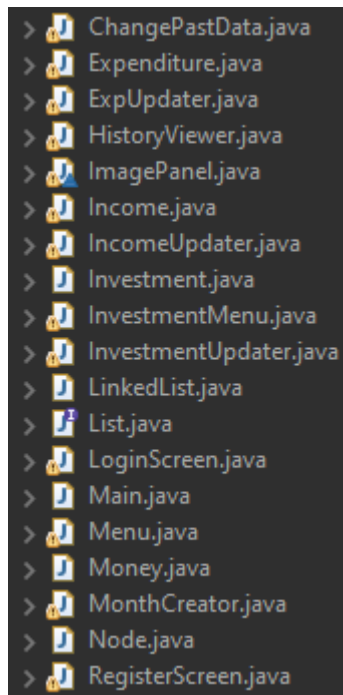


Criterion C: Development

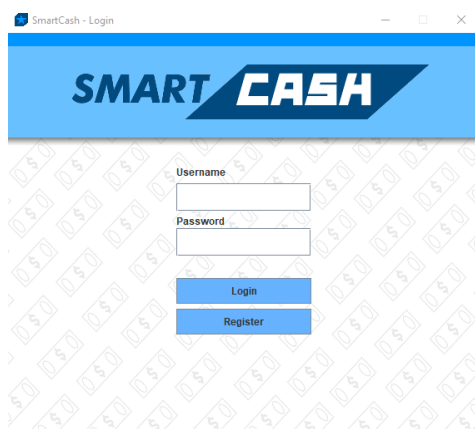
Classes



Functionality

- Polymorphism
- Encapsulation
- Inheritance
- Arrays
- LinkedList
- File Input/Output
- Static Methods & Variables
- Searching
- Sorting

Login Screen & GUI



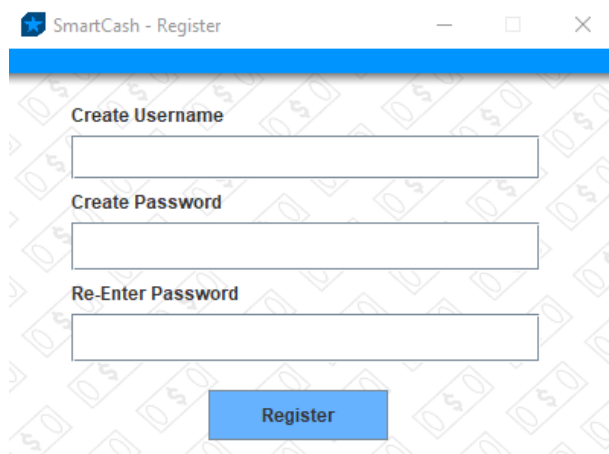
I created my own images and added them to the menu using the `ImagePanel` class, which extends the `JPanel` class introduced by `javax.swing`. The elements such as `JLabel`, `TextField`, `PasswordField`, and

JButtons were also added from this and implemented into this menu. Throughout the program, other features such as JOptionPane and JTextArea were used. Each menu has its own class and extends JFrame.

```
jButton1 = new JButton();
jButton2 = new JButton();
jTextField1 = new JTextField();
jPasswordField1 = new JPasswordField();
jLabel1 = new JLabel();
jLabel2 = new JLabel();

jButton1.setText("Login");
jButton1.setBackground(new Color(102,178,255));
jButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});
```

Registration Screen & Function



Creating an account is necessary to use the program. This screen enables the user to create their account, and after pressing the register button the program will check if the passwords match and if the username is unique & valid. If valid, the login information is added to a string of all login information, and written into the file “accounts.txt”.

```
try {
    LoginScreen.loginInfo += username + " " + password + "\n";
    FileWriter fw = new FileWriter("../TextFiles/accounts.txt");
    BufferedWriter br = new BufferedWriter(fw);
    br.write(LoginScreen.loginInfo);
    br.close();

    jOptionPanel = new JOptionPanel();
    JFrame f = new JFrame();
    JOptionPane.showMessageDialog(f, "You have successfully created an account. You may sign in now.");
    this.setVisible(false);
} catch (FileNotFoundException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
} catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}
```

