

**School of Engineering and Information Technology**

**ASSIGNMENT COVER SHEET**

* Please complete and insert this form as the first page of EACH electronic assignment.
* Submit the assignment with attached coversheet electronically as per the instructions in the Assignment Question sheet.
* Please make sure you keep a copy of the assignment.

**Student Details**

|  |  |  |  |
| --- | --- | --- | --- |
| Surname | Tee | Given name | Yee Kang |
| Student Number | 34315323 | Email | Yeekang88@gmail.com |

**Assignment details**

|  |  |  |  |
| --- | --- | --- | --- |
| Unit name | Principles of Computer Science | Unit Code | ICT167 |
| Unit Coordinator | Kevin Wong | Tutor/Tutorial time | Aaron Yeo Sze Wee |
| Due date/time | 26/Jun/2021 – 11:59PM | Submission date | 26/Jun/2021 |
| Assignment title | FT MUR T221 ICT167 C – Assignment 1 | | |
| Other information | - | | |

***All forms of plagiarism, cheating and unauthorized collusion are regarded seriously by the University and could result in penalties including failure in the unit and possible exclusion from the University. If in doubt, please contact the Unit Coordinator.***

**Student’s Declaration**

Please double click on all the check boxes.

✓

Except where I have indicated, the work I am submitting in this assignment is my own work and has not been submitted for assessment in another unit.

✓

This submission complies with Murdoch University policies regarding plagiarism, cheating and collusion.

✓

I have retained a copy of this assignment for my own records.

**External Documentation**

* **Title :** FT MUR T221 ICT167 C – Assignment 1
* **Author :** Tee Yee Kang
* **Date :** 26/6/2021
* **File Name :** FTC-34315323-Assignment1
* **Purpose :** This program is allowed all the users (customers) to enter and store their name and coin amount into the system and then perform some different tasks provided by the system.

1. **Requirements/Specification:**

This program allowed the user to create multiple MoneyChange objects and store them into a MoneyChange array. In order to create the MoneyChange object, the user needs to enter a name and coin amount for the object. The data types of the name and coin amounts are string and integer, respectively. In addition, the value of the coin amount must be between 5 to 200 and divisible by 5. The default values of the MoneyChange object are “None” for name and 0 for coin amount. User can choose to enter manually or hard code the value of the object. After the object is created, the user can perform many different tasks on the MoneyChange object during the menu stage:

* Task 1 - Enter a name and display change to be given for each denomination
* Task 2 - Find the name with the smallest amount and display change to be given for each denomination
* Task 3 - Find the name with the largest amount and display change to be given for each denomination
* Task 4 - Calculate and display the largest number of coin denomination, and the total number of the coin
* Task 5 - Exit

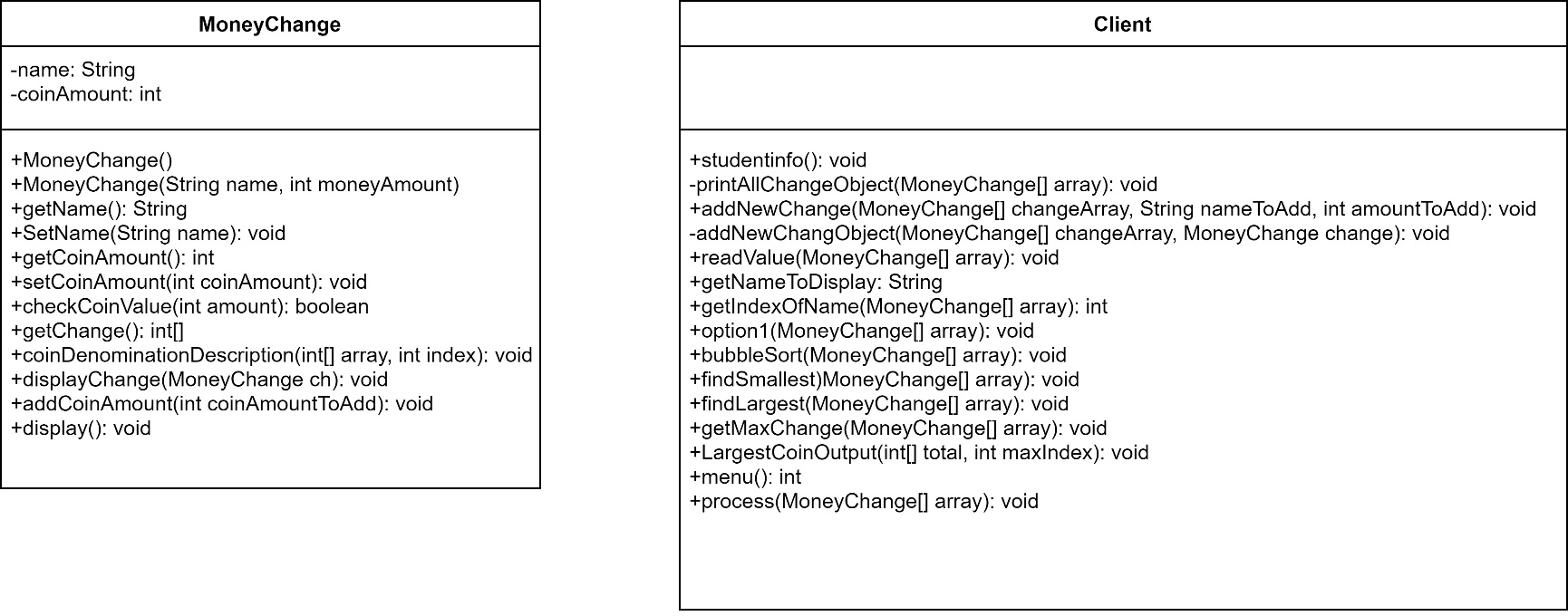
Task 1 to 3 will display the coin denomination of the MoneyChange object based on its condition. Task 4 will display the largest coin denomination in the MoneyChange array. The menu will keep executing after completing a task. The program will only stop executing when the user enters option 5 to exit the program. During the execution, the program also conducts some validation check for user input. The program is able to handle multiple error input (e.g.: invalid coin amount entered, cannot find the name, etc.). When an error is detected, the program will display an error message and prompt the user to enter input again if necessary.

