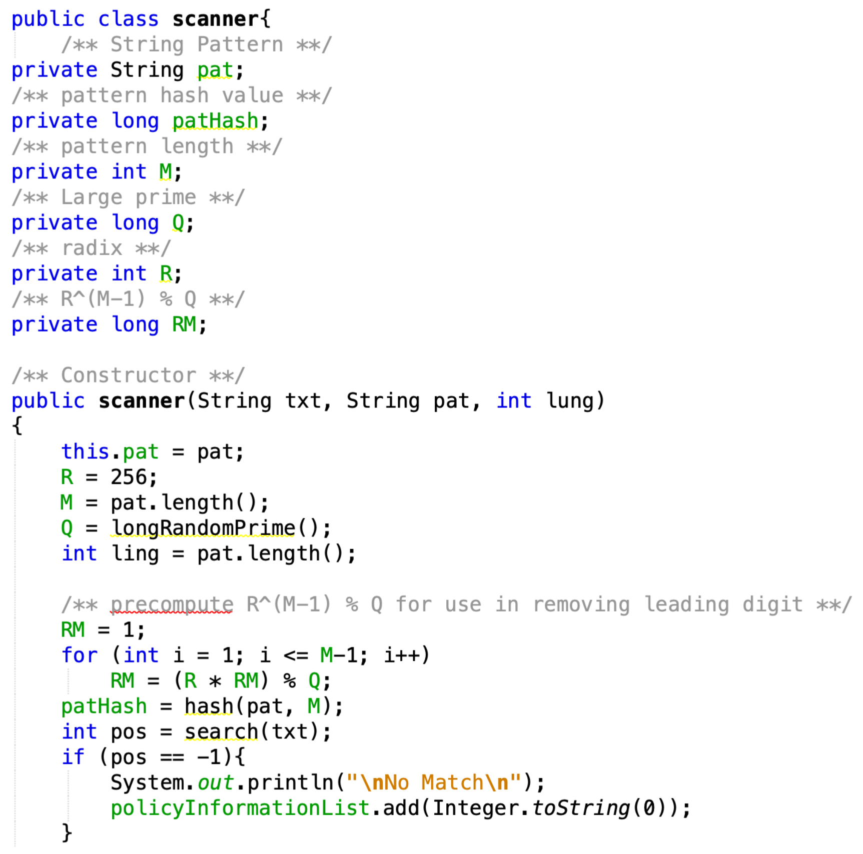
Criterion C - Development

Introduction:

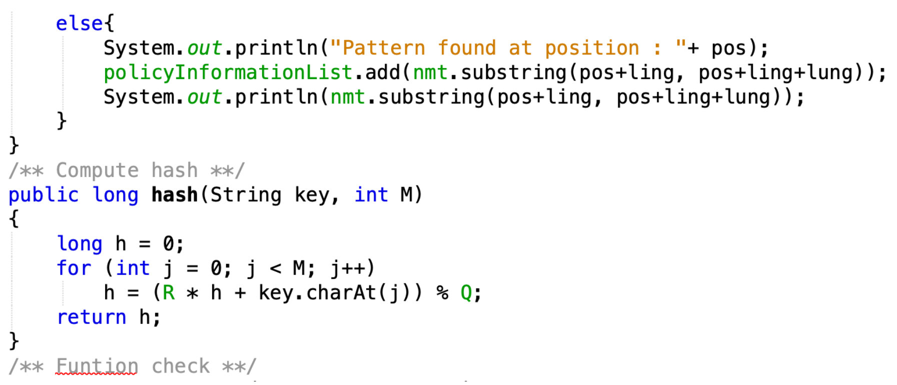
This program has two main functions as requested by my clients, SDK PLanner Group (financial company): it scans for the information of the insurance policy and input –as well as process– data into three tables. Through the use of Java's Swing tools in the Netbeans IDE and OOP (object-oriented programming), I created the GUI (graphic user interface) that the client would easily interact with.

