SE360 Advances in Software Development

Smart Meal Picker

İdil Sanem Gürsoy

20220601040

Lecturer: Ufuk Çelikkan

23.12.2025

Project Overview

Introduction

Deciding what to eat is a common daily problem. People often need to consider many factors such as time, budget, calorie intake, dietary restrictions, and personal taste preferences. Because of these constraints, choosing a suitable meal can become difficult and time-consuming.

The "**Smart Meal Picker**" project is a Java desktop application that helps users decide what to eat based on their preferences. The user selects several options such as eating place, meal category, preparation time, budget level, dietary restrictions, and taste profile. According to these selections, the system suggests appropriate meal options from a predefined list.

The project focuses on using core Java concepts to build a practical and easy to use application for everyday life.

Project Objectives

The main objectives of this project are:

- To design a simiple graphical user interface using Java Swing.
- To model meal selection logic using object oriented programming principles.
- To allow flexible filtering based on user preferences.
- To practice Java collections, enums, and event handling.
- To verify the correctness of the system using Junit tests.

Functional Requirements

The following features are implemented in the system:

- Users can choose where they will eat (eat home, outside, or any).
- Users can select a meal category (main dish, snack, dessert, or drink).
- A maximum preparation time can be specified.
- Meals can be filtered by budget level (low, medium, high).
- Meals can be filtered by calorie level (low, medium, high).
- Dietary preferences such as vegan, vegetarian, and gluten-free are supported.
- Taste preferences can be selected (spicy, sweet, sour, savory).
- The system recommends meals that match the selected criteria.
- The user can add meals to a favorites list.
- Favorite meals are saved and loaded using Java serialization.

Non-Functional Requirements

The following non-functional requirements were considered during the development of the system:

- Usability: The user interface is simple and easy to understand, allowing users to interact with the system without prior training.
- Performance: The system provides meal recommendations instantly since all data is stored in memory.
- Maintainability: The application follows object-oriented design principles, making the code easy to read and extend.
- Reliability: The system produces consistent results for the same user inputs.
- Testability: Core recommendation logic is tested using unit tests implemented with Junit.

System Design and Operation

The application is designed as a simple Java desktop program that follows basic object-oriented principles.

The user interacts with the system through a graphical interface developed using Java Swing. From the main screen, the user selects preferences such as eating place, meal category, maximum preparation time, budget level, calorie level, dietary restrictions, and taste preferences.

When the user clicks the “Suggest Meals” button, the system collects all selected inputs and passes them to the recommendation logic. The recommendation logic filters the predefined list of meals according to the user’s preferences. Only the meals that satisfy all selected constraints are returned as suggestions.

Meal information is stored in memory using Java classes and collections. Fixed attributes such as category, budget level, calorie level, dietary tags, and taste tags are represented using enums, which prevents invalid values and simplifies filtering operations.

The filtered results are displayed to the user in a table format. Additionally, users can add meals to a favorites list, which is saved and loaded using Java serialization.

The core recommendation logic is tested using JUnit 5 to ensure correct behavior under different user preferences.

Architectural Diagram

The overall architecture of the system is shown in Figure 1. The application consists of a user interface layer, a recommendation logic layer, and an in-memory data layer.

Figure 1. Simple architecture of the Smart Meal Picker app.

Conceptual Database Table

Although no external database is used in this project, the data model is designed in a way that is similar to a database table. All data is stored in memory using Java classes and collections.

Column Name	Description
meal_id	Unique identifier of the meal
name	Name of the meal
category	Meal category
place	Eating place
prep_time	Preparation time (minutes)
budget_level	Budget level
calorie_level	Calorie level
diet_tags	Dietary properties
taste_tags	Taste properties

This table corresponds directly to the Meal class used in the implementation.

Java Technologies Used

The following Java technologies were used in this project:

- Java Swing for graphical user interface
- Java Collections Framework
- Java Enums
- Java Serialization
- Junit 5 for unit testing

