

СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“  
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА



КУРСОВ ПРОЕКТ  
ПО СИСТЕМИ, ОСНОВАНИ НА ЗНАНИЯ

Тема:

Система, основана на правила, която по данни за налични хранителни продукти,  
предлага рецепти на ястия, които може да бъдат приготвени от тези продукти

Студенти:

Соня Стоянова  
Йоана Йонкова  
Кристина Цекова

София, януари 2021 г.

## 1. Формулировка на задачата.

Реализирайте система, основана на правила, която по данни за налични хранителни продукти предлага рецепти на ястия, които може да бъдат приготвени от тези продукти. Данните за продуктите да включват: наименование на продукта; енергийна стойност; цена; налично количество. За предлаганите рецепти да се включват: наименование на ястието; максимален брой порции, които може да бъдат приготвени с наличните продукти; енергийна стойност и цена на една порция.

**Уточнения за въведените ограничения:** За изчисляване на броя порции, които могат да се получат от подадените продукти, е необходимо да се изчисли точното количество на продуктите. За целта превръщаме цената и калориите на продуктите, така че те да отговарят за един грам (или един брой), например, казваме едно яйце, а не един грам яйце. Освен това калориите и цената на продуктите се получават в зависимост от тяхното количество. Следователно, крайната цена и калориите на продуктите се образуват като се умножат по количеството. А пък калориите и цената за всяка рецепта се получават като се съберат калориите и цените на съставлящите я продукти. Калориите, цената и количеството не могат да бъдат по - малки или равни на 0. А името на продукт или рецепта не може да бъде null. Правим валидации както в методите в класовете, така и при изграждането на потребителския интерфейс. Въведените от нас рецепти съдържат продукти, необходими за създаването на една порция. Въведените цени за 1 грам (или 1 брой) от продукта не са видими за потребителя. Той може да види само крайната цена за една рецепта, както и цената за една порция от рецептата.

## 2. Използвани алгоритми.

### 2.1. Описание на метода `hasEnoughQuantity`

Методът приема като аргумент списък с продукти, въведени от потребителя, и връща булева стойност. Целта му е да провери дали има достатъчно количество от необходимите продукти за създаването на рецептата.

Най-напред се създава списък от продуктите, които са въведени от потребителя, като се вземат само тези от тях, съдържащи се в конкретната рецепта. След това проверяваме дали броя на продуктите в този списък съответства на броя продукти в рецептата. Ако броят се различава, това означава, че нямаме всички продукти, следователно не можем да направим тази рецепта и метода връща `false`.

Ако броят е равен, тогава сравняваме имената на въведените продукти с имената на продуктите в рецептата. Ако количеството на продуктите, които въвежда потребителя, е по-голямо или равно на вече зададеното, то тогава имаме достатъчно продукти за изготвянето на рецептата и метода връща стойност `true`.

*Псевдокод:*

```
function hasEnoughQuantity(Argument userProducts) returns boolean
    declare array list of products and add all products which are from userProducts and are
    contained in recipe's products
    if the size of the array list is different from the size of the list of recipe's products
    then return false
    loop for each products of the array list
        loop for each products of the recipe
            if their names are equal then
                if the unit of user product is less than the unit of recipe product
                then return false
            end loop
        end loop
    end loop
    return true
end function
```

```
// method which checks if there is enough quantity of the products, so a recipe can be made up of them
public boolean hasEnoughQuantity(ArrayList<Product> userProducts) {
    // intersection is an ArrayList which contains only the products entered by the user which are available in the .csv file
    ArrayList<Product> intersection = (ArrayList<Product>) userProducts.stream().filter((x) -> products.contains(x)).collect(Collectors.toList());

    // if there are not enough products to make one recipe, we return false
    if (intersection.size() != this.products.size()) {
        return false;
    }

    // check if there is the minimal quantity for each product
    for (Product userProduct : intersection) {
        for (Product p : products) {
            if (p.getName().equals(userProduct.getName())) {
                // if there is a product entered by the user which unit is smaller than the unit of the product in the file, return false
                if (userProduct.getUnit() < p.getUnit()) {
                    return false;
                }
            }
        }
    }

    // else, there is enough quantity to make a recipe and we return true
    return true;
}
```

## 2.2. Описание на метода calculateMaxPortions

Методът приема като аргумент списък с продукти, въведени от потребителя, и изчислява максималния брой порции, които могат да се получат от количеството на тези продукти.

Първо създаваме списък от цели числа. Всяко число е резултат от делението на количеството, въведено от потребителя, и количеството в рецептата за даден продукт. Тоест тези числа съответстват на това колко пъти можем да използваме продуктите за тази рецепта. След това взимаме най-малкия елемент от списъка и той показва максималния брой порции, които могат да бъдат направени.

*Псевдокод:*

```
function calculateMaxPortions(Argument input)
  declare empty array list of integers
  loop for each Product element of input
    loop for each Product element of recipe's products
      if their names are equal
        declare userUnit and set it to the unit of the input product
        declare recipeUnit and set it to the unit of the recipe product
        add the integer division of userUnit divided by recipeProduct to
          the array list
      end loop
    end loop
  set the serving to the minimum element of the array list
end function
```

```
// method which calculates the max portions that can be made up of the user products
public void calculateMaxPortions(ArrayList<Product> input) {
    // this list contains the minimal quantity /unit/ of a product
    // and it shows the minimal servings that can be made up of the given products
    ArrayList<Integer> minQuantity = new ArrayList<>();

    // for each entered product from the user list -> input, the minimal quantity can be work out as userUnit / recipeUnit
    for (Product userProduct : input) {
        for (Product recipeProduct : products) {
            if (userProduct.getName().equals(recipeProduct.getName())) {
                Double userUnit = userProduct.getUnit(); // this is the unit which is entered by the user for a product
                Double recipeUnit = recipeProduct.getUnit(); // this is the unit which is entered for the product in the .csv file
                // add the minimal quantity to the ArrayList of integers
                minQuantity.add((int) (userUnit / recipeUnit));
            }
        }
    }

    // the number of servings is actually the minimal quantity from the ArrayList minQuantity
    serving = Collections.min(minQuantity);
}
```

### 2.3. Описание на метода getRecipes

Методът връща списък с всички рецепти, които могат да се получат от списъка с подадени от потребителя продукти.

*Псевдокод:*

```
function getRecipes(Argument input) returns array list of recipes
  declare empty array list of recipes
  loop for each product from the database
    loop for each product of input
      if their names are equal then
        set the calories of the user product to the calories of the product from the
          database
```

```

        set the price of the user product to the price of the product from the
            database
    end loop
end loop
loop for each recipe from the database
    if call function hasEnoughQuantity(input) for the recipe is true then
        call function calculateMaxPortion(input) to set the servings
        add the recipe to the array list
    end loop
return the array list of recipes
end function

```

```

// method which returns an ArrayList of recipes that can be made up of the given ArrayList with products
public ArrayList<Recipe> getRecipes(ArrayList<Product> input) {
    ArrayList<Recipe> result = new ArrayList<>(); //
    // for each product from the ones which the user enter
    // search for it in the list of all products and set the calories and the price for it
    for(Product p : products) {
        for(Product product : input) {
            if(p.getName().equals(product.getName())) {
                product.setCalories(p.getCalories());
                product.setPrice(p.getPrice());
            }
        }
    }
    // for each recipe in the list of all recipes
    // if there is enough quantity of the user products, calculate the max portions for the recipe
    // that can be made of these products and add it to the result
    for (Recipe r : recipes) {
        if (r.hasEnoughQuantity(input)) {
            r.calculateMaxPortions(input);
            result.add(r);
        }
    }
    // finally, return all recipes which are in the result
    return result;
}

```

## 2.4. Описание на метода parseProducts

Помощен метод за четене на рецептите, който прочита продуктите от файла с рецепти, премахвайки скобите и интервалите, така че да се прочете само името на продукта и неговото количество.

