Name: Oliver Legg
Student ID: 201244658

COMP122 - Assessment 1

Information

Name:	Oliver Legg
Email:	O.Legg@student.liverpool.ac.uk
Student ID:	201244658

Requirements

Part 1

a) Extend the `ThreeDice.java` class to a subclass called `ThreeDiceScorer.java`. This subclass will return the dice values plus the number of points earnt for a particular dice roll.

b) Create a new file called `Game.java`. In `Game.java`, take an input for the number of rounds to be played – store this into an array of objects `ThreeDiceScorer`'s. Use data validation so the user can't enter anything invalid (no negative rounds). Store the dice rolls for 2 players in an array of objects.

c) Create a program called `Average.java`. This should calculate the mean of all the possible rolls of 3 fair 6-sided die.

For example:

- 1 1 1 = 1 + 1 + 1 + (60) = 63
- 1 1 2 = 1 + 1 + 2 + (20) = 24
- 1 1 3 = 1 + 1 + 3 + (20) = 25
- …
- 6 6 6 = 6 + 6 + 6 + (60) = 78

This is because 1+1+1 is the dice rolls (sums up to 3). The 60 points are made because 1,1,1 are 3 the same. Adding the sum of the die and the points for the dice roll together, you get a total of 63.

On top of this, I would have to use the ThreeDiceScorer class to get the points for calculating the average. Instantiating it would look like so:

```
ThreeDiceScorer points = new ThreeDiceScorer(x, y, z);
```

d) Slightly alter part c requirement to output the results of the average with 2 6-sided die and 1 6-sided dice being the numbers `{2,3,4,5,6,6}`. The six sides of the 3 dice will occur with equal probability.

The following information should be outputted each round:

- Dice values and number of points earnt for the dice values;

- The winner of the round (with the highest score) or a draw (if both players have equal score);

After all the rounds have been shown, this should be calculated and displayed:

- Total number of rounds won for each player;
- Total points for each player;
- Average points for both players across each round;
- Player who scored the highest total points (the winner);

Part 2

a) Design the program so that the user must input their:
   - (Integer) Phone minutes;
   - (Integer) Number of evening/weekend minutes;
   - (Integer) Broadband usage (Mb) per month;

   These values should all be greater than or equal to zero.

b) Create 3 accounts:
   - Bronze Account;
   - Silver Account;
   - Gold Account;

   These accounts must print their information and calculated information including:
   - Total cost for (extra) broadband usage;
   - Total cost of that account (call costs + broadband costs + plus the package cost)

c) Call your application program `AccountUser.java`. Afterwards, output which account has the total cost that is the cheapest. If the cheapest 2 or 3 are equally the same, then always recommend the one with the most features as you will get your money's worth.

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Analysis and Design

Part 1

For part 1 of this programming exercise, it is my job to create a game between two players where both players roll a dice for a given number of rounds as inputted by the user. Each dice throw is worth a number of points as shown in the following table below:

Description	Points Calculation	Example	Points
Three the same	sumOfDice + 60	3, 3, 3 1, 1, 1	9 + 60 = 69 3+ 60 = 63
A run	sumOfDice + 40	4, 5, 6 3, 1, 2	15 + 40 = 55 6 + 40 = 52
A pair	sumOfDice + 20	1, 1, 4 5, 1, 5	6+20=26 11+20=31
All different	sumOfDice	2, 3, 6 5, 1, 4	11 10

I will have to develop the game so that the user is told what dice values they roll at the end of each and every round. Whoever gets the highest points in a round (not sum of dice) wins the round. Whoever wins most rounds, wins the game. I will have to print to the user who has won the game and will also have to output the average score across all rounds. Validation in my code will be important. I will have to make sure that entering invalid data won't break my program. For example, if the user enters -1 for the number of rounds the user intends to play, I will have to prompt the user to enter the input again with a valid answer

