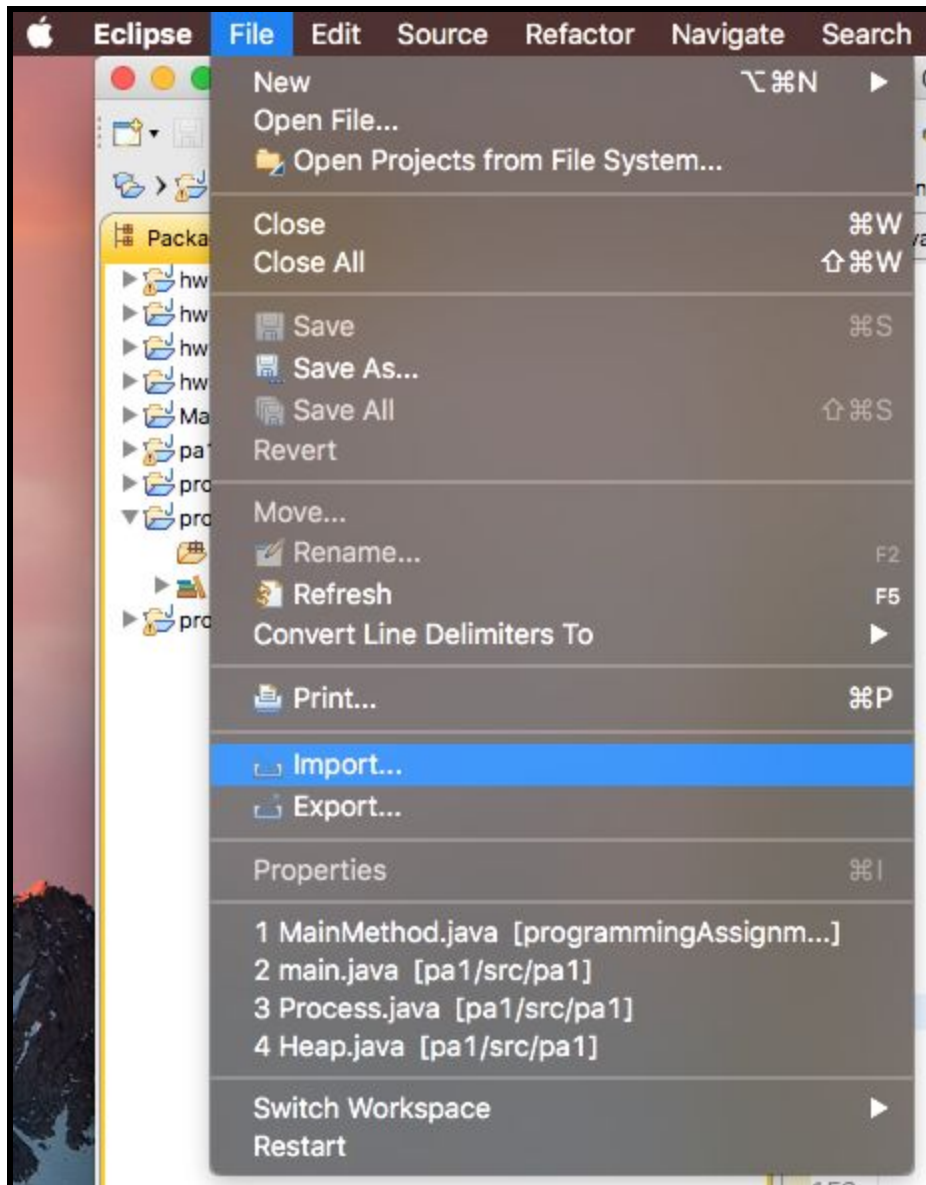
☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

