МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе No3

Специальность ПО11(о)

Выполнил
К. А. Головач,
студент группы ПО11

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«5» апрель 2025 г.

Брест 2025

Вариант 6

Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Python.

Общее задание

• Прочитать задания, взятые из каждой группы, соответствующей одному из трех основных типов паттернов;

• Определить паттерн проектирования, который может использоваться при реализации задания. Пояснить свой выбор;

• Реализовать фрагмент программной системы, используя выбранный паттерн. Реализовать все необходимые дополнительные классы.

Задание 1.

6) Музыкальный магазин. Должно обеспечиваться одновременное обслуживание нескольких покупателей. Магазин должен предоставлять широкий выбор товаров различных музыкальных направлений.

Мы используем фабричный метод, так как нужно создавать разные модели через единый интерфейс.

Выполнение:

Код программы:

task_1.py:

```
import threading
from abc import ABC, abstractmethod

# Базовый класс для всех музыкальных товаров
class MusicProduct(ABC):
    def __init__(self, name: str, price: float):
        self.name = name  # Название товара
        self.price = price  # Цена товара

    @abstractmethod
    def play_sample(self):
        """Воспроизведение демо-версии товара."""
        pass

    def __str__(self):
        return f"{self.name} (${self.price})"  # Строковое представление товара

# Конкретные классы для товаров разных музыкальных направлений
class RockMusic(MusicProduct):
    def play_sample(self):
        print(f"Воспроизводится рок-демо для {self.name}")

class PopMusic(MusicProduct):
    def play_sample(self):
```

```
        print(f"Воспроизводится поп-демо для {self.name}")

class ClassicalMusic(MusicProduct):
    def play_sample(self):
        print(f"Воспроизводится классическое демо для {self.name}")

# Абстрактная фабрика для создания музыкальных товаров
class MusicFactory(ABC):
    @abstractmethod
    def create_product(self, name: str, price: float) -> MusicProduct:
        pass

# Конкретные фабрики для каждого направления
class RockMusicFactory(MusicFactory):
    def create_product(self, name: str, price: float) -> MusicProduct:
        return RockMusic(name, price)

class PopMusicFactory(MusicFactory):
    def create_product(self, name: str, price: float) -> MusicProduct:
        return PopMusic(name, price)

class ClassicalMusicFactory(MusicFactory):
    def create_product(self, name: str, price: float) -> MusicProduct:
        return ClassicalMusic(name, price)

# Класс магазина
class MusicStore:
    def __init__(self):
        self.factories = {
            "rock": RockMusicFactory(),  # Фабрика для рока
            "pop": PopMusicFactory(),  # Фабрика для поп-музыки
            "classical": ClassicalMusicFactory(),  # Фабрика для классической музыки
        }

    def order_product(self, genre: str, name: str, price: float) -> MusicProduct:
        if genre not in self.factories:
            raise ValueError(f"Неизвестный музыкальный жанр: {genre}")
        factory = self.factories[genre]
        product = factory.create_product(name, price)
        print(f"Заказ товара: {product}")
        return product

# Функция для имитации поведения покупателя
def customer_behavior(store, customer_id, orders):
    print(f"Покупатель {customer_id} начал делать покупки...")
    for genre, name, price in orders:
        try:
            product = store.order_product(genre, name, price)
            product.play_sample()  # Воспроизведение демо-версии товара
        except ValueError as e:
            print(f"Покупатель {customer_id}: Ошибка - {e}")
    print(f"Покупатель {customer_id} завершил покупки.")

# Пример использования с многопоточностью
if __name__ == "__main__":
    # Создаем магазин
    store = MusicStore()

    # Определяем заказы для нескольких покупателей
```

```
customer_orders = [
    ("rock", "Лучшие хиты", 19.99),  # Заказ рок-музыки
    ("pop", "Летний хит", 9.99),  # Заказ поп-музыки
    ("classical", "Симфония №5", 29.99),  # Заказ классической музыки
]

# Создаем несколько потоков для имитации одновременного обслуживания
threads = []
for i in range(3):  # 3 покупателя
    thread = threading.Thread(target=customer_behavior, args=(store, i + 1, customer_orders))
    threads.append(thread)
    thread.start()  # Запуск потока

# Ждем завершения всех потоков
for thread in threads:
    thread.join()

print("Все покупатели обслужены.")
```

