Criterion C: Development

Buying Crops

```
private ArrayList<Crop> shoppingCart;
private CropListHistory CLH;
public CropIDMaker id = new CropIDMaker();
```

Initializing three instance variables in the BuyCrops class. The shoppingCart is an arraylist of crops that will contain all the crops to be purchased. CLH is an instance variable of type CropListHistory which will contain the information of all the previously bought crops. The id is an instance variable of type CropIDMaker which will be used to assign unique ID numbers to each crop.

```
public void addCustomeCrop(String name, double price, double space,
String seller, int growTime, int quantity, String file)
    {
       Crop c = null;
        for(int x = 0; x<quantity; x++)
        {
            c = new Crop(name, price, space, seller, id.newCrop(),
growTime, file);
            shoppingCart.add(c);
        }

        boolean cropExists = false;
        for(int x = 0; x<CLH.getList().size(); x++)
        {
            if(CLH.getCrop(x).getName()==c.getName())
            {
                cropExists = true;
            }
        }
```

```
        if(!cropExists)
        {
            CLH.addCrop(c);
        }
    }
```

The method addCustomeCrop from the BuyCrops class checks to see whether any crops in the shoppingCart arraylist is a unique crop that has not been purchased before. It compares each Crops name to the names of Crops stored inside the CLH arraylist of crops. If there's a matching Crop name inside of CLH, then no action is performed. If they're is not Crop from CLH that matches with the crops inside of the shoppingCart, then the Crops are added to CLH.

```
    public void addCropfromCLH(int num, int quantity)
    {
        Crop c = CLH.getCrop(num);
        for(int x = 0; x<quantity; x++)
        {
            c = new Crop(c.getName(), c.price(), c.space(),
c.seller(), id.newCrop(), c.getGrowTime(), c.getFile());
            shoppingCart.add(c);
        }
    }
```

The method addCropfromCLH adds crops if a user decides to purchase a crop that they have previously purchased. It uses the parameter num to determine which crop from CLH to purchase and the quantity parameter to determine how many of the same type of crop from CLH to purchase. These Crops are added to the shoppingCart for later use.

```
    public double getPrice()
    {
        double totalPrice = 0.0;
        for(Crop c : shoppingCart)
        {
            totalPrice += c.price();
        }

        return totalPrice;
    }
```

The method getPrice from the BuyCrops returns a double that represents the total price of all the Crops in the shoppingCart arrayList. It takes each crop, uses its price method which returns the price of the crop, and adds to prices to a local variable totalPrice. totalPrice is then returned.

Crop Array

```
private Crop[][] cropArray;

    public CropArray(int i, int j)
    {
        cropArray = new Crop[i][j];
        for(int x = 0; x<cropArray.length; x++)
        {
            for(int y = 0; y<cropArray[x].length; y++)
            {
                cropArray[x][y] = null;
            }
        }
    }

    public CropArray(Crop[][] cA)
    {
        cropArray = cA;
    }
```

Creates the CropArray instance variable which will store the crops planted in a NxN. This instance variable can be initialized in one of two constructors depending on whether or not the program is being launched for the first time or not. If it is being launched for the first time, then each Crop in the matrix is initialized to null. If the program is not being launched for the first time, then the last known value of the CropArray before the program is closed is passed through this constructor which is used to initialize CropArray.

```
    public String setCrop(Crop c, int i, int j)
    {
        boolean isEnoughSpace = true;
        int spaceNeeded = (int) Math.ceil(Math.sqrt(c.space()));

        if(i<cropArray.length && j<cropArray.length)
        {
            for(int x = i; x<i+spaceNeeded; x++)
            {
                for(int y = j; y<j+spaceNeeded; y++)
                {
                    if(cropArray[i+spaceNeeded][j+spaceNeeded] !=
null)
                    {
                        isEnoughSpace = false;
                    }
                }
            }
        }
        if(isEnoughSpace)
        {
            for(int x = i; x<i+spaceNeeded; x++)
            {
                for(int y = j; y<j+spaceNeeded; y++)
                {
```

```
                    cropArray[x][y] = c;
                }
            }
            return "Crop was add";
        }
        {
            return "There is no space in that region";
        }
    }
```