Screenshots

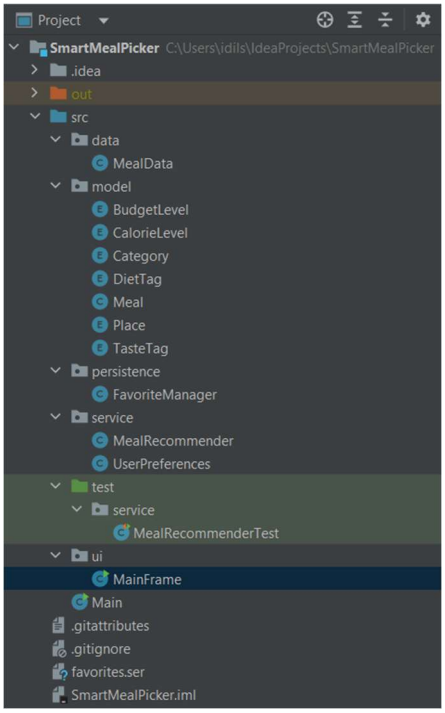

Figure 2. Package and class structure of the Smart Meal Picker project

This screenshot shows the overall package and class structure of the Smart Meal Picker application. The project is organized into separate packages for the user interface (ui), business logic (service), data model (model), data management (data), and persistence operations. Enum types, service classes, and JUnit test classes can also be clearly seen.

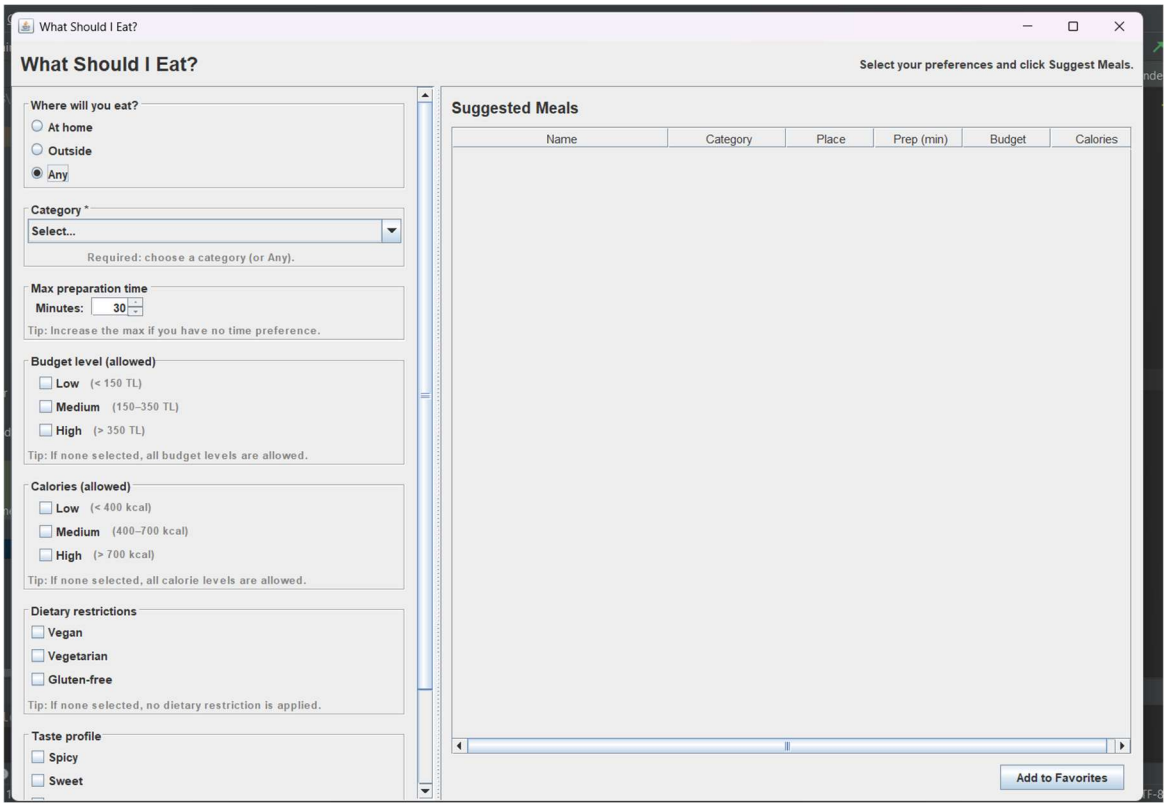

Figure 3. Initial application screen of the Smart Meal Picker

This screenshot shows the Smart Meal Picker application when it is first launched. All preference fields are displayed in their default state before any user interaction. The interface allows the user to select meal preferences such as place, category, preparation time, budget level, calorie level, dietary restrictions, and taste profile.