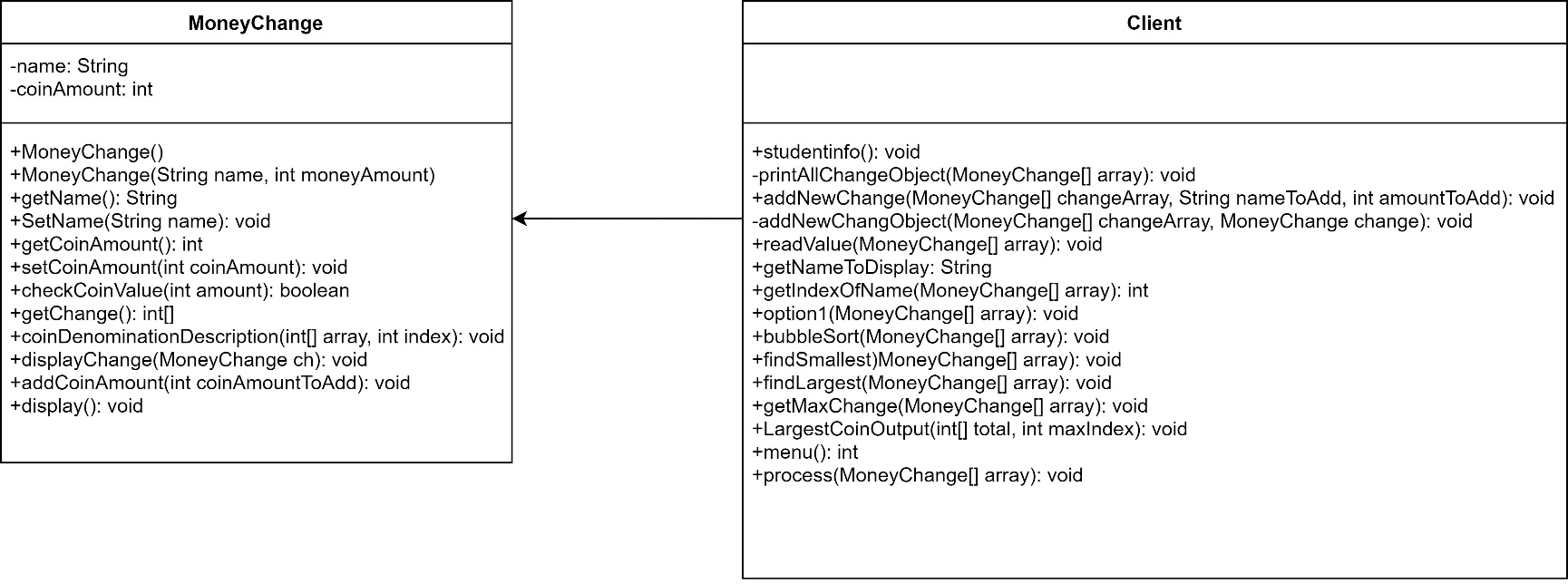
1. **User Guide:**

* To run the program, users only need to insert the entire file into Java SE (JDK).
* At the beginning, the program will ask the user to create the MoneyChange object. User should enter 0 to manually created or 1 to hard code all the value of the MoneyChange object to complete the creation. If the user enters any number apart from 0 or 1, the program will display an error message and prompt the user to enter again.
* If the user chooses to create the MoneyChange object manually, the user will need to enter a name and the coin amount for the object. The program will guide user enters each value one by one. If the user has entered an invalid coin amount (coin amount must be 5-200 and divisible by 5), the program will display an error message to remind the user and ask the user to enter the value again.
* The program will automatically create the MoneyChange objects if the user entered 1 to hard code.
* Next, the program will ask the user whether to continue reading the MoneyChange object after each object is created. The program will keep reading and creating object if the user’s option is y (yes). Program stop reading until the user enters n (no) or the maximum size of the MoneyChange array is reach (array size is 50).
* User can also increase the coin amount of an existing MoneyChange object by entering the same object name. The program will add the new passed in coin amount to the balance of the same name MoneyChange object.
* After completing the creation of the MoneyChange object, the output will display all the objects, including the name and coin amount. Next, the program will display a menu to allow the users to perform different tasks as they wish.
* In the menu stage, the user can enter 1 to retrieve the coin denomination of a particular MoneyChange object by entering the object's name. Users have to follow the program’s instruction to enter the object's name (not case sensitive), which the user wishes to retrieve. The program will then search for the name and display the coin denomination of the object. If there is no matching name, the program will display “Name not found!!” to remind the user.
* After completing the task, the program will continue displaying the menu for the user to perform the next task. The menu will keep displaying after completing a task until the user enters 5 to stop.
* If the user wants to retrieve the coin denomination with the smallest coin amount, the user can enter 2 in the menu stage. After the user enters option 2, the program will automatically display the user’s name holding the smallest coin amount followed by the coin denomination.
* On the other hand, to retrieve the coin denomination with the largest coin amount, the user only needs to enter 3 in the menu stage. Then, the program will automatically display the user’s name who is holding the largest coin amount, followed by the coin denomination.
* Option 4 allows the user to retrieve the coin denomination with the largest number among all other coin denomination. This is the same as options 2 and 3. The user does not need to enter any input values, and the program will automatically display the largest coin denomination and the number of particular coin denominations.
* There might be more than one result in option 4 if there is more than one coin denomination with the same number of largest numbers of coin denomination.
* If the user wants to stop executing the program, the user can enter 5 in the menu stage to exit.
* The program will display an error message to remind the user when the user has entered any option other than 1 to 5 or not a numerical value.

1. **Structure/Design/Algorithm:**

**UML Class Diagram**

****

**Structure ChartExtra instance variables & methods in MoneyChange Class**

\*Pseudocode is in red colour

1. **MoneyChange()**

I have set the name and coin amount of the default constructor to “None” and 0, respectively.

1. **coinDenominationDescription (int [] array, int index)**

This method is just the description/output of all the coin denominations for task/option 1. After the user enters a name for task 1, the program will pass in the array that store the coin denomination and the index of the name user have entered previously. Then this method will print out the description of a particular coin denomination.

Display\_coin\_denomination

Method takes in an int array and a value of int type

CASE OF (value)

1: Print number with denomination of $2

2: Print number with denomination of $1

3: Print number with denomination of 50 cents

4: Print number with denomination of 20 cents

5: Print number with denomination of 10 cents

6: Print number with denomination of 5 cents

END

1. **displayChange(MoneyChange ch)**

This method is like a controller and help the user to calculate and display the coin denomination of the customer by calling the getChange() and coinDenominationDescription() methods.

Call\_calculate\_and\_display\_coin\_denomination

Create an array of int type and size of 6

Call getChange() method to and store in the array created previously

REPEAT

IF coin denomination > 0

Call coinDenominationDescription to print output

ENDIF

UNTIL (reach the array size)

END

1. **display()**

This method is to display the name and coin amount. Therefore, the user does not need to type "System.out.print" all the time to display the details of the MoneyChange object. The user just has to call this method (e.g. objectName.display()) to print out the details of MoneyChange object.

Display\_detiails\_of\_MoneyChange\_object

Print name of MoneyChange object (name)

Print coin amount of MoneyChange object (coinChangeAmount)

END

**Outline of design – MoneyChange Class**

* The instances variables of MoneyChange class are a String name and int coinChangeAmount.
* There are 2 constructors for this class – the default constructor (as mentioned before) and a constructor with 2 parameters.
* The default constructor required a name and a valid coin amount. A valid coin amount must be between 5-200, divisible by 5 and is int value.
* MoneyChange class also contain getter and setter methods (getName(), setName(), getCoinAmount(), setCoinAmount() ). The program uses all these methods to get values and set values of MoneyChange object.
* **checkCoinValue(int amount)**

This method is to check and make sure the user has entered a valid coin amount when creating the MoneyChange object (This method will be called in the constructor and setCoinAmount method).

Boolean checkCoinValue(int amount)

Set boolean valid = false

IF amount >5 AND amount <200 AND amount%5=0

Set valid = true

ENDIF

Return valid

END

* **getChange()**

This method is to calculate the coin denomination for the customer. The change denomination consists of $2, $1, 50 cents, 20 cents, 10 cents, and 5 cents. Each array index represents a value of change denomination. Pass the coin amount through the each while loop to find for number of each change denomination and the running balance will also decrease after a change denomination is obtained. Return the array that store all change denomination after completing the calculation.

Calculate\_coin\_denomination

Create an array of int type

Retrieve the coinChangeAmount and set to tempCoinAmount

Set all array elements to 0

DOWHILE tempCoinAmount >200

tempCoinAmount - = 200

array[0] ++

ENDDO

DOWHILE tempCoinAmount >100

tempCoinAmount - = 100

array[1] ++

ENDDO

DOWHILE tempCoinAmount >50

tempCoinAmount - = 50

array[2] ++

ENDDO

DOWHILE tempCoinAmount >20

tempCoinAmount - = 20

array[3] ++

ENDDO

DOWHILE tempCoinAmount >10

tempCoinAmount - = 10

array[4] ++

ENDDO

DOWHILE tempCoinAmount >5

tempCoinAmount - = 5

array[5] ++

ENDDO

Return array of coin denomination

END

* **addCoinChangeAmount(int coinChangeAmountToAdd)**

This method is to add the passed in coin amount to the current coin amount.

Add\_coin\_amount

coinChangeAmount += coinChangeAmountToAdd

END

**Outline of design – Client Class**

* In the main class, the program allows the user to choose between hard-coding or manually creating MoneyChange objects. The program will guide the user on creating the MoneyChange object if the user chooses to create it manually. The user needs a name and a valid coin amount to create the object. On the other hand, the program will automatically create if the user chooses to hardcode.

Main

Client class

Print student information

Get user’s choice

DOWHILE choice != 0 OR choice !=1

Print error message for invalid choice

Get choice

ENDDO

Set MoneyChange array size

IF choice == 0

Read MoneyChange object details

ELSE

Hard code MoneyChange objects

ENDIF

Print MoneyChange object (name, coinChangeAmount)

Get menu option

END

* **studentInfo()**

This method is all the information of student.

Student\_Information

Print student\_information

END

* **printAllMoneyChangeObject(MoneyChange[] array)**

This method is to print out all the MoneyChange objects including the name and coin amount. The user can reuse this method to print out the MoneyChange object by passing in the MoneyChange array instead of entering in for loop every time.

Print\_All\_MoneyChange\_Object

REPEAT

Retrieve MoneyChange object

IF array elements != null

Print MoneyChange object (name, coinChangeAmount)

ENDIF

UNTIL (reach pass\_in\_array size)

END

* **addNewChange(MoneyChange[] changeArray, String nameToAdd,int amountToAdd)**

This method is to create and store the MoneyChange object into array by passing in the MoneyChange array, String name and the coin amount with int value. If there is same name object, this method will retrieve the object with the same name and add the both coin amount. If there is no march with same name, this method will call addNewChangeObject() method to create a new MoneyChange object.

Add\_new\_change

Set found to false

REPEAT

Retrieve MoneyChange object and set to currentChange

IF currentChange != null

Set currentName = name of first MoneyChange object

IF currentName == nameToAdd

Add amountToAdd to first MoneyChange object

Set found to true

ENDIF

ENDIF

UNTIL (reach pass\_in\_array size)

IF found == true

Print a statement to tell user amount successfully added

ELSE

Create new MoneyChange object

ENDIF

END

* **addNewChangeObject(MoneyChange[] changeArray, MoneyChange change)**

This method is to find for empty slot and create/store the MoneyChange object into the MoneyChange array. The arguments for this method are a MoneyChange array and a MoneyChange object. This method will be called in the addNewChange() method when there is no same name found.