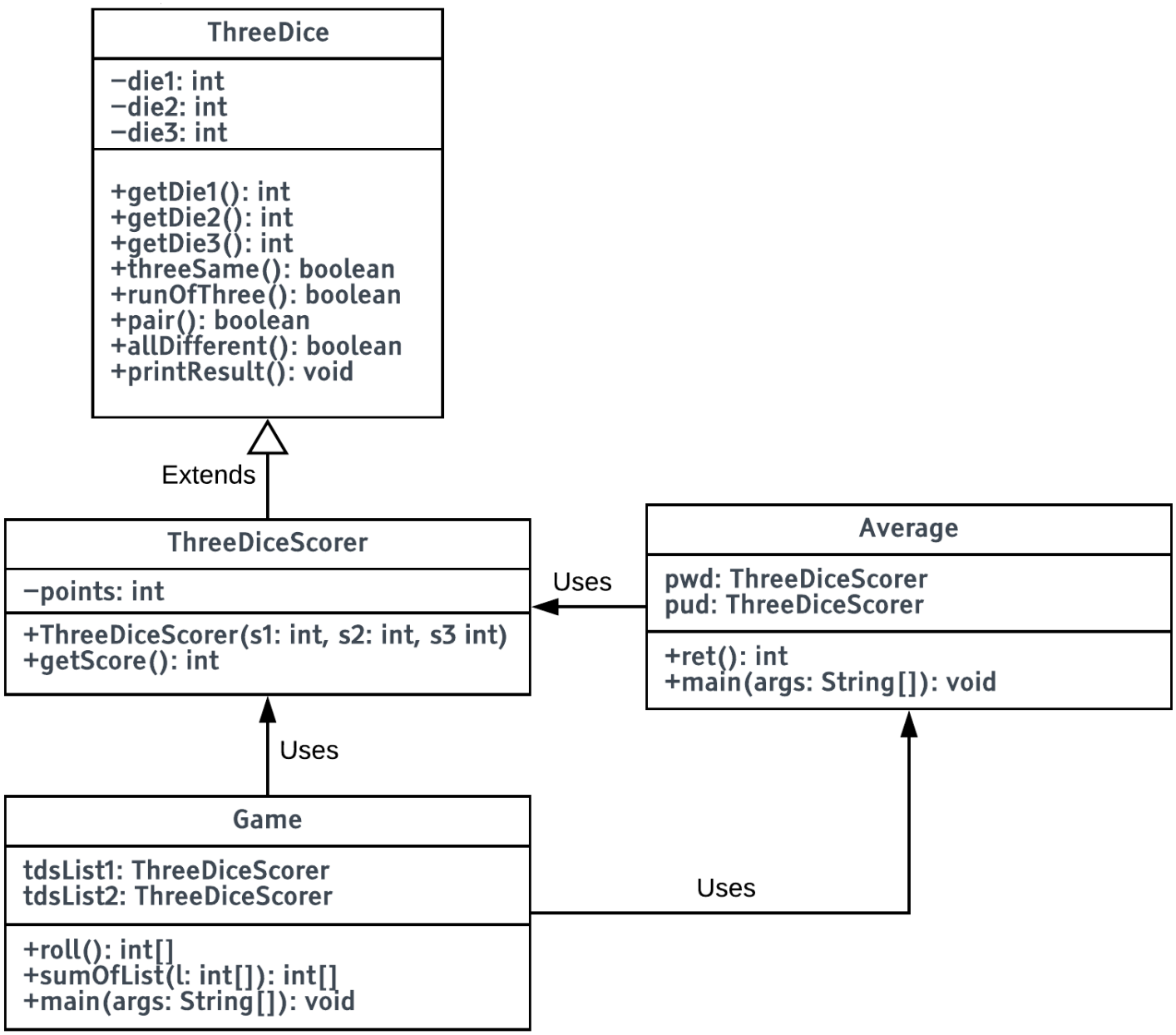
O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Class diagram

I have created a class diagram to display the design and to help to plan my solution. It is important that I know how to will structure my work as this can reduce errors and reduce time and will help for anyone wanting to understand my code.



I chose my design to be like this because I believe it is an efficient way of doing this. If I extend the ThreeDice class, I won't have to rewrite data and I can use the same function that class has. The Average class is it's own independent return average function, I could have easily rewritten this in the game file, but I thought it would be a waste if I rewrote code.