This method from the CropArray class first checks to see if a Crop can be added to the ith row and jth column in the cropArray matrix. It checks to see if there are any crops occupying that position and if the crop requires multiple spots in the garden, checks to see if those extra spots are also occupied by other crops or is outside the array.

```
    public Crop getCrop(int i, int j)
    {
        return cropArray[i][j];
    }
```

Returns a crop at row i and j.

```
    public Crop removeCrop(int ID)
    {
        Crop c = null;
        for(int x = 0; x<cropArray.length; x++)
            {
                for(int y = 0; y<cropArray[x].length; y++)
                {
                    if(cropArray[x][y].getCropID()==ID)
                    {
                        c = cropArray[x][y];
                        cropArray[x][y] = null;
```

```
                    }
                }
            }
        return c;
    }
```

Returns a crop based on its unique identification number by searching through the cropArray. If that crop is found, that spot is replaced with null and the Crop is returned. If no Crop is found that ID, then null is returned.

Event

```
    private String name;
    private int EventID;
    private String description;
    private int time;
    private int day;
    private int year;
    private int month;
    private boolean isDatePassed;
    private int[] monthDays;
    private int[] leap = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
    private int[] notLeap = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31};
```

These are the instance variables for the Event class. The name, EventID, and description are used to describe important events like when crops need to be harvested. The EventID is a unique number used to represent events. Time, day, year, and month represent the date and time the event is associated with. isDatePassed will be false if the current date has not passed the event

date and true otherwise. Month days, leap and notLeap contain information to be used to make a calendar.

```
    public String getName()
    {
        return name;
    }

    public int getEventID()
    {
        return EventID;
    }

    public String getDescription()
    {
        return description;
    }

    public int getDay()
    {
        return day;
    }

    public int getMonth()
    {
        return month;
    }

    public int getYear()
    {
        return year;
    }

    public boolean getIsDatePassed()
    {
        return isDatePassed;
    }
```

Set of methods used to access and modify the instance variables.

```
private void setIsDatePassed(int i)
    {
        Calendar c = Calendar.getInstance();
        Date d = new Date();
        c.setTime(d);
        int doy =  c.get(c.DAY_OF_YEAR);
        if(i>doy)
        {
            isDatePassed = false;
        }
        else
        {
            isDatePassed = true;
        }
    }
```

Method used to initialize isDatePassed. Uses Calendar class and the DAY_OF_YEAR final variable from that class in order to obtain the amount of days that have passed since the start of the year, if the parameter, which represents the Day of the Year value for the event is greater than DAY_OF_YEAR, the method returns false. Otherwise, it returns true.

```
private void setMonthDays()
    {
        Calendar c = Calendar.getInstance();
        Date d = new Date();
        c.setTime(d);
        int y =  c.getWeekYear();
        if(y%4 == 0)
        {
```

```
            if(y%100 == 0)
            {
                if(y%400 == 0)
                {
                   monthDays = leap;
                }
                else
                {
                    monthDays = notLeap;
                }
            }
            else
            {
                monthDays = leap;
            }
        }
        else
        {
            monthDays = notLeap;
        }
    }
```

This method sets the amount of days that are in the current month. It takes whether or not there is a leap year into account. The method by which the leap years were identified is adapted from code found in the Geeks for Geeks website.[1]

[1] "Java Program to Find If a given Year Is a Leap Year." *GeeksforGeeks*, 31 Oct. 2020, www.geeksforgeeks.org/java-program-to-find-if-a-given-year-is-a-leap-year/. Accessed 15 Apr. 2021.