Add\_new\_MoneyChange\_object

Set found to false

REPEAT

Retrieve Moneychange object and set to currentChange

IF currentChange == null

Store the passed in change in passed in MoneyChange array Set found to true

ENDIF

UNTIL (reach pass\_in\_array size OR successful store an object)

IF found == true

Print statement to tell user successfully added new object

ELSE

Print statement to tell user no more empty slot

END

* **readValue(MoneyChange[] array)**

This method is the controller of addNewChange() method. The purpose of this method is to ask the name and coin amount from the user in order to create MoneyChange object. Inside this method will check for the coin amount by calling the checkCoinValue() method. In addition, the program will ask the user whether to continue creating MoneyChange objects after each object is created. This method will keep running until the user have enter n (no).

Read\_value

REPEAT

Get name

Get coin amount

DOWHILE coin amount is not valid

Get coin amount

ENDDO

Create MoneyChange object (call addNewChange() method)

Get continue\_option

UNTIL (reach pass\_in\_array size OR continue\_option = N)

END

* **getNameToDisplay()**

This method is for menu option/task 1. User can use this task to retrieve the coin denomination for specific MoneyChange object by entering the name of the object. I split the whole task into 3 methods. This method is the first part and purpose is to ask the user for the name to display.

Get\_name\_to\_display

Get name

Return name

END

* **getIndexOfName(MoneyChange[] array)**

This method is the second part/method of menu option 1. The purpose for this method is to find the index of the name of MoneyChange object user entered previously. Then return the index for third method to use. If there is no same name found, the method will return the initial value of index (-1). The parameter of this method in an array of MoneyChange type.

Get\_index\_of\_name

Set index to -1

Get name (by calling getNameToDisplay() method)

Set name to name01

REPEAT

IF MoneyChange\_object !=null

IF name01 = name of MoneyChange\_object

Set index = index of MoneyChange\_object

ENDIF

ENDIF

UNTIL (reach pass\_in\_array size OR same name found)

Return index

END

* **option1(MoneyChange[] array)**

This method is the third part/method of menu option 1. This method will get the index of the name of MoneyChange object by calling the getIndexOfName() method and search through the MoneyChange array and display the corresponding coin denomination by calling the displayChange() method. If the index to search is the initial value (-1), it means that no matching object in the MoneyChange array.

Option1

Get index\_of\_name (by calling getIndexOfName() method)

Set index\_of\_name to idx

IF idx < 0

Print statement to tell user no name found

ELSE

Display coin denomination

ENDIF

END

* **bubbleSort(MoneyChange[] array)**

This method is to sort the passed in MoneyChange array in ascending order. The function of menu task/option 2 and 3 is to display the coin denomination MoneyChange object with smallest and largest coin amount. So, in order to do that, I first sorted the array in ascending order. Therefore, option 2 and 3 can retrieve the MoneyChange object with the smallest and largest coin amount base on the index.

Bubble\_sort

Set swap to true

DOWHILE swap == true

REPEAT

IF pass\_in\_array != null

IF coinAmount\_in\_index\_0 > coinAmount\_index\_1

Swap the to array

Set swap to true

ENDIF

ENDIF

UNTIL (reach pass\_in\_array size)

ENDDO

END

* **findSmallest(MoneyChange[] array)**

This method is for menu task/option 2 and purpose is to display the coin denomination of the MoneyChange object with the smallest coin amount. The method will first sort the passed in MoneyChange array by calling bubbleSort() method and pass in MoneyChange array. Next, print out the coin denomination by calling the displayChange() method with arguments array[0]. Using index 0 is because after sorted the array in ascending order, the smallest coin amount will be in index 0.

Find\_smallest

Sort array in ascending order

Print coin denomination (by calling displayChange() method)

END

* **findLargest(MoneyChange[] array)**

This method is for menu task/option 2 and purpose is to display the coin denomination of the MoneyChange object with the largest coin amount. The concept of this is similar to menu option 2. The method will first sort the passed in MoneyChange array by calling bubbleSort() method and pass in MoneyChange array. Next, print out the coin denomination by calling the displayChange(). However, we cannot use index array.length -1 to get the largest coin amount. Because there might be null value in the MoneyChange array. So, I use a for loop to find the largest coin amount in the not null array elements

Find\_largest

Sort array in ascending order

Set maxIndex to 0

REPEAT

IF pass\_in\_array\_elements != null

IF coinAmount\_index0 < coinAmount\_idx

Set maxIndex to idx

ENDIF

ENDIF

UNTIL (reach pass\_in\_array size)

Print coin denomination (by calling displayChange() method)

END

* **getMaxDenomination(MoneyChange[] array)**

This method is for menu task 4. I split this task into 2 different methods. One method is for output design, this method is to obtain the largest number of coin denominations. First, I created 2 int arrays of size 6 for the getChange() method and the running total of each coin denomination. Next, I use a nested for loop to calculate and find the running total of each coin denomination. The outer for loop will call the getChange() method calculate the coin denomination for each object. The inner for loop will store the running total of each coin denomination after each outer calculation has done. After that, I use another for loop to find the largest number of coin denominations in the array, which stores the running total of coin denominations and retrieve the index of the largest number of coin denomination. Lastly, I use a for loop to display the output by calling the largestCoinOutput() method. The if statement inside the for loop is to check if there is more than one largest number of coin denomination.

Get\_max\_denomination

Create an in array of size 6 – change

Create an in array of size 6 – total

REPEAT

IF pass\_in\_array != null

Change = pass\_in\_array[outer]getChange()

REPEAT

total[inner] += change[inner]

UNTIL (6 counts)

ENDIF

UNTIL (reach pass\_in\_array size)

Set maxIndex to 0

REPEAT

IF total[maxIndex] < total[idx]

Set maxIndex to idx

UNTIL (reach total array size)

REPEAT

IF total[maxIndex] == total[idx]

Set maxIndex to idx

Print largest number of coin denomination

ENDIF

UNTIL (reach total array size)

END

* **largestCoinOutput(int[] total, int maxIndex)**

This method is the output of menu task/option 4. This method takes in an int array and the index of the largest number of coin denomination. The switch case will print out the output base on the argument index.

Largest\_coin\_output

CASE OF (pass\_in\_maxIndex)

1: Print the output with denomination of $2

2: Print the output with denomination of $1

3: Print the output with denomination of 50 cents

4: Print the output with denomination of 20 cents

5: Print the output with denomination of 10 cents

6: Print the output with denomination of 5 cents

END

* **menu()**

This method is the output of menu. Inside this method will print out all the description for each menu task and read the user’s choice. Lastly return the user’s choice.

Menu

Print all menu description

Read user’s choice

Return user’s choice

END

* **process(MoneyChange[] array)**

This is the main controller for the whole program. The switch case will call different methods according to user’s choice on menu stage. Inside the main class just need to call this process() method and pass in appropriate argument to run all tasks. If the user enters a non numerical value the error exception will catch and print an error message.

Process

Set option to 0

DOWHILE option != 5

Get option

CASE OF (option)

1: Call task 1

2: Call task 2

3: Call tasks 3

4: Call task 4

5: Exit the program

Default: Print error message

ENDDO

END

1. **Limitation**

**--**

1. **Testing**

**The example of user’ inputs are as follow (not included the error input and yes/no continue option):**

Name: yk

Coin amount: 55

Name: aa

Coin amount: 185

Name: yk

Coin amount: 50

Name: cc

Coin amount: 75

Name: ii

Coin amount: 55

Name: ee

Coin amount: 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test id** | **Test description/justification – what is the test for and why this particular test** | **Actual data for this test** | **Expected output** | **Actual desk check result when desk check is carried out** | **Desk check outcome – Pass/Fail** |
| 1 | Test for the beginning of the program. User should enter 0 for user input or 1 to hard coded to create MoneyChange object. If user enter any thing apart from 0 and 1, program should display error message and ask user to enter again | 5 – (5 is not the acceptable input and program should handle this error and ask user enter again) | Invalid choice !! Enter choice again | Invalid choice !! Enter choice again | Pass |
| 2 | Test for the beginning of the program. User should enter 0 for user input or 1 to hard coded to create MoneyChange object. If user enter 1, program should automatically read the hardcoded object and create the MoneyChange object | 1 – (1 is the input for hardcoded MoneyChange object) | Name:A  CoinChangeAmount:100  Name:B  CoinChangeAmount:55  Name:C  CoinChangeAmount:45  Name:D  CoinChangeAmount:55  Name:E  CoinChangeAmount:15  Name:F  CoinChangeAmount:25  Name:G  CoinChangeAmount:10  Name:I  CoinChangeAmount:60  Name:J  CoinChangeAmount:95 | Name:A  CoinChangeAmount:100  Name:B  CoinChangeAmount:55  Name:C  CoinChangeAmount:45  Name:D  CoinChangeAmount:55  Name:E  CoinChangeAmount:15  Name:F  CoinChangeAmount:25  Name:G  CoinChangeAmount:10  Name:I  CoinChangeAmount:60  Name:J  CoinChangeAmount:95 | Pass |
| 3 | Test for the beginning of the program. User should enter 0 for user input or 1 to hard coded to create MoneyChange object. If user enter 0, program should should start asking user for the name and coin amount to create the MoneyChange object | 0 – (0 is the input for user input MoneyChange object) | Please enter the name of the person:  Please enter the coin value for the person:  Program should ask user for name and coin amount | Please enter the name of the person:  Please enter the coin value for the person: | Pass |