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Pseudocode

Game

```
FUNCTION roll():
        LIST l
        FOR i IN 0,5:
                l[i] <- RANDOM_INTEGER(1,6)
        RETURN l

FUNCTION sumOfList(list):
        sum <- 0
        FOR i IN 0, list.length():
                sum <- sum + l[i]
        RETURN sum

FUNCTION main():
        WHILE TRUE:
                OUTPUT("Input the number of rounds to play (min 0): ")
                rounds <- INPUT_INTEGER
                IF rounds > -1:
                        BREAK
                ELSE:
                        OUTPUT("Please enter a positive integer or 0")
        player1scores <- LIST
        player2scores <- LIST
  tdsList1 <- LIST
  tdsList2 <- LIST
  IF rounds != 0:
                WHILE (rounds >= roundspassed):
                        r <- roll()
                        tdsList1 <- ThreeDiceScorer(r[0],r[1],r[2])
                        tdsList2 <- ThreeDiceScorer(r[3],r[4],r[5])
                        OUTPUT("Round: "+roundspassed)
                        OUTPUT("Player 1: "+tdsList1[roundspassed-1].getDie1()+" "+tdsList1[roundspassed-1].getDie2()+" "+tdsList1[roundspassed-1].getDie3())
                        OUTPUT("Score: "+tdsList1[roundspassed-1].getScore())
                        OUTPUT("Player 2: "+tdsList2[roundspassed-1].getDie1()+" "+tdsList2[roundspassed-1].getDie2()+" "+tdsList2[roundspassed-1].getDie3())
                        OUTPUT("Score: "+tdsList2[roundspassed-1].getScore())
                        IF (tdsList1[roundspassed-1].getScore() > tdsList2[roundspassed-1].getScore()):
```

O.Legg@student.liverpool.ac.uk

```
                        OUTPUT("Round winner is player 1.")
                ELSE:
                        IF (tdsList1[roundspassed-1].getScore() == tdsList2[roundspassed-1].getScore()):
                                OUTPUT("Round is tied!")
                        ELSE:
                                OUTPUT("Round winner is player 2.")
                        roundspassed <- roundspassed + 1
        FOR i IN 0, rounds:
                player1scores[i] <- tdsList1[i].getScore()
                player2scores[i] <- tdsList2[i].getScore()
        FOR i IN 0, rounds:
                IF (player1scores[i] > player2scores[i]):
                        totalWins1 <- totalWins1 + 1
                ELSE:
                        IF (player1scores[i] == player2scores[i]):
                                //do nothing
                ELSE:
                        totalWins2 <- totalWins2 + 1
        OUTPUT("Total wins: Player 1: "+totalWins1+"    Player 2: "+totalWins2)
        OUTPUT("Total points: Player 1: "+sumOfList(player1scores)+" Player 2: "+sumOfList(player2scores))
OUTPUT("Average points per rounds: Player 1: "+Average.ret(player1scores))
OUTPUT("Player 2: "+Average.ret(player2scores))
IF (totalWins1 > totalWins2):
                OUTPUT("Overall points winner is player 1.")
        ELSE:
                IF (totalWins1 == totalWins2):
                        OUTPUT("Overall points outcome is tied!")
                ELSE:
                        OUTPUT("Overall points winner is player 2.")
    ELSE:
                EXIT()
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

ThreeDiceScorer

```
FUNCTION ThreeDiceScorer(s1, s2, s3)
   INHERIT(s1,s2,s3)   //inherits from ThreeDiceScorer
   points <- s1+s2+s3   //adds the dices to get the base points

