# Assessment Brief: Mini Apps (Reassessment)

|  |  |  |
| --- | --- | --- |
| **Module Leader:** Tonderai Maswera | | **Level:** 4 |
| **Module Name:** Programming for Computer Science | | **Module Code:** 55-407816 |
| **Assignment Title:** Mini Java Apps | | |
| **Individual Task** | **Weighting:** 50% | **Magnitude:** 20 hours (notionally) |
| **Submission date/time:**  30th June 2022 at 3pm | **Blackboard submission:** Y **Turnitin submission:** N | **Format:** Single Java file |
| **Planned feedback date:**  21st July 2022 | **Mode of feedback:** Blackboard | **In-module retrieval available:** No |
|  | | |
| **Module Learning Outcomes**   * Describe, recognise, and deploy key concepts that relate to designing small imperative and basic object-oriented applications and algorithms. * Describe, recognise and deploy essential features of a mainstream programming language and use them to implement solutions to a variety of programming problems, selecting appropriate control constructs, data structures and objects. * Select and apply appropriate software tools and program testing and validation techniques on small programs. | | |

## Assessment Brief

In this task you will develop a series of mini applications that are designed to get you thinking logically about constructing code in Java. The concepts required to implement these applications form only a small and introductory selection of features found in the language, although you will need to carefully consider how to put those building blocks together. Alongside the code, you will need to provide a testing plan for one of the mini-apps.

This task can be tackled using the techniques covered in class, however aspects can be better coded with more advanced features. If you choose to research and deploy those advanced features of the language, then you will receive higher marks (please refer to the assessment criteria).

There are four mini applications to create, and all accessed via a menu system. The following sections outline the core details and requirements for each of the applications.

**Menu System**

* Create a top-level menu system that allows the user to select which of the mini applications to run when the application is launched.

P4CS Mini Applications

----------------------

Please select an option:

1) Higher-Lower Game

2) Change Calculator

3) Encrypt Text (Vigenere Cipher)

4) Decrypt Text (Vigenere Cipher)

9) Quit

Please enter option:

* If the user presses a number not in the list of options, display a message and show the menu again
* Once the selected application has finished, the menu should be displayed, and the user asked which app to run (or exit)
* When the user selects the quit option, your code should naturally terminate. **Do not use** System.exit(0) to end your application but do allow control paths to naturally converge at the end of the main method
* **Do not use** any form of recursion to implement the menu system. Use appropriate loops and methods to help split up your code and create a sensible iterative control flow

**1) Higher-Lower Game**

* The computer generates two random numbers between 10 and 30 inclusive
* The first random number is shown to the user as a baseline number and they are asked to guess whether the second number (being kept secret from the user) is higher or lower than the baseline number
* The user enters “higher” or “lower” (case insensitive), which are the only valid inputs. If the user enters anything else, an error message is displayed, and they are asked again for a valid input
* If the user guesses correctly, the secret number becomes the next baseline number to guess against and showed to the user. A new secret random number is generated in the same range (10 to 30) and sets up the next Higher-Lower question; the baseline number shown to the user is the secret number from the previous question after the first question is completed successfully
* The user needs to correctly guess 5 times in a row to win the game
* If the user guesses incorrectly, they lose the game immediately
* When the user wins the game, they are shown how long it took to complete, in seconds
* When the user losses the game, the hidden number is revealed but no time is given
* For any given question, the baseline and secret random numbers cannot be the same – they are always different, but randomly generated from the range 10 to 30 inclusively.

A transcript of this game running is given below, demonstrating the expected behaviour with the user input shown in blue text:

Example 1:

Higher-Lower Game

-----------------

Random numbers in the game of 10 and 30 inclusive

Question 1 out of 5: Number is 26 higher or lower?

low

Unknown Input - Please enter "Higher" or "Lower"

lower

Question 2 out of 5: Number is 18 higher or lower?

LOWER

Close, but not quite. The next number was 20

Example 2:

Higher-Lower Game

-----------------

Random numbers in the game of 10 and 30 inclusive

Question 1 out of 5: Number is 23 higher or lower?

lower

Question 2 out of 5: Number is 15 higher or lower?

lower

Question 3 out of 5: Number is 10 higher or lower?

