

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра программных систем

**Домашнее задание**

**Построение и исследование программной системы на основе шаблонов проектирования**

Выполнил: Щербаков A. А.

Группа № K33202

Проверил: Осипов Н. А.

Санкт-Петербург

2020

**Задание:**

Изучить шаблоны проектирования GoF, закрепить основы разработки систем на их основе и освоить реализацию паттернов проектирования в языке Python.

**Ход работы:**

Создание объектов пицц было реализовано с помощью *фабричного метода*. Таким образом существует абстрактный класс PizzaType, который объявляет фабричный метод create(). Этот метод реализован в классах ExoticPizzaType и ItalianPizzaType, которые создают соответствующие объекты в зависимости от аргумента *фабричного метода*.

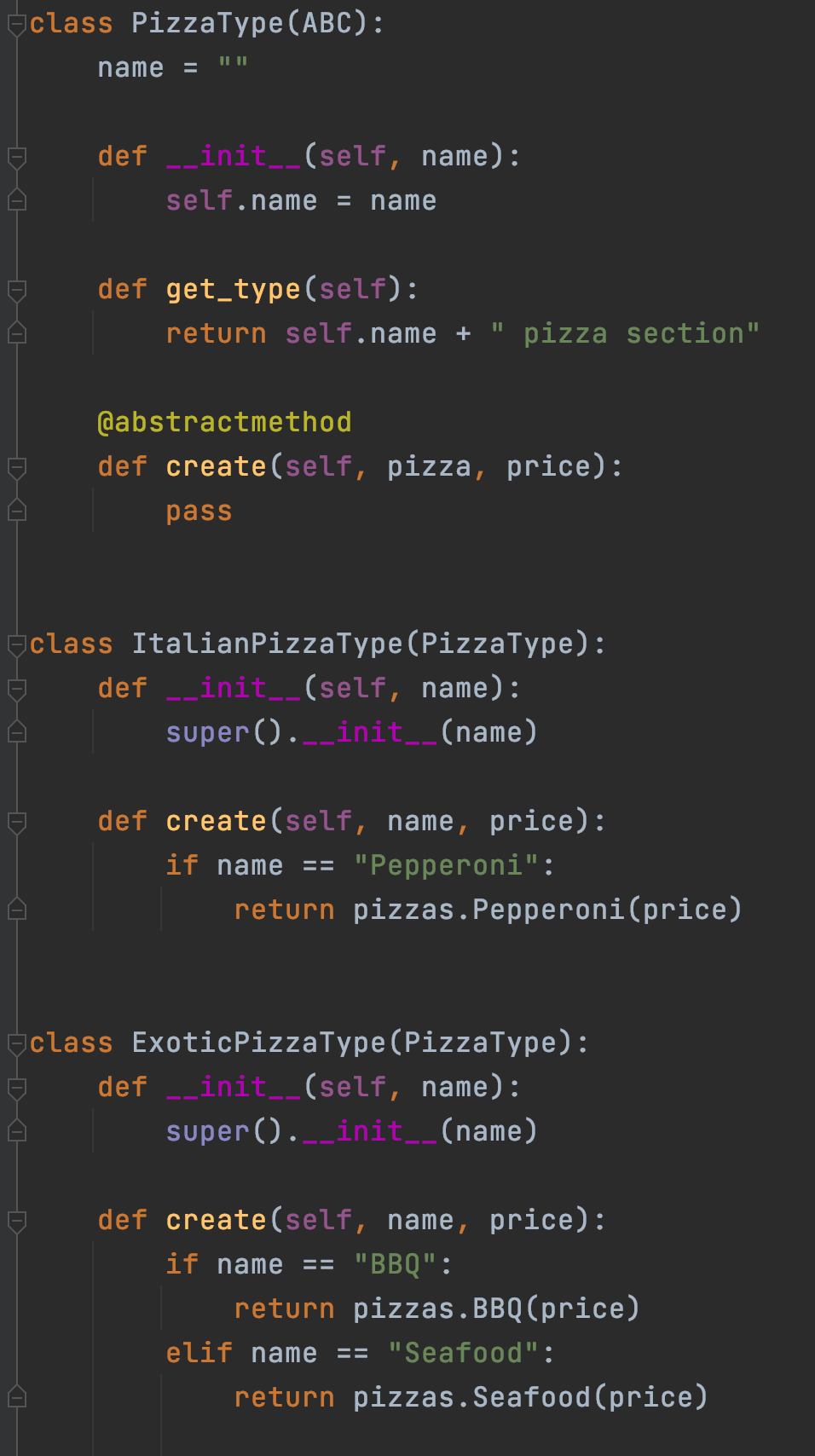


Рисунок 1 — PizzaType и реализации

Сами пиццы являются реализациями абстрактного класса Pizza с полями названия, теста, соуса и начинки, а также методами приготовления bake() и упаковки pack(). Каждая пицца реализует абстрактный метод bake() со своими специфическими характеристиками, тем самым используя *шаблонный метод*.



