

## Criterion C: Development

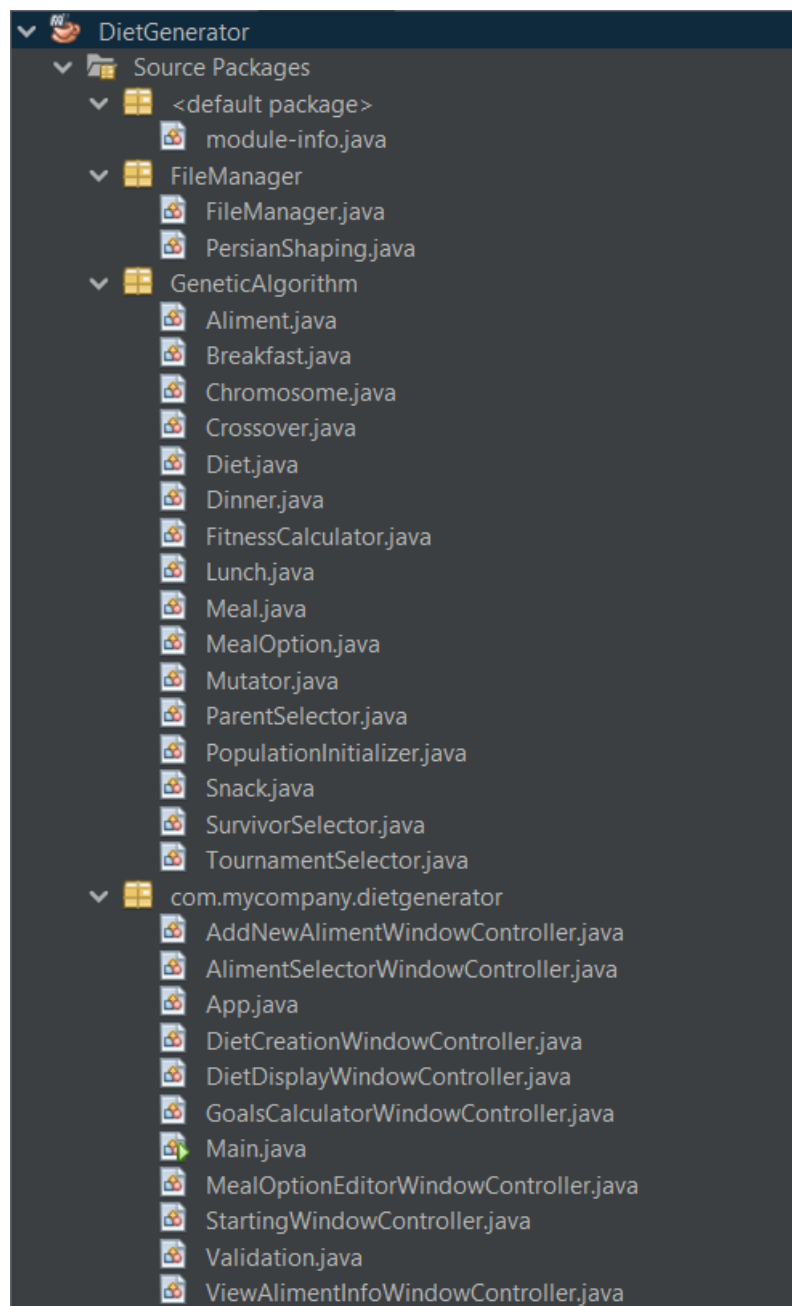
### Contents

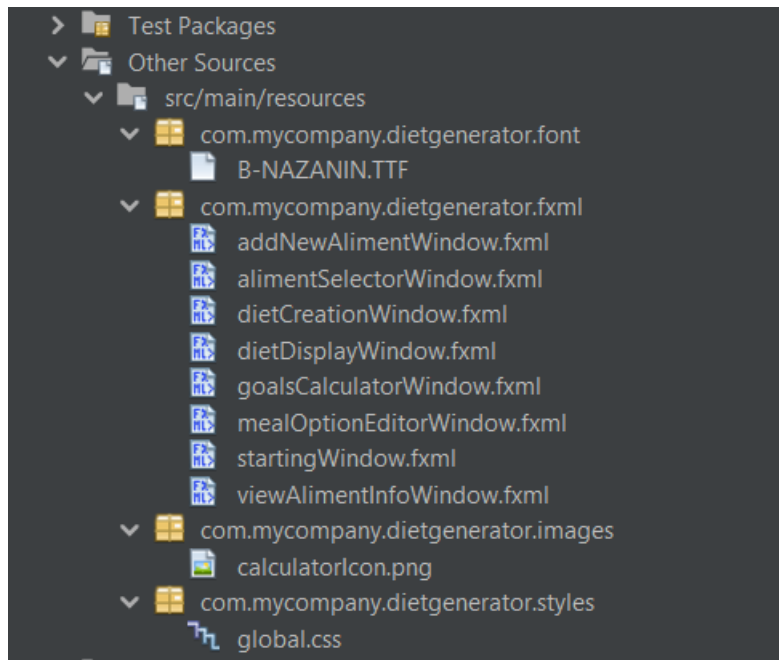
Overview .....	2
Project structure (Object-Oriented Programming).....	2
Main Libraries Used .....	5
Global Variables .....	6
GUI .....	6
Scene Bulider .....	6
Controllers and Event Handling .....	7
Validation and error handling .....	10
File Manager .....	12
Aliments.data file .....	13
Diet Files.....	13
Printing to PDF .....	16
Genetic Algorithm .....	18
Population Initializer .....	21
Fitness calculation.....	22
K-way tournament selection.....	23
Crossover .....	25
Mutator .....	26
Multithreading .....	27
Citations .....	29

## Overview

### Project structure (Object-Oriented Programming)

To develop this project, an Object-Oriented Programming style was utilized, as Java is an OOP language. OOP concepts such as Encapsulation, Abstraction, and Inheritance, also provide modularization and logical separation of code, which makes the product easier to develop and modify in the future. hence, related code is organized into separate files and packages (Figure 1).





