EECS 2030: Programming Exam 1
Fall 2021

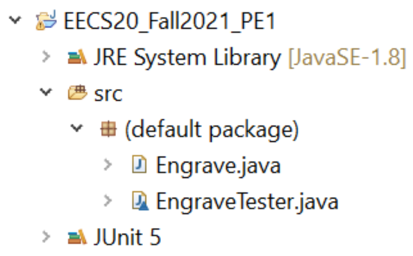
Deadline: Tuesday Oct 19th, 2021 11:59pm

In the first programming exam, we are going to assess your ability for the following skills:

- Ability to design recursive functions.
- Ability to test your code thoroughly, through Junit tests.
- Ability to document your code.

Setting up the environment:

- Download the attached zip file.
- Open eclipse.
- Click on File and select Import
- Choose Existing Projects into Workspace and click Next
- Click on Select Archive File and then Browse.
- Find the zip file that you have already downloaded and click Finish
- Please make sure that you do not already have a project called EECS20_Fall2021_PE1
- Now you should see the following settings in the package explorer.



```
EECS20_Fall2021_PE1
  JRE System Library [JavaSE-1.8]
  src
    (default package)
      Engrave.java
      EngraveTester.java
  JUnit 5
```

Before you start, I would like to remind you that this assignment is different from the labs that you have done until now, as this assignment is an **exam** that you take home. This means you can neither discuss your solution with anybody including your peers nor get help from them. Therefore, please carefully read the statement at the top of `Engrave.java` before you start doing the assignment. When you are finished you will be needed to sign this statement by writing your information (full name and student number) in the gap provided. Please be aware that your assignment WILL NOT be marked if you do not sign the declaration. Also, I would like to inform you that your code will be checked by both a plagiarism detection tool and us to make sure your solution to this problem is unique and not similar to others'.

Refusing to follow the policy stated above, is a violation of academic integrity that will have a serious consequence. Please read about the definition of academic integrity and the penalty that one may face, in case they violate academic integrity in https://lassonde.yorku.ca/student-life/academic-support/academic-honesty-integrity .

Problem Description:

There is a machine shop that is specialized in engraving measures on measuring cups. A measuring cup enters the machine, and the measuring unit is printed on the cup. The cups are made from different materials such as stainless steel, glass, wood and so on. The user programs the machine by pushing several buttons. One button is to specify the material with which the cup has been made. Another button is used to command the machine on the types of the unit that should be printed. For example, pressing the button once, engraves the units in ml (milli litre), and pressing it twice draw the units in Pint and so on.

The picture has been taken from https://istockphoto.com

Your job for this assignment is to write a program for this button when it is pressed once. In other words, we are interested in engraving the ml measuring unit on the cups.

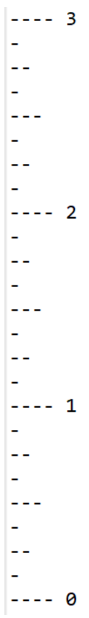
When the user selects to print ml measuring unit on the cup, the system requires some more information to customize the printing to what the user desires.

One information that is required is the size of the cup, which is expressed by the capacity of the cup. This machine is able to engrave on the cups whose capacity is greater than (or equal to) 100 ml.

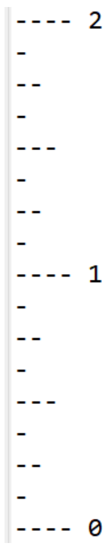
The second information that is required is the length of the line that is drawn next to the number.
To better understand how the system works, see the examples below.

The first one from the left is printed on a cup with the capacity of 300 ml. The user entered 4 for the length of the lines next to the numbers. The second is designed for 200 ml cup with length 4. The third is the design of 200 ml cup with length 2 and finally the last one is for 100 ml cup with length 5.