Рисунок 2 - Абстрактный класс Pizza и его реализация в Pepperoni

Список доступных для заказа пицц находится в файле available.txt в формате <тип>:<название>:<стоимость в евро>. С файлом работает класс-одиночка Menu, асинхронно обрабатывающий его содержимое и возвращающий объекты пицц в методе update().



Рисунок 3 - Класс Menu

Была смоделирована ситуация, когда требуется предобрабатывать данные файла и конвертировать стоимость в доллары. Реализация метода описана в классе LocalPreprocessors. Для осуществления правильного метода был создан *адаптер* классов MenuAdapter, наследующий сразу от двух классов. Все последующие обращения к меню теперь осуществляются с помощью адаптера.



Рисунок 4 - Адаптер класса LocalPreprocessors

Данные заказа хранятся в классе Order. Методы add() и get\_total() позволяют добавить пиццу в заказ и получить сумму. Для расчёта скидки и проверки на ограничение по количеству используются исключения.

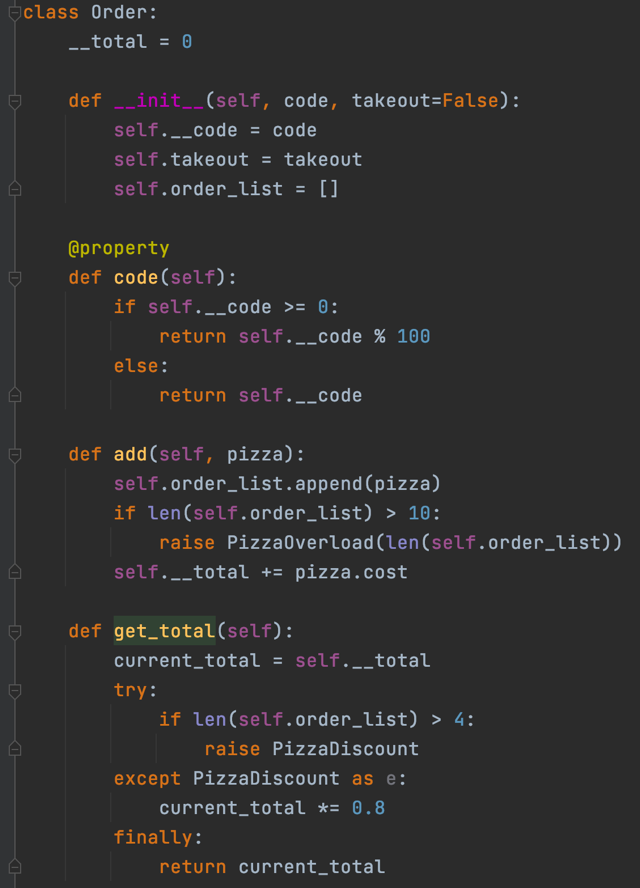


Рисунок 5 - Класс Order

Класс Terminal отвечает за интерфейс приложения. Он обобщает все внутренние методы и приводит всё к простому визуальному оформлению, выступая *фасадом* для всей внутренней логики. Помимо этого, терминал работает по принципам *состояния*, отрисовывая экраны приложения в зависимости от решений пользователя. Экраны наследуются от абстрактного класса Screen и реализуют основные методы обновления интерфейса или контроллера логики. Каждый экран имеет поле, где терминал хранится в качестве контекста, а все переходы осуществляются вызовом его метода segue\_to(screen), исполнение которого делегируется экранам.



Рисунок 6 - Абстрактный класс Screen и часть класса StartScreen

Терминал также использует лог для документирования процессов и вывода их на экран. Класс Log является одиночкой, которая записывает позиции в поле history методом write(message).

