Appendix

Contents

Interview 1

Transcript of initial interview with the client: Mr. Hooria

Interviewee: Elias Hooria

Date/Time: May 23, 2023 (9:00 PM)

Method of interview: Online meeting

(the original interview was conducted in Farsi; this transcript is an English translation)

Me: Hello Mr. Hooria. First of all, thank you for agreeing to participate in this interview with me.

Client: It is my pleasure!

Me: could you give a description of your job? What does your job entail on a daily basis?

Client: as a fitness coach, I conduct group workout sessions every day, with different groups of athletes. The style of training that we follow is called “functional training”, which incorporates various exercise styles, with a strong emphasis on TRX training. Additionally, I also offer nutritional plans to athletes, to help them achieve their fitness goals.

Me: what is the significance of providing diets to athletes?

Client: well, each athlete has his or her own goals: whether it be weight loss, weight gain, weight maintenance and so on. 70% of gym success comes from the athletes’ nutritional diet. In fact, in the three dimensions of physical health (exercise, sleep, nutrition), nutrition is the most significant.

Me: Interesting. So, what is your process for giving athletes nutritional plans? What criteria do you consider when making the nutritional plan?

Client: we have the athletes fill out a questionnaire of around 40 questions and based on their answers we calculate the athlete's nutritional needs, and food preferences. At the most fundamental level, it just comes down to calories and macronutrients; calorie intake, calorie burnt throughout the day, maintenance calorie, etc. There are some formulas used to calculate these values.

Me: and once you have the athlete's nutritional needs, what do you do next? How do you turn that information into a diet?

Client: essentially, I have a list of foods, and their nutritional values. I go through the list and pick foods which match the athlete's needs. Some foods are used more frequently, like chicken or milk which have high protein, but if the athlete is vegan, for example, I must find replacements. I also prefer to make the diets a bit more flexible, by providing alternatives, so the athlete choose what to eat for each meal; I find this makes it easier for athletes to stick to the diet.

Me: alright, and what are the pros and cons of creating nutritional plans manually?

Client: Well, the biggest advantage is that I can consider the athlete's food preferences when devising their nutritional plans. I once had an athlete who really enjoyed sweets, and he insisted that I incorporate some sugar in his diet. With some modifications, I was able to change his diet in a way where he would be able to occasionally have modest amounts of sugar. The main disadvantage is, of course, manually making nutritional plans takes a lot of time. Especially when you want to make the diet flexible and provide different options for each meal. It takes several days to complete a single nutritional plan, and this process is very inefficient. Many of my colleagues, who are also trainers, don't provide nutritional plans to their athletes because of this same issue.

Me: what are your thoughts on having an automated system to generate nutritional plans based on the criteria you mentioned?

Client: well, if the program could consider food preferences when making the diet, that would be terrific! It would make it much more efficient for me, and perhaps I could increase the quantity of the nutritional plans I give to athletes, without compromising quality. Also, the plan could have even more variety in the meal options: for example, the athlete could choose from a selection of 5 breakfasts. It would be great!

Me: I could make such a program for you, with the formulas and details that you mentioned. Given the significance of nutrition, I think it would be a worthwhile project.

Client: yes of course! It would be superb. But is it possible to make it in a way which I can still manually modify the diet? I must be able to have some control in case your program makes a mistake, else it wouldn't be of much use.

Me: Yes of course. I will make it in a way where each of the meal options are editable.

Client: Fantastic! And once the program makes the diet, is it possible to save it as a PDF or something?

Me: Absolutely! I will add the feature to both save as a PDF and as an editable file. So that you may save and modify the diets in the future as well. perfect! I think we can discuss the details of the project afterward once I have made the general outline.

Client: sounds good!

Interview 2

__Transcript of second interview with the client, regarding nutritional science__

__Interviewee__: Elias Hooria

__Date/Time__: June 2, 2023 (10:00 PM)

Method of interview: Online meeting

(the original interview was conducted in Farsi; this transcript is an English translation)

Me: Hello Mr. Hooria, Once again thank you for participating in this interview. Last time you mentioned some formulas for calculating the nutritional needs of a person. Could you explain more about those?

Client: sure, so first of all, the BMR of the athlete must be calculated. BMR stands for Basal Metabolic Rate, and it is the base number of calories a person burns in one day, without considering the calories burnt through activities. In other words, the calories you burn for just being alive. The most common formula for this is the Mifflin formula.

Me: right, and how is the BMR used?

Client: once the BMR is calculated, you must consider the athlete's activity level to find the athlete's maintenance calorie. This is the number of calories required for the athlete to maintain his or her body mass. If the athlete is very active and exercises every day for example, then they burn much more calories in addition to his BMR. In that case for example you add 1000 calories to the BMR to find the maintenance calories. If the athlete is less active, then you add less calories.

Me: so how is this maintenance calorie useful now? What if an athlete wants to lose weight?

Client: If the client wants to gain weight, then you add more calories to the maintenance; this is called a calorie surplus. Typically, around 500 extra calories for mild weight gain. If the athlete wants to lose weight, then you must subtract some calories, so the athlete burns more calories in one day than they consume. This is called a calorie deficit.

Me: Ok so that's for the calories. How about the nutritional values themselves? How much protein is required for example?

Client: Generally, according to the dietary reference intake, the minimum amount of daily protein intake is 0.8 grams of protein per kilogram of bodyweight. But of course, this number is for the general public. For athletes it is recommended 0.8 grams of protein per pound of bodyweight, or around 1.5 grams per kilogram. The rest of the remaining calories is gained evenly from carbs and fats. Although fats may seem like a bad thing, but they are absolutely required for a balanced diet.

Me: Interesting. What about micronutrients then? for example vitamins or sodium, etc. are they considered when creating a diet?

Client: of course, micronutrients are extremely important and essential to a well-balanced diet. However, if the diet consists of a variety of different foods and snacks, the micronutrient needs of the person will almost always be met automatically. Indeed, if an athlete has a special condition or deficiency, then the micronutrients must be considered. In normal cases, though, considering micronutrients would be trivial.

Me: So, for this program do you think it is necessary to consider micronutrients or just base it off macronutrients?

Client: Well of course including micronutrients would be better, just to be on the safe side, but how difficult would it be to implement this?

Me: Since there are much more Micronutrients than Macronutrients, I suspect implementing them would make the algorithm much more complicated and longer to develop. On the other hand, the time for each diet to be generated would increase as well, since there are more constraints to consider.

Client: In that case, don't include micronutrients. In the uncommon scenario where an athlete has a deficiency, I will create the diet manually. Just make sure to create a varied diet, so that the micronutrient needs are met.

Me: very well! Once again thank you for the interview. I will keep you updated.

Interview 3

Transcript of third interview with the client, to evaluate the product

Interviewee: Elias Hooria

Date/Time: February 20, 2024 (5:15 PM)

Method of interview: face-to-face meeting

(the original interview was conducted in Farsi; this transcript is an English translation)

Me: Good evening, Mr. Hooria. I have created the diet generator program as we have discussed. Today I would like to get your feedback on the product in reference to the success criteria we agreed upon.

Client: Very well then.

Me: in relation to success criteria 1 and 2, how did you find the aliment selector and its filter and search functionalities?

Client: The aliment selector is very good. I found it intuitive to use, and I especially liked the filtering functionality. The search is also very good, since the list updates as soon as I begin typing.

Me: That's great to hear! How about success criteria 5 and 6? What are your thoughts about adding and removing aliments?

Client: Oh, they are great. When you gave me the program, I felt like you had included a limited selection of aliments, so I'm glad that I could add and remove my own. The windows, inputs, and buttons work as expected.

Me: Awesome! What about success criteria 3 and 4? Regarding the inputs for the calorie and macronutrients goals, and the validation mechanism. Also how did you find the goals calculator window?

Client: The inputs worked as expected. I manually entered various values and the validation errors were all appropriate. Additionally, the goals calculator worked very well. I found it very convenient and accurate.

Me: Great. That was all for the input, now considering the output—criteria 7 and 8—what do you think about them?

Client: The output window is very well organized. I could easily see all the meals and meal options on one screen. And also, I really liked the editor window. Sometimes when it gave inappropriate meal options, I could change it to correct it.

Me: very well! And now regarding success criterion 9. Did you find the aliment information window useful?

Client: yes, it was implemented just how I wanted. And the pie chart is accurate as well, so yes.

Me: And now criteria 10 and 11? For saving and loading diets as editable files? What did you think about it?

Client: Once again, very well implemented. The file chooser was easy to use, and everything worked as intended. One minor improvement could be adding the load button to other windows as well. Because the only way to load a diet file is from the starting window. Although this is a very minor inconvenience, overall, well done.

Me: Alright, great to hear! And what about saving diet as PDF? Success criteria 12.

Client: It was very easy and convenient to save diets as PDFs. And the PDFs displayed the correct information. Although maybe they could be styled a bit more to make them more visually appealing. But for now, it is completely sufficient.

Me: Finally, what about the quality of the diet itself? Success criteria 13 and 14. Does the diet reach the appropriate standards?

Client: This can definitely be improved. Although the diet does include user preferences, I found that sometimes some of the meal options generated are not within the appropriate calorie and macronutrient range. Even though this happens rarely, I think it still needs to be improved. Moreover, sometimes the generated diet recommends a very small quantity—5 grams for example—of one aliment. 5 grams is really insignificant. And yea that's it, I think if these issues could be solved it would be a great product.

Me: Oh, I understand. Having used the product, do you think this tool will be able to help in your diet creation process?

Client: Definitely, although it may not completely automate the process, I will certainly use it to provide a template which I can further modify and perfect. So yes absolutely, this will make my job easier. Thank you!

Me: I am glad you like it. Hopefully future versions can have significant improvements which will help you even more. Thank you for putting the time and participating in this interview!

Client: No problem!

Code

All files also have a generated Javadoc webpage. Click here to view the Javadoc index page.

GeneticAlgorithm

Aliment.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * An instance of this class represents an aliment, also known as food item
 *
 * @author kxg708
 */
public class Aliment implements Serializable, Cloneable {
    /**
     * Number of calories in each serving of the aliment
     */
    private int caloriePerServing;
    /**
     * Amount of protein (in grams) in each serving of the aliment
     */
    private float proteinPerServing;
    /**
     * Amount of carbohydrate (in grams) in each serving of the aliment
     */
    private float carbPerServing;
    /**
     * Amount of fat (in grams) in each serving of the aliment
     */
    private float fatPerServing;
    /**
     * The magnitude or size of each serving
     */
    private float servingSize;
    /**
     * The measurement unit of each serving
     * - grams
     * - milliliters
     * - teaspoons
     * - tablespoons
     * - glasses
     */
    private String servingUnit;
    /**
     * The number of servings of the aliment
     */
    private float numOfServings;
    /**
     * A list of meal tendencies in which the aliment is suitable to be used. It is a
     * list since one aliment may be suitable for many meals.
     */
    private List<Meal.TENDENCY> mealTendency = new ArrayList<>();
    /**
     * Name of the aliment
     */
    private String name;


    /**
     * Getter for aliment name
     *
     * @return aliment name
     */
    public String getName() {
        return name;
    }

    /**
     * Setter for aliment name
     *
     * @param name the new name to be set.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Getter for calories per serving of the aliment
     *
     * @return calories per serving of the aliment
     */
    public int getCaloriePerServing() {
        return caloriePerServing;
    }

    /**
     * Setter for calories per serving of the aliment
     *
     * @param caloriePerServing the new calories per serving of the aliment
     */
    public void setCaloriePerServing(int caloriePerServing) {
        this.caloriePerServing = caloriePerServing;
    }

    /**
     * Getter for protein per serving of the aliment
     *
     * @return the protein per serving of the aliment
     */
    public float getProteinPerServing() {
        return proteinPerServing;
    }

    /**
     * Setter for protein per serving of the aliment
     *
     * @param proteinPerServing the new protein per serving of the aliment
     */
    public void setProteinPerServing(float proteinPerServing) {
        this.proteinPerServing = proteinPerServing;
    }

    /**
     * Getter for carbohydrate per serving of the aliment
     *
     * @return carbohydrate per serving of the aliment
     */
    public float getCarbPerServing() {
        return carbPerServing;
    }

    /**
     * Setter for carbohydrate per serving of the aliment
     *
     * @param carbPerServing the new carbohydrate per serving of the aliment
     */
    public void setCarbPerServing(float carbPerServing) {
        this.carbPerServing = carbPerServing;
    }

    /**
     * Getter for fat per serving of the aliment
     *
     * @return the fat per serving of the aliment
     */
    public float getFatPerServing() {
        return fatPerServing;
    }

    /**
     * Setter for fat per serving of the aliment
     *
     * @param fatPerServing the new fat per serving of the aliment
     */
    public void setFatPerServing(float fatPerServing) {
    this.fatPerServing = fatPerServing;
    }
```

```java
    /**
     * Getter for meal tendency list
     *
     * @return the current meal tendency list
     */
    public List<Meal.TENDENCY> getMealTendency() {
        return mealTendency;
    }

    /**
     * Add a new meal tendency to the existing list
     *
     * @param mealTendency the new meal tendency to be added
     */
    public void addMealTendency(Meal.TENDENCY mealTendency) {
        this.mealTendency.add(mealTendency);
    }

    /**
     * Getter for serving size of the aliment
     *
     * @return the serving size of the aliment
     */
    public float getServingSize() {
        return servingSize;
    }
    /**
     * Setter for serving size of the aliment
     *
     * @param servingSize the new serving size of the aliment
     */
    public void setServingSize(float servingSize) {
        this.servingSize = servingSize;
    }

    /**
     * Getter for serving unit of the aliment
     *
     * @return the serving unit of the aliment
     */
    public String getServingUnit() {
        return servingUnit;
    }

    /**
     * Setter for serving unit of the aliment
     *
     * @param servingUnit the new serving unit of the aliment
     */
    public void setServingUnit(String servingUnit) {
        this.servingUnit = servingUnit;
    }
    /**
     * Getter for number of servings of the aliment
     *
     * @return the number of servings of the aliment
     */
    public float getNumOfServings() {
        return numOfServings;
    }

    /**
     * Setter for number of servings of the aliment
     *
     * @param numOfServings the new number of servings of the aliment
     */
    public void setNumOfServings(float numOfServings) {
        this.numOfServings = numOfServings;
    }

    /**
     * turns the aliment to a string representation by returning the name
     * of the aliment.
     *
     * @return the name of the aliment
     */
    @Override
    public String toString() {
        return name;
    }

    /**
     * clones the aliment object into another object with a different
     * address in memory, so that the clone can be modified.
     *
     * @return cloned object
     */
    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    /**
     * Checks whether the current aliment is equal to another aliment
     * After safety checks, the comparison is done by checking aliment names.
     *
     * @param object is the object being compared.
     * @return a Boolean representing whether the two objects are equal.
     */
    @Override
    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;

        Aliment aliment = (Aliment) object;
        return name.equals(aliment.getName());
    }

