Class and Method Documentation Overview

Introduction:

This document provides a comprehensive overview of the classes and methods implemented in the restaurant ordering system project. Each class and interface is meticulously designed to solve specific challenges using various design patterns. The following sections detail the purpose, structure, and functionality of each component, ensuring clarity and understanding of their roles within the overall system. This documentation serves as a valuable resource for developers and contributors, offering insights into the design decisions and technical implementations that drive the project.

MenuItemAPI

- **Purpose**: Defines the common interface for menu items, which includes both individual items and composite items (e.g., combo meals). It ensures that all menu items share a common set of methods for interaction.
- **Key Methods**:
    - getName(): Returns the name of the menu item. This is used to display or identify the item in various contexts.
    - getPrice(): Returns the price of the menu item. This method is crucial for calculating the total cost of an order.
    - prepare(): An abstract method that must be implemented by concrete menu items to define how they are prepared. This could involve cooking or assembling the item.

MenuComponentAPI

- **Purpose**: Extends MenuItemAPI to support composite menu components. It allows for hierarchical structures where a menu component can contain other MenuItemAPI objects. This is useful for creating complex menu items like combo meals or special deals.
- **Key Methods**:
    - add(MenuItemAPI menuComponent): Adds a menu component (e.g., a single item or another composite item) to this component. Useful for building hierarchical menu structures.
    - remove(MenuItemAPI menuComponent): Removes a menu component from this component. Allows for dynamic modification of the menu structure.
    - getComponents(): Returns a list of all components contained within this composite. Useful for iterating over the items in a composite menu component.

ComboMeal

- **Purpose**: Represents a composite menu item consisting of multiple MenuItemAPI objects. It allows for bundling multiple items together, often at a discounted price.
- **Key Methods**:
    - addItem(MenuItemAPI item): Adds an item to the combo meal. Allows for building the combo meal by adding different menu items.
    - getPrice(): Calculates the total price of the combo meal, often including a discount. This method aggregates the prices of all included items.
    - prepare(): Prepares all items in the combo meal. This method ensures that each item is prepared as part of the combo.

Drink

- **Purpose**: Represents a single drink item on the menu. It provides specific attributes and behaviors related to drink items.
- **Key Methods**:
    - prepare(): Prepares the drink. This might involve mixing or chilling the drink.
    - getName(): Returns the name of the drink, useful for display and identification.
    - getPrice(): Returns the price of the drink, used for order calculations.
    - toString(): Returns a string representation of the drink, including its name and price.

DrinkBuilderAPI

- **Purpose**: Defines the builder API for creating Drink objects. It provides methods to set the attributes of a drink before constructing it.
- **Key Methods**:
    - setId(int id): Sets the ID of the drink. Useful for identifying the drink in inventory or orders.
    - setCost(double cost): Sets the cost of the drink. Defines the price of the drink.
    - setName(String name): Sets the name of the drink. Used for display and identification.
    - build(): Constructs and returns a Drink object with the specified attributes.

DrinkBuilder

- **Purpose**: Implements DrinkBuilderAPI to provide a concrete builder for Drink objects. It encapsulates the construction process of a drink, allowing for flexible and controlled creation.
- **Key Methods**:
    - setId(int id): Sets the ID and returns the builder for method chaining.
    - setCost(double cost): Sets the cost and returns the builder for method chaining.
    - setName(String name): Sets the name and returns the builder for method chaining.
    - build(): Returns a new Drink object with the attributes set by the builder.

DrinkBuilderFactoryAPI

- **Purpose**: Defines the factory API for creating instances of DrinkBuilder. It abstracts the process of obtaining a builder instance.
- **Key Methods**:
    - getDrinkBuilder(): Returns a new instance of DrinkBuilderAPI. This method provides access to a builder for creating Drink objects.

DrinkBuilderFactory

- **Purpose**: Implements DrinkBuilderFactoryAPI to provide concrete instances of DrinkBuilder. It manages the creation of builder instances.
- **Key Methods**:
    - getDrinkBuilder(): Returns a new DrinkBuilder instance. This method provides access to the actual builder implementation.

SingletonDrinkBuilderFactoryEnum

- **Purpose**: Ensures a single instance of DrinkBuilderFactory using the enum singleton pattern. This pattern ensures that only one instance of the factory exists, providing a global point of access.
- **Key Methods**:
    - getDrinkBuilderFactory(): Returns the single instance of DrinkBuilderFactoryAPI. This method provides access to the singleton instance.