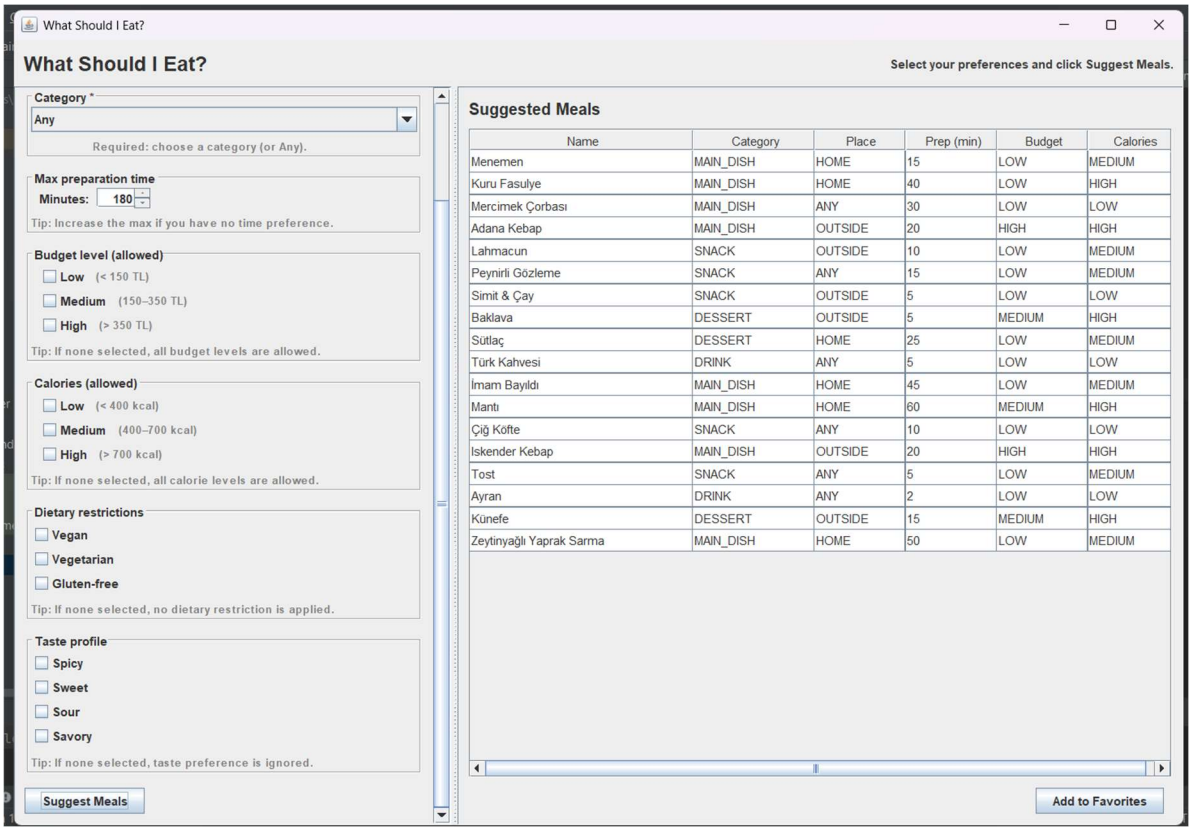

Figure 4. Meal suggestions displayed after applying user preferences

This screenshot shows the Smart Meal Picker application when it is first launched. All preference fields are displayed in their default state before any user interaction. The interface allows the user to select meal preferences such as place, category, preparation time, budget level, calorie level, dietary restrictions, and taste profile.

