**Software Engineering Group Report**

**MCOMD2SWE Software Engineering**

**Assignment Deadline: 27th November, 2020**

**Group Members:**

Nathan Shuttleworth, Charlotte Brown, Audrey Smith, Ben Fowler

**Budget Calculator**

The software project that we officially decided on as a team was a budgeting calculator that was mainly aimed at students. As students ourselves, we believed that this project would be best catered by us as we ourselves were students. This also meant that we were able to better estimate the user stories, speaking from potential users ourselves.

By using GitHub as our main version control, and Trello for our agile sprint board, we were able to keep track of our progress as a team through these third party websites (with links to each in the report).

**GitHub Repository:** <https://github.com/BF118/Software-Engineering.git>

**Trello Board:** <https://trello.com/b/79skyfuF>

Establishing the requirements of the software, our main three concerns were mainly around, 1) how the user would enter the data, and 2) how the user would be given back the data. The notion of different time frames in which the user would track their money became something that shaped the final version of our ideal software iteration, but we focused on the first iteration requirements first.

Keeping the MoSCoW method in mind, we put “Must-Have” elements into the first iteration, such as the base functionality of the inputs and outputs, and left “Should-Have” and “Could-Have” elements for subsequent sprints. These elements included but were not limited to,

The full requirements of our program are listed below:

**Inputs**

* Total loan, separated into three installments
* Essentials per timeframe (with additional labels)
* Food per timeframe
* Luxuries per timeframe
* Study Money per timeframe

**Outputs**

* Total money spent per week
* Money left in budget
* Amount of money spent for essentials and luxuries
* Average amount spent per day

**Saved Data - to be saved to text file**

* Initial money in budget
* Money left in budget
* Total amount spent so far

And with it, the first version of our UI was created around these requirements. As can be seen, this first version highlights the most important functionalities and takes no real design into account. This is because we as a team felt that function was more important, at least initially, than appearance. Later on we tidied up a bit and made the UI more appealing visually.

By using input boxes for the main input section, the list of what the user had inputted would then be saved to variables and listed in the table. With each press of ‘Confirm’, the calculations made would output to the user with the results, but also save the data to a text file or database (tbd).



**Stories and Backlog**

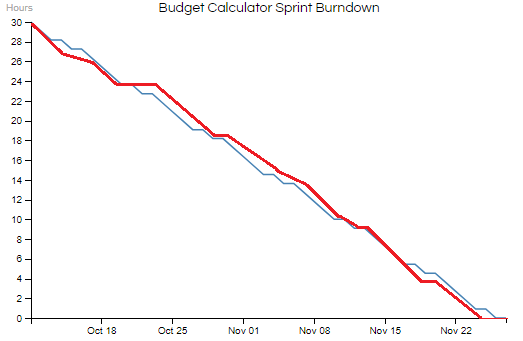
**(need to reformat conditions)**

| **User Story** | **Conditions of Satisfaction** | **Prioritisation/Sizing** |
| --- | --- | --- |
| As a regular person using software  I want to have a UI to input data  So that I can easily update what I am inputting to the system | For users using our software, when they are presented with the menu, they’re able to easily navigate the input boxes | **2** |
| As a regular person using a system  I want the results of the system to be graphically outputted to me  So that they can be clearly understood and used by me | For users using our software, when they are presented with the menu, they’re able to easily read the output boxes | **2** |
| As a person regularly looking to check their budgeting  I want to be able to track what day it is and when I entered information  So that I can see when I spent money | For users using our software, when they are presented with the menu, they’re able to easily perceive the date as well as which days the information they entered was stored | **1** |
| As a developer  I need to have inputs stored clearly for easy access  So that I can grab certain parts for calculations etc. | For our team members working on our software, when we’re developing the code, we should be able to create a class for easy access to input variables | **1** |
| As a developer  i need my output to be calculated elsewhere  so that it makes it easier to understand | For our team members working on our software, when we’re developing the code, we should be able to create a class for easy access to our output variables that’s separate from our calculations | **1** |
| As a someone earning money  I want to be able to enter my income  So that by budget can increase | For potential users of our software, when they come to use our calculator, they should be able to enter their budget easily | **2** |
| As a person who needs to keep track of finance  I want to be able to calculate how much money I have spent in the week so far  So that I can see exactly how much I have spent | For potential users of our software, when they come to use our calculator, they should be able to access how much money they’ve spent so far | **1** |
| As a person looking to manage my money  I want to be able to enter my budget for the week  So that I can use this data with other inputs to work out how much money I have | For users using our software, when they are presented with the menu, they should be able to enter their budget in the correct box without assistance | **1** |
| As a person looking to manage my spending  I want to be able to see how much I spend on different categories  So that I can better work out how much money I'll have left | For users using our software, when inputting their different spending habits/items, they should be able to to select and differentiate the items into categories | **2** |
| As a person looking for an easy-to-understand application to manage my budget  I want to be able to enter and exit the application smoothly  So it doesn't indefinitely run | For users using our software, when entering and exiting the application, it should be made clear how to start up and boot down the calculator | **1** |
| As a developer looking to make a successful application  I want to be able to include validation in my program/UI  So users don't break the software | For users and developers using the software, when inputting numbers and letters in the input boxes, the table should not allow letters in the number boxes and vice versa | **3** |
| As a person managing their finances  I want to have the total spent so far be subtracted from my weekly budget  So that I can see how much money I have left to spend for the week | For users using our software, when using our calculator, they should be able to easily see their remaining budget without assistance | **1** |
| As a person wanting to manage my money  I want to be able to see how much I've spent of each category and their prices together  To better manage my budget | For users using our software, when using our calculator, they should be able to easily see their remaining budget totals for each category without assistance | **2** |