Fries

- **Purpose**: Represents a single fries item on the menu, with attributes and methods specific to fries.
- **Key Methods**:
    - prepare(): Prepares the fries, which might involve cooking or frying.
    - getName(): Returns the name of the fries for display and identification.
    - getPrice(): Returns the price of the fries, used for order calculations.
    - toString(): Returns a string representation of the fries, including its name and price.

FriesBuilderAPI

- **Purpose**: Defines the builder API for creating Fries objects. It provides methods to set the attributes of fries before constructing them.
- **Key Methods**:
    - setId(int id): Sets the ID of the fries for identification.
    - setCost(double cost): Sets the cost of the fries.
    - setName(String name): Sets the name of the fries.
    - build(): Constructs and returns a Fries object with the specified attributes.

FriesBuilder

- **Purpose**: Implements FriesBuilderAPI to provide a concrete builder for Fries objects. It encapsulates the creation process of fries, allowing for flexible configuration.
- **Key Methods**:
    - setId(int id): Sets the ID and returns the builder for method chaining.
    - setCost(double cost): Sets the cost and returns the builder for method chaining.
    - setName(String name): Sets the name and returns the builder for method chaining.
    - build(): Returns a new Fries object with the specified attributes.

FriesBuilderFactoryAPI

- **Purpose**: Defines the factory API for creating FriesBuilder instances. It abstracts the creation of builder instances.
- **Key Methods**:
    - getFriesBuilder(): Returns a new instance of FriesBuilderAPI.

FriesBuilderFactory

- **Purpose**: Implements FriesBuilderFactoryAPI to provide concrete instances of FriesBuilder. It manages the creation of builder instances.
- **Key Methods**:
    - getFriesBuilder(): Returns a new FriesBuilder instance.

SingletonFriesBuilderFactoryEnum

- **Purpose**: Ensures a single instance of FriesBuilderFactory using the enum singleton pattern. Provides a global point of access to the factory.
- **Key Methods**:
    - getFriesBuilderFactory(): Returns the single instance of FriesBuilderFactoryAPI.

Burger

- **Purpose**: Represents a single burger item on the menu, with attributes and methods specific to burgers.
- **Key Methods**:
    - prepare(): Prepares the burger, which might involve cooking or assembling.
    - getName(): Returns the name of the burger for display and identification.
    - getPrice(): Returns the price of the burger, used for order calculations.
    - toString(): Returns a string representation of the burger, including its name and price.

BurgerBuilderAPI

- **Purpose**: Defines the builder API for creating Burger objects. It provides methods to set the attributes of a burger before constructing it.
- **Key Methods**:
    - setId(int id): Sets the ID of the burger for identification.
    - setCost(double cost): Sets the cost of the burger.
    - setName(String name): Sets the name of the burger.
    - build(): Constructs and returns a Burger object with the specified attributes.

BurgerBuilder

- **Purpose**: Implements BurgerBuilderAPI to provide a concrete builder for Burger objects. Encapsulates the burger creation process.
- **Key Methods**:
    - setId(int id): Sets the ID and returns the builder for method chaining.
    - setCost(double cost): Sets the cost and returns the builder for method chaining.
    - setName(String name): Sets the name and returns the builder for method chaining.
    - build(): Returns a new Burger object with the specified attributes.

BurgerBuilderFactoryAPI

- **Purpose**: Defines the factory API for creating BurgerBuilder instances. Abstracts the creation of builder instances.
- **Key Methods**:
    - getBurgerBuilder(): Returns a new instance of BurgerBuilderAPI.

BurgerBuilderFactory

- **Purpose**: Implements BurgerBuilderFactoryAPI to provide concrete instances of BurgerBuilder. Manages the creation of builder instances.
- **Key Methods**:
    - getBurgerBuilder(): Returns a new BurgerBuilder instance.

SingletonBurgerBuilderFactoryEnum

- **Purpose**: Ensures a single instance of BurgerBuilderFactory using the enum singleton pattern. Provides a global point of access to the factory.
- **Key Methods**:
    - getBurgerBuilderFactory(): Returns the single instance of BurgerBuilderFactoryAPI.

OrderItemDecoratorAPI

- **Purpose**: Abstract class for decorators that add additional functionality to MenuItemAPI objects. It allows for extending the behavior of menu items dynamically.
- **Key Methods**:
    - getName(): Returns the name of the decorated item, including any modifications introduced by the decorator.
    - getPrice(): Returns the price of the decorated item, including any additional costs introduced by the decorator.