}
```

Breakfast.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

/**
 * A subclass of the Meal class, which takes in the calorie and macro-nutrient
 * goals and calculates 30% of each, and creates the appropriate fitness object
 * for Breakfast.
 *
 * @author kxg708
 */
public class Breakfast extends Meal {
    /**
     * A percentage constant of 30%, for getting 30% of the total goals.
     */
    public static final float PERCENTAGE_DIVIDER = 0.3f;

    /**
     * The constructor which takes in the calorie and macro-nutrient goals
     * specified in the diet creation window, and considers only 30% of
     * each constraint for the Breakfast meal. It then creates a fitness
     * calculator object based off those values, and passes it to the super
     * class.
     *
     * @param calorieGoal the total calorie goal of the diet
     * @param proteinGoal the total protein goal (in grams) of the diet
     * @param carbGoal the total carbohydrate goal (in grams) of the diet
     * @param fatGoal the total fat goal (in grams) of the diet
     */
    public Breakfast(int calorieGoal, float proteinGoal, float carbGoal, float fatGoal) {
        super();

        int calorieConstraint = Math.round(calorieGoal * PERCENTAGE_DIVIDER);
        float proteinConstraint = proteinGoal * PERCENTAGE_DIVIDER;
        float carbConstraint = carbGoal * PERCENTAGE_DIVIDER;
        float fatConstraint = fatGoal * PERCENTAGE_DIVIDER;
        Meal.TENDENCY mealTendency = Meal.TENDENCY.BREAKFAST;
        super.fitnessCalc = new FitnessCalculator(calorieConstraint, proteinConstraint,
carbConstraint, fatConstraint, mealTendency);
    }

}
```

Chromosome.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import java.util.HashMap;
import java.util.Map;
import com.mycompany.dietgenerator.App;
import com.mycompany.dietgenerator.Validation;
import java.io.Serializable;

/**
 * An instantiation of this class represents a chromosome in the genetic algorithm or in
other words
 * a solution within computational space. The chromosome encoding is done in key/value
pairs in a
 * map where the key represents the index of the aliment (phenotype representation) in the
global
 * variable aliments and the value represents the number of servings of that aliment. The
class also
 * contains a decode method for converting the solution from computational space to
physical space.
 *
 * @author kxg708
 */
public class Chromosome implements Serializable, Cloneable {
    /**
     * A static number for the size of the solution. In other words, how many aliments a
solution should
     * contain. Here it is specified 4
     */
    public static int CHROMOSOME_SIZE = 4;

    /**
     * The map representing the key/value pairs of the chromosome. Here the key is the
aliment index
     * and the value is the number of servings of that aliment.
     */
    private Map<Integer, Float> chromosome;

    /**
```

```java
     * The fitness or the suitability of the solution. The fitness starts off as a large
negative number
     * and approaches 0 as the Solution gets better.
     */
    private double fitness;

    /**
     * The constructor of the Chromosome class, which assigns the chromosome variable to
an empty map.
     */
    public Chromosome() {
        chromosome = new HashMap<>();
    }

    /**
     * Sets the chromosome map of this object to another. It is used in Survivor selection
operations for
     * replacing weak solutions.
     *
     * @param otherChromosome the chromosome whose contents should be copied in this.
     */
    public void setEqualTo(Chromosome otherChromosome) {
        chromosome.clear();
        Integer[] keyset = otherChromosome.getKeyset();

        for (Integer key : keyset) {
            chromosome.put(key, otherChromosome.getGene(key));
        }
    }

    /**
     * Getter for each gene in the chromosome
     *
     * @param key the index of the gene which should be returned.
     * @return the gene with the index of key.
     */
    public float getGene(int key) {
        return chromosome.get(key);
    }

    /**
     * Adds a new gene to the chromosome map. A gene can be considered as a key/value
pair.
     * This is used in Genetic Algorithm operations such as mutation, where the chromosome
     * must be manipulated.
     *
     * @param alimentIndex the index of the aliment, which is the key
     * @param numOfServings the number of servings of that aliment, which is the value.
     */
    public void addGene(int alimentIndex, float numOfServings) {
        chromosome.put(alimentIndex, numOfServings);
    }

    /**
     * removes a gene with the index or key specified.
     *
     * @param alimentIndex the aliment index, or the key of the gene which should be
removed.
     */
    public void removeGene(int alimentIndex) {
        chromosome.remove(alimentIndex);
    }

    /**
     * Getter for chromosome size
     *
     * @return the size of the chromosome map.
     */
    public int getSize() {
        return chromosome.size();
    }

    /**
     * Checks whether an aliment with a certain index exists in the chromosome
     *
     * @param alimentIndex the index of the aliment checked in the global variable
aliments.
     * @return true if the chromosome does contain a key, and false if it doesn't.
     */
    public boolean containsKey(int alimentIndex) {
        return chromosome.containsKey(alimentIndex);
    }

    /**
     * Getter for the fitness of the chromosome.
     *
     * @return the fitness value of the chromosome.
     */
    public double getFitness() {
        return fitness;
    }

    /**
     * Setter for fitness value of the chromosome
     *
     * @param fitness the new fitness value of the chromosome.
     */
    public void setFitness(double fitness) {
        this.fitness = fitness;
    }
```

```java
    /**
     * Converts the chromosome map which is in computational space into a phenotype
representation
     * in physical space. The phenotype representation is an array of aliment objects
which have the
     * number of servings data member assigned.
     *
     * @return an array of aliments with the number of servings set assigned.
     */
    public Aliment[] decode() {
        Aliment[] decoded = new Aliment[chromosome.size()];

        try {
            // loop through each gene and find its
            // representation in App.aliments
            Aliment aliment;
            int index = 0;
            for (Map.Entry<Integer, Float> gene : chromosome.entrySet()) {
                aliment = App.aliments.get(gene.getKey());
                aliment = (Aliment) aliment.clone();

                aliment.setNumOfServings(gene.getValue());
                decoded[index] = aliment;
                index++;
            }
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }

        return decoded;
    }

    /**
     * returns the set of aliment indices or keys of the chromosome map
     *
     * @return an integer array of all the keys in the chromosome map.
     */
    public Integer[] getKeyset() {
        return chromosome.keySet().toArray(new Integer[0]);
    }

    /**
     * Method to be able to clone the chromosome object.
     */
    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
```

```
}
```

Crossover.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import java.io.Serializable;
import java.util.Random;

/**
 * A genetic algorithm static utility class, in charge of creating offspring based on an
 * even array of parent chromosomes. Offspring creation is implemented using the single
 * point cross over algorithm
 *
 * @author kxg708
 */
public class Crossover implements Serializable {
    /**
     * Generates an array of offspring based on an even array of parents. This is done by
     * iterating the parents array in pairs (i, i+1), taking parent[i] as parent 1 and
parent[i + 1]
     * as parent 2 and then performing single point cross over on them which returns 2
offspring
     *
     * @param parents the even array of parents from which the offspring are created.
     * @return an array of generated offspring, with the same length as the parents
     */
    public static Chromosome[] generateOffsprings(Chromosome[] parents) {
        if (parents.length % 2 != 0) {
            System.err.println("Number of parents must be even!");
            System.exit(1);
        }

        // number of offspring is the same as number of parents
        Chromosome[] offsprings = new Chromosome[parents.length];

        Chromosome[] temp;
        for (int i = 0; i + 1 < parents.length;i = i + 2) {
            temp = singlePointCrossOver(parents[i], parents[i + 1]);
            offsprings[i] = temp[0];
            offsprings[i + 1] = temp[1];

        }

        return offsprings;
    }
    /**
     * This method is an implementation of the single point cross over algorithm. It
generates a random
     * crossover index, and swaps the content of each parent chromosome until that index.
There are also
     * two loops at the end of the method, which are implemented to resolve a very
specific bug causing
     * the chromosome sizes to shrink.
     *
     * @param parent1 the first parent
     * @param parent2 the second parent
     * @return an array of size 2 of the generated offspring.
     */
    private static Chromosome[] singlePointCrossOver(Chromosome parent1, Chromosome
parent2) {
        // generate a crossover point
        Random random = new Random();
        int crossOverPoint = random.nextInt(Chromosome.CHROMOSOME_SIZE + 1);

        Integer[] parent1keys = parent1.getKeyset();
        Integer[] parent2keys = parent2.getKeyset();

        Chromosome offSpring1 = new Chromosome();
        Chromosome offSpring2 = new Chromosome();

        // add the keys of one parent until the crossover point is reached
        for (int i = 0; i < crossOverPoint; i++) {
            offSpring1.addGene(parent1keys[i], parent1.getGene(parent1keys[i]));
            offSpring2.addGene(parent2keys[i], parent2.getGene(parent2keys[i]));
        }
        // switch parents and add the genes of the other parent, in turn creating
crossover
        for (int i = crossOverPoint; i < Chromosome.CHROMOSOME_SIZE; i++) {
            offSpring1.addGene(parent2keys[i], parent2.getGene(parent2keys[i]));
            offSpring2.addGene(parent1keys[i], parent1.getGene(parent1keys[i]));
        }

        // in rare occasions, when a key is present in both parents, it can be added
        // to the offspring twice. the second time overwrites the first time and this
causes
        // the chromosome size to shrink. hence this check is necessary
        while (offSpring1.getSize() < Chromosome.CHROMOSOME_SIZE) {
            PopulationInitializer.addRandomGene(offSpring1);
        }
        while (offSpring2.getSize() < Chromosome.CHROMOSOME_SIZE) {
            PopulationInitializer.addRandomGene(offSpring2);
        }

        return new Chromosome[]{offSpring1, offSpring2};
    }
```

```
}
```

Diet.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * An instantiation of this class represents a complete diet with all four meals
 * (Breakfast, Snack, Lunch, and Dinner). When the user saves a diet as an editable
 * file, an object of this class is saved.
 *
 * @author kxg708
 */
public class Diet implements Serializable {
    /**
     * A list of mealOptions in breakfast
     */
    public ArrayList<MealOption> breakfast;

    /**
     * A list of mealOptions in lunch
     */
    private ArrayList<MealOption> lunch;

    /**
     * A list of mealOptions in snack
     */
    private ArrayList<MealOption> snack;

    /**
     * A list of mealOptions in dinner
     */
    private ArrayList<MealOption> dinner;

    /**
     * The name of the editable file. When a new diet is generated, the initial
     * value for this variable is "undefined". it can modified once the user saves
     * it as an editable file. The file name is also displayed in the title of the
     * diet display window.
     */
    public String filename = "undefined";

    /**
     * The path of the editable file. When a new diet is generated, the initial
     * value for this variable is "undefined". it can modified once the user saves
     * it as an editable file.
     */
    public String path = "undefined";

    /**
     * The constructor of the Diet class. It takes in the meal options of each meal
     * and assigns them to the corresponding private data members.
     *
     * @param breakfast a list of mealOptions in the breakfast meal.
     * @param lunch a list of mealOptions in the lunch meal.
     * @param snack a list of mealOptions in the snack meal.
     * @param dinner a list of mealOptions in the dinner meal.
     */
    public Diet(ArrayList<MealOption> breakfast, ArrayList<MealOption> lunch,
ArrayList<MealOption> snack, ArrayList<MealOption> dinner) {
        this.breakfast = breakfast;
        this.lunch = lunch;
        this.snack = snack;
        this.dinner = dinner;
    }

    /**
     * Getter for breakfast mealOptions
     *
     * @return the list of mealOptions in breakfast.
     */
    public ArrayList<MealOption> getBreakfast() {
        return breakfast;
    }

    /**
     * Setter for the breakfast mealOptions
     *
     * @param breakfast the new list of meal options in breakfast.
     */
    public void setBreakfast(ArrayList<MealOption> breakfast) {
        this.breakfast = breakfast;
    }

    /**
     * Getter for lunch mealOptions
     *
     * @return the list of mealOptions in lunch.
     */
    public ArrayList<MealOption> getLunch() {
        return lunch;
    }

    /**
```

```
     * Setter for the lunch mealOptions
     *
     * @param lunch the new list of meal options in lunch.
     */
    public void setLunch(ArrayList<MealOption> lunch) {
        this.lunch = lunch;
    }

    /**
     * Getter for snack mealOptions
     *
     * @return the list of mealOptions in snack.
     */
    public ArrayList<MealOption> getSnack() {
        return snack;
    }

    /**
     * Setter for the snack mealOptions
     *
     * @param snack the new list of meal options in snack.
     */
    public void setSnack(ArrayList<MealOption> snack) {
        this.snack = snack;
    }

    /**
     * Getter for dinner mealOptions
     *
     * @return the list of mealOptions in dinner.
     */
    public ArrayList<MealOption> getDinner() {
        return dinner;
    }

    /**
     * Setter for the dinner mealOptions
     *
     * @param dinner the new list of meal options in dinner.
     */
    public void setDinner(ArrayList<MealOption> dinner) {
        this.dinner = dinner;
    }
}
```

Dinner.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

/**
 * A subclass of the Meal class, which takes in the calorie and macro-nutrient
 * goals and calculates 15% of each, and creates the appropriate fitness object
 * for Dinner.
 *
 * @author kxg708
 */
public class Dinner extends Meal {
    /**
     * A percentage constant of 15%, for getting 15% of the total goals.
     */
    public static final float PERCENTAGE_DIVIDER = 0.15f;

    /**
     * The constructor which takes in the calorie and macro-nutrient goals
     * specified in the diet creation window, and considers only 15% of
     * each constraint for the Dinner meal. It then creates a fitness
     * calculator object based off those values, and passes it to the super
     * class.
     *
     * @param calorieGoal the total calorie goal of the diet
     * @param proteinGoal the total protein goal (in grams) of the diet
     * @param carbGoal the total carbohydrate goal (in grams) of the diet
     * @param fatGoal the total fat goal (in grams) of the diet
     */
    public Dinner(int calorieGoal, float proteinGoal, float carbGoal, float fatGoal) {
        super();

        int calorieConstraint = Math.round(calorieGoal * PERCENTAGE_DIVIDER);
        float proteinConstraint = proteinGoal * PERCENTAGE_DIVIDER;
        float carbConstraint = carbGoal * PERCENTAGE_DIVIDER;
        float fatConstraint = fatGoal * PERCENTAGE_DIVIDER;
        Meal.TENDENCY mealTendency = Meal.TENDENCY.DINNER;

        super.fitnessCalc = new FitnessCalculator(calorieConstraint, proteinConstraint,
carbConstraint, fatConstraint, mealTendency);
    }
}
```

FitnessCalculator.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

import com.mycompany.dietgenerator.App;
import java.io.Serializable;
```

```
/**
 * A genetic algorithm utility class which includes functions for calculating the
 * fitness of a chromosome. Objects of this class take in the calorie, protein,
 * carbohydrate, and fat constraints and finds the variation between the constraints
 * and the chromosome.
 *
 * @author kxg708
 */
public class FitnessCalculator implements Serializable {
    /**
     * Coefficient for adding weight to calorie constraint. value obtained from
     * Trial and error, and can easily be modified.
     */
    private static final float CALORIE_COEFF = 5f;

    /**
     * Coefficient for adding weight to protein constraint. value obtained from
     * Trial and error, and can easily be modified.
     */
    private static final float PROTEIN_COEFF = 10f;

    /**
     * Coefficient for adding weight to fat constraint. value obtained from Trial
     * and error, and can easily be modified.
     */
    private static final float FAT_COEFF = 10f;

    /**
     * Coefficient for adding weight to carbohydrate constraint. value obtained
     * from Trial and error, and can easily be modified.
     */
    private static final float CARB_COEFF = 10f;

