University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

AUTUMN TRIMESTER EXAMINATIONS

ACADEMIC YEAR 2021/2022

COMP20250: Introduction to Java

COMP20300: Java Programming (Mixed Delivery)

Exam Part A and B

Assoc Prof. Chris Bleakley
Dr. Simon Caton

Time Allowed: 3 Hours (+1 hour for video recording)

Answer one question, from Part A (either 1 or 2) and one question from Part B (either 3, 4, or 5).

The ExamResources.zip file has the following contents:

- **Computer.java** (the answer box preload for Q1) and the corresponding test cases: **Q1Tests.java**
- **Team.java** (the answer box preload for Q2) and the corresponding test cases: **Q2Tests.java**
- **Q3Tests.java** – the test cases for Q3
- **Question4.java** (the answer box preload for Q4 – the class you need to complete) and the corresponding test cases: **Q4Tests.java**
- Two input files for Q5: adult.txt (contains well formed input) and adult-error.txt (contains examples with well and badly formed input) and the corresponding test cases: **Q5Tests.java**

Summary of questions:

- **Question 1** focuses on interfaces and inheritance. You will prepare several classes to demonstrate these concepts.
- **Question 2** focuses on class design. You will prepare several classes illustrating key aspects of class design exercised through a logical reasoning exercise.
- **Question 3** focuses on outputting two String-based shapes according to specific input parameters, e.g.:

```
......x......
.....xxx.....
....xxxxx....
...xxxxxxx...
..xxxxxxxxx..
.xxxxxxxxxxx.

xxxxxxxxxxxxx                    ..x......x..
x..x..x..x..x                    ...x....x...
x..x..x..x..x                    x...x..x...x
xxxxxxxxxxxxx                    .x...xx...x.
x..x..x..x..x                    ..x..xx..x..
x..x..x..x..x                    ...xx..xx...
xxxxxxxxxxxxx                    ...xx..xx...
x..x..x..x..x                    ..x..xx..x..
x..x..x..x..x                    .x...xx...x.
xxxxxxxxxxxxx                    x...x..x...x
x..x..x..x..x                    ...x....x...
x..x..x..x..x                    ..x......x..
xxxxxxxxxxxxx
```

- **Question 4** focuses on array and ArrayList manipulation, with elements of searching arrays and fundamental array operations
- **Question 5** focuses on reading text files and using these to construct Objects

Question 1

Answer parts (a) to (f) with solutions coded in Java, and answer part (h) via video recording and demonstration.

35 marks in total

In this question, you will prepare the following classes / interfaces:

- The interface Computer in (a)
- The class CPU in (b)
- The class GPU in (c)
- The class Workstation in (d)
- The class Laptop in (e)
- Use the provided interface Factory in (f)

(a) (2 marks)

Create an **interface** called **Computer** it should have the following methods:

- **getRAM** which returns the amount of RAM (as an **int**)
- **getHDSpace** which returns the amount of hard disk space (as an **int**)
- **getCPUs** which returns an **ArrayList** of **CPUs** (see below)
- **getDevices** which returns an **ArrayList** of input / output devices (as **Strings** representing the name of the device).

The test case for this question is called: **testComputer**. It tests the signature of the method.

(b) (3 marks)

Create a **class** called **CPU** that has the following instance variables (this class is not immutable so add appropriate methods for the instance variables, and appropriate constructor(s)):

- clock speed (a **double**)
- type (a **String**)
- num cores (an **int**)

Also add a **toString** method that produces a **String** in the format:

```
CPU: type, clock speed N, has N cores
```

Finally, add an appropriately implemented **equals** method.

The test cases for this question is called: **testCPU and testToString**.

(c) (2 marks)

Create a class **GPU** which inherits the functionality of **CPU** but also includes an **int** instance variable Memory. Update the **equals** method as needed. Its **toString** should output:

```
GPU: type, clock speed N, has N cores and NGB memory
```

The test cases for this question is called: **testGPU and testToString**.

(d) **(3 marks)**

Create a class called **Workstation** which **implements** the **Computer interface**. This is your base class, the constructor should take values for RAM, HDSpace, an **ArrayList** of **CPUs** and an **ArrayList** of devices (in that order). Add an **appropriate equals** method and a **toString** method that outputs the properties of the **Workstation** in the following format:

```
Workstation: NGB RAM, NTB HD, [list of CPUs], [list
of devices]
```

The test cases for this question is called: **testWorkstationOne and testWorkstationThree**.

(e) (2 marks)

Create a **class Laptop** which inherits functionality from **Workstation**. **Laptops** must have a device called "Screen", and cannot have more than one **GPU** or more than 2 **CPUs**. Its **toString** starts "Laptop" instead of "Workstation" otherwise the output is the same. **Laptop** represents the end of the class hierarchy.

