CE152 Assignment Spring 2017

WARNING AND ADVICE ABOUT POSSIBLE ACADEMIC OFFENCES

Your solutions should be your own unaided work. You can make use of any of the programs from the CE152 lecture notes and the lab solutions. You may use any features from the Java SE API including those not covered in CE152.

You must NOT use any third-party classes (e.g. classes that are not provided as part of the Java SE download). If you use any other sources, you must clearly indicate this as comments in the program, and the extent of the reference must be clearly indicated. For more information, please see the University pages on plagiarism (https://moodle.essex.ac.uk/course/view.php?id=5844) and the Academic Offences Procedures (http://www.essex.ac.uk/about/governance/documents/policies/academic-offences-2016-17.pdf).

DO NOT COPY PROGRAM CODE FOR THIS ASSIGNMENT FROM ANOTHER STUDENT OR FROM THE INTERNET OR FROM ANY OTHER SOURCES. DO NOT LET OTHER STUDENTS COPY YOUR WORK.

Deadline: See Faser (https://www.essex.ac.uk/e-learning/tools/faser)

Frequently Asked Questions (FAQ) (./ce152assFAQ.html)

Submission

The assignment should be submitted via Faser (https://www.essex.ac.uk/e-learning/tools/faser). Your submission should comprise a single zip file containing the source code (i.e. the .java files) for all the classes that you have written as solutions to the assignment tasks. *No other files should be included in the zip file.*

The name of your zip file should include both your name and your registration number.

You will receive a **mark of zero** if you fail to submit your solutions by the deadline.

Demonstration

You will be required to demonstrate your solutions to the assignment tasks in Week 30 (i.e. shortly after the submission deadline). An assignment demonstration schedule will be published closer to that time.

You will receive a **mark of zero** if you fail to demonstrate your work at the allocated time.

Extenuating Circumstances

The standard extenuating circumstances procedures will apply for those who - for circumstances beyond their control - are prevented from submitting work before the deadline or from attending the lab demonstration. Please see the Undergraduate Students' Handbook for the University policies regarding these matters.

---

1. Exercise [20%]

A pizza delivery service sells pizzas in two different sizes: **m**edium and **l**arge. The prices for a basic pizza (without any toppings) are £4 (medium) and £5 (large). Customers can choose up to four toppings from the list below. Toppings are optional - it should also be possible to order a basic pizza without any toppings.

---

	medium	**l**arge
ham	£1.40	£2.10
mozzarella	£1.00	£1.50
olives	£0.80	£1.20
pineapple	£1.00	£1.50
spinach	£0.80	£1.20

Your task is to write a program that helps staff members to input a pizza order and which works out the price. The size of the pizza and the toppings are to be specified by initial letters.

Part A [15%]

Write a class `Exercise1` with a method

```
public static void pizzaServiceA()
```

The method should feature a loop. In each step of the loop, the program should

- ask the staff to enter a pizza order, and then
- display the order details as well as the pizza price.

Pizza orders should be entered as a sequence of up to 5 characters all on one line:

- The first character should be either `m` (meaning medium) or `l` (meaning large).
- This should be followed by zero to four characters indicating the pizza toppings. Each of these characters should be the initial character of one of the toppings. Multiple occurrences of the same topping should be allowed and charged accordingly.

For example, if the staff member inputs

```
mph
```

the program should respond with output as follows:

```
Medium pizza with pineapple, ham, £6.40
```

Multiple occurrences of toppings should be listed multiple times in the output, for example if the staff inputs

```
lsms
```

the program should respond with

```
Large pizza with spinach, mozzarella, spinach, £8.90
```

If the order has no toppings, this should be stated in the output, for example if the staff inputs

```
m
```

the program should respond with

```
Medium pizza with no toppings £4.00
```

Please format the price output so that it shows 2 digits following the decimal point.

The program should check that the user input is valid. If the user input is invalid, the program should display an informative message. It should not show any order details or the price in this case.

The program should terminate if the user enters the word **`quit`**.