    /**
     * Coefficient for adding weight to preference constraint. value obtained from
     * Trial and error, and can easily be modified.
     */
    private static final float PREFERENCE_COEFF = 10f;

    /**
     * Coefficient for adding weight to meal tendency constraint. value obtained
     * from Trial and error, and can easily be modified.
     */
    private static final float MEAL_TENDENCY_COEFF = 100f;

    /**
     * An arbitrary step value to increase the fitness or decrease the fitness
     * based on preference.
     */
    private static final int PREFERENCE_STEP = 5;

    /**
     * An arbitrary step value to increase the decrease the fitness based on meal
     * tendency suitability of the chromosome.
     */
    private static final int MEAL_TENDENCY_STEP = 5;

    /**
     * The meal that this object is being used for. For instance if the algorithm
     * is generating a breakfast, this would be Meal.BREAKFAST
     */
    private Meal.TENDENCY mealTendency;

    /**
     * The number of calories each meal option should be
     */
    private int calorieConstraint;

    /**
     * The amount of protein (in grams) each meal option should be
     */
    private float proteinConstraint;

    /**
     * The amount of carbohydrates (in grams) each meal option should be
     */
    private float carbConstraint;

    /**
     * The amount of fat (in grams) each meal option should be
     */
    private float fatConstraint;

    /**
     * The Constructor of the FitnessCalculator class. In this constructor
     * the constraints are passed in and assigned to the appropriate data
     * members.
     *
     * @param calorieConstraint the calorie constraint
     * @param proteinConstraint the protein constraint
     * @param carbConstraint the carbohydrate constraint
     * @param fatConstraint the fat constraint
     * @param mealTendency the meal tendency constraint
     */
    public FitnessCalculator(int calorieConstraint, float proteinConstraint, float
carbConstraint, float fatConstraint, Meal.TENDENCY mealTendency) {
        this.calorieConstraint = calorieConstraint;
        this.proteinConstraint = proteinConstraint;
        this.carbConstraint = carbConstraint;
```

```java
        this.fatConstraint = fatConstraint;
        this.mealTendency = mealTendency;
    }

    /**
     * Calculates the difference (distance) between the protein of the aliments
     * of a meal option and the specified protein constraints
     *
     * @param aliments the array of aliments being checked
     * @return negative float showing the difference, which is weighted by coefficient
     */
    private float calculateProteinVariation(Aliment[] aliments) {
        float proteinSum = 0;
        for (Aliment aliment : aliments) {
            proteinSum += aliment.getProteinPerServing() * aliment.getNumOfServings();
        }

        return -Math.abs(proteinConstraint - proteinSum) * PROTEIN_COEFF;
    }

    /**
     * Calculates the difference (distance) between the fat of the aliments of a
     * meal option and the specified fat constraints
     *
     * @param aliments the array of aliments being checked
     * @return negative float showing the difference, which is weighted by coefficient
     */
    private float calculateFatVariation(Aliment[] aliments) {
        float fatSum = 0;
        for (Aliment aliment : aliments) {
            fatSum += aliment.getFatPerServing() * aliment.getNumOfServings();
        }

        return -Math.abs(fatConstraint - fatSum) * FAT_COEFF;
    }

    /**
     * Calculates the difference (distance) between the carbohydrate of the aliments
     * of a meal option and the specified carbohydrate constraints
     *
     * @param aliments the array of aliments being checked
     * @return negative float showing the difference, which is weighted by coefficient
     */
    private float calculateCarbVariation(Aliment[] aliments) {
        float carbSum = 0;
        for (Aliment aliment : aliments) {
            carbSum += aliment.getCarbPerServing() * aliment.getNumOfServings();
        }

        return -Math.abs(carbConstraint - carbSum) * CARB_COEFF;
    }

    /**
     * Calculates the difference (distance) between the calorie of the aliments
     * of a meal option and the specified calorie constraints
     *
     * @param aliments the array of aliments being checked
     * @return negative float showing the difference, which is weighted by coefficient
     */
    private float calculateCalorieVariation(Aliment[] aliments) {
        float calorieSum = 0;
        for (Aliment aliment : aliments) {
            calorieSum += aliment.getCaloriePerServing() * aliment.getNumOfServings();
        }

        return -Math.abs(calorieConstraint - calorieSum) * CALORIE_COEFF;
    }

    /**
     * Calculates the variation between the meal tendencies of the aliments of a
     * meal option and the specified meal tendency constraint. If the aliment is
     * not suitable for the specified meal, then a negative score is added.
     *
     * @param aliments the array of aliments being checked
     * @return negative float showing the variation, which is weighted by coefficient
     */
    private float calculateMealTendencyVariation(Aliment[] aliments) {
        float mealTendencySum = 0;

        for (Aliment aliment : aliments) {
            if(!aliment.getMealTendency().contains(mealTendency)) {
                mealTendencySum -= MEAL_TENDENCY_STEP;
            }
        }

        return mealTendencySum * MEAL_TENDENCY_COEFF;
    }

    /**
     * Calculates the variation between user preferences and the aliments int the
     * meal option. for every aliment that the user likes, a positive 3 points are
     * added and for every disliked food, a 3 points are deducted.
     *
     * @param aliments the array of aliments being checked
     * @return negative float showing the variation, which is weighted by coefficient
     */
    private float calculatePreferenceVariation(Aliment[] aliments) {
        float variation = 0;
        for (Aliment aliment : aliments) {
            if(App.likes.contains(aliment)) variation += PREFERENCE_STEP;
```

```java
            if (App.dislikes.contains(aliment)) variation -= PREFERENCE_STEP;
        }

        return variation * PREFERENCE_COEFF;
    }

    /**
     * A method which gets the variations of all the constraints. And combines them
     * to create a final fitness score for the given chromosome.
     *
     * @param chromosome the chromosome for which a fitness is being calculated
     * @return a float representing the fitness value.
     */
    public float calculateFitness(Chromosome chromosome) {
        Aliment[] aliments = chromosome.decode();
        float calorieVar = calculateCalorieVariation(aliments);
        float fatVar = calculateFatVariation(aliments);
        float proteinVar = calculateProteinVariation(aliments);
        float carbVar = calculateCarbVariation(aliments);
        float preferenceVar = calculatePreferenceVariation(aliments);
        float mealTendencyVar = calculateMealTendencyVariation(aliments);

        return (preferenceVar + calorieVar + fatVar + proteinVar + carbVar +
mealTendencyVar);
    }

}
```

Lunch.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

/**
 * A subclass of the Meal class, which takes in the calorie and macro-nutrient
 * goals and calculates 40% of each, and creates the appropriate fitness object
 * for Lunch.
 *
 * @author kxg708
 */
public class Lunch extends Meal {
    /**
     * A percentage constant of 40%, for getting 40% of the total goals.
     */
    public static final float PERCENTAGE_DIVIDER = 0.4f;

    /**
     * The constructor which takes in the calorie and macro-nutrient goals
     * specified in the diet creation window, and considers only 40% of
     * each constraint for the Lunch meal. It then creates a fitness
     * calculator object based off those values, and passes it to the super
     * class.
     *
     * @param calorieGoal the total calorie goal of the diet
     * @param proteinGoal the total protein goal (in grams) of the diet
     * @param carbGoal the total carbohydrate goal (in grams) of the diet
     * @param fatGoal the total fat goal (in grams) of the diet
     */
    public Lunch(int calorieGoal, float proteinGoal, float carbGoal, float fatGoal) {
        super();

        int calorieConstraint = Math.round(calorieGoal * PERCENTAGE_DIVIDER);
        float proteinConstraint = proteinGoal * PERCENTAGE_DIVIDER;
        float carbConstraint = carbGoal * PERCENTAGE_DIVIDER;
        float fatConstraint = fatGoal * PERCENTAGE_DIVIDER;
        Meal.TENDENCY mealTendency = Meal.TENDENCY.LUNCH;

        super.fitnessCalc = new FitnessCalculator(calorieConstraint, proteinConstraint,
carbConstraint, fatConstraint, mealTendency);
    }
}
```

Meal.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import com.mycompany.dietgenerator.Validation;
import java.util.ArrayList;


/**
 * The superclass for Breakfast, Snack, Lunch, and Dinner classes. An instantiation
 * of this class or subclasses represents a meal, which contains meal options, and
 * calls on the meal options run() method on each mealOption to generate the diet.
 *
 * @author kxg708
 */
public class Meal implements Runnable {
    /**
     * A static enumerator for specifying meal type. In different parts of the code.
     */
    public static enum TENDENCY {
        /**
         * Represents the breakfast meal.
         */
        BREAKFAST,
```

```java
        /**
         * Represents the lunch meal.
         */
        LUNCH,
        /**
         * Represents the snack meal.
         */
        SNACK,
        /**
         * Represents the dinner meal.
         */
        DINNER};

    /**
     * A static variable for the number of meal options a meal should have. Can be
     * easily changed by other developers.
     */
    private static int numOfMealOptions = 5;

    /**
     * List of options in the meal. same size as numOfMealOptions.
     */
    private ArrayList<MealOption> mealOptions;

    /**
     * A fitnessCalculator object with the correct constraints for each meal. It is
     * assigned in the subclasses.
     */
    protected FitnessCalculator fitnessCalc;

    /**
     * The constructor for the Meal Class. Assigns the mealOptions list to an
     * empty arrayList. This constructor is called by subclasses.
     */
    public Meal() {
        this.mealOptions = new ArrayList<>();
    }

    /**
     * This method is internally called in {@link
com.mycompany.dietgenerator.DietCreationWindowController}
     * when thread.start(); is called.
     *
     * It creates 5 separate threads for each mealOption, and by calling thread.start(),
     * the run() method of {@link MealOption} is called.
     *
     * In this way 5 mealOptions are generated concurrently.
     */
    @Override
    public void run() {
        try {
            ArrayList<Thread> threads = new ArrayList<>();

            Thread newThread;
            MealOption option;

            for (int i = 0; i < numOfMealOptions; i++) {
                option = new MealOption(fitnessCalc);
                newThread = new Thread(option);
                newThread.start();

                threads.add(newThread);
                mealOptions.add(option);
            }

            for (Thread thread : threads) {
                thread.join();
            }
        } catch(Exception e) {
            Validation.showErrorAlert(e);
        }
    }

    /**
     * Getter for list of options in the meal.
     *
     * @return list of options in the meal.
     */
    public ArrayList<MealOption> getMealOptions() {
        return mealOptions;
    }

    /**
     * Getter for number of options in the meal.
     *
     * @return number of options in the meal.
     */
    public int getNumOfMealOptions() {
        return numOfMealOptions;
    }

    /**
     * Setter for number of options in the meal.
     *
     * @param numOfMealOptions the new number of options in the meal.
     */
    public void setNumOfMealOptions(int numOfMealOptions) {
        this.numOfMealOptions = numOfMealOptions;
    }
```

```
}
```

MealOption.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import java.io.Serializable;
import java.util.List;
import java.util.Arrays;

/**
 * An instance of this class represents a single option in a meal. This class
 * contains methods for running the genetic algorithm to create the meal option,
 *
 * @author kxg708
 */
public class MealOption implements Serializable, Runnable {
    /**
     * The population size for the genetic algorithm.
     */
    public static int POPULATION_SIZE = 30;

    /**
     * The number of solutions which should be compared in each round of the K-way
     * tournament selector.
     */
    public static int K = 4;

    /**
     * Array containing all chromosomes of the population.
     */
    private Chromosome[] genotype;

    /**
     * A list containing the aliments selected by the genetic algorithm. Once the best
     * chromosome is found, the chromosome.decode() method is run to turn it into an
aliment
     * list which is a physical representation of the chromosome.
     */
    private List<Aliment> aliments;

    /**
     * Object for calculating the fitness of each chromosome.
     */
    private FitnessCalculator fitnessCalc;

    /**
```

```
     * Object for selecting the parents of each generation of the algorithm.
     */
    private ParentSelector parentSelector;

    /**
     * Object for finding and replacing the least fit solutions with the generated array
     * of offspring.
     */
    private SurvivorSelector survivorSelector;

    /**
     * The maximum number of generations the algorithm should go through. This is to
prevent
     * an infinite loop.
     */
    private final int maxIterations = 10000;

    /**
     * Constructor for the MealOption class. It takes in a fitness calculator, which is
set
     * with the correct constraints for the meal.
     *
     * @param fitnessCalc the fitness calculator object with which to evaluate the
chromosomes.
     */
    public MealOption(FitnessCalculator fitnessCalc) {
        this.fitnessCalc = fitnessCalc;
    }

    /**
     * This method is internally called in {@link Meal} when thread.start(); is called. It
runs
     * the genetic algorithm and saves the generated meal option in the aliments list.
     */
    @Override
    public void run() {
        Chromosome[] parents; // temporary storage of the parents of each generation
        Chromosome[] offSprings; // temporary storage of the offsprings of each generation

        genotype = PopulationInitializer.populate(POPULATION_SIZE);
        this.parentSelector = new ParentSelector(genotype, K);
        this.survivorSelector = new SurvivorSelector(genotype, K);

        Chromosome fittest = genotype[0]; // holds the fittest meal option in each
generation
        float fitness; // temporary variable for holding fitness values

        int iteration = 0;
        do {
            // assign fitness of the generation
```

```java
            for (Chromosome chromosome : genotype) {
                fitness = fitnessCalc.calculateFitness(chromosome);
                chromosome.setFitness(fitness);
                // find fittest meal option
                if (fitness < fittest.getFitness()) fittest = chromosome;
            }

            // select the top individuals as parents
            parents = parentSelector.getWinners(6);

            // create offsprings based on the chosen parents.
            offSprings = Crossover.generateOffsprings(parents);

            // assign a fitness to the offspring
            for (Chromosome chromosome : offSprings) {
                fitness = fitnessCalc.calculateFitness(chromosome);
                chromosome.setFitness(fitness);
            }

            // replace the worst performers in the population with the offsprings
            survivorSelector.replace(offSprings);

            // randomly mutate the population
            Mutator.mutate(genotype);

            iteration++;
        } while (iteration < maxIterations && fittest.getFitness() < -50);

        aliments = Arrays.asList(fittest.decode());
    }

    /**
     * Getter for the aliments list.
     *
     * @return the aliment list.
     */
    public List<Aliment> getAliments() {
        return aliments;
    }

    /**
     * replaces the aliments list with another aliments list. It is used in
     * {@link com.mycompany.dietgenerator.MealOptionEditorWindowController},
     * to manually modify the meal option.
     *
     * @param aliments the new aliments list which should be added.
     */
    public void addAll(List<Aliment> aliments) {
        this.aliments = aliments;
    }
    /**
     * Method for converting the mealOption to string. by joining the name of each
     * aliment and its amount (serving size * number of servings) with a + sign.
     * This is used in diet display window to display the meal options in the tables.
     */
    @Override
    public String toString() {
        String result = "";

        for (int i = 0; i < aliments.size(); i++) {
            result += aliments.get(i).getName() + " " +
Math.round(aliments.get(i).getNumOfServings()*aliments.get(i).getServingSize()) + " " +
aliments.get(i).getServingUnit();
            if (i != aliments.size() - 1) {
                result += " + ";
            }
        }

        return result;
    }
}
```

Mutator.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

import java.util.Random;