FUNCTION getScore():
   IF (threeSame())
      RETURN points + 60
   IF (runOfThree())
      RETURN points + 40
   IF (pair())
      RETURN points + 20
   RETURN points
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Average

```
FUNCTION ret(l):
        sum <- 0
        FOR i IN 1,l.length:
                sum <- sum + l[i]
        RETURN (sum/l.length())


FUNCTION main():
        unfairdice <- [2,3,4,5,6,6]
        pwdScores  <- [] //points with dice total (pwd)
        pudScores  <- [] //points with unfair dice total (pud)
        index <- 0

        FOR x IN 1,6:
                FOR y IN 1,6:
                        FOR z IN 1,6:
                                pwd <- ThreeDiceScorer(x, y, z) //points with dice (pud)
                                pwdScores[index] <- pwd.getScore()
                                index <- index + 1

        OUTPUT("Average with 3, fair, 6 sided die: ")
        OUTPUT(ret(pwdScores))
        index <- 0
        FOR x IN 1,6:
                FOR y IN 1,6:
                        FOR x IN 0,5:
                                pud <- ThreeDiceScorer(x, y, unfairdice[z]) //points with dice (pud)
                                pudScores[index] <- pud.getScore()
                                index <- index + 1
        OUTPUT("Average with 2, fair, 6 sided die and 1 unfair dice being: {2,3,4,5,6,6}: ")
        OUTPUT(ret(pudScores))
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Part 2

For part 2 I will be writing a program to display information to a potential client for a company that supplies broadband, TV and mobile phone packages together. The company provides three different types of broadband, TV and mobile phone packages:

	Bronze	*Silver*	*Gold*
Text Messages	£0.00	£0.00	£0.00
Package Costs (per month)	£36.00	£46.00	£61.00
Daytime phone costs (per minute)	£0.12	£0.12	£0.00
Evening/Weekend phone costs (per minute)	£0.05	£0.00	£0.00
Number of TV Channels	60	130	230
Broadband (included per month)	500Mb	1000Mb	1520Mb
Broadband (per Mb above included amount)	£0.02	£0.01	£0.01
Free Spotify	False	True	True
Music on demand	False	False	True

