

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 42

КУРСОВАЯ РАБОТА
ЗАЩИЩЕНА С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

старший преподаватель

должность, уч. степень, звание

подпись, дата

С. Ю. Гуков

инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

МАГАЗИН ТОВАРОВ ИЛИ УСЛУГ

по дисциплине:

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326

подпись, дата

Г. С. Томчук

инициалы, фамилия

Санкт-Петербург 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Индивидуальное задание	4
2 Краткое описание хода разработки и назначение используемых технологий.....	6
3 Пользовательская документация.....	8
4 Техническая документация.....	10
4.1 Общая архитектура	10
4.2 Описание основных алгоритмов.....	10
4.3 Описание основных классов, функций и переменных	11
4.4 Описание переменных данных	13
4.5 Используемые паттерны.....	13
4.6 Алгоритмы обработки ошибок	13
4.7 Тестирование	13
5 Описание проведения модульного тестирования.....	14
5.1 Организация тестирования.....	14
5.2 Описание тестовых модулей	14
5.3 Принципы тестирования.....	15
5.4 Запуск тестов	15
5.5 Роль модульного тестирования.....	15
6 Результаты работы программы с примерами разных сценариев	17
ЗАКЛЮЧЕНИЕ.....	21
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	23
ПРИЛОЖЕНИЕ А	24

ВВЕДЕНИЕ

В современном программировании от разработчиков требуется не просто умение писать рабочий код, но и глубокое понимание архитектурных подходов, проектных шаблонов и принципов качественной разработки. В рамках данной курсовой работы ставится задача создать программный продукт средней сложности, моделирующий процесс покупки товаров или услуг в магазине. Главная цель проекта — построение продуманной архитектуры и закрепление навыков использования различных технологий программирования.

Разрабатываемая программа должна позволять пользователю — покупателю с бонусной картой, кошельком и индивидуальным списком покупок — взаимодействовать с магазином. Пользователь сможет добавлять товары или услуги в корзину, при необходимости взвешивать их перед покупкой, и оплачивать заказ с помощью наличных, банковской карты или бонусов. Программа также должна учитывать ситуации нехватки средств или отсутствия взвешивания, предоставляя возможность удалить часть товаров из корзины до тех пор, пока покупка не станет возможной. Таким образом, пользователь должен всегда иметь возможность совершить хотя бы частичную покупку.

Проект может быть реализован как консольное приложение с командно-текстовым интерфейсом или с графическим интерфейсом пользователя (GUI), а также с использованием любого удобного языка программирования.

Структура проекта должна соответствовать принципам объектно-ориентированного программирования и SOLID. В реализации необходимо применить шаблоны проектирования и архитектурные подходы, а интерфейс — быть интуитивно понятным и удобным для пользователя.

1 Индивидуальное задание

Требуется разработать программное приложение, моделирующее процесс покупки товаров или услуг в магазине. Пользователь выступает в роли покупателя, имеющего бонусную карту, кошелек и список необходимых покупок. В рамках работы приложения покупатель может выбирать интересующие товары или услуги, добавлять их в корзину, при необходимости взвешивать перед покупкой, а также исключать ненужные позиции из корзины.

После формирования заказа пользователь получает возможность оплатить покупку. Поддерживаются различные способы оплаты: наличные, бонусные баллы, а также их комбинированное использование — например, часть суммы может быть оплачена баллами, а оставшаяся сумма — деньгами. Программа должна корректно обрабатывать ситуации, при которых: у пользователя недостаточно средств на оплату полной стоимости заказа; в корзине находятся не взвешенные товары. В таких случаях пользователь должен иметь возможность удалить часть товаров из корзины, чтобы завершить покупку с доступными средствами. Обязательным условием является возможность успешного совершения хотя бы частичной покупки — сценарий, при котором пользователь уходит ни с чем, исключается.

Проект может быть реализован как консольное приложение с командно-текстовым интерфейсом, либо как программа с графическим пользовательским интерфейсом (GUI). Язык программирования выбирается разработчиком.

Требования к структуре проекта:

- Реализация ключевых принципов объектно-ориентированного программирования: наследование, инкапсуляция, полиморфизм, абстракция;
- Соблюдение принципов SOLID при проектировании архитектуры;
- Наличие удобного и понятного интерфейса — текстового или графического;

- Использование хотя бы одного архитектурного шаблона;
- Внедрение не менее двух паттернов проектирования.

2 Краткое описание хода разработки и назначение используемых технологий

Разрабатываемый программный продукт представляет собой CLI-приложение, реализующее поведение покупателя и магазина, где осуществляется продажа товаров и предоставление услуг. Проект написан на языке программирования Python с использованием командно-текстового интерфейса. Для взаимодействия с пользователем применена библиотека Rich, позволяющая реализовать удобный и визуально приятный вывод в терминал. Для модульного тестирования отдельных компонентов проекта была выбрана библиотека pytest.

Вся система реализована по архитектурному паттерну MVC (Model-View-Controller). Модель отвечает за бизнес-логику, хранение и обработку данных, представление (View) — за взаимодействие с пользователем через rich-интерфейс, а контроллер управляет сценариями и связывает модель с представлением. Такой подход позволил легко расширять и модифицировать функционал без риска нарушить работу других частей приложения.

В проекте активно используется полиморфизм: например, для товаров и услуг определён общий интерфейс, а конкретные классы реализуют свои версии методов. Это позволило обрабатывать разные типы объектов единообразно, не заботясь о деталях их реализации. Инкапсуляция обеспечила защиту данных и удобство работы с объектами.

В системе оплаты реализован паттерн «Мост» (Bridge), который разделяет абстракцию платежа и конкретные способы оплаты, что делает добавление новых методов оплаты простым и безопасным. Для создания объектов оплаты применяется паттерн «Абстрактная фабрика» (Abstract Factory), позволяющий централизованно управлять созданием экземпляров различных классов оплаты. Также в проекте реализован паттерн «Стратегия» (Strategy) — разные способы оплаты инкапсулируются в отдельных классах и могут подставляться в бизнес-логику по необходимости.

В результате проект получился гибким, расширяемым и легко

поддерживаемым, с чётким разделением ответственности между компонентами и активным использованием современных паттернов проектирования и принципов ООП.

3 Пользовательская документация

Данная программа представляет собой консольный магазин с дружелюбным русскоязычным интерфейсом, предназначенный для покупки товаров и услуг в интерактивном режиме. После запуска приложения пользователю предлагается ввести своё имя, чтобы система могла загрузить или создать индивидуальную сессию с сохранением корзины и баланса. Это позволяет каждому пользователю иметь собственную корзину и профиль, которые автоматически сохраняются и восстанавливаются при последующих входах.