**Estimation of Stories**

Our product backlog column shows the prioritisation and estimation of our stories, using the method of affinity grouping but in groups ranging from 1 (being the highest/biggest priority) to 3 (largest effort).

**Sprint Burndown:**

****

By calculating the amount of effort put into the project each sprint by each member of the team, we managed to create the optimum work effort chart by which the team should attempt to work with (blue line). Estimating the total development time at forty hours of work total, spread between four team members, with a break in development each Monday and Wednesday, our chart factored in rest days for the team.

Looking at how the team worked in the actual sprint development process (red line), we can see that our team worked optimally with the original calculations. Although out of sync and behind by a few hours for the first portion of the sprint (around the end of sprint one and at the beginning of sprint two), the team picked up the pace to polish and debug any remaining issues around the end of sprint three.

**Velocity:**

By also using our established high priority story points on our Trello board, the vast majority of our original points were completed and polished off in time for our deadline. However, the last low-priority story points that were originally planned, such as averaging money spent on different categories per time frames outside of the week. These would have been an admirable addition to the original design of the calculator, but with the time given and the limited sprints, they had to be left behind. However, as these were not originally cemented for implementation in the final version, this is not a big loss on the team’s side.

**Estimated velocity calculations:**

*Per person, per week:*

*2 Weeks per sprint*

*4 Team Members*

*3-4 hours of work per week*

*((ideal work hours \* weeks) \* team members) / 3 = 10*

**Ideally 10 hours of work per sprint.**

First sprint estimated velocity = 10 hours/5 points

Actual velocity:

Achieved majority basic prioritisation (1) story points

Completed 7 story points

Ahead of schedule

Second sprint estimated velocity = 10 hours/6 points

Actual velocity:

Finishing off prioritisation (2) story points

Achieved 5 story points

Behind schedule

Third sprint estimated velocity = 10 hours

**First Sprint Progress:**

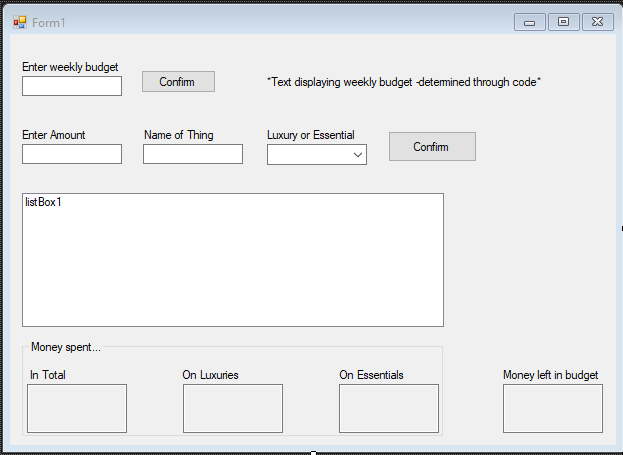
In the first sprint we focused on the basics of our program. This was the UI that we designed above, as well as where those inputs and outputs are stored. We also made sure the version control was correctly implemented so that we could work from home.

