Exam 2 **112 of 112**

CSC 300 Fall 2023

Lecturer: Howard Rosenthal

Dec. 1, 2023

Exam ground rules:

You can do this open book, including access to the Internet, but can ‘t work or consult with any other person.

Programs must be implemented as specified, including all the methods.

Source code must be submitted to [csc300csudh@gmail.com](mailto:csc300csudh@gmail.com)

Remember, I need the eleven .java files. Don’t zip them.

You must also include this statement in the email:

**I avow that I did this exam without help from any other person.**

**Exam is due at 11:00 PM on Sunday Dec. 3, 2023. No late submissions will be accepted. Remember to make sure to submit all your files. Count and make sure that there are 11, and that you submit the correct ones, not an empty file. If you don’t finish submit whatever you have completed for partial credit.**

**General Instructions.**

The following approach should help you work more quickly and efficiently, although you don’t have to do it this way.

Remember, you can’t actually execute the program until you finish coding the **ModelAdvisingCenter** class

1. Read the entire problem at least once.
2. Note that if you code and compile in a certain order you can eliminate syntax and other compile time bugs as you go.
3. Code **AdvisingType** and compile
4. Code **FullTimeAdvising** and **PartTimeAdvising**, compiling each one. As described below these are very similar. I have tried to point out the differences in red, but be sure to check for yourself.
5. Code **Student** and compile.
6. Code **Senior**, **JuniorSoph** and **Freshman**, compiling each one. As described below these are very similar. I have tried to point out the differences in red, but be sure to check for yourself. (Note that their studentTypes, as Strings, are in reverse of the natural order.)
7. Code **Advisor** and compile.
8. Code **StudentPriority** and compile.
9. Code **AdvisingCenter** and compile.
10. Code **ModelAdvisingCenter**
11. Execute the program fixing any bugs/crashes.
12. Test using my inputs and compare to my outputs. Outputs are in the advisingoutputs.txt file in EXAMS/CURRENT SEMESTER EXAMS/EXAM 2; They are also found on canvass.
13. If you follow the detailed instructions below you will get the same outputs.

**Summary:**

The following program simulates a very simplified version of a single AdvisingCenter. The purpose is to watch Students arrive and receive their advising. The usefulness of this simulation is to look at the wait times, figure out what is acceptable, and assign the appropriate number of Advisors. Aside from using my inputs as the test case you can try to run with different numbers of Advisors and see what happens. This example does not include the concept of reservations, although they could be easily added as the model becomes more mature and accurate. (Don’t do this for this exam, but you can try over the break.) This simulation is designed to process any student who arrives before the door closes.

There is no input file to read from. I have included a sample output file advisingoutputs.txt to download from the site at EXAMS/CURRENT SEMESTER EXAMS/EXAM 2 or canvass. It is also an attachment in the e-mail that has this file. Remember to view this file using Notepad++ or jEdit in order to see the columns correctly.

The eleven classes are **AdvisingType**, **FullTimeAdvising**, **PartTimeAdvising**, **Student**, **Senior**, **JuniorSoph**, **Freshman**, **Advisor**, **StudentPriority**, **AdvisingCenter** and **ModelAdvisingCenter**. **ModelAdvisingCenter** contains the main method.

In the descriptions below I do not include the access level of the variables and methods but leave that to you. Remember that encapsulation principles recommend the use of private for instance variables, while static variables are usually public, and you should follow these recommendations. I also do not provide any of the import statements you may need.

The next pages describe each of the classes.

P3 – AdvisingType – 3 points

P3 – FullTimeAdvising – 3 points

P3 – PartTimeAdvising – 3 points

P4 – Student – 15 points

P5 – Senior – 4 points

P5 – JuniorSoph – 4 points

P6 – Freshman – 4 points

P7 – Advisor – 10 points

P8 – StudentPriority - 6 points

P9-11 – AdvisingCenter – 50 points (10 for initializing routines, 20 for running the site, 20 for outputs)

P11 – ModelAdvisingCenter - 10 points

Page 12 has the Sample interactive input I have used. Use those numbers to compare your results with mine.