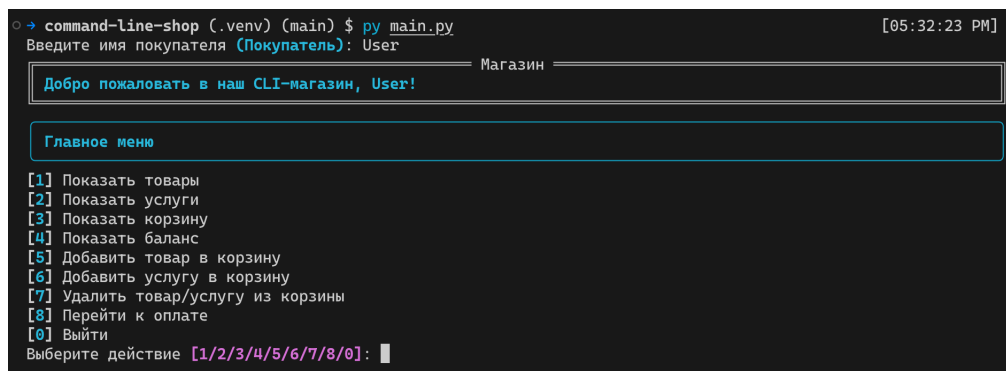


Рисунок 1 — Главное меню приложения

Основное взаимодействие с программой происходит через главное меню, где пользователь может выбрать одно из доступных действий: просмотреть список товаров, ознакомиться с перечнем услуг, проверить содержимое своей корзины, узнать текущий баланс по всем способам оплаты (наличные, карта, бонусы), добавить товар или услугу в корзину, удалить ненужную позицию, а также перейти к оплате. Все действия сопровождаются подробными подсказками и визуальным оформлением с помощью библиотеки `rich`, что делает работу с программой максимально понятной и приятной.

Для добавления товара или услуги в корзину достаточно выбрать соответствующий пункт меню и указать номер интересующей позиции. Для взвешиваемых товаров программа предложит ввести необходимый вес. Корзина отображается в виде таблицы с указанием всех позиций и итоговой суммы. При необходимости любую позицию можно удалить.

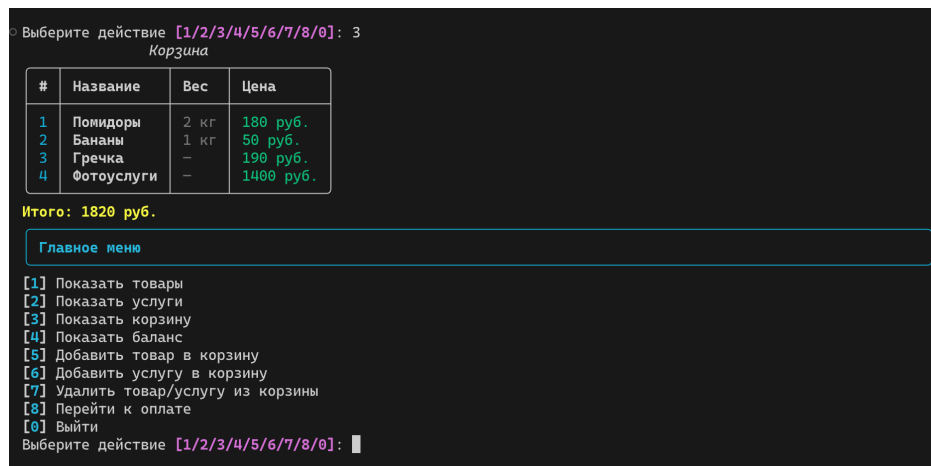


Рисунок 2 — Просмотр корзины

Оплата осуществляется через удобный интерфейс выбора способа оплаты. Пользователь может комбинировать оплату наличными, картой и бонусами, указывая желаемую сумму для каждого способа. Программа автоматически проверяет доступность средств и корректность введенных данных, не позволяя совершить ошибку. В случае успешной оплаты корзина очищается, а остатки товаров и баланс пользователя обновляются.

Все изменения автоматически сохраняются в сессии пользователя, что позволяет продолжить покупки с того же места при следующем запуске программы. Программа поддерживает одновременную работу нескольких пользователей, обеспечивая каждому индивидуальное пространство для покупок.

В случае возникновения ошибок или некорректного ввода программа выводит подробные сообщения и рекомендации по дальнейшим действиям. Таким образом, даже неопытный пользователь сможет легко освоить работу с CLI-магазином и совершать покупки в комфортной интерактивной среде.

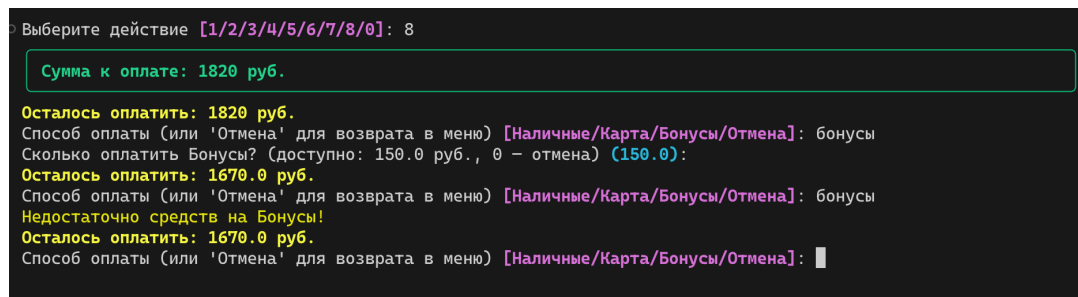


Рисунок 3 — Процесс оплаты товаров

4 Техническая документация

4.1 Общая архитектура

Проект реализован по паттерну MVC (Model-View-Controller), что обеспечивает разделение бизнес-логики, пользовательского интерфейса и управляющих сценариев. Все данные хранятся в формате JSON, поддерживается многопользовательский режим с индивидуальными сессиями. В Приложении А приведена часть исходного кода приложения. Полный исходный код расположен в репозитории по ссылке: <https://github.com/grigorijtomczuk/command-line-shop>.

4.2 Описание основных алгоритмов

Загрузка и сохранение сессии пользователя:

- При запуске программы пользователь вводит имя. Система ищет сессию по имени в файле sessions.json. Если сессия найдена, восстанавливаются корзина и профиль пользователя, иначе создаётся новая сессия с начальными значениями баланса.

- При выходе или оплате корзины данные пользователя и корзины сохраняются в sessions.json. Остатки товаров и услуг обновляются в products.json и services.json.

Добавление товара/услуги в корзину:

- Пользователь выбирает товар или услугу по номеру из списка.
- Для взвешиваемых товаров запрашивается вес, для штучных — количество по умолчанию 1.
- Проверяется наличие товара на складе, после чего товар добавляется в корзину, а его остаток уменьшается.

Удаление из корзины:

- Пользователь выбирает позицию для удаления.
- Товар возвращается на склад (увеличивается остаток), позиция удаляется из корзины.

Оплата:

- Пользователь выбирает способ оплаты (наличные, карта, бонусы) и

сумму для каждого способа.

- Система проверяет корректность суммы и наличие средств.
- Оплата проводится с помощью паттерна «Мост»: объект `Payment` делегирует выполнение конкретному `PaymentMethod`.
- При успешной оплате корзина очищается, баланс пользователя обновляется.