Рисунок с результатами работы программы task_1.py:

```
C:\Users\kirja\AppData\Local\Programs\Python\Python313\pythor
Покупатель 1 начал делать покупки...
Заказ товара: Лучшие хиты ($19.99)
Воспроизводится рок-демо для Лучшие хиты
Заказ товара: Летний хит ($9.99)
Воспроизводится поп-демо для Летний хит
Заказ товара: Симфония №5 ($29.99)
Воспроизводится классическое демо для Симфония №5
Покупатель 1 завершил покупки.
Покупатель 2 начал делать покупки...
Заказ товара: Лучшие хиты ($19.99)
Воспроизводится рок-демо для Лучшие хиты
Заказ товара: Летний хит ($9.99)
Воспроизводится поп-демо для Летний хит
Заказ товара: Симфония №5 ($29.99)
Воспроизводится классическое демо для Симфония №5
Покупатель 2 завершил покупки.
Покупатель 3 начал делать покупки...
Заказ товара: Лучшие хиты ($19.99)
Воспроизводится рок-демо для Лучшие хиты
Заказ товара: Летний хит ($9.99)
Воспроизводится поп-демо для Летний хит
Заказ товара: Симфония №5 ($29.99)
Воспроизводится классическое демо для Симфония №5
Покупатель 3 завершил покупки.
Все покупатели обслужены.
```

Задание 2.
6) Учетная запись покупателя книжного интернет-магазина. Предусмотреть различные уровни учетки в зависимости от активности покупателя. Дополнительные уровни добавляют функциональные возможности и открывают доступ к уникальным предложениям.

В данном случае подойдёт **структурный паттерн "Цепочка обязанностей" (Chain of Responsibility)**
Почему используется "Цепочка обязанностей"?
Гибкость : Легко добавлять новые уровни учетной записи, не изменяя существующий код. Например, если понадобится добавить новый уровень между AdvancedLevel и ExpertLevel, достаточно создать новый класс и вставить его в цепочку.
Разделение ответственности : Каждый уровень отвечает только за свою часть логики. Это упрощает поддержку и тестирование кода.
Динамическое определение обработчиков : Порядок обработчиков можно менять во время выполнения программы, что делает систему более гибкой.
Устранение жесткой зависимости : Клиентский код (класс UserAccount) не знает, какой именно уровень обработает запрос. Он просто передает запрос в начало цепочки, и система сама решает, кто будет обрабатывать запрос.

Выполнение:
Код программы:

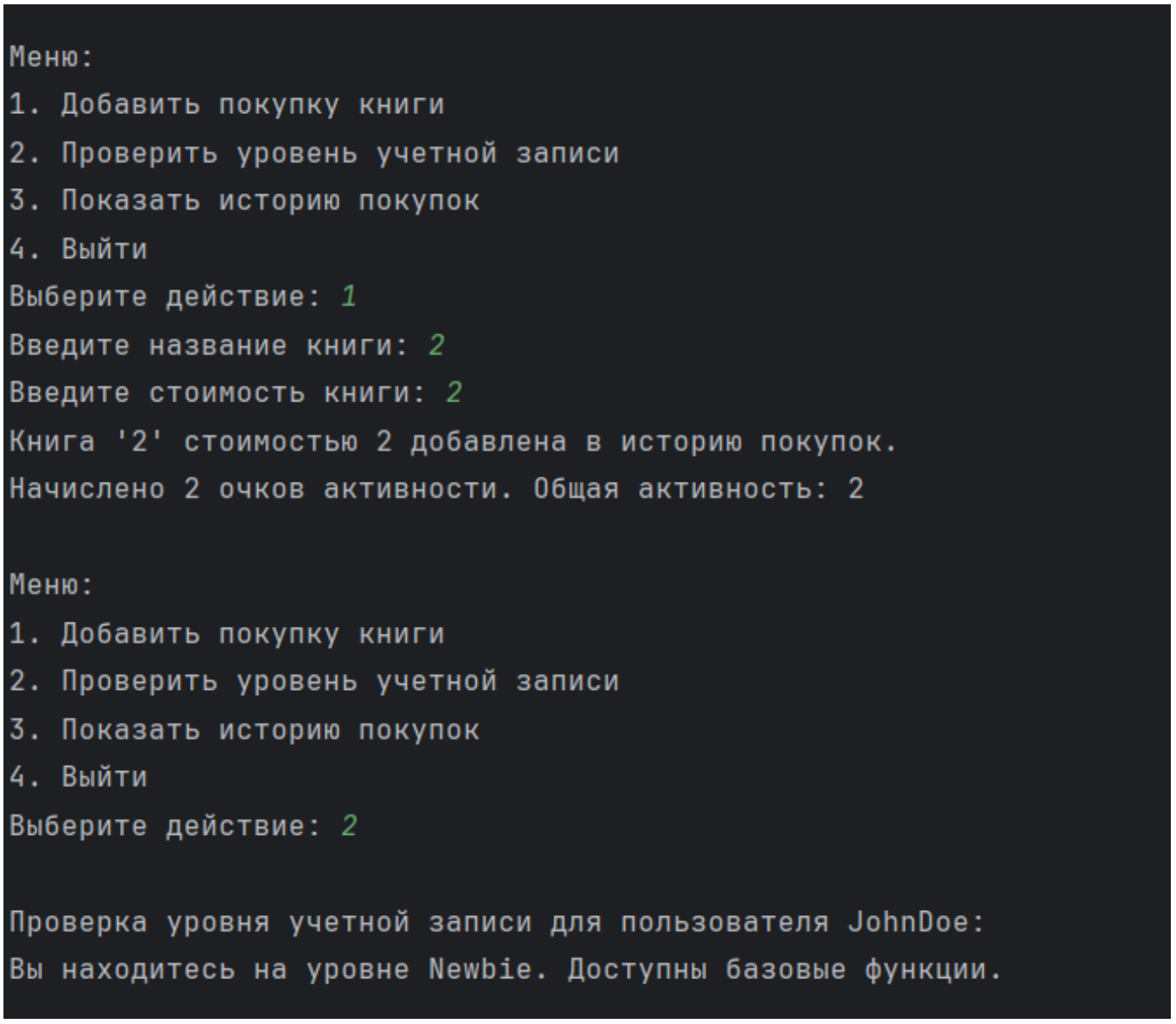
task_2.py:
```
from abc import ABC, abstractmethod

# Базовый класс для всех уровней учетной записи
class AccountLevel(ABC):
    def __init__(self):
        self.next_level = None

    def set_next(self, next_level):
        self.next_level = next_level
        return next_level

    @abstractmethod
    def handle_request(self, user_activity):
        pass

# Уровень 1: Начальный уровень (Newbie)
class NewbieLevel(AccountLevel):
    def handle_request(self, user_activity):
        if user_activity < 10:
            print("Вы находитесь на уровне Newbie. Доступны базовые функции.")
        elif self.next_level:
            self.next_level.handle_request(user_activity)

# Уровень 2: Продвинутый уровень (Advanced)
class AdvancedLevel(AccountLevel):
    def handle_request(self, user_activity):
        if 10 <= user_activity < 50:
```

```
            print("Вы находитесь на уровне Advanced. Доступны скидки до 10%.")
        elif self.next_level:
            self.next_level.handle_request(user_activity)

# Уровень 3: Экспертный уровень (Expert)
class ExpertLevel(AccountLevel):
    def handle_request(self, user_activity):
        if 50 <= user_activity < 100:
            print("Вы находитесь на уровне Expert. Доступны скидки до 25% и эксклюзивные предложения.")
        elif self.next_level:
            self.next_level.handle_request(user_activity)

# Уровень 4: VIP уровень (VIP)
class VIPLevel(AccountLevel):
    def handle_request(self, user_activity):
        if user_activity >= 100:
            print("Вы находитесь на уровне VIP. Доступны скидки до 50%, персональные предложения и
приоритетная поддержка.")
        else:
            print("Ошибка: Уровень учетной записи не определен.")

# Класс для управления учетной записью пользователя
class UserAccount:
    def __init__(self, name):
        self.name = name
        self.user_activity = 0  # Активность пользователя (например, количество заказов)
        self.purchase_history = []  # История покупок
        self.free_books = []  # Бесплатные книги, выбранные после получения VIP-статуса
        self.is_vip = False  # Флаг для отслеживания VIP-статуса

    def add_purchase(self, book_name, book_price):
        """Добавляет покупку книги и начисляет очки активности."""
        activity_points = int(book_price)  # Очки активности равны стоимости книги
        self.user_activity += activity_points
        self.purchase_history.append((book_name, book_price))
        print(f"Книга '{book_name}' стоимостью {book_price} добавлена в историю покупок.")
        print(f"Начислено {activity_points} очков активности. Общая активность: {self.user_activity}")

    def check_account_level(self):
        # Создаем цепочку уровней
        newbie_level = NewbieLevel()
        advanced_level = AdvancedLevel()
        expert_level = ExpertLevel()
        vip_level = VIPLevel()

        # Устанавливаем порядок обработки
        newbie_level.set_next(advanced_level).set_next(expert_level).set_next(vip_level)

        # Проверяем уровень учетной записи
        print(f"\nПроверка уровня учетной записи для пользователя {self.name}:")
        newbie_level.handle_request(self.user_activity)