**abstract class AdvisingType**

abstract class **AdvisingType** has the following instance variable:

String advisingType

There is a single **constructor** with a single parameter with the advisingType which

Calls **setAdvisingType** with advisingType as the parameter.

This class has both a standard **mutator** and **accessor** method for advisingType:

There are two abstract methods:

**setAdvisingNumber** has no parameters and a void return value

**getAdvisingNumber** which has no parameters and returns a String

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**class FullTimeAdvising is a subclass of AdvisingType**

Has a static int variable fullTimeCounter initialized to 0.

Has an instance variable String advisingNumber

Has a single constructor with no parameters which

Calls super with the parameter value “FullTime ”

Calls setAdvisingNumber with no parameters

Overrides the method **setAdvisingNumber**:

Increments fullTimeCounter by 1

Sets the advisingNumber equal to the concatenation of “FullTime ” and fullTimeCounter

Overrides the method **getAdvisingNumber**

returns the reference to the String referred to by advisingNumber

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**class PartTimeAdvising is a subclass of AdvisingType**

Has a static int variable partTimeCounter initialized to 0.

Has an instance variable String advisingNumber

Has a single constructor with no parameters which

Calls super with the parameter value “PartTime ”

Calls setAdvisingNumber with no parameters

Overrides the method **setAdvisingNumber**:

Increments partTimeCounter by 1

Sets the advisingNumber equal to the concatenation of “PartTime ” and partTimeCounter

Overrides the method **getAdvisingNumber**

returns the reference to the String referred to by advisingNumber

**abstract class Student implements the Comparable interface for Student**

Has a static int variable idCounter set to 0 (Make this public)

Has int instance variables arrivalTimeAtAdvisingCenter, advisorStartTime, waitTime, meetingDuration, totalTimeAtAdvisingCenter

Has a AdvisingType instance variable advType

Has a String instance variable studentType

Has a single **constructor** with a parameter for arrivalTime which

Calls setArrivalTimeAtAdvisingCenter with the arrivalTime from the constructor as the parameter

There are simple **accessor** (getXXX) methods for each of the **seven** instance variables

The mutator method for **setArrivalTimeAtAdvisingCenter** accepts an int parameter and has a void return

It sets the arrivalTimeAtAdvisingCenter equal to the input parameter value

The mutator method for **setMeetingDuration** accepts an int parameter and has a void return

It sets the meetingDuration equal to the input parameter value

The **setAdvisorStartTime** accepts an int with the advisorStartTime as the parameter and has a void return

It sets the advisorStartTime equal to the input parameter

It sets the waitTime as the difference between the advisorStartTime and the arrivalTimeAtAdvisingCenter

The **setTotalTimeAtAdvisingCenter** method accepts an int with the endTime as a parameter and has a void return

It sets the totalTimeAtAdvisingCenter to the difference between the endTime and the arrivalTimeAtAdvisingCenter

The **setAdvisingType** method accepts a reference to an AdvisingType as the parameter and has a void return

It sets advType to the parameter (a reference variable)

The **setStudentType** method accepts a String reference to the studentType as the parameter and has a void return

It sets studentType to the parameter (a reference variable)

There are the following 3 **abstract** methods:

**setStudentID** has no parameters and a void return

**getStudentID** has no parameters and returns a String

**compareTo** has a reference to a Student as a parameter and returns an int (this is the method from Comparable, but it is given a body in each of the subclasses)

**class Senior is a subclass of Student**

Has a String instance variable studentID

Has a single **constructor** with an int parameter indicating the arrivalTime, and a AdvisingType parameter advT referring to an AdvisingType

Calls super with arrivalTime as the parameter

Calls setStudentType with “Senior ” as the parameter

Calls setAdvisingType with advT as the parameter

Calls setStudentID with no parameters

There is a normal accessor getStudentID that overrides the abstract version of this method

**setStudentID** has no parameters and a void return – this is an override of the abstract method setStudentID

It increments the static variable idCounter declared in Student

It sets studentID to the concatenation of “CSUDH ”, the studentType, and the Student.idCounter

Note: a studentID might look like this **CSUDH Senior 106**

**compareTo** has a reference to a Student as a parameter and returns an int - this is an override of the abstract method in Student

Returns an int based on a reversal of the natural order for the String studentType. (i.e. Senior will precede JuniorSoph which precedes Freshman.)

