256

With Arraylists

Introduction

This document specifies the programming component of Assignment 2. This component is due by 11pm Saturday of Week 12 (26th October, 2019). **Heavy penalties will apply for late submission.** This is an individual assessment task and must be your own work. You must attribute the source of any part of your code which you have not written yourself. Please note the section on plagiarism in this document.

The assignment must be done using the BlueJ environment.

The Java source code for this assignment must be implemented according to the **Java Coding Standards for this unit.**

Any points needing clarification may be discussed with your tutor in the lab classes.

Completion of this assignment contributes towards the following FIT9131 learning outcomes:

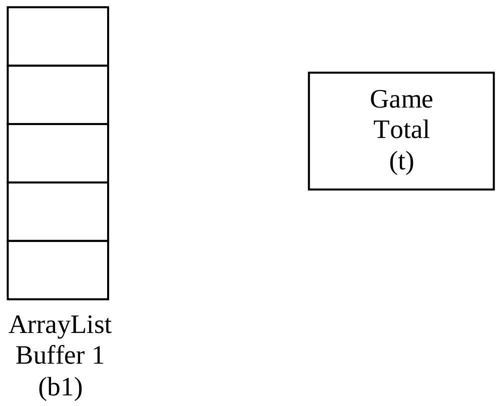
1. *design, construct, test and document small computer programs using Java;*
2. *interpret and demonstrate software engineering principles of maintainability, readability, and modularisation;*
3. *explain and apply the concepts of the "object-oriented" style of programming.*

Specification

In this assignment you will write a program to play the game of 256 With Arraylists. The objective of the game is for the player to keep adding three randomly provided multiples of a number, to attempt to get the total score of 256 or higher. The game is a modified version which allows the player to use up to two arraylists as buffers which can be used to store the numbers and retrieve the numbers. This section specifies the required functionality of the program. **Only a text interface is required for this program**; however, more marks will be gained for a game that is easy to follow with clear information and error messages to the player.

256 With Arraylists (D Level Assignment – Max Mark 79)

The game begins with the program reading the allowed multiples from a file called "**multiples.txt**". (This file is available on moodle). The file will only contain ONE set of mutliples. These are the three numbers which will be randomly provided to the user during the game. For e.g. 2, 4, and 8. The game will operate using one arraylist as a buffer instead of a grid. The maximum number of elements which can be stored in the arraylist is 5. Below is a depiction on the game layout (for illustration only):



The game sequence is as follows:

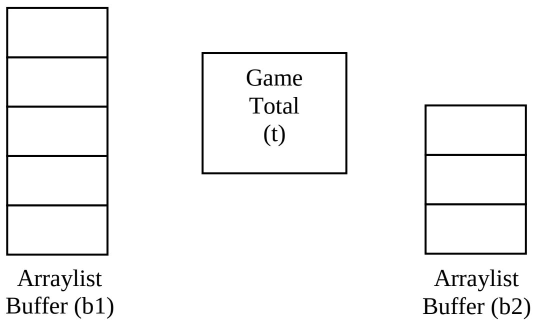
1. The game starts by registering a player to play the game.
2. The Arraylist buffer (b1) is empty.
3. The game then provides a random multiple to the player which is stored in the game total from the available multiples..
4. The player can then perform the following actions:
   - Split:
     - This allows the player to move the number from the game total (t) and add it to the arraylist buffer (b1).
     - A new random multiple is then generated and put in the game total (t)
     - The game then continues from **point 3 above**.
   - Merge:
     - This allows the player to merge the number in the game total box with a **matching number** in the arraylist at any position. The total is then put in the game total box and the number which is added is then removed from the arraylist.
     - After the total is added, the game total (t) now shows the new number. No new random number is generated and the game goes back to **point 4 above** to allow the player to continue.
5. The game ends when either of the following occur:
   - The game total (t) reaches the total of 256 or higher.
   - The arraylist buffer (b1) has reached the maximum limit of 5 numbers and no more numbers can be stored and none of the existing numbers can be merged to the game total (t).

At the end of the game, the program will write the final outcome to the file "**outcome.txt**" which will show the player name and the highest score achieved.

256 With Arraylists (HD Level Assignment – Max Mark 100)

Note: You may complete parts of the HD level to score more marks on the D level assignment.