        # Если достигнут VIP-уровень, предлагаем выбрать бесплатные книги
        if self.user_activity >= 100 and not self.is_vip:
            self.is_vip = True
            self.choose_free_books()

    def choose_free_books(self):
        """Предлагает пользователю выбрать бесплатные книги после получения VIP-статуса."""
```

```
        print("\nПоздравляем! Вы получили VIP-статус!")
        print("Выберите до 3 бесплатных книг из списка:")

        free_book_options = [
            "1. 'Мастер и Маргарита' - Михаил Булгаков",
            "2. '1984' - Джордж Оруэлл",
            "3. 'Убить пересмешника' - Харпер Ли",
            "4. 'Великий Гэтсби' - Фрэнсис Скотт Фицджеральд",
            "5. 'Гордость и предубеждение' - Джейн Остин"
        ]

        for option in free_book_options:
            print(option)

        while len(self.free_books) < 3:
            choice = input(f"Выберите книгу (введите номер от 1 до {len(free_book_options)}): ")
            if choice.isdigit() and 1 <= int(choice) <= len(free_book_options):
                selected_book = free_book_options[int(choice) - 1]
                if selected_book not in self.free_books:
                    self.free_books.append(selected_book)
                    print(f"Книга '{selected_book}' добавлена в вашу коллекцию.")
                else:
                    print("Эта книга уже выбрана. Пожалуйста, выберите другую.")
            else:
                print("Неверный выбор. Пожалуйста, введите корректный номер.")

            if len(self.free_books) < 3:
                continue_choice = input("Хотите выбрать еще одну книгу? (да/нет): ").lower()
                if continue_choice != "да":
                    break

        print("\nВаши бесплатные книги:")
        for book in self.free_books:
            print(f"- {book}")

    def show_purchase_history(self):
        """Выводит историю покупок пользователя."""
        if not self.purchase_history:
            print("История покупок пуста.")
            return

        print("\nИстория покупок:")
        for i, (book_name, book_price) in enumerate(self.purchase_history, start=1):
            print(f"{i}. Книга: {book_name}, Стоимость: {book_price}")

# Пример использования
if __name__ == "__main__":
    # Создаем учетную запись пользователя
    user = UserAccount("JohnDoe")

    while True:
        print("\nМеню:")
        print("1. Добавить покупку книги")
        print("2. Проверить уровень учетной записи")
        print("3. Показать историю покупок")
        print("4. Выйти")

        choice = input("Выберите действие: ")
```

```
    if choice == "1":
        book_name = input("Введите название книги: ")
        book_price = input("Введите стоимость книги: ")
        if book_price.isdigit():
            user.add_purchase(book_name, int(book_price))
        else:
            print("Ошибка: Стоимость книги должна быть числом.")
    elif choice == "2":
        user.check_account_level()
    elif choice == "3":
        user.show_purchase_history()
    elif choice == "4":
        print("Спасибо за использование системы. До свидания!")
        break
    else:
        print("Неверный выбор. Пожалуйста, выберите действие из меню.")
```