/**
 * A genetic algorithm static utility class which contains methods for mutating an array
of
 * chromosomes. Mutation is implemented using the random swap algorithm.
 *
 * @author kxg708
 */
public class Mutator {
    /**
     * The probability with which to perform mutation. set to 15%, but can easily be
     * modified.
     */
    private static final int MUTATION_PROBABILITY = 15;

    /**
     * A small 5% probability with which to add a guided gene in the swap method.
     */
    private static final int GUIDED_GENE_PERCENTAGE = 5;
    /**
     * A static Random object in order to generate random numbers
     */
    private static final Random RANDOM = new Random();

    /**
     * This is a utility method which returns true with a probability specified. it is
     * implemented by checking whether a random number between 1 and 100 is less than
     * the probability specified.
     *
     * @param probability the probability of returning true
     * @return true with a certain probability. Otherwise false.
     */
    public static boolean chance(int probability) {
        // Generates a RANDOM number between 1 and 100
        int randomNumber = RANDOM.nextInt(100) + 1;

        return randomNumber <= probability;
    }

    /**
     * This method is the mutation algorithm. it replaces a random gene from the specified
     * chromosome with a random gene, using the aliments global variable.
     *
     * @param chromosome the chromosome on which mutation should be run
     */
    private static void swap(Chromosome chromosome) {
        int randomIndexRemoved = RANDOM.nextInt(chromosome.getSize());
        Integer[] keys = chromosome.getKeyset();
        chromosome.removeGene(keys[randomIndexRemoved]);

        if (Mutator.chance(GUIDED_GENE_PERCENTAGE)) {
            PopulationInitializer.addGuidedGene(chromosome);
        } else {
            PopulationInitializer.addRandomGene(chromosome);
        }
    }

    /**
     * This method iterates through each chromosome of a population and mutates them with
a
     * probability of MUTATION_PROBABILITY
     *
     * @param genotype the population of chromosomes on which mutation should be run
     */
    public static void mutate(Chromosome[] genotype) {
        for (Chromosome chromosome : genotype) {
            if (chance(MUTATION_PROBABILITY)) {
                swap(chromosome);
            }
        }
    }
}
```

ParentSelector.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import java.io.Serializable;

/**
 * A subclass of the TournamentSelector class, implementing the methods for parent
 * selection operations. These operations include, finding the winner of each round
 * of the tournament algorithm.
 *
 * @author kxg708
 */
public class ParentSelector extends TournamentSelector implements Serializable {
    /**
     * Constructor for the ParentSelector class which takes in the population
     * object and K (the number of selected solutions each round) to implement the
     * tournament selector. These values are passed to the superclass.
     *
     * @param population the population object on which parent selection is done each
generation.
     * @param k the number of selected solutions each round of the tournament selector
algorithm.
     */
    public ParentSelector(Chromosome[] population, int k) {
        super(population, k);
    }

    /**
     * This method finds the winner of one round of the tournament selection algorithm.
     * This is done by finding the fittest solution in the given array of chromosomes.
     *
     * @param chromosomes the array of k chromosomes being checked.
     * @return a single chromosome object which has the highest fitness
     */
    private Chromosome findWinner(Chromosome[] chromosomes) {
        if (chromosomes == null) return null;

        int i = 0;
        Chromosome fittest = chromosomes[i];
        while (i < chromosomes.length) {
            if (chromosomes[i].getFitness() > fittest.getFitness()) {
                fittest = chromosomes[i];
            }
            i++;
        }

        return fittest;
    }

    /**
     * This method runs the tournament selector algorithm a number of times to get a
specified
     * number of winners. These winners are the fittest solutions in the population and
are the
     * parents based off which offspring are made.
     *
     * @param numOfWinners the number of winners (parents) the algorithm should return
     * @return an array of winners with same length as numOfWinners.
     */
    public Chromosome[] getWinners(int numOfWinners) {
        Chromosome[] selected;
        Chromosome[] winners = new Chromosome[numOfWinners];

        for (int i = 0; i < numOfWinners; i++) {
            selected = selectKChromosomes();
            winners[i] = findWinner(selected);
        }

        return winners;
    }
}
```

PopulationInitializer.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;
import com.mycompany.dietgenerator.App;
import java.util.ArrayList;
import java.util.Random;

/**
 * A genetic algorithm utility class which handles population related operations.
 * The main one being, creating an initial population of chromosomes.
 *
 * @author kxg708
 */
public class PopulationInitializer {
    /**
     * A static Random object in order to generate random numbers
```

```java
     */
    private static final Random RANDOM = new Random();

    /**
     * A 20% probability with which to add a guided gene in the create chromosome
     * method.
     */
    private static final int GUIDED_GENE_PROBABILITY = 20;

    /**
     * A method which takes the number of individuals in the population and creates
     * that many chromosomes. It is used to initialize the population.
     *
     * @param amount number of individuals in population.
     * @return an array of chromosomes representing the created population.
     */
    public static Chromosome[] populate(int amount) {
        Chromosome[] population = new Chromosome[amount];

        for (int i = 0; i < amount; i++) {
            population[i] = createChromosome();
        }

        return population;
    }

    /**
     * A method which creates a semi random new chromosome. It uses two utility
     * methods {@link  #addRandomGene(GeneticAlgorithm.Chromosome)} and
     * {@link #addGuidedGene(GeneticAlgorithm.Chromosome)} for adding genes. The
     * number of genes added is the same as chromosome size.
     *
     * @return the created chromosome.
     */
    private static Chromosome createChromosome() {
        Chromosome chromosome = new Chromosome();

        for (int i = 0; i < Chromosome.CHROMOSOME_SIZE; i++) {
            if (Mutator.chance(GUIDED_GENE_PROBABILITY)) {
                addGuidedGene(chromosome);
            } else {
                addRandomGene(chromosome);
            }
        }

        return chromosome;
    }

    /**
     * This method adds a single random gene to the specified chromosome. It
     * generates a random aliment index which is not already in the chromosome
     * and adds it. A do while loop is implemented to find another index, if the
     * one chosen is already in the chromosome.
     *
     * @param chromosome the chromosome to which the gene should be added
     */
    public static void addRandomGene(Chromosome chromosome) {
        int randomIndexAdded;
        float randomFloat = RANDOM.nextFloat() * 5;

        do {
            randomIndexAdded = RANDOM.nextInt(App.aliments.size());
        } while (chromosome.containsKey(randomIndexAdded));

        chromosome.addGene(randomIndexAdded, randomFloat);
    }

    /**
     * This method adds a single gene to the specified chromosome, which is taken
     * from the user's liked aliments list (App.likes). If the preference list is
     * empty or the chromosome already contains all the liked  aliments, then a
     * random gene is added.
     *
     *
     * @param chromosome the chromosome to which the gene should be added
     */
    public static void addGuidedGene(Chromosome chromosome) {
        int randomAvailableIndex;
        int randomIndex;
        int randomIndexAdded;
        float randomFloat = RANDOM.nextFloat() * 5;

        ArrayList<Integer> availableIndices = new ArrayList<>();

        for(int i = 0; i < App.likes.size(); i++) {
            availableIndices.add(i);
        }

        do {
            // if App.likes is empty or the chromosome
            // already contains all liked aliments
            if (availableIndices.isEmpty()) {
                addRandomGene(chromosome);
                return;
            }

            randomAvailableIndex = RANDOM.nextInt(availableIndices.size());
            randomIndex = availableIndices.get(randomAvailableIndex);
            randomIndexAdded = App.aliments.indexOf(App.likes.get(randomIndex));
            availableIndices.remove(randomAvailableIndex);
        } while (chromosome.containsKey(randomIndexAdded));

        chromosome.addGene(randomIndexAdded, randomFloat);
    }
}
```

Snack.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

/**
 * A subclass of the Meal class, which takes in the calorie and macro-nutrient
 * goals and calculates 15% of each, and creates the appropriate fitness object
 * for Snack meal.
 *
 * @author kxg708
 */
public class Snack extends Meal {
    /**
     * A percentage constant of 15%, for getting 15% of the total goals.
     */
    public static final float PERCENTAGE_DIVIDER = 0.15f;

    /**
     * The constructor which takes in the calorie and macro-nutrient goals
     * specified in the diet creation window, and considers only 15% of
     * each constraint for the Snack meal. It then creates a fitness
     * calculator object based off those values, and passes it to the super
     * class.
     *
     * @param calorieGoal the total calorie goal of the diet
     * @param proteinGoal the total protein goal (in grams) of the diet
     * @param carbGoal the total carbohydrate goal (in grams) of the diet
     * @param fatGoal the total fat goal (in grams) of the diet
     */
    public Snack(int calorieGoal, float proteinGoal, float carbGoal, float fatGoal) {
        super();

        int calorieConstraint = Math.round(calorieGoal * PERCENTAGE_DIVIDER);
        float proteinConstraint = proteinGoal * PERCENTAGE_DIVIDER;
        float carbConstraint = carbGoal * PERCENTAGE_DIVIDER;
        float fatConstraint = fatGoal * PERCENTAGE_DIVIDER;
        Meal.TENDENCY mealTendency = Meal.TENDENCY.SNACK;

        super.fitnessCalc = new FitnessCalculator(calorieConstraint, proteinConstraint,
carbConstraint, fatConstraint, mealTendency);
```

```java
    }
}
```

SurvivorSelector.java

Click here for Javadoc related to this file.

```java
package GeneticAlgorithm;

import com.mycompany.dietgenerator.Validation;
import java.io.Serializable;

/**
 * A subclass of the TournamentSelector class, implementing the methods for survivor
 * selection operations. These operations include, finding the loser of each round of
 * the tournament algorithm and replacing the losers with the array of offspring.
 *
 * @author kxg708
 */
public class SurvivorSelector extends TournamentSelector implements Serializable {
    /**
     * Constructor for the SurvivorSelector class which takes in the population
     * object and K (the number of selected solutions each round) to implement the
     * tournament selector. These values are passed to the superclass.
     *
     * @param population the population object on which parent selection is done each
generation.
     * @param k the number of selected solutions each round of the tournament selector
algorithm.
     */
    public SurvivorSelector(Chromosome[] population, int k) {
        super(population, k);
    }

    /**
     * This method finds the loser of one round of the tournament selection algorithm.
     * This is done by finding the least fit solution in the given array of chromosomes.
     *
     * @param chromosomes the array of k chromosomes being checked.
     * @return a single chromosome object which has the least fitness
     */
    private Chromosome findLoser(Chromosome[] chromosomes) {
        if (chromosomes == null) return null;

        int i = 0;
        Chromosome leastFit = chromosomes[i];
        while (i < chromosomes.length) {
            if (chromosomes[i].getFitness() < leastFit.getFitness()) {
                leastFit = chromosomes[i];
            }
            i++;
        }

        return leastFit;
    }

    /**
     * This method takes an array of offspring and runs the tournament selector
     * algorithm on the population to find the weakest chromosomes. then it replaces
     * those "loser" chromosomes with the offspring, using the setEqualTo(); method.
     *
     * @param offSprings the of offspring to be replaced in population.
     */
    public void replace(Chromosome[] offSprings) {
        Chromosome[] selected;
        Chromosome loser;

        for (Chromosome offSpring : offSprings) {
            selected = selectKChromosomes();
            loser = findLoser(selected);
            if (loser.getFitness() < offSpring.getFitness()) {
                try {
                    loser.setEqualTo(offSpring);
                }catch (Exception e) {
                    Validation.showErrorAlert(e);
                }
            }
        }
    }
}
```

TournamentSelector.java

Click here for Javadoc related to this file.

```
package GeneticAlgorithm;

import java.io.Serializable;
import java.util.Random;

/**
 * A genetic algorithm utility class which acts as the superclass for the SurvivorSelector
 * and TournamentSelector subclasses. This class contains shared methods for implementing
 * a K-way tournament selector algorithm which is utilized for the  parent and
 * survivor selection operations in the genetic algorithm.
 *
 * @author kxg708
 */
```

```java
public class TournamentSelector implements Serializable {
    /**
     * It is the population object within the genetic algorithm.
     */
    protected Chromosome[] population;

    /**
     * The number of solutions to select in each round of the tournament selection
     * algorithm.
     */
    private int k;

    /**
     * The constructor of the TournamentSelector algorithm, which is called by
     * the subclasses. It takes in the population object and K (the number of
     * selected solutions each round) to implement the tournament selector. These
     * values are passed to the superclass.
     *
     * @param population the population object on which parent selection is done each
generation.
     * @param k the number of selected solutions each round of the tournament selector
algorithm.
     */
    public TournamentSelector(Chromosome[] population, int k) {
        this.population = population;
        this.k = k;
    }

    /**
     * This method selects k solutions from the population.
     *
     * @return an array of k selected chromosomes.
     */
    protected Chromosome[] selectKChromosomes() {
        Chromosome[] selected = new Chromosome[k];
        Random random = new Random();

        int rand;
        for (int i = 0;i < k;i++) {
            rand = random.nextInt(population.length);
            selected[i] = population[rand];
        }

        return selected;
    }
}
```