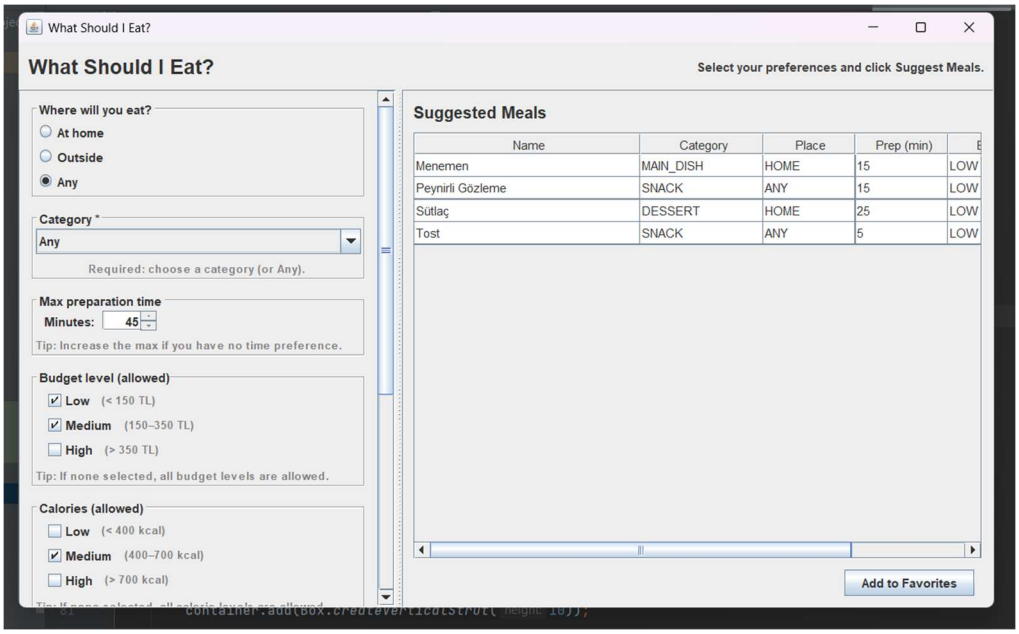

Figure 5. Filtered meal results with multiple preferences applied

This screenshot shows the application after multiple filters are applied by the user. Different preferences such as budget level, calorie level, dietary restrictions, and taste profile are selected. Also the application window is resized to demonstrate the use of scroll bars, ensuring that all interface components and results remain accessible even when the screen size is reduced.

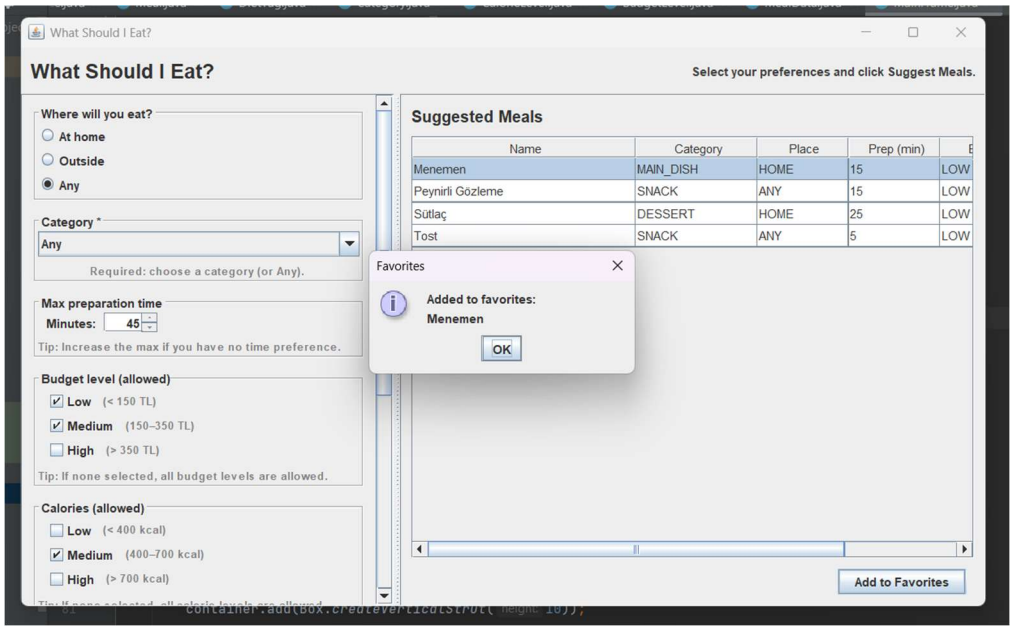

Figure 6. Adding a meal to the favorites list

In this screenshot, the user selects a meal from the results table and adds it to the favorites list by clicking the corresponding button. This interaction demonstrates how users can store preferred meals for later use. The favorites feature is implemented using Java serialization.