Summary list of All Techniques:

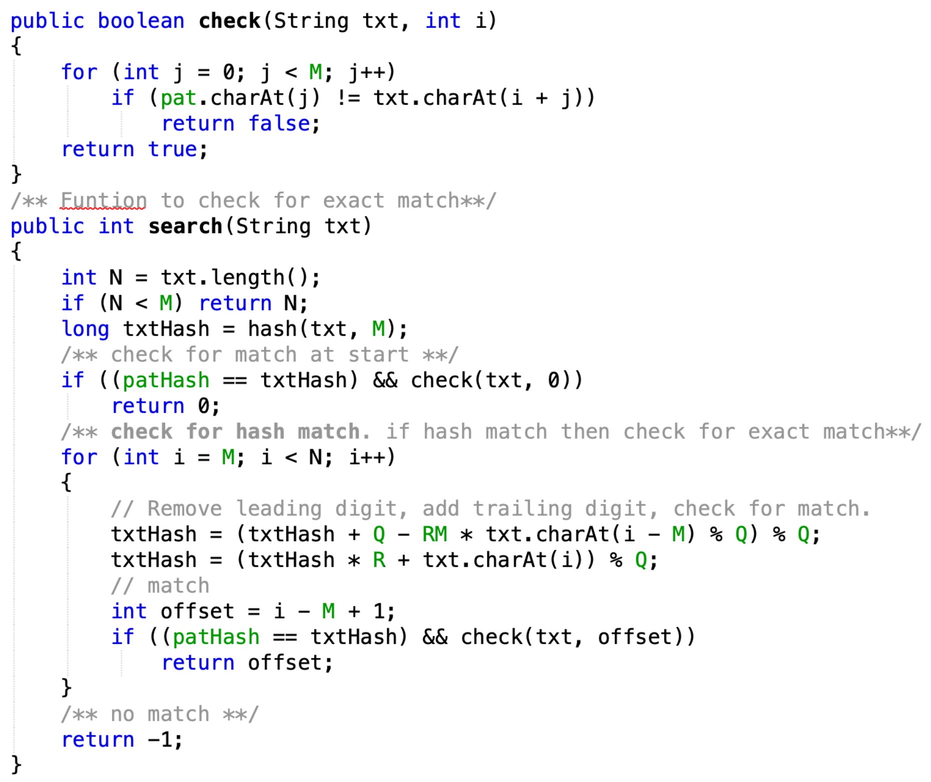
- *Searching or Scanning Class (Rabin Karp Algorithm)*
    - Reasons:
    - The text needed to be scanned is huge considering the number of characters in the policy, therefore the algorithm must not go through every string like the normal sort and search algorithm in order to prevent run time error. Therefore, I use the Rabin Karp Algorithm which utilizes the hash function where characters are being weighed then they are matched together.



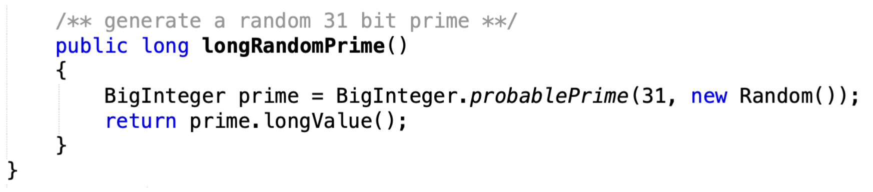
```
public class scanner{
    /** String Pattern **/
private String pat;
/** pattern hash value **/
private long patHash;
/** pattern length **/
private int M;
/** Large prime **/
private long Q;
/** radix **/
private int R;
/** R^(M-1) % Q **/
private long RM;

/** Constructor **/
public scanner(String txt, String pat, int lung)
{
    this.pat = pat;
    R = 256;
    M = pat.length();
    Q = longRandomPrime();
    int ling = pat.length();

    /** precompute R^(M-1) % Q for use in removing leading digit **/
    RM = 1;
    for (int i = 1; i <= M-1; i++)
        RM = (R * RM) % Q;
    patHash = hash(pat, M);
    int pos = search(txt);
    if (pos == -1){
        System.out.println("\nNo Match\n");
        policyInformationList.add(Integer.toString(0));
    }
```



```java
        else{
            System.out.println("Pattern found at position : "+ pos);
            policyInformationList.add(nmt.substring(pos+ling, pos+ling+lung));
            System.out.println(nmt.substring(pos+ling, pos+ling+lung));
        }
    }
    /** Compute hash **/
    public long hash(String key, int M)
    {
        long h = 0;
        for (int j = 0; j < M; j++)
            h = (R * h + key.charAt(j)) % Q;
        return h;
    }
    /** Funtion check **/
```



```java
public boolean check(String txt, int i)
{
    for (int j = 0; j < M; j++)
        if (pat.charAt(j) != txt.charAt(i + j))
            return false;
    return true;
}
/** Funtion to check for exact match**/
public int search(String txt)
{
    int N = txt.length();
    if (N < M) return N;
    long txtHash = hash(txt, M);
    /** check for match at start **/
    if ((patHash == txtHash) && check(txt, 0))
        return 0;
    /** check for hash match. if hash match then check for exact match**/
    for (int i = M; i < N; i++)
    {
        // Remove leading digit, add trailing digit, check for match.
        txtHash = (txtHash + Q - RM * txt.charAt(i - M) % Q) % Q;
        txtHash = (txtHash * R + txt.charAt(i)) % Q;
        // match
        int offset = i - M + 1;
        if ((patHash == txtHash) && check(txt, offset))
            return offset;
    }
    /** no match **/
    return -1;
}
```



```java
    /** generate a random 31 bit prime **/
    public long longRandomPrime()
    {
        BigInteger prime = BigInteger.probablePrime(31, new Random());
        return prime.longValue();
    }
}
```