Add a `main()` method to your class which invokes method `pizzaServiceA()`. Test your program.

Part B [5%]

In class `Exercise1`, add a method

```
public static void pizzaServiceB()
```

The method should implement the same functionality as in Part A, but with one difference: it should not be allowed to place orders where one topping occurs three or more times. For example, the following order should be invalid because spinach occurs three times:

```
msmss
```

The program should print an informative message in this case.

Please note that orders where a topping occurs twice should still be allowed, so for example the following is a valid order:

```
lsoos
```

and the program output should be

```
Large pizza with spinach, olives, olives, spinach, £9.80
```

Add an invocation of `pizzaServiceB()` to the `main()` method. Test your program.

2. Exercise [10%]

Write a class `Exercise2` with a method

```
public static int[] closestToMean (double[][] array)
```

Given a rectangular 2D array, the method should first compute and print the arithmetic mean of all elements. It should then find the array element which is closest to the mean and print this. The method should return the row index `r` and the column index `c` of that array element in form of a two-element integer array `[r,c]`.

For example, if the argument array has the following elements

```
 3 -1 -4  0
 5 -2  9  6
 8  2  4 -9
```

then `mean=1.75`. The array element closest to `mean` is the number `2`, it has distance `0.25` from the mean. The method should print:

```
Mean = 1.75
Closest array element = 2
```

The closest array element is in row `2` and column `1`, hence the method should return an array with elements `[2,1]`. Note that counting of rows and columns starts at zero as usual in Java programming.

Hint. The "distance" of two numbers `x` and `y` can be computed in Java as `Math.abs(x-y)`.

If there is more than one element with the minimal distance to the mean, then the method can return the row and column indices of any of these elements.

In class `Exercise2`, add a method

```
public static void testClosestToMean()
```

This method should invoke `closestToMean()` for the sample array shown above. It should display the result.

In class `Exercise2`, add a `main()` method which invokes `testClosestToMean()`. Check that the result is correct.

3. Exercise [15%]

Part A [8%]

The Catalan numbers (https://en.wikipedia.org/wiki/Catalan_number) form a sequence of natural numbers that occurs in various combinatorial problems. They can be computed using the following two equations:

```
catalan(n) =  2 * (2 *n - 1) * catalan(n - 1) / (n + 1), if n > 0
catalan(0) = 1
```

Create a class `Exercise3` with a method

```
public static long catalan(int n)
```

for computing the n'th Catalan number using the equations above. Your method should check the argument `n` and throw an `IllegalArgumentException` (https://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html) if `n<0` or `n>30`.

In class `Exercise3`, write a method:

```
public static void exercise3a()
```

The method should contain an interactive loop. In each step of the loop, the program should:

- Ask the user to enter an integer `n` with `0<=n<=30` or quit to exit;
- Check whether the user input is `quit` and exit the loop if this is the case
- Otherwise parse the input as an integer `n`, invoke `catalan(n)` and display the result to the user

Method `exercise3a()` should not do any exception handling. You can assume that the user input is either `quit` or an integer.

Add a `main()` method to your class which invokes `exercise3a()`. Test your program both with some valid and some invalid integer inputs.

Part B [4%]

In class `Exercise3`, code a method:

```
public static void exercise3b()
```

This method should perform the same task as `exercise3a()` above but with one difference - if the invocation of `catalan` throws an `IllegalArgumentException`, then it should catch this exception, print a suitable error message, and continue with the interactive loop.

Add an invocation of `exercise3b()` in the `main()` method. Test your program both with some valid and invalid integer inputs.

Part C [3%]

Write a class `TestCatalan` with one or more JUnit tests that check that method `catalan(n)` returns the correct result for six different arguments `n`. Restrict your testing to valid integer arguments.

Run `TestCatalan` in JUnit and check that your `catalan` function passes all the tests.