The test cases for this question is called: **testLaptop and testLaptopErrors**.

(f) (4 marks)

Add a **class** method to **Workstation** called **produceWorkstations** which uses an instance of **Factory** to create an **ArrayList** of **Workstations**. This method should simulate a factory building n workstations with the same specification. You **MUST** use the **Factory** interface, if the tests pass, but you do not use it, you will receive 0 for this question.

The test case for this question is called: **testFactory.**

(g) (2 marks)

Add any additional appropriate input validation to all classes.

(h) (2 x 5 = 10 Marks)

Answer any **TWO** of the video recording questions (see separate PDF, which will be released at 11:30). This question is a MUST PASS question, i.e. a score of at least 4 out of 10 is needed to pass the question.

(i) (7 marks)

Ensure that your code is:

i. well formatted

(0.5 marks)

ii. applies any appropriate best practices and/or minimises repetition

(5.5 marks)

iii. appropriately commented

(1 mark)

--- end of Question 1 ---

Question 2

Answer parts (a) to (e) with solutions coded in Java, and answer part (g) via video recording and demonstration.

35 marks in total

In this question, we will simulate set of Football teams playing games in a league. In this question, you will develop the following classes:

- Team in (a)
- Score in (b)
- Game in (c)
- League in (d)
- Some extensions to Team are made in a more advanced question (e)

Add appropriate validation testing in appropriate places to all classes.

(a) (3 marks)

Create the **class Team** such that:

- Constructor takes as input the **name** of the **team** and their location (as **Strings**)
- Mutator methods to add points for a win (3) and for a draw (1), named **addWin** and **addDraw** respectively.
- Accessor methods for the three variables: **team name**, **location** and **current points**
- A **toString** method that returns a String representation of a team in the form: `<Team Name> from <Location> currently has <N> points.`
  (Note the full stop at the end, and pay attention to when a team has only one point)
- Hint: you may want an **equals** method or other useful methods for use in questions (d) and (e) – marks for these additions are awarded in (d) and (e).

The test case for this question is called: **testTeam**

(b) (2 marks)

Create the **class Score** such that:

- The constructor takes as input two **int** variables: the number of goals for the home **Team** and away **Team** respectively.
- Score is immutable, and cannot take negative input
- Has accessor methods the home and away **Team** score (number of goals scored): **getHomeScore** and **getAwayScore**.

The test case for this question is called: **testScore**

(c) (2 marks)

Create the **class Game** such that:

- The constructor takes as input two **Team** objects and a **Score** object
- A **toString** method that returns a **String** in the form for a win:
  `<Home Team> <N> - <Away Team> <N>; Result: <Team Name> Won;`
  or for a draw:
  `<Home Team> <N> - <Away Team> <N>; Result: Draw`
  Where `<N>` represents the number of goals scored by the corresponding **Team**

The test case for this question is called: **testGame**

(d) (7 marks)

Create the **class League** such that:

- It has a **default** constructor
- Uses an **ArrayList** of **Teams** (to represent the **League**)
- Has a method **addGame**, which based on the result calls the appropriate **addWin** or **addDraw** methods in the corresponding **Team** object (**Hint**: you will need to find it in the **ArrayList**)
- A method **addTeam**, which adds a **Team** to the **League** (and prevents duplicate **Teams**)
- A **toString** method that outputs the contents of the **ArrayList** (one **Team** per line displayed in the order they were added to the League; no ordering by points is necessary)

The test cases for this question is called: **testLeagueOne and testLeagueTwo**

(e) (4 marks)

Add a method **orderedToString** to **League** that reproduces the toString output but orders it by points, and tie breaks by goals scored. Hint: extensions will be needed to **Team** for this.

The test cases for this question is called: **testLeagueOrdered and testLeagueOrderedWithTie**

(f) (2 x 5 = 10 Marks)

Answer any **TWO** of the video recording questions (see separate PDF, which will be released at 11:30). This question is a MUST PASS question, i.e. a score of at least 4 out of 10 is needed to pass the question.

(g) (7 marks)

Ensure that your code is:

i. well formatted

(0.5 marks)

ii. applies any appropriate best practices and/or minimises repetition

(5.5 marks)

iii. appropriately commented

(1 mark)

--- end of Question 2 ---

--- end of Part A ---

Question 3

Answer parts (a) to (c) with solutions coded in Java, and answer part (d) via video recording and demonstration.

35 marks in total

In this question, you will create one class called **Draw.**

Hint: for this question focus on iteratively building up towards the desired pattern.

(a) (4 marks)

Define a class called **Draw**, with 2 class methods: one called **triangle**, and one called **crosses.** These will both output a **String** pattern.

Both methods take 2 **int** and 2 **char** parameters. The first **int** is the height of the pattern, the second **int** is the width.