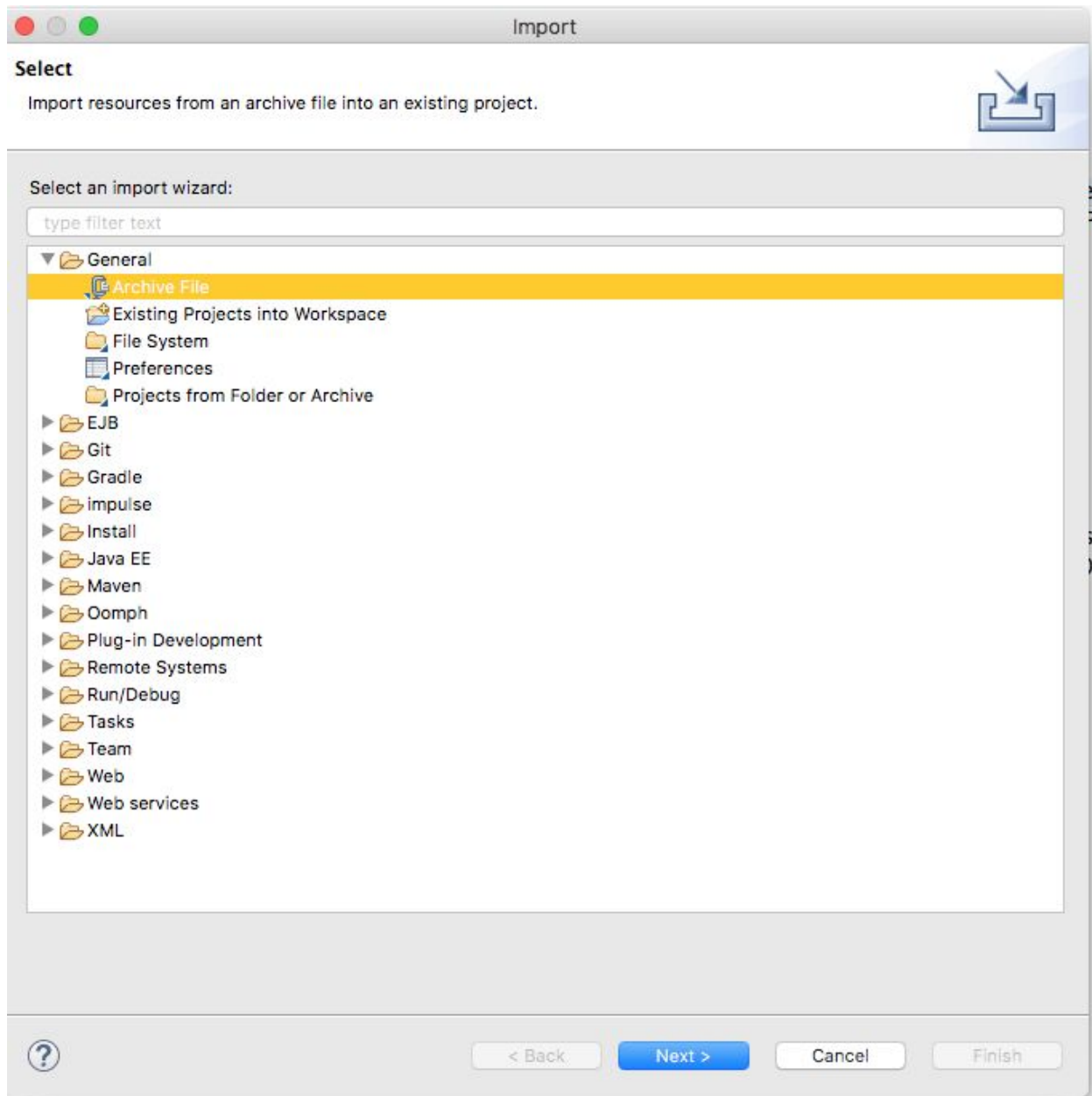
[?](#) [< Back](#) [Next >](#) [Cancel](#) [Finish](#)



