Criterion C: Development
844 words

1. Data structures

1.1. Arrays

Arrays of Ingredient and Exercise Class are created to store data loaded from the data files. They are used in the main class. The data from arrays is later passed to lists.

```
Ingridient[] ing=null ;
Excercise[] exc=null ;
```

1.2. Lists

Lists are the data structure widely used across this programme due to easy manipulation of objects inside of them. They are incorporated into the graphical object JList, from which they can be easily accessed and manipulated by the user.

```
private javax.swing.JList<Object> list, tempList;
```

2. Techniques

2.1. Encapsulation

Object of classes :

a) ChooseFrame , Frame, SaveLoadFrame, ViewFrame
b) Day, Week, Meal and Exercise Plan
c) Exercise and Exercise Plan
d) Ingredient and Meal

were created so that data and methods connected to these classes could operate together.

2.2. Polymorphism, use of tags and switch case

Classes and methods were created so that different data types could be handled under a single interface. In this programme the user can create Exercise Plan as well as Meal from a single ChooseFrame, and save and load data from the same frame. To differentiate between those options constants were used and to compare between them case-switch selection was used. (see 3.2.2)

2.3. Inheritance

ChooseFrame, Frame, SaveLoadFrame and ViewFrame base upon and inherit from JFrame so that they retain similar implementations.

```
public class ChooseFrame extends javax.swing.JFrame {
```

2.4. Serialisation

Instances of objects of Meal, ExercisePlan, Day and Week class are serialised into bytes, allowing them to be stored in a file while keeping their original state.

2.5. File input and output

The data concerning ingredients and exercises is read from text file databases each time the programme starts. The user can save or load previously generated objects into files using appropriate buttons, so that they can be used after closing and reopening of the programme. The user can save a shopping list into a text file.

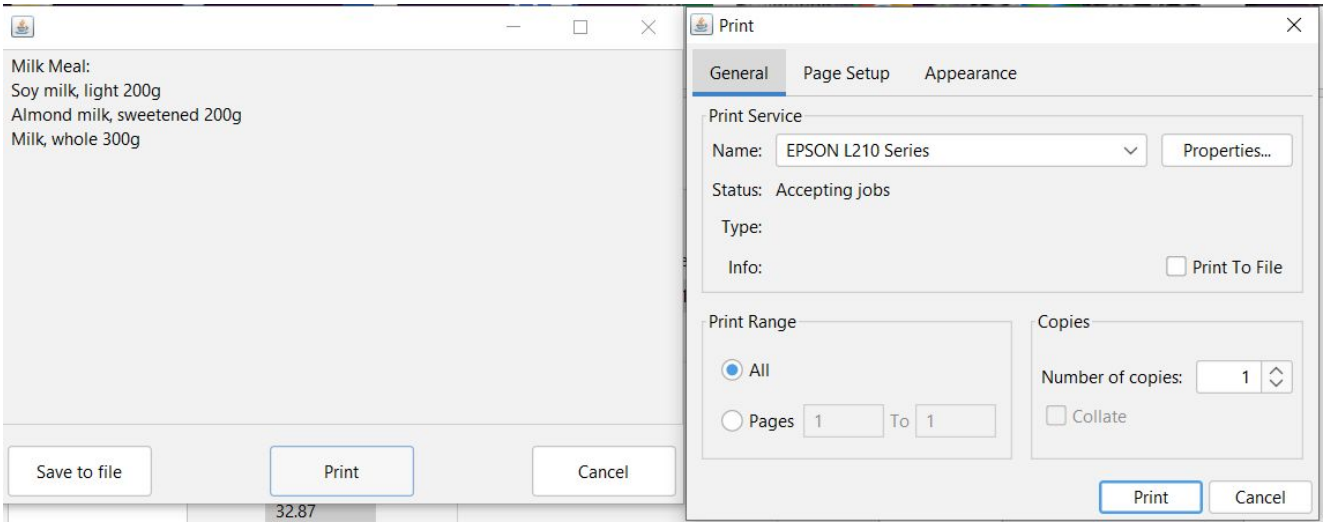
2.6. Parsing a text file

Data concerning ingredients and exercises stored in a text file is properly parsed and turned into objects that are later used to create meals and exercise plans by the user.