FileManager

FileManager.java

Click here for Javadoc related to this file.

```java
package FileManager;

import GeneticAlgorithm.Diet;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import GeneticAlgorithm.Aliment;
import GeneticAlgorithm.MealOption;
import com.ibm.icu.text.ArabicShapingException;
import com.mycompany.dietgenerator.App;
import com.mycompany.dietgenerator.Validation;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import com.ibm.icu.text.Bidi;
import java.util.ArrayList;
import java.util.List;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.PDPageContentStream;
import org.apache.pdfbox.pdmodel.font.PDType0Font;

/**
 * A static class which contains operations related to File Management. These operations
 * include, saving diets as an editable file, saving diets as a PDF, reading and writing
 * to the aliment.data file, and loading diets from editable files.
 *
 * @author kxg708
 */
public class FileManager {
    /**
     * Location of aliment.data file
     */
    public static final String ALIMENT_DATA_FILE_LOCATION = ".\\resources\\aliments.data";

    /**
     * Location of PDF printing font file. The format for this location string
     * is different with the one above, because it is being passed to the
     * App.class.getResource() method.
     */
    public static final String FONT_FILE_LOCATION = "font/B-NAZANIN.TTF";
    /**
     * 50 pixels margin on the PDF printing page.
     */
    public static float pdfPageMargin = 50;

    /**
     * The x-coordinate for printing a on PDF page.
     */
    public static float pdfStartX;

    /**
     * The y-coordinate for printing a on PDF page.
     */
    public static float pdfStartY;

    /**
     * 30 pixels space between lines.
     */
    public static float pdfLineHeight = 30;

    /**
     * Apache PDF box variable for font.
     */
    public static PDType0Font font;

    /**
     * This method opens a file chooser window where user can select a diet file,
     * and calls the private method {@link #readAlimentDataFile()} to read its
     * contents.
     *
     * @return the Diet object read from file. If no file is chosen null is returned.
     * @throws FileNotFoundException in case of error.
     * @throws IOException in case of error.
     * @throws ClassNotFoundException in case of error.
     */
    public static Diet openDietFile() throws FileNotFoundException, IOException,
ClassNotFoundException {
            Stage window = new Stage();
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Open a Diet File");
            fileChooser.getExtensionFilters()
                        .addAll(new FileChooser.ExtensionFilter("Diet", "*.diet"));

            File file = fileChooser.showOpenDialog(window); // file chooser pop-up

            if (file != null) return readDietFile(file);

            // translation of text: no file was opened!
            Validation.showWarningAlert("هیچ فایلی باز نشد!");
            return null;
    }

    /**
     * This method takes in a diet file object and reads and returns its content.
     *
     * @return the read Diet object from file.
     * @throws FileNotFoundException in case of error.
     * @throws IOException in case of error.
     * @throws ClassNotFoundException in case of error.
     */
    private static Diet readDietFile(File file) throws FileNotFoundException, IOException,
ClassNotFoundException {
        FileInputStream inFileStream = new FileInputStream(file);
        ObjectInputStream inObjectStream = new ObjectInputStream(inFileStream);

        Diet diet = (Diet) inObjectStream.readObject();
        return diet;
    }


    /**
     * This method takes a diet and opens a file chooser window for the user to
     * save the diet as a file. It calls the {@link #saveDietFile(java.io.File,
GeneticAlgorithm.Diet)}
     * to do this.
     *
     * This method is called when a diet is being saved in a new file.
     *
     * @param diet the Diet object which should be written to the file.
     */
    public static void saveAsDietFile(Diet diet) {
        try {
            Stage window = new Stage();
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save a Diet File");
            fileChooser.getExtensionFilters()
                       .addAll(new FileChooser.ExtensionFilter("Diet", "*.diet"));
            File file = fileChooser.showSaveDialog(window);
            if (file != null) {
                diet.filename = file.getName();
                diet.path = file.getAbsolutePath();

                saveDietFile(file, diet);
            }
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }

    /**
```

```
     * Takes a Diet object and a File object, and writes the diet to the file. It
     * is used in {@link #saveAsDietFile(GeneticAlgorithm.Diet)} and also in
     * dietDisplayWindow when a previously saved Diet file has been modified and
     * should be saved. In this case, a file chooser is not needed to be displayed.
     *
     * @param diet the Diet object which should be written.
     * @param file the file to which the Diet object should be written.
     * @throws FileNotFoundException in case of error.
     * @throws IOException in case of error.
     */
    public static void saveDietFile(File file, Diet diet) throws FileNotFoundException,
IOException {
        FileOutputStream fileOutputStream = new FileOutputStream(file);
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(diet);
        objectOutputStream.close();
        fileOutputStream.close();
    }


    /**
     * This method draws a horizontal line separator on the PDF page using the
     * PDPageContentStream on a specified line number. This is done to implement
     * Apache PDF Box Printing operations easier, and reduces code redundancy.
     *
     * @param stream the stream for printing to the PDF page.
     * @param lineNum the line number on which it should be drawn.
     * @throws IOException in case of error.
     */
    private static void drawSeperator(PDPageContentStream stream, int lineNum) throws
IOException {
        stream.moveTo(pdfPageMargin, pdfStartY - pdfLineHeight*(lineNum - 1)); // Starting
point of separator line
        stream.lineTo(pdfStartX, pdfStartY - pdfLineHeight*(lineNum - 1)); // Ending point
of separator line
        stream.setStrokingColor(0, 0, 0); // Set color of the separator line
        stream.setLineWidth(1); // Set the width of separator line
        stream.stroke(); // Draw separator line
    }

    /**
     * This method prints a specified text on a specified line number on the
     * PDF page. It is used to implement Apache PDF Box printing operations
     * easier, and reduces code redundancy.
     *
     * @param stream the stream for printing to the PDF page.
     * @param text the text which should be printed.
     * @param lineNum the line number on which it should be drawn.
     * @param fontSize the size of the text font.
     * @throws IOException in case of error.
```

```java
     */
    private static void printText(PDPageContentStream stream, String text, int lineNum,
int fontSize) throws IOException {
        text = processRTLText(text);
        // Calculate the width of the text (AI generated)
        float textWidth = font.getStringWidth(text) / 1000 * fontSize;

        stream.beginText();
        stream.setFont(font, fontSize);
        stream.newLineAtOffset(pdfStartX - textWidth, pdfStartY - pdfLineHeight*(lineNum -
1));
        stream.showText(text);
        stream.endText();
    }

    /**
     * This method prints a single page of the PDF. It takes in the PDF document,
     * the name of the meal (breakfast, lunch, etc.), and the meal options, and
     * creates a new page and adds it to the document.
     *
     * @param document the PDF document to which the page should be printed.
     * @param options the list of meal options which should be printed on each line.
     * @param mealName the name of the meal.
     * @throws IOException in case of error.
     */
    private static void printPage(PDDocument document, ArrayList<MealOption> options,
String mealName) throws IOException {
        PDPage page = new PDPage();
        document.addPage(page);
        PDPageContentStream stream = new PDPageContentStream(document, page);

        pdfStartX = page.getMediaBox().getWidth() - pdfPageMargin;
        pdfStartY = 700;

        printText(stream, mealName, 1, 24);
        drawSeperator(stream, 2);
        // translation of text: please choose one of the options below
        printText(stream, "یکی از گزینه های زیر را انتخاب کنید", 3, 12);
        drawSeperator(stream, 4);

        //print mealOptions
        for (int i = 0; i < options.size();i++) {
            printText(stream, options.get(i).toString(), i + 5, 12);
        }

        stream.close();
    }

    /**
     * This method takes in a Diet object, and prints the meal options of each meal
     * on a separate page in a PDF, using {@link
#printPage(org.apache.pdfbox.pdmodel.PDDocument, java.util.ArrayList, java.lang.String)}
     * and opens a file chooser to save the PDF location.
     *
     * If no file is selected by file chooser, it is not saved anywhere.
     *
     * @param diet the Diet object which should be printed to the PDF.
     */
    public static void saveAsPDF(Diet diet) {
        try {
            PDDocument document = new PDDocument();
            font = PDType0Font.load(document,
App.class.getResourceAsStream(FONT_FILE_LOCATION));

            printPage(document, diet.getBreakfast(), "صبحانه");
            printPage(document, diet.getSnack(), "میان وعده");
            printPage(document, diet.getLunch(), "ناهار");
            printPage(document, diet.getDinner(), "شام");


            Stage window = new Stage();
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save as PDF");
            fileChooser.getExtensionFilters()
                       .addAll(new FileChooser.ExtensionFilter("PDF", "*.pdf"));
            File file = fileChooser.showSaveDialog(window);
            if (file != null) {
                document.save(file);
            }

        } catch (IOException e) {
            Validation.showErrorAlert(e);
        }
    }
    /**
     * To be able to print RTL Persian text with Apache PDF Box, this method had to be
     * included. It is taken from source:
     * https://stackoverflow.com/questions/48284888/writing-arabic-with-pdfbox-with-
correct-characters-presentation-form-without-bei
     *
     * @param rawText text to be processed
     * @returns formatted text.
     */
    private static String processRTLText(String rawText) {
        try {
            Bidi bidi = new Bidi((new
PersianShaping(PersianShaping.LETTERS_SHAPE)).shape(rawText), 127);
            bidi.setReorderingMode(0);
            return bidi.writeReordered(2);
        }
```

```java
        catch (ArabicShapingException ase3) {
            return rawText;
        }
    }

    /**
     * This method reads the list of aliments in the aliments.data file, and saves it in
     * the App.aliments global variable. It is called each time the program is launched.
     */
    public static void readAlimentDataFile() {
        try {
            ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(ALIMENT_DATA_FILE_LOCATION));
            App.aliments = (List<Aliment>) inputStream.readObject();
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }

    /**
     * This method takes an aliment object and adds it to the App.aliments global
variable.
     * It also updates the data file by running {@link #rewriteToAlimentDataFile()}
     *
     * It is used when creating a new aliment.
     * @param aliment the aliment to be added.
     */
    public static void addToAlimentDataFile(Aliment aliment) {
        App.aliments.add(aliment);
        rewriteToAlimentDataFile();
    }

    /**
     * This method takes an aliment object and removes it from the App.aliments global
variable.
     * It also updates the data file by running {@link #rewriteToAlimentDataFile()}
     *
     * It is used when deleting an aliment.
     * @param aliment the aliment to be removed.
     */
    public static void removeFromAlimentDataFile(Aliment aliment) {
        App.aliments.remove(aliment);
        rewriteToAlimentDataFile();
    }

    /**
     * This method writes the App.aliments list to the data file, in order to capture any
additions
     * or removal of aliments.
     */
    private static void rewriteToAlimentDataFile() {
        try {
            File file = new File(ALIMENT_DATA_FILE_LOCATION);
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file));

            oos.writeObject(App.aliments);
            oos.close();

        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }
}
```

PersianShaping.java

****Disclaimer**: This file was taken from, and can be viewed at https://drive.google.com/file/d/1g9j4oH-kPbzsNqF1VZ7jFeMEjOBois0j/view*

GUI

App.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import FileManager.FileManager;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;
import GeneticAlgorithm.Aliment;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;

/**
 * The main class of this program. It is based off of the template which is
 * initially generated by NetBeans.
 *
 * @author kxg708
 */

public class App extends Application {
    /**
     * The list of aliments which can be accessed globally. it is taken from
```

```java
     * the data file.
     */
    public static List<Aliment> aliments = new ArrayList<>();

    /**
     * The list of user liked aliments which can be accessed globally.
     */
    public static ObservableList<Aliment> likes = FXCollections.observableArrayList();

    /**
     * The list of user disliked aliments which can be accessed globally.
     */
    public static ObservableList<Aliment> dislikes = FXCollections.observableArrayList();


    /**
     * A method which starts the JavaFX GUI application. It is internally called
     * by launch() in Main.main(). It first reads the aliment.data file and then
     * displays the starting window of the program.
     *
     * @param stage JavaFX stage.
     * @throws IOException in case errors occur.
     */
    @Override
    public void start(Stage stage) throws IOException {
        FileManager.readAlimentDataFile();

        FXMLLoader loader = getWindow("startingWindow");
        Scene scene = new Scene(loader.load());
        StartingWindowController controller = loader.getController();
        controller.injectStage(stage);
        stage.setScene(scene);
        stage.show();
    }

    /**
     * Method which takes in an FXML file name and returns the FXMLLoader object
     * of that file.
     *
     * @param fxmlName the FXML file name.
     * @return FXMLLoader object for the specified file.
     * @throws IOException in case errors occur
     */
    public static FXMLLoader getWindow(String fxmlName) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource("fxml/" + fxmlName +
".fxml"));
        return fxmlLoader;
    }
}
```

Main.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

/**
 * The Main class of the program
 *
 * @author kxg708
 */
public class Main {
    /**
     * The main class of the program. It calls the launch() method which runs the
     * JavaFx program, by internally calling App.start();
     *
     * @param args list of arguments
     */
    public static void main(String[] args) {
        App.launch(App.class, args);
    }
}
```

AddNewAlimentWindowController.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import GeneticAlgorithm.Aliment;
import FileManager.FileManager;
import GeneticAlgorithm.Meal;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;
import javafx.fxml.Initializable;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ChoiceBox;
import javafx.stage.Stage;

/**
 * FXML Controller class for add new aliment window.
 *
 * @author kxg708
 */
public class AddNewAlimentWindowController implements Initializable {

    /**
```

```
     * Text Field for aliment name input.
     */
    @FXML private TextField nameTextField;

    /**
     * Text Field for aliment calorie per serving input.
     */
    @FXML private TextField calorieTextField;

    /**
     * Text Field for aliment serving size input.
     */
    @FXML private TextField servingSizeTextField;

    /**
     * Text Field for aliment carbohydrate (in grams) per serving input.
     */
    @FXML private TextField carbTextField;

    /**
     * Text Field for aliment fat (in grams) per serving input.
     */
    @FXML private TextField fatTextField;

    /**
     * Text Field for aliment protein (in grams) per serving input.
     */
    @FXML private TextField proteinTextField;

    /**
     * Choice box for serving size unit.
     */
    @FXML private ChoiceBox<String> unitChoiceBox;

    /**
     * Check box for snack meal tendency.
     */
    @FXML private CheckBox snackChckBx;

    /**
     * Check box for lunch meal tendency.
     */
    @FXML private CheckBox lunchChckBx;

    /**
     * Check box for dinner meal tendency.
     */
    @FXML private CheckBox dinnerChckBx;

    /**
```

```
     * Check box for breakfast meal tendency.
     */
    @FXML private CheckBox breakfastChckBx;

    /**
     * The stage representing the GUI window.
     */
    private Stage stage;

    /**
     * The array of units available in unitChoiceBox.
     * - gram
     * - milliliter
     * - teaspoon
     * - tablespoon
     * - amount
     * - glasses
     */
    private String[] units = {"گرم", "میلی لیتر", "قاشق چایخوری", "قاشق غذاخوری", "عدد", "لیوان"};

    /**
     * This method is run when the Window is first created. Here, the text fields
     * are turned numeric and the choice box options are added.
     *
     * @param url variable not used. Internally passed.
     * @param rb variable not used. Internally passed.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        Validation.createNumeric(calorieTextField);
        Validation.createNumeric(servingSizeTextField);

        unitChoiceBox.getItems().addAll(units);
    }

    /**
     * Method for controller to access its own stage.
     *
     * @param stage the GUI window.
     */
    public void injectStage(Stage stage) {
        this.stage = stage;
    }

    /**
     * Button event handler which creates the aliment and adds it to the aliment.data
     * file. First validation checks are in place to ensure the correct data is
     * being entered. The created aliment is added to the data file using
     * FileManager.addToAlimentDataFile() Once it is added, the method closes the
     * window.
```

```java
 */
@FXML
private void add() {
    Aliment aliment = new Aliment();
    List<Meal.TENDENCY> mealTendency = new ArrayList<>();

    if (Validation.isTxtFldEmpty(nameTextField) ||
        Validation.isTxtFldEmpty(servingSizeTextField) ||
        Validation.isTxtFldEmpty(fatTextField) ||
        Validation.isTxtFldEmpty(proteinTextField) ||
        Validation.isTxtFldEmpty(calorieTextField) ||
        Validation.isTxtFldEmpty(carbTextField)) {
        return;
    }

    if (unitChoiceBox.getSelectionModel().getSelectedItem() == null) {
        Validation.showWarningAlert("هیچ واحدی انتخاب نشده است");
        return;
    }

    if (breakfastChckBx.isSelected()) {
        mealTendency.add(Meal.TENDENCY.BREAKFAST);
    }
    if (lunchChckBx.isSelected()) {
        mealTendency.add(Meal.TENDENCY.LUNCH);
    }
    if (snackChckBx.isSelected()) {
        mealTendency.add(Meal.TENDENCY.SNACK);
    }
    if (dinnerChckBx.isSelected()) {
        mealTendency.add(Meal.TENDENCY.DINNER);
    }

    if (mealTendency.size() == 0) {
        Validation.showWarningAlert("این خوراکی مناسب چه وعده ها ای هست؟");
        return;
    }

    try {
        int calorieValue = Integer.parseInt(calorieTextField.getText());
        float proteinValue = Float.parseFloat(proteinTextField.getText());
        float carbValue = Float.parseFloat(carbTextField.getText());
        float fatValue = Float.parseFloat(fatTextField.getText());
        float calorieSum = proteinValue * 4f + carbValue * 4f + fatValue * 9f;
        // since each gram of protein has 4 calories
        // and each gram of carb has 4 calories
        // and each gram of fat has 9 calories

        // check whether the calorie goal matches the sum of the calories
        // of the macronutrients
```