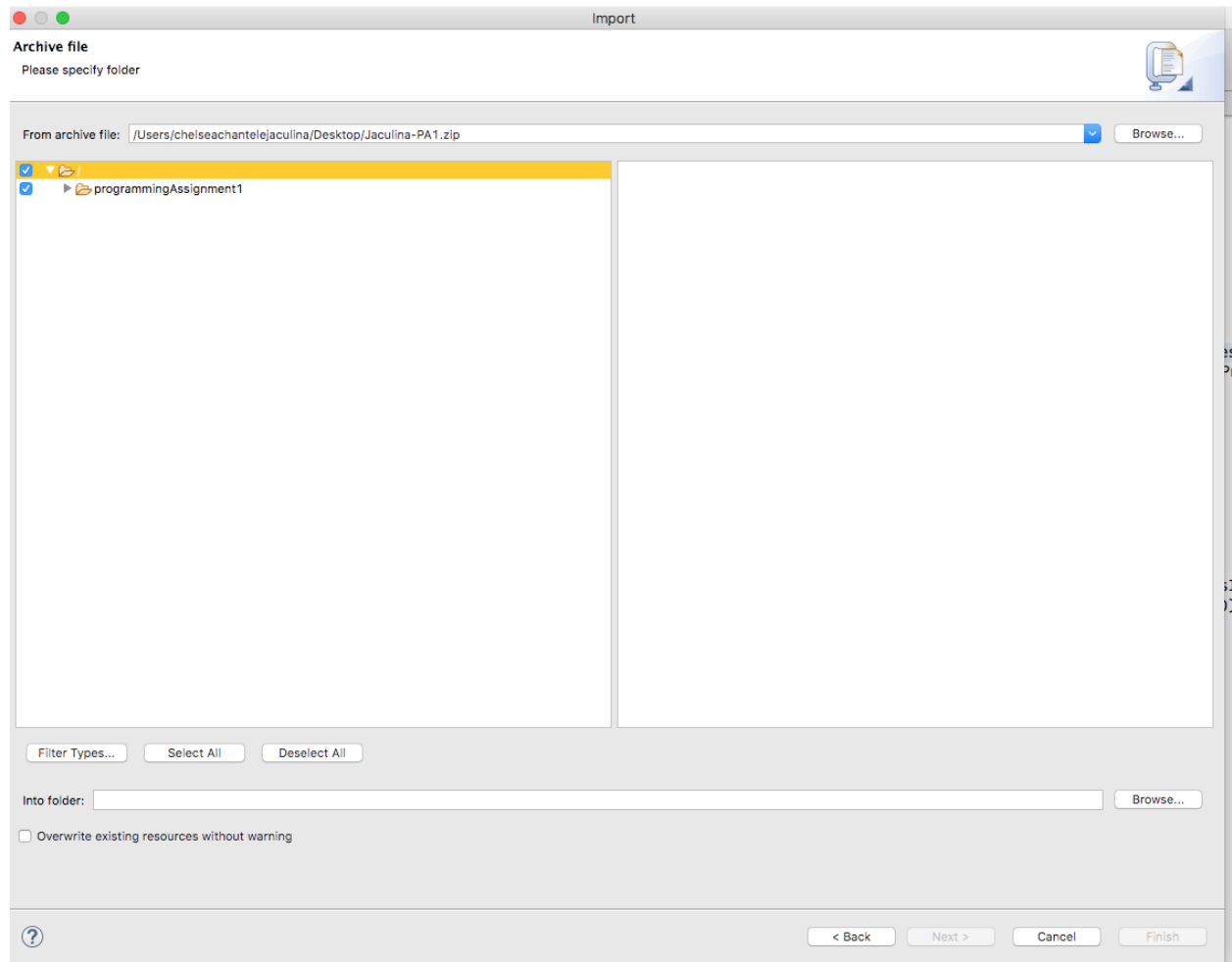
3. Select File → Import