Data

```
    private static HashMap<String, String> signInCredentails;
    private static Crop[] values;
    private static CropArray CA;
    private static CropInventory CI;
    private static CropInventory HCI;
    private  EventList el;
    double balance;
    int gardenNum;
    CropIDMaker CIDM;
    database d = new database();
```

Creating instance variables for Data class.

```
public boolean credentailCheck(String username, String password)
    {
        if(signInCredentails.get(username) != null )
        {

if(signInCredentails.get(username).compareTo(password)==0)
            {
                return true;
            }
        }
        return false;
    }
```

Checks to see whether the username and password values are inside the signInCredentails hashmap. If the hashmap returns the same String found in the password parameter when the username parameter is passed through it, then the credailCheck method will return true. It will return false otherwise.

```
public void addNewLogin(String username, String password)
    {
```

```
        signInCredentails.put(username, password);
    }
```

Adds a new username and password to the hashmap when a new account is created.

```
    public void update()
    {
        d.setMap(this.getMap());
        d.setCropArray(this.getArray());
        d.setCI(this.getCI());
        d.setHCI(this.getHCI());
        d.setEL(this.getEL());
        d.setBalance(this.getBalance());
        d.setGardenNum(this.getGardenNum());
        d.update();

    }
```

Before the program is closed, this method is run in order to pass the instance variables from the GUI to the database class to be stored for use when the program is reopened.

LoginPage

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JTabbedPane;
import java.util.ArrayList;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
```

Packages imported for program GUI. Many of the imports are used to create the swing objects.

```
        if (e.getSource() == createAcountButton) {
            getContentPane().removeAll();
            getContentPane().add(container2);
            repaint();
            printAll(getGraphics());
        }
       //Coding Part of showPassword JCheckBox
        if (e.getSource() == showPassword) {
            if (showPassword.isSelected()) {
                passwordField.setEchoChar((char) 0);
            } else {
                passwordField.setEchoChar('*');
            }
        }
        if (e.getSource() == showPassword2) {
            if (showPassword2.isSelected()) {
                passwordField2.setEchoChar((char) 0);
                passwordField3.setEchoChar((char) 0);
            } else {
                passwordField2.setEchoChar('*');
                passwordField3.setEchoChar('*');
            }
        }
        if(e.getSource() == createAcount)
        {
            if(passwordField2.getText().equalsIgnoreCase("") ||
passwordField3.getText().equalsIgnoreCase(""))
            {
                JOptionPane.showMessageDialog(this, "Please type in a
password");
            }
            else if(userTextField2.getText().equalsIgnoreCase(""))
            {
                JOptionPane.showMessageDialog(this, "Please type in a
Username");
            }
```

```
            else
if(passwordField2.getText().equalsIgnoreCase(passwordField3.getText()
))
            {
                JOptionPane.showMessageDialog(this, "Created a Acount
Succefully. Login!");
                data.addNewLogin(userTextField2.getText(),
passwordField2.getText());
                getContentPane().removeAll();
                getContentPane().add(container1);
                repaint();
                printAll(getGraphics());
            }
            else
            {
                JOptionPane.showMessageDialog(this, "Your passwords
do not match");
            }
        }
```

A conditional found in the action performed method. It can convert the letters into “*” and back. It compares the password and confirm password text fields in order to ensure that they match. If they do, the account will be created.

```
        if(e.getSource()==Buy)
        {
            CropListHistory tempclh = purchase.getCLH();
            for(Crop C : purchase.getShoppingCart())
            {
                tempclh.addCrop(C);
            }
            purchase.setCLH(tempclh);
            for(Crop C : purchase.getShoppingCart())
```

```
            {
                data.addCI(C);
            }
            System.out.println(-purchase.getPrice());
            data.addBalance(-1*purchase.getPrice());
            purchase.buyCrops();

            this.refreshOldCropNames();
            this.refreshValues();
            getContentPane().removeAll();
            getContentPane().add(tabPane);
            repaint();
            printAll(getGraphics());
            JOptionPane.showMessageDialog(this, "Your Crops were
Purchased");
            this.refreshCropSales();
        }
```

This is a conditional within the actionPerformed method. It creates a receipt of all the products the user could purchase by adding all the Strings produced from the toStrings of the crops from a shoppingCart arrayList to a JList.