*Figure 1. Project folder structure*

Encapsulation was properly used throughout the code. One great example is the *FitnessCalculator* class, which also provides abstraction by hiding internal methods and attributes. (Figure 2)

```

public class FitnessCalculator implements Serializable {
    /** Coefficient for adding weight to calorie constraint ...4 lines */
    private static final float CALORIE_COEFF = 5f;

    /** Coefficient for adding weight to protein constraint ...4 lines */
    private static final float PROTEIN_COEFF = 10f;

    /** Coefficient for adding weight to fat constraint ...4 lines */
    private static final float FAT_COEFF = 10f;

    /** Coefficient for adding weight to carbohydrate constraint ...4 lines */
    private static final float CARB_COEFF = 10f;

    /** Coefficient for adding weight to preference constraint ...4 lines */
    private static final float PREFERENCE_COEFF = 10f;

    /** Coefficient for adding weight to meal tendency constraint ...4 lines */
    private static final float MEAL_TENDENCY_COEFF = 100f;

    /** An arbitrary step value to increase the fitness or decrease the fitness ...4 lines */
    private static final int PREFERENCE_STEP = 5;

    /** An arbitrary step value to increase the decrease the fitness based on meal ...4 lines */
    private static final int MEAL_TENDENCY_STEP = 5;

    /** The meal that this object is being used for ...4 lines */
    private Meal.TENDENCY mealTendency;

    /** The number of calories each meal option should be ...3 lines */
    private int calorieConstraint;

    /** The amount of protein (in grams) each meal option should be ...3 lines */
    private float proteinConstraint;

    /** The amount of carbohydrates (in grams) each meal option should be ...3 lines */
    private float carbConstraint;

    /** The amount of fat (in grams) each meal option should be ...3 lines */
    private float fatConstraint;

    /** The Constructor of the FitnessCalculator class ...11 lines */
    public FitnessCalculator(int calorieConstraint, float proteinConstraint, float carbConstraint,

    /** Calculates the difference (distance) between the protein of the aliments ...7 lines */
    private float calculateProteinVariation(Aliment[] aliments) {...8 lines }

```

Figure 2. Encapsulation and Abstraction in FitnessCalculator class

Moreover, inheritance was also used to reduce code redundancy. The K-way Tournament selector algorithm is a great example of this. Both *ParentSelector* and *SurvivorSelector* implement this algorithm, but in slightly different ways. Hence, they share similar code through the parent class. (Figure 3)

```

public class TournamentSelector implements Serializable {
    /** It is the population object within the genetic algorithm ...3 lines */
    protected Chromosome[] population;

    /** The number of solutions to select in each round of the tournament selection algorithm ...4 lines */
    private int k;

    /** The constructor of the TournamentSelector algorithm, which is called by the main class ...4 lines */
    public TournamentSelector(Chromosome[] population, int k) {...4 lines }

    /** This method selects k solutions from the population ...5 lines */
    protected Chromosome[] selectKChromosomes() {...12 lines }
}

```

```

public class SurvivorSelector extends TournamentSelector implements Serializable {
    /** Constructor for the SurvivorSelector class which takes in the population ...4 lines */
    public SurvivorSelector(Chromosome[] population, int k) {
        super(population, k);
    }

    /** This method finds the loser of one round of the tournament selection algorithm ...14 lines */
    private Chromosome findLoser(Chromosome[] chromosomes) {...14 lines }

    /** This method takes an array of offspring and runs the tournament selector algorithm ...16 lines */
    public void replace(Chromosome[] offsprings) {...16 lines }
}

```

```

public class ParentSelector extends TournamentSelector implements Serializable {
    /** Constructor for the ParentSelector class which takes in the population ...4 lines */
    public ParentSelector(Chromosome[] population, int k) {
        super(population, k);
    }

    /** This method finds the winner of one round of the tournament selection algorithm ...14 lines */
    private Chromosome findWinner(Chromosome[] chromosomes) {...14 lines }

    /** This method runs the tournament selector algorithm a number of times to get the winners ...11 lines */
    public Chromosome[] getWinners(int numOfWinners) {...11 lines }
}

```

*Figure 3. inheritance in ParentSelector and SurvivorSelector classes*

## Main Libraries Used

- **JavaFX**<sup>1</sup> - Modern GUI framework which also works with Scenebuilder to efficiently create UI and add CSS styling
- **Apache PDFBox**<sup>2</sup> - Library for printing diets to PDF

<sup>1</sup> <https://openjfx.io/>

<sup>2</sup> <https://pdfbox.apache.org/>

- **ICU4J<sup>3</sup>** - An Arabic shaping library to fix issues when printing Persian letters to PDF
- **java.util.Random** – Mostly used in genetic algorithm to create element of chance.  
Ex. in mutation and crossover
- **java.io.Serializable** – Saving aliment and diet objects to datafiles.

## Global Variables

The project contains 3 important global variables which can be accessed throughout the program.

```
/** The list of aliments which can be accessed globally ...3 lines */
public static List<Aliment> aliments = new ArrayList<>();

/** The list of user liked aliments which can be accessed globally ...3 lines */
public static ObservableList<Aliment> likes = FXCollections.observableArrayList();

/** The list of user disliked aliments which can be accessed globally ...3 lines */
public static ObservableList<Aliment> dislikes = FXCollections.observableArrayList();
```

*Figure 4. Global variables in App.java*

- **Aliments:** list of all aliments. It is retrieved from the aliments.data file when program is launched (see File Manager).
- **Likes:** list of aliments that user likes.
- **Dislikes** is the list of aliments that user dislikes.

## GUI

### Scene Bulider

For developing the GUI, a drag and drop software called SceneBuilder<sup>4</sup> was used which saved each window as fxml file (Figure 5). For consistent naming, each fxml filename has the "Window" suffix (Figure 1).