The game begins with the program reading the allowed multiples from a file called "**multiples.txt**". (This file is available on moodle). The file will only contain MANY sets of mutliples. The user can **choose** which multiple they would like to use when playing the game. These are the three numbers which will be randomly provided to the user during the game. The game will operate using **TWO** arraylist as buffers instead of a grid. The maximum number of elements which can be stored in the first arraylist is 5. The maximum number of elements which can be stored in the second arraylist is 3. The game also allows the user to set the game total up to which the game will be played. Below is a depiction on the game layout (for illustration only):



The game sequence is as follows:

1. The game starts by registering a player to play the game.
2. The player is prompted to enter the game total upto which the game will be played. The game total must be greater than 32 and a multiple of 8.
3. The player is then prompted to select from the possible list of multiple with which to play the game. (These have been read from the file).
4. Both the Arraylist buffers (b1 and b2) are empty.
5. The game then provides a random multiple to the player which is stored in the game total.
6. The player can then perform the following actions:
    - Split Right (→)
        - This allows the player to add the game total (t) number to the right arraylist buffer (b2)
        - A new random multiple is then generated and put in the game total (t)
        - The game then continues from **point 5 above**.
    - Merge Right (←←)
        - This allows the player to merge the number in the game total box with a **matching number** in any position in the right arraylist buffer. The total is then put in the game total box and the number is removed from the arraylist.
        - After the total is added, the game total (t) now shows the new number. No new random number is generated and the game goes back to **point 6 above** to allow the player to continue.
    - Split Left (←)
        - This allows the player to add the game total (t) number to the left arraylist buffer (b1)

        - A new random multiple is then generated and put in the game total (t)
        - The game then continues from **point 5 above**.
    - Merge Left (→→)
        - This allows the player to merge the number in the game total box with a **matching number** in any position in the left arraylist buffer. The total is then put in the game total box and the number is removed from the arraylist.
        - After the total is added, the game total (t) now shows the new number. No new random number is generated and the game goes back to **point 6 above** to allow the player to continue.

7. The game ends when either of the following occur:
    - The game total (t) reaches the total of whatever the user entered in point 2 above or higher.
    - The arraylist buffers (b1 and b2) have reached the maximum size limit allowed (depending on the difficulty level) and no more numbers can be stored and none of the existing numbers can be merged to the game total (t).

At the end of the game, the program will write the final outcome to the file "**outcome.txt**" which will show the player name and the highest score achieved.

The following are some of the requirements for the game:

- Player name must be between 3 and 10 characters, cannot be blank and cannot contain only spaces.
- The arraylist buffers will not allow more elements to be stored beyond the assigned size limit (defined by the difficulty level).

Hints and Suggestions

Your assignment may want to use pass by reference to achieve a well-designed program. Additionally, should your program wish to accept only String inputs from the user, you may want to look up the **Integer.parseInt()** method which belongs to the Integer class. This can be used along with exception handling to ensure your program doesn't crash.

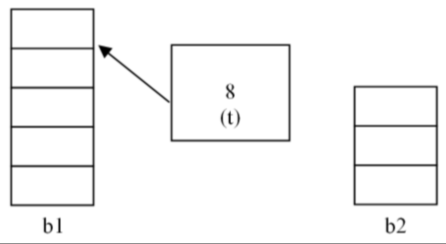
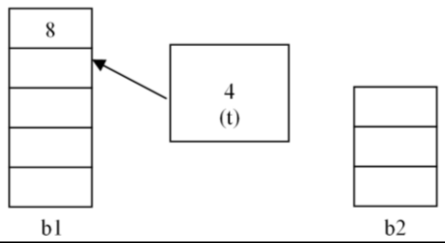
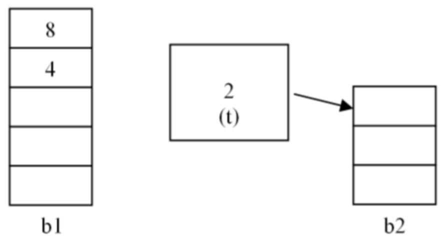
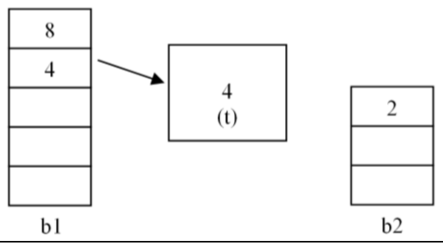
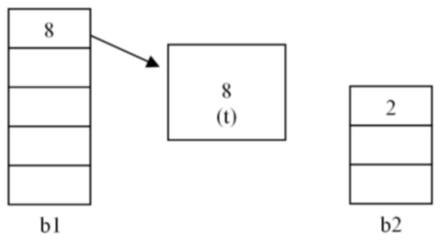
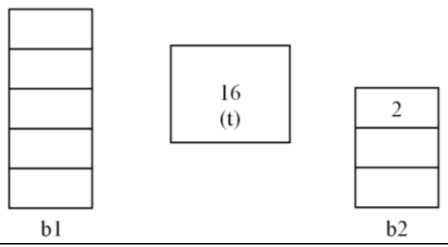
The new random number is only generated, IF the value for user choose to Split the value (i.e. move it to the array list).

