

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ  
Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

Специальность ПО11(о)

Выполнил  
И. А. Головач,  
студент группы ПО11

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

## Вариант 5

**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.

Завод по производству смартфонов. Обеспечить создание нескольких различных моделей мобильных телефонов с заранее выбранными характеристиками.

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

Выполнение:

**Код программы:**

#### lab3\_1.py:

```
from typing import Dict, Type

class Smartphone:
    def __init__(self, name: str):
        self.name = name

    def get_specs(self) -> str:
        return f"Smartphone: {self.name}"

class iPhone(Smartphone):
    def __init__(self):
        super().__init__("iPhone 15 Pro | A17 Pro | 6.1\" OLED | 48MP")

class Galaxy(Smartphone):
    def __init__(self):
        super().__init__("Galaxy S23 Ultra | Snapdragon 8 Gen 2 | 6.8\" AMOLED | 200MP")

class Pixel(Smartphone):
    def __init__(self):
        super().__init__("Pixel 8 Pro | Tensor G3 | 6.7\" OLED | 50MP")

class SmartphoneFactory:
```

```

_models: Dict[str, Type[Smartphone]] = {
    "iphone": iPhone,
    "galaxy": Galaxy,
    "pixel": Pixel,
}

@staticmethod
def create_smartphone(model_name: str) -> Smartphone:
    model_class = SmartphoneFactory._models.get(model_name.lower())
    if model_class:
        return model_class()
    raise ValueError(f"Unknown smartphone model: {model_name}")

if __name__ == "__main__":
    print("Available models: iphone, galaxy, pixel")
    print("Enter 'q' to quit.")

    while True:
        user_input = input("Enter smartphone model: ").strip()
        if user_input == "q":
            break

        try:
            phone = SmartphoneFactory.create_smartphone(user_input)
            print(phone.get_specs())
        except ValueError as e:
            print(e)

```

**Рисунок с результатами работы программы lab3\_1.py:**

```

Available models: iphone, galaxy, pixel
Enter 'q' to quit.
Enter smartphone model: g
Unknown smartphone model: g
Enter smartphone model: iphone
Smartphone: iPhone 15 Pro | A17 Pro | 6.1" OLED | 48MP
Enter smartphone model: galaxy
Smartphone: Galaxy S23 Ultra | Snapdragon 8 Gen 2 | 6.8" AMOLED | 200MP
Enter smartphone model: pixel
Smartphone: Pixel 8 Pro | Tensor G3 | 6.7" OLED | 50MP
Enter smartphone model: q

Process finished with exit code 0

```

## Задание 2.

Проект «Электронный градусник». В проекте должен быть реализован класс, который дает возможность пользоваться аналоговым градусником так же, как и электронным. В классе «Аналоговый градусник» хранится высота ртутного столба и границы измерений (верхняя и нижняя).

В данном случае подойдет **структурный паттерн "Адаптер" (Adapter)**, который позволит использовать аналоговый градусник как электронный, обеспечивая совместимость интерфейсов.

### Почему именно Адаптер?

**Совместимость интерфейсов** — позволяет использовать старый (AnalogThermometer) через новый интерфейс (DigitalThermometer).

**Гибкость** — если позже добавится ещё один тип градусника, его можно будет адаптировать так же.

**Соответствие принципу Open/Closed** — код AnalogThermometer не меняется, добавляется только адаптер.

Если бы мы не использовали адаптер, пришлось бы переписывать логику аналогового градусника, что нарушило бы **принцип единственной ответственности (SRP)**.