<sup>3</sup> <https://unicode-org.github.io/icu/userguide/icu4j/>

<sup>4</sup> <https://gluonhq.com/products/scene-builder/>

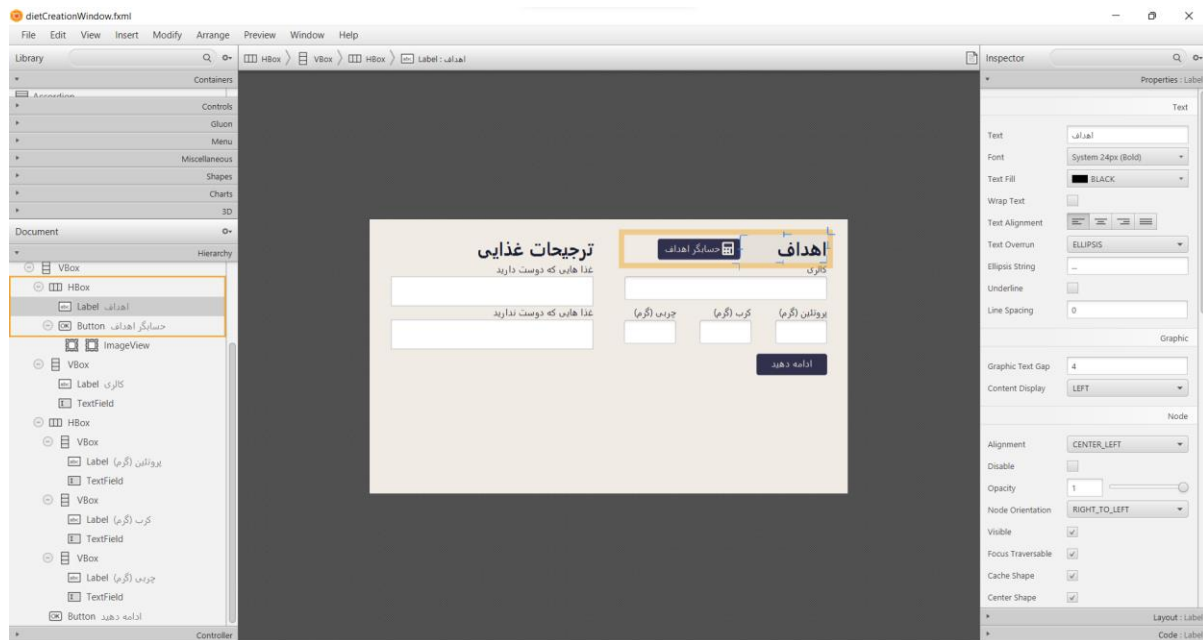


Figure 5. Scene Builder

This allowed faster development which demonstrates appropriate use of existing tools.

## Controllers and Event Handling

Each fxml file has a corresponding controller with the suffix “WindowController” (Figure 1). Controllers handle GUI events such as button clicks (Figure 6).



Figure 6. Event Handling

In order to communicate between controllers, I made methods with the prefix “inject”, which allow one controller to inject an element (List, TextField, etc.) into another controller so that it may be manipulated by that controller (Figure 7).