Note: compareTo will be used in StudentPriority

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**class JuniorSoph is a subclass of Student**

Has a String instance variable studentID

Has a single **constructor** with an int parameter indicating the arrivalTime, and a AdvisingType parameter advT referring to a AdvisingType

Calls super with arrivalTime as the parameter

Calls setStudentType with “JuniorSoph ” as the parameter

Calls setAdvisingType with advT as the parameter

Calls setStudentID with no parameters

There is a normal accessor getStudentID that overrides the abstract version of this method

**setStudentID** has no parameters and a void return – this is an override of the abstract method setStudentID

It increments the static variable idCounter declared in Student

It sets studentID to the concatenation of “CSUDH ”, the studentType, and the Student.idCounter

Note: a studentID might look like this **CSUDH JuniorSoph 32**

**compareTo** has a reference to a Student as a parameter and returns an int - this is an override of the abstract method in Student

Returns an int based on a reversal of the natural order for the String studentType. (i.e. Senior will precede JuniorSoph which precedes Freshman.)

Note: compareTo will be used in StudentPriority

**class Freshman is a subclass of Student**

Has a String instance variable studentID

Has a single **constructor** with an int parameter indicating the arrivalTime, and a AdvisingType parameter advT referring to a AdvisingType

Calls super with arrivalTime as the parameter

Calls setStudentType with “Freshman ” as the parameter

Calls setAdvisingType with advT as the parameter

Calls setStudentID with no parameters

There is a normal accessor getStudentID that overrides the abstract version of this method

**setStudentID** has no parameters and a void return – this is an override of the abstract method setStudentID

It increments the static variable idCounter declared in Student

It sets studentID to the concatenation of “CSUDH ”, the studentType, and the Student.idCounter

Note: a studentID might look like this **CSUDH Freshman 78**

**compareTo** has a reference to a Student as a parameter and returns an int - this is an override of the abstract method in Student

Returns an int based on a reversal of the natural order for the String studentType. (i.e. Senior will precede JuniorSoph which precedes Freshman.)

Note: compareTo will be used in StudentPriority

**class Advisor**

Has a static int variable advisorCounter set to 0

Has the boolean instance value isFree

Has the int instance values advisorIDNumber, totalStudentsSeenByAdvisor, timeRemainingForMeeting,

Has the reference variable assignedStudent of type Student

Has a single default **constructor**

Calls setAdvisorIDNumber with no parameters

Calls setIsFree with true as the value of the parameter

There are simple **accessor** (getXXX) methods for each of the **five** instance variables

**setAdvisorIDNumber** has no parameters and a void return

It increments the advisorCounter by 1

It sets the advisorIDNumber equal to the advisorCounter

**setIsFree** has a boolean parameter and a void return

It sets isFree to the value of the parameter

**setAssignedStudent** has a reference to a Student as the parameter and a void return

It sets the reference variable assignedStudent to the value of the parameter

It calls setIsFree with false as the parameter

**removeAssignedStudent** has no parameters and returns a reference to the assignedStudent

Create a temporary Student reference variable tempStudent equal to the current assignedStudent for this Advisor

Set the assignedStudent variable to null

Call setIsFree with the parameter true

Increment totalStudentsSeenByAdvisor by 1

Return the value in the tempStudent reference variable

**setTimeRemainingForMeeting** has an int parameter and a void return

set timeRemainingForMeeting to the value of the parameter

**decrementTimeRemainingForMeeting** has no parameters and a void return

It decrements timeRemainingForMeeting by 1

**toString** has no parameters and returns a String – (normal override)

String should look like this (of course the numbers will be different)

**Advisor 8 advises 42 students**

**class StudentPriority implements the Comparator interface for Student**

Note: The idea is that a Senior has higher priority than JuniorSoph who has higher priority than a Freshman.