First Sprint Tasks:

* Create Version Control
* UI Design
* UI Inputs
* UI Outputs
* Class for storing inputted data
* Class for output calculations

The code for this sprint can be seen in the Appendices.

**MainCalculator.cs [Design]**



**User Manual**

*The main calculator UI was functional, and so users would be able to click buttons and enter text in the correct boxes. However, the dropdown menu has no selection coded in, nor do any of the buttons have any functionality. A user manual would be redundant/unneeded until the application began to develop more.*

**Second Sprint Progress:**

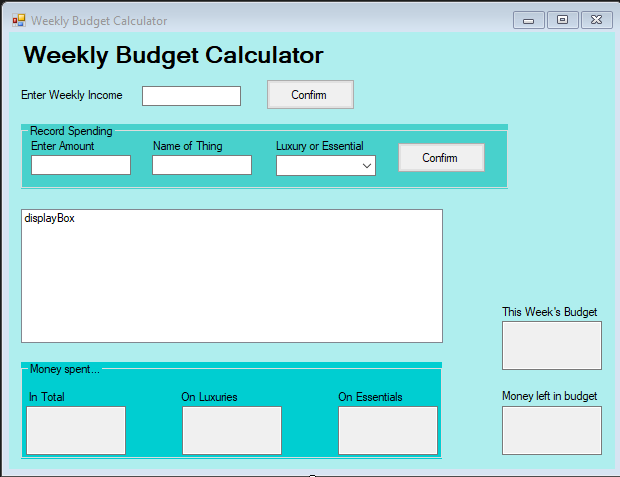
In the second sprint we worked towards bridging the gap between the inputs and outputs. The classes not only store the data, but they begin to work with it and put the variables together. Alongside this, we worked on optimising the UI menu for a smoother user experience, adding colour and formatting to the labels and the main background.

Second Sprint Tasks:

* Enter money coming in
* Enter money spent
* Enter luxury or essential
* Saving data

The code for this sprint can be seen in the Appendices.

**MainCalculator.cs [Design]**

****

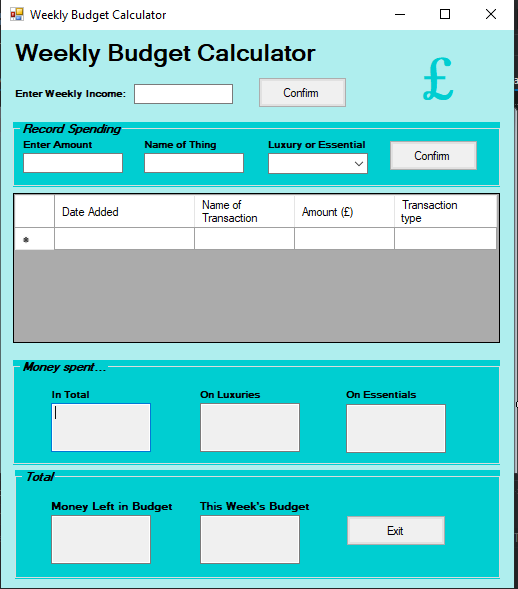
**User Manual**

*The main menu allows users currently to input their weekly income into the textbox. When pressing confirm, the calculator stores the weekly budget into another textbox for display and clarification.*

*Although the UI looks much smoother appealing, the rest of the menu is inoperable. The ‘Luxury or Essential’ box has no dropdown options, yet the Record Spending input boxes allow for inputting anyways (but without allowing users to include spaces in ‘Name of Thing’.*

**Third Sprint Progress:**

For our third and final sprint we focused on getting all of the calculations done as well as finishing up some things in the backend to do with the storage of data that will allow the users to record and save what they have entered for later convenience. Some more UI changes were also implemented to make the functions more clear.