**Test Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | After the user have enter n (no) to stop creating MoneyChange object, the program should first display all the MoneyChange object followed by the menu. | N – (n means no and upper/lower case are acceptable, program should stop reading value for creating object) | Name:yk  CoinChangeAmount:105  Name:aa  CoinChangeAmount:185  Name:cc  CoinChangeAmount:75  Name:ii  CoinChangeAmount:55  Name:ee  CoinChangeAmount:5 | Name:yk  CoinChangeAmount:105  Name:aa  CoinChangeAmount:185  Name:cc  CoinChangeAmount:75  Name:ii  CoinChangeAmount:55  Name:ee  CoinChangeAmount:5  (Program successfully display the expected output) | Pass |
| 5 | Test for menu task 1. When user enter task 1 in the menu stage. Program should ask for a customer name and display the coin denomination of the particular customer. | Menu: 1  Name: yk  (yk is an existing name of MoneyChange object, program should be able to retrieve the particular MoneyChange object) | =====< Change >=====  $1 : 1  5 cents : 1  ===================== | =====< Change >=====  $1 : 1  5 cents : 1  ===================== | Pass |
| 6 | Not exist name test for menu task 1. When user enters a name that not exist in the MoneyChange array, program should display a message to tell the user there is no such customer in the system. | Menu: 1  Name: NoName  (Program should fail to find the particular customer and display not found message. | Name not found!! | Name not found!! | Pass |
| 7 | Test for menu task 2. Task 2 allow user to retrieve the coin denomination of the customer with the smallest coin amount. No input value required for this task. Program should automatically display the output after user selects the task/option. | Menu: 2  (Program should automatically display the coin denomination of the customer who holding the smallest coin amount. In this case the smallest is 5. | The smallest coin amount is : 5  ===< Change >===  5 cents : 1  ================= | The smallest coin amount is : 5  ===< Change >===  5 cents : 1  ================= | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 8 | Test for menu task 3. Task 3 allow user to retrieve the coin denomination of the customer with the largest coin amount. No input value required for this task. Program should automatically display the output after user selects the task/option. | Menu: 3  (Program should automatically display the coin denomination of the customer who holding the largest coin amount. In this case the largest is 185. Output must fulfill the “give as much of the higher valued coins as possible” requirement. | The largest coin amount is : 185  ===<Change >===  $1 : 1  50 cents : 1  20 cents : 1  10 cents : 1  5 cents : 1  ================ | The largest coin amount is : 185  ===<Change >===  $1 : 1  50 cents : 1  20 cents : 1  10 cents : 1  5 cents : 1  ================ | Pass |
| 9 | Test for menu task 4. Task 4 will find and display the largest number of coin denomination among the 6 different coin denomination. No user input required for this task. Program should perform the task automatically after user selects the task/option. | Menu: 4  (Program should automatically display the largest number of coin denomination. Result might have more than 1 coin denomination if there is more than 1 coin denomination with the same largest number of coin denomination. In this case, the largest is 5 cents. | The largest number of coin denomination is:  5 cent  The total number of 5 cent is: 5 | The largest number of coin denomination is:  5 cent  The total number of 5 cent is: 5 | Pass |
| 10 | Test for task 5. Task 5 is to allow the user to exit the program. It is because the program will continue executing after each task is completed. Program only stop when user enter 5 to quit. | Menu: 5  (Program will display a greeting sentence and stop running) | Thank you Bye!! | Thank you Bye!! | Pass |
| 11 | Test for invalid input for menu stage. The menu stage only accepts a numeric value between 1 to 5, if user enter any number apart from 1-5, program will display an error message and continue running instead of crash the whole program. | Menu: 8  (8 is not an acceptable value. Program should be smart enough to handle this error input.) | Invalid option! | Invalid option! | Pass |
| 12 | Test for invalid input for menu stage. The menu stage only accepts a numeric value between 1 to 5, if user enter a non-numeric value, program will also display an error message and continue running instead of crash the whole program. | Menu: b  (b is not a numeric value. Program should be able to handle this error input | Use a numerical value ! | Use a numerical value ! | Pass |

**Results of Program Testing**

* **Hard codes**

The following table is the testing for hard codes MoneyChange object. All the codes are already in the client class. User only need to choose 1 at the beginning of the system. Hard codes MoneyChange object should be able to perform all the tasks at the menu stage. The test also includes some error input from user. The program should be able to handle some of these error input instead of crash the program.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test id** | **Test description/justification – what is the test for and why this particular test** | **Actual data for this test** | **Expected output** | **Actual program output when test is carried out** | **Test run outcome – Pass/Fail** |
| 1 | According to the requirement, the program need to display the student information at the beginning. Once we run the program, the student information will display at the very beginning. | No data required | Student information | Student information displayed at the beginning of the program | Pass |
| 2 | After the student info, program will prompt user to choose 0 (read user input) or 1 (hard coded input). If user entered value less than 0 or greater than 1, program will display error message and prompt user choice again. | 5 – (5 is out of the acceptable range, so that we can test for error message) | Error message and prompt for user input again | Successfully display error message and ask user to enter a new input again | Pass |
| 3 | After the student info, program will prompt user to choose 0 (read user input) or 1 (hard coded input). When user entered 1, hardcoded objects will be created and display out. Then program will display the menu. | 1 – (1 is the acceptable option, program should read in all hardcoded MoneyChange objects automatically) | Display all MoneyChange objects and menu | Successfully display all objects and menu | Pass |
| 4 | When user entered option 1 in menu stage, program should ask for a name of MoneyChange object from user and display the coin denomination for that MoneyChange object. | Menu: 1  Name: a  -(test for task 1 with an existing object name) | ===< Change >===  $1 : 1  ================ | Program display the coin denomination of MoneyChange object a | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | Option 1 in menu stage will ask user for a name and display its coin denomination. When user entered a name that not exist in the MoneyChange, program will display “ Name not found!! ” to tell the user there is not match object. | Name: p  -(test task 1 with a name that no matching object) | Name not found!! | Program display the expected result to tell user that the name user has entered is not found. Because There is no customer p in the hard coded MoneyChange object | Pass |
| 6 | When user entered option 2 in the menu stage, program will find the smallest coin amount in the MoneyChange array and display its coin denomination. | Menu: 2 –  (Test for task 2 and program should display the coin denomination of object with the smallest coin amount) | G has the smallest coin amount  The smallest coin amount is : 10  ===< Change >====  10 cents : 1  ================= | G has the smallest coin amount in hardcoded object. Program successfully display the coin denomination of the MoneyChange object with smallest coin amount | Pass |
| 7 | When user entered option 3 in the menu stage, program will find the largest coin amount in the MoneyChange array and display its coin denomination. | Menu: 3 –  (Test for task 3 and program should display coin denomination of object with the largest coin amount) | A has the largest coin amount  The largest coin amount is : 100  ====< Change >===  $1 : 1  ================== | A has the largest coin amount in hardcoded object. Program successfully display the coin denomination of the MoneyChange object with the largest coin amount. | Pass |
| 8 | When user entered option 4 in the menu stage, program will find and display the largest total coin denomination among all hardcoded MoneyChange objects. | Menu: 4 –  (Test for task 4 and program should display the largest number of coin denomination. In hardcoded case, largest number of coin denomination is 5 cent | The largest number of coin denomination is:  5 cent  The total number of 5 cent is: 6 | The largest number of coin denomination in hardcoded MoneyChange object is 5 cent. Program successfully get the expected output. | Pass |
| 9 | When user entered option 1-4 in the menu stage, program will keep running and display the menu after each task has completed (option 1-4). Program will stop execute when user enter option 5. | Menu: 5 –  (Test if the program will terminate when user enter 5) | Thank you, Bye!! | Program display the expected sentence and stop executing. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | Menu stage allowed user choose between tasks 1 – 5. If user enter number with negative or greater than 5 values, program will display error message and ask user to enter the option again. | Menu: 8 –  (There is no task 8 and test if the program will display an error message to remind the user) | Invalid option! | Program display the expected output and redisplay the menu for user to enter again. | Pass |
| 11 | Menu stage allowed user to choose between tasks 1 – 5. If user enter a non-numerical value, program will display error message and ask user to enter the option again. | Menu: b –  (b is not a numerical value and test is the program will display an error message to remind the user instead of crash the program.) | Use a numerical value! | Program display the expected output and redisplay the menu for user to enter again. | Pass |

**Screen capture of program’s output – Hardcoded**

|  |  |
| --- | --- |
| Test id | Program Output |
| 1 |  |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |
| 4 |  |

|  |  |
| --- | --- |
| 5 |  |
| 6 |  |

|  |  |
| --- | --- |
| 7 |  |
| 8 |  |

|  |  |
| --- | --- |
| 9 |  |
| 10 |  |

|  |  |
| --- | --- |
| 11 |  |

* **User’s Input**

The following table is the testing for user’s input of MoneyChange object. User has to enter the name and coin amount manually to create the MoneyChange object. The program will guide the user to create objects step by step. If the user enters a name of MoneyChange object that exist in the MoneyChange array, the program will add and update the coin amount in the existing MoneyChange object. The program allows the user to keep creating the MoneyChange object until the user decided to stop by entering n (no) when ask by the program. All the 5 different tasks also work for the user input MoneyChange object. The program should be able to handle some error input such as invalid coin amount, non-numerical value and so on instead of crash the whole program.