2.7. Printing directly from the programme

The user is able to print shopping lists directly using the programme interface, provided their printer is connected to the device on which the programme is run.

Example:



2.8. Use of specialised libraries

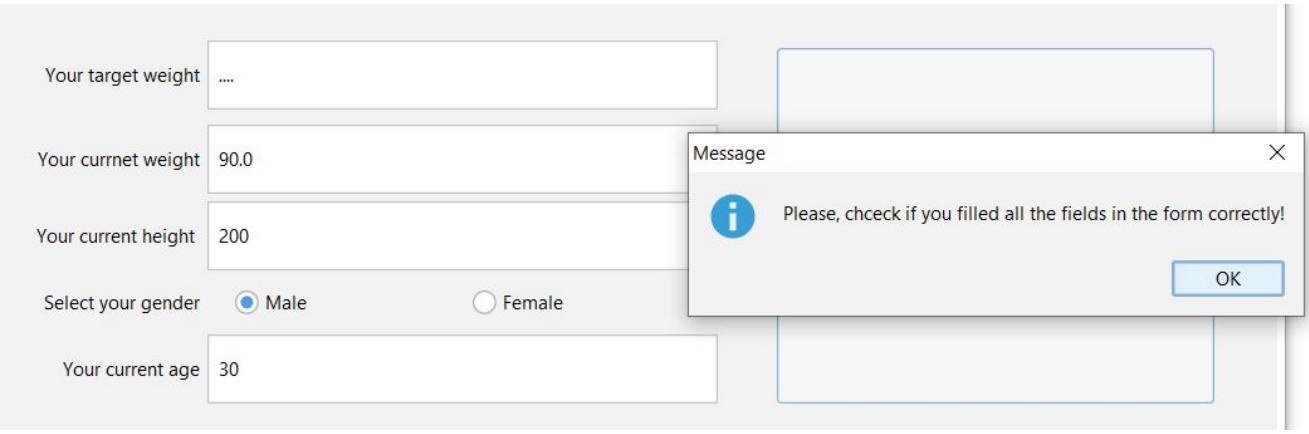
JFreeChart library was used to graphically display predicted effects of the diet and FlatLaf library was used to improve the look of the graphic interface.

Example: Chart generated in the programme using the library and FlatLaf look of the UI.

2.9 Error handling

Potential exceptions are caught, the user is informed about the error and can continue working with the programme.

Example



3. Working of the methods used

Annotated parts of the most important methods can be found below.

3.1 Initialising data

The code for initialising Ingredients, Exercises and user data is very similar. Only methods for initialising exercises will be shown here as an example. The method for initialising Ingredients differs and user data (Person) differs only in names and number of fields in the object initialised object or array of objects.

```
public static Excercise[] initialiseExcercises() {

        StringTokenizer st = null; //string tokenizer initialisation
        BufferedReader br = null; //buffere reader initialisation
        String line ;
        Excercise [] exc = null; // an array of excercise class objects is initialised
        int i=0 ;
        try {
            br = new BufferedReader(new FileReader(new File("..\\Internal\\resources\\excercises\\dataExcercises.txt"))); // feed buffered reader the text file with data
        }
        catch(FileNotFoundException fnfe) { //exception is caught if there is no data file
            JOptionPane.showMessageDialog(null,fnfe.getMessage());
        }
        try {
            if((line=br.readLine())==null){ //assigne variable "line" first line from buffered reader at the same time checking if the file is not empty
                JOptionPane.showMessageDialog(null,"Data File is empty!"); // if the data file is empty the user is notified
                System.exit(1); // termination with positive error code
            }
            else {
                int size = Integer.parseInt(line); // datafile is structured such that the first line is the number of objetcts stored in the file
                exc = new Excercise[size]; // An array is given appropriate size
            }
```

```
            while((line=br.readLine())!=null) { // iterate over each line in the file fetching the data from buffered reader to the variable "line"
                st= new StringTokenizer(line, " "); //split lines according to established format
                st.nextToken(); // first token is an index so get next token
                exc[i]=new Excercise() ; // create new object in array
                exc[i].setEnergy(Float.parseFloat(st.nextToken()));  // set the appropirate value for the field of the object
                String name="" ;
                while(st.hasMoreTokens()){ // get the name of the object
                name+=st.nextToken()+" ";
                }
                exc[i].setName(name); // set the name of the object
                i++; // increment the index of the array

            }
        }
        catch (NullPointerException | IOException npe) { //exception is caught if buffered reader points to null
            JOptionPane.showMessageDialog(null, npe.getMessage());
        }

        return Arrays.copyOf(exc, i); // return the array with data
    }
```

