Fast-Food Order System

CS121 course project

**Iakovos-Ioannis Anagnostakis** & **Maksym Rudenko**

Repository: https://github.com/Maksimrudenk/FastFood-GUI.git

Description:

Java-swing desktop application with graphical user interface.

The Fast-Food Order System is a Java application designed to facilitate the ordering process in a restaurant. It provides a graphical user interface (GUI) for users to order various food items such as burgers, side dishes, and drinks. The system allows customization of food items and calculates the total price of the order.

On the main window user can choose one of the dishes on the menu, after the corresponding button was clicked the window for customization for chosen dish is open. There the user can customize, according to his/her preferences, and create the dish unit, that will be added to the cart. On the main page the total price of all dish-units will be displayed. The process can be repeated.

Architecture:

The application follows a modular architecture, with clear separation of concerns between different components. The main components of the system include:

MainFrame (RestaurantOrderSystem): This is the main frame of the application. It initializes the GUI, handles user interactions, and displays the total price of the order.

FoodFrame: This frame provides a window for the customization of food items. In interaction with AttributeBox and PanelBuilder, dynamically generates UI elements (using java reflection) based on the type of food selected by the user.

Cart: This class serves as the storage for created food items. It maintains a list of food items added to the order and updates the total price displayed in the main frame.

Food Types (Burger, SideDish, Drink): These classes represent different types of food items available for ordering. Each food type extends the abstract class Food and implements the calculatePrice() method to calculate the price based on customizable components.

ComponentAbstract: This abstract class represents customizable components of food items (e.g., meat, sauce, topping). It defines the structure for storing the selected variant and extracting the price from it.

Design Choices:

Modularity: The system is designed with modularity in mind, making it easy to extend and maintain. Each component has a well-defined responsibility, which promotes code reusability and scalability.

Object-Oriented Design: The use of object-oriented principles such as inheritance, encapsulation, and polymorphism enhances the flexibility and maintainability of the codebase.

Dynamic UI Generation: The FoodFrame dynamically generates UI elements based on the selected food type, allowing for a flexible and intuitive user experience.

Exception Handling: The program includes exception handling mechanisms to gracefully handle errors that may occur during the execution, ensuring robustness and reliability.

Lombok Annotations: The use of Lombok annotations such as @Data reduces boilerplate code for getter/setter methods and enhances code readability.

Possibilities for future:

Enhanced Customization: The system could be enhanced to support more advanced customization options for food items, such as additional ingredients or dietary preferences.

User Authentication: Implementing user authentication and profiles could enable features like order history and personalized recommendations.

Database Integration: Integrating a database backend could allow for persistent storage of orders and menu items, enabling features like inventory management and reporting.

Menu expansion: dynamic UI generation and Food form allow for easiness in adding new dishes to the menu.

Responsibilities:

**Iakovos-Ioannis Anagnostakis**

* Main Idea
* Primary coding
* Presentation development

**Maksym Rudenko**

* Extended coding
* Code merging
* Documentation writing