- Functions:
- The overall algorithm is as explained in the previous paragraph; the class contains five main methods. The class takes in three parameters –text txt for scanning, key pat, and length of the information– and returns no arguments.The hash function finds the numerical value or weight of all characters in the text sent, then the check function does the same for the key. Then, the search function will check for matches. The longRandomPrime is used for division in calculating the hash value in the key, where prime will give them the single precision arithmetic.
- Encapsulation:
- The class is often called for finding each element, hence there are no repeated types of codes rewritten throughout the program. Hence, encapsulation of this method allows reuse of code avoiding repetitions. The code is displayed below:

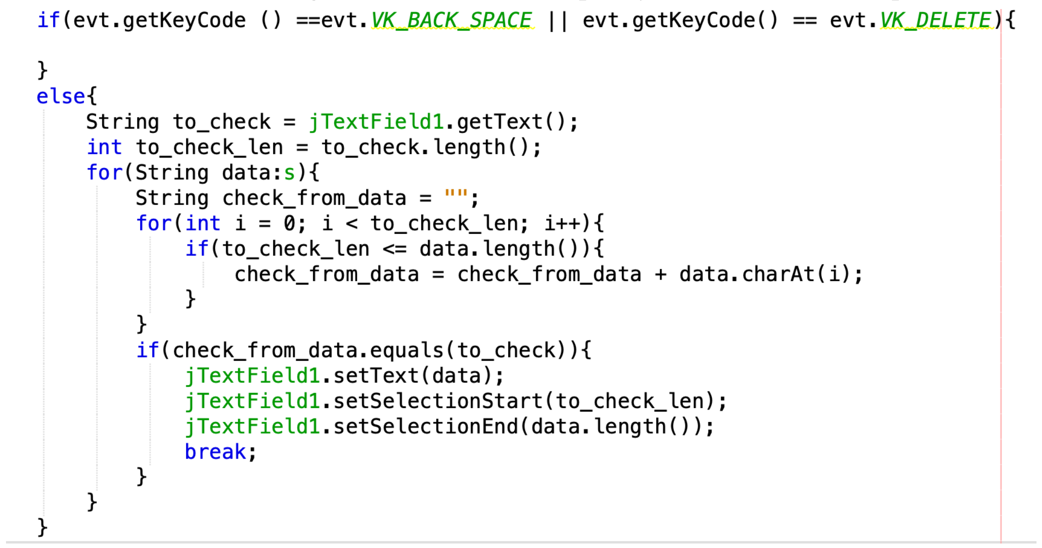
```
scanner w = new scanner(text, pattern4, 1);
String pattern5 = policyInformationList.get(4) + "/"
```

- <u>Preparation of Text for Scanning:</u>

```
nmt = addPolicy.getText().replaceAll("\\s", "");
```

- The text pasted can have many spaces which would interrupt the hash function as the spaces between keys or characters are not consistent when a pdf scanned document is turned into .txt text. Therefore, all spaces are to be replaced by "" using the method .replaceAll.