Рисунок с результатами работы программы task_2.py:



```
Меню:
1. Добавить покупку книги
2. Проверить уровень учетной записи
3. Показать историю покупок
4. Выйти
Выберите действие: 3

История покупок:
1. Книга: 2, Стоимость: 2

Меню:
1. Добавить покупку книги
2. Проверить уровень учетной записи
3. Показать историю покупок
4. Выйти
Выберите действие: 1
Введите название книги: 12345
Введите стоимость книги: 12345
Книга '12345' стоимостью 12345 добавлена в историю покупок.
Начислено 12345 очков активности. Общая активность: 12347

Меню:
1. Добавить покупку книги
2. Проверить уровень учетной записи
3. Показать историю покупок
4. Выйти
```

```
Меню:
1. Добавить покупку книги
2. Проверить уровень учетной записи
3. Показать историю покупок
4. Выйти
Выберите действие: 2

Проверка уровня учетной записи для пользователя JohnDoe:
Вы находитесь на уровне VIP. Доступны скидки до 50%, персональные предложения и приоритетная поддержка.

Поздравляем! Вы получили VIP-статус!
Выберите до 3 бесплатных книг из списка:
1. 'Мастер и Маргарита' - Михаил Булгаков
2. '1984' - Джордж Оруэлл
3. 'Убить пересмешника' - Харпер Ли
4. 'Великий Гэтсби' - Фрэнсис Скотт Фицджеральд
5. 'Гордость и предубеждение' - Джейн Остин
Выберите книгу (введите номер от 1 до 5): 1
Книга '1. 'Мастер и Маргарита' - Михаил Булгаков' добавлена в вашу коллекцию.
Хотите выбрать еще одну книгу? (да/нет):
```