Requirements

You **MUST** show your tutor the D level version of the assignment prior to submitting the HD Level. To do so, you may make a copy of your D level assignment which can be shown to your tutor during your scheduled tutorial.

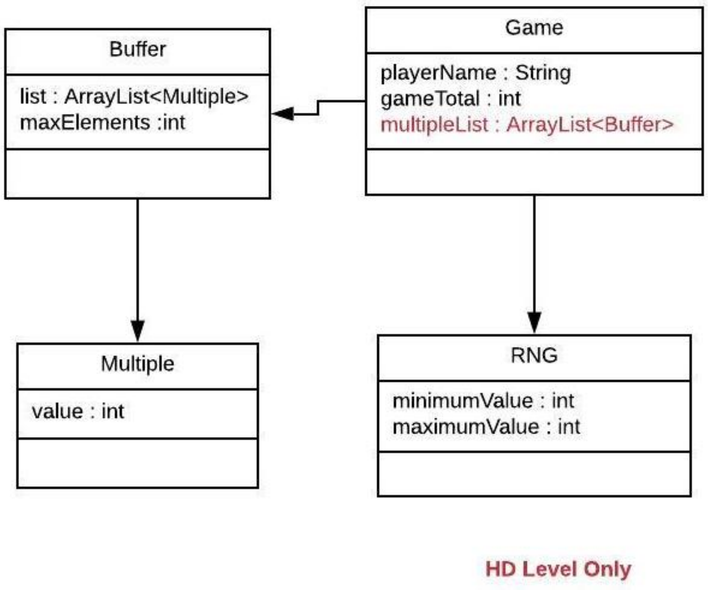
Gameplay Example (HD Level Only)

The following shows a single game play using the multiples 2, 4, and 8. And the winning game total is going to be 16.

	
Step 1 – Game begins and generates a random number in the game total (t). The user selects Split Left. The game will then moves the current game total (t) and adds it to the arraylist buffer (b1). A new random number is then generated and stored in the game total (t)	Step 2 – The user selects Split Left again. The game then moves the current game total (t) and adds it to the arraylist buffer (b1). A new random number is then generated and stored in the game total (t)
	
Step 3 – User selects Split Right. The game then moves the current game total (t) and adds it to the arraylist buffer (b1). b1 now has 2 elements stored in there. A new random number is then generated and stored in the game total (t)	Step 4 – User selects Merge Left. The game then searches the arraylist buffer (b1) for a value which matches that in the game total (t). If found, both numbers are added, and the matching element in the arraylist is deleted.
	
Step 5 - User selects Merge Left again. The game then searches the arraylist buffer (b1) for a value which matches that in the game total (t). If found, both numbers are added, and the matching element in the arraylist is deleted.	Step 6 – Player has reached the desired total of 16 (for this illustration example only). So the game ends. The system displays that the player wins.

Program design

Your program must consist of at least the following classes: *Game, Buffer, Multiple, and RNG.* A more scalable design with additional support classes which can be reused will get more marks. The following is the proposed class diagram for the program.The remainder of this section gives details of these classes. There are suggested fields for each class. If you include any extra fields then you must be able to justify these. You may want to include comments in your program to state any assumptions made.