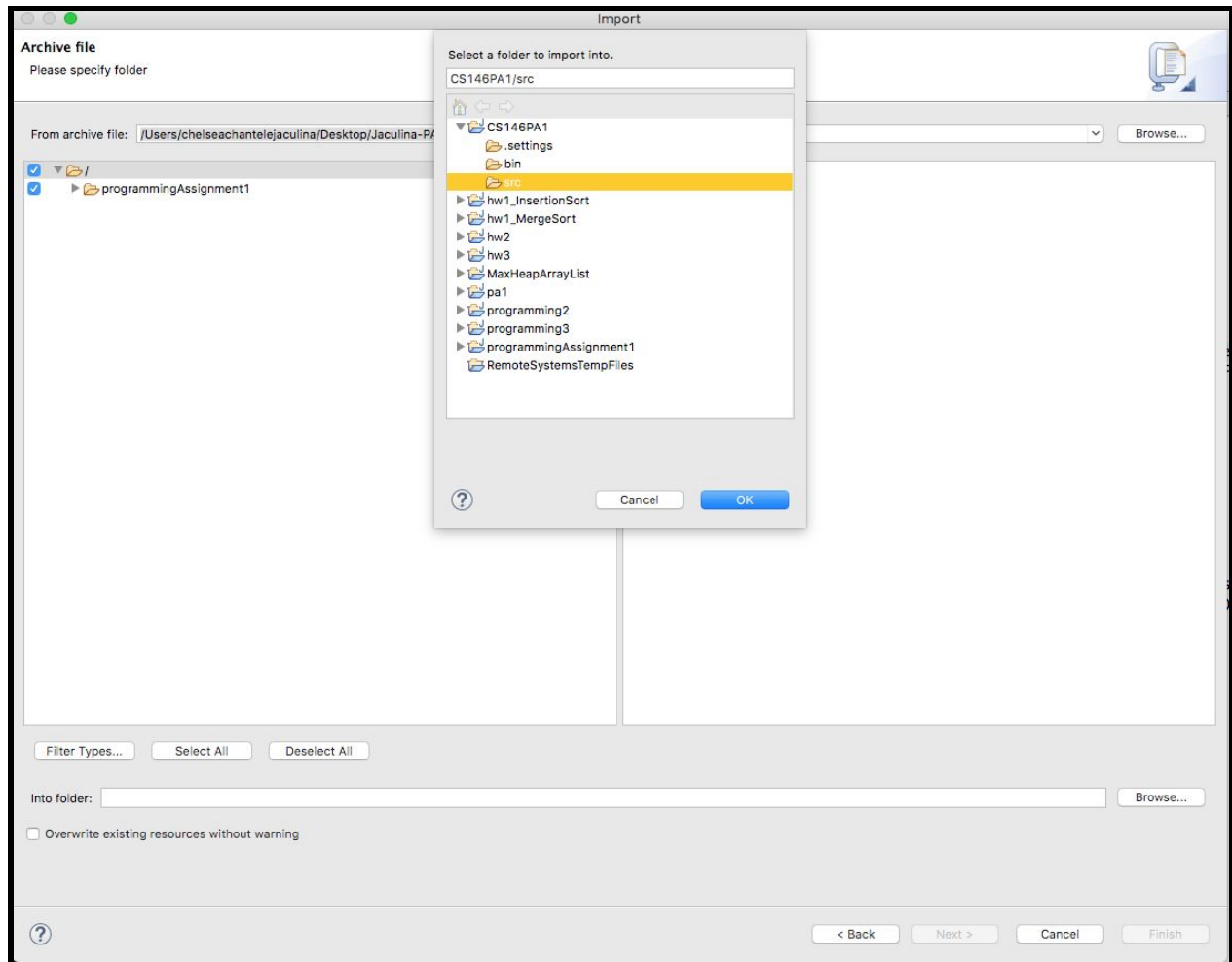
3. Select **General** → **Archive File**. Click Next.