For testing the program, there are 5 MoneyChange objects in total.

**The example of user’ inputs are as follow (not included the error input and yes/no continue option):**

Name: yk

Coin amount: 55

Name: aa

Coin amount: 185

Name: yk

Coin amount: 50

Name: cc

Coin amount: 75

Name: ii

Coin amount: 55

Name: ee

Coin amount: 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test id** | **Test description/justification – what is the test for and why this particular test** | **Actual data for this test** | **Expected output** | **Actual program output when test is carried out** | **Test run outcome – Pass/Fail** |
| 1 | According to the requirement, the program need to display the student information at the beginning. Once we run the program, the student information will display at the very beginning. | No data required | Student information | Student information displayed at the beginning of the program. | Pass |
| 2 | After the student info, program will prompt user to choose 0 (read user input) or 1 (hard coded input). If user entered value less than 0 or greater than 1, program will display error message and prompt user choice again. | 5 – (5 is out of the acceptable range, so that we can test for error message) | Error message and prompt for user input again | Successfully display error message and ask user to enter a new input again. | Pass |
| 3 | After the student info, program will prompt user to choose 0 (read user input) or 1 (hard coded input). When user entered 0, program will start asking user for MoneyChange object and coin amount. | 0 – (0 is acceptable option, program should start ask for name and coin amount) | Please enter the name of the person:  Please enter the coin value for the person: | Program successfully asking user for name and coin amount of MoneyChange object. | Pass |
| 4 | After read the name and coin amount from user, program will ask user to enter Y to continue and N to stop reading input. Both upper/lower case are acceptable. | y – (y means yes and program should keep asking details to create MoneyChange object) | Please enter the name of the person:  Please enter the coin value for the person | Program continue asking user to enter name and coin amount of MoneyChange object. | Pass |
| 5 | After read the name and coin amount from user, program will ask user to enter Y to continue and N to stop reading input. Both upper/lower case are acceptable. When user entered N, program stop reading input and display the objects have been created. | N – (n means no and tell the program to stop asking and creating object, **program should stop**) | Program display “You have stop reading object !!” and display the existing objects | Program successfully stop and display the expected output. All MoneyChange objects are created. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | When reading coin amount for MoneyChange object, the coinCheckValue method will check for valid coin amount entered, if user entered invalid coin amount, program will display error message and ask user to enter the coin amount again. | 44 – (44 is an invalid coin amount, program should alert user for the invalid coin amount and ask user to enter again) | The coin value must be between 5-200 and divisible by 5  Please enter the coin value again. | Program display the error message and prompt for a new coin amount because 44 is not a valid coin amount. | Pass |
| 7 | If user entered the same name MoneyChange objects, program will add the coin amount to the same name object instead of creating a new object. | Name: yk  Coin: 55  Name: yk  Coin: 50  - (same name object to test the program to add up the coin amount) | When displaying the MoneyChange object, it should be only one object with name yk and the coin amount is 105 | Program display the expected result:  Name: yk  CoinChangeAmount:105 | Pass |
| 8 | When user entered option 1 in menu stage, program should ask for a name of MoneyChange object from user and display the coin denomination for that MoneyChange object. This method should also work for user input MoneyChange object | Menu: 1  Name: yk  -(test for task 1 with an existing object name) | ===< Change >===  $1 : 1  5 cents : 1  ================ | Program display the coin denomination of MoneyChange object with a name yk. The coin amount is correct (105) because there is 2 same name entered during the reading stage. | Pass |
| 9 | Option 1 in menu stage will ask user for a name and display its coin denomination. When user entered a name that not exist in the MoneyChange, program will display “ Name not found!! ” to tell the user there is not match object. | Name: NoName  -(test task 1 with a name that no matching object) | Name not found!! | Program display the expected result to tell user that the name user has entered is not found. Because there is no object with a name call “NoName” | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | When user entered option 2 in the menu stage, program will find the smallest coin amount in the MoneyChange array and display its coin denomination. | Menu: 2 –  (Test for task 2 and program should display the coin denomination of object with the smallest coin amount. In this case, customer ee has the smallest coin amount – 5 cent) | Customer ee has the smallest coin amount  The smallest coin amount is : 5  ====<Change >======  5 cents : 1  =================== | Customer ee has the smallest coin amount – 5 cents in total. Program successfully get the correct coin amount and display the coin denomination | Pass |
| 11 | When user entered option 3 in the menu stage, program will find the largest coin amount in the MoneyChange array and display its coin denomination. | Menu: 3 –  (Test for task 3 and program should display coin denomination of largest coin amount. In this case, the largest coin amount is 185) | Customer aa has the largest coin amount  The largest coin amount is : 185  =====<Change >=====  $1 : 1  50 cents : 1  20 cents : 1  10 cents : 1  5 cents : 1  =================== | Yk has the largest coin amount. Program successfully display the coin denomination of the MoneyChange object with the largest coin amount. The output also fulfill the “give as much of the higher valued coins as possible” requirement. | Pass |
| 12 | When user entered option 4 in the menu stage, program will find and display the largest total coin denomination among all MoneyChange objects. | Menu: 4 –  (Test for task 4 and program should display the largest number of coin denomination. Scenario here I give 4 new user’s input to test for more than 1 coin denomination with largest number of coin denomination.  **4 objects in total –**  **2 coin amount is 55**  **2 coin amount is 10**  Result should have 3 coin denomination with the same number of largest coin denomination. | The largest number of coin denomination is:  50 cent  The total number of 50 cent is: 2  The largest number of coin denomination is:  10 cent  The total number of 10 cent is: 2  The largest number of coin denomination is:  5 cent  The total number of 5 cent is: 2 | MoneyChange objects included:  Name: aa  CoinChangeAmount:15  Name: bb  CoinChangeAmount:15  Name: cc  CoinChangeAmount:50  Name: dd  CoinChangeAmount:50  The successfully output display all 3 largest coin denomination. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | When user entered option 1-4 in the menu stage, program will keep running and display the menu after each task has completed (option 1-4). Program will stop execute when user enter option 5. | Menu: 5 –  (Test if the program will terminate when user enter 5) | Thank you, Bye!! | Program display the expected sentence and stop executing. | Pass |
| 14 | Menu stage allowed user choose between tasks 1 – 5. If user enter number with negative or greater than 5 values, program will display error message and ask user to enter the option again. This method should work for both hardcoded and user ‘s input MoneyChange object. | Menu: 8 –  (There is no task 8 and test if the program will display an error message to remind the user) | Invalid option! | Program display the expected output and display the menu for user to enter again. | Pass |
| 15 | Menu stage allowed user to choose between tasks 1 – 5. If user enter a non-numerical value, program will display error message and ask user to enter the option again. This method should work for both hardcoded and user ‘s input MoneyChange object. | Menu: b –  (b is not a numerical value and test is the program will display an error message to remind the user instead of crash the program.) | Use a numerical value! | Program display the expected output and display the menu for user to enter again. | Pass |

**Screen capture of program’s output – User’s input**

|  |  |
| --- | --- |
| Test id |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |

1. **Java source code**

* **MoneyChange class**

//Title : FT MUR T221 ICT167 C – Assignment 1 (MoneyChange class)

//Author : Tee Yee Kang

//Date : 26/Jun/2021

//File Name : FTC-34315323-Assignment 01

//Purpose :This program is to allowed all the users (customers) to enter and

// store their name and coin amount into the system and then

// perform some different tasks provided by the system.

