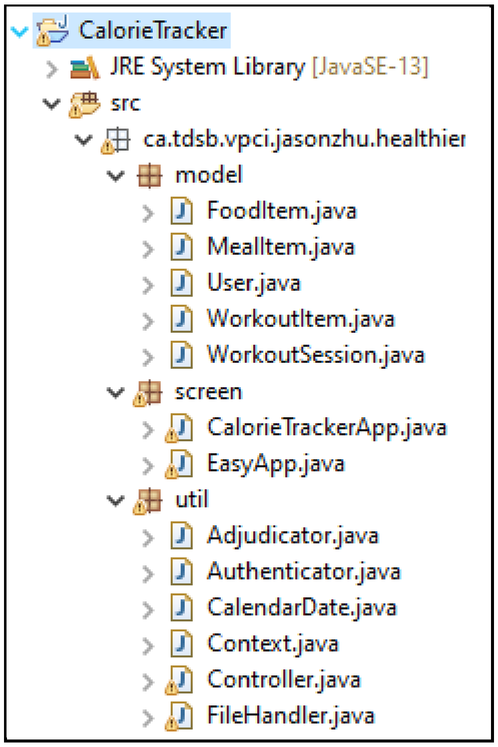
Criterion C: Development

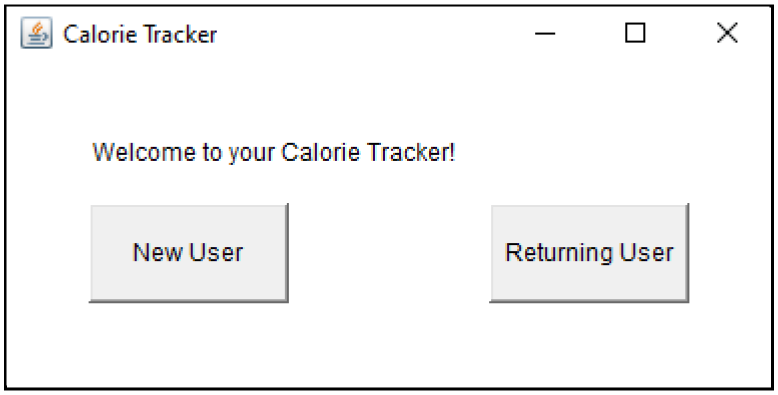
The program is written using Java. Users input their daily diets and exercise into the program, and it calculates the net calories of the user for each day. The program then gives the user information about their daily calorie intake. Various text files are used to save data of the program.

Program Functions

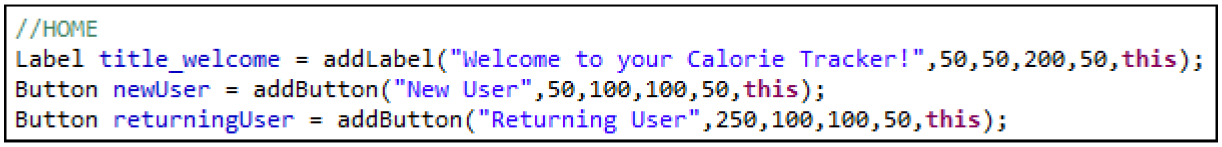


```
CalorieTracker
  JRE System Library [JavaSE-13]
  src
    ca.tdsb.vpci.jasonzhu.healthier
      model
        FoodItem.java
        MealItem.java
        User.java
        WorkoutItem.java
        WorkoutSession.java
      screen
        CalorieTrackerApp.java
        EasyApp.java
      util
        Adjudicator.java
        Authenticator.java
        CalendarDate.java
        Context.java
        Controller.java
        FileHandler.java
```

Welcome

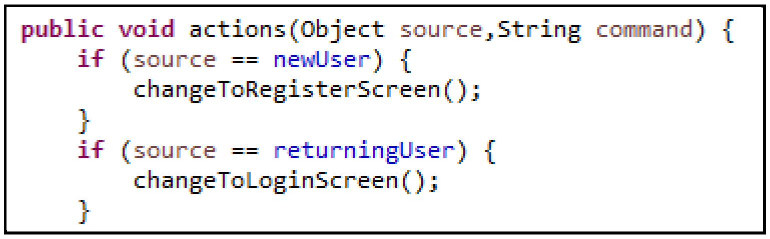


The GUI in the program is made with EasyApp.java, which was found online. For this screen, I used buttons and labels, as shown below.



```
//HOME
Label title_welcome = addLabel("Welcome to your Calorie Tracker!",50,50,200,50,this);
Button newUser = addButton("New User",50,100,100,50,this);
Button returningUser = addButton("Returning User",250,100,100,50,this);
```

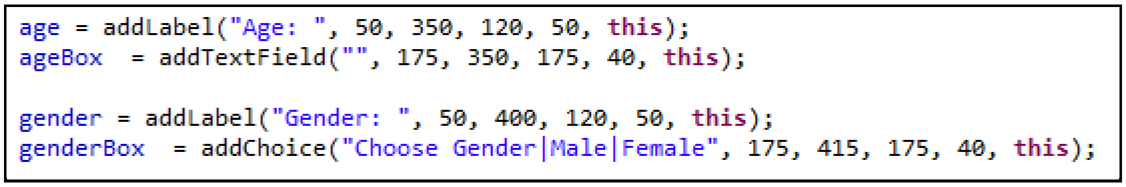
When buttons are clicked, the actions method is called, and a certain function is performed. When “New User” is clicked, the user is brought to the register screen, while “Returning User” brings the user to the log-in screen.



```
public void actions(Object source,String command) {
    if (source == newUser) {
        changeToRegisterScreen();
    }
    if (source == returningUser) {
        changeToLoginScreen();
    }
```