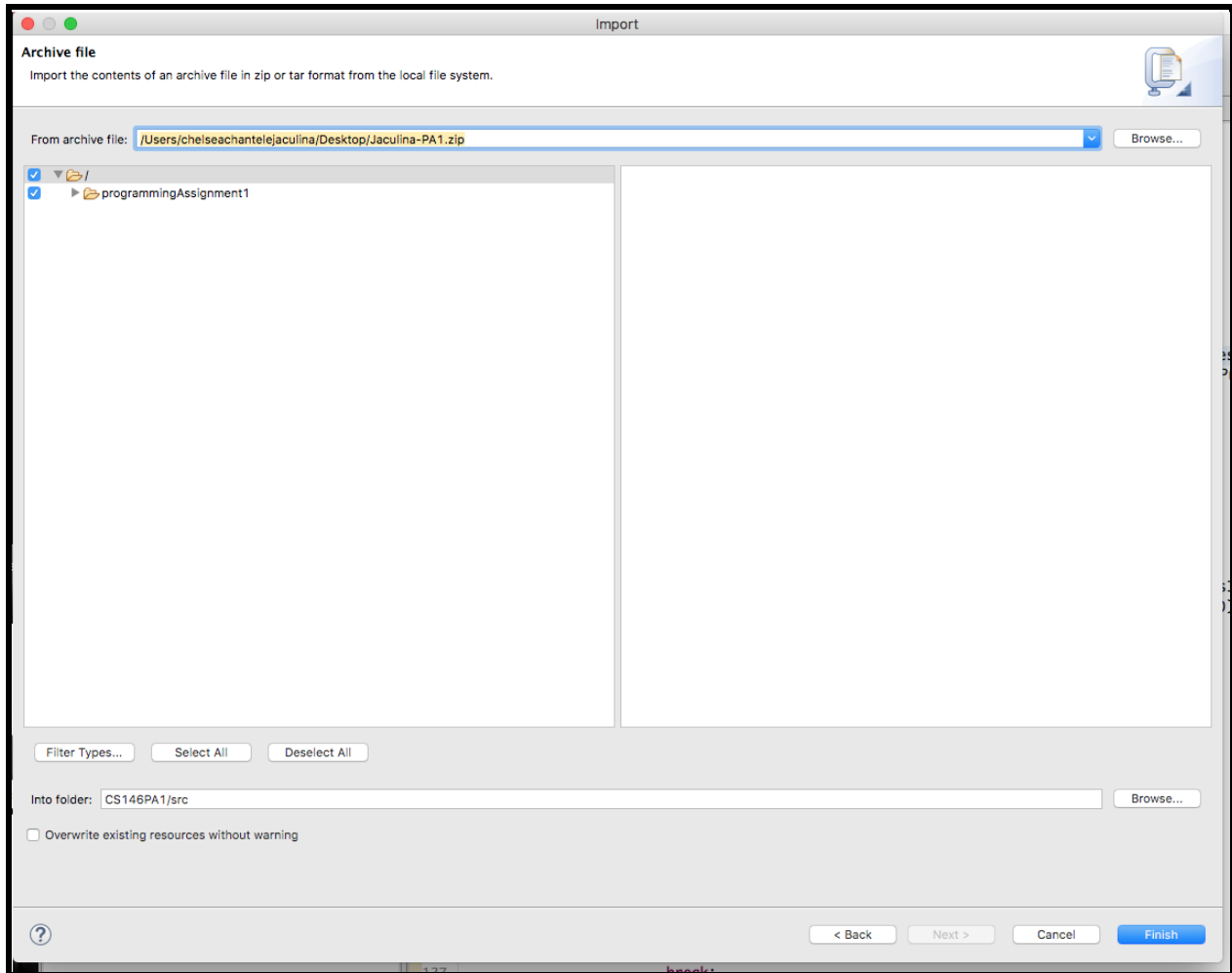
4. Click on Browse and search for the zip file on your Desktop space.