// 2 input values required for the MoneyChange objects - String name of customer, int coin amount

// coin amount must be between 5-200 and divisible by 5

**import** java.util.Scanner;

/\*\*

\* **@author** TeeYeeKang yeekang88 @ gmail.com

\* **@version** 1.1

\*/

**public** **class** MoneyChange {

**static** Scanner *input* = **new** Scanner(System.***in***);

//instance variables

/\*\*

\* The String name

\*/

**private** String name;

/\*\*

\* The int coinChangeAmount

\*/

**private** **int** coinChangeAmount;

//default constructor

/\*\*

\* Default Constructor of MoneyChange class

\* <p>

\* This class is used to represent the default value of String name and int coinChangeAmount

\* </p>

\* Precondition - Nil <br>

\* Postcondition - A MoneyChange object is created with the default value of name and coinChangeAmount

\*/

**public** MoneyChange() {

name = "None";

coinChangeAmount = 0;

}

//constructor with 2 parameters

//created an object with a String name and int coinChangeAmount

//coinChangeAmount must be valid (true), else set to default value

/\*\*

\* Constructor of MoneyChange class

\* <p>

\* This class is used to represent a String with name and int with coinChangeAmount

\* </p>

\* Precondition - The value of name and coinChangeAmount must be a String and positive int value respectively and cannot be null.<br>

\* Postcondition - A MoneyChange object is created with the value of name and coinChangeAmount

\* **@param** name Name of MoneyChange object

\* **@param** coinChangeAmount Coin amount of MoneyChange object

\*/

**public** MoneyChange(String name, **int** coinChangeAmount) {

**this**.name = name;

**if**(*checkCoinValue*(coinChangeAmount)) {

**this**.coinChangeAmount = coinChangeAmount;

}

}

//Accessor method

/\*\*

\* This method is used to get the name of the MoneyChange object

\* <p>

\* Precondition: Name is initialised at the Constructor <br>

\* Postcondition: The name of the MoneyChange object is returned.

\* </p>

\* **@return** name Return the MoneyChange object name

\*/

**public** String getName() {

**return** name;

}

/\*\*

\* This method is used to set the name of the MoneyChange object

\* <p>

\* Precondition: Take in a name of MoneyChange object with String value and cannot be null <br>

\* Postcondition: The name of the MoneyChange object is set.

\* </p>

\* **@param** name Name of MoneyChange object

\*/

**public** **void** setName(String name) {

**this**.name = name;

}

/\*\*

\* This method is used to get the coinChangeAmount of the MoneyChange object

\* <p>

\* Precondition: coinChangeAmount is initialised at the Constructor <br>

\* Postcondition: The coinChangeAmount of the MoneyChange object is returned.

\* </p>

\* **@return** coinChangeAmount Return the MoneyChange object coin coin change amount

\*/

**public** **int** getCoinAmount() {

**return** coinChangeAmount;

}

/\*\*

\* This method is used to set the coinChangeAmount of the MoneyChange object

\* <p>

\* Precondition: Pass in coinChangeAmount of MoneyChange object with int value between 5-200 and divisible by 5 <br>

\* Postcondition: The coinChangeAmount of the MoneyChange object is set if it is valid coinChangeAmount else set to 0 (default value).

\* </p>

\* **@param** coinChangeAmount Coin amount of MoneyChange object

\*/

**public** **void** setCoinAmount(**int** coinChangeAmount) {

//check for valid coinChangeAmount - true

//if not valid - set to default value 0

**if**(*checkCoinValue*(coinChangeAmount)) {

**this**.coinChangeAmount = coinChangeAmount;

}

}

//check for valid coin amount - return true if valid else false

/\*\*

\* A method to check for valid coin amount of the MoneyChange object entered by user

\*\* <p>

\* Precondition - The passed in coin amount must be between 5 to 200 and divisible by 5. <br>

\* Postcondition - Boolean valid will become true if coin amount is valid and return the boolean valid.

\* </p>

\* **@param** amount with int data type

\* **@return** valid true if valid coin amount else false

\*/

**public** **static** **boolean** checkCoinValue(**int** amount) {

**boolean** valid = **false**;

**if**((amount >= 5) && (amount <= 200) && (amount % 5 == 0)) {

valid = **true**;

}

**return** valid;

}

//calculate coin change denomination for MoneyChange object

/\*\*

\* This method is used to find the coins change denomination of the MoneyChange object and store in an int array

\*\* <p>

\* Precondition - Nil <br>

\* Postcondition - After completed the calculation, return the changeToReturn array.

\* </p>

\* **@return** changeToReturn-array Return the number of each coin denomination

\*/

**public** **int**[] getChange(){

//create an array to store the coin denomination

**int**[] changeToReturn = **new** **int**[6];

//running ballance

**int** tempCoinAmount = coinChangeAmount;

changeToReturn[0] = 0;//no of $2

changeToReturn[1] = 0;//no of $1

changeToReturn[2] = 0;//no of 50 cents

changeToReturn[3] = 0;//no of 20 cents

changeToReturn[4] = 0;//no of 10 cents

changeToReturn[5] = 0;//no of 5 cents

**while**((**double**) tempCoinAmount >= 200) {

//balace minus 200

tempCoinAmount -= 200;

//coin denomination increase by 1 each time

changeToReturn[0] ++;

}

**while**((**double**) tempCoinAmount >= 100) {

tempCoinAmount -= 100;

changeToReturn[1] ++;

}

**while**((**double**) tempCoinAmount >= 50) {

tempCoinAmount -= 50;

changeToReturn[2] ++;

}

**while**((**double**) tempCoinAmount >= 20 ) {

tempCoinAmount -= 20;

changeToReturn[3] ++;

}

**while**((**double**) tempCoinAmount >= 10) {

tempCoinAmount -= 10;

changeToReturn[4] ++;

}

**while**((**double**) tempCoinAmount >= 5) {

tempCoinAmount -= 5;

changeToReturn[5] ++;

}

**return** changeToReturn;

}

//description/output of each coin denomination

/\*\*

\* This method is used to display the description/output of different coin denomination.

\*\* <p>

\* Precondition - Take in an array of int type and index with positive int value.<br>

\* Postcondition - Display the output of each coin denomination including the amount of each coin denomination

\* </p>

\* **@param** array An array with int type

\* **@param** index A number with int value

\*/

**public** **static** **void** coinDenominationDescription(**int**[] array, **int** index){

**switch**(index) {

**case** 0: System.***out***.println("$2 : " + array[0]);

**break**;

**case** 1: System.***out***.println("$1 : " + array[1]);

**break**;

**case** 2: System.***out***.println("50 cents : " + array[2]);

**break**;

**case** 3: System.***out***.println("20 cents : " + array[3]);

**break**;

**case** 4: System.***out***.println("10 cents : " + array[4]);

**break**;

**case** 5: System.***out***.println("5 cents : " + array[5]);

**break**;

}

}

//display final output of each coin change denomination

/\*\*

\* This method is used as a controller for methods getChange and coinDenominationDescription

\*\* <p>

\* Precondition - Passed in a MoneyChange object.<br>

\* Postcondition - All coin change denomination with a value greater than 0 will be displayed

\* </p>

\* **@param** ch MoneyChange object

\*/

**public** **static** **void** displayChange(MoneyChange ch) {

//declare array size

**int** [] change = **new** **int**[6];

//call getChange method and store in array change

change = ch.getChange();

System.***out***.println("========< Change >======== ");

**for**(**int** idx = 0; idx < change.length;idx++) {

//print the coin denomination with value only

**if**(change[idx] > 0) {

*coinDenominationDescription*(change, idx);

}

}

System.***out***.println("========================== ");

}

//add passed in coin amount to the existing coin amount

/\*\*

\* This method is used to add coin amount

\*\* <p>

\* Precondition - The coin amount must be positive int value and cannot be null.<br>

\* Postcondition - The balance increase by the coin amount passed in.

\* </p>

\* **@param** coinChangeAmountToAdd Coin amount with int value

\*/

**public** **void** addCoinChangeAmount(**int** coinChangeAmountToAdd) {

**this**.coinChangeAmount += coinChangeAmountToAdd;

}

//Display the output of MoneyChange object including name and coinChangeAmount

/\*\*

\* This method is to display the name and coin amount of MoneyChange object

\*\* <p>

\* Precondition - Nil<br>

\* Postcondition - Display output of MoneyChange object.

\* </p>

\*/

**public** **void** display() {

System.***out***.println("Name:"+name);

System.***out***.println("CoinChangeAmount:"+coinChangeAmount);

}

//testing purpose

**public** **static** **void** main(String[] args) {

//create MoneyChange object

MoneyChange change01 = **new** MoneyChange();

Scanner input = **new** Scanner(System.***in***);

**char** ch = 'Y';

**do** {

System.***out***.print("Please enter the name of the person: ");

change01.setName(input.next()); //set name

System.***out***.print("\nPlease enter the coin value for the person(range 5 to 200, multiple of 5):");

**int** coin = input.nextInt();//get coin input

//check for valid coin amount

**if**(!*checkCoinValue*(coin)) {

System.***out***.println("\nIncorrect coin value. Must be in the range 5 to 200, and multiple of 5.");

}

//set coin amount

change01.setCoinAmount(coin);

//ask user whether to continue or stop

System.***out***.println("\nDo you wish to continue (Y/N)");

**char** option = input.next().charAt(0);

ch = Character.*toUpperCase*(option);

**if**(ch == 'N') {

System.***out***.println("You quit the program! !");

}

}**while**(ch == 'Y');

}//end of main

}//end of MoneyChange class

* **Client class**

//Title : FT MUR T221 ICT167 C – Assignment 1 (Client class)

//Author : Tee Yee Kang

//Date : 26/Jun/2021

//File Name : FTC-34315323-Assignment 01

//Purpose :This program is to allowed all the users (customers) to enter and

// store their name and coin amount into the system and then

// perform some different tasks provided by the system.