This class has one method compare with two parameters, both of which are references to Student objects.

If the two Students are of the same studentType (you can use compareTo here), the one who arrived earlier has a higher priority. This means return a negative, 0 or positive value based on the difference in the arrivalTimeAtAdvisingCenter between the first and second Student.

Otherwise use the result of compareTo for the two Students to obtain the return value

class **AdvisingCenter**

This class has the following instance variables:

A Random variable randy

A PriorityQueue of type Student called waitingQ

An ArrayList of type Student called processedStudents - may be instantiated in variable declaration or in the constructor)

An array of type Advisor called advisors (remember this is an array, **not** an ArrayList))

A String variable deptName

An int variable currentTime set to 0

It has a single **constructor** with three parameters: String name, int seed, int numAdvisors

It simple sets deptName to name (You can do this without calling a mutator)

It creates the waitingQ object using a new instance of StudentPriority as the parameter, ensuring that waitingQ prioritizes according to StudentPriority rules

It creates a Random object called randy based on the seed

It calls createAdvisors with numAdvisors as the parameter

**createAdvisors** has a single int parameter called numAdvisors and a void return

It creates the advisor array with a length equal to the value of numAdvisors

For each reference in the array it creates an Advisor (Remember that the Advisor constructor has no parameters)

**openAdvisingCenter** has no parameters and a void return

*Note: This starts the clock at time 0 and runs for 10 minutes (i.e. 0<=currentTime<10) of loading up the line before the AdvisingCenter doors open and service begins. currentTime should have a value of 10 when this is finished*

for (currentTime = 0; currentTime < 10; currentTime++) //This simulates early arriving Students

Do **10** times

Generate a random number of type int from 1 to 30 (inclusive meaning all of 1 through 30 using randy) and use it below

if the random number is less than 5

Create and add a Senior to the waitingQ with the currentTime and a new FullTimeAdvising object reference as the parameters

else if the random number is less than 6

Create and add a Senior to the waitingQ with the currentTime and a new PartTimeAdvising object reference as the parameters

else if the random number is less than 14

Create and add a JuniorSoph to the waitingQ with the currentTime and a new FullTimeAdvising object reference as the parameters

else if the random number is less than 16

Create and add a JuniorSoph to the waitingQ with the currentTime and a new PartTimeAdvising object reference as the parameters

if the random number is less than 29

Create and add a Freshman to the waitingQ with the currentTime and a new FullTimeAdvising object reference as the parameters

else

Create and add a Freshman to the waitingQ with the currentTime and a new PartTimeAdvising object reference as the parameters

**operateAdvisingCenter** has a parameter int called durationForArriving and a void return

Create an int variable endArrivalsTime and set it equal to the sum of the currentTime and durationForArriving

Create any necessary temporary variables for this method i.e. tempStudent

**while** the size of processedStudents does not equal the Student.idCounter **or** the currentTime is less than the endArrivalsTime (note: all arrivals get processed this way)

if the currentTime is less than the endArrivalsTime //This simulates arriving Students until the office doors close

Do **8** times

Generate a random number of type int from 1 to 30 (inclusive meaning all of 1 through 30 using randy) and use it below

if the random number is less than 6

Create and add a Senior to the waitingQ with the currentTime and a new FullTimeAdvising object reference as the parameters

else if the random number is less than 7

Create and add a Senior to the waitingQ with the currentTime and a new PartTimeAdvising object reference as the parameters

else if the random number is less than 15

Create and add a JuniorSoph to the waitingQ with the currentTime and a new FullTimeAdvising object reference as the parameters

else if the random number is less than 18

Create and add a JuniorSoph to the waitingQ with the currentTime and a new PartTimeAdvising object reference as the parameters

if the random number is less than 29

Create and add a Freshman to the waitingQ with the currentTime and a new FullTimeAdvising object reference as the parameters

else

Create and add a Freshman to the waitingQ with the currentTime and a new PartTimeAdvising object reference as the parameters