```
            float calorieVariation = Math.abs( calorieValue - calorieSum ) / (float)
calorieValue;

            if (calorieVariation > 0.1) {
                Validation.showWarningAlert("کالری پروتئین، کربوهیدرات و چربی وارد شده با هدف کالری وارد شده
مطابقت ندارد!");
                return;
            }


            aliment.setName(nameTextField.getText());
            aliment.setCaloriePerServing(calorieValue);
            aliment.setServingSize(Float.parseFloat(servingSizeTextField.getText()));
            aliment.setServingUnit(unitChoiceBox.getSelectionModel().getSelectedItem());
            aliment.setCarbPerServing(carbValue);
            aliment.setFatPerServing(fatValue);
            aliment.setProteinPerServing(proteinValue);

            for (Meal.TENDENCY tendency : mealTendency) {
                aliment.addMealTendency(tendency);
            }


            FileManager.addToAlimentDataFile(aliment);
            stage.close();

        } catch (NumberFormatException e) {
            Validation.showWarningAlert("فقط ورودی اعداد مجاز است!");
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }
}
```

AlimentSelectorWindowController.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import javafx.fxml.FXML;
import javafx.scene.control.ListView;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import GeneticAlgorithm.Aliment;
import GeneticAlgorithm.Meal;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
```

```java
import java.util.Arrays;
import java.util.ResourceBundle;
import java.util.stream.Collectors;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.Label;
import javafx.scene.control.MenuItem;

/**
 * FXML Controller class for aliment selector window. When the window is created,
 * a list object is passed to this controller. The selected aliment is then added
 * to that list. Examples include the likes list view and the dislikes list view.
 *
 * @author kxg708
 */
public class AlimentSelectorWindowController implements Initializable {
    /**
     * ListView for search results
     */
    @FXML private ListView<Aliment> searchList;

    /**
     * Text Field for search input
     */
    @FXML private TextField searchField;

    /**
     * Text Field for number of servings input (only visible when servingSelectable
     * is true)
     */
    @FXML private TextField numOfServingsTxtField;

    /**
     * CheckBox for filtering breakfast aliments
     */
    @FXML private CheckBox breakfastFilterChckBx;

    /**
     * CheckBox for filtering dinner aliments
     */
    @FXML private CheckBox dinnerFilterChckBx;

    /**
     * CheckBox for filtering lunch aliments
     */
    @FXML private CheckBox lunchFilterChckBx;
```

```java
/**
 * CheckBox for filtering snack aliments
 */
@FXML private CheckBox snackFilterChckBx;

/**
 * GUI Label (only visible when servingSelectalbe is true.
 */
@FXML private Label servingSizeLbl;


/**
 * The list which contains the contents of searchList.
 */
private ObservableList<Aliment> observableList;

/**
 * The stage representing the GUI window.
 */
private Stage stage;

/**
 * A flag representing whether the selector can specify number of servings or
 * not. It is true when manually modifying the meal options.
 */
private boolean servingSelectable;

/**
 * This method is run when the Window is first created. Here, the global variable
 * App.aliments is added as the contents of the search list view. However, once
 * search and filter takes place, the observable list will be used. Moreover, a
 * context menu is added to the search list view and the servingSelectable flag is
 * automatically set to false.
 *
 * @param url variable not used. Internally passed.
 * @param rb variable not used. Internally passed.
 */
@Override
public void initialize(URL url, ResourceBundle rb) {
    this.servingSelectable = false;
    searchList.getItems().addAll(App.aliments);
    addContextMenu(searchList);
}

/**
 * Utility method which takes in a ListView and adds a JavaFX context menu to it.
 *
 * @param listView the ListView object to which the context menu should be added.
 */
```

```java
    private void addContextMenu(ListView<Aliment> listview) {
        ContextMenu contextMenu = new ContextMenu();

        MenuItem deleteItem = new MenuItem("حذف");
        deleteItem.setOnAction(e -> {
            Aliment selectedAliment = listview.getSelectionModel().getSelectedItem();
            listview.getItems().remove(selectedAliment);
            FileManager.FileManager.removeFromAlimentDataFile(selectedAliment);
        });

        MenuItem addItem = new MenuItem("خوراکی جدید");
        addItem.setOnAction(e -> {
            swtichToAddNewAliment();
        });

        MenuItem viewItemInfo = new MenuItem("اطلاعات خوراکی");
        viewItemInfo.setOnAction(e -> {
            try {
                Aliment selectedAliment = listview.getSelectionModel().getSelectedItem();

                Stage stage = new Stage();
                FXMLLoader window = App.getWindow("viewAlimentInfoWindow");

                Scene scene = new Scene(window.load());
                stage.setScene(scene);

                ViewAlimentInfoWindowController controller = window.getController();
                controller.injectAliment(selectedAliment);
                stage.show();

            } catch (Exception error) {
                Validation.showErrorAlert(error);
            }
        });

        contextMenu.getItems().addAll(deleteItem, addItem, viewItemInfo);

        listview.setContextMenu(contextMenu);
    }

    /**
     * Method which takes in a search and list of foods, and returns another list
     * which contains the matching aliments. This search code was inspired by:
     * https://youtu.be/VUVqamT8Npc?feature=shared
     *
     * @param searchWords the search prompt.
     * @param listOfFoods the list of aliments to search.
     * @return a list of aliments which match the search prompt.
     */
    private List<Aliment> searchList(String searchWords, List<Aliment> listOfFoods) {
        List<String> searchWordsArray = Arrays.asList(searchWords.split(" "));

        ArrayList<Meal.TENDENCY> filters = new ArrayList<>();
        if (breakfastFilterChckBx.isSelected()) filters.add(Meal.TENDENCY.BREAKFAST);
        if (lunchFilterChckBx.isSelected()) filters.add(Meal.TENDENCY.LUNCH);
        if (snackFilterChckBx.isSelected()) filters.add(Meal.TENDENCY.SNACK);
        if (dinnerFilterChckBx.isSelected()) filters.add(Meal.TENDENCY.DINNER);

        return listOfFoods.stream().filter(inputs -> {
            if (filters.isEmpty()) {
                return searchWordsArray.stream().allMatch(
                        word ->
inputs.getName().toLowerCase().contains(word.toLowerCase()));
            }
            for (Meal.TENDENCY filter : filters) {
                if (inputs.getMealTendency().contains(filter)) {
                    return searchWordsArray.stream().allMatch(
                        word ->
inputs.getName().toLowerCase().contains(word.toLowerCase()));
                }
            }
            return false;
        }).collect(Collectors.toList());
    }

    /**
     * Method to access the list to which the selected aliment should be added.
     * For instance, the list from likesListView or dislikesListView.
     *
     * @param list the list which is being injected.
     */
    public void injectObservableList(ObservableList<Aliment> list) {
        this.observableList = list;
    }

    /**
     * Method to set servingSelectable true or false. It also determines the
     * visibility of servingSizeLbl and numOsServingsTxtField
     *
     * @param selectable value of the servingSelectable Boolean.
     */
    public void isServingSelectable(boolean selectable) {
        this.servingSelectable = selectable;
        numOfServingsTxtField.setVisible(selectable);
        servingSizeLbl.setVisible(selectable);
    }

    /**
     * Method for controller to access its own stage.
     *
     * @param stage the GUI window.
     */
    public void injectStage(Stage stage) {
        this.stage = stage;
    }

    /**
     * Key press event handler which runs the search method and updates the searchList.
     * This method gets run each time a new key in the search prompt is pressed.
     */
    @FXML
    private void search() {
        searchList.getItems().clear();
        searchList.getItems().addAll(searchList(searchField.getText(), App.aliments));
    }

    /**
     * Button event handler which opens an add new aliment window.
     */
    @FXML
    private void swtichToAddNewAliment() {
        try {
            FXMLLoader loader = App.getWindow("addNewAlimentWindow");
            stage.getScene().setRoot(loader.load());
            AddNewAlimentWindowController controller = loader.getController();
            controller.injectStage(stage);


        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }

    /**
     * Button event handler which takes the selected aliment and adds it to the
     * injected observableList.
     */
    @FXML
    private void selectAliment() throws CloneNotSupportedException {
        Aliment selectedAliment = searchList.getSelectionModel().getSelectedItem();

        if (selectedAliment == null) {
            // translation: no aliment selected!
            Validation.showWarningAlert("هیچ خوراکی انتخاب نشده!");
        } else if (servingSelectable && numOfServingsTxtField.getText().isBlank()) {
            // translation: please enter the number of servings.
            Validation.showWarningAlert("لطفا تعداد وعده ها را وارد کنید!");
        } else {
            selectedAliment = (Aliment) selectedAliment.clone();
            if (servingSelectable) {
                try {

selectedAliment.setNumOfServings(Float.parseFloat(numOfServingsTxtField.getText()));
                } catch (NumberFormatException e) {
                    // translation: only number inputs are allowed.
                    Validation.showWarningAlert("فقط ورودی اعداد مجاز است!");
                    return;
                } catch (Exception e) {
                    Validation.showErrorAlert(e);
                    return;
                }
            }
            observableList.add(selectedAliment);

            stage.close();
        }
    }

}
```

DietCreationWindowController.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.ListView;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;
import GeneticAlgorithm.Aliment;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseButton;
import GeneticAlgorithm.Diet;
import GeneticAlgorithm.Meal;
import GeneticAlgorithm.Breakfast;
import GeneticAlgorithm.Snack;
import GeneticAlgorithm.Lunch;
import GeneticAlgorithm.Dinner;


/**
 * FXML Controller class for diet creation window.
```

```java
 *
 * @author kxg708
 */
public class DietCreationWindowController implements Initializable {
    /**
     * ListView for user Likes input.
     */
    @FXML private ListView<Aliment> likesListView;

    /**
     * ListView for user Dislikes input.
     */
    @FXML private ListView<Aliment> dislikesListView;

    /**
     * TextFields for calorie and macro-nutrient goals.
     */
    @FXML private TextField calorieTextField, proteinTextField, fatTextField,
carbTextField;

    /**
     * This method is run when the Window is first created. The global variables
     * App.likes and App.dislikes are set to the contents list of the listViews,
     * context menus are added to the list views, and the goals text fields are
     * turned numeric.
     *
     * @param url variable not used. Internally passed.
     * @param rb variable not used. Internally passed.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {

        likesListView.setItems(App.likes);
        dislikesListView.setItems(App.dislikes);

        addContextMenu(likesListView);
        addContextMenu(dislikesListView);

        Validation.createNumeric(calorieTextField);
        Validation.createNumeric(proteinTextField);
        Validation.createNumeric(carbTextField);
        Validation.createNumeric(fatTextField);
    }

    /**
     * Utility method which takes in a ListView and adds a FXML context menu to it.
     *
     * @param listView the ListView object to which the context menu should be added.
     */
    private void addContextMenu(ListView<Aliment> listview) {
        ContextMenu contextMenu = new ContextMenu();

        MenuItem deleteItem = new MenuItem("حذف");
        deleteItem.setOnAction(e -> {
            Aliment selectedAliment = listview.getSelectionModel().getSelectedItem();
            listview.getItems().remove(selectedAliment);
        });

        contextMenu.getItems().addAll(deleteItem);

        listview.setContextMenu(contextMenu);
    }

    /**
     * Mouse click event handler which displays the aliment selector window when
     * the left click mouse button is pressed on a list view (likes or dislikes).
     * It then injects the List which called it to the window controller.
     *
     * @param event variable containing information about the event, such as source.
     */
    @FXML
    private void switchToAlimentSelectorWindow(MouseEvent event) {

        try {
            if (event.getButton() == MouseButton.PRIMARY) {
                Stage alimentSelectionWindow = new Stage();
                FXMLLoader window = App.getWindow("alimentSelectorWindow");

                Scene scene = new Scene(window.load());

                AlimentSelectorWindowController controller = window.getController();

                ListView<Aliment> list = (ListView<Aliment>) event.getSource();

                controller.injectObservableList(list.getItems());
                controller.injectStage(alimentSelectionWindow);

                alimentSelectionWindow.setScene(scene);
                alimentSelectionWindow.show();
            }
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }

    /**
     * Button event handler which displays the automatic goals calculator window,
     * and injects the goals text fields to be populated.
     */
    @FXML
    private void switchToGoalsCalculatorWindow() {
        try {
            Stage goalsCalculatorStage = new Stage();
            FXMLLoader window = App.getWindow("goalsCalculatorWindow");

            Scene scene = new Scene(window.load());

            GoalsCalculatorWindowController controller = window.getController();
            controller.injectStage(goalsCalculatorStage);
            controller.injectFields(calorieTextField, proteinTextField, carbTextField,
fatTextField);
            goalsCalculatorStage.setScene(scene);
            goalsCalculatorStage.show();
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }

    }

    /**
     * Button event handler which gets a generated diet using {@link #generateDiet()}
     * and injects it to a created diet display window.
     */
    @FXML
    private void switchToDietDisplayWindow() {
        try {
            Diet diet = generateDiet();

            if (diet == null) return;

            Stage dietDisplayStage = new Stage();
            FXMLLoader window = App.getWindow("dietDisplayWindow");
            Scene scene = new Scene(window.load());

            DietDisplayWindowController controller = window.getController();
            controller.injectStage(dietDisplayStage);
            controller.injectDiet(diet);

            dietDisplayStage.setScene(scene);

            dietDisplayStage.show();
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }

    }

    /**
     * Method which creates 4 Meal objects and calls their run() methods on 4 separate
threads.
```

```
     * This generates 5 meal options for each Meal object. It then adds the Meals to a
Diet
     * object and returns it.
     *
     * @return the generated Diet object.
     */
    private Diet generateDiet() {

        if (!isCalorieTxtFldValid() || !areMacronutrientTxtFldsValid()) return null;

        int calorieGoal = Integer.parseInt(calorieTextField.getText());
        int proteinGoal = Integer.parseInt(proteinTextField.getText());
        int carbGoal = Integer.parseInt(carbTextField.getText());
        int fatGoal = Integer.parseInt(fatTextField.getText());


        Meal breakfastMeal = new Breakfast(calorieGoal, proteinGoal, carbGoal, fatGoal);
        Meal snackMeal = new Snack(calorieGoal, proteinGoal, carbGoal, fatGoal);
        Meal lunchMeal = new Lunch(calorieGoal, proteinGoal, carbGoal, fatGoal);
        Meal dinnerMeal = new Dinner(calorieGoal, proteinGoal, carbGoal, fatGoal);


        Thread breakfastThread = new Thread(breakfastMeal);
        Thread snackThread = new Thread(snackMeal);
        Thread lunchThread = new Thread(lunchMeal);
        Thread dinnerThread = new Thread(dinnerMeal);

        breakfastThread.start();
        snackThread.start();
        lunchThread.start();
        dinnerThread.start();

        try {
            breakfastThread.join();
            snackThread.join();
            lunchThread.join();
            dinnerThread.join();

            Diet diet = new Diet(breakfastMeal.getMealOptions(),
                                lunchMeal.getMealOptions(),
                                snackMeal.getMealOptions(),
                                dinnerMeal.getMealOptions());

            return diet;
        } catch (Exception e) {
            Validation.showErrorAlert(e);
            return null;
        }
    }
```