**User Manual**

*All of the features are now present on the system. Firstly, the weekly income is entered at the top. This is the weeks budget, and is stated as the initial amount left in the budget. Next, the user enters their transactions for the week. They may enter as many as they like, stating the amount spent, the name of the transaction and whether they were spending money on luxury or essential items or services. This data is then inputted onto the grid, alongside the data it was inputted. The numbers below corresponding to the amount of money spent and the amount left is then updated. When the user presses the on-screen exit button, the data is saved, so that when the user picks up again they are able to see this and continue where they left off.*

**Fixed Date Planning:**

| As a regular person using a system.I want the results of the system to be graphically outputted to me. So that they can be clearly understood and used by me | 2 |  |
| --- | --- | --- |
| As a regular person using software.I want to have a UI to input data.So that I can easily update what I am inputting to the system | 2 | 4 |
| As a user that requires a budget planner. I require a user friendly design  So that I can better use the software | 1 | 5 |
| As a developer. I need to have inputs stored clearly for easy access  So that I can grab certain parts for calculations etc. | 1 | 6 |
| As a person regularly looking to check their budgeting.I want to be able to track what day it is and when I entered information.So that I can see when I spent money | 1 | 7 |
| As a developer. I need my output to be calculated else  so that it makes it easier to understand | 1 | 8 |
| As someone earning money. I want to be able to enter my income  So that by budget can increase | 2 | 10 |
| As a person looking to manage my spending. I want to be able to see how much I spend on different categories. So that I can better work out how much money I'll have left | 2 | 12 |
| As a developer looking to make a successful application. I want to be able to include validation in my program/UI. So users don't break the software | 3 | 15 |
| As a person who needs to keep track of finance. I want to be able to calculate how much money I have spent in the week so far. So that I can see exactly how much I have spent | 1 | 16 |
| As a person looking to manage my money. I want to be able to enter my budget for the week  So that I can use this data with other inputs to work out how much money I have | 1 | 17 |
| As a person looking for an easy-to-understand application to manage my budget  I want to be able to enter and exit the application smoothly. So it doesn't indefinitely run | 1 | 18 |
| As a person wanting to manage my money  I want to be able to see how much I've spent of each category and their prices together  To better manage my budget | 2 | 20 |
| As a person who regularly checks their budget. I want to be able to easily re-access my past results . So that I don't have to keep re-entering my information | 2 | 22 |
| As a person managing their finances. I want to have the total spent so far be subtracted from my weekly budget. So that I can see how much money I have left to spend for the week | 1 | 23 |
| As a person looking to manage their finances. I want to be able to see how much money I spend per day on average at the end of the week. So that I can better see how I should even out my spending | 1 | 24 |
| As a student my loan is split up per semester. i want to know how much i get per semester  so i can see how much i can spend | 2 | 26 |
| As a person managing their money. I want to be able to see what I am spending my money on  So that I can see the total amount spent on a certain thing e.g. food | 1 | 27 |
| As a person receiving multiple instalments of money. I want to be able to budget according to weeks, months and years. So that I can keep track of my money easier | 3 | 30 |
| As a student I need to spend an amount on food.I want to spend an average amount on food per week. So that i keep to my budget | 3 | 33 |
| As a student who needs to budget their finances.I need to be able to separate my food budget from my main budget. So I can save my money easier | 3 | 36 |
| As a student who needs to budget their finances. I need to be able to separate my food budget from my main budget. So I can save my money easier | 3 | 39 |

**Appendix**

**First Sprint Code:**

**Input.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Calculator

{

class Inputs

{

string nameOfTransaction;

string luxuryOrEssential;

double amountSpent;

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Calculator

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainCalculator());

}

}

}

**MainCalculator.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Calculator

{

public partial class MainCalculator : Form

{

double weeklyBudget;

public MainCalculator()

{

InitializeComponent();

}

private void TXTB\_NUM\_ONLY(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsDigit(e.KeyChar)) //Makes it so numbers can only be input

{

e.Handled = true;

}

}

private void TXTB\_CHAR\_ONLY(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !char.IsLetter(e.KeyChar)) //Makes it so only letters can be input

{

e.Handled = true;

}

}

private void uiEnterBudgetButton\_Click(object sender, EventArgs e)

{

weeklyBudget = double.Parse(uiEnterBudgetTextBox.Text);

uiWeeklyBudgetLabel.Text = ("Weekly budget = " + weeklyBudget.ToString());

}

private void uiTotalLuxTextbox\_TextChanged(object sender, EventArgs e)

{

}

private void uiEnterBudgetLabel\_Click(object sender, EventArgs e)

{

}

}

}

**Second Sprint Code:**

**Input.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace Calculator

{

class Inputs

{

string nameOfTransaction;

string luxuryOrEssential;

double amountSpent;

public Inputs(string transactionName, string transactionType, double transactionAmount)