I plan to develop this system using javas inheritance system having a standard account holding the account information that they share. I will then extend the standard account into bronze silver and gold giving them their own values (i.e. the cost of their own packages and services). The client who intends to use this system will enter 3 inputs at the start of the program:

- Daytime minutes (in phone calls)
- Evening / weekend minutes (in phone calls)
- How many megabytes are used per month on their broadband.

I will have to validate this input and make sure that user doesn't enter a negative integer. Afterwards, my program will calculate how much each package will end up costing based upon how many minutes / Mb they use. To make it easy for the user I will output which package would end up being the cheapest. If the package costs are all equally the same, I will recommend the gold account as it offers the most packages – on top of this I will explain to the user why I offered the gold account to the user. If there are 2 packages which are the cheapest and are equally as cheap, I will again offer the one with most packages and explain why I did that.

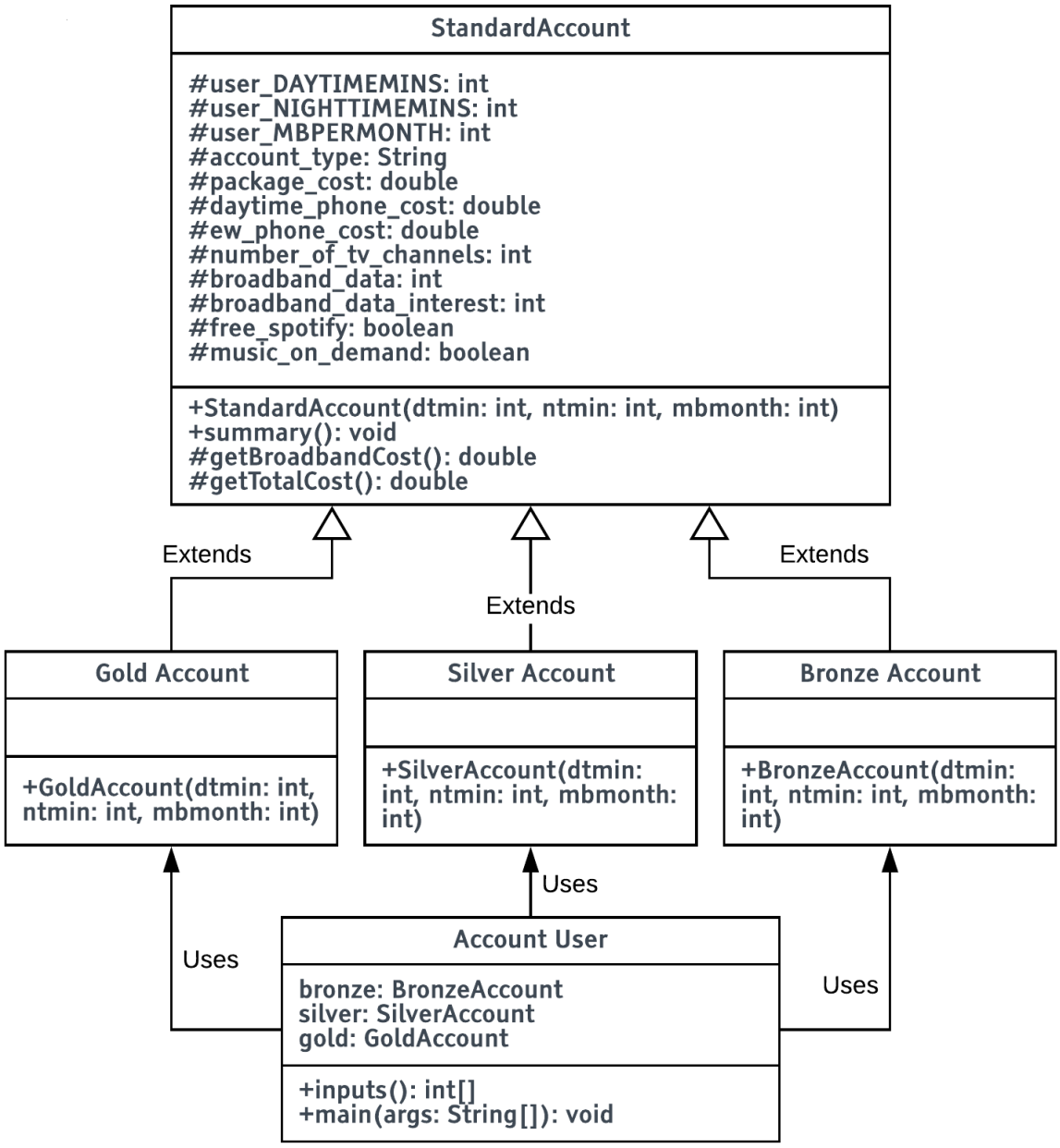
O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Class diagram

I have created a class diagram to display the design and to help to plan my solution. It is important that I know how to will structure my work as this can reduce errors and reduce time and will help for anyone wanting to understand my code.



I wanted my class design and structure to be like the way I designed it because it's very versatile. If I wanted to implement another package, I could just extend the standard account or the gold, silver or bronze account depending on what data differences there were. The account user can easily instantiate and remove the accounts. If I was to redesign my UML I would create a method in my account user that can sort multiple account packages to find the cheapest solution. This way I can implement a new package that can be easily added and removed.

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Pseudocode

Standard Account