Выполнение:

### Код программы:

#### Lab3\_2.py:

```
from typing import Protocol

class DigitalThermometer(Protocol):
    def get_temperature(self) -> float:
        ...

class AnalogThermometer:
    def __init__(self, min_temp: float, max_temp: float):
        self.min_temp = min_temp
        self.max_temp = max_temp
        self.mercury_height: float = 0.0

    def set_mercury_height(self, height: float) -> None:
        if height < 0 or height > 100:
            raise ValueError("Height must be between 0% and 100%")
        self.mercury_height = height

    def get_analog_temp(self) -> float:
        return self.min_temp + (self.max_temp - self.min_temp) * (self.mercury_height /
100)

class ThermometerAdapter:
    def __init__(self, analog_therm: AnalogThermometer):
        self.thermometer = analog_therm
```

```

def get_temperature(self) -> float:
    return self.thermometer.get_analog_temp()

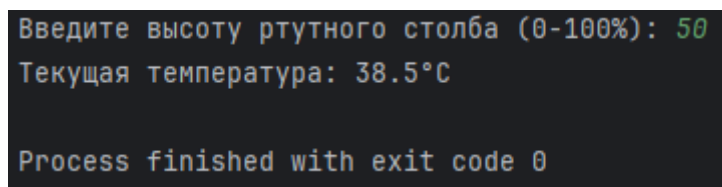
if __name__ == "__main__":
    analog_therm = AnalogThermometer(min_temp=35.0, max_temp=42.0)

    try:
        height = float(input("Введите высоту ртутного столба (0-100%): "))
        analog_therm.set_mercury_height(height)
    except ValueError as e:
        print(f"Ошибка: {e}")
        exit()

    digital_therm = ThermometerAdapter(analog_therm)
    print(f"Текущая температура: {digital_therm.get_temperature():.1f}°C")

```

### Рисунок с результатами работы программы lab3\_2.py:

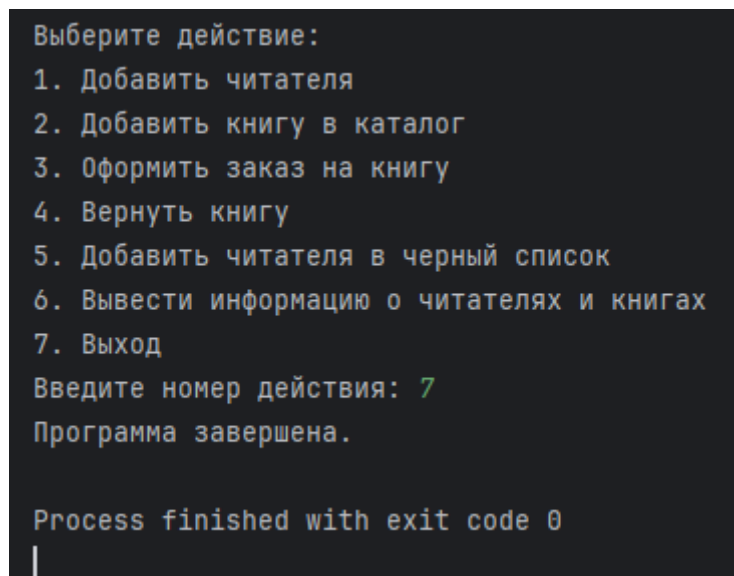


```

Введите высоту ртутного столба (0-100%): 50
Текущая температура: 38.5°C

Process finished with exit code 0

```



```

Выберите действие:
1. Добавить читателя
2. Добавить книгу в каталог
3. Оформить заказ на книгу
4. Вернуть книгу
5. Добавить читателя в черный список
6. Вывести информацию о читателях и книгах
7. Выход
Введите номер действия: 7
Программа завершена.

Process finished with exit code 0
|

```

### Задание 3.

Проект «Банкомат». Предусмотреть выполнение основных операций (ввод пин-кода, снятие суммы, завершение работы) и наличие различных режимов работы (ожидание, аутентификация, выполнение операции, блокировка – если нет денег). Атрибуты: общая сумма денег в банкомате, ID.