```
       if(e.getSource()==JB1)
        {
           int num = 0;
           if(!set[num])
           {
                JPanel t = new JPanel();
                t.add(new JLabel("Please make a selection:"));
                DefaultComboBoxModel model = new
DefaultComboBoxModel();
                for (Crop value : values) {
                    System.out.println(value.getName());
                    model.addElement(value.getName());
                }
```

```
                JComboBox comboBox = new JComboBox(model);
                t.add(comboBox);
                int iResult = JOptionPane.showConfirmDialog(null, t,
"Flavor", JOptionPane.OK_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE);
                switch (iResult) {
                    case JOptionPane.OK_OPTION:
                        result[num] = (String)
comboBox.getSelectedItem();
                        break;
                }
                Crop[] tempvalues = new Crop[values.length-1];
                ArrayList<Crop> tempvalues2 = new ArrayList<Crop>();
                int i = 0;
                int ii = 0;
                for(Crop c : values)
                {

                    if((c.getName()!=result[num] || ii>0) && (i>=1 ||
c.getName()!=result[num]))
                    {
                        tempvalues[i] = c;
                        tempvalues2.add(c);
                        i++;

                    }
                    else
                    {
                        crop = c;
                        ii++;
                    }

                }
                values = tempvalues;
                data.setCropArray(tempvalues);
                CropInventory CI = new CropInventory(tempvalues2);
                data.setCI(CI);
                JB1.setText(result[num]);
                set[num] = true;
```

```
                Calendar calender = Calendar.getInstance();
                System.out.println(calender.get(Calendar.DATE));
                data.getEL().addEvent(new Event(("Harvest " +
crop.getName()), crop.getCropID(), "", 0,
calender.get(Calendar.DATE), calender.get(Calendar.YEAR),
calender.get(Calendar.MONTH)+crop.getGrowTime()+1));
                values2[num] = new Event(("Harvest " +
crop.getName()), crop.getCropID(), "", 0,
calender.get(Calendar.DATE), calender.get(Calendar.YEAR),
calender.get(Calendar.MONTH)+crop.getGrowTime()+1);
                this.updateMonth();
                //data.setCropArray(data.getCropArray.setCrop())
            }
            else
            {
                Calendar calender = Calendar.getInstance();
                HarvestQuestion.setText("Crop Name: " + result[num] +
" || Harvest Date: " +
calender.get(Calendar.MONTH)+crop.getGrowTime()+1);

                HarvestCrop.remove(HarvestQuestion);
                HarvestCrop.revalidate();
                HarvestCrop.repaint();

HarvestQuestion.setVerticalAlignment(SwingConstants.BOTTOM);
                HarvestQuestion.setBounds(20, 20, 300, 30);

HarvestQuestion.setHorizontalAlignment(SwingConstants.CENTER);
                HarvestCrop.add(HarvestQuestion);
                HarvestCrop.add(Harvest);
                ImageIcon image = new
ImageIcon(cropImageText.getText());
                Image image2 = image.getImage();
                Image image3 = image2.getScaledInstance(200, 200,
java.awt.Image.SCALE_SMOOTH);
                int res = JOptionPane.showConfirmDialog(null,
HarvestCrop,  "File",
                JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.PLAIN_MESSAGE, new ImageIcon(image3));
```

```
            HarvestCrop.repaint();
            if(res==0)
            {
                //System.out.println(crop+ "''");
                data.addCropTOHCI(crop);

                set[num] = false;
                JB1.setText("Empty" + (num + 1));
                //System.out.println(values[num]);
                HarvestCrop.remove(HarvestQuestion);
                this.refreshCropSales();
                EventList El = data.getEL();
                El.removeEvent(values2[num].getEventID());
                data.setEL(El);

            }
         }
        HarvestCrop.remove(Harvest);
        this.updateMonth();
    }
```

Found in actionPerformed method. Handles all possible actions when a button from the garden array is pressed. If no Crop has been planted, a JOption pane with a JCombo box with each crop found in the inventory will be displayed prompting the user to select the crop to place on that patch of garden. If a crop has already been planted, then the user will be asked if they want to harvest the plant in another JOption pane. This option pane also uses a scaled down ImageIcon to display a picture of the crop.