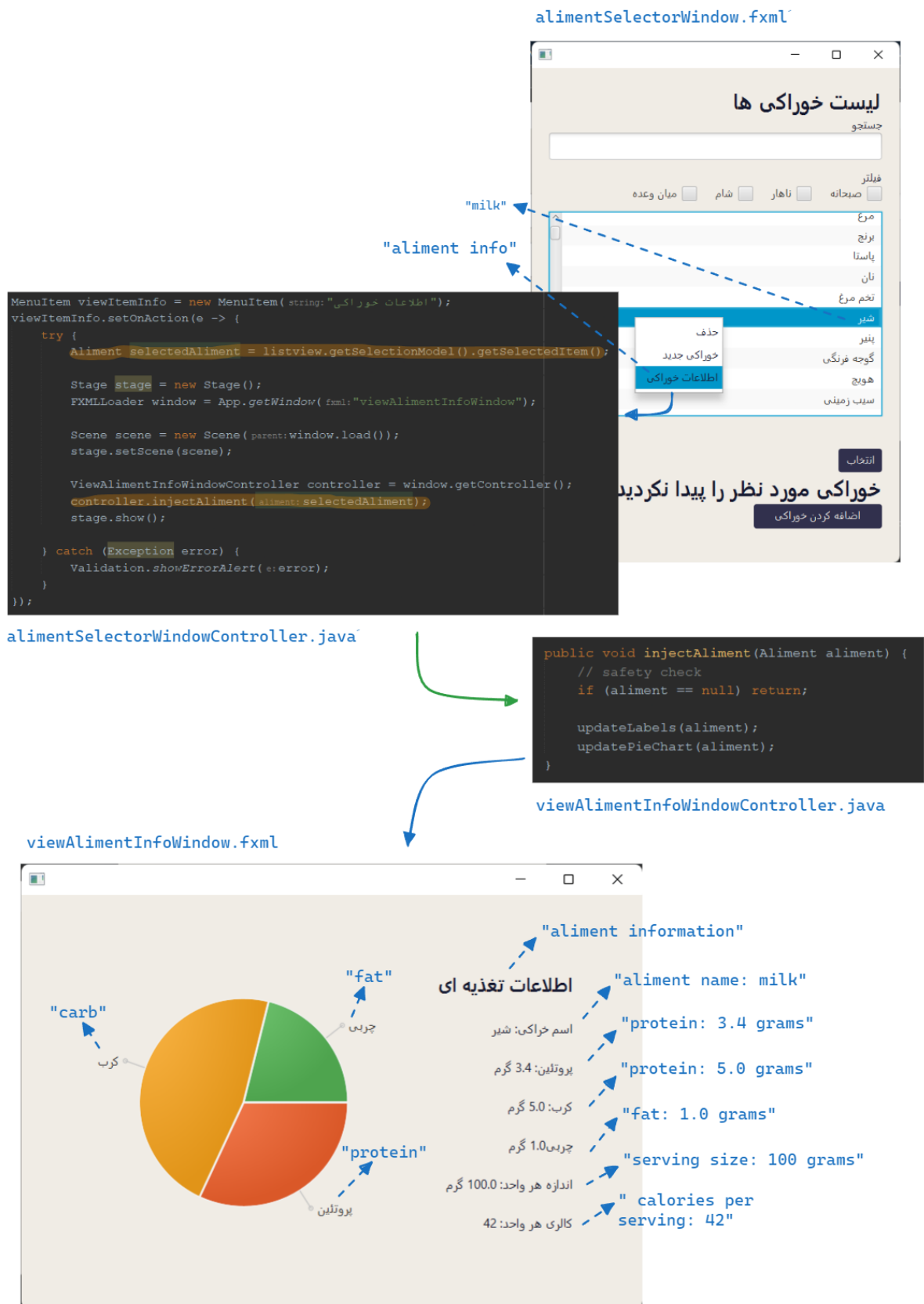


Figure 7. Communication between controllers

## Validation and error handling

To validate user input and handle errors, I created a dedicated class Validation.java. The class contains various methods such as (Figure 8)

```
/**
 * 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( at:Alert.AlertType.ERROR);
    alert.setTitle( string:"error");
    alert.setHeaderText( string:"خطایی رخ داد");
    alert.setContentText( string: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( at:Alert.AlertType.WARNING);
    alert.setTitle( string:"warning");
    alert.setHeaderText( string:"! هشدار");
    alert.setContentText( string:message);

    alert.showAndWait();
}
```

Figure 8. Warning and Error Alerts

This makes handling errors more efficient since code is not repeated and can easily be modified by changing the methods (Figure 9).

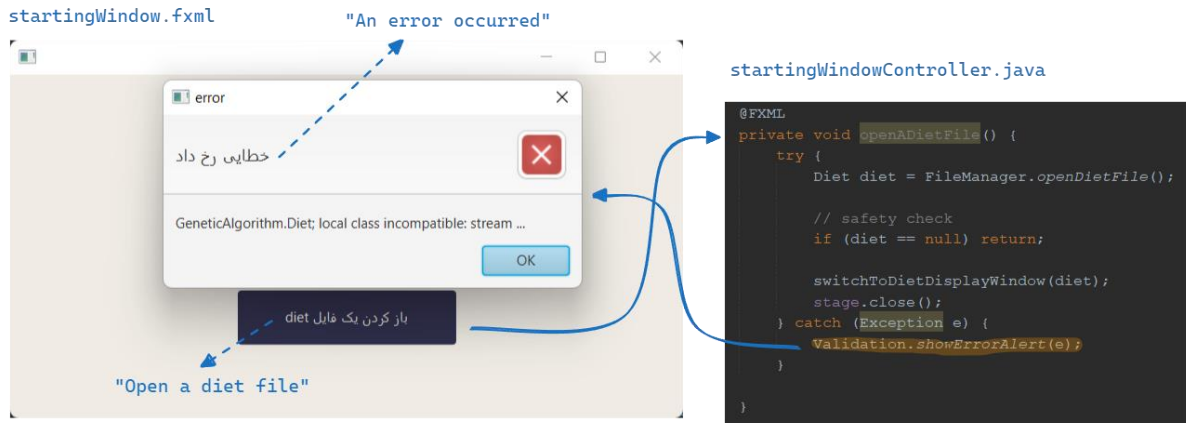


Figure 9. Example usage of Validation class methods

Other methods include validation checks such as (Figure 10):

```
/** Method which takes a Text Field and checks whether it is empty ...7 lines */
public static boolean isEmptyTextField(TextField txtFld) {
    if (txtFld.getText().isEmpty()) {
        // translation of text: please fill out all fields
        showWarningAlert(message: "لطفاً تمام ورودی ها را پر کنید!");
        return true;
    }
    return false;
}

/** Method which takes a Text Field and checks whether it contains numeric values .
public static boolean isNumericTextField(TextField txtFld) {
    if (!txtFld.getText().matches(regex: "\\d*")) {
        // translation of text: only numbers must be entered
        showWarningAlert(message: "فقط ورودی اعداد مجاز است!");
        return false;
    }
    return true;
}

/** Method which checks if a radio button from an inputted Toggle Group has been se
public static boolean isRadioSelected(ToggleGroup group) {
    RadioButton selected = (RadioButton) group.getSelectedToggle();
    if (selected == null) {
        // translation of text: please select an option!
        showWarningAlert(message: "لطفاً یک گزینه را انتخاب کنید!");
        return false;
    }
    return true;
}
```

Figure 10. Validation check methods

Example usage (Figure 11)