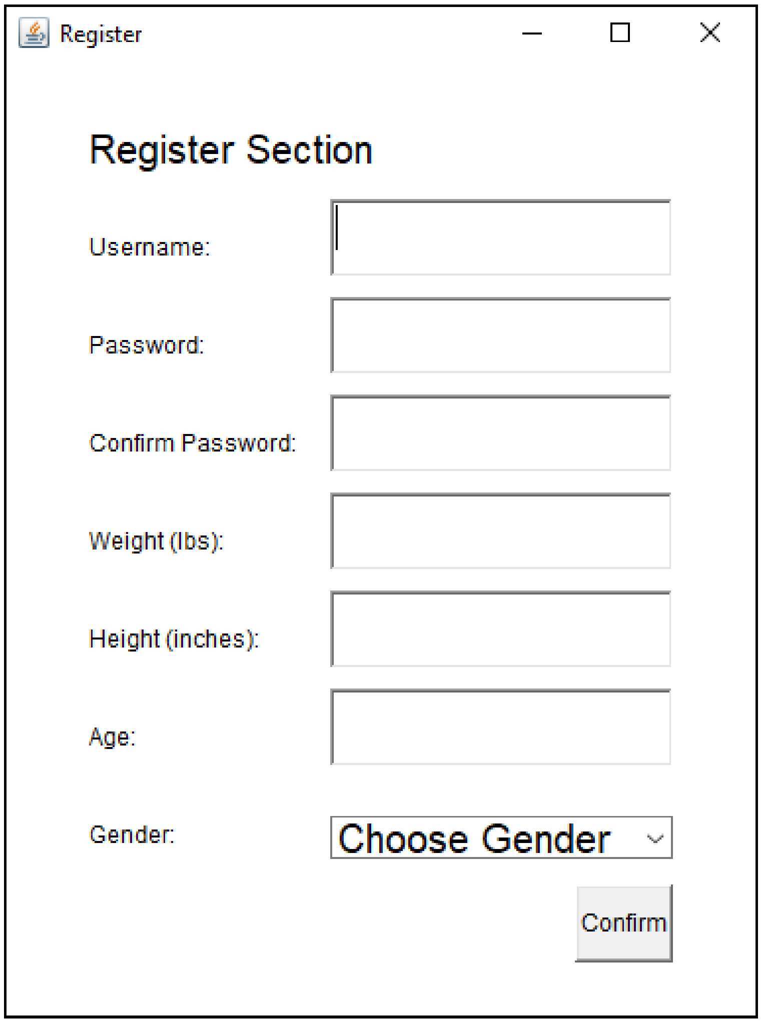
Register

The user enters each of the following parameters to register. Textfields (user input) and choices (drop down menu) are additional features from EasyApp that are used in the register screen.

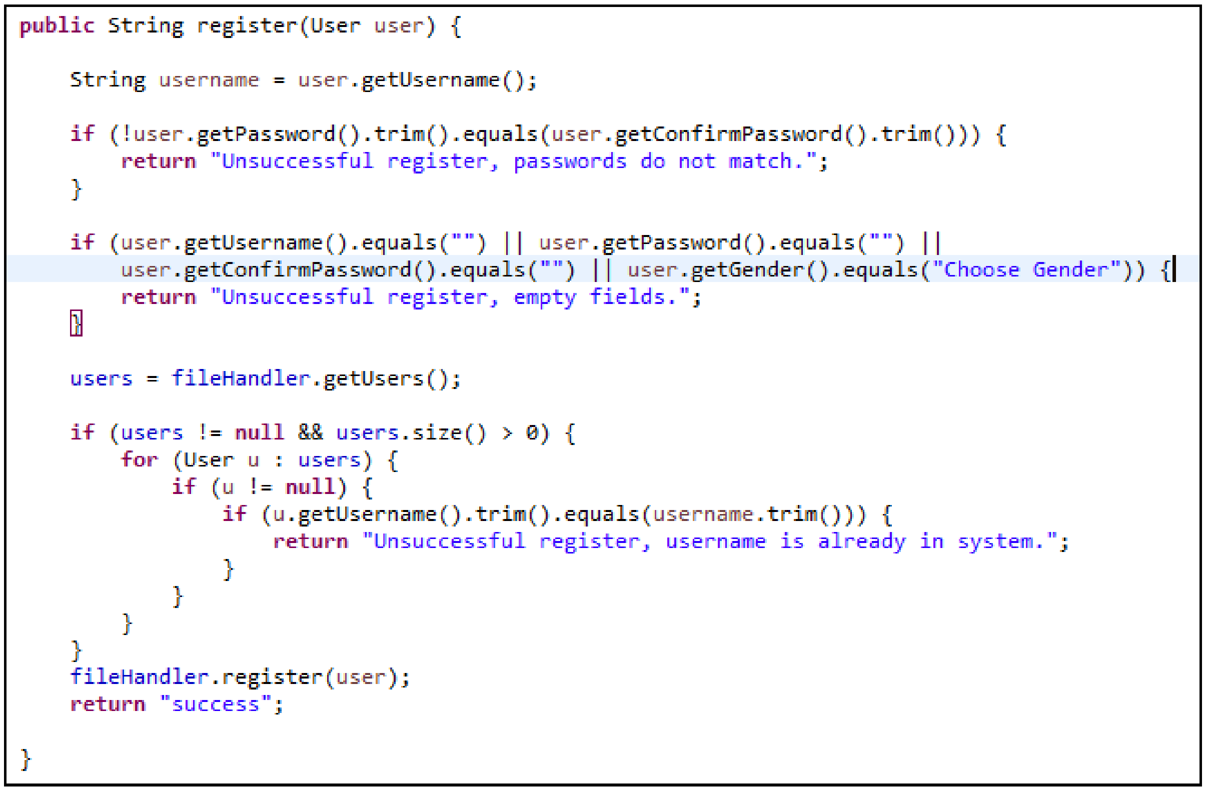


```
age = addLabel("Age: ", 50, 350, 120, 50, this);
ageBox  = addTextField("", 175, 350, 175, 40, this);

gender = addLabel("Gender: ", 50, 400, 120, 50, this);
genderBox  = addChoice("Choose Gender|Male|Female", 175, 415, 175, 40, this);
```

Here is the register screen:

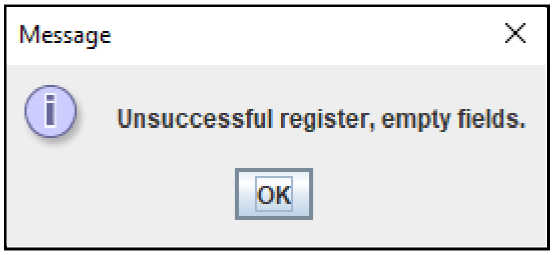


When the “Confirm” button is clicked, the following method is called from the Authenticator class which checks if each parameter is valid to be passed into a user object.



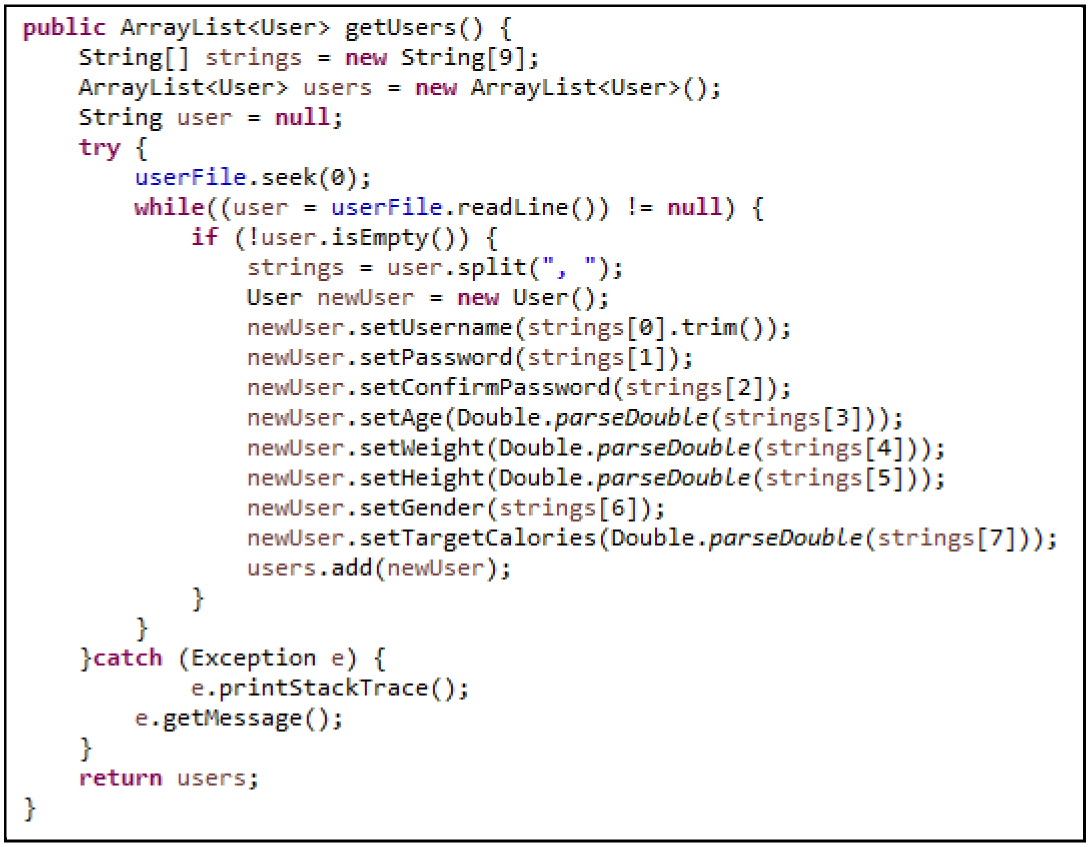
```
public String register(User user) {

    String username = user.getUsername();

    if (!user.getPassword().trim().equals(user.getConfirmPassword().trim())) {
        return "Unsuccessful register, passwords do not match.";
    }

    if (user.getUsername().equals("") || user.getPassword().equals("") ||
        user.getConfirmPassword().equals("") || user.getGender().equals("Choose Gender")) {
        return "Unsuccessful register, empty fields.";
    }

    users = fileHandler.getUsers();

    if (users != null && users.size() > 0) {
        for (User u : users) {
            if (u != null) {
                if (u.getUsername().trim().equals(username.trim())) {
                    return "Unsuccessful register, username is already in system.";
                }
            }
        }
    }
    fileHandler.register(user);
    return "success";

}
```