4.3 Описание основных классов, функций и переменных

`shop.py` (`ShopModel`):

- `products`: список объектов `Product`, загружается из `products.json`.
- `services`: список объектов `Service`, загружается из `services.json`.
- `customer`: объект `Customer`, хранит профиль пользователя.
- `cart`: объект `Cart`, хранит текущую корзину пользователя.
- `save_session()`: сохраняет сессию пользователя и актуальные остатки товаров/услуг.
- `load_session(path, customer_name)`: загружает сессию по имени пользователя.
- `add_to_cart(idx)`, `add_weighted_to_cart(idx, weight)`: добавляют товар в корзину.
- `add_service_to_cart(idx)`: добавляет услугу в корзину.
- `remove_from_cart(idx)`: удаляет позицию из корзины.
- `checkout(payments)`: проводит оплату корзины.

`product.py` (`Product`) — класс товара, наследует `ShopItem`:

- `requires_weighing`: флаг, требует ли товар взвешивания.
- `get_total_price()`: возвращает стоимость товара (с учётом веса).

`service.py` (`Service`) — класс услуги, наследует `ShopItem`:

- `cost`: стоимость услуги.
- `get_total_price()`: возвращает стоимость услуги.

`cart.py` (`Cart`):

- `items`: список объектов `ShopItem` (товары и услуги).
- `add_item(item)`: добавляет позицию в корзину.

- `remove_item(index)`: удаляет позицию по индексу.
- `calculate_total()`: возвращает общую сумму корзины.

`payment.py`:

- `PaymentMethod`: абстрактный класс для способов оплаты.
- `CashPayment`, `CardPayment`, `BonusPayment`: конкретные реализации способов оплаты.
- `Payment`: абстракция (паттерн «Мост»), делегирует выполнение `PaymentMethod`.

`transaction.py`:

- `Transaction`: класс для проведения оплаты.
- `attempt_payment(payments)`: принимает список платежей, проводит оплату через объекты `Payment` или `PaymentMethod`.

`factory.py`:

- `PaymentFactory (ABC)`: абстрактная фабрика для создания способов оплаты.
- `ConcretePaymentFactory`: конкретная фабрика, создаёт объекты `CashPayment`, `CardPayment`, `BonusPayment` по строковому коду.

`session.py`:

- `save_session(customer, cart_items, products, services, ...)`: сохраняет сессию пользователя и остатки товаров/услуг.
- `load_session(customer_id, ...)`: загружает сессию пользователя по `id`.

`utils.py`:

- `load_products(path)`: загружает список товаров из JSON.
- `load_services(path)`: загружает список услуг из JSON.

`shop_view.py`:

- Класс для вывода информации пользователю через rich-интерфейс.
- `prompt_str`, `prompt_int`, `prompt_action`, `prompt_payment_method`, `prompt_payment_amount`: методы для ввода данных.
- `show_products`, `show_services`, `show_cart`, `show_balance`, `show_message`, `show_panel`, `show_error`: методы для вывода информации.

shop_controller.py

- ShopController: управляет сценарием работы пользователя.
- main_menu(): отображает главное меню.
- run(): основной цикл работы.
- handle_add_to_cart, handle_add_service_to_cart,

handle_remove_from_cart, handle_checkout: обработчики пользовательских действий.

4.4 Описание переменных данных

- products.json: список всех товаров с остатками.
- services.json: список всех услуг.
- sessions.json: список сессий пользователей (корзина и профиль для каждого пользователя).

4.5 Используемые паттерны

- Мост (Bridge) — абстракция Payment и реализация PaymentMethod.
- Абстрактная фабрика (Abstract Factory) — создание способов оплаты.
- Стратегия (Strategy) — разные способы оплаты инкапсулируются в отдельных классах.

4.6 Алгоритмы обработки ошибок

Все пользовательские ошибки (некорректный ввод, недостаточно средств, пустая корзина) обрабатываются через собственные исключения и выводятся пользователю в понятном виде.

4.7 Тестирование

В папке tests реализованы юнит-тесты для корзины, оплаты и транзакций, что обеспечивает надёжность и корректность работы основных компонентов системы.

5 Описание проведения модульного тестирования

В проекте реализовано модульное тестирование для ключевых компонентов системы, что обеспечивает надёжность и корректность работы бизнес-логики CLI-магазина. Тесты размещены в директории tests и покрывают основные сценарии использования корзины, оплаты и транзакций.

5.1 Организация тестирования

Для тестирования используется фреймворк pytest, который позволяет легко запускать и расширять набор тестов. Каждый тестовый модуль (test_cart.py, test_payment.py, test_transaction.py) содержит отдельные функции, проверяющие корректность работы соответствующих классов и методов.

5.2 Описание тестовых модулей

1. test_cart.py

Этот модуль проверяет работу корзины (Cart). Тестируются сценарии добавления товаров и услуг, а также корректность расчёта итоговой суммы.

- Проверяется, что после добавления нескольких товаров сумма корзины соответствует ожидаемой.
- Проверяется, что услуги корректно добавляются в корзину и их стоимость учитывается в общем итоге.

2. test_payment.py

Модуль тестирует различные способы оплаты, реализованные через паттерн «Стратегия» и «Мост».

- Для каждого способа оплаты (наличные, карта, бонусы) проверяется успешное списание средств при достаточном балансе и невозможность оплаты при недостатке средств.
- Тесты гарантируют, что после оплаты баланс пользователя уменьшается на нужную сумму, а при попытке переплаты операция отклоняется.

3. test_transaction.py

Этот модуль проверяет проведение транзакций с использованием корзины и различных способов оплаты.

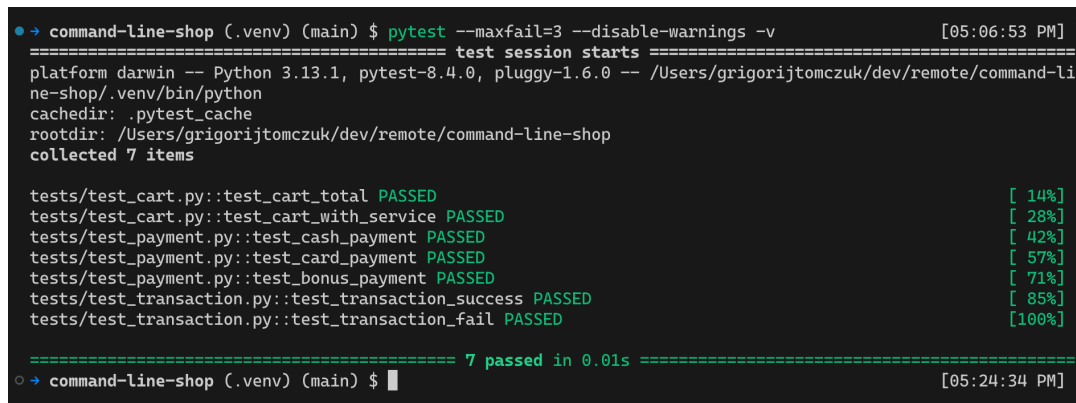
- Тестируется успешная оплата корзины с использованием нескольких способов оплаты одновременно.
- Проверяется, что при недостатке средств транзакция не проходит, а балансы пользователя остаются неизменными.