Each file has its defined format, in which index, name and other appropriate fields are included. The first line in the file states the number of lines below this line so the number of objects. The file is iterated over, splitting the lines into tokens, fetching the values for different fields, from which new instances of objects are created. . For example the line : “01018 3.5 bicycling bicycling, leisure, 5.5 mph” creates an instance of Exercise class with *energy: 3.5* and *name:bicycling bicycling, leisure, 5.5 mph.*

3.2 Creating Meal and Exercise Plan

Due to the technique of polymorphism and tags the frame used for creating meals and exercise plans is the same class

3.2.1 Searching algorithm in a list of objects

The algorithm finds all matching objects in terms of name, cuts from the list non all matching ones and then updates the list

```
    private void searchBarKeyPressed(java.awt.event.KeyEvent evt) {
        if(searchBar.getText().contains("Search")){ // eliminate the search text on the bar, so that the user does not have to
            searchBar.setText("");
}
        if(searched){ // If there was a previous search, get all ingredients back into the seatch list
         ((DefaultListModel) this.searchList.getModel()).clear();
             for (Object olist1 : olist) {
             ((DefaultListModel) this.searchList.getModel()).addElement(olist1);
 }
searchList.updateUI(); // update UI
searched=false; // signal no search was performed on the last key event
        }

if(evt.getExtendedKeyCode()==10){ // if user press "Enter" key starts search
             int c=0; // counter that counts the number of found elements and also the last position in list where the matching element is
             Object temp ;
             for(int i=c; i<( this.searchList.getModel()).getSize(); i++){
                 if(((DefaultListModel) this.searchList.getModel()).elementAt(i).toString().toLowerCase().startsWith(searchBar.getText().toLowerCase())){ // if element matches
                    temp= ((DefaultListModel) this.searchList.getModel()).elementAt(c);
                    ((DefaultListModel) this.searchList.getModel()).set(c,((DefaultListModel) this.searchList.getModel()).getElementAt(i));
                     ((DefaultListModel) this.searchList.getModel()).set(i,temp); // those three lines swap the matching element with the a one non mathching with the least index.
                     // This results in the matching elements grouping from index 0 to c
                 c++;
                 }

             }
             System.out.println(c);
                ((DefaultListModel) this.searchList.getModel()).removeRange(c,(this.searchList.getModel()).getSize()-1 ); // cut the list to matching elements only
              searched=true; // Singal that search has been performed
              searchList.updateUI(); //Update UI
           searchList.ensureIndexIsVisible(0); // Bring the user to the top of the list
 }

    }
```