Game:

The main class of the program. It will allow the user to start the game. It generally handles all the inputs and outputs to the user unless another utility class has been included to do this. It has the following attributes – PlayerName which store the players name which must be between 3 and 10 characters long and cannot be blank or all spaces; and gameTotal which indicates the game total upto which the game will be played. The HD level of the assignment has one additional attributes, multipleList - which stores the possible multiples with which the game can be played read from the file. (<u>If you have difficulty applying this concept, discuss alternative approaches with your tutor</u>).

Buffer:

This class describes a single arraylist buffer for the game. The attribute includes an arraylist which stores objects of the multiple class, and an integer which stores the maximum number of elements which can be stored in the arraylist buffer.

Multiples:

This class stores an integer value for a given multiple which is used in the game.

RNG:

An object of the RNG class will generate a random number from a minimum value to a maximum value. It will have two fields, maximumValue – maximum value of the random number, and minimumValue – minimum value of the random number

Test Strategy

Using the template discussed in Week 7's lecture, your assignment must also provide a detailed test strategy for the Buffer class only.

Assessment

Programming Task Assessment (20%)

Assessment for this component will be done via an interview with your tutor.

The marks for your program code will be allocated as follows:

- 10% - Test Strategy for the Buffer class only.
- 35% - Object-oriented design quality. This will be assessed on appropriate implementation of classes, fields, constructors, methods and validation of the object's state and adherence to Java coding standards
- 55% - Program functionality in accordance to the requirements.

You must submit your work by the submission deadline on the due date (a late penalty of 20% per day, inclusive of weekends, of the possible marks will apply - up to a maximum of 100%). There will be no extensions - so start working on it early.

Marks will be deducted for untidy/incomplete submissions, and non-conformances to the FIT9131 Java Coding Standards.

Please note that your program code will be submitted to a code similarity checker.

Interview

You will be asked to demonstrate your program at an "interview" following the submission date. At the interview, you will be asked to explain your code/design, modify your code, and discuss your design decisions and alternatives. Marks will not be awarded for any section of code/design/functionality that you cannot explain satisfactorily (the marker may also delete excessive in-code comments before you are asked to explain that code). In other words, you will be assessed on your understanding of the code, and not on the actual code itself.

Interview times will be arranged in the tutorial labs in Week 12. The interviews will take place in week 14. It is your responsibility to attend the lab and arrange an interview time with your tutor. **Any student who does not attend an interview will receive a mark of 0 for the assignment.**

Submission Requirements

The assignment must be uploaded to Moodle by 11pm Saturday of Week 12 (26th October, 2019).

The submission requirements for Assignment 2 are as follows:

A .zip file uploaded to Moodle containing the following components:

- the BlueJ project you created to implement your assignment.
- a completed **Assignment Cover Sheet**. This will be available for download from the unit's Moodle site before the submission deadline. You simply complete the editable sections of the document, save it, and include it in your .zip file for submission.

The .zip file should be named with your Student ID Number. For example, if your id is 12345678, then the file should be named 12345678_A2.zip. Do not name your zip file with any other name.

It is your responsibility to check that your ZIP file contains all the correct files, and is not corrupted, before you submit it. If you tutor cannot open your zip file, or if it does not contain the correct files, you will not be assessed.

Marks will be deducted for any of these requirements that are not complied with.

Warning: there will be no extensions to the due date. **Any late submission will incur the 20% per day penalty.** It is strongly suggested that you submit the assignment well before the deadline, in case there are some unexpected complications on the day (e.g. interruptions to your home internet connection).

Extensions

All requests for extensions must be sent to Mark Creado (mark.creado@monash.edu). Requests must follow the faculty guidelines, include the required forms and be submitted as soon as possible. In addition, when emailing for an extension, please remember to include all code completed until that time. This assignment cannot be completed in a few days and requires students to apply what we learn each week as we move closer to the submission date.

Plagiarism

Cheating and plagiarism are viewed as serious offences. In cases where cheating has been confirmed, students have been severely penalised, with penalties ranging from losing all marks for an assignment, to facing disciplinary action at the Faculty level. Monash has several policies in relation to these offences and it is your responsibility to acquaint yourself with these.

Plagiarism (http://www.policy.monash.edu/policy-bank/academic/education/conduct/plagiarism-policy.html)