*Псевдокод:*

```

function parseProducts(String input) returns an array list of products
    declare an array list result which will save all products from the file
    declare a product <- Product which will save the current product from the file

```

```

declare and initialize an array with strings -> parsed which removes the ')' and the
    space between the unit and the name of the product in the file
declare a String tmp which takes the last symbol from each line and removes it -> [')']
for each String s of parsed
    declare and initialize a String array only with the name and the unit of a product
    declare a flag found which shows if a product is found
    for each product p of products
        if the name of a product is in the list of all products
            set flag = true
            initialize the product <- new Product(name of the product, get its
                calories from our list of products, get its price from the list of products,
                    get its quantity from the file with recipes
            break
        if the product is not found
            throw an exception to show that exactly this product was not found
    add the product to the result
return result
end function

```

```

private ArrayList<Product> parseProducts(String input) throws InvalidObjectException {
    ArrayList<Product> result = new ArrayList<>(); // the result will be saved in this ArrayList
    Product product = null; // declare a product
    String[] parsed = input.split( regex: "[)][" ); // remove the ')' and the ' ' between the unit and the name of product

    String tmp = parsed[parsed.length - 1]; // take the last char from the input line (which is ')')
    tmp = tmp.substring(0, tmp.length() - 1); // remove the last char - ')' from the input
    parsed[parsed.length - 1] = tmp; // replace the tmp which was ')' with ''

    // for each string from parsed
    for (String s : parsed) {
        String[] nameAndQuantity = s.split( regex: "[ ]([)]" ); // remove the ' ' and the '()'
        boolean found = false; // boolean flag which shows if a product name is found
        // for each product from the ArrayList of products
        for (Product p : products) {
            // nameAndQuantity[0] -> the name of a product
            if (p.getName().equals(nameAndQuantity[0])) {
                found = true; // we found a name of a product
                // initialize the product with the values in the .csv file for recipes using its constructor
                // nameAndQuantity[1] -> shows the unit
                product = new Product(nameAndQuantity[0], p.getCalories(), p.getPrice(),
                    Double.parseDouble(nameAndQuantity[1]));
                break;
            }
        }
        // check if the product is entered correctly in the recipe file
        if (!found) {
            throw new InvalidObjectException("The product " + nameAndQuantity[0] + " was not found!\n");
        }
        // finally, add the products to the result
        result.add(product);
    }
    // and return result
    return result;
}

```



## 2.5. Описание на метода readRecipes

Методът приема име на файл и прочита рецептите, които са въведени в него.

*Псевдокод:*

```
function readRecipes(String filename)
    declare variable openFile <- BufferedReader
    try
        initialize openFile with the name of file <- FileReader
        read the first line
        declare and initialize a variable line which reads the text of each line
        while the text in line is not null
            declare and initialize a String array lineData with the text without ','
            if the lineData has length > 0
                add the recipe in an array list which is declared as a private data
                member to the class {read the name of a recipe, call the
                parseProducts function to read the products}
        catch each exception if there is any
    end function
```

```
// method which reads the recipes from the .csv file
private void readRecipes(String filename) {
    BufferedReader openFile; // open the file for reading
    try {
        // get the file
        openFile = new BufferedReader(new FileReader(filename));
        //Read to skip the header
        openFile.readLine();

        String line = "";
        //Reading from the second line
        while ((line = openFile.readLine()) != null) {
            String[] lineData = line.split(regex: ",");

            if (lineData.length > 0) {
                recipes.add(new Recipe(
                    lineData[0], // read the name of a recipe
                    parseProducts(lineData[1]) // read the list of products
                ));
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

## 2.6. Описание на метода readProducts

Този метод прочита информацията за всички въведени във файла продукти.

*Псевдокод:*

```
function readProducts(String filename)
    declare variable openFile <- BufferedReader
    try
        initialize openFile with the name of file <- FileReader
        read the first line
        declare and initialize a variable line which reads the text of each line
        while the text in line is not null
            declare and initialize a String array lineData with the text without ','
            if the lineData has length > 0
                add the product in an array list which is declared as a private data
                member to the class {read the name of a product,
                read its calories, read its price, read its unit}
        catch each exception if there is any
    end function
```

```
// method which reads the products from the .csv file
private void readProducts(String filename) {
    BufferedReader openFile; // open the file for reading
    try {
        // get the file
        openFile = new BufferedReader(new FileReader(filename));
        //Read to skip the header
        openFile.readLine();

        String line = "";
        //Reading from the second line
        while ((line = openFile.readLine()) != null) {
            String[] lineData = line.split( regex: ",");

            if (lineData.length > 0) {
                //Save the values in the training data
                products.add(new Product(
                    lineData[0], // read the name of a product
                    Double.parseDouble(lineData[1]), // read the calories
                    Double.parseDouble(lineData[2]), // read the price
                    Double.parseDouble(lineData[3]) // read the unit (quantity)
                ));
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



## 2.7. Описание на метода btnAddOnAction

Той е част от реализацията на потребителския интерфейс.

Целта му е да добавя всеки избран от потребителя продукт от падащото меню в текстова област, предназначена за въведените продукти.

*Псевдокод:*

```
function btnAddOnAction(event)
    if the user does not enter a quantity
        return an alert of type Alert.AlertType.ERROR which shows a message to
        make the user to enter a quantity
    if the user enters an invalid name of a product
        return an alert of type Alert.AlertType.ERROR which shows a message to
        make the user to enter a valid name or to choose a valid product from our list
        in the combo box
    else if the user enters a valid name of a product
        for each product from our array list of products which is a private data member
        of the class Controller
            if the list contains this product
                add it to the text area
        set the text of the text field for the quantity to null so the user can enter another value
end function
```

@FXML

```
void btnAddOnAction(ActionEvent event) {
    txtaRecipes.setText("");

    // validate that the user has entered a quantity
    // if not then show an alert with message for error and return
    if (txtQuantity.getText() == null || txtQuantity.getText().equals("")) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Invalid input");
        alert.setHeaderText("Please, enter a quantity!");
        alert.showAndWait();
        return;
    }

    // validate that the user has entered a valid name of a product
    // if not then show an alert with message for error and return
    if (!(cboChooseProduct.getItems().contains(cboChooseProduct.getValue()))) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Invalid product");
        alert.setHeaderText("Please, choose a valid product!");
        alert.showAndWait();
        txtQuantity.setText("");
        cboChooseProduct.setValue(null);
        return;
    }

    // else, add each product in the text area
    for (Product p : productList) {
        if (p.getName().equals(cboChooseProduct.getValue())) {
            txtaProducts.setText(txtaProducts.getText() + p.toString() + ", Entered quantity: " + txtQuantity.getText() + '\n');
        }
    }

    // finally, make the text null, so the user can enter a new product
    txtQuantity.setText("");
    cboChooseProduct.setValue(null);
}
```

## 2.8. Описание на метода btnGenerateOnAction

Той е част от реализацията на потребителския интерфейс. Идеята е в текстова област да се генерират всички рецепти, които могат да се направят от въведените от потребителя продукти.