HIGHER

Question 4 out of 5: Number is 13 higher or lower?

higher

Question 5 out of 5: Number is 19 higher or lower?

higher

Nice job... you got all the way to the end in 35 seconds

**2) Change Calculator**

* The user enters a whole, positive number (**as an integer**) and your app will calculate the minimum number of coins and notes needed to make up that number (that represent an amount in GBP pence)
* Valid currency denominations are:
  + £50 note
  + £20 note
  + £10 note
  + £5 note
  + £2 coin
  + £1 coin
  + 50p coin
  + 20p coin
  + 10p coin
  + 5p coin
  + 2p coin
  + 1p coin
* Your algorithm must use an array to store the valid currency denominations and a loop to calculator what combination of denominations results in the minimum number of those items.
* The output must only display the denominations needed and how many of them along with a total item count.

A transcript of this app is given below:

Example 1:

Change Calculator

-----------------

Please enter amount in pence:

123

Optimal change for 123p is:

1 x £1 coin

1 x 20p coin

1 x 2p coin

1 x 1p coin

Total items are 4

Example 2:

Change Calculator

-----------------

Please enter amount in pence:

1059

Optimal change for 1059p is:

1 x £10 note

1 x 50p coin

1 x 5p coin

2 x 2p coin

Total items are 5

**3) Encrypt Text (using Vigenère Cipher)**

* Input a plain-text message and key and encrypt it using the [Vigenère cipher - Wikipedia](https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher) – See for a full description about how this cipher works
* The valid alphabet symbols for both message and key that the user can input are:
  + A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, [space]
  + [space] is the usual space character between words
  + This alphabet is fixed – do not add or remove characters from those given above
* Lower-case characters are accepted as their uppercase versions and share the same character in the alphabet – i.e. if the user enters ‘a’ or ‘A’, they are both taken as an uppercase ‘A’
* If a user provides an input that includes any invalid characters, they are given a message and asked to enter their message again
* After the user has entered a valid input (that has been checked), they are asked for a valid key
* If the key uses invalid alphabet symbols, the user is asked to enter another key until a valid one is provided
* The plain-text message is encrypted using the Vigenère Cipher and the user-entered key. The encrypted messaged is displayed to the user

A transcript of the encryption algorithm running is given below, demonstrating the expected behaviour with the user input shown in blue text:

Encrypt Text

------------

Please enter text to encrypt:

HELLO-INVALID\_CHARACTERS?'

Invalid input text

Please enter text to encrypt:

This is my sample message

Please enter key:

INVALID\_CHARACTERS?

Invalid key

Please enter key:

Secret Key

Encoded string is: ' ALK9D1RJQLRWC3T4DJQ2 WCXI'

**4) Decrypt Text (using Vigenère Cipher)**

* Provide a decryption application that takes a Vigenère Cipher-encoded message and decrypts it
* The user inputs and validations must follow the same structure as those outlined in the Encrypt Text mini-app
* Refactor and reuse code between this mini-app and the encryption algorithm that is common between the two applications.

## Testing

Provide a test plan for the Change Calculator app. Put this above the main function for that app as a comment block. The test plan should include inputs, expected outputs, actual outputs, and should be sufficient to ensure your code is tested to a suitable level.

## General Principles, Code Development and Hints

* While you can assume the user enters input valid for the data type expected, you will obtain marks for researching and implementing techniques that guards against invalid input types. Minimally, you should check for range correctness (e.g. the menu system should give an error when “123” is entered)
* **Use sensible methods and functions** to divide the complete application up into logical sections. There is a minimum expectation that each of the mini-apps will be in their own method/function, which itself may use more functions.
* Avoid repeating sections of code by **reusing methods and functions** where possible (for example, to get valid input from the user in a specified range)
* **Create a single keyboard input Scanner in the main method and pass it to other functions that need it.**
* **Avoid all global variables** – all variables should be declared within the scope of a method or function or further nested construct
* You will get credit for what you have achieved so do not worry if you don’t get it all working
* **Comment your code** to an appropriate level, use sensible naming and make the code readable using a consistent and sensible layout.
* **It is expected that all source code will go into a single .java file**
* There is no single right solution, but some code is better than others… think about what you are doing, what you want to achieve and implement it in a sensible, robust and efficient way
* **Evolve your code and regularly test what you have written**; when we put together applications, we do not sit down and write it in one go and expect it to run… we build the application in small stages, progressively adding in functionality and testing and debugging as we go (yes, everyone needs to test and debug code they write no matter how long they have been coding for so get practiced at doing so)

