UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08014 INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Tuesday 8th May 2018

09:30 to 11:30

INSTRUCTIONS TO CANDIDATES

1. **Note that all questions are compulsory.**
2. **Remember that a file that does not compile, does not pass the simple JUnit tests provided, or uses Java packages will get no marks.**
3. **This is an Open Book exam. You may bring in your own material on paper. No electronic devices are permitted.**
4. **CALCULATORS MAY NOT BE USED.**

Convener: I. Simpson
External Examiner: I. Gent

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. In this task you will implement a buffer and functionality to add and remove elements or check the contents of the buffer. In computer science a buffer is used to temporarily store data while it is being moved from one place to the other. The buffer you will implement is able to store data of the type `int`. All its data is kept in an internal integer array which has a maximum capacity specified in the constructor.

   Your task is to implement the subclass `RingBuffer` of `Buffer`, representing a special kind of `Buffer` which is filled with data by adding to the back and emptied by removing from the front. The special feature of a RingBuffer is its ability to wrap saved data entries around the end of the underlying array. It does that by keeping track of the location of the initial data entry in the buffer and the location of the last data entry.

   To illustrate its functionality, consider the following examples:

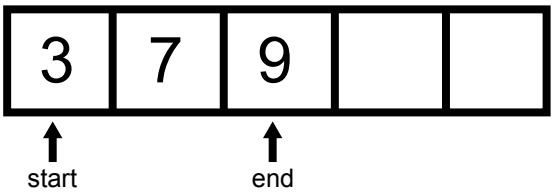

Figure 1: A ring buffer of size five filled with three integer entries. *Start* and *End* markers keep track of the first and the last element in the buffer.

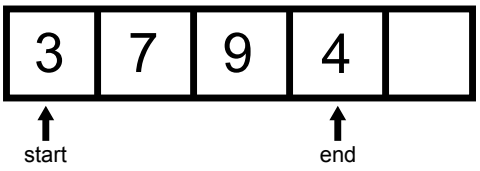

Figure 2: Adding an element to the end moves the *End* marker forward.

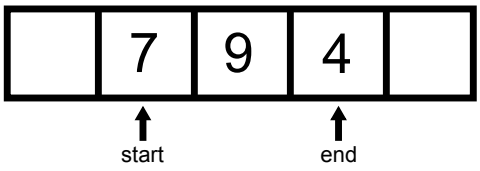

Figure 3: Removing an element from the start moves the *Start* marker forward.

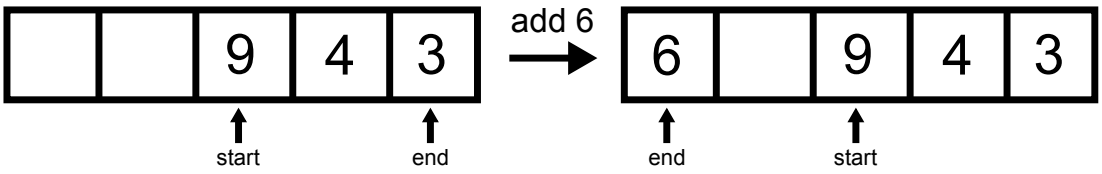

Figure 4: If the *End* marker reaches the end of the underlying array and there is space in the front of the array, the *End* marker will wrap around.

QUESTION CONTINUES ON NEXT PAGE

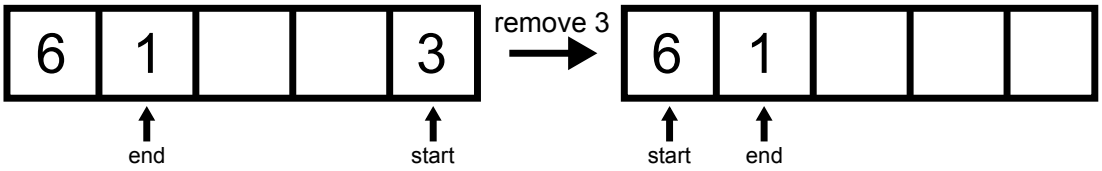

Figure 5: The same behaviour is true for the *Start* marker.