If certain conditions are not reached, the method returns an error message describing the cause of the error. For example, if a string is entered for some fields, this message will be displayed:



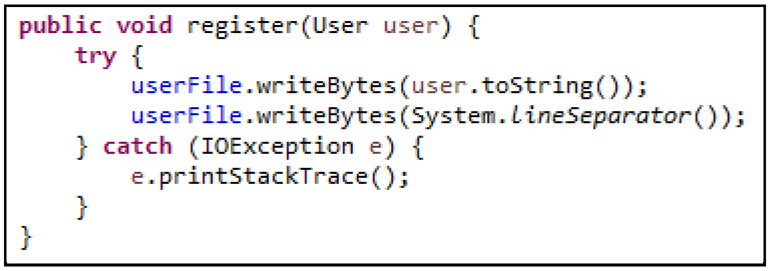
Unsuccessful register, empty fields.

To find if the username was already taken or not, another method is called in the FileHandler class to retrieve all the users in an ArrayList that are already registered in the text file, users.txt. The program uses linear search to find if the username was taken or not.



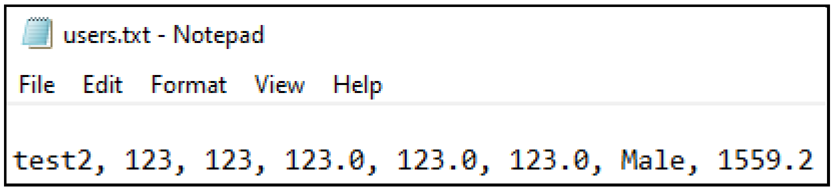
```java
public ArrayList<User> getUsers() {
    String[] strings = new String[9];
    ArrayList<User> users = new ArrayList<User>();
    String user = null;
    try {
        userFile.seek(0);
        while((user = userFile.readLine()) != null) {
            if (!user.isEmpty()) {
                strings = user.split(", ");
                User newUser = new User();
                newUser.setUsername(strings[0].trim());
                newUser.setPassword(strings[1]);
                newUser.setConfirmPassword(strings[2]);
                newUser.setAge(Double.parseDouble(strings[3]));
                newUser.setWeight(Double.parseDouble(strings[4]));
                newUser.setHeight(Double.parseDouble(strings[5]));
                newUser.setGender(strings[6]);
                newUser.setTargetCalories(Double.parseDouble(strings[7]));
                users.add(newUser);
            }
        }
    }catch (Exception e) {
            e.printStackTrace();
        e.getMessage();
    }
    return users;
}
```

If the parameters are all correct and the username is not taken, another method in the FileHandler class is called to write the user's information in a text file, users.txt.



```java
public void register(User user) {
    try {
        userFile.writeBytes(user.toString());
        userFile.writeBytes(System.lineSeparator());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

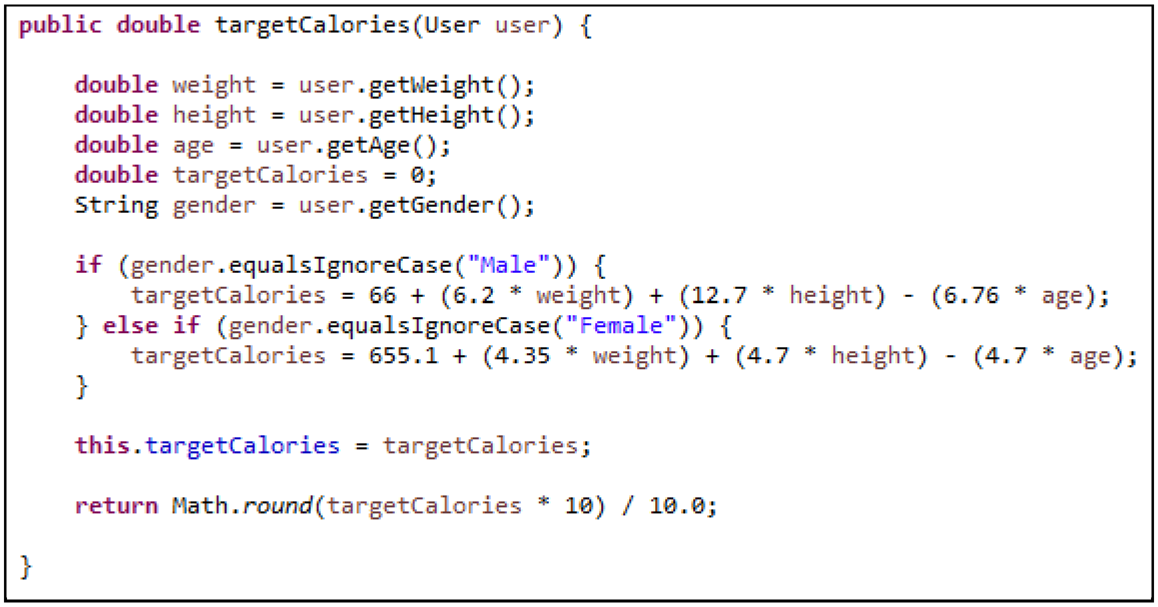
When written, the information in the file will look like this:



```
test2, 123, 123, 123.0, 123.0, 123.0, Male, 1559.2
```

(Username, password, confirm password, weight, height, age, gender, calories burned at rest)

When a user is registered, the Adjudicator class then calculates the calories the user burns at rest using a formula from the internet.

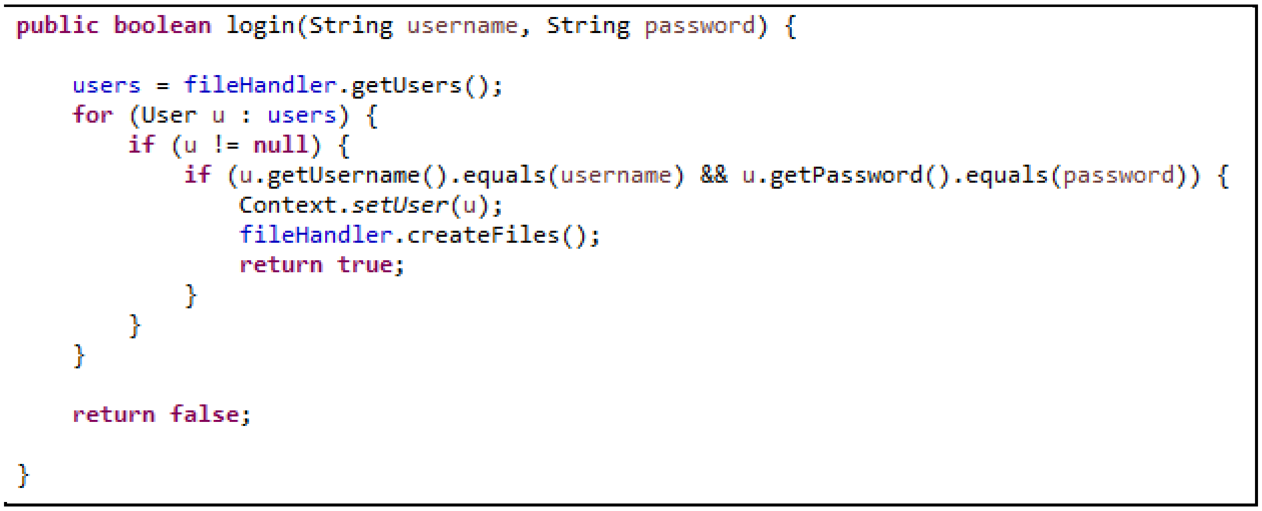


```java
public double targetCalories(User user) {

    double weight = user.getWeight();
    double height = user.getHeight();
    double age = user.getAge();
    double targetCalories = 0;
    String gender = user.getGender();

    if (gender.equalsIgnoreCase("Male")) {
        targetCalories = 66 + (6.2 * weight) + (12.7 * height) - (6.76 * age);
    } else if (gender.equalsIgnoreCase("Female")) {
        targetCalories = 655.1 + (4.35 * weight) + (4.7 * height) - (4.7 * age);
    }

    this.targetCalories = targetCalories;

    return Math.round(targetCalories * 10) / 10.0;

}
```

The formula uses weight, height, age, and gender, which were entered by the user in registration.

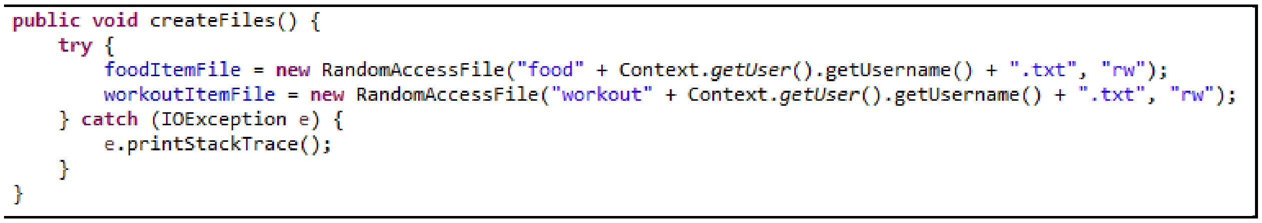
Log-In

If “Returning User” is clicked from the welcome screen, or the user is finished registering, they will be taken to the Log-In screen.



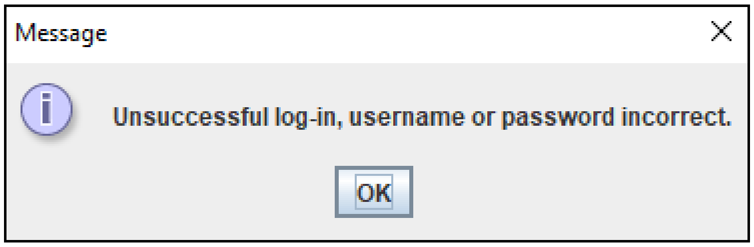
```java
public boolean login(String username, String password) {

    users = fileHandler.getUsers();
    for (User u : users) {
        if (u != null) {
            if (u.getUsername().equals(username) && u.getPassword().equals(password)) {
                Context.setUser(u);
                fileHandler.createFiles();
                return true;
            }
        }
    }

    return false;

}
```

The Log-In method in the Authenticator class is similar to the register method because it also searches the users retrieved from the text file and sees if the username and password match. If the credentials are correct, the user will be allowed into the program, and text files for the food and workouts of the user will be created. The Context class is a class to store static values, such as which user is logged in at the current moment.



```
public void createFiles() {
    try {
        foodItemFile = new RandomAccessFile("food" + Context.getUser().getUsername() + ".txt", "rw");
        workoutItemFile = new RandomAccessFile("workout" + Context.getUser().getUsername() + ".txt", "rw");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

If the credentials are wrong, an error message will be displayed:

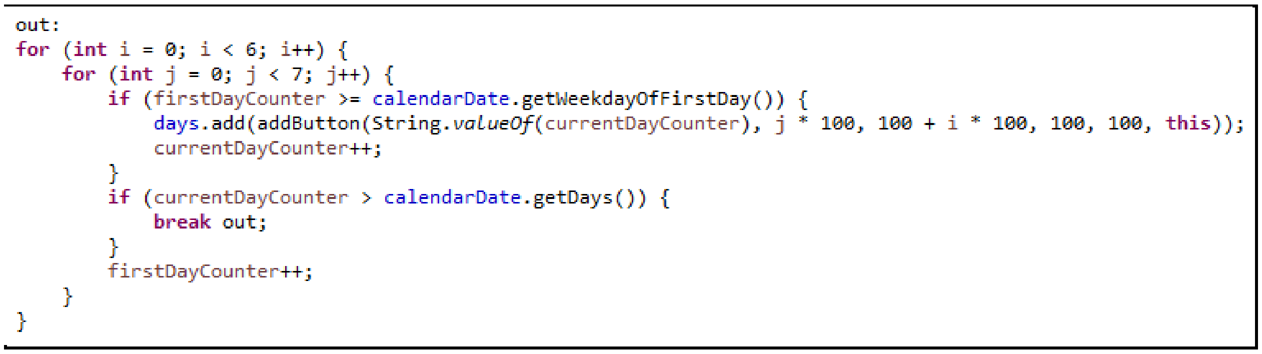


Message

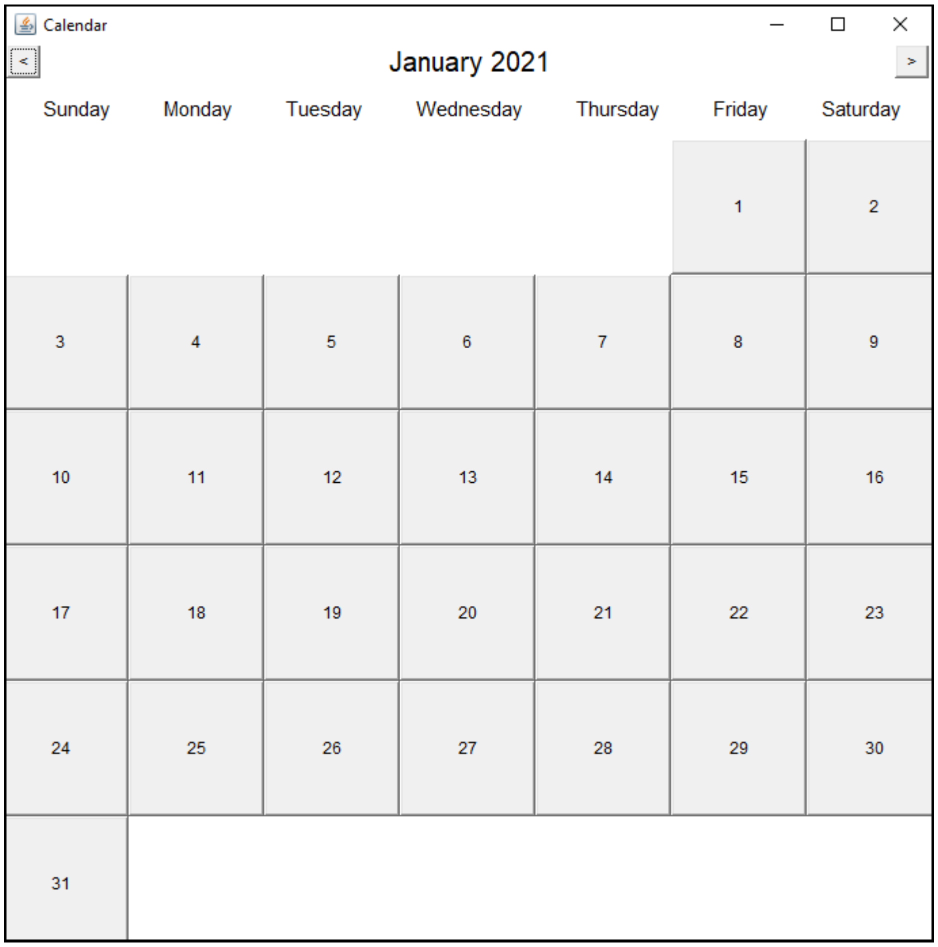
Unsuccessful log-in, username or password incorrect.

Calendar

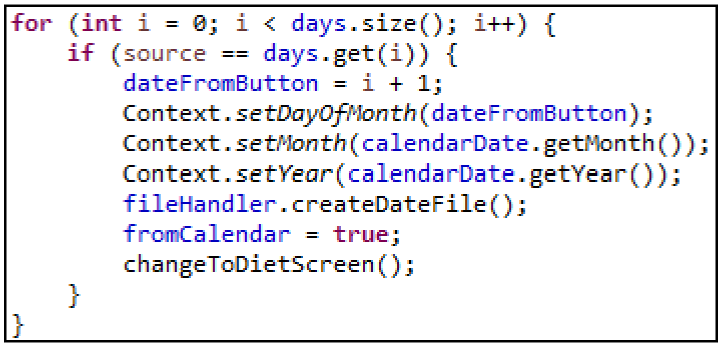
Once the user successfully logs in, they will be taken to the calendar. Each day is a button which can be clicked to access the foods/workouts of the specific day, and they were printed by using an ArrayList and for loop. The program finds out what weekday the month starts on, and then begins printing the buttons from that weekday.



```
out:
for (int i = 0; i < 6; i++) {
    for (int j = 0; j < 7; j++) {
        if (firstDayCounter >= calendarDate.getWeekdayOfFirstDay()) {
            days.add(addButton(String.valueOf(currentDayCounter), j * 100, 100 + i * 100, 100, 100, this));
            currentDayCounter++;
        }
        if (currentDayCounter > calendarDate.getDays()) {
            break out;
        }
        firstDayCounter++;
    }
}
```



When the day button is clicked, the program takes the day shown on the button and updates the date in the Context class. Files for the foods and workouts of the specific days are also created. The program then goes to the diet screen.



```
for (int i = 0; i < days.size(); i++) {
    if (source == days.get(i)) {
        dateFromButton = i + 1;
        Context.setDayOfMonth(dateFromButton);
        Context.setMonth(calendarDate.getMonth());
        Context.setYear(calendarDate.getYear());
        fileHandler.createDateFile();
        fromCalendar = true;
        changeToDietScreen();
    }
}
```

Here are methods in the CalendarDate class used to find dates.

```
public int getYear() {
    return cal.get(Calendar.YEAR);
}

public int getTodaysDate() {
    cal.setTime(date);
    return cal.get(Calendar.DAY_OF_MONTH);
}

public int getWeekdayOfFirstDay() {
    cal.set(Calendar.DAY_OF_MONTH, 1);
    date = cal.getTime();
    return cal.get(Calendar.DAY_OF_WEEK);
}
```

On the top corners of the screen, it is seen that there are two buttons with “<” and “>”. These allow the user to navigate through the months.

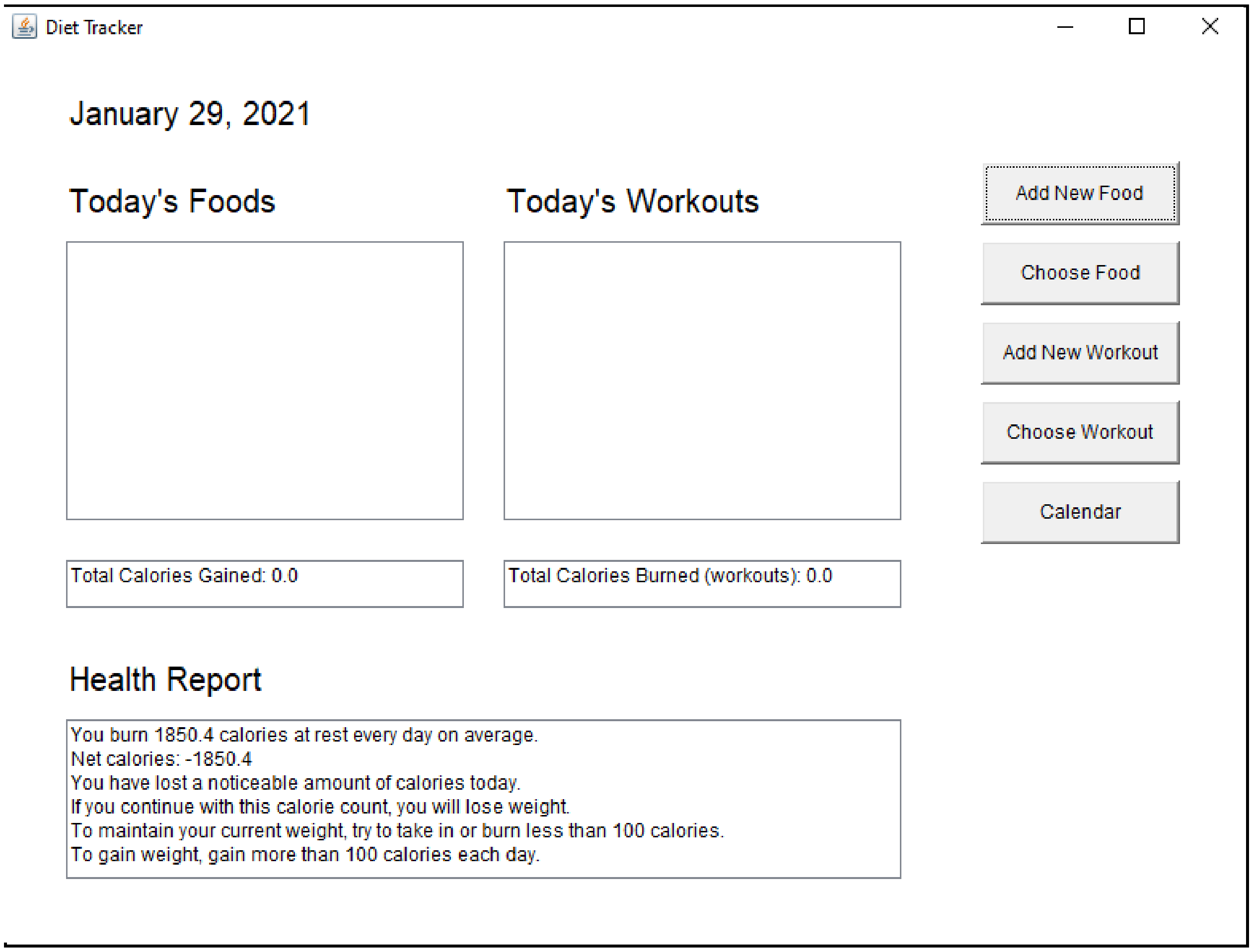
```
public void setToPrevMonth() {
    cal.add(Calendar.MONTH, -1);
    date = cal.getTime();
}

public void setToNextMonth() {
    cal.add(Calendar.MONTH, 1);
    date = cal.getTime();
}
```