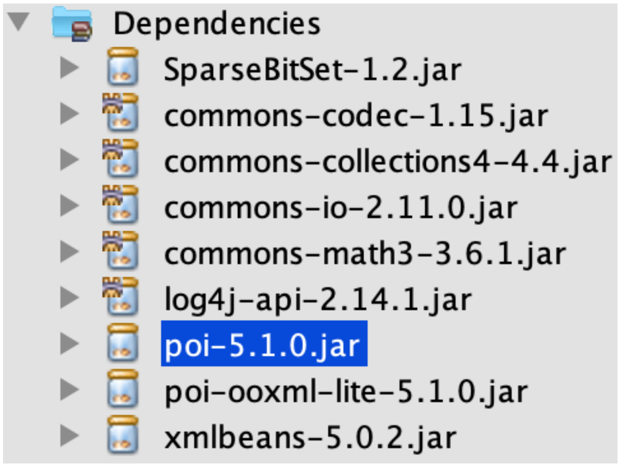
- *Nested Loops for Autofill Text:*
    - <u>Reasons:</u>
    - The scanning may not be accurate as there may be glitches or difference in information length, therefore, the scanning data may not be accurate. The Autofill will correct the basic things such as the name of policy where the common pattern is normal.



```
if(evt.getKeyCode () ==evt.VK_BACK_SPACE || evt.getKeyCode() == evt.VK_DELETE){

}
else{
    String to_check = jTextField1.getText();
    int to_check_len = to_check.length();
    for(String data:s){
        String check_from_data = "";
        for(int i = 0; i < to_check_len; i++){
            if(to_check_len <= data.length()){
                check_from_data = check_from_data + data.charAt(i);
            }
        }
        if(check_from_data.equals(to_check)){
            jTextField1.setText(data);
            jTextField1.setSelectionStart(to_check_len);
            jTextField1.setSelectionEnd(data.length());
            break;
        }
    }
}
```

- <u>Function:</u>
- The if conditions check whether or not the client is deleting what they wrote and thence the less possibilities that the key is what they are looking for. Otherwise, the nested loop will run through the data structure tree s (which will be mentioned later) where data input will be checked and the possible keys will come up after a certain amount of chars matched.

- *Export to Excel*
- <u>Libraries:</u>
    - <u>Reasons:</u>
    - The apache POI library needs to be downloaded and imported first.



```
▼ Dependencies
    ▶ SparseBitSet-1.2.jar
    ▶ commons-codec-1.15.jar
    ▶ commons-collections4-4.4.jar
    ▶ commons-io-2.11.0.jar
    ▶ commons-math3-3.6.1.jar
    ▶ log4j-api-2.14.1.jar
    ▶ poi-5.1.0.jar
    ▶ poi-ooxml-lite-5.1.0.jar
    ▶ xmlbeans-5.0.2.jar
```

- Code:
    - Reasons:
    - The client company usually formats the information into excel then organizes it to look more customer friendly for the presentation to the customers. Therefore, exporting to excel may allow client company to work in a more familiar application as well.

```java
public void openFile(String file){
    try{
        File path = new File(file);
        Desktop.getDesktop().open(path);
    }catch(IOException ioe){
        System.out.println(ioe);
    }
}
```

```java
try{
    JFileChooser jFileChooser = new JFileChooser();
    jFileChooser.showSaveDialog(this);
    File saveFile = jFileChooser.getSelectedFile();

    if(saveFile != null){
        saveFile = new File(saveFile.toString()+".xlsx");
        HSSFWorkbook wb = new HSSFWorkbook();
        HSSFSheet sheet = wb.createSheet("customer");

        HSSFRow rowCol = sheet.createRow(0);
        for(int i=0;i<jTable1.getColumnCount();i++){
            Cell cell = rowCol.createCell(i);
            cell.setCellValue(jTable1.getColumnName(i));
        }

        for(int j=0;j<jTable1.getRowCount();j++){
            HSSFRow row = sheet.createRow(j+1);
            for(int k=0;k<jTable1.getColumnCount();k++){
                Cell cell = row.createCell(k);
                if(jTable1.getValueAt(j, k)!=null){
                    cell.setCellValue(jTable1.getValueAt(j, k).toString());
                }
            }
        }
```

```
            FileOutputStream out = new FileOutputStream(new File(saveFile.toString()));
            wb.write(out);
            wb.close();
            out.close();
            openFile(saveFile.toString());
        }else{
            JOptionPane.showMessageDialog(null,"Error al generar archivo");
        }
    }catch(FileNotFoundException e){
        System.out.println(e);
    }catch(IOException io){
        System.out.println(io);
    }
}
```