3.2.2 Adding Ingredient/Exercise to a list

This piece of code allows the user to add ingredients/exercises to a list from which a meal/exercise plan will be created.

```
  private void addPartButtonActionPerformed(java.awt.event.ActionEvent evt) {
      short q=0;
      switch (tag){ // checks whether the programme operates on excercises or ingredients
              case Tags.MEAL:
          try{
       q=Short.parseShort(JOptionPane.showInputDialog("Input quantity of the ingirdient in grams")); // displays input window
          } catch(java.lang.NumberFormatException e){ // handles erorneous input
          JOptionPane.showMessageDialog(this, "Input a proper number", null, JOptionPane.ERROR_MESSAGE);
          return;
          }
      Ingridient ing= new Ingridient(((Ingridient)((DefaultListModel) this.searchList.getModel()).getElementAt(searchList.getSelectedIndex())));// creates new ingredient using list
      ing.setQuantity(q);// sets inputed quantity
     ((DefaultListModel) this.outputList.getModel()).addElement(ing);// adds ingredient to the list from which meal will be created
     return;
              case Tags.EXCERCISES:
          try{
       q=Short.parseShort(JOptionPane.showInputDialog("Input the duration of the excercise"));// displays input window
          } catch(java.lang.NumberFormatException e){ // handles erronerous input
          JOptionPane.showMessageDialog(this, "Input a proper number", null, JOptionPane.ERROR_MESSAGE);
          return;
          }
      Excercise exc = new Excercise(((Excercise)((DefaultListModel) this.searchList.getModel()).getElementAt(searchList.getSelectedIndex()))); // creates new excercise using list
      exc.setTime(q); //sets inputed time
      ((DefaultListModel) this.outputList.getModel()).addElement(exc); //adds to the list from which excercise plan will be created
      }
  }
```

3.2.3 Creating Meal/Exercise plan from a list

This piece of code allows the user to create a meal/exercise plan from a chosen list of ingredients/exercises. It adds it to the list in the main frame and day object.

```
private void addActionPerformed(java.awt.event.ActionEvent evt) {
   if(((DefaultListModel) this.outputList.getModel()).size()==0){// checks if list is empty
   JOptionPane.showMessageDialog(this,"Select something"); // informs user
   return;
   }
   switch (tag){ // checks whether the programme operates od ingredients or excercises
       case Tags.MEAL:
       Ingridient[] ming= new Ingridient[((DefaultListModel) this.outputList.getModel()).size()]; // fetch size of the list
    for(int i=0; i<ming.length; i++){ // put elements of the list into array
    ming[i]= (Ingridient) ((DefaultListModel) this.outputList.getModel()).getElementAt(i);
    }
   meal= new Meal(ming); // create new meal using the array
   meal.setName(JOptionPane.showInputDialog("Name your meal")); // user names the meal
  ((DefaultListModel<Object>)list.getModel()).addElement(meal); // add meal to the list of meals in the main frame
    day.addMeal(meal); // add meal to the day object
    frame.updateFields(); // update numerical fields in the main frame

  ((DefaultListModel) this.outputList.getModel()).clear(); // clear the list of chosen ingredients
   return;
       case Tags.EXCERCISES:
   Excercise[] exc=new Excercise[((DefaultListModel) this.outputList.getModel()).size()]; // fetch size of the list
   for(int i=0; i<exc.length ; i++){ // put elements of the list into array
   exc[i]=(Excercise) ((DefaultListModel) this.outputList.getModel()).getElementAt(i);
   }
   ExcercisePlan excPlan=new ExcercisePlan(exc,user);// create new excercise plan using the array and user data
   excPlan.setName(JOptionPane.showInputDialog("Name your excercise plan")); // user names excercise plan
    ((DefaultListModel<Object>)list.getModel()).addElement(excPlan); // add the plan to the list in mane frame
    day.addExcercisePlan(excPlan); // add the plan to the day object
    frame.updateFields(); // udpate numerical fields in the main frame
  ((DefaultListModel) this.outputList.getModel()).clear(); // clear the least of chosen excercises

   }
   frame.updateTips();// update tips in main frame
```