Note. JUnit testing will be covered during Week 24. You can find a list with some Catalan numbers at the On-Line Encyclopedia of Integer Sequences (https://oeis.org/A000108)

4. Exercise [25%]

The following online folder contains some mazes stored in separate files:

```
https://orb.essex.ac.uk/ce/ce152/data/assign/mazes (https://orb.essex.ac.uk/ce/ce152/data/assign/mazes)
```

You should make copies of these three files in a convenient directory on your M drive. Then take a look at the files. Each of them contains a quadratic maze with walls indicated by a #-symbol. As an example, here is file maze21.txt (https://orb.essex.ac.uk/ce/ce152/data/assign/mazes/maze21.txt):

```
#####################
#     #       #     #
### # # # ### ### # #
#   # # #   #   # # #
# ### # ### ### # ###
# #   # #   #   #   #
# # ### # ### ##### #
#   #   #   # #     #
# ### ##### # # ### #
#   #     # # # # # #
### ### # # ### # # #
#   # # # #     # # #
# # # # ### ##### # #
# #   #   # #   # # #
# ### ### ### # # # #
#           #   # # #
# ########### ### # #
# #   #   #   #   # #
### # # # ##### # # #
#   #   #       #   #
#####################
```

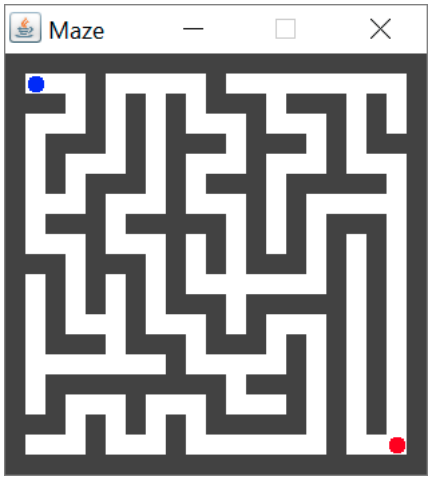
Part A [7%]

Write a class `Exercise4` with a program that reads such a maze file into a two-dimensional boolean array. Then display the array on the console with one line for each row. Represent array elements using blank-symbols and #-symbols so that the console output has the same format as the maze file, see the example above.

You are free to add further classes or methods as you see fit.

Part B [8%]

Expand the program you wrote in Part A so that it displays the maze using Java2D graphics. Also draw a blue circle in the top-left corner so as to indicate the starting-point of the maze and a red circle in the bottom-right corner to indicate the target-point. The format should be similar to the image below:



Part C [10%] (Challenge)

Expand the program you wrote in Part B so that the user can traverse the maze using the arrow keys on the keyboard. The current position should be indicated by a filled blue circle. The keyboard control should work as follows:

- Right-arrow: move blue circle to the right if possible.
- Down-arrow: move blue circle down if possible.
- Left-arrow: move blue circle to the left if possible.
- Up-arrow: move blue circle upwards if possible.

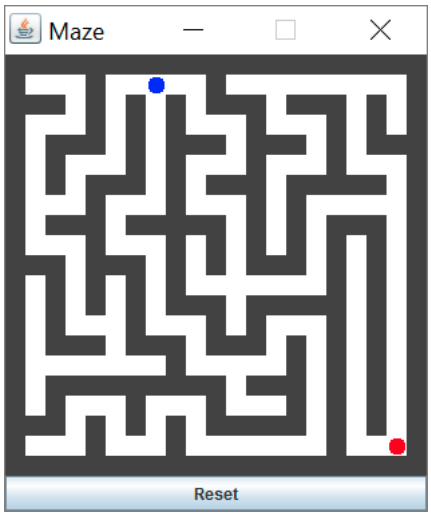
The walls of the maze should be respected when traversing the maze, - it should not be possible for the blue circle to go across a wall.

The GUI should also have a `Reset` button, see image below. Pressing this button should reset the current position and the blue circle to the initial position in the top left corner.

The program should display a congratulation message when the user reaches the red circle in the bottom-right corner. Use a JOptionPane to show this message, see this documentation (http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html)

Hints.