- <u>Functions:</u>
- The main functions are to create new files and new documents in excel and filling in the information for the table. The try function allows the block of codes to be tested for any errors. The loop function is extracting or scanning through the table then transferring it into the new .xlsx files.

- *Method Call of Current Date:*
    - <u>Reasons:</u>
    - The calculations of paid premium and age as well as other benefits from the policy requires the current date.

```
LocalDate current_date = LocalDate.now();
int current_Year = current_date.getYear() + 543;
```

- <u>Functions:</u>
- The class LocalDate imported contains the method getYear() and the function is called as shown below. This is the property of Encapsulation where complicated class functions are imported instead of written all over the program.

```
import java.time.LocalDate;
```

```
public int getYear() {
    return year;
}
```

Abstract Data Structure:

- *Trees Storing Possible Policy Name:*
    - <u>Reason:</u>
    - The possible names are recorded in trees in order to make searching for appropriate names in nested loops for autofill text faster.

```
s = new TreeSet<String>();
s.add("บำนาญ");
s.add("ตลอดชีพ");
s.add("ออมทรัพย์");
```

- <u>Function:</u>
- The tree organizes text into order for search of keywords in the autofill text function when called.

- *LinkedLists:*
    - <u>Reason:</u>
    - The information is categorized into different linkedlist. The different types are for different tables.

```
public LinkedList<String> policyInformationList = new LinkedList();
public LinkedList<String> summaryList = new LinkedList();
public LinkedList<String> healthList = new LinkedList();
```

- Functions:
- The LinkedList contains nodes where each payload will contain the information scanned from the policy. The actions mostly done are get() and .add() in order to add information or call for that specific information to be processed.

```
summaryList.add(jTextField1.getText());
summaryList.get(1)
```

```
 for (int i = 0; i < 5; i++){
     total += Integer.parseInt(healthList.get(i));
 }
```

- Function:
- This loop function gets access to all elements in the health linked list, therefore, adding all the health benefits for policy analysis. Further, as the elements in linked list are strings, the information must be converted from string to int using the Integer.parseInt() method.

- *Array:*
  - Reason:
  - A row of table can be compared to one array where each object will be in each column.

```
model.addRow(new Object[]{name1, healthList.get(0), healthList.get(1),healthList.get(2),
    healthList.get(3),healthList.get(4),healthList.get(5), Integer.toString(until)});
```

- Functions:
- New object type array is created where each object may not have the same Java primitive type (String or int). The action .addRow is a method from another imported class; the method will add this array to the table as a new role.

Additional Tools:

- *Import Class:*
  - Reasons: There are many tasks or parts of this program that requires specified function (methods), therefore each import class performs specific tasks.

```
import javax.swing.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Random;
import java.math.BigInteger;
import java.util.LinkedList;
import java.util.*;
import java.time.LocalDate;
import java.awt.Desktop;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
```

- Functions:
- There are various import classes that are user defined objects where each specific method for each class of each type of object will be used for specific functions. The imported class also allows encapsulation since there are several functions in each class of each object that could be called several times. For instance, the filling table method.

```
DefaultTableModel model = (DefaultTableModel)jTable1.getModel();
```

- This initializes the model as defaultTableModel type where it casts jTable into this type and thus could do actions on it like adding rows in the code below.

```
model.addRow(new Object[]{column1, column2, column3, column4,
    column5, column6, column7});
```