3.3 Creating the predicted effects of a diet

This piece of code creates a chart, on which predicted effects of the diet with different calorie balances are displayed.

```
clearButtonActionPerformed(null); //clear previous analysis
int avg=0;
for(int i=0;  i<period.getDays().length; i++){ // update excercise data with current user data
    for(int j=0; j<period.getDays()[i].getExcercises().length; j++){
period.getDays()[i].getExcercises()[j].updateUser(user);
    }
period.getDays()[i].recalculate(); // recalculate energy balance to account for change in user data
avg+=user.getBMR()-period.getDays()[i].getEnergyBalance();// sum energy deficits
}

avg=avg/period.getDays().length; //calculate average energy deficit
reportArea.append("Your average calorie deficit is "+avg+"\n"); // display the deficit
DefaultCategoryDataset data= new DefaultCategoryDataset() ; // create dataset
data.addValue(user.getWeight(),"Predicted weight",""+0); // add base value
double prevVal=user.getWeight(); // get previous weight value
if(!(predictEffects(0,1.0,data,"Predicted weight"))){ // check if predicted effects function runs sucessfully, if not, inform the user
reportArea.setText("");
JOptionPane.showMessageDialog(this, "You will not achieve your target weight- the diet is not compatible with your goal");
return;
}
predictEffects(0,1.5,data,"Predicted weightwith 50% more");
predictEffects(0,2.0,data,"Predicted weight with 100% more");
predictEffects(0,3.0,data,"Predicted weightwith 200% more"); //above three predict with different values of potential energy deficit
user.recalculateBMR(); //recalculte user BMR
```

The algorithm takes all imputed days of the diet, and loops them until the goal is reached. This allows for flexible plans- any number of days period. If the diet does not work, the user is informed. Otherwise the algorithm calculates future weight using calorie balance and the user's BMR.

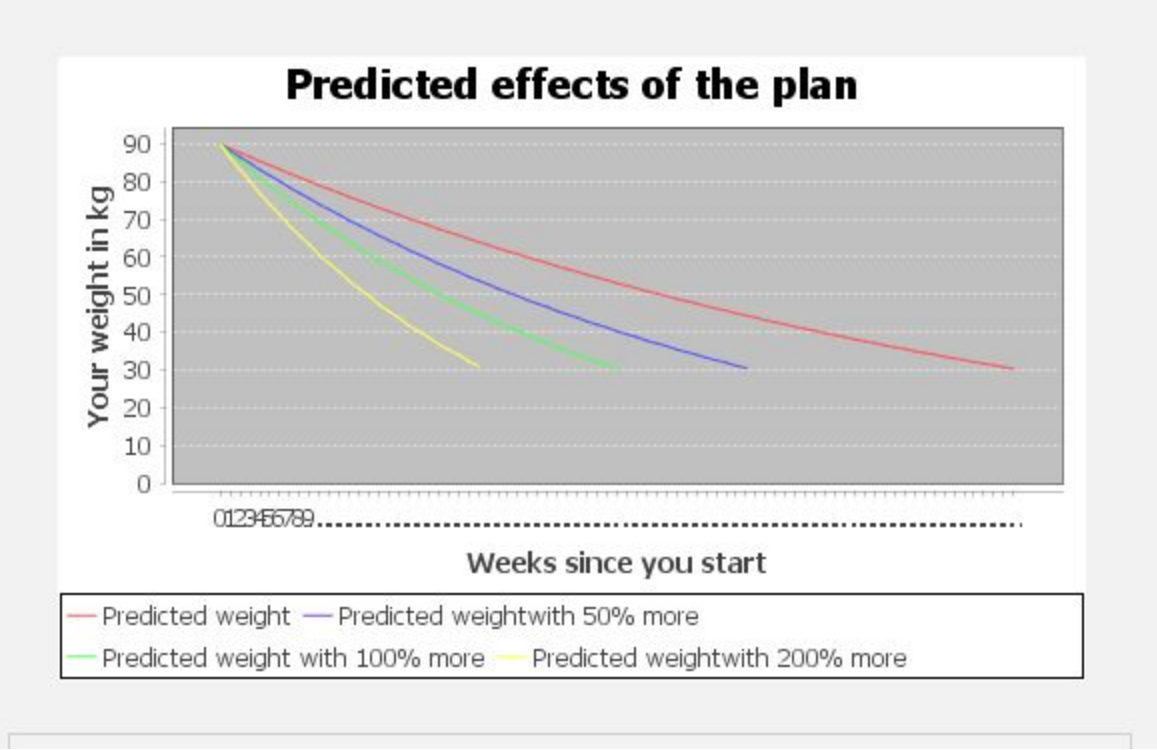
```
public boolean predictEffects(int shift, double multiplier, DefaultCategoryDataset data, String title){
    double  nextVal=user.getWeight(); //get user weight
    double difference=Math.abs(user.getWeight()-user.getTarget()); // get the difference between weight and target weight
    double nextPrevVal=nextVal-(user.getBMR()-period.getDays()[0%period.getDays().length].getEnergyBalance())*multiplier*0.0001285714; // get weight after a day of diet
    if(Math.abs(nextPrevVal-user.getTarget())>difference){ // if the difference is greater, it means the diet does the opposite what is supposed to, hence prediction is temrinated
    return false; //signal bad diet
    }

    int i=0; //count days
    int c=0; //count weeks
    if(nextVal>=user.getTarget()){ // this statement is NOT redundant. It ensures that the programme goes through only one of the loops, eliminating bugs
    while(nextVal>=user.getTarget()){ // loop until the target weight is achieved
         user.calculateBMR((float) nextVal,user.getHeight(),user.getAge(),user.isGender()); // recalculate BMR after weight loss to account for further weight loss
         nextVal-=(user.getBMR()-period.getDays()[i%period.getDays().length].getEnergyBalance())*multiplier*0.0001285714; // get weight in next day
         i++;
         if(i%7==0 && i!=0){ // if a week ends
             c++;
         data.setValue(nextVal,title,""+c); // set weekly value
         user.calculateBMR((float) nextVal,user.getHeight(),user.getAge(),user.isGender()); // recalculate BMR, just to be sure
         }


    }
    }else{
        while(nextVal<=user.getTarget()){
         user.calculateBMR((float) nextVal,user.getHeight(),user.getAge(),user.isGender()); //recalculate BMR
          nextVal-=(user.getBMR()-period.getDays()[i%period.getDays().length].getEnergyBalance())*multiplier*0.0001285714; // get weight in next day
         i++;
         if(i%7==0 && i!=0){ // if a week ends
             c++;
          data.setValue(nextVal,title,""+c);// set weekly value
         user.calculateBMR((float) nextVal,user.getHeight(),user.getAge(),user.isGender()); //recalculate BRM, just to be sure

         }
```