The month is changed, and the calendar screen is reprinted to show the new month.

Diet Screen

The diet screen consists of 5 lists that display text. There are also 5 buttons on the right side.

```
dailyFoods = addList("", 50, 150, 250, 175, this);
```



Total Calories Gained: 0.0

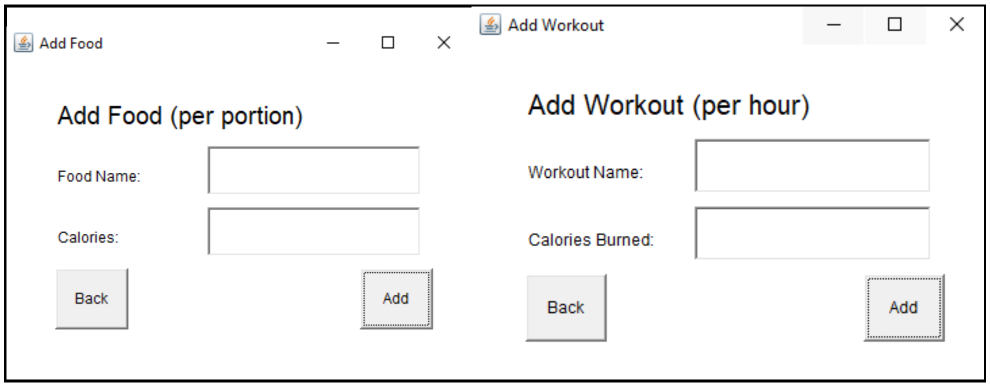
Total Calories Burned (workouts): 0.0

Health Report

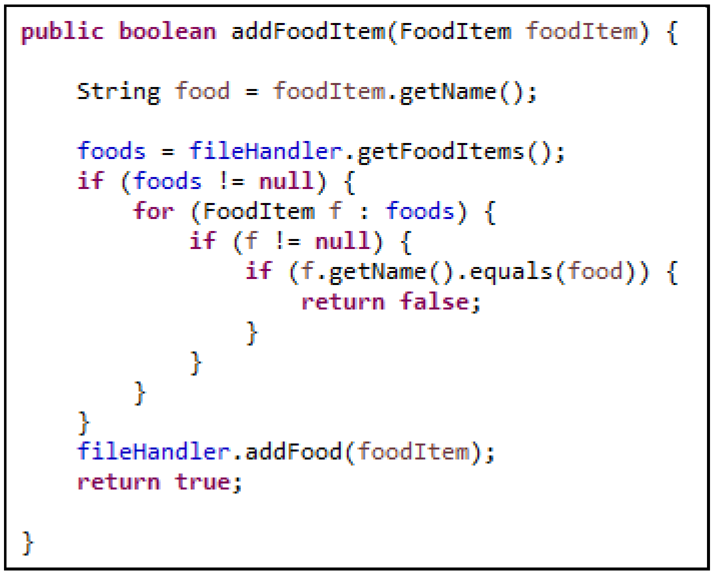
You burn 1850.4 calories at rest every day on average.
Net calories: -1850.4
You have lost a noticeable amount of calories today.
If you continue with this calorie count, you will lose weight.
To maintain your current weight, try to take in or burn less than 100 calories.
To gain weight, gain more than 100 calories each day.

Add New Food/Workout

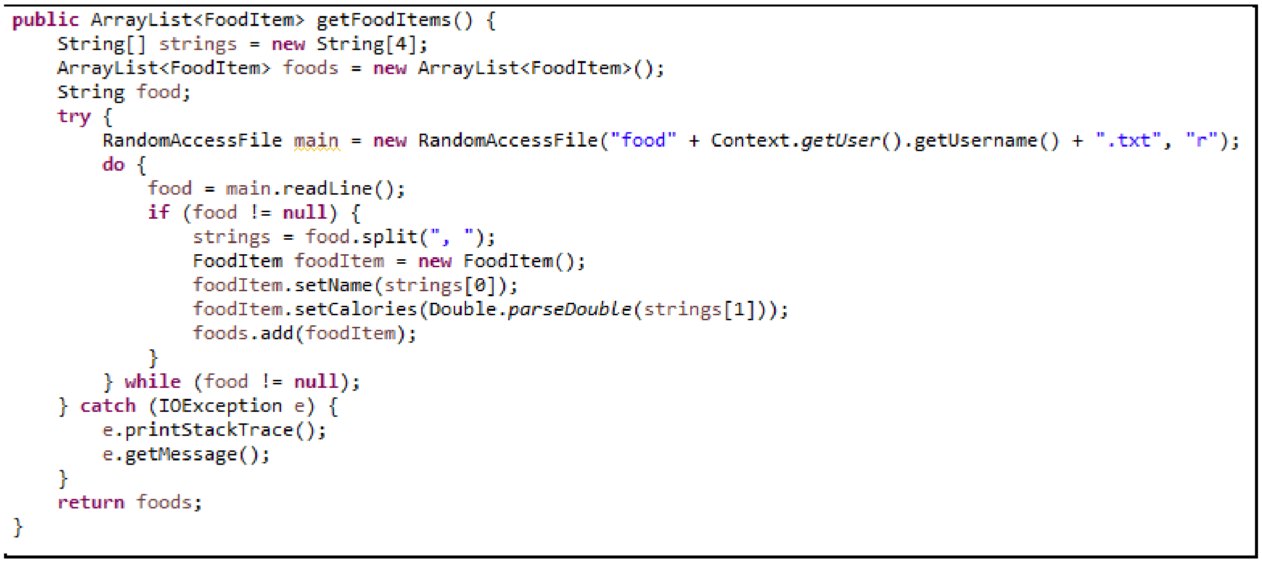
When the user wants to add a food, this screen will appear.