The method **crosses** also takes a fifth parameter which is a **boolean** representing the direction of the crosses in the pattern: **true** means they should be of the form a grid (i.e. vertical and horizontal lines), **false** means they should be of the form an x (i.e. 4 diagonal lines).

Ensure that the following condition is maintained for both methods:

- The **char** parameters cannot be equal

Ensure that the following condition is maintained for the method **triangle:**

- The width and height must always be in the range 5-20 (inclusive)
- Width must always be larger than height, in fact it must always at least equal to (2 x h) – 1. Otherwise the triangle won't fit.
- Width must always be odd

Ensure that the following conditions are maintained for the method **crosses:**

- The width and height must always be within 5-25 (inclusive)
- Width and height must always be equal for diagonal crosses (i.e. when the **Boolean** is false)

The test case for this question is called: **testPartA**. It tests:

i. the signature of the method, and
ii. whether an error is thrown in response to invalid input

(b) (8 marks)
Implement the method **triangle** such that it returns a **String** that contains a pattern of a triangle. See examples overleaf.

Example 1:

```
String s = Draw.triangle(6,13,'x', '.');
System.out.println(s);

......x......
.....xxx.....
....xxxxx....
...xxxxxxx...
..xxxxxxxxx..
.xxxxxxxxxxx.
```

Example 2:

```
String s = Draw.triangle(7,19,'o', '-');
System.out.println(s);

---------o---------
--------ooo--------
-------ooooo-------
------ooooooo------
-----ooooooooo-----
----ooooooooooo----
---ooooooooooooo---
```

Essentially, the first **char** represents the background of the triangle, and the second **char** the foreground. The triangle is an equilateral triangle and is always centred in the output.

The test case for this question is called: **testPartB**, it tests the two examples above.

(c) (8 marks)
Implement the method **crosses** such that it returns a **String** that contains a pattern of a vertical/horizontal or diagonal crosses. See examples below. The first **char** is always the **cross**, and the second **char** is always background of the pattern.

For vertical/horizontal the "cross" part or first **char** is every three characters. For the diagonal the "cross" parts sit on the centre diagonals shifted by 2 to create four lines.

Example 1 (vertical/horizontal crosses):
```
String s = Draw.crosses(13, 13, 'x', '.', true);
System.out.println(s);
```

```
xxxxxxxxxxxxx
x..x..x..x..x
x..x..x..x..x
xxxxxxxxxxxxx
x..x..x..x..x
x..x..x..x..x
xxxxxxxxxxxxx
x..x..x..x..x
x..x..x..x..x
xxxxxxxxxxxxx
x..x..x..x..x
x..x..x..x..x
xxxxxxxxxxxxx
```

Example 2 (diagonal crosses):
```
String s = Draw.crosses(12, 12, 'x', '.', false);
System.out.println(s);
```

```
..x......x..
...x....x...
x...x..x...x
.x...xx...x.
..x..xx..x..
...xx..xx...
...xx..xx...
..x..xx..x..
.x...xx...x.
x...x..x...x
...x....x...
..x......x..
```

The test case for this question is **testPartC**, it tests the two above examples.

(d) (2 x 5 = 10 Marks)
Answer any **TWO** of the video recording questions (see separate PDF, which will be released at 11:30). This question is a MUST PASS question, i.e. a score of at least 4 out of 10 is needed to pass the question.

(e) (5 marks)
Ensure that your code is:

i. well formatted
(0.5 marks)

ii. applies any appropriate best practices and/or minimises repetition
(3.5 marks)

iii. appropriately commented
(1 mark)

--- end of Question 3 ---

Question 4

Answer parts (a) to (c) with solutions coded in Java, and answer part (d) via video recording and demonstration.

35 marks in total

In this question, you will complete the methods in the class called **Question4**

(a) (9 marks)
Complete the method **getMostCommon** which receives an **Object** array and returns the **String** value of the most common element in the array. If there is no "most common" element return the **String** "it's a tie"

Example 1:
```
Array contains (1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 1, 1)
Returned value: "1"
```

Example 2:
```
Array contains ("1", "1", 1, 1, 2, 2, 2, "a", "Exam")
Returned value: "1"
```

Example 3:
```
Array contains (1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 6)
Returned value: "it's a tie"
```

The test case for this question is called: **testPartA**, it tests the three examples above.

(b) (5 marks)
Complete the method **findPairs** which receives an **array** of **integers** and an **int** and returns an **ArrayList** of instances of **Pair** (this class is available in the Question.java file). You should search the array for any two **ints** whose product (i.e. when multiplied) is equal to the **int** parameter. When you find a pair, create an instance of **Pair** and add it to the **ArrayList** to be returned. The order of the variables in the constructor is irrelevant. Do not modify the **Pair** class.

