ADDITIONAL ASSIGNMENT – 202475B ITC228 Programming in Java

Due Date: Wednesday, 5th March

Value: 30%

Submission: Email to FOBJBS-Subject-Admin@csu.edu.au

TASK:

Task 1 Computing Future Investment Value

value: 8 marks

Write a method that computes future investment value at a given interest rate for a specified number of years. The future investment is determined using the following formula:

futureInvestmentValue =

investmentAmount x (1 + monthlyInterestRate)numberOfYears*12

Use the following method header:

public static double futureInvestmentValue(

double investmentAmount, **double** monthlyInterestRate, **int** years)

For example, futureInvestmentValue(10000, 0.05/12, 5) returns 12833.59.

Write a test program that prompts the user to enter the investment amount (e.g., 1000) and the interest rate (e.g., 9%) and prints a table that displays future value for the years from 1 to 30, as shown below:

```
The amount invested: 1000

Annual interest rate: 9%

Years       Future Value

1          1093.80

2          1196.41

...

29          13467.25
```

30 14730.57

Analysis & Design: Describe the problem including input and output in your own words and the major steps for solving the problem.

Task 2 Validating Credit Cards

value: 10 marks

Problem Description:

Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with:

4 for Visa cards

5 for Master cards

37 for American Express cards

6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine if a card number is entered correctly or if a credit card is scanned correctly by a scanner. Almost all credit card numbers are generated following this validity check, commonly known as the Luhn check or the Mod 10 check, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

2 * 2 = 4

2 * 2 = 4

4 * 2 = 8

1 * 2 = 2

6 * 2 = 12 (1 + 2 = 3)

5 * 2 = 10 (1 + 0 = 1)

8 * 2 = 16 (1 + 6 = 7)

4 * 2 = 8

2. Now add all single-digit numbers from Step 1.

4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

3. Add all digits in the odd places from right to left in the card number.

6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

4. Sum the results from Step 2 and Step 3.

37 + 38 = 75

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a long integer. Display whether the number is valid or invalid.

Here are sample runs of the program:

Sample 1:

```
Enter a credit card number as a long integer: 4246345689049834

4246345689049834 is invalid
```

Sample 2:

```
Enter a credit card number as a long integer: 4388576018410707

4388576018410707 is valid
```

Analysis & Design: Describe the problem including input and output in your own words and the major steps for solving the problem.

Task 3 University Subject

Value: 12 marks

For this task you will create a *Subject* class, whose instances will represent the subjects for study at a university. A subject will have a name, just a *String*, and a subject code, which is a six-character *String*. The first three characters of a subject code are alphabetic and the last three are numeric. The first three characters define the subject's discipline area. A subject code must be unique but do not need to check subject name for uniqueness.

You will also write a *TestSubject* class to test the use of your *Subject* class. In particular this will maintain an array of subjects. In order to manage the uniqueness of the subject codes, your program will need to display information about existing subject codes as well as checking that any new subject code supplied by the user is not the same as any existing subject code.

The following state and functionality should be provided for the *Subject* class:

- Two fields will hold the subject's name (e.g. Programming in java 1) and the six-character subject code (e.g. ITC228).

- A constructor will allow a name and a new, validated subject code to be provided when a new subject is created.
- Getters will provide access to the attributes.
- An accessor method called *codeMatches* will return a boolean value indicating if the subject's code matches the string argument provided. "Matches" is used here in the same sense as for the *matches* method of the *String* class.
- A *toString* method will return a string containing the subject code and subject name.

To assist with managing subject codes and their uniqueness you will provide the *Subject* class with some class methods as follows (you may add more method9s) if you need):

- An *isValidCode* method will accept a string that is a possible new subject code, and return a boolean indicating whether it satisfies the structural requirements for a subject code (i.e. first three characters are letters and last three characters are digits).
- A *codeExists* method will accept an array of *Subject* objects and a possible new subject code. It will return a boolean indicating whether that code has already been allocated to one of the subjects in the array.

Your *TestSubject* program will perform the following sequence of actions, using good design techniques such as in the appropriate use of methods:

- An initial array of *Subject* objects will be created from any data in a file that was previously saved by the program (not using programming in java, just open a text file and write subject name and code and then save the file). You need to read those data from the file and process other requirements (using java programming)
- The user interaction will then proceed to allow the user to add one or more new subjects to the array. If the user wishes to add new subjects, the existing subjects should be displayed. Each subject code entered by the user should be checked against the existing subject codes. The user can enter any new subject, but only non-existing subject codes and their names should be added in the subject list, otherwise, give opportunity to enter new code if the entered subject code already exists in the list. The user should be given the choice of repeating the processing for more subjects.
- When the user has finished adding subjects, and only if subjects have indeed been added, the program will overwrite the data file with the updated data if anyone open the file using file explorer, he/she can see all subjects including the newly added subjects.