5.3 Принципы тестирования

- Каждый тест изолирован и не зависит от состояния других тестов.
- Для тестов используются искусственно созданные объекты (товары, услуги, пользователи), что позволяет контролировать все параметры и предусмотреть граничные случаи.
- Тесты покрывают как положительные сценарии (успешные операции), так и негативные (ошибки, недостаток средств, некорректные данные).

5.4 Запуск тестов

Для запуска всех тестов достаточно выполнить команду `pytest`. Результаты выполнения тестов отображаются в консоли, что позволяет быстро выявить и устранить возможные ошибки в логике приложения.



```
➔ command-line-shop (.venv) (main) $ pytest --maxfail=3 --disable-warnings -v [05:06:53 PM]
===== test session starts =====
platform darwin -- Python 3.13.1, pytest-8.4.0, pluggy-1.6.0 -- /Users/grigorijtomczuk/dev/remote/command-line-shop/.venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/grigorijtomczuk/dev/remote/command-line-shop
collected 7 items

tests/test_cart.py::test_cart_total PASSED [ 14%]
tests/test_cart.py::test_cart_with_service PASSED [ 28%]
tests/test_payment.py::test_cash_payment PASSED [ 42%]
tests/test_payment.py::test_card_payment PASSED [ 57%]
tests/test_payment.py::test_bonus_payment PASSED [ 71%]
tests/test_transaction.py::test_transaction_success PASSED [ 85%]
tests/test_transaction.py::test_transaction_fail PASSED [100%]

===== 7 passed in 0.01s =====
➔ command-line-shop (.venv) (main) $ [05:24:34 PM]
```

Рисунок 4 — Вывод автоматизированного процесса тестирования

5.5 Роль модульного тестирования

Модульные тесты позволяют:

- Быстро обнаруживать ошибки при изменении кода.
- Гарантировать корректную работу основных бизнес-процессов

(добавление в корзину, оплата, транзакции).

- Упрощать рефакторинг и внедрение новых функций без риска сломать существующую логику.

Таким образом, модульное тестирование является неотъемлемой частью процесса разработки и поддержки качества данного проекта.

6 Результаты работы программы с примерами разных сценариев

На рис. 5–17 изображены возможные пути пользователя приложения — пользователь входит под своим именем и просматривает доступные страницы (каталоги, доступные средства, корзина).

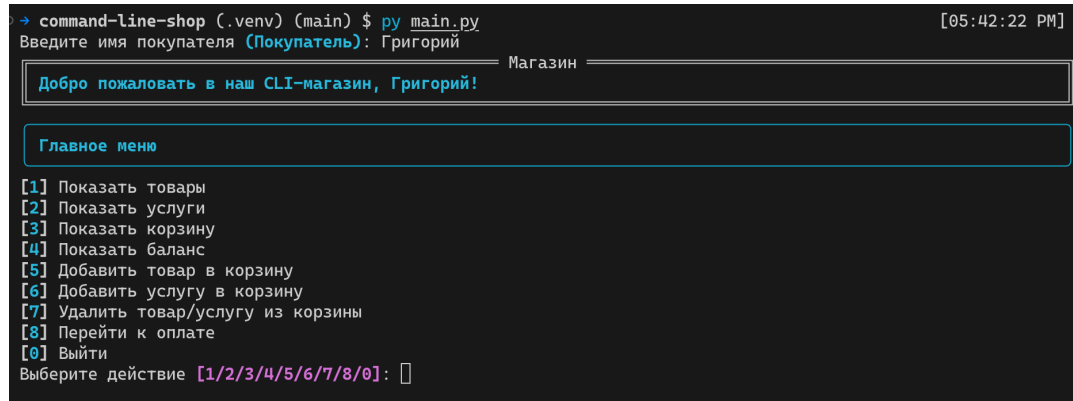


Рисунок 5 — Вход в приложение

Выберите действие [1/2/3/4/5/6/7/8/9/0]: 1

Доступные товары

#	Название	Описание	Тип	Цена	Остаток
1	Помидоры	Помидоры сливовидные, кг	Взвешиваемый	90 руб./кг	58 кг
2	Сыр	Сыр твердый Российский, 200г	Штучный	130 руб./шт	63 шт
3	Хлеб	Хлеб пшеничный нарезной, 400г	Штучный	140 руб./шт	59 шт
4	Морковь	Морковь столовая, кг	Взвешиваемый	70 руб./кг	81 кг
5	Молоко	Молоко пастеризованное 2.5%, 1л	Штучный	120 руб./шт	68 шт
6	Батон	Батон нарезной, 350г	Штучный	150 руб./шт	56 шт
7	Сахар	Сахар-песок, 1кг	Штучный	180 руб./шт	44 шт
8	Яблоки	Свежие зеленые яблоки, кг	Взвешиваемый	40 руб./кг	100 кг
9	Масло сливочное	Масло сливочное 82.5%, 180г	Штучный	170 руб./шт	48 шт
10	Картофель	Картофель молодой, кг	Взвешиваемый	60 руб./кг	92 кг
11	Бананы	Спелые бананы из Эквадора, кг	Взвешиваемый	50 руб./кг	96 кг
12	Свинина	Свинина на кости, кг	Взвешиваемый	110 руб./кг	72 кг
13	Гречка	Крупа гречневая, 900г	Штучный	190 руб./шт	40 шт
14	Огурцы	Огурцы грунтовые, кг	Взвешиваемый	80 руб./кг	84 кг
15	Куриное филе	Филе куриное охлажденное, кг	Взвешиваемый	100 руб./кг	76 кг
16	Яйца	Яйца куриные С1, 10шт	Штучный	160 руб./шт	50 шт

Рисунок 6 — Просмотр каталога товаров

Выберите действие [1/2/3/4/5/6/7/8/9/0]: 2

Доступные услуги

#	Название	Описание	Цена
1	Детский праздник	Организация и проведение детских праздников	1300 руб.
2	Услуги электрика	Вызов электрика на дом, мелкий и крупный ремонт	1700 руб.
3	Чистка ковров	Выездная чистка ковров на дому	900 руб.
4	Перевозка грузов	Грузоперевозки по городу и области	1500 руб.
5	Стрижка	Мужская или женская стрижка в салоне	400 руб.
6	Доставка на дом	Доставка продуктов до двери	300 руб.
7	Сборка мебели	Сборка корпусной мебели любой сложности	1100 руб.
8	Мойка машины	Комплексная мойка кузова и салона	600 руб.
9	Фотоуслуги	Профессиональная фотосъемка и обработка фото	1400 руб.
10	Маникюр	Профессиональный маникюр с покрытием	500 руб.
11	Услуги сантехника	Вызов сантехника, устранение протечек, замена оборудования	1800 руб.
12	Репетиторство	Индивидуальные занятия по математике, русскому языку и др.	1200 руб.
13	Ремонт обуви	Мелкий ремонт обуви: замена набойки, прошивка	800 руб.
14	Установка техники	Установка бытовой техники: стиральные машины, плиты и др.	1000 руб.
15	Ремонт компьютеров	Диагностика и ремонт ПК и ноутбуков	1600 руб.
16	Химчистка	Глубокая химчистка мебели или авто	700 руб.