Taking into account BMR makes those predictions much more accurate than standard ones. This can be seen by the shape of the curves which depict longer plans , which otherwise would be simple straight lines, and that would not not represent the process of weight change accurately.



3.4 Giving tips to daily plans

To advise the user on their diet, the tips were introduced. This function checks whether the nutritional value of a one day plan is within given boundaries, which are 10% from theoretically best nutritional values.

```
public void updateTips(){
    tipsArea.setText(""); // reset tips
   float lb=(float) 0.9; // declare lower boundary
   float ub=(float) 1.1; // declare upper boundary
   if(day.getEnergy()<1200){ //check whether there is enoung calories
   tipsArea.setText("Eat more calories!"+"\n");
   }
   else{ // all conditions under here check whether the nutritional value is withing given boudnaries
            if((float)(day.getCarbohydrates()/day.getEnergy())<(((float)300/(float)2000)*(lb-(float)0.4))){
                tipsArea.append("Get more carbohydrates"+"\n");
            }else{
                if((float)(day.getCarbohydrates()/day.getEnergy())>(((float)300/(float)2000)*(ub))){
                tipsArea.append("Remove some carbohydrates"+"\n");
                    }
            }
            if((float)(day.getFat()/day.getEnergy())<(((float)67/(float)2000)*lb)){
                tipsArea.append("Get more fats"+"\n");
            }else{
                if((float)(day.getFat()/day.getEnergy())>(((float)300/(float)2000)*(ub))){
                tipsArea.append("Remove some fats"+"\n");
                    }
            }
            if((float)(day.getSugars()/day.getEnergy())<(((float)50/(float)2000)*lb)){
                 tipsArea.append("Get more sugars"+"\n");
            }else{
                if((float)(day.getSugars()/day.getEnergy())>(((float)300/(float)2000)*(ub))){
                tipsArea.append("Remove some sugars"+"\n");
                    }
            }
            if((float)(day.getProtein()/day.getEnergy())<(((float)150/(float)2000)*lb)){
                tipsArea.append("Get more protein"+"\n");
            }else{
                if((float)(day.getProtein()/day.getEnergy())>(((float)300/(float)2000)*(ub+(float)0.2))){
                tipsArea.append("Remove some protein"+"\n");
                    }
            }
            if(user.getBMR()>=day.getEnergyBalance()){
            tipsArea.append("You will lose some weight");
            } else{
                tipsArea.append("You will gain some weight");
            }
        }
    }
```