// 2 input values required for the MoneyChange objects - String name of customer, int coin amount

// coin amount must be between 5-200 and divisible by 5

**import** java.util.\*;

/\*\*

\* **@author** TeeYeeKang yeekang88 @ gmail.com

\* **@version** 1.1

\*/

**public** **class** Client {

**static** Scanner *input* = **new** Scanner (System.***in***);

**public** **static** **void** main(String[] args) {

//display student information

*studentInfo*();

//ask for user's choice

//hand code or hard code

System.***out***.println("Read user input - 0");

System.***out***.println("Hard code input - 1");

System.***out***.print("Enter your choice : ");

**int** choice = *input*.nextInt();

//loop until user entered a valid choice

**while**((choice <0) || (choice >1)) {

System.***out***.println("Invalid choice !! Enter choice again");

choice = *input*.nextInt();

}

//declare MoneyChange array size

MoneyChange[] changeArrayList = **new** MoneyChange[50];

**if**(choice == 0) {

//hand code

*readValue*(changeArrayList);

}**else** {

/\*\*

\* Hard coded value for MoneyChange class

\* <p>

\*/

changeArrayList = **new** MoneyChange[9];

//create and store objects into MoneyChange array

MoneyChange change01 = **new** MoneyChange("A", 100);

changeArrayList[0] = change01;

MoneyChange change02 = **new** MoneyChange("B", 55);

changeArrayList[1] = change02;

MoneyChange change03 = **new** MoneyChange("C", 45);

changeArrayList[2] = change03;

MoneyChange change04 = **new** MoneyChange("D", 55);

changeArrayList[3] = change04;

MoneyChange change05 = **new** MoneyChange("E", 15);

changeArrayList[4] = change05;

MoneyChange change06 = **new** MoneyChange("F", 25);

changeArrayList[5] = change06;

MoneyChange change07 = **new** MoneyChange("G", 10);

changeArrayList[6] = change07;

MoneyChange change08 = **new** MoneyChange("I", 60);

changeArrayList[7] = change08;

MoneyChange change09 = **new** MoneyChange("J", 95);

changeArrayList[8] = change09;

}

//display all elements in array (object)

*printAllMoneyChangeObject*(changeArrayList);

//menu

*process*(changeArrayList);

}//end of main class

//student info method

/\*\*

\* This method is used display the student information

\*\* <p>

\* Precondition - Nil.<br>

\* Postcondition - Display student information.

\* </p>

\*/

**public** **static** **void** studentInfo() {

System.***out***.println("Name : Tee Yee kang" );

System.***out***.println("StudentNo : 34315323" );

System.***out***.println("Mode Of Enrolment : Full Time" );

System.***out***.println("Tutor : Aaron Yeo" );

System.***out***.println("Tutorial Attendence Day : Friday" );

System.***out***.println("Tutorial Attendence Time : 1030-1230" );

System.***out***.println("----------------------------------------------\n" );

}//end of method

//print all MoneyChange objects by calling display() method

/\*\*

\* This method is used to display all MoneyChange objects inside MoneyChange array

\*\* <p>

\* Precondition - Pass in a MoneyChange array.<br>

\* Postcondition - Display the name and coin amount of all MoneyChange object.

\* </p>

\* **@param** array An array of type MoneyChange

\*/

**private** **static** **void** printAllMoneyChangeObject(MoneyChange[] array) {

System.***out***.println("\n----------All MoneyChange Objects----------");

//use for loop to display all objects

**for**(**int** idx=0;idx<array.length;idx++) {

MoneyChange currentChange = array[idx];

**if**(currentChange!=**null**) {

currentChange.display();

}

}

}//end of method

//find for same name MoneyChange object and add passed in coin amount

//OR create a new MoneyChange object by calling the addNewChangeObject() method

/\*\*

\* This method is used to add coin amount to existing MoneyChange object or create a new MoneyChange, no null values are allowed.

\*\* <p>

\* Precondition - Pass in a MoneyChange array, String name and a coin amount with positive int value.<br>

\* Postcondition - Calling addNewChangeObject method to created new object or updated the coin amount of existing MoneyChange object.

\* </p>

\* **@param** changeArray An array of type MoneyChange

\* **@param** nameToAdd Name of String type and to be added into the MoneyChange array

\* **@param** amountToAdd Coin amount with int value and to be added into the MoneyChange array

\*/

**public** **static** **void** addNewChange(MoneyChange[] changeArray, String nameToAdd,**int** amountToAdd) {

**boolean** found = **false**;

**for**(**int** i=0;i<changeArray.length;i++) {

MoneyChange currentChange = changeArray[i];

**if**(currentChange!=**null**) {

//get the existing name of MoneyChange object

String currentName = currentChange.getName();

//compare is name to add is same as existing name

**if**(currentName.equals(nameToAdd)) {

currentChange.addCoinChangeAmount(amountToAdd); //add the coin amount is same name

found = **true**;

}

}

}

**if**(found) {

System.***out***.println("Amount added to the name");

}**else** {

//if not same name found, create a new MoneyChange object

MoneyChange newChangeObject = **new** MoneyChange(nameToAdd, amountToAdd);

*addNewMoneyChangeObject*(changeArray,newChangeObject);

}

}//end of method

//create and store new MoneyChange object into available array slot

/\*\*

\* A private method to create a new MoneyChange array and store it into the MoneyChange array.

\*\* <p>

\* Precondition - Pass in a MoneyChange array and a MoneyChange object.<br>

\* Postcondition - For loop will search through the array and find for an empty slot to store new MomeyChange object.<br>

\* After stored, found become true.

\* </p>

\* **@param** changeArray An array of type MoneyChange

\* **@param** nameToAdd Name of String type and to be added into the MoneyChange array

\* **@param** amountToAdd Coin amount with int value and to be added into the MoneyChange array

\*/

**private** **static** **void** addNewMoneyChangeObject(MoneyChange[] changeArray, MoneyChange change) {

**boolean** found = **false**;

//look for a null slot and add the new change object in

**for**(**int** i=0;i<changeArray.length;i++) {

MoneyChange currentChange = changeArray[i];

**if**(currentChange==**null**) {

changeArray[i] = change;

found = **true**;

**break**;

}

}

**if**(found) {

System.***out***.println("New change object created");

}**else** {

System.***out***.println("No empty slots found");

}

}//end of method

//read values of MoneyChange object from user

//keep asking and stop by user's option (Y/N)

/\*\*

\* This method is used to read values from user in order to create MoneyChange object

\*\* <p>

\* Precondition - Pass in an array of type of MoneyChange.<br>

\* Postcondition - All existing or newly created MoneyChange objects will be store into the MoneyChange array.<br>

\* </p>

\* **@param** array An array of type MoneyChange

\*/

**public** **static** **void** readValue(MoneyChange[] array) {

System.***out***.println("\nEnter name and coin amount to create at least 9 MoneyChange object ");

**for**(**int** idx = 0; idx < array.length; idx ++) {

//prompt for name

System.***out***.print("\nPlease enter the name of the person: ");

String name = *input*.next();

//prompt for coin amount

System.***out***.print("Please enter the coin value for the person:");

**int** coin = *input*.nextInt();

//check for valid coin amount

//prompt for coin amount again if not valid

**while**(!MoneyChange.*checkCoinValue*(coin)) {

System.***out***.println("\nThe coin value must be between 5-200 and divisible by 5");

System.***out***.print("Please enter the coin value again: ");

coin = *input*.nextInt();

}

//create object

*addNewChange*(array, name, coin);

//ask user option to continue or stop executing

System.***out***.print("\nDo you want to continue ?? Y/N : ");

**char** opt = *input*.next().charAt(0);

opt = Character.*toUpperCase*(opt);

**if** (opt == 'N'){

System.***out***.println("You have stop reading object !!");

**break**;

}

}//end of for loop

}//end of method

//option 1

//Enter a name and display change to be given for each denomination

//prompt user enter a name to display

/\*\*

\* This method is for option 1 - Ask user for the name to display coin denominaiton.

\*\* <p>

\* Precondition - Nil<br>

\* Postcondition - Return the name enter by user.

\* </p>

\* **@return** name Return the name with String type entered by user

\*/

**public** **static** String getNameToDisplay() {

String name = "";

System.***out***.print("\nEnter a name to display his/her coin amount: ");

name = *input*.next();

**return** name;

}//end of method

//find the index of name entered by user

/\*\*

\* This method is for option 1 - Find the index of name entered by user previously.