End of the do loop

//Now check on the busy Advisors

for each Advisor referenced in the advisors array

if the Advisor isn’t free (i.e. getIsFree returns false)

Call decrementTimeRemainingForMeeting() for this Advisor

if the timeRemainingForMeeting for this Advisor is 0

Retrieve the reference to the Student currently assigned to this Advisor (use a tempStudent variable to capture the reference and the method getAssignedStudent())

Add that tempStudent to the processedStudents ArrayList

Call setTotalTimeAtAdvisingCenter for this tempStudent with the currentTime

Call removeAssignedStudent() for this Advisor

//Now Assign Students to free Advisors. Note: this section must come after the above section

For each Advisor referenced in the advisors array

if isFree is true for the Advisor (i.e. getIsFree returns true) and the waitingQ is not empty

Remove a Student from the waitingQ (use a tempStudent variable to capture the reference)

Call setAdvisorStartTime for tempStudent with the currentTime

For this Advisor call setAssignedStudent with a reference tempStudent just removed from the waitingQ

Use randy to help get a number from 10 to 20 minutes (inclusive) calling it meetingDuration

Call setMeetingDuration for the tempStudent with meetingDuration as the parameter

Call setTimeRemainingForMeeting for this Advisor with this meetingDuration as the value the parameter

Increment currentTime by 1. Note this happens inside the **while loop just before looping**

End the while (put in here for clarity of description only

**generateAdvisingCenterResults** has a String parameter representing the physical output file and returns a void (remember the throws IOException)

This method should write data to the output file. advisingouput.txt is a sample based on following my instructions and the interactive input below.

First create a PrintWriter based on the physical output file name in the method’s parameter list.

Print the title (i.e. Data For CSUDH Advising Center For Computer Science Department)

The three parts of this output are:

First is a summary of the number of Students processed by each Advisor.

The second part provides average total time (to two decimal places) by Student type, and then for all Students

You will need to separate the people in the processedStudents ArrayList (using instanceof) to collect the correct times. Remember for Students as a whole to aggregate the total amount of wait time over the total number of Students.

The third part is a table of all the Students. Create an iterator on the processedStudents ArrayList to do this. The table will have the headers above and corresponding values in columns.

STUDENT ID, STUDENT TYPE, ADVISING TYPE, ADVISING NUMBER, ARRIVAL TIME, WAIT TIME, MEETING TIME, TOTAL TIME

Note: The TotalTime column has the time for the Student at the ArrivalCenter (you have created a method to get this already)

Remember to close the PrintWriter

class **ModelAdvisingCenter** (throws an IOException)

Create a Scanner to read keyboard input

Request and read in the department name at the AdvisingCenter ( using nextLine()), a seed, and the number of Advisors

Create a new AdvisingCenter with those 3 values as parameters (see below)

Call openAdvisingCenter for the just created AdvisingCenter with no parameters

Request and read in the number of minutes to keep the site open for newly arriving Students (durationForArriving) after the doors open.

Call operateAdvisingCenter for the AdvisingCenter created with durationForArriving in minutes as the parameter

Request and read in the name of the output file (a String)

Call generateAdvisingCenterResults for the AdvisingCenter created with the name of the output file as a parameter

**Sample Interactive Input:**

Please enter the name of department name at the Advising Center at CSUDH:

Computer Science

Please enter a seed value as an int:

5

Please enter the number of advisors as an int:

40

Please enter the number of minutes to keep the Advising Center open:

120

Please enter the name of the output file for Advising Center results: advisingoutputs.txt