3.5 Generating shopping lists

This algorithm saves an object (in this case Meal class) to a file with a suggested directory.

```
public void savePeriodList(){ //
 JFileChooser jf= new JFileChooser("..\\Internal\\resources\\shopping\\"); // choose file or create new one
     jf.setMultiSelectionEnabled(false); //disable multi selection
     jf.showSaveDialog(this); //show save window
     File file = jf.getSelectedFile(); // get file
        try{
            FileWriter writer = new FileWriter(jf.getSelectedFile().getAbsolutePath()+".txt", true); // create writer
         for (Day dayl : period.getDays()) { //iterate over all days
             for (Meal meal : dayl.getMeals()) { // iterate over all meals in a day
                  writer.write("" + meal.getName()); // write name of the meal
                   writer.write("\r\n");   // write new line
                 for (Ingridient ingridient : meal.getIngridients()) { // iterate over all ingredients in a meal
                     writer.write("" + ingridient.toString()); // write the name of the ingredient
                     writer.write("\r\n");   // write new line
                 }
                 writer.write("\r\n");   // write new line
             }
         }

        writer.close();   // close writer
             } catch (IOException e) { // handle exception
                 JOptionPane.showMessageDialog(this, e.getMessage());
             }
}
```

3.6 Loading and saving an object from file

This piece of code handles a button in a frame that loads and saves instances of Day class. This frame is opened from the main frame.

```
   private void dayButtonActionPerformed(java.awt.event.ActionEvent evt) {
       if(tag.equals(Tags.SAVE)){ //check tag
           if(inputFrame.day.getMeals().length+inputFrame.day.getExcercises().length>0){ //check if day is not empty
       inputFrame.saveDay(); // call save function
           }else{
           JOptionPane.showMessageDialog(this, "Your day is empty"); // inform the user their day is empty
           }
       }
       if(tag.equals(Tags.LOAD)){ // check tag
       inputFrame.loadDay(); // call function that loads the object
       inputFrame.updateMealList(); // call function that updates meal list in main frame
       inputFrame.updateExcercisePlanList(); // call function that updates excercise plan list in main frame
       }
       this.update(); // call functions that updates tips, fields and UI in main frame trhough a function in this frame
       this.dispose(); // close this frame
   }
```

3.6.1 saveDay function

```
 public void saveDay(){
 JFileChooser jf= new JFileChooser("..\\Internal\\resources\\days\\"); // create and set file chooser directory
        jf.setVisible(true);
        jf.setMultiSelectionEnabled(false);  // disable multiselection
        jf.showSaveDialog(this); // show window

        try {
            ObjectOutputStream os = new ObjectOutputStream(
                new BufferedOutputStream(new FileOutputStream(jf.getSelectedFile().getAbsolutePath()))); // establish output stream
            os.writeObject(day); // write object to file
            os.close(); // close output stream
        } catch (IOException e) { // catch exception
            System.out.println(e.toString());
        }
 }
```