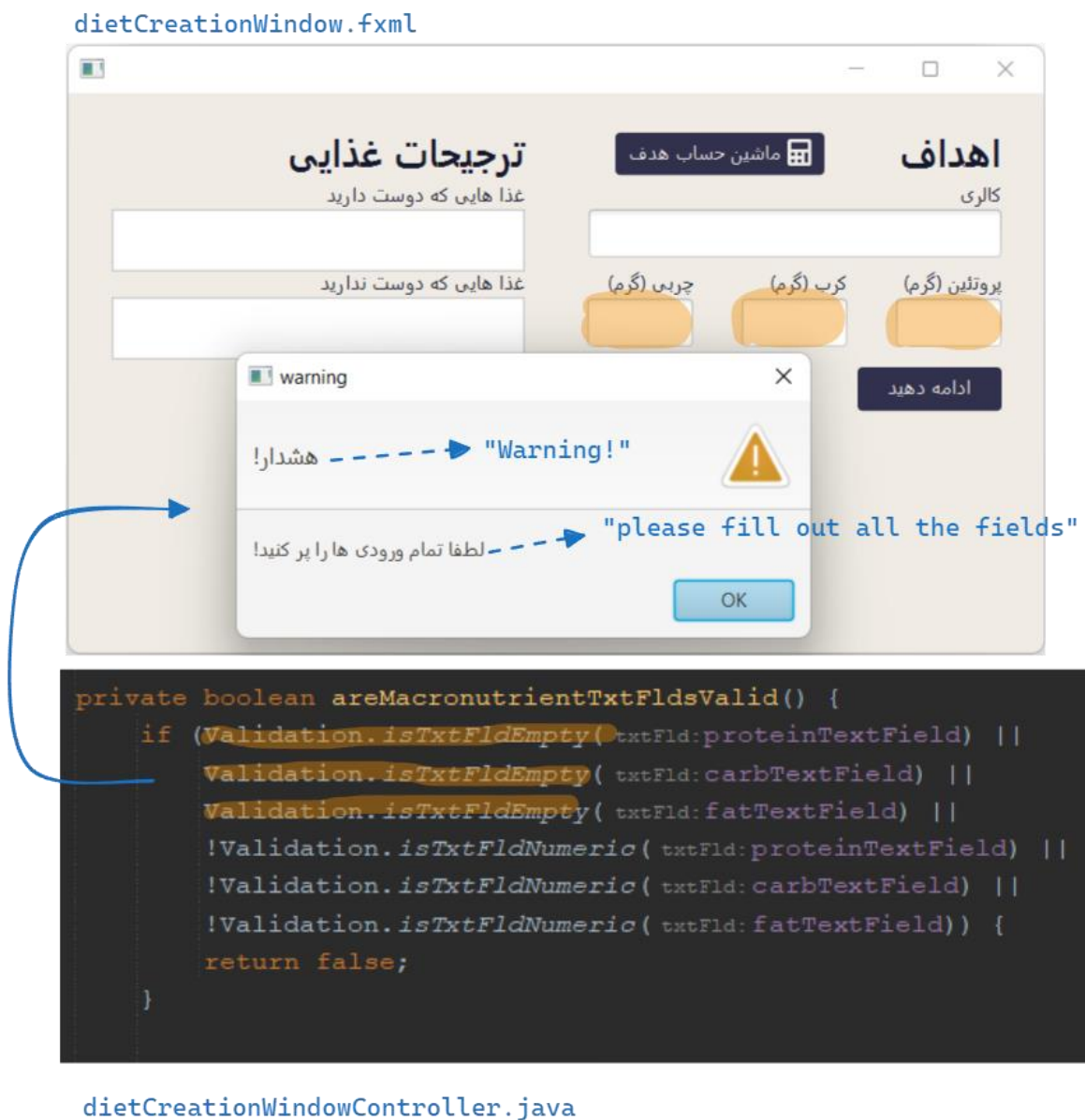


Figure 11. Validation check methods

## File Manager

The file manager class contains methods to read/write to different data files and turn Diets into PDFs.

## Aliments.data file

In this project, all aliment data are stored in the aliments.data file. This class contains 4 methods for manipulating aliments.data (Figure 12)

FileManager.java

```
/** This method reads the list of aliments in the aliments ...4 lines */
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 ...7 lines */
public static void addToAlimentDataFile(Aliment aliment) {
    App.aliments.add(aliment);
    rewriteToAlimentDataFile();
}

/** This method takes an aliment object and removes it from the App ...7 lines */
public static void removeFromAlimentDataFile(Aliment aliment) {
    App.aliments.remove(aliment);
    rewriteToAlimentDataFile();
}

/** This method writes the App ...4 lines */
private static void rewriteToAlimentDataFile() {
    try {
        File file = new File(pathname, ALIMENT_DATA_FILE_LOCATION);
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file));
        oos.writeObject(App.aliments);
        oos.close();
    } catch (Exception e) {
        Validation.showErrorAlert(e);
    }
}
```



alimentSelectorWindow.fxml

Figure 12. Methods for manipulating aliments.data

Here, App.aliments is a temporary in-memory representation of the datafile. Each time the variable is changed (add/remove), the datafile is rewritten to permanently store the changes.

## Diet Files

The class also contains methods for reading/writing diet files to save the generated diets in an editable format (Figure 13 and 14).

