

**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 | Lim | Given name | Wen Chao |
| Student Number | CT0360379/34368872 | Email | CT0360379@kaplan.edu.sg |

**Assignment details**

|  |  |  |  |
| --- | --- | --- | --- |
| Unit name | Principle of Computer Science | Unit Code | ICT 167 |
| Unit Coordinator | Dr Hone Xie | Tutor/Tutorial time | Chong Siew Cheong |
| Due date/time | 22 Oct 2021 | Submission date | 18 Oct 2021 |
| Assignment title | ICT 167 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.

# 1. Title

Title: Coin Optimizer

Author: Lim Wen Chao

Date: 23/9/2021

File names: Client.java, Change.java, CoinDenoms.java

This is the documentation for Coin Optimizer, a program that helps to calculate how many coins of each type to give given a specific amount of money.

# 2. Requirements/Specification

Coin Optimizer is a program that will help users figure out how much change in each coin type to give to a person. It can calculate for multiple persons and know which person has the largest or smallest amount, the total number of coins for each denomination, and the total number of coins needed.

## Program Requirements/scope:

* There should be at least two classes, Change & Client
* Change class should have at least two instance variables, person name & coin amount
* All instance variables above should be private
* Client class should not need to change when using different coin denominations
* Change class should have at least one default constructer and a constructer with the person name and coin amount as parameter
* Change class should not have direct input and output function
* Change class should have getter and setter methods
* Client class should read input from the user
* Client class should store input from users using the Change class
* The program should validate that coin is divisible by 5
* To stop the program from asking for more person, ask the user if they have more to enter after each person-coin set
* If a person name is the same for multiple person-coin set input by the user, then add them together
* The program should prioritise giving the highest coin denomination
* Coin denominations with 0 are not displayed
* The program has five functions available to the user once person data input from the user ends.
  + Function 1 will ask the user to enter a name and display the coin changes for that person
  + Function 2 will display the name with the least amount of money and the corresponding coin changes
  + Function 2 will display the name with the most amount of money and the corresponding coin changes
  + Function 4 will display the total number of coins for each denomination and the sum of the total
  + Function 5 will quit the program

## Assumptions

* Person name is a one-word string
* The coin value is an integer value
* The coin value is in cents only
* Users will not want to quit when entering person name and coin value
* Users can only quit the program after entering all person name and coin value data
* The system is only for Singapore and Australia

# 3. User Guide

## Prerequisites

Ensure that JDK 11 are installed. To install JDK in Visual Studio Code. Open the Command palette (Ctrl+Shift+P) then search for Java: Configure Java Runtime. Install JDK 11 and restart Visual Studio Code.

Graphical user interface, text

Description automatically generated

Fig 3.1

Ensure that the extension, Extension Pack for Java is installed. To install, go to extension(Ctrl+ Shift+X) and search for ‘Extension Pack for Java’