```
ABSTRACT CLASS StandardAccount:
        PROTECTED user_DAYTIMEMINS      //users parameters    [integer]
        PROTECTED user_NIGHTTIMEMINS    //users parameters    [integer]
        PROTECTED user_MBPERMONTH       //users parameters    [integer]
        PROTECTED account_type          //bronze/silver/gold  [String]
        PROTECTED package_cost          //per month
        PROTECTED daytime_phone_cost    //per minute          [pence]
        PROTECTED ew_phone_cost         //per minute          [pence]
        PROTECTED number_of_tv_channels //                    [integer]
        PROTECTED broadband_data        //megabytes per month [integer]
        PROTECTED broadband_data_interest //amount over limit [pence]
        PROTECTED free_spotify          //£0 spotify on S+G   [boolean]
        PROTECTED music_on_demand       //music on demand S+G [boolean]

        FUNCTION StandardAccount(dtmin, ntmin, mbmonth):
            user_DAYTIMEMINS   <- dtmin
            user_NIGHTTIMEMINS <- ntmin
            user_MBPERMONTH    <- mbmonth

        //prints out the cost of overything and what's included
        FUNCTION summary():
                OUTPUT("Account Summary for "+account_type+" Account:")
                OUTPUT("Package Cost: "+package_cost))
                OUTPUT("Cost of daytime calls: "+daytime_phone_cost)+"/min");
                OUTPUT("Cost of evening and weekend calls: "+ew_phone_cost)+"/min")
                OUTPUT("Number of Channels: "+number_of_tv_channels)
                OUTPUT("Broadband Included: "+broadband_data+"Mb")
                OUTPUT("Broadband Cost (above included limit): "+broadband_data_interest))
                OUTPUT("Total daytime calls cost: "+(daytime_phone_cost*user_DAYTIMEMINS)))
                OUTPUT("Total evening calls cost: "+(ew_phone_cost*user_NIGHTTIMEMINS)))
                OUTPUT("Total (extra) broadband cost: "+getBroadbandCost()))
                OUTPUT("Total cost: "+getTotalCost()))
```

O.Legg@student.liverpool.ac.uk

```
        OUTPUT("Spotify Account provided: "+free_spotify)
        OUTPUT("Music on Demand provided: "+music_on_demand)


//broadband_data, broadband_data_interest, user_MBPERMONTH
FUNCTION getBroadbandCost():
        RETURN broadband_data_interest*MAX(0, user_MBPERMONTH-broadband_data))
        //the smallest number the number returned above is 0 because of the MAX function

//gets the total cost
FUNCTION getTotalCost() :
        RETURN (daytime_phone_cost*user_DAYTIMEMINS)+(ew_phone_cost*user_NIGHTTIMEMINS)+getBroadbandCost()+package_cost
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Gold Account

```
CLASS GoldAccount EXTENDS StandardAccount:
        FUNCTION GoldAccount(int dtmin, int ntmin, int mbmonth):
                INHERIT(dtmin, ntmin, mbmonth)
                account_type <- "Gold"
                package_cost <- 61.00
                daytime_phone_cost <- 0.0
                ew_phone_cost <- 0.0
                number_of_tv_channels <- 230
                broadband_data <- 1520
                broadband_data_interest <- 0.01
                free_spotify <- true
                music_on_demand <- true
```

Silver Account

```
CLASS SilverAccount EXTENDS StandardAccount:
        FUNCTION SilverAccount(dtmin, ntmin, mbmonth):
                INHERIT (dtmin, ntmin, mbmonth)
                account_type <- "Silver"
                package_cost <- 46.00
                daytime_phone_cost <- 0.12
                ew_phone_cost <- 0.00
                number_of_tv_channels <- 130
                broadband_data <- 1000
                broadband_data_interest <- 0.01
                free_spotify <- true
                music_on_demand <- false
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

Bronze Account

```
CLASS BronzeAccount EXTENDS StandardAccount:
        FUNCTION BronzeAccount(dtmin, ntmin, mbmonth):
                INHERIT(dtmin, ntmin, mbmonth)
                account_type <- "Bronze"
                package_cost <- 36.00
                daytime_phone_cost <- 0.12
                ew_phone_cost <- 0.05
                number_of_tv_channels <- 60
                broadband_data <- 500
                broadband_data_interest <- 0.02
                free_spotify <- false
                music_on_demand <- false
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Account User

```
FUNCTION inputs():
        inputs <- LIST
        WHILE (TRUE):
                OUTPUT("Please enter the number of daytime minutes used per month: ")
                inputs[0] <- INPUT()
                IF (inputs[0] >= 0)
                        BREAK
                ELSE
                        OUTPUT("Number has to be greater than or equal to 0")

        WHILE (TRUE):
                OUTPUT("Please enter the number of evening/weekend minutes used per month: ")
                inputs[1] <- INPUT()
                IF (inputs[1] >= 0)
                        BREAK
                ELSE
                        OUTPUT("Number has to be greater than or equal to 0")

        WHILE (TRUE):

                OUTPUT("Please enter the number of Megabytes used per month: ")
                inputs[2] <- INPUT()
                IF (inputs[2] >= 0)
                        BREAK
                ELSE
                        OUTPUT("Number has to be greater than or equal to 0")
        RETURN inputs

FUNCTION main():
        inputs <- inputs()
        bronze <- BronzeAccount(inputs[0],inputs[1], inputs[2])
        silver <- SilverAccount(inputs[0],inputs[1], inputs[2])
        gold   <- GoldAccount(inputs[0],inputs[1], inputs[2])
        bronze.summary()
        silver.summary()
        gold.summary()
        cost = []
        temp = 0.00
        cost[0] <- bronze.getTotalCost()
        cost[1] <- silver.getTotalCost()
        cost[2] <- gold.getTotalCost()
        //putting the cheapest cost in order
        IF (cost[1] < cost[0]):
          tmp <- cost[1]
```