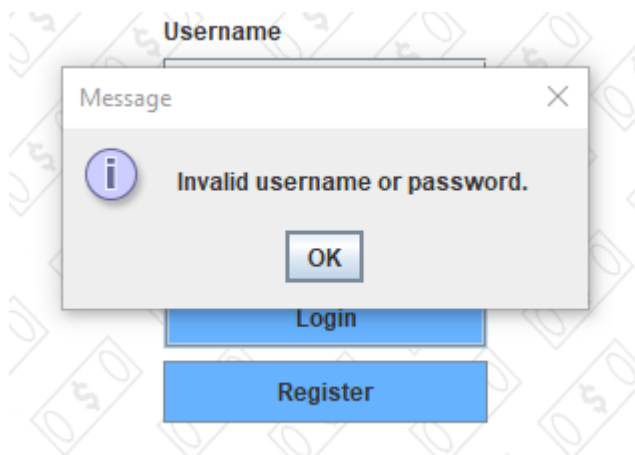
Within the text file, the data will be formatted as such:
username password

Login Function

```
private void jButton1ActionPerformed(ActionEvent evt) {
    String username = jTextField1.getText();
    String password = jPasswordField1.getText();
    boolean success = false;
    String existingUser = "";
    String existingPswd = "";
    try {
        Scanner sc = new Scanner(logins);
        while (sc.hasNextLine()) {
            String data = sc.nextLine();
            existingUser = data.substring(0, data.indexOf(' '));
            existingPswd = data.substring(data.indexOf(' ')+1,data.length());
            if(username.equals(existingUser) && password.equals(existingPswd)){
                System.out.println("Successful Login");
                success = true;
                if(firstLogin){
                    firstLogin = false;
                    currentUser = username;
                    createFile();
                    Menu.menu();
                }
            }
        }
    }
}
```

When the login button is pressed, the program gets the text from the two fields and scans through the file “logins” which is “accounts.txt” to find a matching set of information. If the information is incorrect, the program notify the user with an error message.

```
if(!success){
    JOptionPane.showMessageDialog(f,"Invalid username or password.");
}
```



If the data matches data in the file, the menu screen will open and the login screen will disappear.

Money Class

```
public class Money {
    private String month;
    public Money(String month) {
        this.month = month;
    }

    public String getMonth() {
        return this.month;
    }
    public void setMonth(String month){
        this.month = month;
    }
}
```

The Money class is a simple class which stores the month information. Since the Income and Expenditure classes each store different types of monetary information but have month in common, the two were made to inherit the Money class.

Income Class

```
public class Income extends Money{
    static String currentMonth = "000000";
    private double taxRate;
    private double rawIncome;
    private double income;
    static String incomeInfo = "";
    static int totalMonths = 0;
    static File f = new File("TextFiles/" + LoginScreen.currentUser + "info.txt");
    static int index = 0;

    public Income(String month, double taxRate, double rawIncome){
        super(month);
        this.taxRate = taxRate;
        this.rawIncome = rawIncome;
        if(taxRate == 0.0) this.income = rawIncome;
        else this.income = rawIncome - rawIncome * (taxRate/100.0);
        if(currentMonth.equals("000000")) {
            currentMonth = month;
        }
        try {
            Scanner sc = new Scanner(f);
            while (sc.hasNextLine()) {
                String data = sc.nextLine();
                if(!sc.hasNextLine()){
                    currentMonth = data.substring(0, data.indexOf(' '));
                }
                if(totalMonths == 0) {
                    incomeInfo += data + "\n";
                }
            }
            sc.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
        totalMonths++;
    }
}
```

The Income object has its own private variables taxRate, rawIncome, and income. It also has static variables incomeInfo, which stores all the data from the “usernameinfo.txt” text file, totalMonths which tracks the number of entries, index which stores the selected index of the LinkedList of the current month, and the File f which contains all the user data. The constructor of this class also tries to set the current month in order for the menu screen to know which data to show.

The object also comes with getter and setter methods for the private variables, as well as the ability to update the income based on the rawIncome and taxRate variables. This showcases encapsulation, which is beneficial in ensuring that no other methods will be able to change the data in this class and that there are no issues with multiple instances of the Income object.

```
public void updateIncome(){
    if(taxRate == 0.0) this.income = rawIncome;
    else this.income = rawIncome - rawIncome * (taxRate/100.0);
}
```

Expenditure Class

The Expenditure class is similar to the Income class as they both inherit Money, but has its own set of private variables and methods. It has getter and setter methods for each private variable and methods to calculate total expenditure.

```
private double luxuryExp = 0;
private double foodExp = 0;
private double transportExp = 0;
private double utilityExp = 0;
private double otherExp = 0;
```

```
double calcPastExp() {
    return luxuryExp+foodExp+transportExp+utilityExp+otherExp;
}
```

Polymorphism - Expenditure, Income, Money

Money - calculate() method

This method returns the double representation of the month.

```
public double calculate() {
    return Double.parseDouble(month);
}
```

Income - calculate() method

This method calculates the income.

```
public double calculate(){
    if(taxRate == 0.0) this.income = rawIncome;
    else this.income = rawIncome - rawIncome * (taxRate/100.0);
    return this.income;
}
```

Expenditure - calculate() method


This method calculates the total expenditure.


```
public double calculate() {
    return luxuryExp+foodExp+transportExp+utilityExp+otherExp;
}
```

Polymorphism is beneficial here because it reduces the amount of code that needs to be written for the program, allowing for editing to be easier.