Graphical user interface, text

Description automatically generated

fig 3.2

1. Right-click the source code folder and open it with Visual Studio Code.

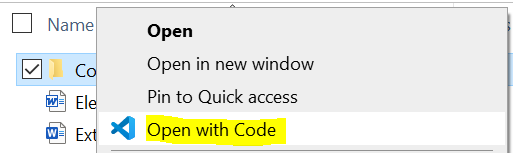


Fig 3.3

1. Click on the Run menu and click on Start Debugging to start the program.

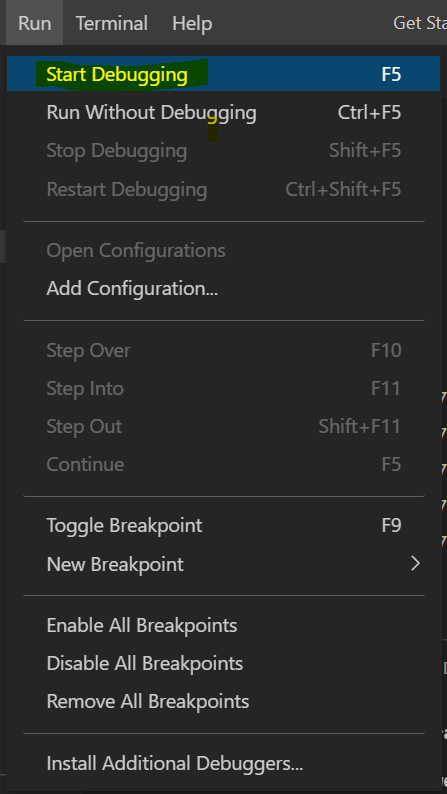


Fig 3.4

1. Select Java

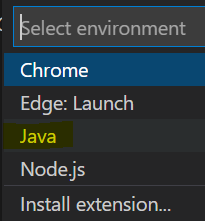


Fig 3.5

1. You will be greeted with the message below. Enter Y to use test data. Enter N to enter custom data. If Y, go to step 9. If N continues to step 5.

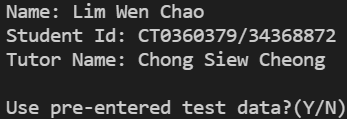


Fig 3.6

1. If selected N, the message below will be displayed. Select and enter the code beside the country name.

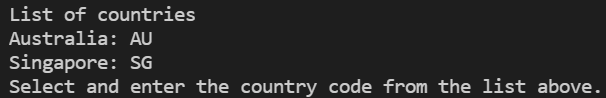


Fig 3.7

1. Next, enter the name of the person.



Fig 3.8

1. Next, enter the amount of money the person has.



Fig 3.9

1. Next, the system will prompt you if there is no more person data to enter. If Y, return to step 6.



Fig 3.10

1. The functions menu is displayed. Enter the corresponding number to select and use a function.

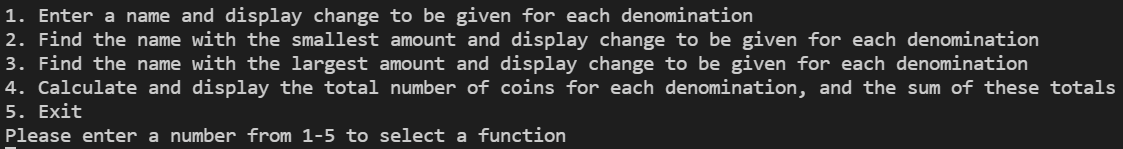


Fig 3.11

1. If you selected function 1, you would need to enter a name to search. For functions 2,3, and 4, no other input will be requested. Function 5 will quit the program.



Fig 3.12

# 4. Structure/Design/Algorithm

Extra instance variable coinChange & coinDenom are included in the Change class. The coinChange variable is used to store the coin change in each denomination for that Change object. The coinDenom variable is used to store the coin denomination in use.

The calCoinDenom method is used to calculate the coin change per denomination for that object.

The searchPersonName method will search the Change objects and return coinChange for the given person name.

The searchLeastCoin method will search the Change objects and return coinChange for the person with the smallest coinAmt value.

The searchMostCoin method will search the Change objects and return coinChange for the person with the largest coinAmt value.

The coinNumPerDenom method will add the number of coins in each denomination for all created objects and return the result in an array.

The sumCoinNum method will add the result from coinNumPerDenom to obtain the total coins needed and return the number.