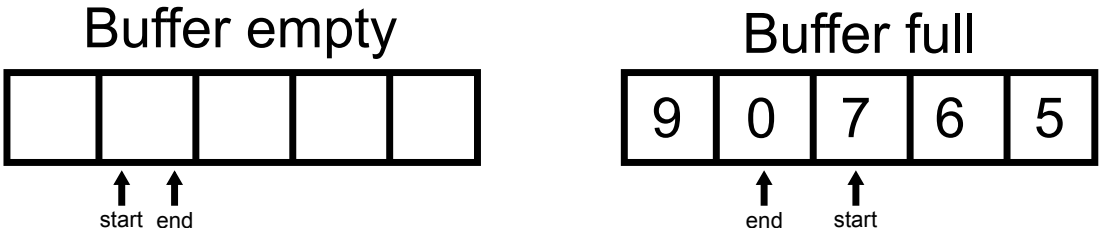

Figure 6: When the buffer is empty, both *Start* and *End* markers point to the same array index. **The same is true if the buffer contains a single element.**

From the exam template directory, please use all of the following files to answer this question (if you are using Eclipse, make sure you import ALL of them):

- Buffer.java
  Work with the `Buffer` class provided in the exam's template folder, NOT the `Buffer` class from the Java API!
- RingBufferBasicTest.java

Please execute the following steps for your implementation of `RingBuffer`:

(a) Define the class `RingBuffer`, extending `Buffer`. It should have three instance variables:

- a private variable `start` of type `int` representing the start marker
- a private variable `end` of type `int` representing the end marker
- a private variable `elementCount` of type `int` representing how many elements are currently in the array

[5 marks]

(b) Write two public constructors for `RingBuffer`. Your first constructor must take an `int` which is the buffer's total capacity. Invoke the `Buffer` constructor as appropriate which will will take care of the array initialisation and checks if the given capacity is in a valid range. Initialise `RingBuffer`'s marker instance members and element count to zero.

Your second constructor is a zero argument constructor that invokes your first constructor with a default capacity of ten.

QUESTION CONTINUES ON NEXT PAGE

NOTE: The buffer's capacity refers to the length of the underlying array and is constant throughout the execution of the program. The element count refers to the amount of valid data currently stored in the buffer and can vary over the course of the program's execution. [*10 marks*]

(c) Write a public instance method `isFull`, which takes no arguments and returns a `boolean` value indicating whether the buffer is filled up to its capacity. **HINT consider using elementCount here.** [*5 marks*]

(d) Write a public instance method `isEmpty`, which takes no arguments and returns a `boolean` value indicating whether the buffer is completely empty. **HINT consider using elementCount here.** [*5 marks*]

(e) Override `Buffer`'s instance method `clear`, which empties the buffer. Use the same access modifier for your overridden method as is used in `Buffer`. In your implementation, use `Buffer`'s `clear` method to clear out all entries from the array. Then, make sure all markers and the element counter are updated accordingly. [*5 marks*]

(f) Write a public instance method `addToBuffer`, taking an `int` argument and returning nothing. This method will add the given integer to the end of the buffer. **All input elements can expected to be positive integers or zero.**

If the buffer is full, the argument should be ignored and the following **exact** message should be printed (including dot and newline): `Buffer is full.`

Make sure you move the `start` and `end` markers as described above. Also, keep track of the `elementCount`. [*10 marks*]

(g) Write a public instance method `getFromBuffer`, taking no arguments and returning an `int`. This method will remove the first integer entry from the buffer and return it. **An empty space in the underlying buffer array must be set to -1.**

If the buffer is empty, -1 should be returned and the following **exact** message should be printed (including dot and newline): `Buffer is empty.`

Make sure you move the `start` and `end` markers as described above. Also, keep track of the `elementCount`. [*10 marks*]

The file you must submit for this question is `RingBuffer.java`. Before you submit, check that it compiles, passes the basic JUnit tests provided and uses no packages, otherwise it will get 0 marks.

2. In this question you will implement parts of a Battleships game.

Game In Battleships, ships of varying length are placed on a square game grid. Ship pieces can be horizontally or vertically aligned. A player can fire shots at different positions on the game grid trying to hit the ships. If all parts of a ship were hit, the ship is destroyed.

Game Grid In your program the game grid is a square two-dimensional array of characters called `grid` where the first index is for y coordinates and the second index for x coordinates. Each character in a game grid is a field in the grid and can either be a '#' representing an empty field or a capital letter, e.g. 'A', representing a ship piece. Pieces of the same ship always have the same capital letter. You can assume that ships have a length between 1 and grid side length.

You must never change the contents of a game grid array.

	0	1	2	3	4
0	#	#	#	#	#
1	#	#	B	B	B
2	A	#	#	#	#
3	A	#	#	C	#
4	#	#	#	#	#

Figure 7: This example grid has a side length of five and three ships A, B and C.