Don't forget to invoke method `repaint()` after making changes to the GUI. Also make sure that the component with the key listener gets the focus, see the Key Listener tutorial (https://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html).



5. Exercise [30%]

The online CSV (https://en.wikipedia.org/wiki/Comma-separated_values) file

```
https://orb.essex.ac.uk/ce/ce152/data/assign/hills.csv (https://orb.essex.ac.uk/ce/ce152/data/assign/hills.csv)
```

contains data about several thousands British and Irish hills.

You should make a copy of this file in a convenient directory on your M drive. Then take a look at the file. Each row describes a hill in the following format:

Number	**Name**	**County**	**Height** (in metres)	**Latitude**	**Longitude**
255	Ben Nevis	Highland	1344.5	56.796849	-5.003525

In each row, the six fields are separated by commas. You can assume that there are no other commas in the CSV file.

Part A [7%]

Create a class `Hill` that represents the data for one hill as in file `hills.csv`. Each `Hill` object should have six fields corresponding to the columns in that file:

- An integer **number**
- A **name**
- A **county** name

- The **height** (in metres)
- The **latitude**
- The **longitude**

The class should also have:

- A constructor with six arguments for initialising the six fields.
- An instance method `toString()` which returns a string in the same format as the rows in the CSV file:
  ```
  255,Ben Nevis,Highland,1344.5,56.796849,-5.003525
  ```

Then create a class `Exercise5` with the following method:

```
public static void exercise5a() {
  Hill benNevis = new Hill(255, "Ben Nevis", "Highland", 1344.5, 56.796849, -5.003525);
  System.out.println(benNevis);
}
```

Add a `main()` method to class `Exercise5` that invokes `exercise5a()`. Test your program by running the class.

Part B [6%]

In class `Hill`, write a method

```
public static List<Hill> readHills()
```

The method should read the hill data from file `hills.csv`. For each row, it should create a corresponding `Hill` object. The method should return a list containing these objects in the same order as they are in the file.

In class `Exercise5`, create a method:

```
public static void exercise5b()
```

This method should invoke `readHills()` and display the first 20 list elements on the console.

Add an invocation of `exercise5b()` in the `main()` method and test your program.

Hints:

- See Week 19 lecture notes for sample code for reading data from a CSV file.
- Remember that classes `Double` and `Integer` have methods to parse a number from a string.

Part C [5%]

In class `Exercise5`, write a method

```
public static void exercise5c()
```

The method should read the hill data from file `hills.csv` into a list using method `readHills()`. It should then start an interactive loop. In each step of the loop, the program should:

1. Request the user to enter the name of a hill.
2. Respond by printing to the console the details of all matching hills. A hill is said to be matching if its name **starts** with the string entered by the user. Matching should be case-insensitive.

The program should exit if the user enters `quit`. See a sample interaction below:

```
Please enter a hill name or quit to exit:
Cadair
1862,Cadair Berwyn,Powys,832.0,52.8806,-3.380993
1863,Cadair Berwyn North Top,Denbighshire,827.0,52.883881,-3.380235
1865,Cadair Bronwen,Denbighshire,783.4,52.901461,-3.372899
1868,Cadair Bronwen NE Top,Denbighshire,700.0,52.906631,-3.358802
1911,Cadair Idris - Penygadair,Gwynedd,893.0,52.699618,-3.908792
4587,Cadair Fawr,Rhondda Cynon Taff,485.0,51.800059,-3.483636
12140,Cadair Ifan Goch,Conwy,207.0,53.185429,-3.810762
Please enter a hill name or quit to exit:
sno
1742,Snowdon - Yr Wyddfa,Gwynedd,1085.0,53.068496,-4.076231
10028,Snougie of Long Hill,Shetland Islands,75.0,60.259782,-1.208551
13597,Snokoe Hill,Northumberland,191.0,54.953663,-2.028509
13802,Snoddle Hill,Rochdale,277.0,53.66063,-2.073073
Please enter a hill name or quit to exit:
quit
Good-bye
```

Add an invocation of `exercise5c()` in the `main()` method and test your program.