## UML class diagram

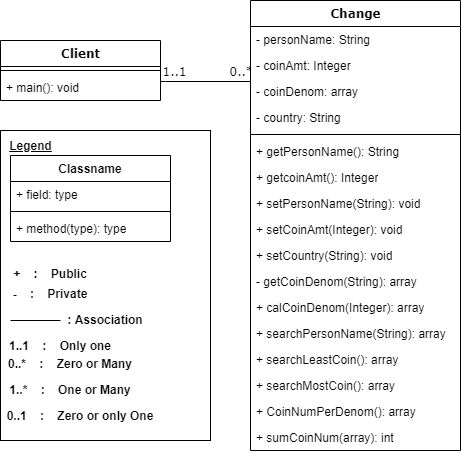


Fig 4.1

## Structure chart

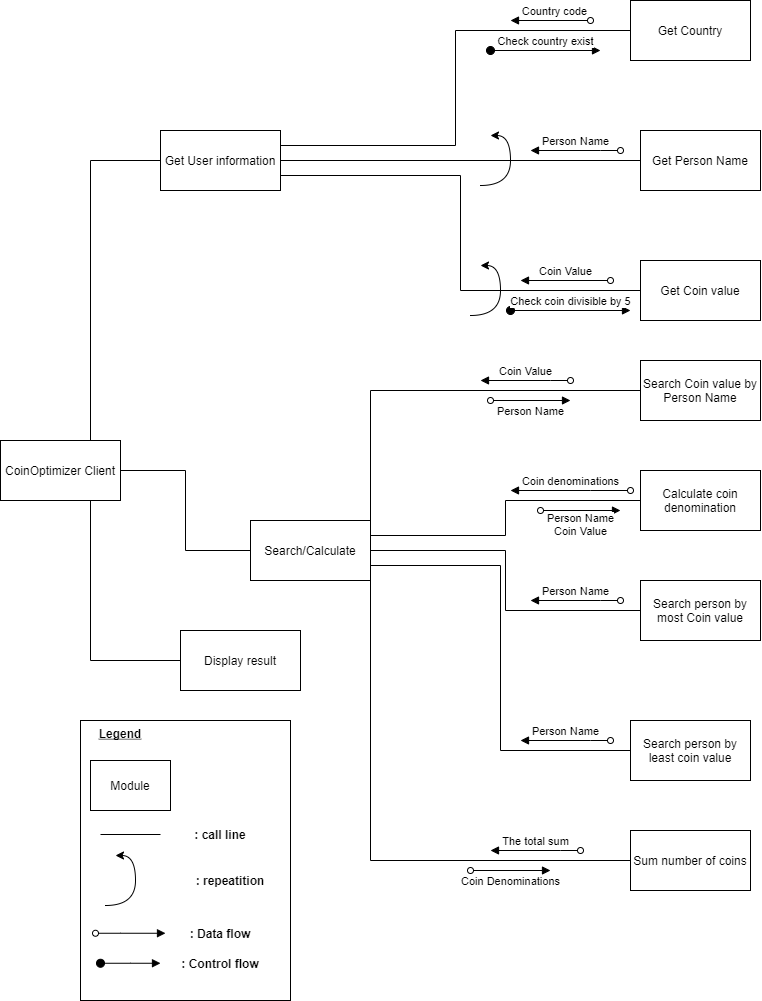


Fig 4.2

## Pseudocodes

High-level pseudocode:

1. Ask the user for the country to determine coin denomination (Singapore & Australia only)
2. Ask the user for person name and coin value inputs (Same person multiple times will add value instead)
3. Ask user if there are more person to input
   1. If yes, repeat steps 1 & 2
   2. If no, then continue to step 3
4. Ask the user which of the five functions to run
   1. Option 1 (Display change for one person)
      1. Ask the user for a person name
         1. If person name exist
            1. Display the coin change in each denomination
         2. If the person name does not exist
            1. Tell the user that the person does not exist
   2. Option 2 (Display change for the person with the least money)
      1. Display person name for the person with least money
      2. Display the change for each denomination for the person with the least money
   3. Option 3 (Display change for the person with the most money)
      1. Display person name for the person with the most money
      2. Display the change for each denomination for the person with the most money
   4. Option 4 (Display total coins needed for each denomination and the total number of coins)
      1. Display total coins needed for all person for each denomination
      2. Display total coins needed for all person for all denomination
   5. Option 5 (Exit)
      1. End Program

Low-level pseudocode:



# 5. O-O design concepts

## Information hiding/Encapsulation

Information hiding/encapsulation is putting codes in its method and using the method to perform a task, but the programmer using the method does not need to know how the method performs the task.