Для реализации банкомата с различными режимами работы идеально 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:**

### Lab3\_3.py:

```
from abc import ABC, abstractmethod

class ATMState(ABC):
    @abstractmethod
    def insert_card(self, atm: 'ATM') -> None:
        pass

    @abstractmethod
    def enter_pin(self, atm: 'ATM', pin: str) -> None:
        pass

    @abstractmethod
    def withdraw(self, atm: 'ATM', amount: float) -> None:
        pass

    @abstractmethod
    def eject_card(self, atm: 'ATM') -> None:
        pass

    @abstractmethod
    def display_menu(self, atm: 'ATM') -> None:
        pass

class IdleState(ATMState):
    def insert_card(self, atm: 'ATM') -> None:
        print("\nКарта вставлена. Введите PIN.")
        atm.set_state(AuthenticationState())

    def enter_pin(self, atm: 'ATM', pin: str) -> None:
        print("\nОшибка: Сначала вставьте карту.")

    def withdraw(self, atm: 'ATM', amount: float) -> None:
        print("\nОшибка: Сначала вставьте карту и введите PIN.")

    def eject_card(self, atm: 'ATM') -> None:
        print("\nОшибка: Карта не вставлена.")

    def display_menu(self, atm: 'ATM') -> None:
        print("\n=== Меню ===")
        print("1. Вставить карту")
        print("2. Проверить баланс банкомата")
        print("3. Выйти")
        choice = input("Выберите действие: ")
        if choice == "1":
            self.insert_card(atm)
        elif choice == "2":
            print(f"\nОбщий баланс банкомата: {atm.total_cash} рублей")
```

```

elif choice == "3":
    print("До свидания!")
    exit()
else:
    print("Неверный ввод. Попробуйте ещё раз.")

class AuthenticationState(ATMState):
    def insert_card(self, atm: 'ATM') -> None:
        print("\nОшибка: Карта уже вставлена.")

    def enter_pin(self, atm: 'ATM', pin: str) -> None:
        if pin == "1234": # Пример правильного PIN
            print("\nPIN верный. Выберите операцию.")
            atm.set_state(OperationState())
        else:
            print("\nНеверный PIN. Попробуйте ещё раз.")

    def withdraw(self, atm: 'ATM', amount: float) -> None:
        print("\nОшибка: Сначала введите PIN.")

    def eject_card(self, atm: 'ATM') -> None:
        print("\nКарта извлечена.")
        atm.set_state(IdleState())

    def display_menu(self, atm: 'ATM') -> None:
        print("\n=== Меню ===")
        print("1. Ввести PIN")
        print("2. Извлечь карту")
        print("3. Проверить баланс банкомата")
        print("4. Выйти")
        choice = input("Выберите действие: ")
        if choice == "1":
            pin = input("Введите PIN: ")
            self.enter_pin(atm, pin)
        elif choice == "2":
            self.eject_card(atm)
        elif choice == "3":
            print(f"\nОбщий баланс банкомата: {atm.total_cash} рублей")
        elif choice == "4":
            print("До свидания!")
            exit()
        else:
            print("Неверный ввод. Попробуйте ещё раз.")

class OperationState(ATMState):
    def insert_card(self, atm: 'ATM') -> None:
        print("\nОшибка: Карта уже вставлена.")

    def enter_pin(self, atm: 'ATM', pin: str) -> None:
        print("\nОшибка: PIN уже введён.")

    def withdraw(self, atm: 'ATM', amount: float) -> None:
        if amount <= atm.total_cash:
            print(f"\nВыдано {amount} рублей.")
            atm.total_cash -= amount
            print(f"Остаток в банкомате: {atm.total_cash} рублей")
            atm.set_state(IdleState())
        else:
            print("\nНедостаточно средств. Банкомат заблокирован.")