5. Then click on the bottom **Browse** button and insert it to the Java project folder “CS146PA1” → **src** folder.



6. Select Finish.



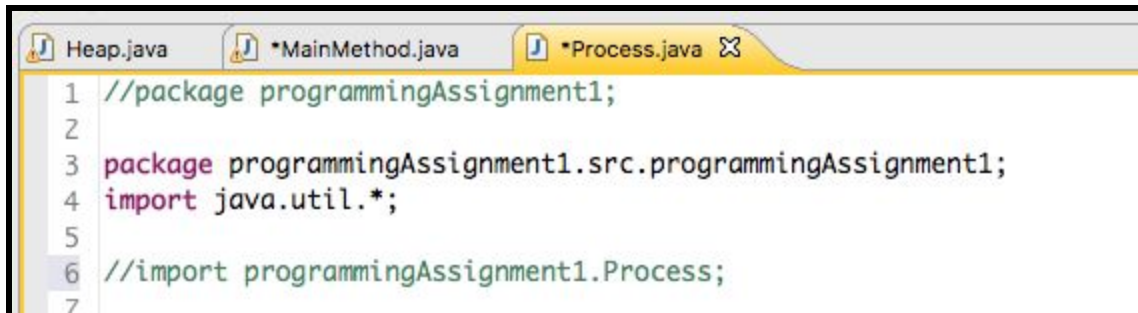
7. Change the package name to programmingAssignment1.src.programmingAssignment1 in each class

The image shows a snippet of Java code in a code editor. The code is as follows:

```
1 //package programmingAssignment1;  
2 package programmingAssignment1.src.programmingAssignment1;  
3 import java.util.*;
```



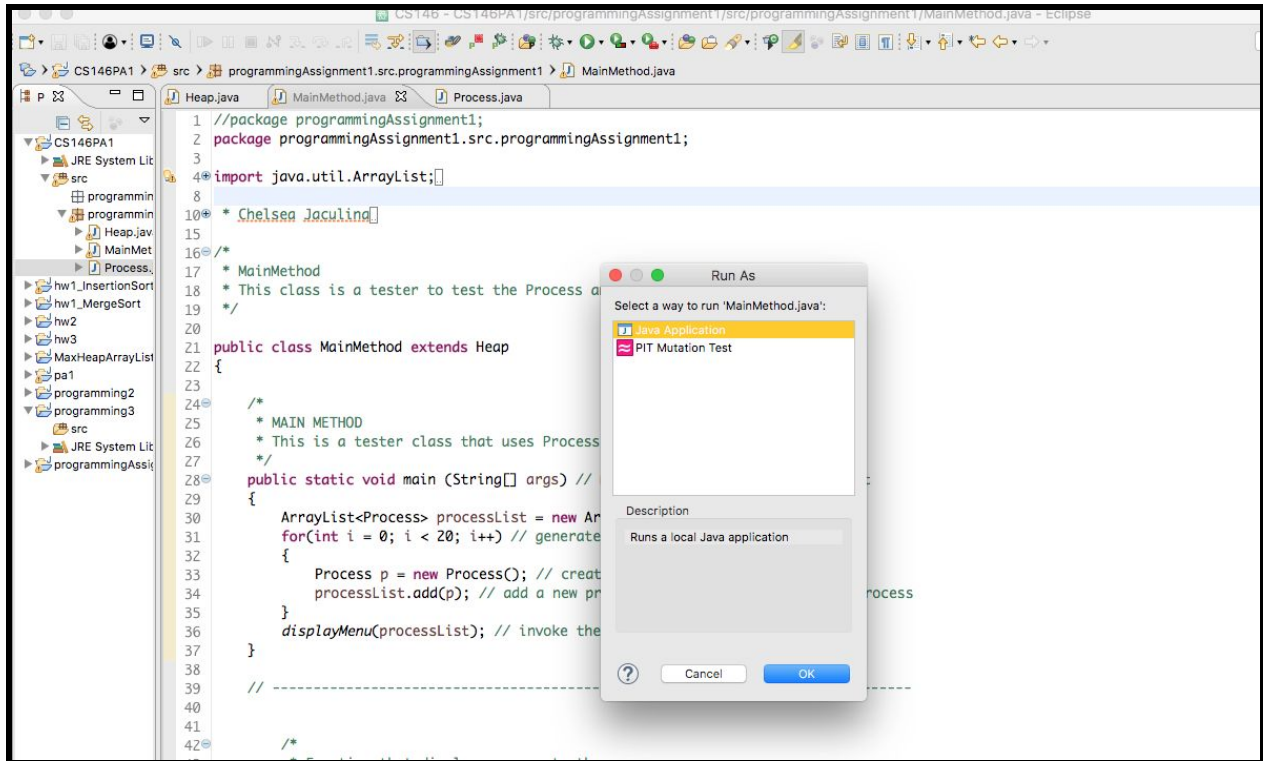
```
1 //package programmingAssignment1;
2 package programmingAssignment1.src.programmingAssignment1;
3
4 import java.util.ArrayList;
5
6
7
8
9
10 * Chelsea Jaculina
11
12
13
14
15
```