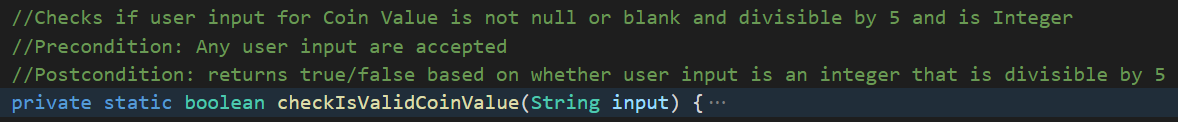


Fig 5.1

For example, the checkIsValidCoinValue method will check if the given input from the user is a valid coin value amount. The user of this method will not know how it does that to use this method. All the users of this method need to know is to pass user input to check validity.

## Pre-conditions and Post-conditions

Pre-condition is the conditions that are required to be fulfilled before the method can be used. Post-conditions are what the method will do once used.

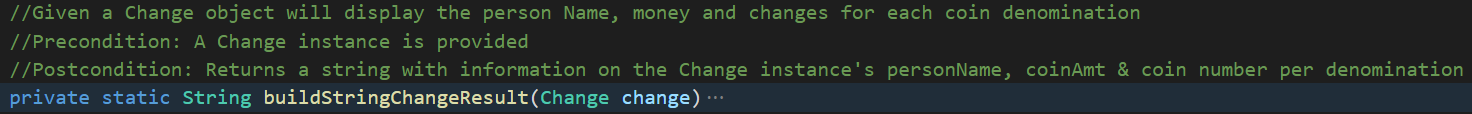


Fig 5.2

For example, the buildStringChangeResult’s pre-condition requires a Change instance to be provided. The post-condition is that it will parse and return the result string that will show the person name, coin amount, and the coin change for each denomination.

## Constructors

Constructors are a method used to create a new object.

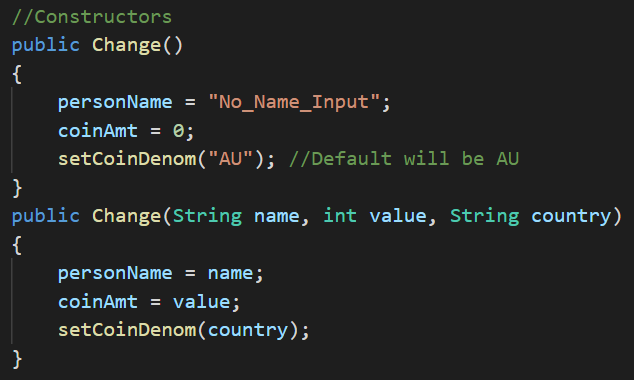


Fig 5.3

Invoking a new Change() will create a new Change object.

## Assessors and Mutators(Getter/Setter)

Assessors are the method that allows you to get the value of a private instance variable.

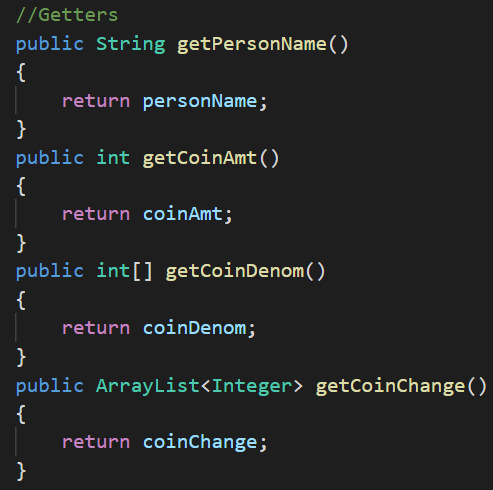


Fig 5.4

Invoking any of the assessor methods above will return the value.