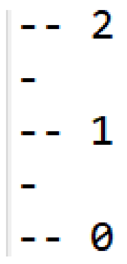
As you have probably noticed, the machine draws the intervals between the major lines (i.e., the lines which are next to the numbers). For example, if you look at the fourth example, you will see the length of the major line is 5. The line that indicates 50 ml has the length of 4. The lines that indicate 25 and 75 ml has the length of 3 and so on. This pattern continues to the point where a line of length one is drawn. This problem is easily solved using recursion. I am going to walk you through the steps to help you write the code for this specific button of the machine.



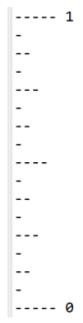
```
---- 3
-
--
-
---
-
--
-
---- 2
-
--
-
---
-
--
-
---- 1
-
--
-
---
-
--
-
---- 0
```



```
---- 2
-
--
-
---
-
--
-
---- 1
-
--
-
---
-
--
-
---- 0
```



```
-- 2
-
-- 1
-
-- 0
```



```
----- 1
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
----- 0
```

Task 1:

The first method that I suggest you implement **recursively** is a method called `drawLine`. This method draws a line using dash (-) character [i.e., minus sign]. The first input of the method specifies the length of the line, and the second input shows the label that should be printed next to the line. Please note that we ignore the zeros in front of the numbers for simplicity. For example, number 3 means 300 ml.

```
public void drawLine (int tickLength, char tickLabel)
```

Sample Call:

```
    drawLine (5, '3');
```

Corresponding Output:

```
    ----- 3   // please note that there is a blank between the last dash and number 3
```

Sample Call:

```
    drawLine (6, ' ');
```

Corresponding Output:

```
    ------
```

The `engrave` class has an instance variable called `charPrintCount` that holds the number of characters that is printed every time that drawLine is called. Therefore, the value of `charPrintCount` will be `7` after `drawLine(5, '3')` is called, and will be `8` after `drawLine(6, ' ')` is called. Remember that a space is always printed between the last dash and the label.

Marking Scheme:

- [20 points]: For the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.
- [5 points]: For the javaDoc, inner documentation if required, and code style.

Task 2:

The second **recursive** method is called `drawIntervals` and prints the intervals between two major lines. The input to this method is length of the major lines. This method requires to call the method you have written for task 1 to draw the lines.

```
public void drawIntervals(int tickLength)
```

Sample Call:

```
drawIntervals(2);
```

Corresponding Output:

```
-
--
-
```

Sample Call:

```
drawIntervals(3);
```

Corresponding Output:

```
-
--
-
---
-
--
-
```

Marking Scheme:

- [20 points]: For the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.
- [5 points]: For the javaDoc, inner documentation if required, and code style.

Task 3:

At this point, you have implemented the most difficult part. For this task, you are supposed to implement `engrave` method, which is the **recursive** method that is called when the button is pressed. This method gets two input parameters. The first, `tickLength`, is the number of dashes that is printed next to the numbers. This is previously referred to as the length of the major lines. The second attribute, `magnitude`, points to the capacity of the cup. So, if magnitude is equal to 3, it means the capacity of the cup is 300 ml. The magnitude of a cup cannot be a negative number.
This method calls the other two methods that you have implemented in task 1 and task 2.

```
public void engrave(int tickLength, int magnitude)
```

Sample Call:

```
engrave(4, 3);
```

Corresponding Output:

The left most picture in page two of this document.

Sample Call:

```
engrave(5, 1);
```

Corresponding Output:

The right most picture in page two of this document.

Marking Scheme:

- [20 points]: For the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.
- [5 points]: For the javaDoc, inner documentation if required, and code style.