\*\* <p>

\* Precondition - Pass in a MoneyChange array<br>

\* Postcondition - Search for same name and return the index of the name. If no same found return the initial index value.

\* </p>

\* **@param** array An array with type MoneyChange

\* **@return** index Return the index of name entered by user or -1 is no same name found

\*/

**public** **static** **int** getIndexOfName(MoneyChange[] array) {

**int** index = -1;

//get the name entered by user

String name01 = *getNameToDisplay*();

//find for same name

**for**(**int** idx = 0; idx < array.length; idx++) {

**if**(array[idx] != **null**) {

**if**(name01.equalsIgnoreCase(array[idx].getName())) { //ignore case

index = idx;

**break**;

}

}

}

**return** index;

}//end of method

//option 1 - display the coin change denomination for particular name

/\*\*

\* This method is the controller of getNameToDisplay and getIndexOfName methods in order to find the coin denomination of particular name.

\*\* <p>

\* Precondition - Pass in a MoneyChange array<br>

\* Postcondition - Display the coin denomination of particular name of MoneyChange object

\* </p>

\* **@param** array An array of type MoneyChange.

\*/

**public** **static** **void** option1(MoneyChange[] array) {

//get index of name

**int** idx = *getIndexOfName*(array);

**if**(idx < 0) {

System.***out***.println("\nName not found!!");

}**else** {

MoneyChange.*displayChange*(array[idx]);

}

}//end of method

//bubble sort array - ascending order

/\*\*

\* This method is to sort the passed in array in ascending order by using bubbleSort method.

\*\* <p>

\* Precondition - Pass in a MoneyChange array<br>

\* Postcondition - The passed in array has been sorted in ascending order.

\* </p>

\* **@param** array An array of type MoneyChange.

\*/

**public** **static** **void** bubbleSort(MoneyChange[] array){

**boolean** swap = **true**;

**while** (swap) {

swap = **false**;

**for** (**int** i = 1; i < array.length; i++) {

**if**(array[i] !=**null**) {

**if** (array[i - 1].getCoinAmount() > array[i].getCoinAmount()) {

// swap the numbers

MoneyChange temp = array[i - 1];

array[i - 1] = array[i];

array[i] = temp;

swap = **true**;

}

}

}

}

}//end of method

//option 2 - find smallest coin amount and display its coin denomination

/\*\*

\* This method is for option 2 - display the smallest coin amount among the MoneyChange objects.

\*\* <p>

\* Precondition - Pass in a MoneyChange array<br>

\* Postcondition - Display the coin denomination with smallest coin amount.

\* </p>

\* **@param** array An array of type MoneyChange.

\*/

**public** **static** **void** findSmallest(MoneyChange[] array) {

//sort array in ascending order first

*bubbleSort*(array);

//after sorted, index 0 will be the smallest

System.***out***.println("\nCustomer " + array[0].getName() + " has the smallest coin amount");

System.***out***.println("The smallest coin amount is : " + array[0].getCoinAmount());

//display coin denomination

MoneyChange.*displayChange*(array[0]);

}//end of method

//option 3 - find largest coin amount and display its coin denomination

/\*\*

\* This method is for option 3 - find and display the largest coin amount among the MoneyChange objects.

\*\* <p>

\* Precondition - Pass in a MoneyChange array<br>

\* Postcondition - Find and display the coin denomination with largest coin amount.

\* </p>

\* **@param** array An array of type MoneyChange.

\*/

**public** **static** **void** findLargest(MoneyChange[] array) {

//sort array in ascending order

*bubbleSort*(array);

//set largest to 0 first

**int** maxIndex = 0;

//search for largest

**for**(**int** idx = 1; idx<array.length;idx++) {

**if**(array[idx]!=**null**) {

**if**(array[maxIndex].getCoinAmount() < array[idx].getCoinAmount()) {

maxIndex = idx;

}

}

}

//display output

System.***out***.println("\nCustomer " + array[maxIndex].getName() + " has the largest coin amount");

System.***out***.println("The largest coin amount is : " + array[maxIndex].getCoinAmount());

MoneyChange.*displayChange*(array[maxIndex]);

}//end of method

//option 4 - find largest coin denomination

/\*\*

\* This method is to find and display the largest coin denomination

\*\* <p>

\* Precondition - Pass in MoneyChange array<br>

\* Postcondition - Calculate all total of all coin denomination and display the largest coin denomination

\* </p>

\* **@param** array An array with type MoneyChange

\*/

**public** **static** **void** getMaxDenomination(MoneyChange[] array) {

//declare array size

//change is to store getChange()

//total is to store the running total coin denomination

**int**[] change = **new** **int**[6];

**int**[] total = **new** **int**[6];

**for**(**int** outer=0; outer<array.length; outer++) {

**if**(array[outer] != **null**) {

change = array[outer].getChange();

//store running total coin denomination

**for**(**int** inner=0; inner < 6; inner++) {

total[inner]+=change[inner];

}

}

}

//find largest

//set max to array index 0 first

**int** maxIndex = 0;

//find index of largest coin denomination

**for**(**int** idx=1; idx<total.length;idx++) {

**if**(total[maxIndex] < total[idx]) {

maxIndex = idx;

}

}

//display output for largest coin denomination

**for**(**int** idx=0;idx<total.length;idx++) {

//if there are more than 1 largest coin denomination

**if**(total[maxIndex] == total[idx]) {

maxIndex = idx;

//display output

*largestCoinOutput*(total, maxIndex);

}

}

}//end of method

//output for each largest coin denomination

/\*\*

\* This method is for the output/description for largest coin denomination

\*\* <p>

\* Precondition - Pass in an int type array and the index (int type) of largest coin denomination array<br>

\* Postcondition - Print out particular output of largest coin denomination base on passed in arguments

\* </p>

\* **@param** total An array of int type

\* **@param** maxIndex The index (int type) of largest coin denomination array

\*/

**public** **static** **void** largestCoinOutput(**int**[] total, **int** maxIndex) {

//display output

//switch the index-maxIndex

System.***out***.println("\nThe largest number of coin denomination is:");

**switch**(maxIndex) {

**case** 0: System.***out***.println("2 dollar");

System.***out***.println("The total number of 2 dollar is: " + total[0]);

**break**;

**case** 1: System.***out***.println("1 dollar");

System.***out***.println("The total number of 1 dollar is: " + total[1]);

**break**;

**case** 2: System.***out***.println("50 cent");

System.***out***.println("The total number of 50 cent is: " + total[2]);

**break**;

**case** 3: System.***out***.println("20 cent");

System.***out***.println("The total number of 20 cent is: " + total[3]);

**break**;

**case** 4: System.***out***.println("10 cent");

System.***out***.println("The total number of 10 cent is: " + total[4]);

**break**;

**case** 5: System.***out***.println("5 cent");

System.***out***.println("The total number of 5 cent is: " + total[5]);

**break**;

}

}//end of method

//display menu and prompt for user choice

/\*\*

\* This method is to display the menu and ask for user's choice.

\*\* <p>

\* Precondition - Nil<br>

\* Postcondition - Prompt user to enter a choice and return user's choice.

\* </p>

\* **@return** Return user's choice

\* **@throws** Exception Choice must be numeric value and between 1-5

\*/

**public** **static** **int** menu() **throws** Exception{

System.***out***.println("\n---------< Menu ----------");

System.***out***.println("[1] - Enter a name and display change to be given for each denomination");

System.***out***.println("[2] - Find the name with the smallest amount and display change to be \n given for each denomination");

System.***out***.println("[3] - Find the name with the largest amount and display change to be \n given for each denomination");

System.***out***.println("[4] - Calculate and display the largest number of coin denomination, \n and the total number of the coin");

System.***out***.println("[5] - Exit program! ");

System.***out***.print("\nEnter your choice : ");

**int** choice = Integer.*parseInt*(*input*.next());

**return** choice;

}//end of method

//Main controller that call all 4 options/tasks base on user's choice

/\*\*

\* This method is the main controller which call and execute method/task according to user's choice

\*\* <p>

\* Precondition - Pass in MoneyChange array<br>

\* Postcondition - Perform the task/method selected by user.

\* </p>

\* **@param** array An array with type MoneyChange

\*/

**public** **static** **void** process(MoneyChange[] array){

**int** option = 0;

**do**

{

**try**{

option = *menu*();

**switch**(option)

{

**case** 1 : *option1*(array);

**break**;

**case** 2 : *findSmallest*(array);

**break**;

**case** 3 : *findLargest*(array);

**break**;

**case** 4 : *getMaxDenomination*(array);

**break**;

**case** 5 : System.***out***.println("\nThank you Bye!!");

**break**;

**default** : System.***out***.println("\nInvalid option!");

}

}**catch** (Exception e) //if user enter non-numerical value

{

System.***out***.println("\nUse a numerical value !");

}

}**while** (option != 5);

}//end of method

}//end of client class