Example 1:
```
Array contains (1, 1, 1, 2, 2, 1, 1, 1)
int parameter = 4
```
The **ArrayList** should contain **one instance** of **Pair** with the values (2, 2).

Example 2:
```
Array contains (1, 2, 3, 4, 5, 6, 7, 8)
int parameter = 12
```
The ArrayList should contain **two instances** of **Pair** with the values (2, 6) and (3, 4).

Example 3:
```
Array contains (1, 1, 1, 1, 1, 1)
int parameter = 3
```
The ArrayList should be empty as there are no ways to compute the product of 3 using any two elements in this array

The test case for this question is called: **testPartB**, it tests the three examples above.

(c) (6 marks)
Complete the method **reverseArray** which receives a two-dimensional array of **integers.** The method should return the array such that it is reversed in the first dimensions if the first **boolean** parameter is **true** and the second dimension if the second **boolean** parameter is **true**.

Example (both parameters are true):
```
Array is [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Returned is [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
```

The test case for this question is called: **testPartC**, it tests the example above and several others.

(d) (2 x 5 = 10 Marks)
Answer any **TWO** of the video recording questions (see separate PDF, which will be released at 11:30). This question is a MUST PASS question, i.e. a score of at least 4 out of 10 is needed to pass the question.

(e) (5 marks)
Ensure that your code is:

i. well formatted

(0.5 marks)

ii. applies any appropriate best practices and/or minimises repetition

(3.5 marks)

iii. appropriately commented

(1 mark)

--- end of Question 4 ---

Question 5

Answer parts (a) to (d) with solutions coded in Java, and answer part (e) via video recording and demonstration.

35 marks in total

In this question, you will create one class called **Adult**

NOTE you need to modify lines 11 and 12 in **Q5Tests.java** to reflect where you have saved the test files (**adult.txt** and **adult-errors.txt**) for this question. Remember on Windows, you need to escape a backslash in a file path, i.e. "\" should be "\\".

(a) (8 marks)
Create a class called **Adult** it should have the following instance variables:

- **Education**: which can have the values: Bachelors, Masters, or Doctorate
- **Status**: which can have the values: Married, Single, or Divorced
- **Gender**: which can have the values: Male or Female
- **Degree**: which can have the values: CS, Econ or Arts
- **Salary**: which indicates whether the **Adult** earns more (true) or less (false) than 50k indicated in the file as >50K and <=50K respectively.

Decide which types make the most sense for these variables but provide accessor methods that return a **String** value (as indicated above) for each of them except Salary which should remain a **boolean**.

The constructor for the Adult class should receive a single String, which will be a comma separated representation of all variables in the order presented above.

E.g.:
```
Bachelors,Married,CS,Male,<=50K
```

Should be constructed as an Adult, that has a Bachelors, is Married, studied CS, is Male and earns less than 50k.

The test case for this question is called: **testPartA**, it tests the above example using the accessor methods for each of the parameters.

(b) (2 mark)
Create a **toString** method that returns a String in the following form:
```
Adult (Male, Single) studied Economics (Masters) earns less
than 50k
Adult (Female, Married) studied CS (Masters) earns more than
50k
```

The test case for this question is called: **testPartB**, it tests that the example used in part (a) produces the correct String output.

(c) (5 marks)
Add a **class method** called **read**, that receives a File object, and returns an ArrayList of Adults.

Each line in the file corresponds to one **String** that can be used to construct an **Adult** instance.

The test case for this question is called: **testPartC**, it tests that the file **adult.txt** produces an **ArrayList** of **Adults**, and checks the first two elements of this **ArrayList** and that it has the correct size.

(d) (5 marks)
Make your **read** method from question 8(c) and/or your **constructor** from question 8(a) appropriately handle the following error conditions:

- Errors in the file: values for parameters that are not mentioned in part (a)
- Receiving a file that doesn't exist
- Incomplete lines, i.e. one or more parameters are missing.

In the event of any of these errors occurring, you should print an appropriate warning message to standard out, but still return an **ArrayList** of all valid **Adults**.

The test case for this question is called: **testPartD**, it tests that the file **adult-error.txt** produces an **ArrayList** of **Adults**, and checks the first two elements of this **ArrayList** and that it has the correct size.

(e) (2 x 5 = 10 Marks)
Answer any **TWO** of the video recording questions (see separate PDF, which will be released at 11:30). This question is a MUST PASS question, i.e. a score of at least 4 out of 10 is needed to pass the question.

(f) (5 marks)
Ensure that your code is:

iv. well formatted
(0.5 marks)

v. applies any appropriate best practices and/or minimises repetition
(3.5 marks)

vi. appropriately commented
(1 mark)

--- end of Question 5 ---

--- end of Part B ---