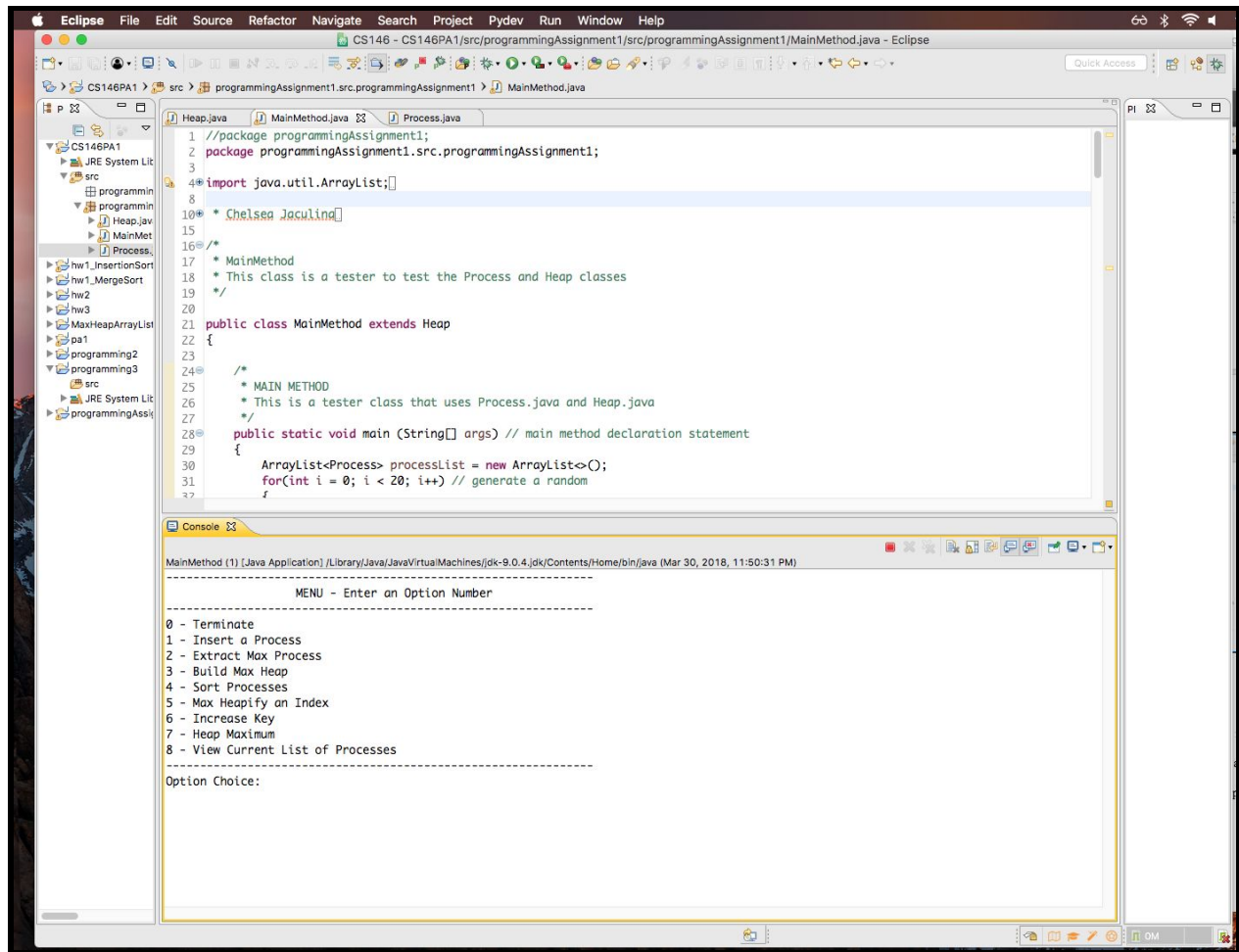
```
1 //package programmingAssignment1;
2
3 package programmingAssignment1.src.programmingAssignment1;
4 import java.util.*;
5
6 //import programmingAssignment1.Process;
7
```