*Псевдокод:*

function btnGenerateOnAction(event)

    if the user does not enter any product

        return an alert of type Alert.AlertType.ERROR which shows a message to make the user to enter a product

    else if there are some products

        declare a variable input of type array list which will save only the name of the product and the entered by the user quantity

        declare and initialize a variable products of type array of String which get the text from the text area with products, dividing the products with new line

        for each String p of products

            declare an array of Strings subProducts which contains the information without the comma and the space after it

            add all products to the input

    set text to the text area for recipes using the method getRecipes <- KnowledgeBase

    if there are not any valid recipes

        return an alert of type Alert.AlertType.ERROR which shows a message to make the user to enter new products

end function

```
void btnGenerateOnAction(ActionEvent event) {
    // validate that the user has entered some products
    // if not then show an alert with message for error and return
    if (txaProducts.getText().equals("")) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("The product list is empty!");
        alert.setHeaderText("Please, enter at least two products!");
        alert.showAndWait();
        return;
    }
    ArrayList<Product> input = new ArrayList<>();
    String[] products = txaProducts.getText().split( regex: "\\n"); // show each product in a new line
    for (String p : products) { // for each string in the list of products
        String[] subProducts = p.split( regex: "[,] "); // remove the ',' and the space
        input.add(new Product(
            subProducts[0].split( regex: "[:] ")[1], // add get the name of a product
            calories: null,
            price: null,
            Double.parseDouble(subProducts[subProducts.length - 1].split( regex: "[:] ")[1]) // and its quantity
        ));
    }
    // finally, show all recipes in the text area
    txaRecipes.setText(knowledgeBase.getRecipes(input).toString());
    if (txaRecipes.getText().equals("")) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("No recipes for these products!");
        alert.setHeaderText("Try with new products!");
        alert.showAndWait();
        return;
    }
    txaProducts.setText("");
}
```

## 2.9. Описание на метода btnRemoveLastOnAction

Той е част от реализацията на потребителския интерфейс. Идеята е да се премахва последно въведеният продукт от потребителя, в случай че е сгрешил при избора на продукт.

*Псевдокод:*

```
function btnRemoveLastOnAction(event)
    if the user does not enter any product in the text area for products
        return an alert of type Alert.AlertType.ERROR which shows a message that
        this operation can not be done
    declare variable result <- StringBuilder which takes the last entered product
    declare and initialize a String array tmp which get the text from the text area for
    products, divided by a new line
    for each value i to the product before the last one
        append all products except the last one
    set text to the text area of products but without the last product
end function
```

@FXML

```
void btnRemoveLastOnAction(ActionEvent event) {
    // validate that the user has entered at least one product in the text area for the products
    // if not then show an alert with message for error and return
    if (txaProducts.getText().equals("")) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Invalid operation");
        alert.setHeaderText("The list of products is empty!");
        alert.showAndWait();
        return;
    }

    StringBuilder result = new StringBuilder();
    String[] tmp = txaProducts.getText().split( regex "[\\n]"); // contains all products from the text area for products
    // get the last entered product with tmp.length - 1 and remove it
    for (int i = 0; i < tmp.length - 1; i++) {
        result.append(tmp[i]).append('\\n');
    }
    txaProducts.setText(result.toString());
}
```

## 2.10. Въвеждане на продуктите в падащото меню

```
// read the files with products and recipes
knowledgeBase = new KnowledgeBase( productsFile: "products.csv", recipesFile: "recipes.csv");
productArrayList = knowledgeBase.getProducts(); // get all products from the knowledge base

// add all names of products in the combo box so the user can choose from them
for (Product product : productArrayList) {
    cboChooseProduct.getItems().add(product.getName());
}

// some additional formatting of the text areas and the combo box
txaProducts.setTextWrapText(Boolean.TRUE);
txaRecipes.setTextWrapText(Boolean.TRUE);
cboChooseProduct.setEditable(Boolean.TRUE);
```

## 3. Описание на програмната реализация.

Проектът ни се състои от две части. Първата - функционалността на системата, се съдържа в пакета - **backend**, и тя описва всички създадени класове, необходими за реализирането на системата. А втората - потребителският интерфейс, се намира в пакета - **gui**.

### 3.1. Описание на пакета backend.

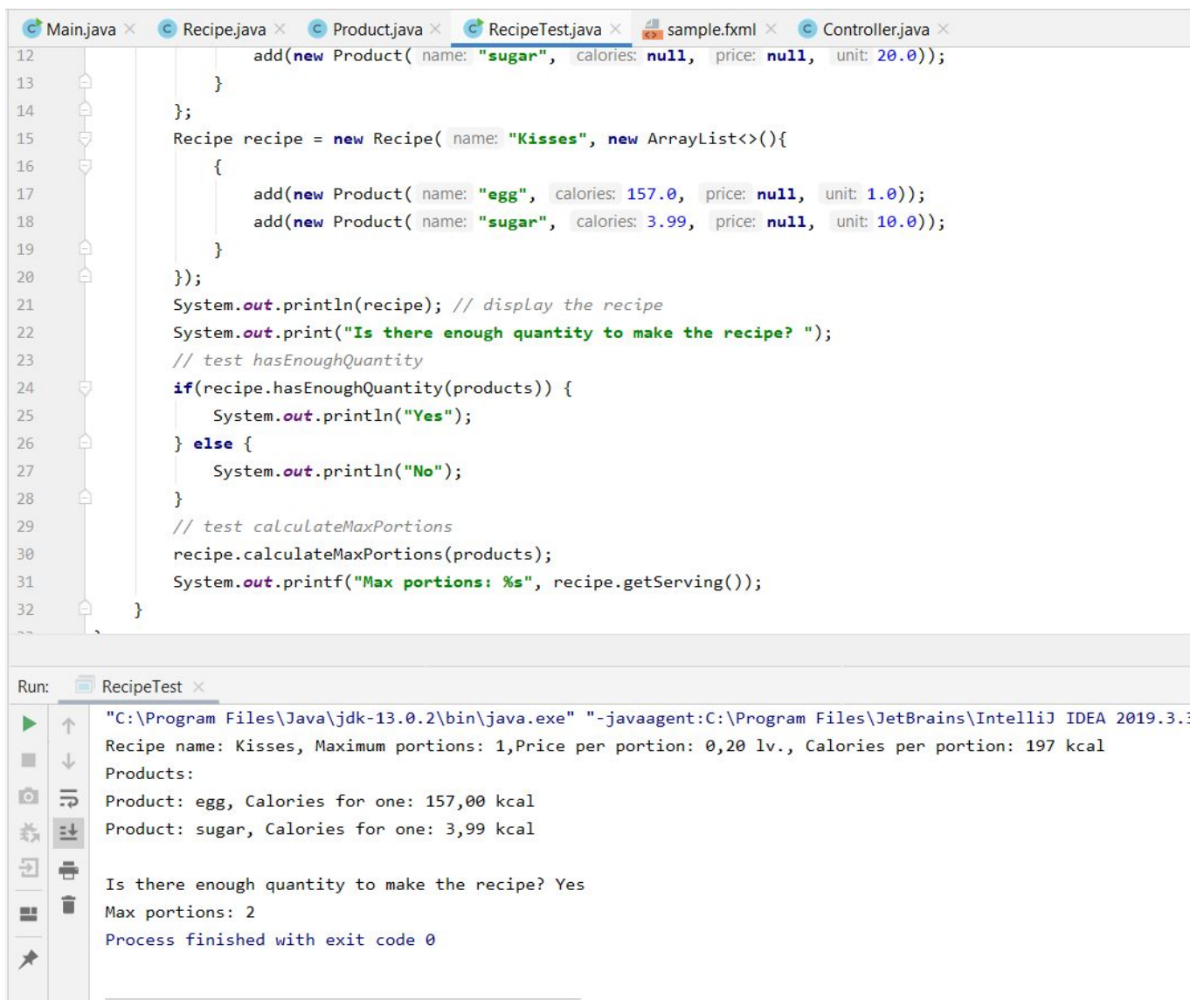
Като за начало, въвеждаме рецептите и продуктите, необходими за тях в отделни .xlsx файлове след което ги преобразуваме в .csv файлове, за да можем да четем данните от тях.

Започваме със създаването на клас Product, който описва продукт със следните характеристики – име, калории, цена и количество (unit). В този клас имаме конструктор, който създава продукт, както по зададени от потребителя стойности, така и със стойности по подразбиране, които ние сме задали. Освен това имаме методи за задаване и валидация на стойностите, както и методи за достъпване на член-данните на класа. Накрая в метода toString() извеждаме информацията за продукта.