Mutators are methods that allow you to change the value of a private instance variable

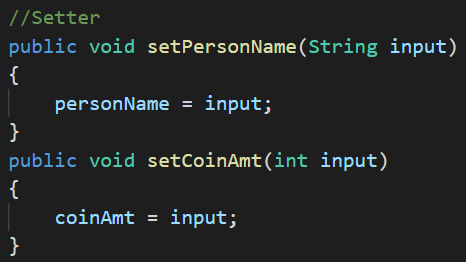


Fig 5.5

Invoking any of the mutator methods above will set the corresponding variable to that value.

## Helper

The helper method is a method that does something and can be repeatedly used.

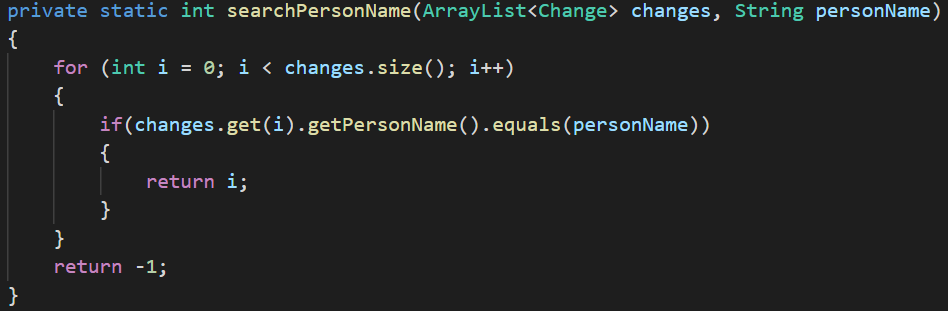


Fig 5.6

The seachPersonName method can search all Change objects for the Change object with the same name.

This method is used to check if the person exists while the user is entering person information and for function 1 of CoinOptimizer.

## Modularisation

Modularisation is the principle of writing code in small independent chunks. Those chunks can be reused throughout the program and can be invoked as long as parameters are provided.

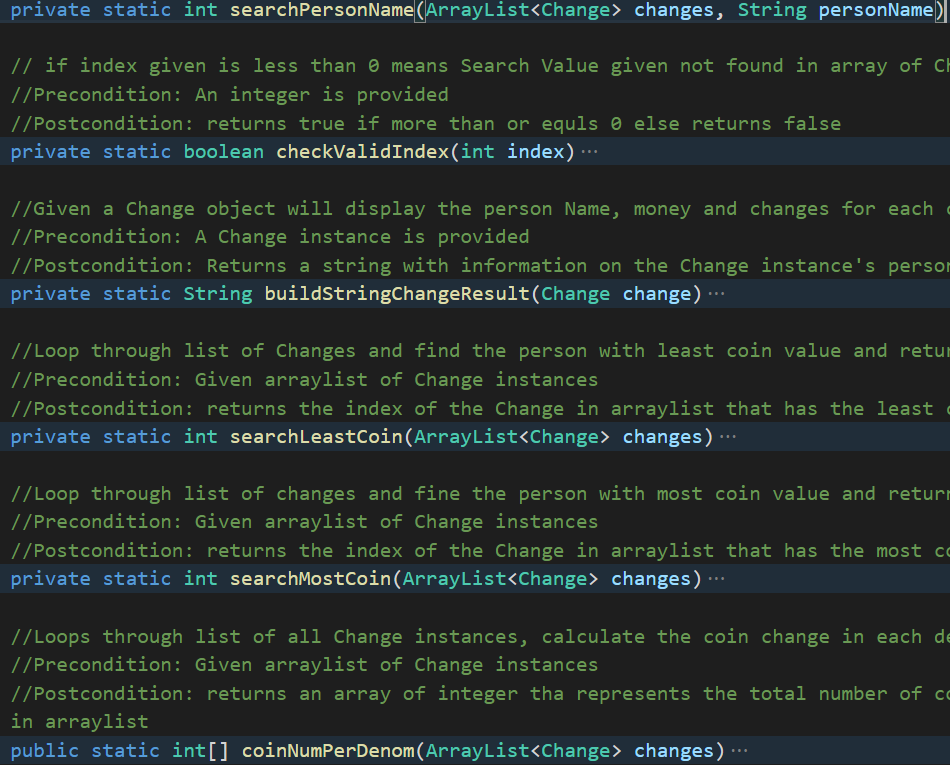


Fig 5.7

For example, the searchLastCoin and buildStringChangeResult methods can obtain the result required for function 2 of CoinOptimizer. At the same time, searchMostCoin and buildStringChangeResult methods could get the result needed for function 3 of CoinOptimizer.