This works similarly to registration. The program searches the food file of the user to see if the food entered is already in the system. Workout works similarly.

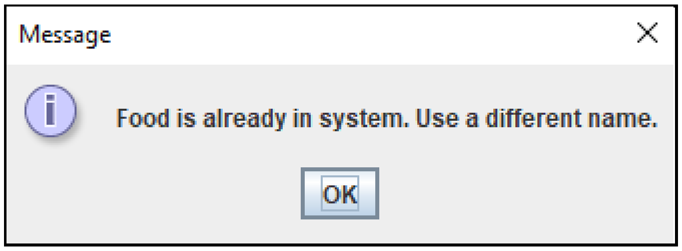


```java
public boolean addFoodItem(FoodItem foodItem) {

    String food = foodItem.getName();

    foods = fileHandler.getFoodItems();
    if (foods != null) {
        for (FoodItem f : foods) {
            if (f != null) {
                if (f.getName().equals(food)) {
                    return false;
                }
            }
        }
    }
    fileHandler.addFood(foodItem);
    return true;

}
```



```java
public ArrayList<FoodItem> getFoodItems() {
    String[] strings = new String[4];
    ArrayList<FoodItem> foods = new ArrayList<FoodItem>();
    String food;
    try {
        RandomAccessFile main = new RandomAccessFile("food" + Context.getUser().getUsername() + ".txt", "r");
        do {
            food = main.readLine();
            if (food != null) {
                strings = food.split(", ");
                FoodItem foodItem = new FoodItem();
                foodItem.setName(strings[0]);
                foodItem.setCalories(Double.parseDouble(strings[1]));
                foods.add(foodItem);
            }
        } while (food != null);
    } catch (IOException e) {
        e.printStackTrace();
        e.getMessage();
    }
    return foods;
}
```

If the food is not in the file already, the food will be entered into the file via FileHandler. If it is, an error message will show:

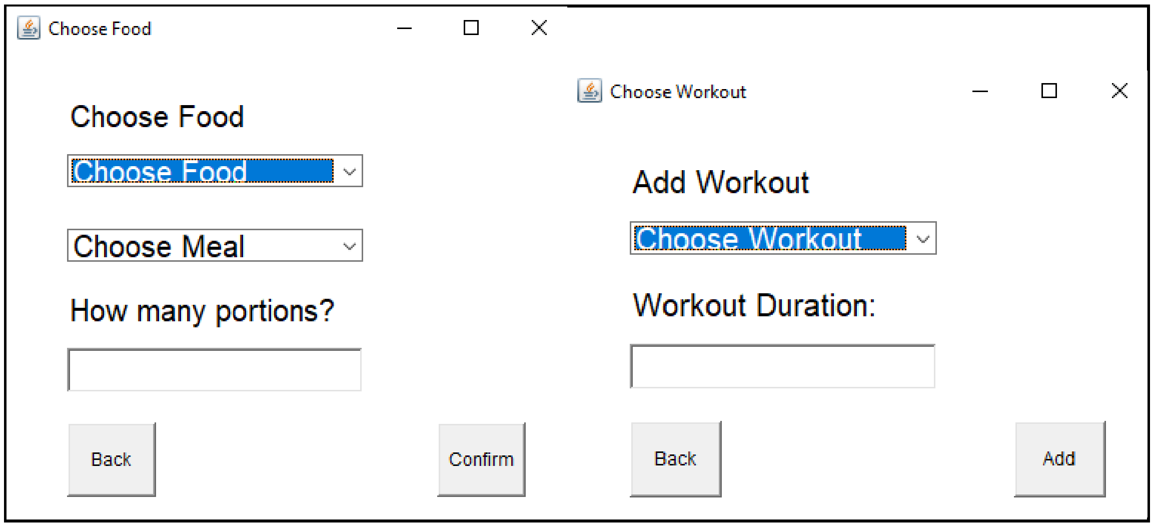


Message

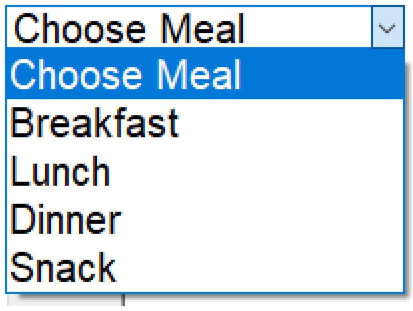
Food is already in system. Use a different name.

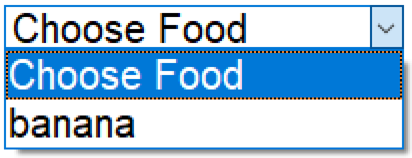
Choose Food/Workout

Once foods/workouts are added into the system, the user can choose the specific foods to add to the diet screen.

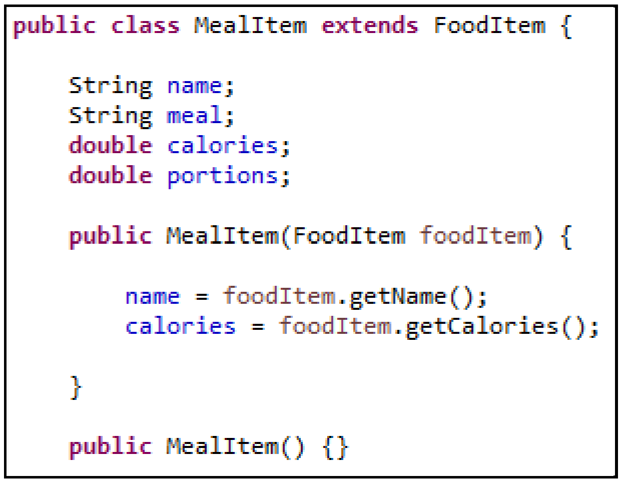


The drop down menus allow the user to choose which food they ate that day and which meal they ate it for. They also enter how many portions they ate. These are the drop down menus:





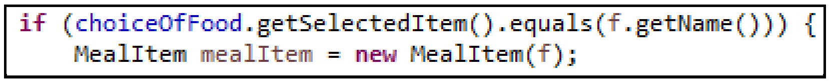
When adding a food/workout to a day, inheritance is used. The food is now from the MealItem class which extends FoodItem, and the workout is from the WorkoutSession class which extends WorkoutItem.



```java
public class MealItem extends FoodItem {

    String name;
    String meal;
    double calories;
    double portions;

    public MealItem(FoodItem foodItem) {

        name = foodItem.getName();
        calories = foodItem.getCalories();

    }

    public MealItem() {}
```

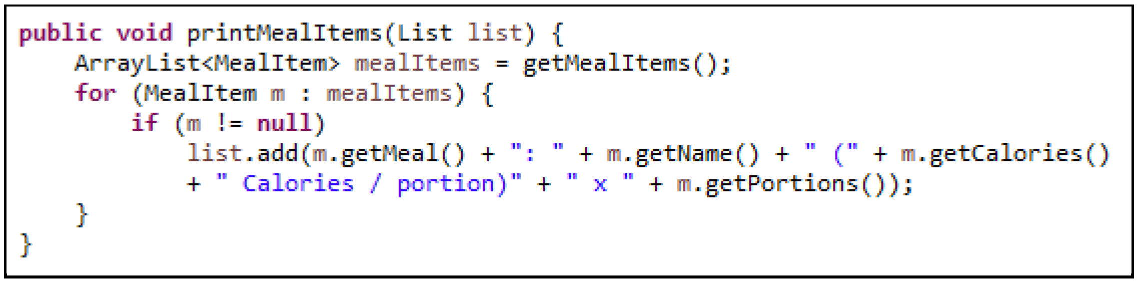
One of the constructors for MealItem/WorkoutSession is that they take the name of a FoodItem/Workout Item. The only difference is that the MealItem has a meal variable and a portions variable, and the WorkoutSession has a session time variable. This way, portions/meal/time can be manipulated for each individual item without affecting the item in the main file.