## Grading Guidelines

The marking of this task is done based on your ability to successfully choose and apply appropriate programming structures to the coding tasks. You will need to fully implement the require functionality to obtain the highest marks, although marks will still be awarded for partial solutions and subject to the level of completion.

Marks will be awarded against the following areas of competence

| **Area of Competence** | **Description** | **Percentage Available** |
| --- | --- | --- |
| Data types | Appropriate selection of base types to suit the data being stored and manipulated. Avoiding use of global/static variables. Avoid wrapper classes when not needed | 10% |
| Control Structures | Sensible control structures have been selected and demonstrated: if, elseif – for-loops, while loops and do-while loops. Sensible use of break | 30% |
| Use of methods | Methods have been used to break the code down into sensible sections, including using return value and parameters | 15% |
| Conciseness of code | Have you written your code in such a way that you are not unnecessarily repeating code and have you written it concisely – the more convoluted the solution, the more code you will need to write and the more likely it is you will have errors and find it harder to maintain. Keep it simple. Do control paths naturally converge? Avoid having a contrived runtime pathway through your code. | 10% |
| Testing | A test plan (expected output against input across a sensible choice of inputs) | 15% |
| Defensive coding | Coding to guard against invalid user input – from basic if-test to try and catch, etc | 10% |
| Formatting | Meaningful names for methods, variables and consistent use of case. Consistent formatting including indentation and appropriate use of whitespace (new lines). Use of code comments – this should include introducing what each method does at the start and inter-code comments | 10% |

The University common grading descriptor (provided at the end of this document) will be used to determine marks in each area. The marking scheme embeds the concept of extended work by rewarding the highest marks to those who demonstrate evidence of independent investigation, learning, critical thought, and problem analysis (via good code solutions).

## Submission Process

Your assignment should be submitted electronically through the module Blackboard site as a **single, uncompressed .java file**. This will ensure you can preview the format and submission within the Blackboard preview window. **Make sure you upload the .java file and not the .class file.**

Your last on-time attempt will be viewed and graded (as per university regulations).

Make sure that you upload the correct file by checking once you have submitted (i.e. preview your submission in Blackboard and check it and/or download it again from Blackboard). Mistakes discovered after the deadline cannot be corrected; it is your responsibility to ensure that you submit the correct files by the deadline. You may be asked to provide a walkthrough of your code during which you will need to discuss all aspects of the work you submitted with your grade being subject to a successful walkthrough and discussions of your work.

## Avoiding Plagiarism

Looking for snippets of code on the internet and searching for materials that will help solve parts of a problem is part of software development. If you do this, you **must** add a comment in your code stating the URL for snippets of code that have heavily influenced your solution and a short summary of what you have taken / learnt to demonstrate you understand the code and not just unashamedly lifted it. You should not really be directly copy from the internet (or elsewhere), even once referenced, but making it your own code. There is no need to reference core learning that you apply to this task – for example, if you have learnt how to use arrays in general and applied it here, that is fine.

This assessment is about you demonstrating your knowledge and understanding when it comes to Java programming and not the internet’s, your friends, or Discord’s. If you try to pass off work as your own that isn’t, then an academic conduct meeting will be arranged, and you’ll likely get zero marks. A second offense can lead to withdrawal from the course, so please ask if you’re unsure.

# Assessment Criteria: Using [University Grade Descriptors](https://students.shu.ac.uk/regulations/assessment_awards/University%20Grade%20Descriptors%20(Level%204;%20new%20Level%207).pdf)

Text

Description automatically generated