Рисунок 7 — Просмотр каталога услуг

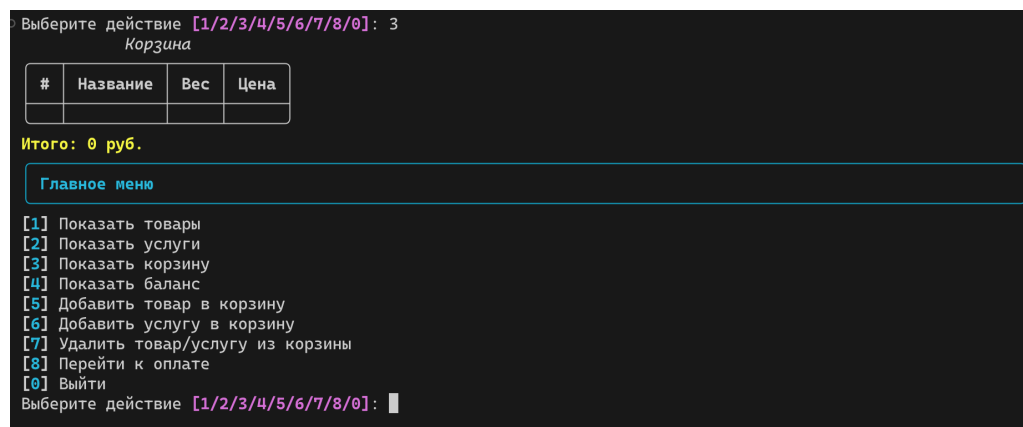


Рисунок 8 — Просмотр корзины

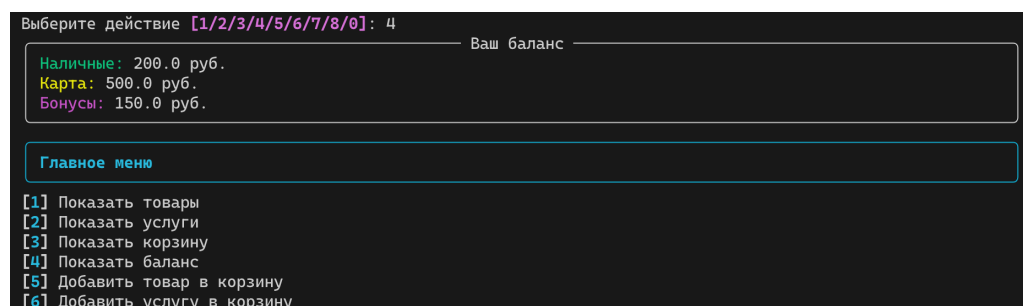


Рисунок 9 — Просмотр доступных средств

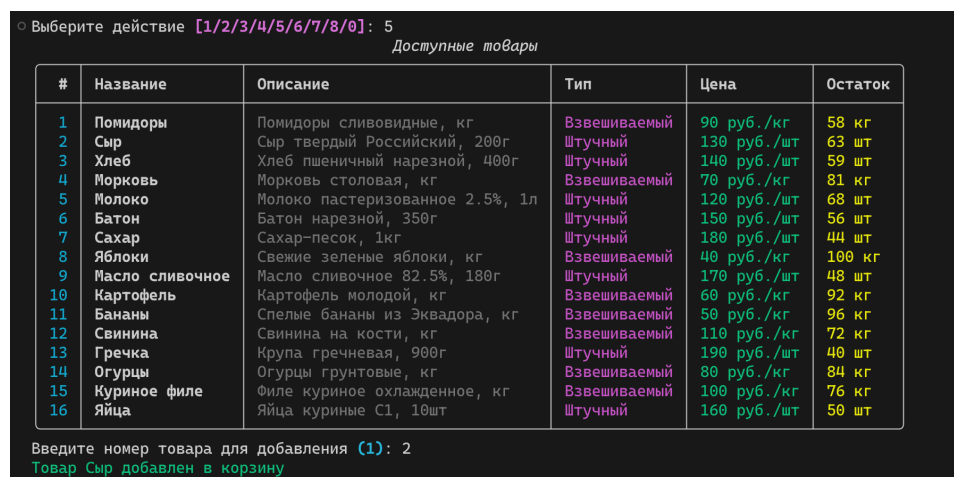


Рисунок 10 — Добавление товара в корзину

Выберите действие [1/2/3/4/5/6/7/8/0]: 6

Доступные услуги

#	Название	Описание	Цена
1	Детский праздник	Организация и проведение детских праздников	1300 руб.
2	Услуги электрика	Вызов электрика на дом, мелкий и крупный ремонт	1700 руб.
3	Чистка ковров	Выездная чистка ковров на дому	900 руб.
4	Перевозка грузов	Грузоперевозки по городу и области	1500 руб.
5	Стрижка	Мужская или женская стрижка в салоне	400 руб.
6	Доставка на дом	Доставка продуктов до двери	300 руб.
7	Сборка мебели	Сборка корпусной мебели любой сложности	1100 руб.
8	Мойка машины	Комплексная мойка кузова и салона	600 руб.
9	Фотоуслуги	Профессиональная фотосъемка и обработка фото	1400 руб.
10	Маникюр	Профессиональный маникюр с покрытием	500 руб.
11	Услуги сантехника	Вызов сантехника, устранение протечек, замена оборудования	1800 руб.
12	Репетиторство	Индивидуальные занятия по математике, русскому языку и др.	1200 руб.
13	Ремонт обуви	Мелкий ремонт обуви: замена набойки, прошивка	800 руб.
14	Установка техники	Установка бытовой техники: стиральные машины, плиты и др.	1000 руб.
15	Ремонт компьютеров	Диагностика и ремонт ПК и ноутбуков	1600 руб.
16	Химчистка	Глубокая химчистка мебели или авто	700 руб.

Введите номер услуги для добавления (1): 14

Услуга добавлена в корзину

Рисунок 11 — Добавление услуги в корзину

Выберите действие [1/2/3/4/5/6/7/8/0]: 7

Корзина

#	Название	Вес	Цена
1	Сыр	—	130 руб.
2	Установка техники	—	1000 руб.

Итого: 1130 руб.

Введите номер позиции для удаления (1): 2

Позиция удалена из корзины

Рисунок 12 — Удаление услуги из корзины

Выберите действие [1/2/3/4/5/6/7/8/0]: 8

Сумма к оплате: 130 руб.

Осталось оплатить: 130 руб.

Способ оплаты (или 'Отмена' для возврата в меню) [Наличные/Карта/Бонусы/Отмена]: наличные

Сколько оплатить Наличные? (доступно: 200.0 руб., 0 — отмена) (130): 50

Осталось оплатить: 80.0 руб.