Task 4:

The class has an instance variable called *`drawnObject`*, which is filled by the number of dashes that is printed in each line. This variable is an arrayList that can store an unlimited number of elements in itself. Every time that your code prints a line, the number of printed dashes [and the label if applicable] should be stored in the arrayList. If the line is labeled with a number, the number of dashes will be followed with a comma, a space and the number that is printed. For example, for the fourth example above [i.e., engrave(5, 1)], the arrayList contains the following elements. As you see at index zero "5, 1" is stored, which means the first line that is printed, contains 5 dashes followed by number 5. At index 1, number 1 is stored, which means only one dash is printed in the second line.

```
[5, 1] [1] [2] [1] [3] [1] [2] [1] [4] [1] [2] [1] [3] [1] [2] [1] [5, 0]
```

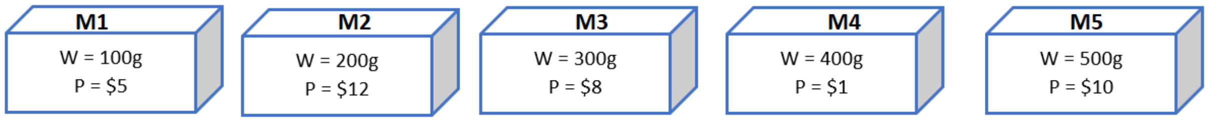
PLEASE NOTE: we will NOT look at your output to check the correctness of your code, as the code is graded by JUnit test cases. Therefore, it is important that for all the tasks above, you make sure that the *drawnObject* List is filled in correctly.

Hint: `drawnObject` gets its element in `engrave` and `drawIntervals` methods only. The `drawLine` method does NOT fill up the array.

Task 5:

Now that the products are ready, we are going to pack them and send them to the vendors. The measuring cups have made from different materials (e.g., wood, glass, plastic, stainless steel, etc.) and therefore have different `weight` and `price`.

The boxes that we have available for packing can tolerate a certain weight `w`. It is desired to put a number of measuring cups into the box so that the total weight of the cups <= `w` and the total price is maximised. For example, suppose that the box that we have can tolerate 600g and we have 6 measuring cups as below.



All the possible combinations of the measuring cups, which can fit the box is as below:

- [M1, M5] price $15
- [M2, M4] price $13
- [M1, M2, M3] price $25

The proper selection for our purpose is M1, M2 and M3 because not only the weight <= 600, but also it contains the most expensive cups amongst all the cups.

In this packaging, we do not want to have two items from the same type. Therefore, selecting 3 M2 cups is not accepted, however it worth more than other selections.

The method that solves this problem has 4 inputs. The first input is an array of integer that holds the weight of each cup. The second input is also an array of int that holds the price of the cups. These two arrays are parallel, which means index `n` of the first array pertains to the same index of the second array. Thus, the length of both the arrays is the same.

The third input is the highest weight that the selected cup(s) can have. Finally, the fourth input is the array index. At first call, index holds the length of the arrays.

Hint: `maxWeight` and `index` may change in every method call.

```
public static int cupSelection(int weight[], int value[], int maxWeight, int index)
```

The return value of this function is the total price of the selection. This method should also be implemented **recursively**.

Sample Call:

```
int value[] = new int[] {60, 120, 100 , 100, 30 , 20};
int weight[] = new int[]  {10, 30,  20  , 20,  30 , 10};
cupSelection(weight, value,50,  6)
```

Corresponding Output:

```
260
```

Sample Call:

```
int value[] = new int[] {60, 50, 20};
int weight[] = new int[]  {30,  30 , 10};
System.out.println(cupSelection(weight, value,50,  3));
```

Corresponding Output:

```
80
```

Marking Scheme:

- [20 points]: For the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.
- [5 points]: For inner documentation.

Checklist:

Before you submit, check this checklist to ensure that you have done all we asked you to do.

- ☐ I have finished tasks 1, 2,3, 4 and 5.
- ☐ I have signed the statement on academic honesty.
- ☐ I have removed/commented out the `main()` method, if applicable, to ensure that I have tested my method with the sample tests provided and my methods signature conforms with the tester's requirement.
- ☐ drawObject arrayList is filled in correctly with my output.
- ☐ I have tested my code with more JUnit test cases, which I have written myself as I am aware that a small sample of test cases have been provided.

What and how to Submit:

- You only submit one file, which is the completed Engrave.java file.
- Submit your solution in eClass, where you have downloaded these instructions.