After the user enters the parameters, the program takes the FoodItem selected and passes it in to the MealItem.

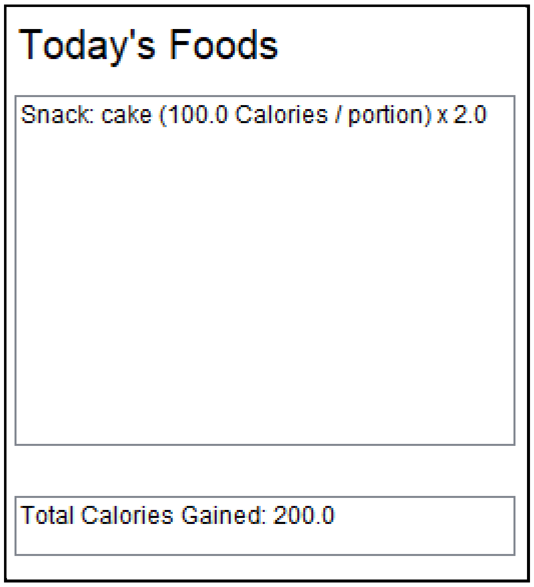


```java
if (choiceOfFood.getSelectedItem().equals(f.getName())) {
    MealItem mealItem = new MealItem(f);
```

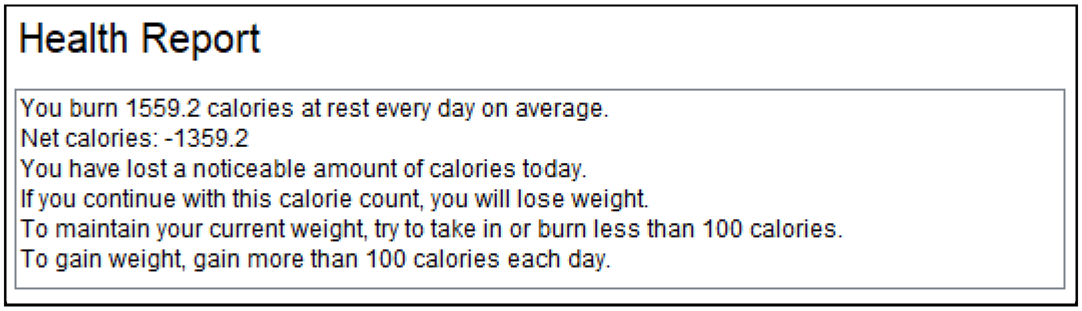
After the food/workout is selected, it is entered into a file for that specific date and it is displayed on the diet screen:



```java
public void printMealItems(List list) {
    ArrayList<MealItem> mealItems = getMealItems();
    for (MealItem m : mealItems) {
        if (m != null)
            list.add(m.getMeal() + ": " + m.getName() + " (" + m.getCalories()
            + " Calories / portion)" + " x " + m.getPortions());
    }
}
```

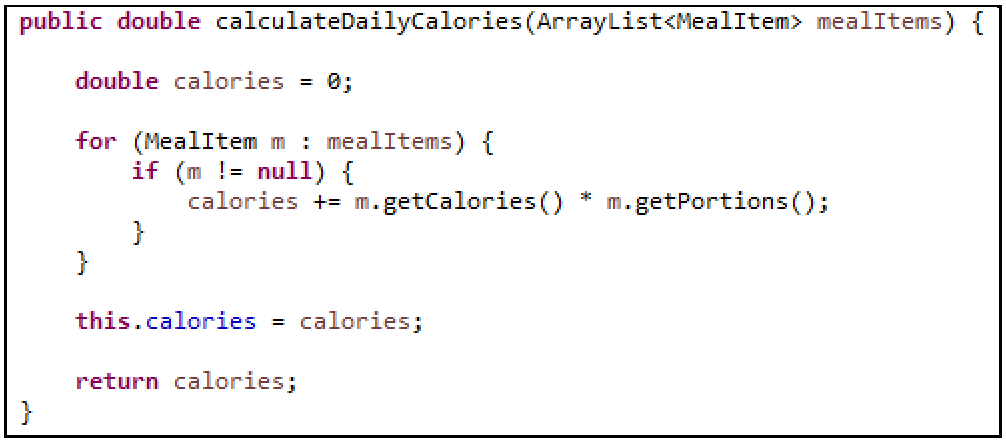


```
Snack: cake (100.0 Calories / portion) x 2.0

Total Calories Gained: 200.0
```

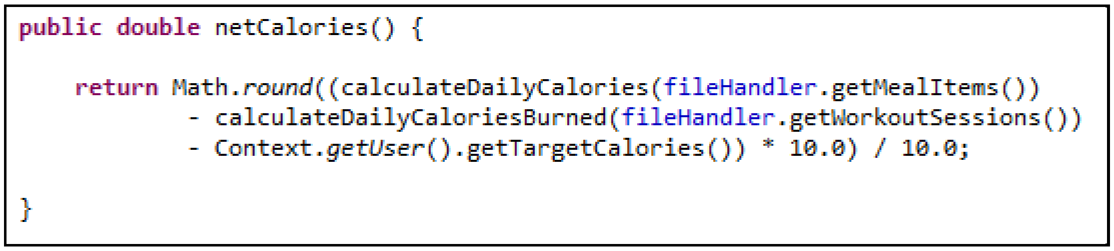


```
You burn 1559.2 calories at rest every day on average.
Net calories: -1359.2
You have lost a noticeable amount of calories today.
If you continue with this calorie count, you will lose weight.
To maintain your current weight, try to take in or burn less than 100 calories.
To gain weight, gain more than 100 calories each day.
```

The total calories gained and the net calories are calculated using formulas in the Adjudicator class. The comments in the health report are also from the Adjudicator, and displays comments based on the net calories.



```java
public double calculateDailyCalories(ArrayList<MealItem> mealItems) {

    double calories = 0;

    for (MealItem m : mealItems) {
        if (m != null) {
            calories += m.getCalories() * m.getPortions();
        }
    }

    this.calories = calories;

    return calories;
}
```



```
public double netCalories() {

    return Math.round((calculateDailyCalories(fileHandler.getMealItems())
            - calculateDailyCaloriesBurned(fileHandler.getWorkoutSessions())
            - Context.getUser().getTargetCalories()) * 10.0) / 10.0;

}
```

Calories gained are calculated by adding together all the calories of the foods, and calories burned are calculated by adding together all the calories burned from the workouts.

Adding and choosing the workouts are coded almost the same as the foods.

Word Count: 989