Both functions used the buildStringChangeResult.

# 6. Limitations

There is no program shortfall based on the scope of the program.

# 7. Testing

The testing will test both valid and invalid scenarios for each input (Person Name, Coin Amount), then it will test each function of the program.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Id | Test Description | Test Data | Expected Output | Actual Output | Test result |
| 01 | Valid CountryCode are accepted | “AU” | Country selected: Australia |  | Pass |
| 02 | Invalid CountryCodes are not accepted and will be request again | “testCode” | Please select and enter a valid country code from the list. |  | Pass |
| 03 | Change with valid name & coin change amount are accepted and change able to be retrieved & displayed correctly using function 1 | Name: “Test1” CoinAmt: 385 | Customer:  Test1 385 cents  Change:  200 cents: 1  100 cents: 1  50 cents: 1  20 cents: 1  10 cents: 1  5 cents: 1 |  | Pass |
| 04 | Empty/Blank names are not accepted when entering user info | Name: “” | Person name cannot be blank |  | Pass |
| 05 | Change with negative coin amount are not accepted | CoinAmt: -15 | Coin value must be positive and in multiples of 5 |  | Pass |
| 06 | Change with coin amount that is not divisible by five are not accepted | CoinAmt: 11 | Coin value must be positive and in multiples of 5 |  | Pass |
| 07 | CoinOptimizer can accept multiple Change information and be correctly retrieved using function 1 | Name: “Test1”, ”Test2”, “Test3”  CoinAmt: “385”, “145”, “25” | Customer:  Test1 385 cents  Change:  200 cents: 1  100 cents: 1  50 cents: 1  20 cents: 1  10 cents: 1  5 cents: 1  Customer:  Test2 145 cents  Change:  100 cents: 1  20 cents: 2  5 cents: 1  Customer:  Test3 25 cents  Change:  20 cents: 1  5 cents: 1 |  | Pass |
| 08 | Entering the same Person name again will add the coin amount entered to the existing amount | Name: “Test1”  , “Test1”  CoinAmt: “385”, “100” | Customer:  Test1 485 cents  Change:  200 cents: 2  50 cents: 1  20 cents: 1  10 cents: 1  5 cents: 1 |  | Pass |
| 09 | Search for non-existing person using function 1 | Name: <Any name not entered> | Person not found. |  | Pass |
| 10 | System searches and displays the person’s change with least money correctly with function 2 | Name: “Test1”, ”Test2”, “Test3”  CoinAmt: “385”, “145”, “25” | Customer:  Test3 25 cents  Change:  20 cents: 1  5 cents: 1 |  | Pass |
| 11 | System searches and displays the person’s change with most money correctly with function 3 | Name: “Test1”, ”Test2”, “Test3”  CoinAmt: “385”, “145”, “25” | Customer:  Test1 385 cents  Change:  200 cents: 1  100 cents: 1  50 cents: 1  20 cents: 1  10 cents: 1  5 cents: 1 |  | Pass |
| 12 | CoinOptimizer calculates and displays total coins change number per denomination & total coin needed for all denominations correctly using function 4 | Name: “Test1”, ”Test2”, “Test3”  CoinAmt: “385”, “145”, “25” | Change: 200 cents: 1  100 cents: 2  50 cents: 1  20 cents: 4  10 cents: 1  5 cents: 3  Total Coins: 12 |  | Pass |
| 13 | CoinOptimizer will quit if the user selects function 5 | “5” | CoinOptimizer stops and quit | Program quits | Pass |