След това създаваме друг клас Recipe, който описва една рецепта, включвайки информация за името на рецептата, порциите, цената, калориите и списък от продуктите, необходими за създаването ѝ. Аналогично на класа Product, имаме конструктор, който създава рецепта по име и списък от продукти. Методите за валидация задават калориите и цената за всяка рецепта, като се съберат калориите и цените на съставлящите я продукти.

Освен методи за достъпване на член-данните и за извеждане на информацията за рецептата, имаме и два други метода – `hasEnoughQuantity` и `calculateMaxPortions`, които описваме в точка 2. Използвани алгоритми.

За да проверим функционалността на класа `Recipe`, създаваме тестов клас `RecipeTest`. В него въвеждаме продукти, с които знаем, че можем да създадем определен брой порции от дадена рецепта. В случая въвеждаме яйца и захар, които са за една порция от рецептата за сладки (целувки):



```
12      add(new Product( name: "sugar", calories: null, price: null, unit: 20.0));
13    }
14  };
15  Recipe recipe = new Recipe( name: "Kisses", new ArrayList<>(){
16    {
17      add(new Product( name: "egg", calories: 157.0, price: null, unit: 1.0));
18      add(new Product( name: "sugar", calories: 3.99, price: null, unit: 10.0));
19    }
20  });
21  System.out.println(recipe); // display the recipe
22  System.out.print("Is there enough quantity to make the recipe? ");
23  // test hasEnoughQuantity
24  if(recipe.hasEnoughQuantity(products)) {
25    System.out.println("Yes");
26  } else {
27    System.out.println("No");
28  }
29  // test calculateMaxPortions
30  recipe.calculateMaxPortions(products);
31  System.out.printf("Max portions: %s", recipe.getServing());
32 }
```

Run: RecipeTest

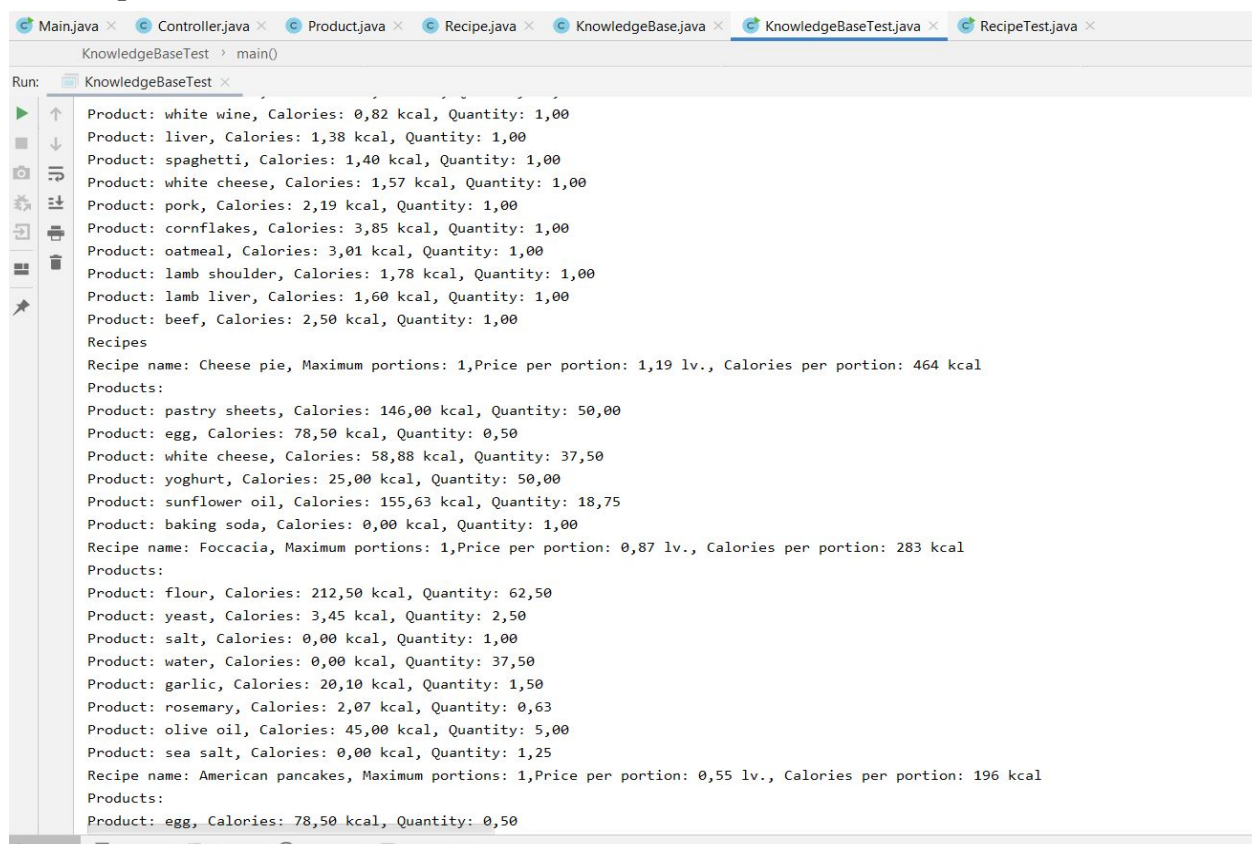
```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.1\lib\idea_rt.jar=12000:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.1\bin" -Dfile.encoding=UTF-8
Recipe name: Kisses, Maximum portions: 1,Price per portion: 0,20 lv., Calories per portion: 197 kcal
Products:
Product: egg, Calories for one: 157,00 kcal
Product: sugar, Calories for one: 3,99 kcal

Is there enough quantity to make the recipe? Yes
Max portions: 2
Process finished with exit code 0
```



Накрая създаваме класа KnowledgeBase, съдържащ списък с продукти и списък с рецепти. В конструктора на класа се осъществява четенето на информацията от .csv файловете и по този начин ги добавяме в списъците. Освен методите за четене имаме и метод getRecipes, който приема списък с продукти и връща като резултат списък с рецептите, които могат да се получат от тях, ако има такива. Ако няма такива, връща като резултат празния списък.

Малка част от тестване на четенето от файловете, тъй като продуктите и рецептите са голям брой:



```
Run: KnowledgeBaseTest
Product: white wine, Calories: 0,82 kcal, Quantity: 1,00
Product: liver, Calories: 1,38 kcal, Quantity: 1,00
Product: spaghetti, Calories: 1,40 kcal, Quantity: 1,00
Product: white cheese, Calories: 1,57 kcal, Quantity: 1,00
Product: pork, Calories: 2,19 kcal, Quantity: 1,00
Product: cornflakes, Calories: 3,85 kcal, Quantity: 1,00
Product: oatmeal, Calories: 3,01 kcal, Quantity: 1,00
Product: lamb shoulder, Calories: 1,78 kcal, Quantity: 1,00
Product: lamb liver, Calories: 1,60 kcal, Quantity: 1,00
Product: beef, Calories: 2,50 kcal, Quantity: 1,00
Recipes
Recipe name: Cheese pie, Maximum portions: 1, Price per portion: 1,19 lv., Calories per portion: 464 kcal
Products:
Product: pastry sheets, Calories: 146,00 kcal, Quantity: 50,00
Product: egg, Calories: 78,50 kcal, Quantity: 0,50
Product: white cheese, Calories: 58,88 kcal, Quantity: 37,50
Product: yoghurt, Calories: 25,00 kcal, Quantity: 50,00
Product: sunflower oil, Calories: 155,63 kcal, Quantity: 18,75
Product: baking soda, Calories: 0,00 kcal, Quantity: 1,00
Recipe name: Foccia, Maximum portions: 1, Price per portion: 0,87 lv., Calories per portion: 283 kcal
Products:
Product: flour, Calories: 212,50 kcal, Quantity: 62,50
Product: yeast, Calories: 3,45 kcal, Quantity: 2,50
Product: salt, Calories: 0,00 kcal, Quantity: 1,00
Product: water, Calories: 0,00 kcal, Quantity: 37,50
Product: garlic, Calories: 20,10 kcal, Quantity: 1,50
Product: rosemary, Calories: 2,07 kcal, Quantity: 0,63
Product: olive oil, Calories: 45,00 kcal, Quantity: 5,00
Product: sea salt, Calories: 0,00 kcal, Quantity: 1,25
Recipe name: American pancakes, Maximum portions: 1, Price per portion: 0,55 lv., Calories per portion: 196 kcal
Products:
Product: egg, Calories: 78,50 kcal, Quantity: 0,50
```