Способ оплаты (или 'Отмена' для возврата в меню) [Наличные/Карта/Бонусы/Отмена]: бонусы

Сколько оплатить Бонусы? (доступно: 150.0 руб., 0 — отмена) (80.0):

Успех

Покупка успешна!

Рисунок 13 — Успешная оплата товара

Выберите действие [1/2/3/4/5/6/7/8/0]: 5

Доступные товары

#	Название	Описание	Тип	Цена	Остаток
1	Помидоры	Помидоры сливовидные, кг	Взвешиваемый	90 руб./кг	58 кг
2	Сыр	Сыр твердый Российский, 200г	Штучный	130 руб./шт	63 шт
3	Хлеб	Хлеб пшеничный нарезной, 400г	Штучный	140 руб./шт	59 шт
4	Морковь	Морковь столовая, кг	Взвешиваемый	70 руб./кг	81 кг
5	Молоко	Молоко пастеризованное 2.5%, 1л	Штучный	120 руб./шт	68 шт
6	Батон	Батон нарезной, 350г	Штучный	150 руб./шт	56 шт
7	Сахар	Сахар-песок, 1кг	Штучный	180 руб./шт	44 шт
8	Яблоки	Свежие зеленые яблоки, кг	Взвешиваемый	40 руб./кг	100 кг
9	Масло сливочное	Масло сливочное 82.5%, 180г	Штучный	170 руб./шт	48 шт
10	Картофель	Картофель молодой, кг	Взвешиваемый	60 руб./кг	92 кг
11	Бананы	Спелые бананы из Эквадора, кг	Взвешиваемый	50 руб./кг	96 кг
12	Свинина	Свинина на кости, кг	Взвешиваемый	110 руб./кг	72 кг
13	Гречка	Крупа гречневая, 900г	Штучный	190 руб./шт	40 шт
14	Огурцы	Огурцы грунтовые, кг	Взвешиваемый	80 руб./кг	84 кг
15	Куриное филе	Филе куриное охлажденное, кг	Взвешиваемый	100 руб./кг	76 кг
16	Яйца	Яйца куриные С1, 10шт	Штучный	160 руб./шт	50 шт

Введите номер товара для добавления (1): 100000

Некорректный номер товара

Рисунок 14 — Ошибочный ввод

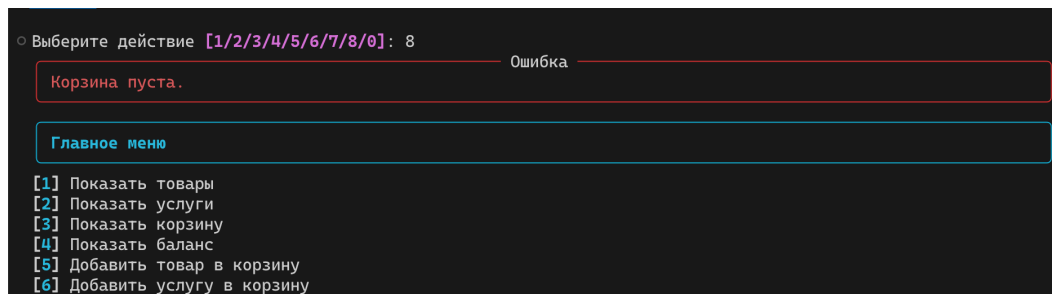


Рисунок 15 — Ошибочный ввод (оплата пустой корзины)