You need to submit a single zip file containing:

1. All java and class files
2. A pdf file, analysis & design descriptions, a snapshot of the program output, and UML design (where required, see marking guide)

MARKING CRITERIA:

Marking guide of the specific tasks:

Assessment 3 (Total marks 30)					
Task 1 (total marks 8)					
Criteria	**Marks**	**HD**	**DI**	**CR**	**Pass**
a. Execution: Program launches, executes and terminates without crashing; program executes as specified.	1.0	Provide java file and it executes without crashing towards intended output (up to 1.0)	Provide java file and it executes without crashing towards intended output but one method is missing (up to 0.85)	Provide java file and it executes without crashing towards intended output but one/two methods are missing (up to 0.75)	Provide java file and it executes without crashing towards intended output but a significant number of methods are missing (0.5)
b. Program design & implementation: An appropriate main method with the inputs, processing and outputs specified in the question	4.0	Execute and compute the correct population with proper java structure, logical flow for any value and correct outputs (up to 4.0)	Execute and compute the correct population with proper java structure, logical flow for any value and outputs with minor errors (up to 3.0)	Execute and compute the population for all given examples correctly (up to 2.5)	Execute and compute the population correctly for some cases of the given examples (up to 2.0)
c. Presentation: Code uses good style (indentation, comments)	1.0	Maintain proper naming convention of all variables, proper indentation on each block/line of code(s), and provide important	Maintain proper naming convention of all variables, proper indentation on each block of codes, and provide comments	Provide a number of variable names but not all by maintaining proper naming convention, occasionally proper indentation, and comments on only important	Provide arbitrary variable names, no proper indentation, and very few comments (up to 0.5)

		inline comments (up to 1.0)	on only important calculation (up to 0.75)	calculation (up to 0.65)	
d. Submission: The documents with all components (java code, testing outputs and analysis & design descriptions)	2.0	Provide all components with convincing various output and analysis & design descriptions (up to 2.0)	Provide all components with various output and analysis & design descriptions (up to 1.75)	Provide all components (up to 1.5)	Provide all components (up to 1)
Task 2 (total marks 10)					
Criteria	**Marks**	**HD**	**DI**	**CR**	**Pass**
a. Execution: Program launches, executes and terminates without crashing; program executes as specified.	1.0	Provide java file and it executes without crashing towards intended output (1.0)	Provide java file and it executes without crashing towards intended output with almost all options are included (0.75)	Provide java file and it executes without crashing towards intended output but few options are missing (0.65)	Provide java file and it executes without crashing towards intended output but a significant number of options are missing (0.5)
b. Program design & implementation: An appropriate main method with the inputs, processing and	6.0	Execute and compute the correct population with proper java structure, logical flow	Execute and compute the correct population with proper java structure, logical flow	Execute and compute the population for all given examples correctly (up to 4)	Execute and compute the population correctly for some cases of the given

outputs specified in the question		for any value and correct outputs (up to 6.0)	for any value and outputs with minor errors(up to 5.0)		examples (up to 3)
c. Presentation: Code uses good style (indentation, inline comments)	1.0	Proper indentation and comments in each block and major lines (up to 1.0)	Proper indentation and comments in each block and major lines (up to 1.0)	Proper indentation and comments in each block and major lines (up to 1.0)	Provide sample outputs (up to 0.5)
d. Submission: The documents with all components (java code, testing outputs and analysis & design descriptions)	2.0	Provide all components with convincing various output and analysis & design descriptions (up to 2.0)	Provide all components with various output (up to 1.75)	Provide all components (up to 1.5)	Provide all components (up to 1.0)
Task 3 (total marks 12)					
Criteria	**Marks**	**HD**	**DI**	**CR**	**Pass**
a. Execution: Program launches, executes and terminates without crashing; program executes as specified.	1.0	Provide java file and it executes without crashing towards intended output (up to 1.0)	Provide java file and it executes without crashing towards intended output with almost all options are included (0.75)	Provide java file and it executes without crashing towards intended output but few options are missing (0.65)	Provide java file and it executes without crashing towards intended output but a significant number of options are missing (0.5)

b. UML design: For the Subject class	1.0	UML with all components and their modifier, argument, and return type (up to 1.0)	UML with all components and their modifier, argument, and return type (up to 1.0)	UML with all components and their modifier, argument, and return type (up to 1.0)	UML with all components (up to 0.5)
c. Program design & implementation: Subject (4), TestSubject (2), and file read/write (2) classes are implemented as specified, showing good logic.	8.0	Implement and integrate Subject and TestSubject classes with all methods using file read & write maintaining logical flow (8)	Implement and integrate Subject and TestSubject classes with all methods using file read & write with minor error (6-7)	Implement and integrate Subject and TestSubject classes with majority of the methods using file read & write (4.5-5.5)	Implement and integrate Subject and TestSubject classes with some methods using file read & write (4.0)
d. Presentation: Code uses good style (indentation, comments)	1.0	Maintain proper naming convention of all variables, proper indentation on each block/line of code(s), and provide important inline comments (up to 1.0)	Maintain proper naming convention of all variables, proper indentation on each block of codes, and provide comments on only important calculation (up to 0.75)	Provide a number of variable names but not all by maintaining proper naming convention, occasionally proper indentation, and comments on only important calculation (up to 0.65)	Provide arbitrary variable names, no proper indentation, and very few comments (up to 0.5)

e. Submission: The documents with all components (java code and testing outputs)	1.0	Provide all components with convincing various output (up to 1.0)	Provide all components with various output (up to 0.75)	Provide all components (up to 0.65)	Provide all components (up to 0.5)