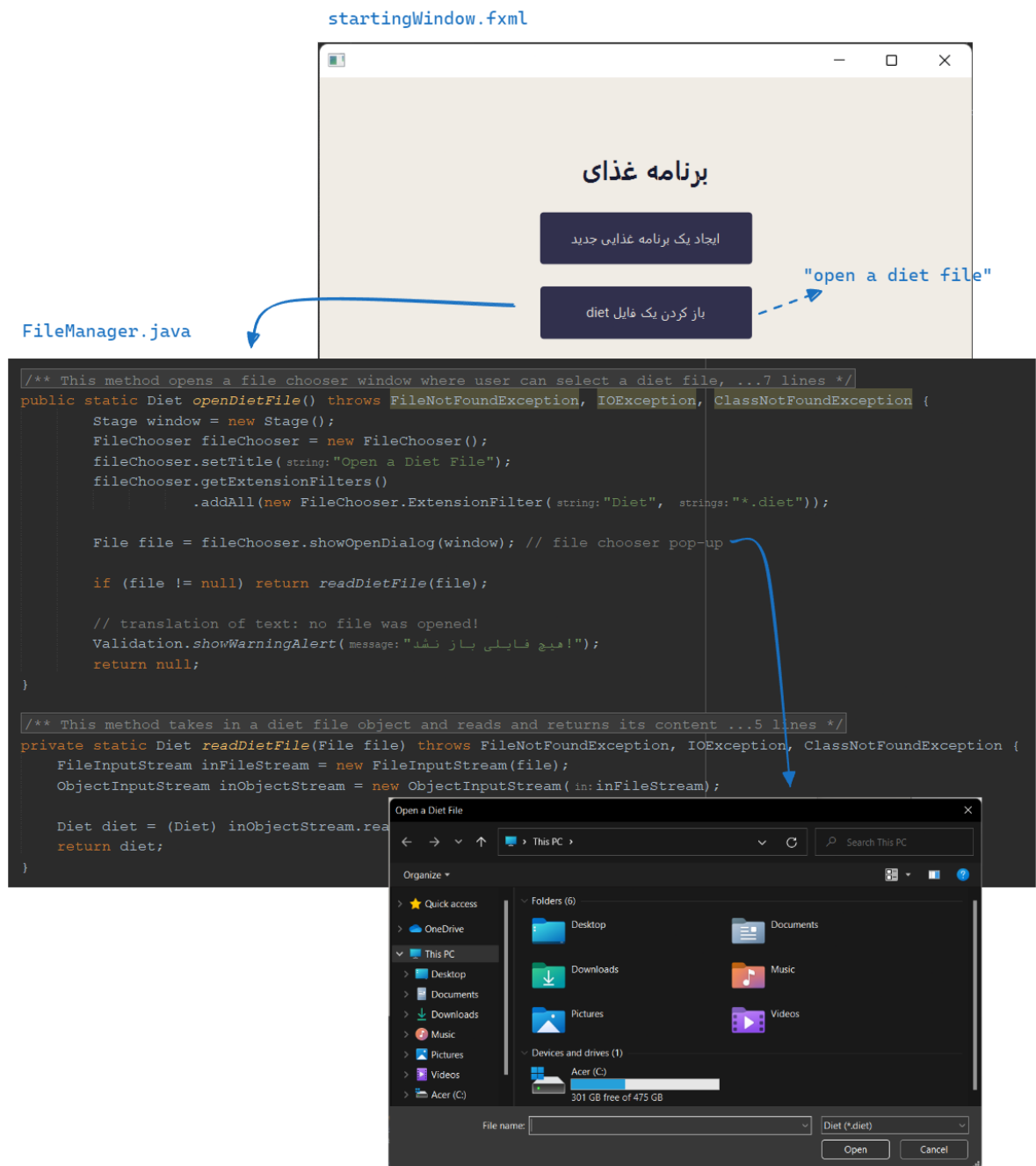


Figure 13. Reading a Diet File



FileManager.java

Figure 14. Saving a Diet File

For creating a diet file, the code was split into two parts, to avoid repetition of code, hence obeying DRY coding practice.

### **Printing to PDF**

Finally, for turning diets into PDFs, apache PDFBox library was used (Figure 15)



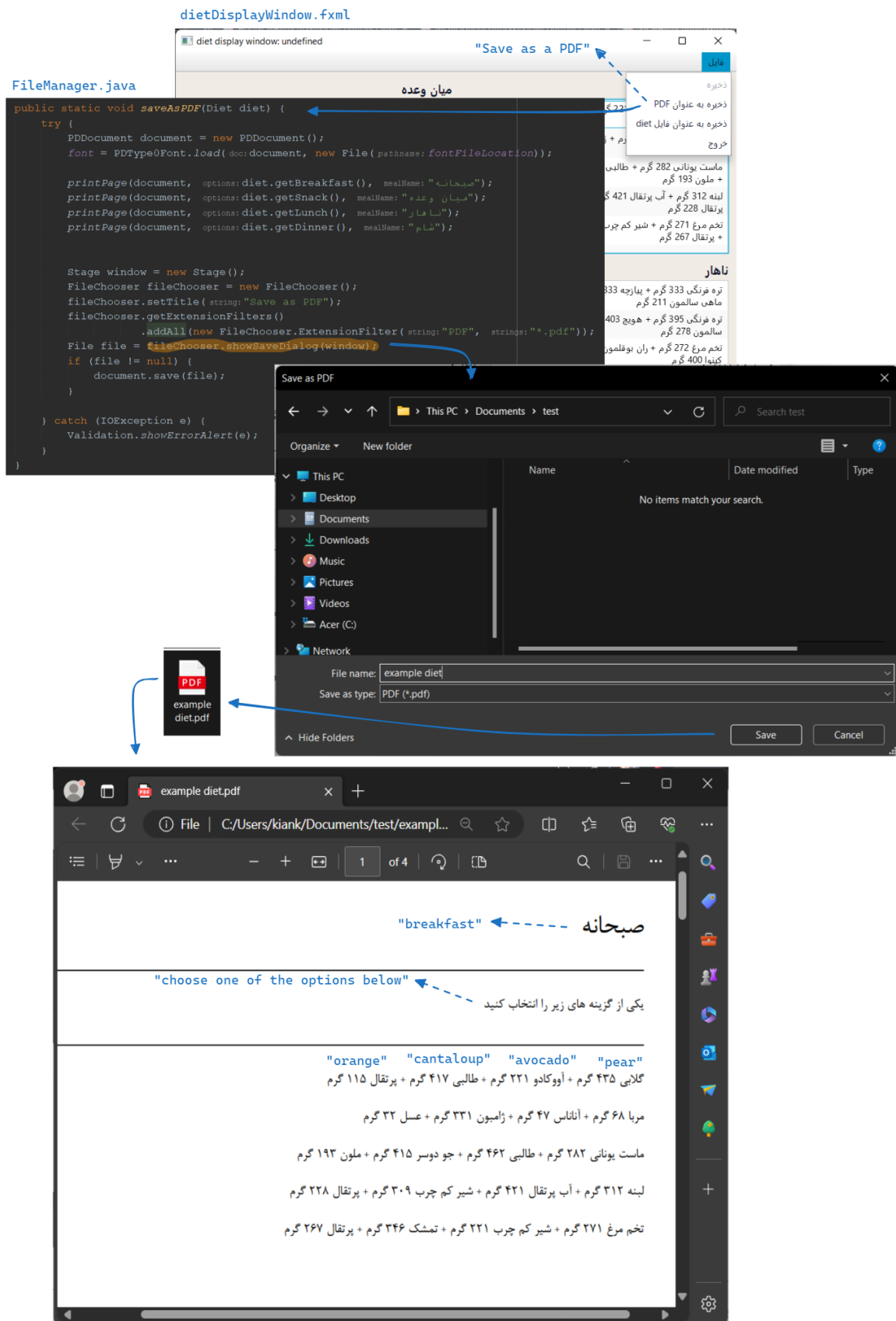


Figure 15. Saving diet as a PDF

Since the code for printing was verbose, I made different methods to reduce redundant code and provide layers of abstraction (Figure 16)

```

/** This method prints a single page of the PDF ...9 lines */
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, sourcePage:page);

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

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

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

    stream.close();
}

/** This method draws a horizontal line separator on the PDF page using the ...8 lines */
private static void drawSeperator(PDPageContentStream stream, int lineNum) throws IOException {
    stream.moveTo(x:pdfPageMargin, pdfStartY - pdfLineHeight*(lineNum - 1)); // Starting point of separator line
    stream.lineTo(x:pdfStartX, pdfStartY - pdfLineHeight*(lineNum - 1)); // Ending point of separator line
    stream.setStrokingColor(r:0, g:0, b:0); // Set color of the separator line
    stream.setLineWidth(lineWidth: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 ...10 lines */
private static void printText(PDPageContentStream stream, String text, int lineNum, int fontSize) throws IOException {
    text = processRTLText(rawText: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();
}

```

Figure 16. Modularization of PDF printing code

## Genetic Algorithm

Within my project a genetic algorithm is used to generate a meal option. Genetic Algorithms are useful here since it would take many years to find the most optimal solution with other optimization techniques. Instead, a heuristic approach is more appropriate.