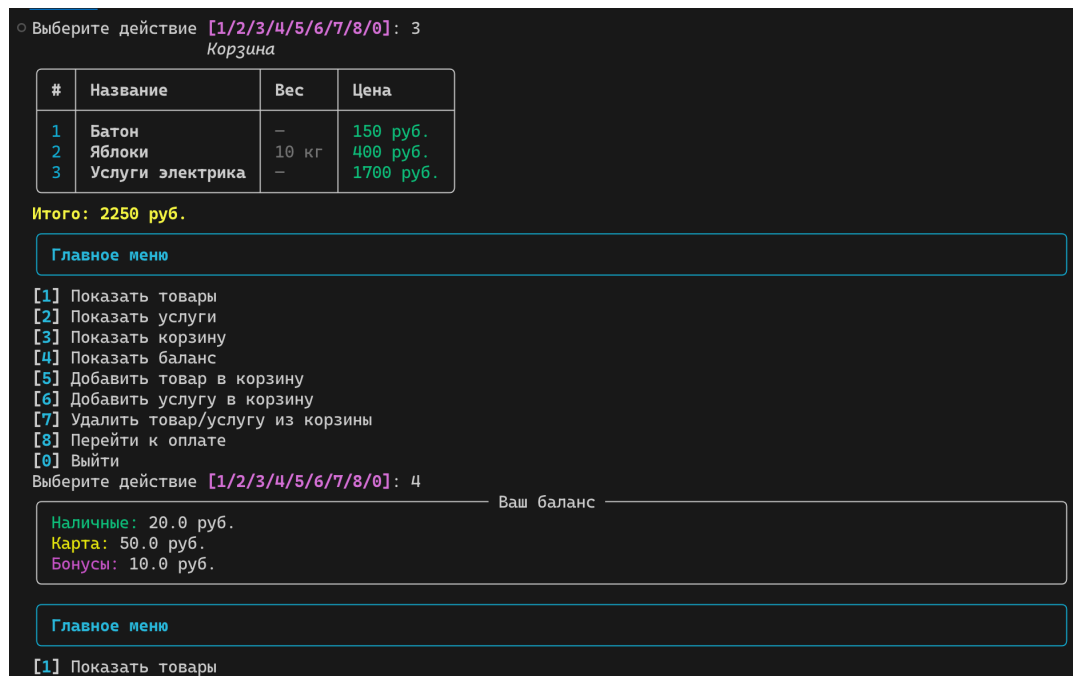


Рисунок 16 — Пользователь взял много товаров, которые не сможет оплатить

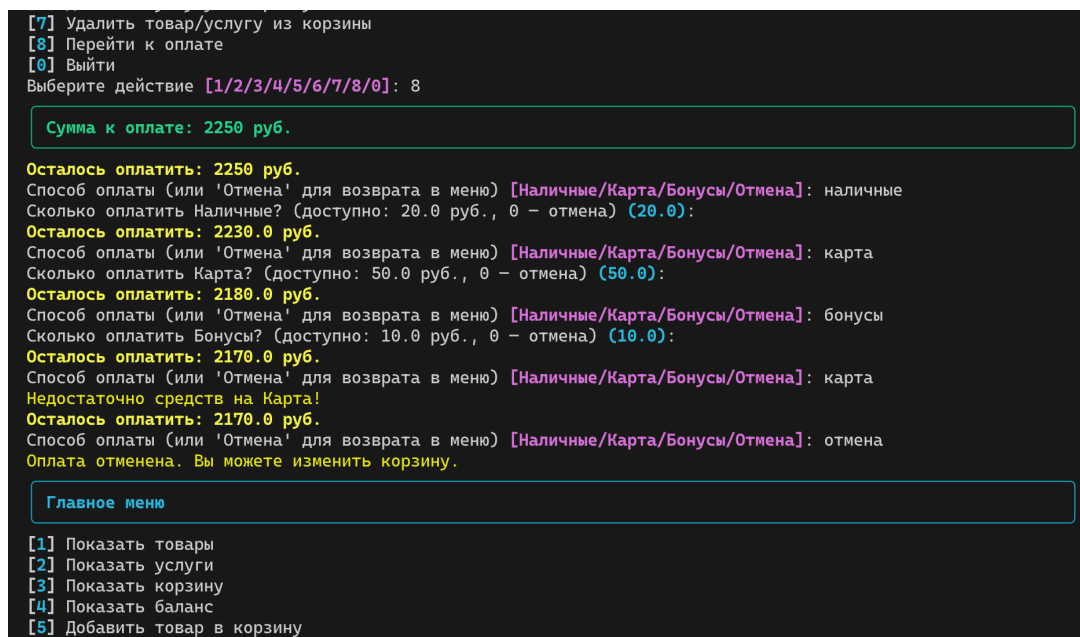


Рисунок 17 — Попытка оплаты и последующая отмена

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана полнофункциональная программная система, моделирующая процесс покупки товаров и услуг в магазине с использованием командно-текстового интерфейса. Проект реализован на языке Python с применением современных технологий, включая библиотеку Rich для визуализации интерфейса и pytest для модульного тестирования.

Особое внимание в процессе разработки было уделено соблюдению принципов объектно-ориентированного программирования (инкапсуляция, наследование, полиморфизм, абстракция), а также принципов SOLID, что позволило достичь модульности, читаемости и лёгкости сопровождения кода. В архитектуре проекта использован шаблон MVC, обеспечивший чёткое разделение логики, представления и управляющих компонентов. Такое разделение упростило внедрение новых функций, отладку и тестирование.

В проекте были реализованы ключевые паттерны проектирования:

- «Мост» (Bridge) — для отделения абстракции платежа от конкретных реализаций способов оплаты,
- «Стратегия» (Strategy) — для гибкой подмены логики оплаты,
- «Абстрактная фабрика» (Abstract Factory) — для централизованного создания платёжных объектов.

Также обеспечена поддержка частичной оплаты с возможностью комбинирования способов (наличные, карта, бонусы), а логика приложения не допускает ситуаций, при которых пользователь остаётся без покупки — реализована проверка наличия хотя бы одного доступного товара в корзине.

В системе реализована поддержка многопользовательских сессий с автоматическим сохранением состояния корзины, баланса и остатков товаров. Проведённое модульное тестирование подтвердило корректность реализации основных сценариев, включая добавление и удаление товаров, обработку ошибок и проведение транзакций.

Таким образом, поставленные в курсовой работе задачи были успешно

выполнены. Итоговый программный продукт обладает гибкой архитектурой, расширяемой логикой, удобным интерфейсом и высоким уровнем надёжности, что свидетельствует о практическом закреплении теоретических знаний и навыков, полученных в процессе изучения дисциплины «Технологии программирования».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2020. — 368 с.
2. Разработка программного обеспечения с использованием архитектуры MVC [Электронный ресурс]. — URL: <https://habr.com/ru/articles/256109/> (дата обращения: 05.06.2025).
3. Объектно-ориентированное программирование (ООП) в Python [Электронный ресурс]. — URL: <https://realpython.com/python3-object-oriented-programming/> (дата обращения: 05.06.2025).
4. Шаблоны проектирования на Python с примерами [Электронный ресурс]. — URL: <https://refactoring.guru/ru/design-patterns/python> (дата обращения: 05.06.2025).
5. Документация библиотеки Rich для Python [Электронный ресурс]. — URL: <https://rich.readthedocs.io/en/stable/> (дата обращения: 05.06.2025).
6. Марков С. А. Программирование на языке Python. Базовый курс. — М.: БХВ-Петербург, 2021. — 432 с.
7. Документация по pytest [Электронный ресурс]. — URL: <https://docs.pytest.org/en/stable/> (дата обращения: 09.06.2025).
8. Принципы SOLID в объектно-ориентированном программировании [Электронный ресурс]. — URL: <https://habr.com/ru/articles/319300/> (дата обращения: 08.06.2025).
9. Python Software Foundation. Python 3 Documentation [Электронный ресурс]. — URL: <https://docs.python.org/3/> (дата обращения: 05.06.2025).
10. Грокоп Д. Чистый Python. Тонкости программирования для профи. — СПб.: Питер, 2020. — 352 с.

ПРИЛОЖЕНИЕ А

shop.py

```
import json
import os
from copy import deepcopy
from uuid import uuid4

from models.cart import Cart
from models.customer import Customer
from models.exceptions import InvalidInputError,
NotEnoughFundsError, ShopError
from models.product import Product
from models.transaction import Transaction
from services.factory import ConcretePaymentFactory
from services.utils import load_products, load_services

class ShopModel:
    __session_file_path = "data/sessions.json"
    __products_file_path = "data/products.json"
    __services_file_path = "data/services.json"

    def __init__(self, customer_name=None):
        self.products = load_products("data/products.json")
        self.services = load_services("data/services.json")
        try:
            self.load_session(self.__session_file_path,
customer_name)
        except FileNotFoundError:
            self.customer = Customer(
                str(uuid4()), customer_name, cash=200.0,
card=500.0, bonuses=150.0
            )
            self.cart = Cart()
            self.transaction = Transaction(self.cart, self.customer)

    def save_session(self):
        os.makedirs("data", exist_ok=True)
        cart_items = []
        for item in self.cart.items:
            d = {"name": item.name}
```



```

        if hasattr(item, "weight") and item.weight is not None:
            d["weight"] = item.weight
        cart_items.append(d)
    session = {
        "customer": {
            "customer_id": self.customer.customer_id,
            "name": self.customer.name,
            "cash": self.customer.cash,
            "card": self.customer.card,
            "bonuses": self.customer.bonuses,
        },
        "cart": cart_items,
    }
    # Сохраняем сессию в файл
    try:
        with open(self.__session_file_path, "r", encoding="utf-8") as f:
            all_sessions = json.load(f)
            if not isinstance(all_sessions, list):
                all_sessions = []
    except Exception:
        all_sessions = []
    # ищем по customer_id, если есть — обновляем, иначе
    добавляем
    updated = False
    for i, s in enumerate(all_sessions):
        if s.get("customer", {}).get("customer_id") == self.customer.customer_id:
            all_sessions[i] = session
            updated = True
            break
    if not updated:
        all_sessions.append(session)
    with open(self.__session_file_path, "w", encoding="utf-8") as f:
        json.dump(all_sessions, f, ensure_ascii=False,
            indent=2)

    # сохраняем актуальные остатки товаров и услуг в
    data/products.json и data/services.json
    # продукты
    products_data = [