Задание 3.

6) Проект «Принтер». Предусмотреть выполнение операций (печать, загрузка бумаги, извлечение зажатой бумаги, заправка картриджа), режимы – ожидание, печать документа, зажатие бумаги, отказ – при отсутствии бумаги или краски, атрибуты – модель, количество листов в лотке, % краски в картридже, вероятность зажатия.

Подойдёт **паттерн "Состояние" (State)**. Этот поведенческий паттерн позволяет объекту изменять своё поведение при изменении внутреннего состояния, что соответствует переключению между режимами банкомата.

Выполнение:

Код программы:

task_3.py:

```
from abc import ABC, abstractmethod
import time


# Интерфейс состояния
class PrinterState(ABC):
    @abstractmethod
    def print_document(self, printer):
        pass

    @abstractmethod
    def load_paper(self, printer):
        pass

    @abstractmethod
    def remove_jammed_paper(self, printer):
        pass

    @abstractmethod
```

```
    def refill_cartridge(self, printer):
        pass


# Конкретные состояния
class IdleState(PrinterState):
    def print_document(self, printer):
        if printer.paper_count > 0 and printer.ink_level > 0:
            print("Начало печати...")
            printer.set_state(PrintingState())
            printer.print_document()  # Переходим в состояние печати
        elif printer.paper_count == 0:
            print("Отказ: нет бумаги!")
            printer.set_state(OutOfPaperState())
        elif printer.ink_level == 0:
            print("Чистый лист выдан (нет чернил).")
            printer.paper_count -= 1
            printer.set_state(IdleState())

    def load_paper(self, printer):
        print(f"Загрузка бумаги. Было {printer.paper_count} листов.")
        printer.paper_count += 50  # Загружаем 50 листов
        print(f"Теперь {printer.paper_count} листов.")

    def remove_jammed_paper(self, printer):
        print("Зажатой бумаги нет.")

    def refill_cartridge(self, printer):
        print("Картридж заправлен.")
        printer.ink_level = 100


class PrintingState(PrinterState):
    def print_document(self, printer):
        print("Процесс печати...")
        time.sleep(1)  # Имитация времени печати
        printer.paper_count -= 1
        printer.ink_level -= 10  # На одну страницу уходит 10% чернил
        printer.print_count += 1  # Увеличиваем счетчик печатей
        print("Документ напечатан.")

        # Проверяем, нужно ли зажевать бумагу
        if printer.print_count % 2 == 0:
            print("Бумага зажевалась!")
            printer.set_state(PaperJamState())
        else:
            printer.set_state(IdleState())

    def load_paper(self, printer):
        print("Операция недоступна во время печати.")

    def remove_jammed_paper(self, printer):
        print("Операция недоступна во время печати.")

    def refill_cartridge(self, printer):
        print("Операция недоступна во время печати.")


class PaperJamState(PrinterState):
```

```
    def print_document(self, printer):
        print("Операция недоступна при зажатии бумаги.")

    def load_paper(self, printer):
        print("Операция недоступна при зажатии бумаги.")

    def remove_jammed_paper(self, printer):
        print("Извлечение зажатой бумаги...")
        printer.set_state(IdleState())

    def refill_cartridge(self, printer):
        print("Операция недоступна при зажатии бумаги.")


class OutOfPaperState(PrinterState):
    def print_document(self, printer):
        print("Отказ: нет бумаги!")

    def load_paper(self, printer):
        print(f"Загрузка бумаги. Было {printer.paper_count} листов.")
        printer.paper_count += 50  # Загружаем 50 листов
        print(f"Теперь {printer.paper_count} листов.")
        printer.set_state(IdleState())

    def remove_jammed_paper(self, printer):
        print("Зажатой бумаги нет.")

    def refill_cartridge(self, printer):
        print("Операция недоступна при отсутствии бумаги.")


# Контекст (Принтер)
class Printer:
    def __init__(self, model, paper_count=10, ink_level=100):
        self.model = model
        self.paper_count = paper_count
        self.ink_level = ink_level
        self.state = IdleState()
        self.print_count = 0  # Счетчик успешных печатей

    def set_state(self, state):
        self.state = state

    def print_document(self):
        self.state.print_document(self)

    def load_paper(self):
        self.state.load_paper(self)

    def remove_jammed_paper(self):
        self.state.remove_jammed_paper(self)

    def refill_cartridge(self):
        self.state.refill_cartridge(self)


# Функция для вывода меню и обработки выбора пользователя
def main():
    printer = Printer(model="HP LaserJet Pro", paper_count=10, ink_level=100)
```

```
    while True:
        print("\n=== Меню принтера ===")
        print("1. Печать документа")
        print("2. Загрузка бумаги")
        print("3. Извлечение зажатой бумаги")
        print("4. Заправка картриджа")
        print("5. Выход")
        print(f"Текущее состояние: {printer.state.__class__.__name__}")
        print(f"Бумага: {printer.paper_count}, Чернила: {printer.ink_level}%")

        choice = input("Выберите действие (1-5): ")

        if choice == "1":
            printer.print_document()
        elif choice == "2":
            printer.load_paper()
        elif choice == "3":
            printer.remove_jammed_paper()
        elif choice == "4":
            printer.refill_cartridge()
        elif choice == "5":
            print("Выход из программы.")
            break
        else:
            print("Неверный выбор. Попробуйте снова.")


if __name__ == "__main__":
    main()
```

Рисунки с результатами работы программы task_3.py:

```
e tool windows нтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: IdleState
Бумага: 10, Чернила: 100%
Выберите действие (1-5): 1
Начало печати...
Процесс печати...
Документ напечатан.

=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: IdleState
Бумага: 9, Чернила: 90%
Выберите действие (1-5): 1
Начало печати...
Процесс печати...
Документ напечатан.
Бумага зажевалась!
```

```
=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: PaperJamState
Бумага: 8, Чернила: 80%
Выберите действие (1-5): 1
Операция недоступна при зажатии бумаги.

=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: PaperJamState
Бумага: 8, Чернила: 80%
Выберите действие (1-5):
```

```
Текущее состояние: PaperJamState
Бумага: 8, Чернила: 80%
Выберите действие (1-5): 3
Извлечение зажатой бумаги...

=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: IdleState
Бумага: 8, Чернила: 80%
Выберите действие (1-5): 1
Начало печати...
Процесс печати...
Документ напечатан.

=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: IdleState
Бумага: 7, Чернила: 70%
Выберите действие (1-5):
```

```
5. Выход
Текущее состояние: IdleState
Бумага: 7, Чернила: 70%
Выберите действие (1-5): 2
Загрузка бумаги. Было 7 листов.
Теперь 57 листов.

=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: IdleState
Бумага: 57, Чернила: 70%
Выберите действие (1-5): 4
Картридж заправлен.

=== Меню принтера ===
1. Печать документа
2. Загрузка бумаги
3. Извлечение зажатой бумаги
4. Заправка картриджа
5. Выход
Текущее состояние: IdleState
Бумага: 57, Чернила: 100%
Выберите действие (1-5):
```

Вывод: приобрёл навыки применения паттернов проектирования при решении практических задач с использованием языка Python.