Тестване на метода getRecipes с примерни продукти, чиито калории и цена са null, защото се генерират на базата на въведените от нас калории и цена в таблицата с всички продукти:

```
ArrayList<Product> products = new ArrayList<>() {
    {
        add(new Product( name: "pastry sheets", calories: null, price: null, unit: 800.0));
        add(new Product( name: "egg", calories: null, price: null, unit: 8.0));
        add(new Product( name: "white cheese", calories: null, price: null, unit: 600.0));
        add(new Product( name: "yoghurt", calories: null, price: null, unit: 800.0));
        add(new Product( name: "sunflower oil", calories: null, price: null, unit: 300.0));
        add(new Product( name: "baking soda", calories: null, price: null, unit: 16.0));
        add(new Product( name: "flour", calories: null, price: null, unit: 1000.0));
        add(new Product( name: "yeast", calories: null, price: null, unit: 40.0));
        add(new Product( name: "salt", calories: null, price: null, unit: 16.0));
        add(new Product( name: "water", calories: null, price: null, unit: 600.0));
        add(new Product( name: "garlic", calories: null, price: null, unit: 24.0));
        add(new Product( name: "rosemary", calories: null, price: null, unit: 10.0));
        add(new Product( name: "olive oil", calories: null, price: null, unit: 80.0));
        add(new Product( name: "sea salt", calories: null, price: null, unit: 20.0));
        add(new Product( name: "butter", calories: null, price: null, unit: 125.0));
        add(new Product( name: "sugar", calories: null, price: null, unit: 60.0));
        add(new Product( name: "vanilla", calories: null, price: null, unit: 1.0));
        add(new Product( name: "baking powder", calories: null, price: null, unit: 5.0));
        add(new Product( name: "cherries", calories: null, price: null, unit: 125.0));
    }
};
```



Резултат от метода: С тези продукти могат да се направят 7 различни рецепти.

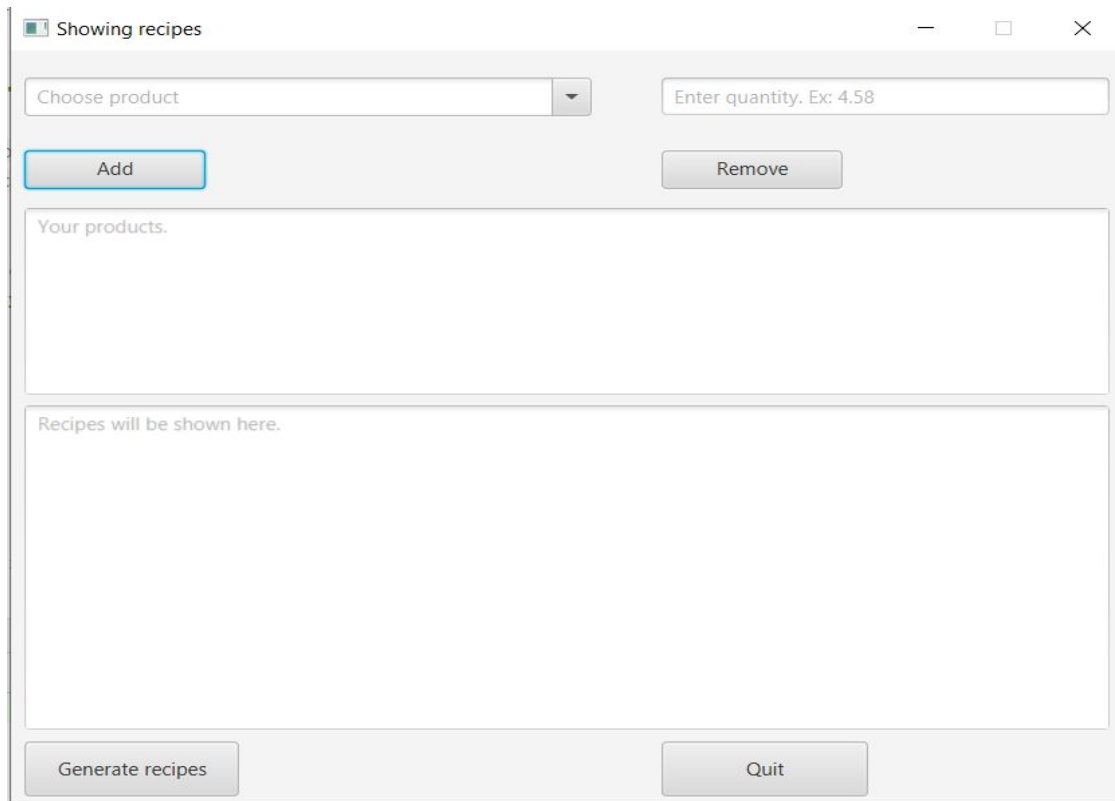
```
KnowledgeBaseTest x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.3\lib\ide
[Recipe name: Cheese pie, Maximum portions: 16,Price per portion: 1,19 lv., Calories per portion: 464 kcal
Products:
Product: pastry sheets, Calories: 146,00 kcal, Quantity: 50,00
Product: egg, Calories: 78,50 kcal, Quantity: 0,50
Product: white cheese, Calories: 58,88 kcal, Quantity: 37,50
Product: yoghurt, Calories: 25,00 kcal, Quantity: 50,00
Product: sunflower oil, Calories: 155,63 kcal, Quantity: 18,75
Product: baking soda, Calories: 0,00 kcal, Quantity: 1,00
, Recipe name: Foccacia, Maximum portions: 16,Price per portion: 0,87 lv., Calories per portion: 283 kcal
Products:
Product: flour, Calories: 212,50 kcal, Quantity: 62,50
Product: yeast, Calories: 3,45 kcal, Quantity: 2,50
Product: salt, Calories: 0,00 kcal, Quantity: 1,00
Product: water, Calories: 0,00 kcal, Quantity: 37,50
Product: garlic, Calories: 20,10 kcal, Quantity: 1,50
Product: rosemary, Calories: 2,07 kcal, Quantity: 0,63
Product: olive oil, Calories: 45,00 kcal, Quantity: 5,00
Product: sea salt, Calories: 0,00 kcal, Quantity: 1,25
, Recipe name: Buns, Maximum portions: 12,Price per portion: 0,99 lv., Calories per portion: 418 kcal
Products:
Product: egg, Calories: 104,56 kcal, Quantity: 0,67
Product: yoghurt, Calories: 16,67 kcal, Quantity: 33,33
Product: flour, Calories: 158,66 kcal, Quantity: 46,67
Product: baking soda, Calories: 0,00 kcal, Quantity: 0,67
Product: sunflower oil, Calories: 138,28 kcal, Quantity: 16,66

, Recipe name: Homemade cookies, Maximum portions: 4,Price per portion: 0,45 lv., Calories per portion: 161 kcal
Products:
Product: egg, Calories: 26,06 kcal, Quantity: 0,17
Product: yoghurt, Calories: 4,17 kcal, Quantity: 8,33
Product: sunflower oil, Calories: 20,75 kcal, Quantity: 2,50
Product: baking soda, Calories: 0,00 kcal, Quantity: 0,33
Product: vanilla, Calories: 0,72 kcal, Quantity: 0,25
Product: flour, Calories: 99,16 kcal, Quantity: 29,17
Product: sugar, Calories: 10,31 kcal, Quantity: 2,58
, Recipe name: Kisses, Maximum portions: 30,Price per portion: 0,04 lv., Calories per portion: 39 kcal
Products:
Product: egg, Calories: 31,40 kcal, Quantity: 0,20
Product: sugar, Calories: 7,98 kcal, Quantity: 2,00
, Recipe name: Cherry cake, Maximum portions: 6,Price per portion: 0,67 lv., Calories per portion: 315 kcal
Products:
Product: flour, Calories: 79,32 kcal, Quantity: 23,33
Product: egg, Calories: 26,06 kcal, Quantity: 0,17
Product: butter, Calories: 154,79 kcal, Quantity: 20,83
Product: sugar, Calories: 39,90 kcal, Quantity: 10,00
Product: vanilla, Calories: 0,48 kcal, Quantity: 0,17
Product: salt, Calories: 0,00 kcal, Quantity: 0,17
Product: baking powder, Calories: 1,47 kcal, Quantity: 0,83
Product: cherries, Calories: 12,92 kcal, Quantity: 20,83
, Recipe name: Omelet, Maximum portions: 4,Price per portion: 0,72 lv., Calories per portion: 369 kcal
Products:
Product: egg, Calories: 314,00 kcal, Quantity: 2,00
Product: white cheese, Calories: 54,95 kcal, Quantity: 35,00
Product: salt, Calories: 0,00 kcal, Quantity: 2,00
]

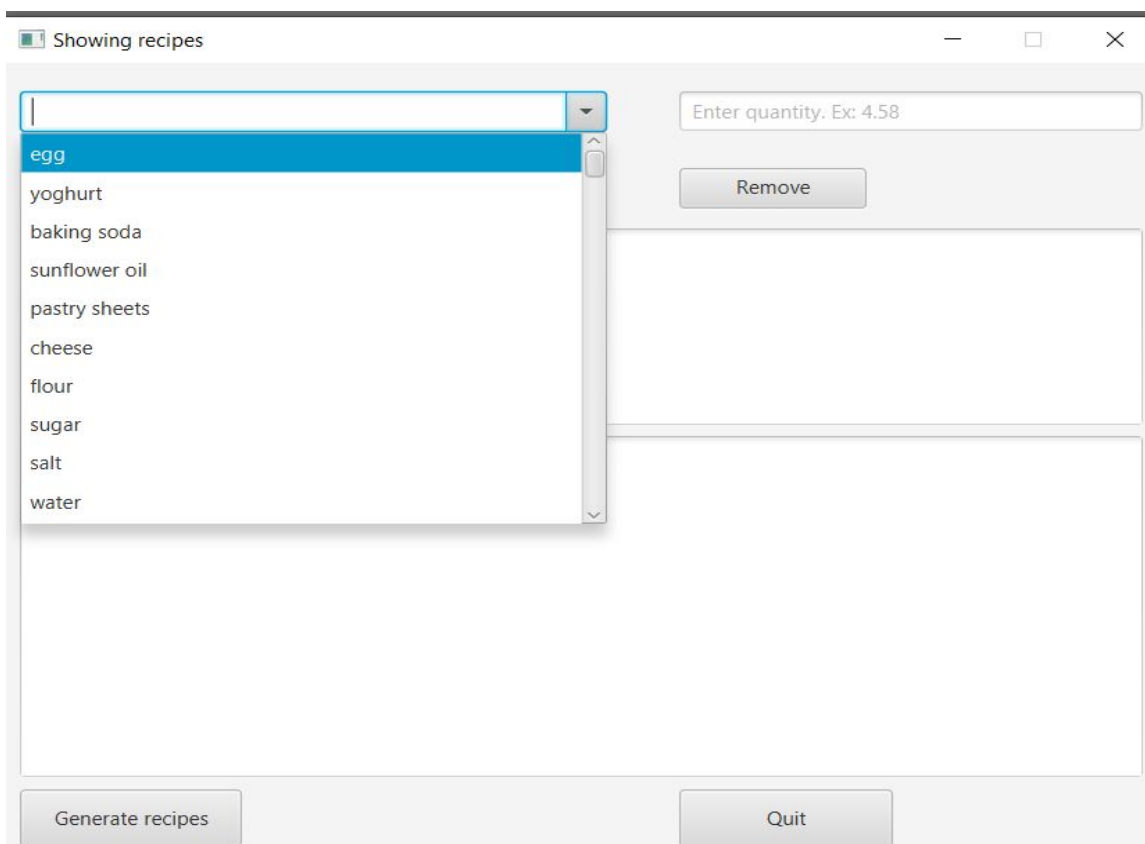
Process finished with exit code 0
```