```java
    /**
     * Method which checks validation of the calorie text field.
     *
     * @return true if valid, false if invalid
     */
    private boolean isCalorieTxtFldValid() {
        if (Validation.isTxtFldEmpty(calorieTextField) ||
            !Validation.isTxtFldNumeric(calorieTextField)) {
            return false;
        }

        int calorieValue = Integer.parseInt(calorieTextField.getText());

        if (calorieValue <= 500) {
            Validation.showWarningAlert("کالری نمی تواند کمتر از 500 باشد");
            return false;
        }

        return true;
    }

    /**
     * Method which checks the validation of the macro-nutrient text fields.
     *
     * @return true if valid, false if invalid
     */
    private boolean areMacronutrientTxtFldsValid() {
        if (Validation.isTxtFldEmpty(proteinTextField) ||
            Validation.isTxtFldEmpty(carbTextField) ||
            Validation.isTxtFldEmpty(fatTextField) ||
            !Validation.isTxtFldNumeric(proteinTextField) ||
            !Validation.isTxtFldNumeric(carbTextField) ||
            !Validation.isTxtFldNumeric(fatTextField)) {
            return false;
        }


        int proteinValue = Integer.parseInt(proteinTextField.getText());
        int carbValue = Integer.parseInt(carbTextField.getText());
        int fatValue = Integer.parseInt(fatTextField.getText());

        // range check
        if (proteinValue <= 10 || carbValue <= 10 || fatValue <= 10) {
            Validation.showWarningAlert("پروتئین، کربوهیدرات و چربی نمی تواند کمتر از 10 گرم باشد");
            return false;
        }

        int calorieValue = Integer.parseInt(calorieTextField.getText());
        int calorieSum = proteinValue * 4 + carbValue * 4 + fatValue * 9;
        // since each gram of protein has 4 calories
        // and each gram of carb has 4 calories
        // and each gram of fat has 9 calories

        // check whether the calorie goal matches the sum of the calories
        // of the macronutrients
        float calorieVariation = Math.abs( calorieValue - calorieSum ) / (float)
calorieValue;

        if (calorieVariation > 0.1) {
            Validation.showWarningAlert(" کالری پروتئین، کربوهیدرات و چربی وارد شده با هدف کالری وارد شده مطابقت
ندارد!");
            return false;
        }

        return true;
    }
}
```

DietDisplayWindowController.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import javafx.fxml.FXML;
import javafx.stage.Stage;
import FileManager.FileManager;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.Initializable;
import javafx.scene.control.ListView;
import GeneticAlgorithm.MealOption;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.ListCell;
import javafx.scene.control.MenuItem;
import javafx.scene.text.Text;
import GeneticAlgorithm.Diet;
import java.io.File;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;

/**
 * FXML Controller class for diet display window.
 *
 * @author kxg708
 */
public class DietDisplayWindowController implements Initializable {
    /**
     * The stage representing the GUI window.
     */
    private Stage stage;

    /**
     * The Diet object being displayed
     */
    private Diet diet;

    /**
     * ListView for breakfast meal options
     */
    @FXML private ListView<MealOption> breakfast;

    /**
     * ListView for snack meal options
     */
    @FXML private ListView<MealOption> snack;

    /**
     * ListView for lunch meal options
     */
    @FXML private ListView<MealOption> lunch;

    /**
     * ListView for dinner meal options
     */
    @FXML private ListView<MealOption> dinner;

    /**
     * Button for saving diet file. It is disabled if the diet has been newly
     * generated and has not been previously saved before.
     */
    @FXML private MenuItem saveBtn;

    /**
     * This method is run when the Window is first created. Here, the meal
     * list views are set to cell wrap, and a JavaFX context menu is added
     * to them.
     *
     * @param url variable not used. Internally passed.
     * @param rb variable not used. Internally passed.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        setCellWrap(breakfast);
        setCellWrap(snack);
        setCellWrap(lunch);
        setCellWrap(dinner);

        addContextMenu(breakfast);
        addContextMenu(snack);
        addContextMenu(lunch);
        addContextMenu(dinner);
    }

    /**
     * Utility method which takes in a ListView and adds a JavaFX context menu to it.
     *
     * @param listView the ListView object to which the context menu should be added.
     */
    private void addContextMenu(ListView<MealOption> listview) {
        ContextMenu contextMenu = new ContextMenu();

        MenuItem editItem = new MenuItem("ویرایش");
        editItem.setOnAction(e -> {
            try {
                MealOption selectedMealOption =
listview.getSelectionModel().getSelectedItem();
                Stage mealOptionEditorWindow = new Stage();
                FXMLLoader window = App.getWindow("mealOptionEditorWindow");
                Scene scene = new Scene(window.load());


                MealOptionEditorWindowController controller = window.getController();
                controller.injectMealOption(selectedMealOption, listview);

                mealOptionEditorWindow.setScene(scene);
                mealOptionEditorWindow.show();
            } catch (Exception exception) {
                Validation.showErrorAlert(exception);
            }

        });

        MenuItem refreshItem = new MenuItem("بازسازی");
        refreshItem.setOnAction(e -> {
            try {
                MealOption selectedMealOption =
listview.getSelectionModel().getSelectedItem();

                selectedMealOption.run();
                listview.refresh();

            } catch (Exception exception) {
                Validation.showErrorAlert(exception);
            }

        });

        MenuItem deleteItem = new MenuItem("حذف");
```

```java
        deleteItem.setOnAction(e -> {
            MealOption selectedAliment = listview.getSelectionModel().getSelectedItem();
            listview.getItems().remove(selectedAliment);
        });

        contextMenu.getItems().addAll(editItem, deleteItem, refreshItem);

        listview.setContextMenu(contextMenu);
    }

    /**
     * Method to make the cells of a ListView to wrap when the text overflows.
     *
     * AI (ChatGPT) generated:
     * prompt: I have a listView in JavaFX, and I need a way to make the cells
     * wrap to the next line when the text overflows
     *
     * @param listView the listView to on which to set this property
     */
    private void setCellWrap(ListView<MealOption> listView) {
        listView.setCellFactory(param -> new ListCell<MealOption>() {
            private Text text;
            {
                text = new Text();
                text.wrappingWidthProperty().bind(listView.widthProperty().subtract(50));
            }
            @Override
            protected void updateItem(MealOption item, boolean empty) {
                super.updateItem(item, empty);

                if (empty || item == null) {
                    setGraphic(null);
                } else {
                    text.setText(item.toString());
                    setGraphic(text);
                }
            }
        });
    }

    /**
     * Method for controller to access its own stage.
     *
     * @param stage the GUI window.
     */
    public void injectStage(Stage stage) {
        this.stage = stage;
    }

    /**
```

```
     * Method to inject the diet that is being displayed, into controller.
     *
     * @param diet diet being displayed by the window.
     */
    public void injectDiet(Diet diet) {
        this.diet = diet;
        breakfast.getItems().addAll(diet.getBreakfast());
        snack.getItems().addAll(diet.getSnack());
        lunch.getItems().addAll(diet.getLunch());
        dinner.getItems().addAll(diet.getDinner());

        updateWindowOnDietSave();
    }

    /**
     * Method for updating the window title and toggling the saveBtn availability
     */
    private void updateWindowOnDietSave() {
        if (diet.filename.equals("undefined")) {
            stage.setTitle("diet display window: undefined");
            saveBtn.setDisable(true);
        } else {
            stage.setTitle("diet display window: " + diet.filename);
            saveBtn.setDisable(false);
        }
    }

    /**
     * Button event handler which closes the window
     */
    @FXML
    private void close() {
        stage.close();
    }

    /**
     * Button event handler which saves the diet in a new diet file.
     */
    @FXML
    private void saveAsDiet() {
        FileManager.saveAsDietFile(diet);
        updateWindowOnDietSave();
    }

    /**
     * Button event handler which saves modifications to a previously
     * saved diet file.
     */
    @FXML
    private void save() {
```

```java
        try {
            File file = new File(diet.path);
            FileManager.saveDietFile(file, diet);
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }

    }

    /**
     * Button event handler which saves the diet as a PDF file.
     */
    @FXML
    private void saveAsPDF() {
        FileManager.saveAsPDF(diet);
    }
}
```

GoalsCalculatorWindowController.java

Click here for Javadoc related to this file.

```java
package com.mycompany.dietgenerator;


import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.stage.Stage;

/**
 * FXML Controller class goals calculator window. The class takes in
 * the goals text fields and populates them with the calculated values.
 *
 * for sedentary: little to no exercise multiply by a factor of 1.2
 * for light exercise multiply by a factor of 1.375
 * for moderate exercise multiply by a factor of 1.46
 * for active exercise multiply by a factor of 1.55
 * for very active multiply by a factor of 1.75
 * for extra active multiply by a factor of 1.9
 *
 *
 * for mild weight gain: +250 calories
 * for weight gain: +500 calories
 * for extreme weight gain: +1000 calories
 *
 * for mild weight loss: -250 calories
 * for weight loss: -500 calories
 * for extreme weight loss: -1000 calories
 *
 * @author kxg708
 */
public class GoalsCalculatorWindowController implements Initializable {
    /**
     * TextField for height.
     */
    @FXML private TextField heightTextField;

    /**
     * TextField for age.
     */
    @FXML private TextField ageTextField;

    /**
     * TextField for weight.
     */
    @FXML private TextField weightTextField;

    /**
     * RadioButton for light activity level.
     */
    @FXML private RadioButton rBtnActivityLvlLight;

    /**
     * RadioButton for moderate activity level.
     */
    @FXML private RadioButton rBtnActivityLvlModerate;

    /**
     * RadioButton for active activity level.
     */
    @FXML private RadioButton rBtnActivityLvlActive;

    /**
     * RadioButton for female gender
     */
    @FXML private RadioButton rBtnFemale;

    /**
     * RadioButton for male gender
     */
    @FXML private RadioButton rBtnMale;

    /**
     * ToggleGroup for activity.
     */
    @FXML private ToggleGroup activity;

    /**
     * ToggleGroup for gender.
     */
    @FXML private ToggleGroup gender;

    /**
     * ToggleGroup for goals.
     */
    @FXML private ToggleGroup goals;

    /**
     * RadioButton for extreme weight loss goal.
     */
    @FXML private RadioButton rBtnExtremeWeightLoss;

    /**
     * RadioButton for mild weight loss goal.
     */
    @FXML private RadioButton rBtnMildWeightLoss;

    /**
     * RadioButton for weight loss goal.
     */
    @FXML private RadioButton rBtnWeightLoss;

    /**
     * RadioButton for weight maintenance goal.
     */
    @FXML private RadioButton rBtnWeightMaintenance;

    /**
     * RadioButton for extreme weight gain goal.
     */
    @FXML private RadioButton rBtnExtremeWeightGain;

    /**
     * RadioButton for mild weight gain goal.
     */
    @FXML private RadioButton rBtnMildWeightGain;

    /**
     * RadioButton for weight gain goal.
     */
    @FXML private RadioButton rBtnWeightGain;

    /**
     * The stage representing the GUI window.
     */
    private Stage stage;

    /**
     * TextField for calorie input to be populated.
     */
    private TextField calorieTextField;

    /**
     * TextField for carbohydrate input to be populated.
     */
    private TextField carbTextField;

    /**
     * TextField for protein input to be populated.
     */
    private TextField proteinTextField;

    /**
     * TextField for fat input to be populated.
     */
    private TextField fatTextField;

    /**
     * Variable for calculated calorie goal.
     */
    private int calorieGoal;

    /**
     * Variable for calculated protein goal.
     */
    private float proteinGoal;

    /**
     * Variable for calculated carbohydrate goal.
     */
    private float carbGoal;

    /**
     * Variable for calculated fat goal.
     */
    private float fatGoal;

    /**
     * This method is run when the Window is first created. Here,
     * the height, age, and weight text fields are turned numeric.
     *
     * @param url variable not used. Internally passed.
     * @param rb variable not used. Internally passed.
     */
```

```java
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        Validation.createNumeric(heightTextField);
        Validation.createNumeric(ageTextField);
        Validation.createNumeric(weightTextField);
    }
    /**
     * Method for controller to access its own stage.
     *
     * @param stage the GUI window.
     */
    public void injectStage(Stage stage) {
        this.stage = stage;
    }

    /**
     * Method for controller to access goal Text Fields.
     *
     * @param calorieTextField the calorie Text Field to be populated.
     * @param proteinTextField the protein Text Field to be populated.
     * @param carbTextField the carbohydrate Text Field to be populated.
     * @param fatTextField the fat Text Field to be populated.
     */
    public void injectFields(TextField calorieTextField, TextField proteinTextField,
TextField carbTextField, TextField fatTextField) {
        this.calorieTextField = calorieTextField;
        this.proteinTextField = proteinTextField;
        this.carbTextField = carbTextField;
        this.fatTextField = fatTextField;
    }

    /**
     * Method which uses specified text and radio button
     * inputs and calculates the calorie goal, and saves it
     * in this.calorieGoal.
     */
    private void calcCalorieGoal() {
        float bmrConst, activityConst, goalConst;

        if (rBtnMale.isSelected()) {
            bmrConst = 5;
        } else if (rBtnFemale.isSelected()) {
            bmrConst = -161;
        } else {
            // translation: please fill out all fields!
            Validation.showWarningAlert("لطفا تمام اطلاعات را پر کنید!");
            return;
        }
        // activity constants obtained from https://www.calculator.net/calorie-
calculator.html
        if (rBtnActivityLvlLight.isSelected()) {
            activityConst = 1.375f;
        } else if (rBtnActivityLvlModerate.isSelected()) {
            activityConst = 1.46f;
        } else if (rBtnActivityLvlActive.isSelected()) {
            activityConst = 1.55f;
        } else {
            // translation: please fill out all fields!
            Validation.showWarningAlert("لطفا تمام اطلاعات را پر کنید!");
            return;
        }

        // goal constants obtained from https://www.calculator.net/calorie-calculator.html
        if (rBtnExtremeWeightLoss.isSelected()) {
            goalConst = -1000;
        } else if (rBtnWeightLoss.isSelected()) {
            goalConst = -500;
        } else if (rBtnMildWeightLoss.isSelected()) {
            goalConst = -250;
        } else if (rBtnWeightMaintenance.isSelected()) {
            goalConst = 0;
        } else if (rBtnMildWeightGain.isSelected()) {
            goalConst = 250;
        } else if (rBtnWeightGain.isSelected()) {
            goalConst = 500;
        } else if (rBtnExtremeWeightGain.isSelected()) {
            goalConst = 1000;
        } else {
            // translation: please fill out all fields!
            Validation.showWarningAlert("لطفا تمام اطلاعات را پر کنید!");
            return;
        }

        float height = Float.parseFloat(heightTextField.getText());
        float age = Float.parseFloat(ageTextField.getText());
        float weight = Float.parseFloat(weightTextField.getText());

        float BMR = 10.0f*weight + 6.25f*height - 5.0f*age + bmrConst;

        float dailyCalorieIntake = BMR * activityConst;
        calorieGoal = Math.round(dailyCalorieIntake + goalConst);
    }

    /**
     * Method which uses specified text and radio button inputs and calculates
     * the macro-nutrient goals, and saves it in the appropriate data members.
     */
    private void calcMacroGoals() {
        float weight = Float.valueOf(weightTextField.getText());
        if (rBtnExtremeWeightLoss.isSelected()) {
            proteinGoal = weight * 2.2f * 1.2f;
        } else if (rBtnMildWeightLoss.isSelected()) {
            proteinGoal = weight * 2.2f * 1.2f;
        } else if (rBtnWeightLoss.isSelected()) {
            proteinGoal = weight * 2.2f * 1f;
        } else if (rBtnWeightMaintenance.isSelected()) {
            proteinGoal = weight * 2.2f * 1f;
        } else if (rBtnMildWeightGain.isSelected()) {
            proteinGoal = weight * 2.2f * 0.9f;
        } else if (rBtnWeightGain.isSelected()) {
            proteinGoal = weight * 2.2f * 0.8f;
        } else if (rBtnExtremeWeightGain.isSelected()) {
            proteinGoal = weight * 2.2f * 0.8f;
        }
        fatGoal = weight * 2.2f * 0.6f;
        carbGoal = (calorieGoal - proteinGoal*4f - fatGoal*9f) / 4f;
    }

    /**
     * Button event handler which first validates the inputs and then gets
     * the calculated goals, and populates the injected Text Fields. It then
     * closes the window.
     */
    @FXML
    private void calculate() throws IOException {
        if (Validation.isTxtFldEmpty(heightTextField) ||
            Validation.isTxtFldEmpty(ageTextField) ||
            Validation.isTxtFldEmpty(weightTextField) ||
            !Validation.isTxtFldNumeric(heightTextField) ||
            !Validation.isTxtFldNumeric(ageTextField) ||
            !Validation.isTxtFldNumeric(weightTextField) ||
            !Validation.isRdBtnSelected(activity) ||
            !Validation.isRdBtnSelected(goals) ||
            !Validation.isRdBtnSelected(gender)) {
            return;
        }

        try {
            calcCalorieGoal();
            calcMacroGoals();

            calorieTextField.setText(String.valueOf(calorieGoal));
            proteinTextField.setText(String.valueOf(Math.round(proteinGoal)));
            fatTextField.setText(String.valueOf(Math.round(fatGoal)));
            carbTextField.setText(String.valueOf(Math.round(carbGoal)));
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
        stage.close();
    }
}
```