Income & Expenditure Data

The income & expenditure data of the user will be saved in their own text file, specific to their username. For example, someone with the username “username” will have their income data stored in “usernameinfo.txt” and expenditure data stored in “usernameinfoExp.txt”. This is to ensure that multiple users can use the program at once without interfering with each others’ data.

 usernameinfo.txt

 usernameinfoExp.txt

If the user has already entered information and is re-opening the program, the program will try to read that data and add it to a LinkedList.

```
static LinkedList i = new LinkedList();
static LinkedList exp = new LinkedList();
```

```

public static void initLists(){
    File f = new File("./TextFiles/" + LoginScreen.currentUser + "info.txt");
    File fe = new File("./TextFiles/" + LoginScreen.currentUser + "infoExp.txt");
    // Reading existing income data
    try {
        Scanner sc = new Scanner(f);
        while (sc.hasNextLine()) {
            String data = sc.nextLine();
            String month = data.substring(0, data.indexOf(' '));
            double taxRate = Double.parseDouble(data.substring(data.indexOf(' ')+1, data.indexOf('-')));
            double rawIncome = Double.parseDouble(data.substring(data.indexOf('-')+1, data.length()));
            i.insert(Income.totalMonths, new Income(month, taxRate, rawIncome));
        }
        sc.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }

    // Reading existing expenditure data
    try {
        Scanner sc = new Scanner(fe);
        while (sc.hasNextLine()) {
            String data = sc.nextLine();
            String month = data.substring(0, data.indexOf(' '));
            double lux = Double.parseDouble(data.substring(data.indexOf('l')+1, data.indexOf('f')));
            double food = Double.parseDouble(data.substring(data.indexOf('f')+1, data.indexOf('t')));
            double transp = Double.parseDouble(data.substring(data.indexOf('t')+1, data.indexOf('u')));
            double utility = Double.parseDouble(data.substring(data.indexOf('u')+1, data.indexOf('o')));
            double other = Double.parseDouble(data.substring(data.indexOf('o')+1, data.length()));
            exp.insert(Expenditure.totalEntries, new Expenditure(month, lux, food, transp, utility, other));
        }
        sc.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

```

The LinkedLists are static in order for all classes to read the data. The LinkedList data type was used in order for data to be added in chronological order, which is helpful in case the user missed a month and wanted to insert a month in between two existing months.

```

public boolean insert(int i, Object item){
    Node prev;
    Node newNode = new Node(item);

    if(i == 0){
        if(!isEmpty())
            newNode.setNext(header);
        header = newNode;
    }
    else{
        prev = traverse(i - 1);
        if(prev == null)
            return false;
        newNode.setNext(prev.getNext());
        prev.setNext(newNode);
    }
    size++;
    return true;
}

```

Input Income/Expenditure Data

SmartCash - Financial Tracking Application

SMART CASH

Exit

Welcome back, username. Month: Add a month

Monthly Income: \$0.0 Update

Monthly Expenses: \$0 [Enter data] Update

Surplus Money: \$0 [Enter Data]

New Month Past Months Change Past Data

Savings & Investment Center

Personalized Advice

You are saving a good amount of money.
You are spending a healthy portion of your income on luxury goods!
Please enter transport expenditure data!
Please enter food expenditure data.

Pressing the “New Month” button will open the following screen:

SmartCash - Add New Entry

Select Month Enter Year

January

Create New Month

Exit

In this window, you must select a month and enter a year. If there is existing data, the program will check to ensure that the same month is not being created twice.

```
Main.exp.insert(index, new Expenditure(month, 0.0, 0.0, 0.0, 0.0, 0.0));  
Main.i.insert(index, new Income(month, 0.0, 0.0));
```

```
private void rewriteFromScratch() {  
    Expenditure.expInfo = "";  
    Income.incomeInfo = "";  
    for(int i = 0; i < Main.i.size; i++) {  
        Income.incomeInfo += ((Income) Main.i.lookup(i)).getMonth() + " " + ((Income) Main.i.lookup(i)).getAmount() + "  
    }  
    for(int i = 0; i < Main.exp.size; i++) {  
        Expenditure.expInfo += ((Expenditure) Main.exp.lookup(i)).getMonth() + " " + "1" + ((Expenditure) Main.exp.lookup(i)).getAmount() + "  
    }  
    try {  
        FileWriter fw = new FileWriter("./TextFiles/" + LoginScreen.currentUser + "infoExp.txt");  
        BufferedWriter br = new BufferedWriter(fw);  
        br.write(Expenditure.expInfo);  
        br.close();  
    } catch (FileNotFoundException e) {  
        System.out.println("An error occurred.");  
        e.printStackTrace();  
    } catch (IOException e) {  
        System.out.println("An error occurred.");  
        e.printStackTrace();  
    }  
    try {  
        FileWriter fw = new FileWriter("./TextFiles/" + LoginScreen.currentUser + "info.txt");  
        BufferedWriter br = new BufferedWriter(fw);  
        br.write(Income.incomeInfo);  
        br.close();  
    } catch (FileNotFoundException e) {  
        System.out.println("An error occurred.");  
        e.printStackTrace();  
    } catch (IOException e) {  
        System.out.println("An error occurred.");  
        e.printStackTrace();  
    }  
}
```

If the month is added in between two months, the strings Expenditure.expInfo and Income.incomeInfo are reinitialized using the two for loops and the file is rewritten from scratch.

```
if(Income.totalMonths <= 1) index = Income.totalMonths;
else if(Income.totalMonths > 1){
    for(int i = 0; i < Main.exp.size; i++) {
        String mData = ((Expenditure) Main.exp.lookup(i)).getMonth().substring(0,2);
        int yData = Integer.parseInt(((Expenditure) Main.exp.lookup(i)).getMonth().substring(2));
        if(year < yData) {
            System.out.println("if");
            index = i;
            break;
        }
        else if(year == yData && i < Main.exp.size-1) {
            if(Integer.parseInt(month) > Integer.parseInt(mData) && Integer.parseInt(month) < Integer.parseInt(((Expenditure) Main.exp.lookup(i+1)).getMonth().substring(0,2))) {
                System.out.println("else if");
                index = i+1;
                break;
            }
        }
        else if(year == yData && i == Main.exp.size-1) {
            System.out.println("else if 2");
            if(Integer.parseInt(month) > Integer.parseInt(mData) && year > Integer.parseInt(((Expenditure) Main.exp.lookup(i-1)).getMonth().substring(2))) {
                index = i+1;
                break;
            }
            else if(Integer.parseInt(month) > Integer.parseInt(mData)) {
                index = i+1;
                break;
            }
        }
        else if(year > yData) {
            System.out.println("else");
            index = i+1;
        }
    }
}
if(index == -1) index = Income.totalMonths;
```

The program calculates the correct index for which the new month should be inserted into the LinkedList.

Otherwise, the data is simply added to the end of the strings and the file is rewritten. If there is no existing data, the income and tax rate will be set to 0 for the newly created object. If there is data, the income and tax rate of the previous month will be used. All expenditure data is set to 0 regardless.

Welcome back, username. Month: January 2021
Monthly Income: \$0.0
Monthly Expenses: \$0 [Enter data]
Surplus Money: \$0 [Enter Data]

After creating a new month, the menu screen will be updated with the new month.

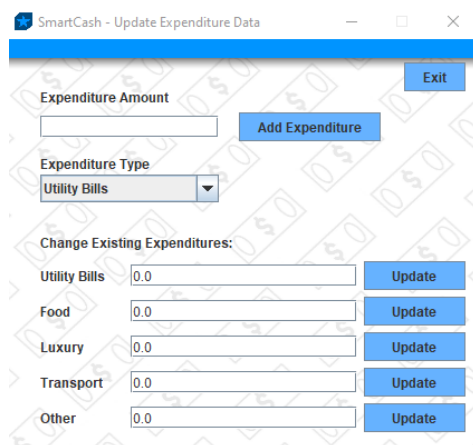
Changing Data

SmartCash - Update Income Data

Monthly Income:

Tax Rate:

Manually Change Income:



SmartCash - Update Expenditure Data

Expenditure Amount: Exit

Add Expenditure

Expenditure Type: Utility Bills

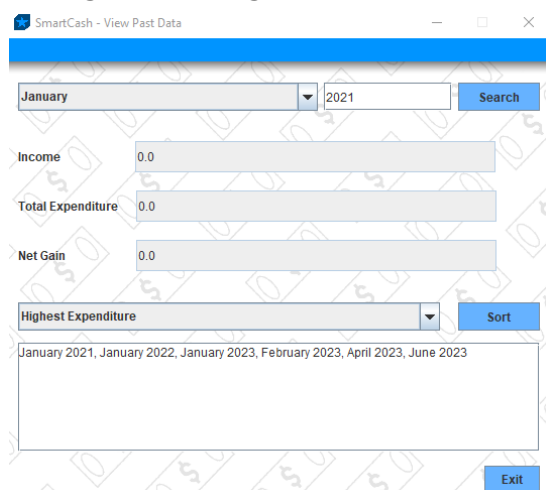
Change Existing Expenditures:

Utility Bills	<input type="text" value="0.0"/>	Update
Food	<input type="text" value="0.0"/>	Update
Luxury	<input type="text" value="0.0"/>	Update
Transport	<input type="text" value="0.0"/>	Update
Other	<input type="text" value="0.0"/>	Update

Upon interacting with the two “Update” buttons on the main menu, respective windows will open prompting the input of updated information. The entered information will be updated in the text file for the selected month. The program takes the data from the text fields and adds them to the data string, which gets written into the text files.

```
private void jButton5ActionPerformed(ActionEvent evt) {
    try{
        double expAmt = Double.parseDouble(jTextField5.getText());
        ((Expenditure) Main.exp.lookup(Expenditure.index)).setTransportExp(expAmt);
        if(Expenditure.index == Expenditure.totalEntries -1){
            Expenditure.updateAll();
            rewrite();
        }
        else {
            rewriteFromScratch();
        }
        jOptionPane1 = new JOptionPane();
        JFrame f = new JFrame();
        JOptionPane.showMessageDialog(f,"You have successfully updated your Transport expenditure data.");
        jTextField1.setText("");
        Menu.reload();
        this.reload();
    }
    catch(NumberFormatException e){
        jTextField2.setText("Invalid value. Please try again.");
    }
}
```

Sorting & Searching



SmartCash - View Past Data

January Search

Income:

Total Expenditure:

Net Gain:

Highest Expenditure Sort

January 2021, January 2022, January 2023, February 2023, April 2023, June 2023

Exit

The “Past Months” button on the menu will open a window that allows the user to sort and search through their data. The user may sort their months of data by highest/lowest expenditure and highest/lowest income.

```

public static String selecIncSortMax(LinkedList a) {
    String b = "";
    Income[] e = new Income[Income.totalMonths];

    for(int i = 0; i < a.size; i++) {
        e[i] = ((Income) a.lookup(i));
    }

    double max = -1;
    Income temp;
    Income tempmax = null;
    int pos = 0;
    for(int i = 0; i < a.size; i++) {
        for(int j = i; j < a.size; j++) {
            if(max == -1) {
                max = e[j].getIncome();
                tempmax = e[j];
                pos = j;
            }
            else if(max < e[j].getIncome()) {
                max = e[j].getIncome();
                tempmax = e[j];
                pos = j;
            }
        }
        temp = e[i];
        e[pos] = temp;
        e[i] = tempmax;

        max = -1;
    }

    for(int i = 0; i < a.size; i++) {
        if(i != a.size-1) b+=e[i].monthToString(e[i].getMonth(), Income.totalMonths) + ", ";
        else b+=e[i].monthToString(e[i].getMonth(), Income.totalMonths);
    }
    return b;
}

```

In order to keep the LinkedList in chronological order, the LinkedList data is imported into an array which is then sorted. The contents of this array will be printed into the text area, in the user's desired order. A selection sort was used to sort since most lists in this program would be small, making the method a very efficient one.

```

public static int seqExpSearch(LinkedList b, String m) {
    for(int i = 0; i < b.size; i++) {
        if(((Expenditure) b.lookup(i)).getMonth().equals(m)) return i;
    }
    return -1;
}

```

For the same reasons, a sequential search was used to search for the months. The LinkedList lookup(int i) method was used to do so. The returned value from this method had to be cast to Expenditure or Income in order to use their getMonth() method, since the lookup method returned type Object.

```

public Object lookup(int i){
    Node n = traverse(i);
    if(n == null) return null;
    return n.getData();
}

```

This method, alongside the insert method, used the traverse method to look through each node in the LinkedList.

```

private Node traverse(int i){
    Node n = header;
    if(i < 0) return null;

    for(int j = 0 ; j < i ; j++){
        if(n == null) return null;
        n = n.getNext();
    }
    return n;
}

```

This method checks if the LinkedList is connected and returns the ith node. The traverse method is useful to implement in many other methods within the LinkedList.