O.Legg@student.liverpool.ac.uk

```
  cost[1] <- cost[0]
  cost[0] <- tmp

IF (cost[2] < cost[1]):
  tmp <- cost[2]
  cost[2] <- cost[1]
  cost[1] <- tmp

IF (cost[1] < cost[0]):
  tmp <- cost[1]
  cost[1] <- cost[0]
  cost[0] <- tmp

IF (cost[0] == cost[2]):
        OUTPUT("All account types are equally as cheap. However, we recommend that you use the "+gold.account_type+" account.")
        OUTPUT("This is because you get the most TV channels and a free Spotify account.")
        OUTPUT("You also have the ability to have unlimited calls and have less interest on going over broadband interest.")
ELSE:
        IF (cost[0] == cost[1]):
                IF ((cost[0] == bronze.getTotalCost() AND cost[1] == silver.getTotalCost() ) OR (cost[0] == silver.getTotalCost() AND cost[1] == bronze.getTotalCost())):
                        OUTPUT("Bronze account and Silver Account types are equally as cheap. However, we recommend that you use the "+silver.account_type+" account.")
                        OUTPUT("This is because you get more TV channels a free Spotify account and music on demand.")
                        OUTPUT("You will also get more minutes than the bronze account and you're charged less interest on going over broadband limit.")
                ELSE:
                        IF ((cost[0] == bronze.getTotalCost() AND cost[1] == gold.getTotalCost() ) OR (cost[0] == gold.getTotalCost() AND cost[1] == bronze.getTotalCost())):
                                OUTPUT("Bronze and Gold account types are equally as cheap. However, we recommend that you use the "+gold.account_type+" account.")
                                OUTPUT("This is because you get the most TV channels, a free Spotify account and music on demand.")
                                OUTPUT("You also have the ability to have unlimited calls and you're charged less interest on going over broadband limit.")
                        ELSE:
                                IF ((cost[0] == silver.() AND cost[1] == gold.getTotalCost() ) OR (cost[0] == gold.getTotalCost() AND cost[1] == silver.getTotalCost())):
                                        OUTPUT("Gold and Silver account types are equally as cheap. However, we recommend that you use the "+gold.account_type+" account.")
                                        OUTPUT("This is because you get the most TV channels, a free Spotify account and music on demand.")
                                        OUTPUT("You also have the ability to have unlimited calls and you're charged less interest on going over broadband limit.")
        ELSE:
                IF (cost[0] == bronze.getTotalCost()):
                        OUTPUT(bronze.account_type+" Account is cheapest cost.")
                ELSE:
                        IF (cost[0] == silver.getTotalCost()):
                                OUTPUT(silver.account_type+" Account is cheapest cost.")
                        ELSE:
                                OUTPUT(gold.account_type+" Account is cheapest cost.")
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Testing

Part 1

Test number	Description	Expected Results	Actual results	Remedial Action
1	Input validation 1	Entering a negative number will ask the user to enter a valid answer and ask the question again	My program told the user to enter a valid answer and ask the question again	-
2	Input validation 2	Inputting a char or string will crash the program	Crash	-
3	Points calculation	Points are calculated correctly per round	Points were calculated correctly every round	-
4	Random ints	All dice rolls are equally as random	All dice rolls were equally as random	-
5	Total Wins	total wins are correct	total wins were correct	-
6	Round Result	Round results being correct e.g. win or draw and displaying the winner of the player	Round results were correct. Correct player winner and was displayed. When there was a draw, it displayed that there was a draw.	-
7	Round number	Correct round was displayed and increments each round	The correct round was displayed and incremented as expected	-
8	Average	The correct average of points for each player will be outputted at the end of the game	The correct average of points for each player was outputted at the end of the game correctly	-
9	Average of all dice possibilities	The Average of all possible rolls are outputted correctly	Correct: 24.944444444444443	-