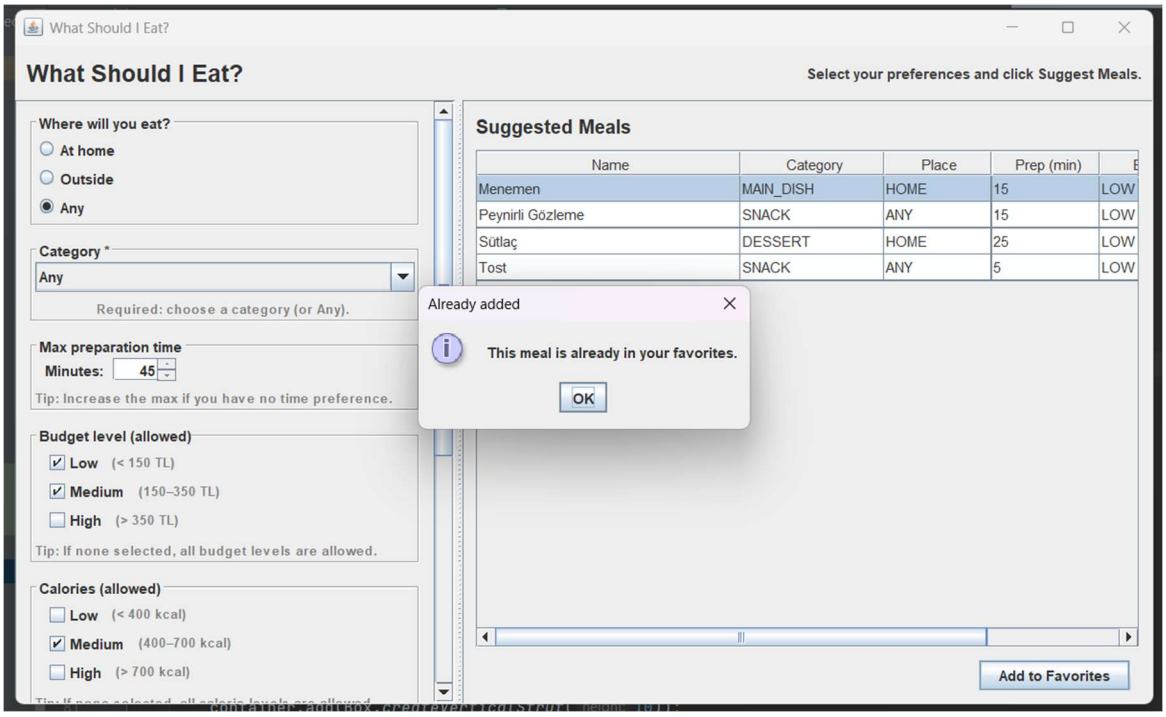

Figure 7. Warning message when attempting to add a duplicate favorite meal

In this screenshot, the user selects a meal from the results table and adds it to the favorites list by clicking the corresponding button. This interaction demonstrates how users can store preferred meals for later use. The favorites feature is implemented using Java serialization.

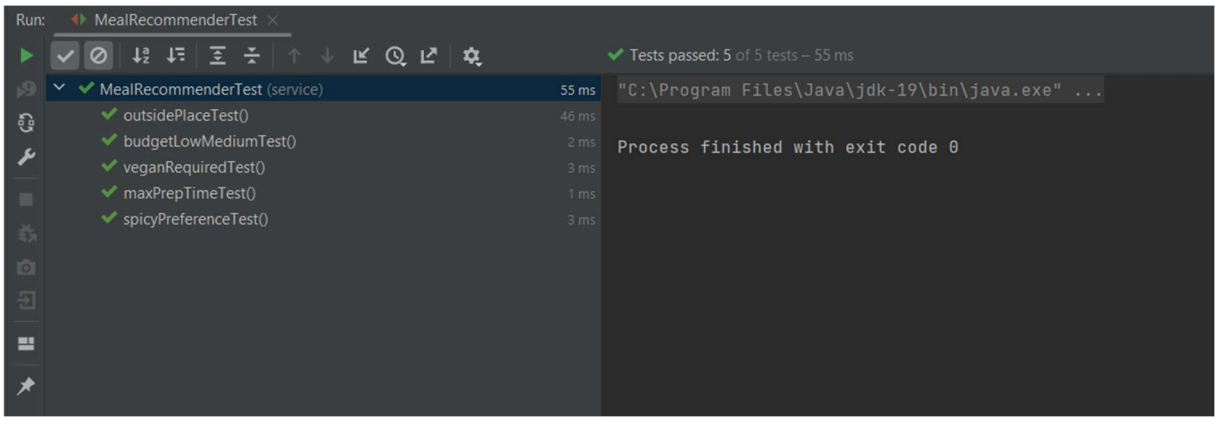

Figure 8. Successful execution of JUnit test cases

This screenshot shows the system behavior when the user attempts to add a meal to the favorites list that already exists. The application detects the duplicate entry and displays a warning message to inform the user. This mechanism prevents duplicate data and improves user experience.

User Manual

1. Launch the application.
2. Select where you want to eat.
3. Choose a meal category.
4. Set your preferences such as budget, calories, and dietary restrictions.
5. Click the “Suggest Meals” button.
6. View the recommended meals in the table.
7. Optionally, add a meal to favorites.

Tool and AI Usage

During the development of this project, various tools and resources were used to support the implementation and design process.

Online resources such as documentation pages, forums, and tutorial videos (including Java Swing examples available on Youtube) were consulted to better understand user interface components and event handling mechanisms.

AI-based tools were used during the brainstorming and development stages to clarify design decisions, improve code structure, and refine problem solving approaches.

In addition, informal feedback was collected from peers regarding the usability and layout of the user interface. This feedback was used to improve the overall user friendliness and clarity of the application.

All final design decisions, implementation choices, and report content were review and finalized by the developer.