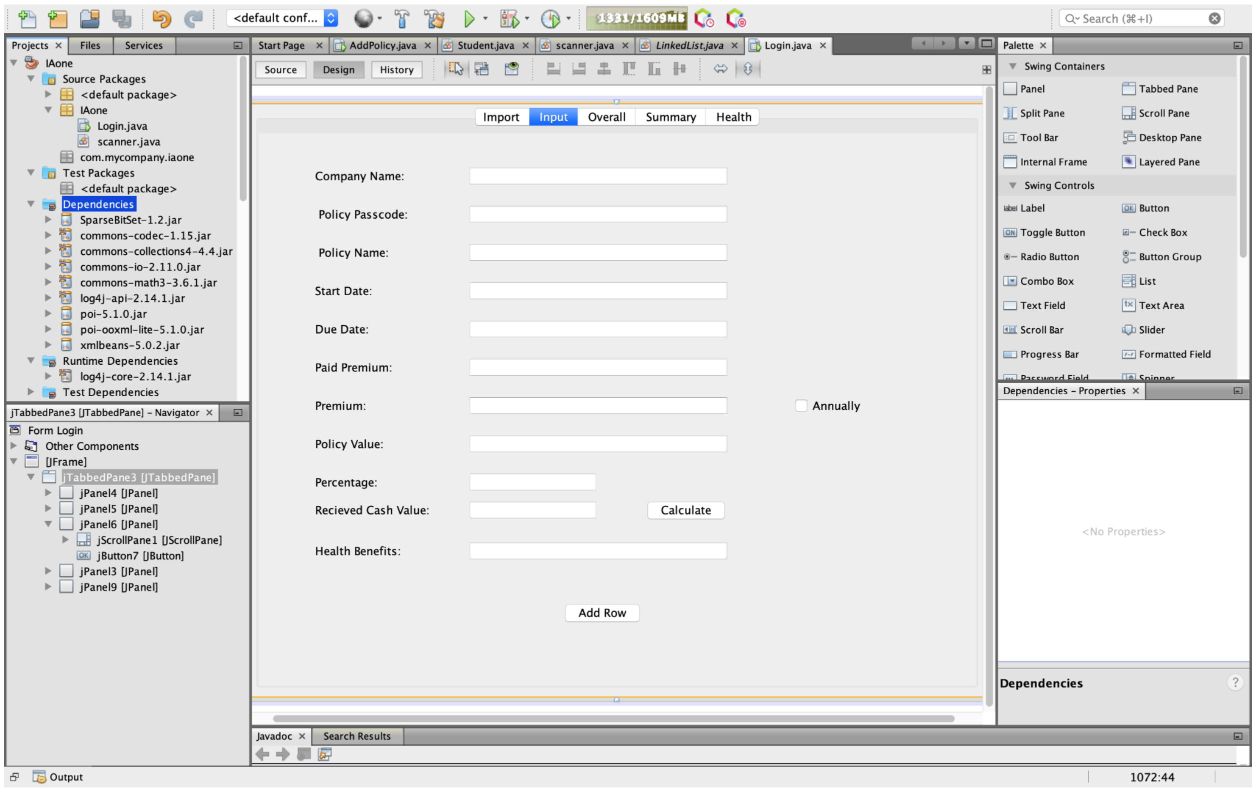
- Send Table to Excel:

```
FileOutputStream out = new FileOutputStream(new File(saveFile.toString()));
wb.write(out);
wb.close();
out.close();
openFile(saveFile.toString());
```

- This uses the method from org.apache.poi.hssf.usermodel.HSSFSheet; which has a specified function that is not primarily from Netbeans or Java. The mechanism of imported classes are all similar to previously mentioned .getYear method.

Software Tools Used:

The program is highly suitable for any types of clients as this Integrated Development Environment (IDE) Netbeans is very feasible to use. It contains several convenient features and GUI tools helping with several complicated tasks hence making me work at a more efficient speed. Furthermore, it allows many dependencies which leads to several import classes that allow for various functions hence further maximizing the potentials of the program.



Word Count: 920