10	Average of all possibilities with unfair dice	The Average of all unfair dice possible rolls to be outputted correctly with the dice being {2,3,4,5,6,6}	Correct: 25.77777777777778	-

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Part 2

Test number	Description	Expected Results	Actual results	Remedial Action
1	Input validation 1 Q1	Entering a negative number will ask the user to enter a valid answer and ask the question again	My program told the user to enter a valid answer and ask the question again	-
2	Input validation 2 Q1	Inputting a char or string will crash the program	Crash	-
3	Input validation 1 Q2	Entering a negative number will ask the user to enter a valid answer and ask the question again	My program told the user to enter a valid answer and ask the question again	-
4	Input validation 2 Q2	Inputting a char or string will crash the program	Crash	-
5	Input validation 1 Q3	Entering a negative number will ask the user to enter a valid answer and ask the question again	My program told the user to enter a valid answer and ask the question again	-
6	Input validation 2 Q3	Inputting a char or string will crash the program	Crash	-
7	Information of Bronze account entered correctly	All costs and information are correct	All costs and information are correct	-
8	Information of Silver account entered correctly	All costs and information are correct	All costs and information are correct	-
9	Information of Gold account entered correctly	All costs and information are correct	All costs and information are correct	-
10	Total daytime costs	Calculation of all 3 accounts are correct for the total daytime cost	All costs are correct, and calculation have been done correctly	-
11	Total evening call cost	Calculation of all 3 accounts are correct for the total evening call cost	All costs are correct, and calculation have been done correctly	-
12	Total (extra) broadband cost	Calculation of all 3 accounts are correct for the total extra broadband cost	All costs are correct, and calculation have been done correctly	-
13	Total cost	Calculation of all 3 accounts are correct for the total cost	All costs are correct, and calculation have been done correctly	-

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Extra questions

Part 1

Question C

To get all the possible rolls, I created 3 nested FOR loops which all run from 1 to 6 which would increment on the values of x, y and z. As it loops through them all, it will run 6 * 6 * 6 times (216). Which is the total number of dice possibilities. As a result. The average result ended up being:

24.944444444444443

Question D

This was very similar to question C. I created 3 FOR loops. 2 of them ran from 1 to 6 just like the last one. But I created a list with the values of the unfair dice {2,3,4,5,6,6} therefore I would have to utilise this somehow. The final for loop would need to iterate on the index of the values in the list. Therefore, the loop ran from 0 to 6 and outputted as 2, 3, 4, 5, 6, 6 instead of it being 0,1,2,3,4,5 or 1,2,3,4,5,6.

This is the result:

25.77777777777778

Part 2

Question D

“Suppose that the company wants to introduce a “Platinum Account”, which has a monthly cost of £75, increases the number of TV channels to 275, and has unlimited broadband (i.e. no set limit on broadband included, so no “additional” costs on a per Mb basis). Otherwise, all other features about the Platinum Account are the same as the Gold Account. How could you implement this additional account, in terms of your class hierarchy and the way you have implemented your application program? Describe clearly what you would do to add this new account type. What would you do in terms of Java programming, e.g. modifying/overriding methods, adding new constants, etc?”

To implement this new Platinum Account, I would extend the Gold class and set the values to the information that had to be changed. I wouldn’t have to use `@Override` as I am not overwriting any methods in my case, I’m just changing protected attributes and have no constants needed to change. The constructor I use would be very similar to the gold account, silver account and bronze account however, I am only changing the attributes as I mentioned before. I would require the same parameters to be passed as it will still require a user input to calculate the costs.

O.Legg@student.liverpool.ac.uk