In Process.java make sure to also comment out the second import statement.

8. Save all java files. Then run the Main Method by clicking the Green circle button on the top. Run it as a Java application.



9. After running the main method you should see the **Menu**. Input your option choice!



Step 4 Installation Needed

To install the application to your computer, you must have a Java IDE. It is recommended to use Eclipse IDE since the application has been developed on Eclipse Java IDE for Web Developers Version Oxygen.1a Release (4.7.1a).

Problems Encountered

The first problem encountered when implementing the CPU scheduling system was to understand the concept of what a CPU scheduling system actually does. Once that was finally understood, another problem was translating the Introduction to Algorithms pseudo codes textbook from Chapter 6 to Java source code. Most of the pseudo codes that are in the text book start at indices 1. This became a problem to wrap my head around and kept giving me index out of bounds errors since Eclipse Java is 0 based. After understanding the off by 1 error, testing the functions were just as difficult since the Heap.java class needed extend the Process.java class. A third problem that was encountered was picking to use arrays or arraylists. Though retrieving values from the arrays are simpler, using an arraylists made more sense to implement the CPU system since arraylists size can change unlike arrays. Other problems I encountered when creating this CPU scheduling system was creating the main method tester class. The menu itself was hard to run since I kept getting errors when testing each heap function. To find the errors I needed to use the debugger raw out the expected outputs, as well as incorporate print statements.

Lessons Learned

Many lessons were learned in developing this project. The first and most important lesson includes understanding why we use heaps for creating priority queues. Even though there are many other algorithms and data structures out there, implementing this CPU scheduling system made me realize that the priority queue doesn't depend on the FIFO policy, but depends on the highest priority no matter how long one process time is. Using the max heap functions heap support inserting new priorities, removing the highest priority, and increase its key value. Priority queues can be applied to many situations such as an emergency room or a printer.

A second lesson that I learned from this project was to how to write a max heap class by following the outline of the textbook's pseudocode. The max heap functions can be implemented in many ways, but I learned how to read pseudocode for max heaps and translate it to workable source code.

Another lesson that I learned was how to create my own tester class and command line interface. This taught be the ability to provide how I wanted to display a simple output and use all of the functions that I implemented.

Lastly, this CPU scheduling assignment explained to my why priority queues use heapsort. The running time of heapsort is $O(n \log n)$ which quite fast but not as fast as quicksort.