The meal options exist both as genotypes (computational space) and phenotypes (physical space).<sup>5</sup>

The genotype consists of an array of chromosomes. Each chromosome represents a possible solution (meal option). The phenotype consists of an array of aliments, which are the physical representation of the chromosomes.

The chromosomes are encoded as HashMap, where key represents the index of the aliment in App.aliments, and value represents num of servings of that aliment (Figure 17)


```
/**
 * 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.
 */
private Map<Integer, Float> chromosome;
```

*Figure 17. chromosome HashMap*

The decode method of the chromosome converts it into phenotype space (Figure 18)

---

<sup>5</sup> [https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_fundamentals.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fundamentals.htm)

<pre> HashMap {   35: 1.2,   44: 0.4,   89: 3.2,   64: 2.3 } </pre>	<p>decode()</p> 	<pre> Aliment[] {   Aliment(), /* index 35 */   Aliment(), /* index 44 */   Aliment(), /* index 89 */   Aliment() /* index 64 */ } </pre>
---	---	---

Chromosome.java

```

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( index:gene.getKey());
            aliment = (Aliment) aliment.clone();

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

    return decoded;
}

```

Figure 18. Decoding chromosome from computational space to physical space

The genetic algorithm is run in the MealOption class (Figure 19)

```

@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( amount: POPULATION_SIZE);
    this.parentSelector = new ParentSelector( population: genotype, k: K);
    this.survivorSelector = new SurvivorSelector( population: genotype, k: 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
        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( numOfWinners: 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( a: fittest.decode());
}

```

*Figure 19. Genetic Algorithm code*

The algorithm utilizes a combination of sentinel and count controlled loop, to make sure the loop does not run infinitely or excessively.

## Population Initializer

This class creates an initial population of nearly random chromosomes.

```

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

/** A method which takes the number of individuals in the population and
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 ...8 lines */
private static Chromosome createChromosome() {
    Chromosome chromosome = new Chromosome();

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

    return chromosome;
}

```

*Figure 20. Initializing the population*

80 percent of genes are completely random, while 20 percent are guided. A guided gene is a random selection from the App.likes array. This is done to also include user preferences.

## Fitness calculation

Fitness is calculated based on variation. The less variation it has, the closer it is to 0, and hence it is a more suitable solution. Example variation calculations<sup>6</sup>

---

<sup>6</sup> See full FitnessCalculator.java file in Appendix

```

/** Calculates the difference (distance) between the calorie of the aliments ...7 lines */
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 ...8 lines */
private float calculateMealTendencyVariation(Aliment[] aliments) {
    float mealTendencySum = 0;

    for (Aliment aliment : aliments) {
        if(!aliment.getMealTendency().contains(0:mealTendency)) {
            mealTendencySum -= MEAL_TENDENCY_STEP;
        }
    }

    return mealTendencySum * MEAL_TENDENCY_COEFF;
}

/** Calculates the variation between user preferences and the aliments int the ...8 lines */
private float calculatePreferenceVariation(Aliment[] aliments) {
    float variation = 0;
    for (Aliment aliment : aliments) {
        if(App.likes.contains(0:aliment)) variation += PREFERENCE_STEP;
        if (App.dislikes.contains(0:aliment)) variation -= PREFERENCE_STEP;
    }

    return variation * PREFERENCE_COEFF;
}

```

Figure 21. Calculating the variation between solution and constraints

In each method, the solution is compared with the constraints and a variation is calculated which is multiplied by a coefficient to add weight.

## K-way tournament selection

For parent selection and survivor selection, a K-way tournament selection algorithm is used to prevent elitism<sup>7</sup>. In this algorithm, K random solutions are chosen from the population (Figure 22).

<sup>7</sup> [https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_parent\\_selection.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm)

```

protected Chromosome[] selectKChromosomes() {
    Chromosome[] selected = new Chromosome[k];
    Random random = new Random();

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

    return selected;
}

```

*Figure 22. Selecting K random solutions from population*

From these K solutions the winner/loser is chosen (the higher K, the more elitism) (Figure 23)

```

/** This method finds the winner of one round of the tournament selection
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
public Chromosome[] getWinners(int numWinners) {
    Chromosome[] selected;
    Chromosome[] winners = new Chromosome[numWinners];

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

    return winners;
}

```

*Figure 23. winner is found based on fitness value.*



Each winner represents 1 parent. The opposite process of finding loser is done in survivor selection.

## Crossover

The crossover class is responsible for creating offspring chromosomes from an array of parents. The algorithm creates 2 offspring for each 2 parents.

```
public class Crossover implements Serializable {  
    /** Generates an array of offspring based on an even array of parents ...8 lines */  
    public static Chromosome[] generateOffsprings(Chromosome[] parents) {  
        if (parents.length % 2 != 0) {  
            System.err.println(x: "Number of parents must be even!");  
            System.exit( status: 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;  
    }  
}
```