```

```

        {
            "item_id": getattr(p, "item_id", None),
            "name": p.name,
            "description": getattr(p, "description", ""),
            "price_per_unit": p.price_per_unit,
            "requires_weighing": getattr(p,
"requires_weighing", False),
            "quantity": p.quantity,
        }
        for p in self.products
    ]
    with open(self.__products_file_path, "w", encoding="utf-8")
as f:
        json.dump(products_data, f, ensure_ascii=False,
indent=2)

    # услуги
    services_data = [
        {
            "item_id": getattr(s, "item_id", None),
            "name": s.name,
            "description": getattr(s, "description", ""),
            "cost": getattr(s, "cost", s.price_per_unit),
        }
        for s in self.services
    ]
    with open(self.__services_file_path, "w", encoding="utf-8")
as f:
        json.dump(services_data, f, ensure_ascii=False,
indent=2)

    def load_session(self, path, customer_name):
        try:
            with open(path, "r", encoding="utf-8") as f:
                all_sessions = json.load(f)
                if not isinstance(all_sessions, list):
                    all_sessions = []
        except Exception:
            all_sessions = []
        # ищем сессию по имени
        session = None
        for s in all_sessions:

```

```

        if s.get("customer", {}).get("name") == customer_name:
            session = s
            break
    if session is None:
        raise FileNotFoundError("Нет подходящей сессии для загрузки")

    self.customer = Customer(
        session["customer"]["customer_id"],
        session["customer"]["name"],
        session["customer"]["cash"],
        session["customer"]["card"],
        session["customer"]["bonuses"],
    )
    self.cart = Cart()
    for item in session["cart"]:
        # Сначала ищем среди товаров
        prod = next((p for p in self.products if p.name == item["name"]), None)
        if prod:
            product_to_add = Product(
                item_id=prod.item_id,
                name=prod.name,
                description=prod.description,
                price_per_unit=prod.price_per_unit,
                requires_weighing=prod.requires_weighing,
                quantity=prod.quantity,
            )
            if product_to_add.requires_weighing:
                product_to_add.weigh(item.get("weight", 0))
            self.cart.add_item(product_to_add)
            continue
        # Если не найдено среди товаров, ищем среди услуг
        serv = next((s for s in self.services if s.name == item["name"]), None)
        if serv:
            from models.service import Service

            service_to_add = Service(
                item_id=serv.item_id,
                name=serv.name,
                description=serv.description,
                cost=serv.price_per_unit,

```

```

        )
        self.cart.add_item(service_to_add)

def checkout(self, payments):
    payment_factory = ConcretePaymentFactory()

    if not self.cart.items:
        raise InvalidInputError("Корзина пуста!")
    total = self.cart.calculate_total()
    left = total
    virtual_balance = {
        "cash": self.customer.cash,
        "card": self.customer.card,
        "bonuses": self.customer.bonuses,
    }
    withdrawn = {"cash": 0, "card": 0, "bonuses": 0}
    for payment in payments:
        method = payment["method"]
        amount = payment["amount"]
        if method not in virtual_balance:
            raise InvalidInputError(f"Неизвестный способ
оплаты: {method}")
        available = virtual_balance[method] - withdrawn[method]
        if amount <= 0 or amount > available or amount > left:
            raise NotEnoughFundsError(
                f"Недостаточно средств на {method} или
некорректная сумма!"
            )
        withdrawn[method] += amount
        left -= amount

    # бизнес-логика транзакции
    payment_objs = [
        {
            "method":
payment_factory.create_payment(p["method"]),
            "amount": p["amount"],
        }
        for p in payments
    ]
    success = self.transaction.attempt_payment(payment_objs)
    if success:

```

```

        self.cart.items.clear()
        return True
    else:
        for payment in payments:
            method = payment["method"]
            amount = payment["amount"]
            setattr(self.customer, method,
getattr(self.customer, method) + amount)
            raise NotEnoughFundsError("Ошибка оплаты! Недостаточно
средств.")

    def add_service_to_cart(self, idx):
        if idx < 0 or idx >= len(self.services):
            raise InvalidInputError("Некорректный номер услуги")
        service = self.services[idx]
        self.cart.add_item(service)

    def remove_from_cart(self, idx):
        if not self.cart.items:
            raise InvalidInputError("Корзина пуста.")
        if idx < 0 or idx >= len(self.cart.items):
            raise InvalidInputError("Некорректный номер позиции")
        item = self.cart.items[idx]
        for p in self.products:
            if p.name == item.name:
                if getattr(p, "requires_weighing", False):
                    p.quantity += getattr(item, "weight", 0) or 0
                else:
                    p.quantity += 1
        self.cart.remove_item(idx)

    def add_to_cart(self, idx):
        if idx < 0 or idx >= len(self.products):
            raise InvalidInputError("Некорректный номер товара")
        product = self.products[idx]
        if getattr(product, "requires_weighing", False):
            raise InvalidInputError(
                "Для взвешиваемых товаров используйте
add_weighted_to_cart"
            )
        if product.quantity < 1:

```

```

        raise InvalidInputError(f"Товар {product.name} закончился!")
        self.cart.add_item(product)
        product.quantity -= 1

    def add_weighted_to_cart(self, idx, weight):
        if idx < 0 or idx >= len(self.products):
            raise InvalidInputError("Некорректный номер товара")
        product = self.products[idx]
        if not getattr(product, "requires_weighing", False):
            raise InvalidInputError("Этот товар не требует взвешивания")
        if weight <= 0:
            raise InvalidInputError("Вес должен быть положительным числом")
        if weight > product.quantity:
            raise InvalidInputError(
                f"Недостаточно товара на складе! Осталось: {product.quantity} кг"
            )
        from copy import deepcopy

        product_copy = deepcopy(product)
        product_copy.weigh(weight)
        self.cart.add_item(product_copy)
        product.quantity -= weight

```

transaction.py

```
from models.cart import Cart
from models.customer import Customer

class Transaction:
    def __init__(self, cart: Cart, customer: Customer):
        self.cart = cart
        self.customer = customer

    def attempt_payment(self, payments):
        """
        payments: список словарей вида {"method": PaymentMethod,
        "amount": число}
        Поддерживает паттерн Bridge: можно передавать как
        PaymentMethod, так и Payment.
        """
        from models.payment import Payment

        total = self.cart.calculate_total()
        paid = 0
        for method in payments:
            part = min(method["amount"], total - paid)
            pay_obj: Payment = method["method"]
            success = pay_obj.pay(part, self.customer)
            if success:
                paid += part
            if paid >= total:
                return True
        return paid >= total
```

factory.py

```
from abc import ABC, abstractmethod

from models.payment import BonusPayment, CardPayment, CashPayment,
Payment

# Абстрактный класс фабрики
class PaymentFactory(ABC):
    @abstractmethod
    def create_payment(self, method):
        pass

# Конкретная фабрика для создания способов оплаты
class ConcretePaymentFactory(PaymentFactory):
    def create_payment(self, method):
        if method == "cash":
            return Payment(CashPayment())
        elif method == "card":
            return Payment(CardPayment())
        elif method == "bonuses":
            return Payment(BonusPayment())
        else:
            raise ValueError(f"Неизвестный способ оплаты:
{method}")
```