Part D [6%]

Write a method

```
public static void exercise5d()
```

The method should read the hill data from file `hills.csv` into a list using method `readHills()`. It should then:

1. sort the list of hills by name in alphabetical order and print the first 20 elements of the list; and then
2. perform another sorting of the list of hills but this time by height in descending order (=highest hills first) and then print the first 20 elements of the list.

Sorting should be performed by invoking method Collections.sort (https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#sort-java.util.List-) making use of appropriate implementations of interfaces `Comparable<Hill>` and/ or `Comparator<Hill>` which you will need to code.

Part E [6%]

In class `Hill`, write a method

```
public static Map<String, Set<Hill>> hillsByCounty(List<Hill> hills)
```

Given a list of `Hill` objects, the method should return a map which associates each county with the set of hills in that county. The map entries should be sorted by county names and each of the sets of hills should be sorted by hill names.

Then write a method:

```
public static void exercise5e()
```

The method should read the hill data from file `hills.csv` into a list using method `readHills()`. It should invoke method `hillsByCounty` with that list. This method invocation returns a map. For each of the first three counties in that map, the program should display:

- The name of the county
- The names and the height of the first three hills associated with the county in the map.

This should result in program output as follows:

```
### County: Aberdeen
Anguston Hill 117.0
Beans Hill 146.0
Brimmond Hill 266.0
### County: Aberdeenshire
A'Chioch 1151.0
Aitionn Hill 273.0
Allt Sowan Hill 568.0
### County: Anglesey
Barclodiau 168.0
Bron Alar 64.0
Bryn Carmel 116.0
```

Assignment Marking Criteria

Exercise	Weighting	Criteria
1A	15%	5 marks for correct pizza prices, 3 marks for display of correct order details, 2 marks for correct formatting of price, 5 marks for proper validation of user input with informative error messages. Partial marks for partially correct solutions.
1B	5%	5 marks for correct behaviour. No partial credits.
2	10%	3 marks for computing the correct mean, 3 marks for computing the closest element, 3 marks for returning the correct result. 1 mark for correct method `testClosestToMean()`. No partial credits.
3A	8%	3 marks for computing correct `Catalan` numbers, 2 mark for correct throwing of exception, 3 marks for correct interactive loop in method `exercise3a`. No partial credits.
3B	4%	2 marks for correct catching of exception. 2 marks for correct handling of the caught exception. No partial credits.
3C	3%	0.5 marks for each check of the correctness of `catalan(n)` for a different n using JUnit.
4A	7%	5 marks for correct reading of maze from file into a boolean 2D array and printing its elements to the console, 2 marks for proper formatting of output.
4B	8%	6 marks for correct maze display, 2 marks for correct display of start point and end point.
4C	10%	3 marks for movement of blue circle in line with keyboard control. 2 marks for respecting maze walls. 3 marks for proper implementation of the reset button. 2 marks for showing a dialog box when the target is reached. No partial credits.
5a	7%	3 marks for a class definition with suitable fields, 2 marks for a correct constructor, 2 marks for a correct `toString()` method.
5b	6%	5 marks for correct implementation of method `readHills`. 1 mark for correct testing in method `exercise5b().`
5c	5%	3 marks for displaying correct results, 2 marks for correct behaviour of the interactive loop.
5d	6%	2 marks for correct sorting by hill name, 2 marks for correct sorting by hill height in descending order, 2 marks for correct testing in method `exercise5d()`. No partial credits.
5e	6%	2 marks for method `hillsByCounty` returning a map with correct keys and with value sets containing the correct hills. 1 mark for correct order of map entries. 1 mark for correct order of hills in sets. 2 marks for correct testing in method `exercise5e()`. No partial credits.

Acknowledgements: The hill data was taken from the "Database of British and Irish hills", see http://www.hills-database.co.uk/ (http://www.hills-database.co.uk/)