#### 4. Потребителски интерфейс.

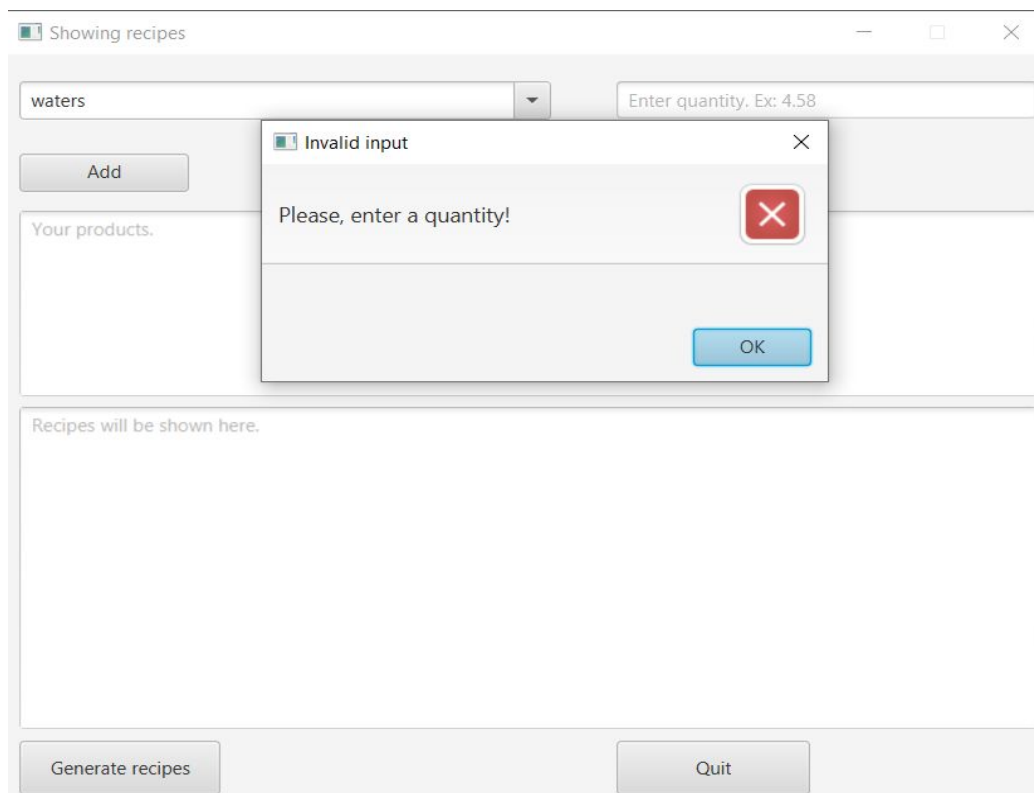
За да представим нагледно работата на системата, основана на правила, която създадохме, решихме да направим потребителски интерфейс с помощта на JavaFX, който има следния вид:



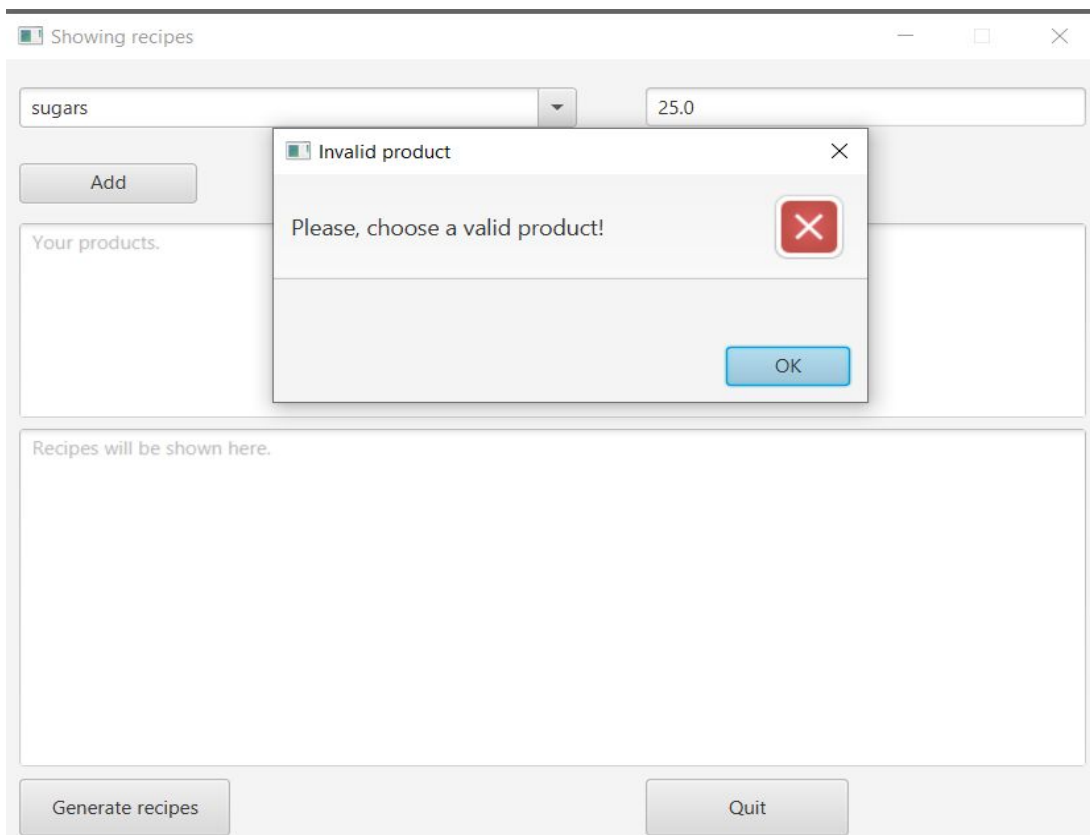
Всеки потребител има възможност да избира продукти от падащото меню, в което сме ги добавили:



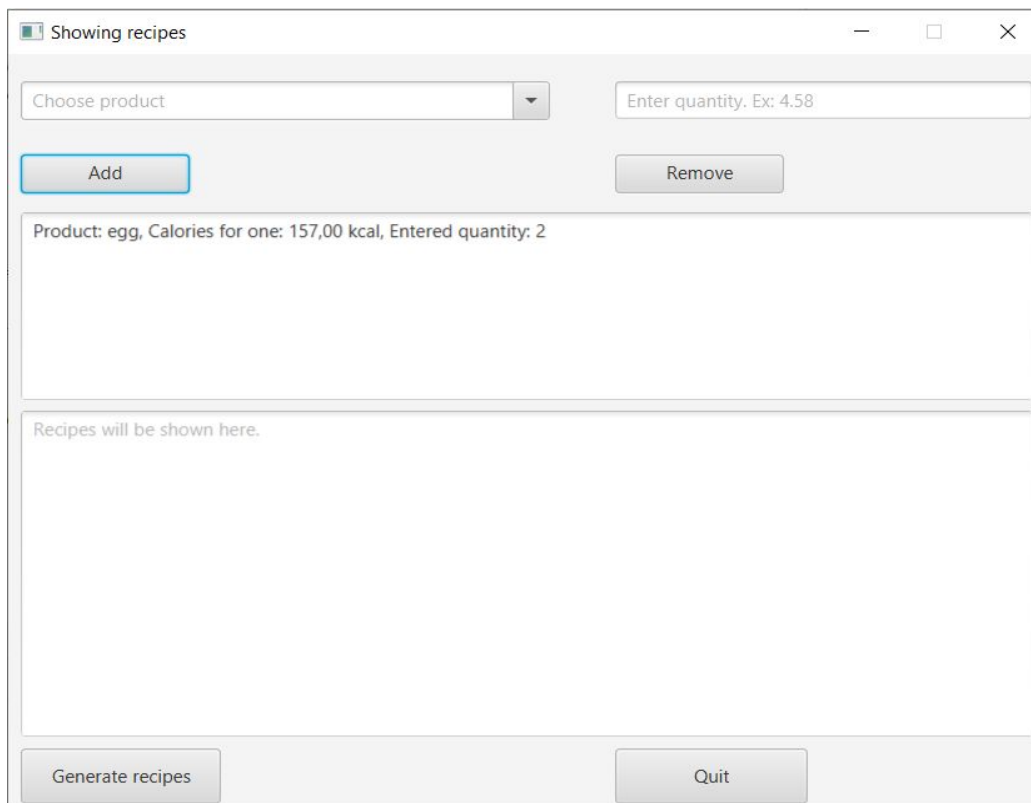
След като е избрал продукт, е задължително да въведе налично количество. В противен случай се извежда съобщение за грешка, което приканва потребителя да въведе количество:



Освен да избира от нашето меню, той има право и да въвежда сам имена на продукти. В случай че въведе невалидно име или име, което не се среща в нашия списък, получава следното съобщение:

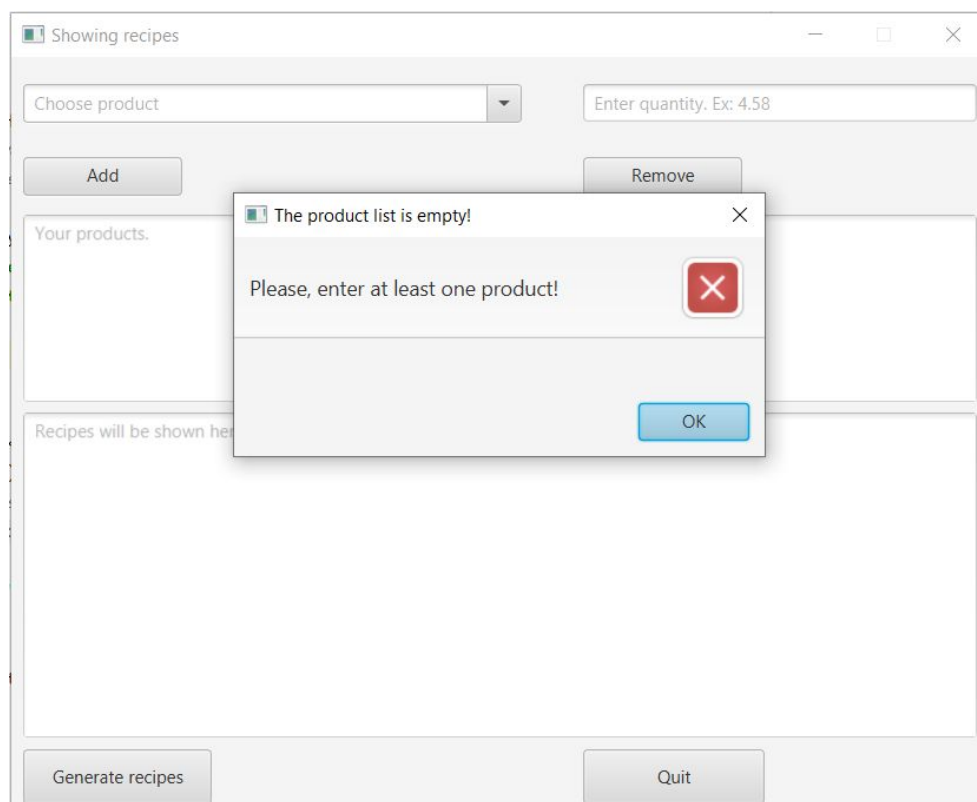


След като вече потребителят е въвел валидно име на продукт и налично количество, натискайки бутона Add, продуктът се добавя в текстовата област под бутона:

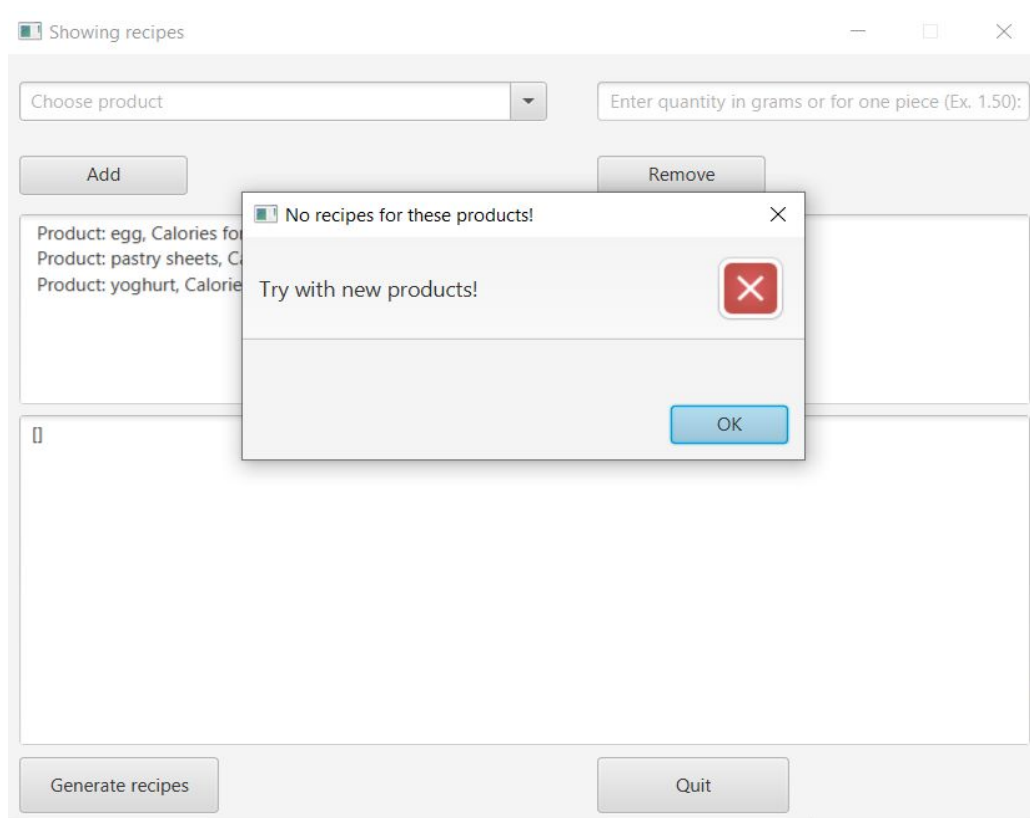


По този начин, колкото и продукти да избере потребителят, всички те ще се добавят в текстовата област. С помощта на бутона Remove потребителят може да премахва последния добавен продукт и тогава той се изтрива от списъка с въведени продукти.

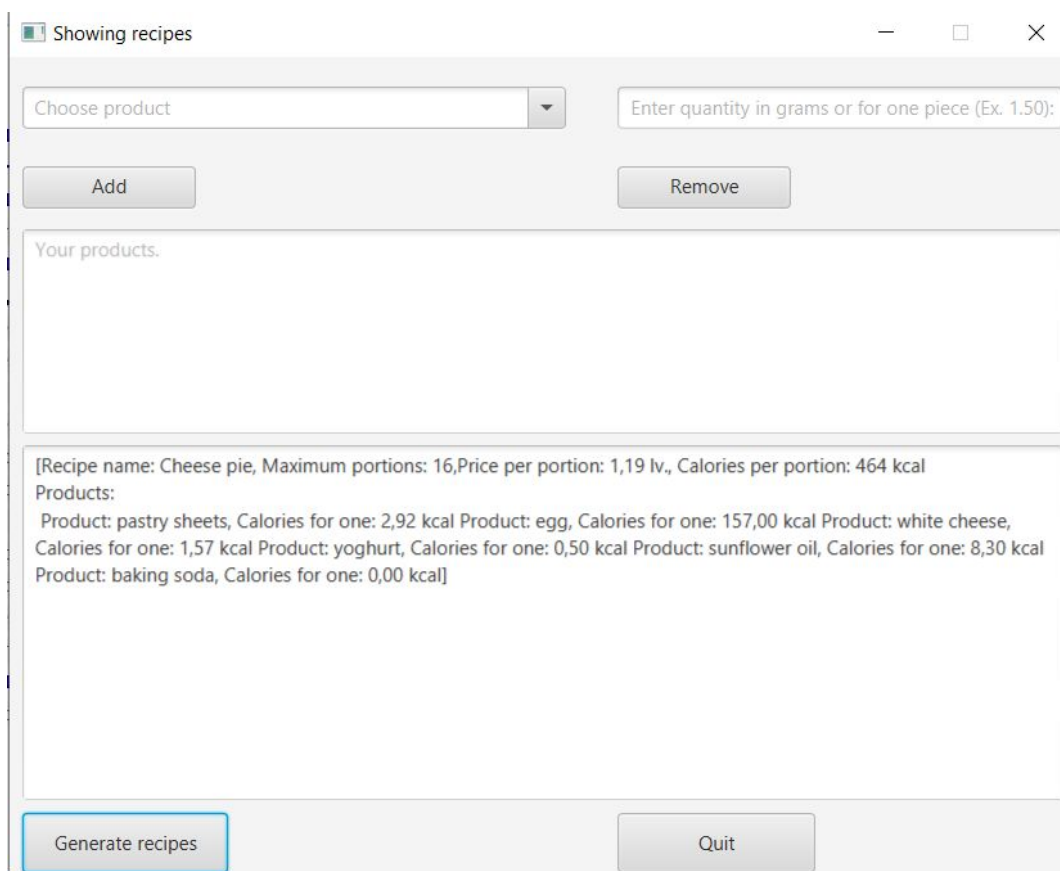
Бутонът Generate генерира рецептите, които могат да се получат на базата на въведените от потребителя продукти. В случай, че полето с продукти е празно, се извежда съобщение за грешка:



Ако пък няма подходяща рецепта за тези продукти, връщаме празния списък и отново съобщение за грешка:



Иначе, ако всички продукти могат да участват в рецепта, информация за нея се извежда в текстовата област под продуктите, като списъкът с продуктите се нулира, за да се позволи въвеждането на нови продукти:



## 5. Литература

<https://www.supichka.com>

<https://spoonacular.com/food-api>

<http://www.okeybg.com/blog/view/72/Tablitza-za-energiinata-stoinost-na-hranitelnite-produkti.html>

<http://gotvene.com>

[https://gotvach.bg/n5-37225-%D0%94%D0%BE%D0%BC%D0%B0%D1%88%D0%BD%D0%B8\\_%D0%B3%D0%BE%D1%82%D0%B2%D0%B0%D1%80%D1%81%D0%BA%D0%B8\\_%D0%BC%D0%B5%D1%80%D0%BA%D0%B8](https://gotvach.bg/n5-37225-%D0%94%D0%BE%D0%BC%D0%B0%D1%88%D0%BD%D0%B8_%D0%B3%D0%BE%D1%82%D0%B2%D0%B0%D1%80%D1%81%D0%BA%D0%B8_%D0%BC%D0%B5%D1%80%D0%BA%D0%B8)