MealOptionEditorWindowController.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import GeneticAlgorithm.MealOption;
import GeneticAlgorithm.Aliment;
import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.ResourceBundle;
import javafx.collections.ListChangeListener;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;

/**
 * FXML Controller class for meal option editor window. Meal option modification
 * is done by modifying a table of aliments, and then adding that table's items
 * to the meal option whenever the table is modified.
 *
 * @author kxg708
 */
public class MealOptionEditorWindowController implements Initializable {
    /**
     * The meal option being edited.
     */
    private MealOption mealOption;

    /**
     * The ListView from which the meal option came from.
     */
    private ListView<MealOption> mealOptionListView;

    /**
     * Table which shows the information of each aliment in the meal option.
     */
    @FXML private TableView<Aliment> table;

    /**
     * Calorie per serving column in table
     */
    @FXML private TableColumn<Aliment, Integer> calorieColumn;

    /**
     * Fat per serving (in grams) column in table
     */
    @FXML private TableColumn<Aliment, Float> fatColumn;

    /**
     * Name column in table
     */
    @FXML private TableColumn<Aliment, Float> nameColumn;

    /**
     * Number of servings column in table
     */
    @FXML private TableColumn<Aliment, Float> numOfServingsColumn;

    /**
     * Protein per serving (in grams) column in table
     */
    @FXML private TableColumn<Aliment, Float> proteinColumn;

    /**
     * Carbohydrate per serving (in grams) column in table
     */
    @FXML private TableColumn<Aliment, Float> carbColumn;

    /**
     * Label showing total calories of Meal Option
     */
    @FXML private Label totalCaloriesLabel;

    /**
     * Label showing total carbohydrates (in grams) of Meal Option
     */
    @FXML private Label totalCarbLabel;

    /**
     * Label showing total fat (in grams) of Meal Option
     */
    @FXML private Label totalFatLabel;

    /**
     * Label showing total protein (in grams) of Meal Option
     */
    @FXML private Label totalProteinLabel;

    /**
     * This method is run when the Window is first created. In this method
     * the table columns are set using setCellValueFactory
     *
     * @param url variable not used. Internally passed.
     * @param rb variable not used. Internally passed.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));
        calorieColumn.setCellValueFactory(new
PropertyValueFactory<>("caloriePerServing"));
        proteinColumn.setCellValueFactory(new
PropertyValueFactory<>("proteinPerServing"));
        fatColumn.setCellValueFactory(new PropertyValueFactory<>("fatPerServing"));
        numOfServingsColumn.setCellValueFactory(new
PropertyValueFactory<>("numOfServings"));
        carbColumn.setCellValueFactory(new PropertyValueFactory<>("carbPerServing"));
    }

    /**
     * Method which injects the meal option being edited and the list view from which
     * the meal option was taken, so that the list view can be updated as well.
     *
     * @param mealOption the MealOption object being manually modified.
     * @param mealOptionListView the ListView from which the meal option was taken.
     */
    public void injectMealOption(MealOption mealOption, ListView<MealOption>
mealOptionListView) {
        this.mealOption = mealOption;
        this.mealOptionListView = mealOptionListView;
        table.getItems().addAll(mealOption.getAliments());

        updateTotalLabels();

        // AI (ChatGPT) generated
        table.getItems().addListener((ListChangeListener.Change<? extends Aliment> change)
-> {
            updateTotalLabels();
            updateMealOption();
        });
    }

    /**
     * Method which calculates the sum of the calorie, protein, carbohydrate, and fat
     * contents of the meal option by multiplying the number of servings of each aliment
     * by the values per serving, and updates the labels with these values.
     */
```

```java
    private void updateTotalLabels() {
        double calorieSum = 0;
        double proteinSum = 0;
        double carbSum = 0;
        double fatSum = 0;

        for (Aliment aliment : table.getItems()) {
            calorieSum += aliment.getCaloriePerServing() * aliment.getNumOfServings();
            proteinSum += aliment.getProteinPerServing() * aliment.getNumOfServings();
            carbSum += aliment.getCarbPerServing() * aliment.getNumOfServings();
            fatSum += aliment.getFatPerServing() * aliment.getNumOfServings();
        }

        totalCaloriesLabel.setText("کالری کل: " + Math.round(calorieSum));
        totalProteinLabel.setText("پروتئین کل: " + Math.round(proteinSum) + " گرم");
        totalCarbLabel.setText("کربوهیدرات کل: " + Math.round(carbSum) + " گرم");
        totalFatLabel.setText("چربی کل: " + Math.round(fatSum) + " گرم");
    }

    /**
     * Button event handler which removes an aliment from the meal option.
     */
    @FXML
    private void removeAliment() {
        Aliment selectedAliment = table.getSelectionModel().getSelectedItem();
        table.getItems().remove(selectedAliment);
    }

    /**
     * Button event handler which opens an aliment selector with the serving selectable
     * flag set to true, and adds the selected aliment to the meal option. It also passes
     * the table.getItems() list to the aliment selector controller.
     */
    @FXML
    private void addAliment() throws IOException {
        FXMLLoader window = App.getWindow("alimentSelectorWindow");
        Stage stage = new Stage();
        Scene scene = new Scene(window.load());
        stage.setScene(scene);
        AlimentSelectorWindowController controller = window.getController();
        controller.isServingSelectable(true);
        controller.injectObservableList(table.getItems());
        controller.injectStage(stage);
        stage.show();
    }

    /**
     * Utility function which takes the modified list of aliment, and adds it to the meal
option.
     * It also refreshes the mealOption list view so changes are displayed.
     */
    private void updateMealOption() {
        mealOption.addAll(new ArrayList<>(table.getItems()));
        mealOptionListView.refresh();
    }

}
```

StartingWindowController.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import javafx.fxml.FXML;
import FileManager.FileManager;
import GeneticAlgorithm.Diet;
import java.io.IOException;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 * FXML Controller class for Starting window.
 *
 * @author kxg708
 */
public class StartingWindowController {
    /**
     * The stage representing the GUI window.
     */
    private Stage stage;

    /**
     * Method for controller to access its own stage.
     *
     * @param stage the GUI window.
     */
    public void injectStage(Stage stage) {
        this.stage = stage;
    }

    /**
     * Method which takes in a Diet object and displays it in a new diet
     * display window.
     *
     * @param diet the Diet object to be displayed in window.
     */
    private void switchToDietDisplayWindow(Diet diet) throws IOException {
        Stage dietDisplayStage = new Stage();
        FXMLLoader window = App.getWindow("dietDisplayWindow");

        Scene scene = new Scene(window.load());
        DietDisplayWindowController controller = window.getController();
        controller.injectStage(dietDisplayStage);
        controller.injectDiet(diet);
        dietDisplayStage.setScene(scene);
        dietDisplayStage.show();
    }

    /**
     * Button event handler which displays the diet creation window.
     */
    @FXML
    private void swtichToDietCreationWindow() {
        try {
            Stage dietCreationStage = new Stage();
            FXMLLoader window = App.getWindow("dietCreationWindow");

            Scene scene = new Scene(window.load());
            dietCreationStage.setScene(scene);
            dietCreationStage.show();

            stage.close();
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }
    }

    /**
     * Button event handler which opens a diet file using file manager and displays it
     * in diet display window using {@link
#switchToDietDisplayWindow(GeneticAlgorithm.Diet)}
     */
    @FXML
    private void openADietFile() {
        try {
            Diet diet = FileManager.openDietFile();

            // safety check
            if (diet == null) return;

            switchToDietDisplayWindow(diet);
            stage.close();
        } catch (Exception e) {
            Validation.showErrorAlert(e);
        }

    }
```

```
}
```

ViewAlimentInfoWindowController.java

Click here for Javadoc related to this file.

```java
package com.mycompany.dietgenerator;

import javafx.fxml.FXML;
import javafx.scene.chart.PieChart;
import GeneticAlgorithm.Aliment;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.Label;

/**
 * FXML Controller class for viewing aliment information window.
 *
 * @author kxg708
 */
public class ViewAlimentInfoWindowController {
    /**
     * Pie chart for showing aliment macro-nutrients.
     */
    @FXML private PieChart pieChart;

    /**
     * Label for showing aliment carbohydrates (in grams) per serving
     */
    @FXML private Label carbLbl;

    /**
     * Label for showing aliment protein (in grams) per serving
     */
    @FXML private Label proteinLbl;

    /**
     * Label for showing aliment fat (in grams) per serving
     */
    @FXML private Label fatLbl;

    /**
     * Label for showing aliment name
     */
    @FXML private Label nameLbl;

    /**
     * Label for showing size of each serving of aliment
     */
    @FXML private Label servingSizeLbl;

    /**
     * Label for showing aliment calorie per serving
     */
    @FXML private Label caloriesPerServingLbl;

    /**
     * Method for controller to access the aliment whose information is being displayed.
     *
     * @param aliment the aliment whose information is being displayed.
     */
    public void injectAliment(Aliment aliment) {
        // safety check
        if (aliment == null) return;

        updateLabels(aliment);
        updatePieChart(aliment);
    }

    /**
     * Updates the pie chart data given an aliment
     *
     * @param aliment the aliment whose information the pie chart shows.
     */
    private void updatePieChart(Aliment aliment) {
        ObservableList<PieChart.Data> chartData = FXCollections.observableArrayList(
                new PieChart.Data("پروتئین", aliment.getProteinPerServing() * 4),
                new PieChart.Data("کرب", aliment.getCarbPerServing() * 4),
                new PieChart.Data("چربی", aliment.getFatPerServing() * 9)
        );

        pieChart.setData(chartData);
    }

    /**
     * Updates the labels with the given aliment information.
     *
     * @param aliment the aliment whose information the labels should show.
     */
    private void updateLabels(Aliment aliment) {
        nameLbl.setText("اسم خوراکی: " + aliment.getName());
        proteinLbl.setText("پروتئین: " + String.valueOf(aliment.getProteinPerServing()) + "
گرم");
        fatLbl.setText("چربی" + String.valueOf(aliment.getFatPerServing()) + " گرم");
        carbLbl.setText("کرب: " + String.valueOf(aliment.getCarbPerServing()) + " گرم");
        servingSizeLbl.setText("اندازه هر واحد: " + String.valueOf(aliment.getServingSize())
                                + " " + aliment.getServingUnit());
        caloriesPerServingLbl.setText("کالری هر واحد: " + aliment.getCaloriePerServing());
    }
```

```
}
```

Validation.java

Click here for Javadoc related to this file.

```
package com.mycompany.dietgenerator;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.scene.control.Alert;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;


/**
 * Static class containing methods for validation of inputs and error handling.
 *
 * @author kxg708
 */
public class Validation {

    /**
     * Method which creates a pop up which displays error message
     *
     * @param e the Exception object which contains stackTrace and message.
     */
    public static void showErrorAlert(Exception e) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("error");
        alert.setHeaderText("خطایی رخ داد");
        alert.setContentText(e.getMessage());

        e.printStackTrace();
        alert.showAndWait();
    }

    /**
     * Method which creates a warning pop up with custom message
     *
     * @param message the custom message to be displayed
     */
    public static void showWarningAlert(String message) {
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.setTitle("warning");
        alert.setHeaderText("هشدار!");
        alert.setContentText(message);

        alert.showAndWait();
    }

    /**
     * Method which takes a TextField and adds a property which only allows numeric input.
     * Taken from source: https://stackoverflow.com/questions/7555564/what-is-the-
recommended-way-to-make-a-numeric-textfield-in-javafx
     *
     * @param txtFld the TextField to which the property is being added
     */
    public static void createNumeric(TextField txtFld) {
        txtFld.textProperty().addListener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
                if (!newValue.matches("\\d*")) {
                    txtFld.setText(newValue.replaceAll("[^\\d*]", ""));
                }
            }
        });
    }

    /**
     * Method which takes a Text Field and checks whether it is empty. If it is, a
     * warning is displayed.
     *
     * @param txtFld The Text Field being checked.
     * @return true if empty, and false if contains text.
     */
    public static boolean isTxtFldEmpty(TextField txtFld) {
        if (txtFld.getText().isBlank()) {
            // translation of text: please fill out all fields
            showWarningAlert("لطفا تمام ورودی ها را پر کنید!");
            return true;
        }
        return false;
    }

    /**
     * Method which takes a Text Field and checks whether it contains numeric values.
     * If it isn't, a warning is displayed.
     *
     * @param txtFld The Text Field being checked.
     * @return true if numeric, and false if contains text.
     */
    public static boolean isTxtFldNumeric(TextField txtFld) {
        if (!txtFld.getText().matches("\\d*")) {
            // translation of text: only numbers must be entered
            showWarningAlert("فقط ورودی اعداد مجاز است!");
            return false;
        }
```

```java
        return true;
    }

    /**
     * Method which checks if a radio button from an inputted Toggle Group has
     * been selected or not. If no button is selected a warning is displayed.
     *
     * @param group Toggle Group being checked
     * @return true if a button is selected, and false if no radio button is selected
     */
    public static boolean isRdBtnSelected(ToggleGroup group) {
        RadioButton selected = (RadioButton) group.getSelectedToggle();
        if (selected == null) {
            // translation of text: please select an option!
            showWarningAlert("لطفا یک گزینه را انتخاب کنید!");
            return false;
        }
        return true;
    }
}
```