{

nameOfTransaction = transactionName;

luxuryOrEssential = transactionType;

amountSpent = transactionAmount;

}

private void LuxOrEss()

{

if (luxuryOrEssential == "Luxury")

{

}

else

{

}

}

}

}

**Output.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Calculator

{

class Outputs

{

#region Attributes

private decimal leftBud;

private decimal Total;

private decimal spentLux;

private decimal spentEss;

#endregion

#region properties

public decimal GetleftBud

{

get

{

return leftBud;

}

set

{

leftBud = value;

}

}

public decimal GetTotal

{

get

{

return Total;

}

set

{

Total = value;

}

}

public decimal GetspentLux

{

get

{

return spentLux;

}

set

{

spentLux = value;

}

}

public decimal Getspentess

{

get

{

return spentEss;

}

set

{

spentEss = value;

}

}

#endregion

#region Constructors

public Outputs(decimal Tot, decimal left, decimal Lux, decimal Ess)

{

Total = Tot;

leftBud = left;

spentLux = Lux;

spentEss = Ess;

}

#endregion

}

}

**MainCalculator.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Calculator

{

public partial class MainCalculator : Form

{

double weeklyIncome;

double currentMoneyInBudget;

double TotalNumberSpent;

double TotalLuxSpent;

double TotalEssSpent;

string filePath\_Inputted = @"C:\Users\Nathan\Documents\GitHub\Software-Engineering\Calculator\Calculator\previouslyInputtedData.txt";

string filePath\_Calculations = @"C:\Users\Nathan\Documents\GitHub\Software-Engineering\Calculator\Calculator\previousCalculations.txt";

public MainCalculator()

{

InitializeComponent();

using (StreamReader sr = new StreamReader(filePath\_Inputted))

{

string line;

while ((line = sr.ReadLine()) != null)

{

string[] splitData = new string[5];

splitData = line.Split(' ');

Inputs previousData = new Inputs(splitData[2], splitData[4], double.Parse(splitData[3]));

string dateOfTransaction = splitData[0] + " " + splitData[1];

uiTransactionDataGrid.Rows.Add(dateOfTransaction, previousData.nameOfTransaction, previousData.amountSpent, previousData.luxuryOrEssential);

}

}

using (StreamReader sr = new StreamReader(filePath\_Calculations))

{

string line;

string[] linesRead = new string[5];

try

{

while ((line = sr.ReadLine()) != null)

{

for (int i = 0; i > 5; i++)

{

linesRead[i] = line;

}

}

weeklyIncome = double.Parse(linesRead[0]);

TotalNumberSpent = double.Parse(linesRead[1]);

TotalLuxSpent = double.Parse(linesRead[2]);

TotalEssSpent = double.Parse(linesRead[3]);

currentMoneyInBudget = double.Parse(linesRead[4]);

uiTotalIncomeTextBox.Text = ("£" + weeklyIncome.ToString());

uiIncomeTextBox.ReadOnly = true;

uiMoneyLeftTextbox.Text = ("£" + currentMoneyInBudget);

uiTotalLuxTextbox.Text = TotalLuxSpent.ToString();

uiTotalEssTextbox.Text = TotalEssSpent.ToString();

}

catch (Exception e) { }

}

}

private void TXTB\_CHAR\_ONLY(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !char.IsLetter(e.KeyChar)) //Makes it so only letters can be input

{

e.Handled = true;

}

}

private void TXTB\_NUM\_ONLY(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsDigit(e.KeyChar)&&(e.KeyChar != '.')) //Makes it so numbers can only be input

{

e.Handled = true;

}

}

private void uiIncomeButton\_Click(object sender, EventArgs e)

{

weeklyIncome = double.Parse(uiIncomeTextBox.Text);

uiTotalIncomeTextBox.Text = ("£" + weeklyIncome.ToString());

uiIncomeTextBox.ReadOnly = true;

currentMoneyInBudget = weeklyIncome;

uiMoneyLeftTextbox.Text = ("£" + currentMoneyInBudget);

}

public void uiSpendButton\_Click(object sender, EventArgs e)

{

if (uiIncomeTextBox.Text.Trim() == string.Empty)