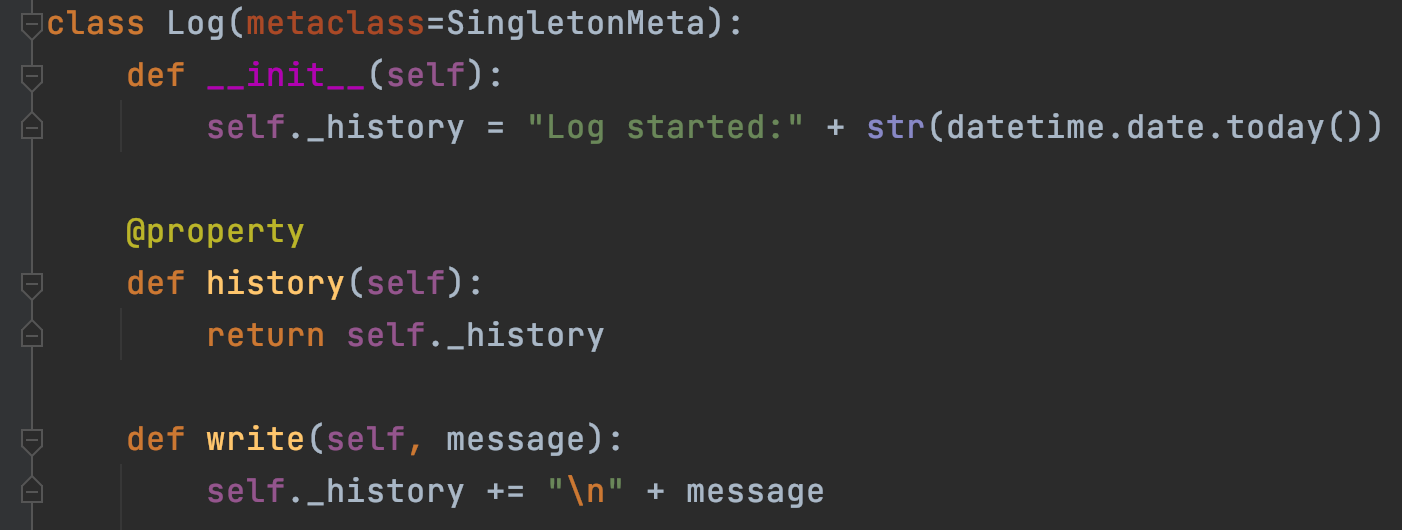


Рисунок 7 - Класс Log

Все одиночки проекта реализованы с помощью метакласса MetaSingleton, который гарантирует единственность инициализации классов.

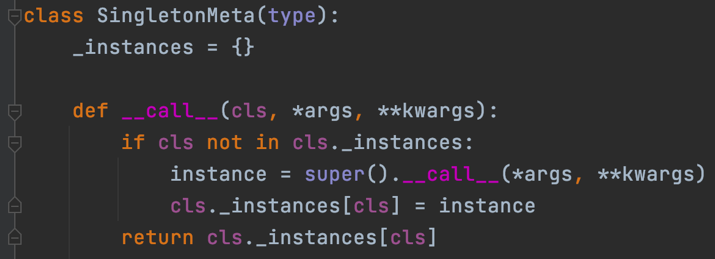


Рисунок 8 - Метакласс MetaSingleton

Проект инициализируется классом Client, который создаёт экземпляр терминала и позволяет его запустить.

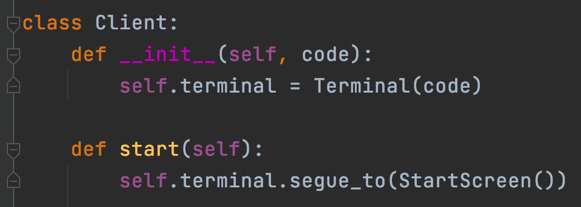


Рисунок 9 - Класс Client

Запущенное приложение представляет из себя визуальный интерфейс Tkinter и позволяет выбирать тип заказа, наполнять его пиццами, завершать и перезапускать.