ExtraCheeseDecorator

- **Purpose**: Concrete decorator that adds extra cheese to a MenuItemAPI object. It extends the base functionality of a menu item by adding cheese.
- **Key Methods**:
    - getName(): Returns the name with "Extra Cheese" appended, reflecting the added ingredient.
    - getPrice(): Adds the cost of extra cheese to the original price of the menu item.

SpecialSauceDecorator

- **Purpose**: Concrete decorator that adds special sauce to a MenuItemAPI object. It extends the base functionality by adding a special sauce.
- **Key Methods**:
    - getName(): Returns the name with "Special Sauce" appended, reflecting the added ingredient.
    - getPrice(): Adds the cost of special sauce to the original price of the menu item.

PaymentProcessorAPI

- **Purpose**: Defines the interface for processing payments using different payment strategies. It allows for flexibility in how payments are handled.
- **Key Methods**:
    - setStrategy(PaymentStrategy strategy): Sets the payment strategy to be used for processing payments.
    - processPayment(double amount): Processes the payment using the set strategy, facilitating the payment process.

PaymentProcessor

- **Purpose**: Implements PaymentProcessorAPI to manage payment processing. It delegates the actual payment handling to a PaymentStrategy.
- **Key Methods**:
    - setStrategy(PaymentStrategy strategy): Sets the payment strategy and prepares the processor for payment.
    - processPayment(double amount): Executes the payment process using the selected strategy.

PaymentStrategyAPI

- **Purpose**: Defines the interface for various payment strategies. It standardizes the way payments are handled across different methods (e.g., credit card, cash).
- **Key Methods**:
    - pay(double amount): Executes the payment for the specified amount. Each implementation defines how the payment is processed.

CreditCardPayment

- **Purpose**: Implements PaymentStrategyAPI for processing payments via credit card. It provides the logic specific to credit card transactions.
- **Key Methods**:
    - pay(double amount): Prints a message indicating that the payment is processed using a credit card, simulating the transaction.

CashPayment

- **Purpose**: Implements PaymentStrategyAPI for processing payments via cash. It provides the logic specific to cash transactions.
- **Key Methods**:
    - pay(double amount): Prints a message indicating that the payment is made with cash, simulating the transaction.

OrderObservableAPI

- **Purpose**: Defines the observable interface for orders, supporting the observer pattern to notify observers of changes in the order status.
- **Key Methods**:
    - addItemToOrder(MenuItemAPI item): Adds an item to the order and notifies observers about the change.
    - removeItemFromOrder(MenuItemAPI item): Removes an item from the order and notifies observers about the change.
    - getItems(): Returns a list of items in the order, useful for reviewing the order contents.
    - notifyObservers(): Notifies all registered observers about changes in the order.

OrderObserverAPI

- **Purpose**: Defines the observer interface for monitoring changes in orders. It receives notifications when the observable order changes.
- **Key Methods**:
    - update(OrderObservableAPI order): Called when the observable order is updated. The observer can take action based on the changes.

KitchenStaff

- **Purpose**: Implements OrderObserverAPI to react to changes in the order, such as when new items are added or removed. Represents staff responsible for preparing the order.
- **Key Methods**:
    - update(OrderObservableAPI order): Processes updates from the order observable, such as preparing new items or adjusting the preparation queue.

Restaurant

Description: The Restaurant class is a singleton that manages the menu and orders within the restaurant. It provides methods for adding and removing menu items, handling orders, and processing payments. The class ensures there is only one instance of the restaurant throughout the application, which is accessed through the getInstance() method. It maintains a list of menu items and orders, allowing for operations such as showing the menu, adding orders, and processing payments for specific orders.

Key Methods:

- **getInstance()**: Returns the single instance of the Restaurant class.
- **addMenuItem(MenuItemAPI item)**: Adds a menu item to the restaurant's menu.
- **removeMenuItem(MenuItemAPI item)**: Removes a menu item from the restaurant's menu.
- **getItemById(int id)**: Retrieves a menu item by its ID.
- **addOrder(OrderAbservableAPI order)**: Adds an order to the list of orders and places the order.
- **removeOrder(OrderAbservableAPI order)**: Removes an order from the list of orders.
- **payOrder(OrderAbservableAPI order, double amount)**: Processes payment for a specific order and removes it from the list.
- **showMenu()**: Displays all menu items with their names and prices.