```

```

        atm.set_state(BlockedState())

def eject_card(self, atm: 'ATM') -> None:
    print("\nКарта извлечена.")
    atm.set_state(IdleState())

def display_menu(self, atm: 'ATM') -> None:
    print("\n=== Меню ===")
    print("1. Снять деньги")
    print("2. Проверить баланс банкомата")
    print("3. Извлечь карту")
    print("4. Выйти")
    choice = input("Выберите действие: ")
    if choice == "1":
        try:
            amount = float(input("Введите сумму для снятия: "))
            self.withdraw(atm, amount)
        except ValueError:
            print("Неверный ввод. Введите число.")
    elif choice == "2":
        print(f"\nОбщий баланс банкомата: {atm.total_cash} рублей")
    elif choice == "3":
        self.eject_card(atm)
    elif choice == "4":
        print("До свидания!")
        exit()
    else:
        print("Неверный ввод. Попробуйте ещё раз.")

class BlockedState(ATMState):
    def insert_card(self, atm: 'ATM') -> None:
        print("\nОшибка: Банкомат заблокирован. Обратитесь в банк.")

    def enter_pin(self, atm: 'ATM', pin: str) -> None:
        print("\nОшибка: Банкомат заблокирован. Обратитесь в банк.")

    def withdraw(self, atm: 'ATM', amount: float) -> None:
        print("\nОшибка: Банкомат заблокирован. Обратитесь в банк.")

    def eject_card(self, atm: 'ATM') -> None:
        print("\nКарта извлечена. Банкомат остаётся заблокированным.")
        atm.set_state(IdleState())

    def display_menu(self, atm: 'ATM') -> None:
        print("\n=== Меню ===")
        print("1. Извлечь карту")
        print("2. Проверить баланс банкомата")
        print("3. Выйти")
        choice = input("Выберите действие: ")
        if choice == "1":
            self.eject_card(atm)
        elif choice == "2":
            print(f"\nОбщий баланс банкомата: {atm.total_cash} рублей")
        elif choice == "3":
            print("До свидания!")
            exit()
        else:
            print("Неверный ввод. Попробуйте ещё раз.")

```



```
class ATM:
    def __init__(self, atm_id: str, total_cash: float):
        self.atm_id = atm_id
        self.total_cash = total_cash # Общая сумма денег в банкомате
        self.state: ATMState = IdleState()

    def set_state(self, state: ATMState) -> None:
        self.state = state

    def insert_card(self) -> None:
        self.state.insert_card(self)

    def enter_pin(self, pin: str) -> None:
        self.state.enter_pin(self, pin)

    def withdraw(self, amount: float) -> None:
        self.state.withdraw(self, amount)

    def eject_card(self) -> None:
        self.state.eject_card(self)

    def display_menu(self) -> None:
        self.state.display_menu(self)

def main():
    atm = ATM("ATM-001", 10000.0)
    print("Добро пожаловать в банкомат!")
    while True:
        atm.display_menu()

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

## Рисунки с результатами работы программы lab3\_3.py:

```
Добро пожаловать в банкомат!

=== Меню ===
1. Вставить карту
2. Проверить баланс банкомата
3. Выйти
Выберите действие: 1

Карта вставлена. Введите PIN.

=== Меню ===
1. Ввести PIN
2. Извлечь карту
3. Проверить баланс банкомата
4. Выйти
Выберите действие: 1
Введите PIN: 1256

Неверный PIN. Попробуйте ещё раз.

=== Меню ===
1. Ввести PIN
2. Извлечь карту
3. Проверить баланс банкомата
4. Выйти
Выберите действие: 1
Введите PIN: 1234

PIN верный. Выберите операцию.

=== Меню ===
1. Снять деньги
2. Проверить баланс банкомата
3. Извлечь карту
4. Выйти
Выберите действие: 2
```

```
Общий баланс банкомата: 10000.0 рублей
```

```
=== Меню ===
```

1. Снять деньги
2. Проверить баланс банкомата
3. Извлечь карту
4. Выйти

```
Выберите действие: 1
```

```
Введите сумму для снятия: 8000
```

```
Выдано 8000.0 рублей.
```

```
Остаток в банкомате: 2000.0 рублей
```

```
=== Меню ===
```

1. Вставить карту
2. Проверить баланс банкомата
3. Выйти

```
Выберите действие: 3
```

```
До свидания!
```

```
Process finished with exit code 0
```

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