Рисунок 10 – Приветствие и тип заказа

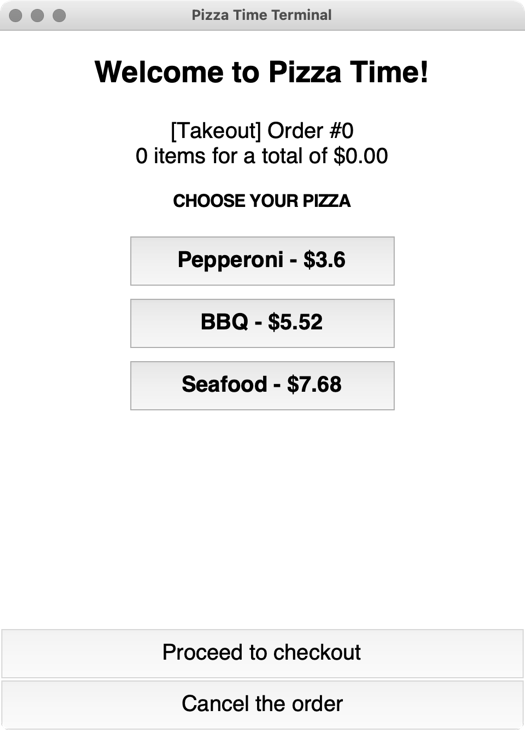


Рисунок 11 - Выбор пиццы

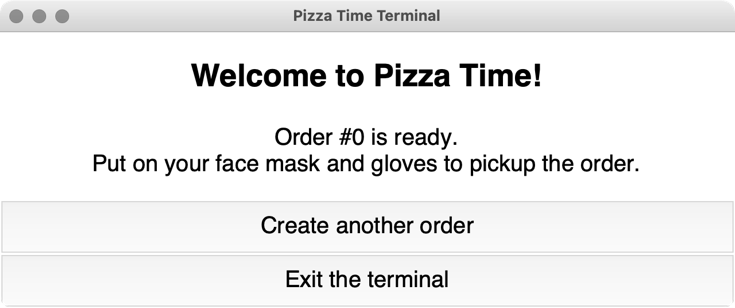


Рисунок 12 - Завершение заказа

Финальная UML-диаграмма проекта со всеми использованными классами и паттернами проектирования GoF:

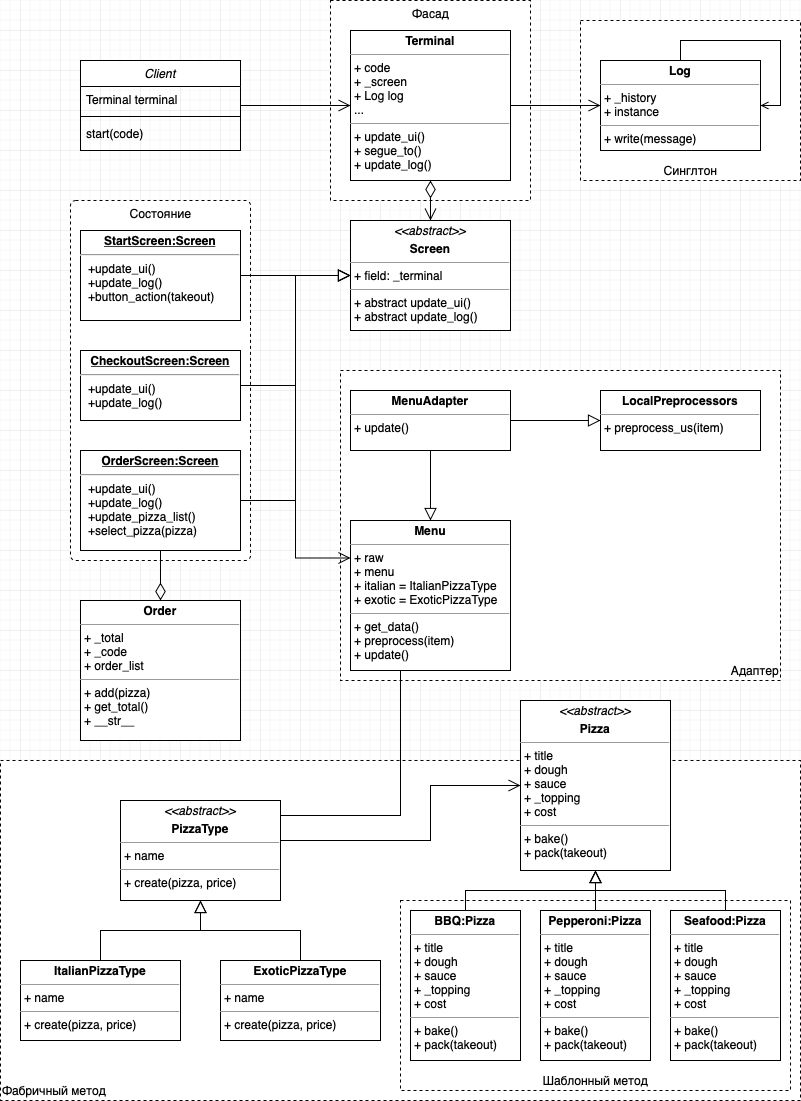


Рисунок 13 - Диаграмма классов

**Вывод.**

Была создана система заказа пиццы на языки Python с использованием паттернов проектирования Gang-of-Four и описана его структура в виде UML-диаграммы.