Utils Class A utils class `BattleShipsUtils` is provided with a method `parseGrid` for parsing a game grid from a given file name as well as a method `printGrid` for printing a game grid. Also a method `generateShots` to help you generate random shots at the game board. You can use those methods to retrieve and output data and test your implementation.

Game Data Files Two game data files called `game01.txt` and `game02.txt` with example game grids are provided for you. You can alter them or create your own with a simple text editor to test your program.

BattleShips Skeleton For this question you are provided with a `BattleShips` skeleton implementation in file `BattleShips.java`. The skeleton has method stumps for your implementation where you can fill in your solutions.

BattleShips main method The skeleton also comes with a main function which is already filled with an example execution of the game. You are not marked on the contents of the main function, so feel free to alter it for testing your code. However, you must make sure that your final solution has *no compiler errors!*

QUESTION CONTINUES ON NEXT PAGE

You can execute the main method the same way as you are used to from the labs. The required command line argument is the name of a game txt file, e.g. `game01.txt`.

Position Class The provided position class is used to encapsulate x and y coordinates into a single object. You will have to use it for your solutions, however, you must not change it.

JDK Library Classes In this question you will use the collection classes `ArrayList` and `Hashtable` which are familiar from lectures and labs. You may need to look at the JDK documentation for it. Note that the types of your methods will involve `Map` and `List`, thus hiding, from clients, which implementation of the `Map` or `List` interface is used; your code is expected to create, concretely, `Hashtable`s and `ArrayList`s.

Argument Assumptions You may assume that none of the object arguments to your methods are `null`.

From the exam template directory, please use all of the following files to answer this question (if you are using Eclipse, make sure you import ALL of them):

- BattleShips.java
- BattleShipsUtils.java
- Position.java
- BattleShipsBasicTest.java
- game01.txt
- game02.txt

For your solution, implement the following three game aspects:

(a) Implement the method `fireShot` as given in the skeleton file. This method fires a shot at a field in the game grid. It takes a game grid, `grid`, and a `Position`, `shot`, and returns the character at the coordinates in the grid specified by `shot`. If the shot is out of bounds, you should return a '#' character. For each shot print a message **exactly** as specified below (only single spaces between words):

- In the case of the sample grid specified in Figure 7 and a shot at position (1,1), print: `(1,1): Miss`
- In the case of the sample grid specified in Figure 7 and a shot at position (4,1), print: `(4,1): Hit B`
- In the case of the sample grid specified in Figure 7 and a shot at position (-2,10), print: `(-2,10): Out of Bounds`

Hint: The `Position` class conveniently implements the `toString` method. [*15 marks*]

QUESTION CONTINUES ON NEXT PAGE

(b) Implement the method `findShips` as given in the skeleton file. This method finds each ship on the game grid and their corresponding sizes, i.e. how many pieces they have. It takes a game grid, `grid`, and returns a `Map` mapping `Character`s to `Integer`s. The `Character`s in the `Map` are the names of the ships on the game grid while the `Integer`s indicate their length. For the sample grid specified in Figure 7, `findShips` would return the following mapping (not necessarily in that order):

```
A -> 2
B -> 3
C -> 1
```

If the given grid has no ships, you should return an empty `Map`.

HINT: Remember to use the concrete type Hashtable as return value here. [*15 marks*]

(c) Implement the method `fireShots` as given in the skeleton file. This method fires a series of shots at the game board an checks how many ships were completely destroyed. It takes a game grid, `grid`, and a `List` of `Position`s, `shots`, and returns a `List` of `Characters`. All entries in `shots` are coordinates of shots and can potentially hit a ship. The `List` can have a shot at the same coordinates multiple times.

Process the given `List` of shots and return a new `List` filled with the `Character`s of each ship that has been completely destroyed by the fired shots. If no ship was destroyed, you should return an empty `List`. For each shot that was fired, print whether it hit, missed or was out of bounds as specified in the first part of this question.

In the case of the sample grid specified in Figure 7 and a series of shots at the positions (0,2),(1,4),(0,3),(3,1),(3,3), the method should return (not necessarily in that order): `[A,C]`

And print the following output:

```
(0,2): Hit A
(1,4): Miss
(0,3): Hit A
(3,1): Hit B
(3,3): Hit C
```

HINT: Consider using the methods you have implemented in previous parts of this question. Remember to use the concrete type ArrayList as return value here. [*20 marks*]

The file you must submit for this question is `BattleShips.java`. Before you submit, check that it compiles, passes the basic JUnit tests provided and uses no packages, otherwise it will get 0.