3.6.2 loadDay function

```
public void loadDay(){

    JFileChooser jf= new JFileChooser("..\\Internal\\resources\\days\\"); // create and set files chooser directory
       jf.setVisible(true);
       jf.setMultiSelectionEnabled(false); // disable multi selection
       jf.showOpenDialog(this); // show window

       ObjectInputStream is=null; // declare input stream
       try {
           FileInputStream fis = new FileInputStream(jf.getSelectedFile().getAbsolutePath()); // declare file input stream using selected path
           is = new ObjectInputStream( new BufferedInputStream(fis) ); // create object input stream
           ((DefaultListModel)excerciseList.getModel()).clear(); // clear excercise list in main frame
            ((DefaultListModel)mealList.getModel()).clear(); // clear meal list in main frame
           day=(Day)is.readObject(); // read day object from file
           if(day.getExcercises().length!=0){ // check whether there are any excercises
           if(!user.equals(day.getExcercises()[0].getUser())){ // check if the user data is different
               for (ExcercisePlan excercise : day.getExcercises()) { // iterate over all excercises
                   excercise.updateUser(user); // update the user of the excercise
               }
           }
           }
           is.close(); // close input stream
       } // below catches any exceptions
       catch(NullPointerException e) {
           System.out.println("NullPointerException: "+e.getMessage());
       }
       catch(ClassNotFoundException e) {
           System.out.println("ClassNotFoundException: "+e.getMessage());
       }
       catch(FileNotFoundException e) {
           System.out.println("FileNotFoundException: "+e.getMessage());
       }
       catch(EOFException e) {
           System.out.println("EOFException: "+e.getMessage());
       } catch (IOException ex) {
           Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);
       }
}
```

3.6.3 updateMealList function and updateExcercisePlanList function

```
public void updateMealList(){
    for (Meal meal : day.getMeals()) { // Add each meal from day to the list
        ((DefaultListModel<Object>)mealList.getModel()).addElement(meal);
    }
}
public void updateExcercisePlanList(){
    for (ExcercisePlan excercise : day.getExcercises()) { //Add each excercise plan from day to the list
        ((DefaultListModel<Object>)excerciseList.getModel()).addElement(excercise);
    }
}
```

3.6.4 update function

This function call three function inside the main frame

```
public void update(){
inputFrame.updateFields();
inputFrame.updateTips();
inputPanel.updateUI();
}
```

updateFields

```
public void updateFields(){
    this.energyField.setText(String.valueOf(day.getEnergy()));
    this.carbohydratesField.setText(String.valueOf(day.getCarbohydrates()));
    this.fatField.setText(String.valueOf(day.getFat()));
    this.sugarsField.setText(String.valueOf(day.getSugars()));
    this.proteinField.setText(String.valueOf(day.getProtein()));
    this.balanceField.setText(String.valueOf(day.getEnergyBalance()));
    this.timeField.setText(String.valueOf(day.getTimeSpent()));
    this.energyBurntField.setText(String.valueOf(day.getCaloriesBurnt()));
    this.calorieDeficitField.setText(String.valueOf(user.getBMR()-day.getEnergyBalance()));
}
```

Aforementioned updateTips (3.4) and generic updateUI.

3.7 Initializing FlatLaf

```
        try {
    UIManager.setLookAndFeel( new com.formdev.flatlaf.FlatLightLaf() );
} catch( UnsupportedLookAndFeelException ex ) {
    System.err.println( "Failed to initialize LaF" );
}
```

4. Transforming the xlsx file into text file using R to obtain data for ingredients.

```
library(readxl)
X2015_2016_FNDDS_At_A_Glance_FNDDS_Nutrient_Values <- read_excel("2015-2016 FNDDS At A Glance - FNDDS Nutrient Values.xlsx") ## read from excel file to table
View(X2015_2016_FNDDS_At_A_Glance_FNDDS_Nutrient_Values) ## view file
x<-X2015_2016_FNDDS_At_A_Glance_FNDDS_Nutrient_Values
signif(x[,2:6],3) ## approximate values to 3 significant figures
write.table(x,sep = "@","C:\\Users\\micha\\Documents\\NetBeansProjects\\Internal\\src\\ingridients\\dataIngridients.txt",quote = FALSE) ## write table to text file
```