{

MessageBox.Show("Please enter something in Weekly Income");

return; // return because we don't want to run normal code of button click

}

if (uiAmountTexbox.Text.Trim() == string.Empty|| uiItemNameTextbox.Text.Trim() == string.Empty)

{

MessageBox.Show("Please enter something in the empty Textbox");

return; // return because we don't want to run normal code of button click

}

string currentTransactionType = uiDropdown.SelectedItem.ToString();

string currentTransactionName = uiItemNameTextbox.Text;

double currentTransactionAmount = double.Parse(uiAmountTexbox.Text);

string currentDate = DateTime.Now.ToString();

Inputs latestInput = new Inputs(currentTransactionName, currentTransactionType, currentTransactionAmount);

uiTransactionDataGrid.Rows.Add(currentDate, latestInput.nameOfTransaction, latestInput.amountSpent, latestInput.luxuryOrEssential);

currentMoneyInBudget = currentMoneyInBudget - currentTransactionAmount;

uiMoneyLeftTextbox.Text = ("£" + currentMoneyInBudget.ToString());

string lineToWrite = (currentDate + " " + latestInput.nameOfTransaction + " " + latestInput.amountSpent + " " + latestInput.luxuryOrEssential);

using (StreamWriter sw = new StreamWriter(filePath\_Inputted))

{

sw.WriteLine(lineToWrite);

}

//Calculation

double CurrentAmountSpent = double.Parse(uiAmountTexbox.Text);

TotalNumberSpent = TotalNumberSpent + CurrentAmountSpent;

uiTotalSpentTextbox.Text = TotalNumberSpent.ToString();

//Validation

if (uiIncomeTextBox.Text.Trim() == string.Empty)

{

MessageBox.Show("Please enter something in the textbox");

return; // return because we don't want to run normal code of button click

}

if (currentMoneyInBudget <= 0)

{

MessageBox.Show("This will mean you don't have enough money left for the week");

return; // return because we don't want to run normal code of button click

}

if (currentTransactionType =="Luxury")

{

double totalAmount = TotalLuxSpent;

double TotalSpent = Convert.ToDouble(uiAmountTexbox.Text);

TotalLuxSpent = TotalSpent + totalAmount;

uiTotalLuxTextbox.Text = TotalLuxSpent.ToString();

}

if (currentTransactionType == "Essential")

{

double TotalAmount = TotalEssSpent;

double TotalSpent = Convert.ToDouble(uiAmountTexbox.Text);

TotalEssSpent = TotalSpent + TotalAmount;

uiTotalEssTextbox.Text = TotalEssSpent.ToString();

}

}

private void button1\_Click(object sender, EventArgs e)

{

using (StreamWriter sw = new StreamWriter(filePath\_Calculations))

{

sw.WriteLine(weeklyIncome);

sw.WriteLine(TotalNumberSpent);

sw.WriteLine(TotalLuxSpent);

sw.WriteLine(TotalEssSpent);

sw.WriteLine(currentMoneyInBudget);

}

System.Windows.Forms.Application.Exit();

}

}

}

**Third Sprint Code:**

**Inputs.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace Calculator

{

public class Inputs

{

public string nameOfTransaction;

public string luxuryOrEssential;

public double amountSpent;

public Inputs(string transactionName, string transactionType, double transactionAmount)

{

nameOfTransaction = transactionName;

luxuryOrEssential = transactionType;

amountSpent = transactionAmount;

}

private void LuxOrEss()

{

if (luxuryOrEssential == "Luxury")

{

}

else

{

}

}

}

}

**Outputs.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Calculator

{

class Outputs

{

#region Attributes

private decimal leftBud;

private decimal Total;

private decimal spentLux;

private decimal spentEss;

#endregion

#region properties

public decimal GetleftBud

{

get

{

return leftBud;

}

set

{

leftBud = value;

}

}

public decimal GetTotal

{

get

{

return Total;

}

set

{

Total = value;

}

}

public decimal GetspentLux

{

get

{

return spentLux;

}

set

{

spentLux = value;

}

}

public decimal Getspentess

{

get

{

return spentEss;

}

set

{

spentEss = value;

}

}

#endregion

#region Constructors

public Outputs(decimal Tot, decimal left, decimal Lux, decimal Ess)

{

Total = Tot;

leftBud = left;

spentLux = Lux;

spentEss = Ess;

}

#endregion

}

}