*Figure 24. perform singlePointCrossOver for each pair of parents*

Single point cross over is a technique where a random point on a parent chromosome is selected, and the contents of each parent are swapped. (Figure 25)

```

private 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], servingSize:parent1.getGene(parent1keys[i]));
        offSpring2.addGene(parent2keys[i], servingSize: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], servingSize:parent2.getGene(parent2keys[i]));
        offSpring2.addGene(parent1keys[i], servingSize: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( chromosome:offSpring1);
    }
    while (offSpring2.getSize() < Chromosome.CHROMOSOME_SIZE) {
        PopulationInitializer.addRandomGene( chromosome:offSpring2);
    }

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

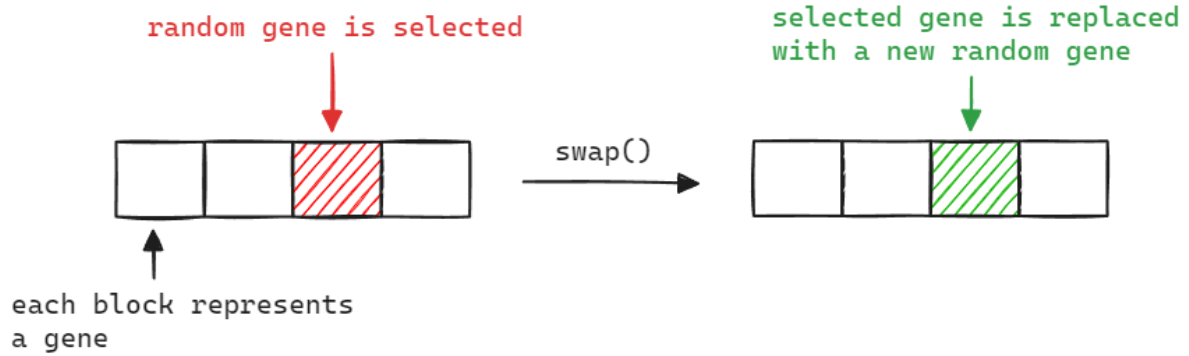
```

*Figure 25. code for single point crossover*

through trial and error I found that if the last two checks are not made, the size of the chromosome eventually shrinks.

## Mutator

Mutator class also helps prevent elitism and maintain meal variety. With a low probability a single random gene is removed from the chromosome, and a new one added (either random or guided). (Figure 26)



```

/** This method is the mutation algorithm ...6 lines */
private static void swap(Chromosome chromosome) {
    int randomIndexRemoved = RANDOM.nextInt( bound: chromosome.getSize());
    Integer[] keys = chromosome.getKeyset();
    chromosome.removeGene( keys[randomIndexRemoved]);

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

/** This method iterates through each chromosome of a population and mut.
public static void mutate(Chromosome[] genotype) {
    for (Chromosome chromosome : genotype) {
        if (chance( probability: MUTATION_PROBABILITY)) {
            swap(chromosome);
        }
    }
}

```

Figure 26. mutating chromosomes with low probability

## Multithreading

To improve computation time, the genetic algorithms are run on separate threads, as they are independent of each other.

Within each meal, 5 mealOptions are generated using multithreading. The mealOption.run() method is automatically called by thread.start() (Figure 27)

```

/** This method is internally called in {@link com ...9 lines */
@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( target: option);
            newThread.start();

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

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

```

*Figure 27. multithreading within the Meal class, to generate mealOptions*

When creating a diet, the meals are also executed using multithreading (Figure 28)

```

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

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

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

    diet = new Diet( breakfast: breakfastMeal.getMealOptions(),
                    lunch: lunchMeal.getMealOptions(),
                    snack: snackMeal.getMealOptions(),
                    dinner: dinnerMeal.getMealOptions());

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

```

Figure 28. multithreading within the dietCreationWindowController class, to create 4 meals.

## Citations

Throughout development, some third-party sources were used to aid coding. For instance, (figure 29) code from Stack Overflow<sup>8</sup> was taken to avoid spending redundant time writing already available code. The used of third-party sources is appropriate since it demonstrates an effective use of available resources in development.

```

/**
 * 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-java
 *
 * @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(regex: "\\d+")) {
                txtFld.setText( string: newValue.replaceAll( regex: "[^\\d+]", replacement: ""));
            }
        }
    });
}

```

<sup>8</sup> <https://stackoverflow.com/questions/7555564/what-is-the-recommended-way-to-make-a-numeric-textfield-in-java> and <https://stackoverflow.com/questions/48284888/writing-arabic-with-pdfbox-with-correct-characters-presentation-form-without-bei>

```

/**
 * 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-b
 */
@param rawText text to be processed
@return formatted text.
*/
private static String processRTLText(String rawText) {
    try {
        Bidi bidi = new Bidi( paragraph: (new PersianShaping( options: PersianShaping.LETTERS_SHAPE)).shape( text: rawText), flags: 127);
        bidi.setReorderingMode( reorderingMode: 0);
        return bidi.writeReordered( options: 2);
    }
    catch (ArabicShapingException ase3) {
        return rawText;
    }
}

```

Figure 29. code taken from stackoverflow.com.

Similarly, code from YouTube<sup>9</sup> was adapted for the GUI search functionality (figure 30).

```

/**
 * 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( a: searchWords.split( regex: " "));

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

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

```

Figure 30. Search code taken from YouTube.

<sup>9</sup> <https://youtu.be/VUVqamT8Npc?feature=shared>

Furthermore, the Persian shaping patch file for ICU4J library from the internet<sup>10</sup>. AI was also used to aid with some specific code such as the following (figure 31).

```

vate static void printText(PDPageContentStream stream, String text,
    text = processRTLText( rawText:text);
    // Calculate the width of the text (AI generated)
    float textWidth = font.getStringWidth(text) / 1000 * fontSize;

    stream.beginText();
    stream.setFont(font, fontSize);

```

```

/**
 * 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(ov: listView.widthProperty().subtract(1:50));
        }
        @Override
        protected void updateItem(MealOption item, boolean empty) {
            super.updateItem(t:item, bln:empty);

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

```

```

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

```

Figure 31. specific code generated by AI.

<sup>10</sup> <https://drive.google.com/file/d/1g9j4oH-kPbzsNqF1VZ7jFeMEjOBois0j/view>

Finally, values from calculator.net<sup>11</sup> were taken for calculations in *goalsCalculatorWindowController.java*, as per the client's request (figure 32).

```
// 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(message: "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(message: "لطفا تمام اطلاعات را پر کنید");
    return;
}
```

Figure 32. values for goals calculator.

Word count (excluding table of contents, headings, captions, and footers): 1123

---

<sup>11</sup> <https://www.calculator.net/calorie-calculator.html>