МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

Специальность ПО-13

Выполнил:

Д. А. Марковский

студент группы ПО-13

Проверил:

А. А. Крощенко

Доцент кафедры ИИТ

06.03.2026

Брест 2026

Цель работы:

Приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Python.

Задание 1.

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

Код программы:

```
class Smartphone:
    def __init__(self, **kwargs):
        self.model = kwargs.get('model', 'Неизвестная модель')
        self.screen_size = kwargs.get('screen_size', 0)
        self.ram = kwargs.get('ram', 0)
        self.storage = kwargs.get('storage', 0)
        self.camera_mp = kwargs.get('camera_mp', 0)
        self.battery = kwargs.get('battery', 0)
        self.is_on = False

    def __str__(self):
        status = "включен" if self.is_on else "выключен"
        return (f"{self.model}: экран {self.screen_size}\", {self.ram}ГБ ОЗУ, "
                f"{self.storage}ГБ память, камера {self.camera_mp}МП, "
                f"батарея {self.battery}мАч ({status})")

    def turn_on(self):
        if not self.is_on:
            self.is_on = True
            print(f"{self.model} включен")
        else:
            print(f"{self.model} уже включен")

    def turn_off(self):
        if self.is_on:
            self.is_on = False
            print(f"{self.model} выключен")
        else:
            print(f"{self.model} уже выключен")


class SmartphoneFactory:
    def create_smartphone(self):
        pass


class SamsungGalaxyFactory(SmartphoneFactory):
    def create_smartphone(self):
        return Smartphone(
            model="Samsung Galaxy S23",
            screen_size=6.1,
            ram=8,
            storage=256,
            camera_mp=50,
            battery=3900
        )


class IPhoneFactory(SmartphoneFactory):
    def create_smartphone(self):
        return Smartphone(
            model="iPhone 15 Pro",
            screen_size=6.1,
```

```
            ram=6,
            storage=256,
            camera_mp=48,
            battery=3274
        )


class XiaomiFactory(SmartphoneFactory):
    def create_smartphone(self):
        return Smartphone(
            model="Xiaomi 13 Pro",
            screen_size=6.73,
            ram=12,
            storage=512,
            camera_mp=50,
            battery=4820
        )


class GooglePixelFactory(SmartphoneFactory):
    def create_smartphone(self):
        return Smartphone(
            model="Google Pixel 8 Pro",
            screen_size=6.7,
            ram=12,
            storage=128,
            camera_mp=50,
            battery=5050
        )


class OnePlusFactory(SmartphoneFactory):
    def create_smartphone(self):
        return Smartphone(
            model="OnePlus 11",
            screen_size=6.7,
            ram=16,
            storage=256,
            camera_mp=50,
            battery=5000
        )


class SmartphonePlant:
    def __init__(self, name):
        self.name = name
        self.produced_phones = []
        self.factories = {
            "samsung": SamsungGalaxyFactory(),
            "iphone": IPhoneFactory(),
            "xiaomi": XiaomiFactory(),
            "google": GooglePixelFactory(),
            "oneplus": OnePlusFactory()
        }

    def __str__(self):
        return f"Завод '{self.name}' (произведено смартфонов:
{len(self.produced_phones)})"

    def produce_smartphone(self, model_key):
        if model_key.lower() in self.factories:
            factory = self.factories[model_key.lower()]
            phone = factory.create_smartphone()
            self.produced_phones.append(phone)
            print(f"{self.name} произвел: {phone.model}")
            return phone
        print(f"Модель '{model_key}' не производится на заводе")
```

```
        return None

    def produce_multiple(self, orders):
        print(f"\nПроизводство '{orders}' на заводе '{self.name}':")
        results = []
        for model_key, count in orders.items():
            for _ in range(count):
                phone = self.produce_smartphone(model_key)
                if phone:
                    results.append(phone)
        return results

    def show_production_stats(self):
        print(f"\nСтатистика '{self.name}':")
        if not self.produced_phones:
            print("Завод еще ничего не произвел...")
            return

        stats = {}
        for phone in self.produced_phones:
            stats[phone.model] = stats.get(phone.model, 0) + 1

        print(f"Всего произведено: {len(self.produced_phones)} смартфонов")
        for model, count in stats.items():
            print(f" - {model}: {count} шт.")

    def show_all_phones(self):
        print(f"\nСмартфоны произведённые '{self.name}':")
        if not self.produced_phones:
            print("<пусто>")
        else:
            for i, phone in enumerate(self.produced_phones, 1):
                print(f"{i}. {phone}")


plant = SmartphonePlant("ТехноСмарт")
print(plant)
print()

samsung = plant.produce_smartphone("samsung")
iphone = plant.produce_smartphone("iphone")
xiaomi = plant.produce_smartphone("xiaomi")
print()

plant.produce_smartphone("nokia")
print()

print("Тестирование смартфонов:")
if samsung:
    samsung.turn_on()
    samsung.turn_on()
    samsung.turn_off()
print()

if iphone:
    iphone.turn_on()
    iphone.turn_off()
print()

phone_orders = {
    "samsung": 2,
    "iphone": 1,
    "google": 2,
    "oneplus": 1
}
plant.produce_multiple(phone_orders)
print()
```

```
plant.show_production_stats()
print()

plant.show_all_phones()
print()

plant2 = SmartphonePlant("ЭкономСмарт")
print(plant2)

print("\nПроизводство на втором заводе:")
plant2.produce_smartphone("xiaomi")
plant2.produce_smartphone("xiaomi")
plant2.produce_smartphone("oneplus")
print()

plant2.show_production_stats()
plant2.show_all_phones()
```

Спецификация ввода

Ввод не требуется – основная программа приводит тестовое исполнение функциональности.

Спецификация вывода

В консоль выводятся результаты создания завода, производства отдельных моделей смартфонов, попытки производства неподдерживаемой модели, тестирования смартфонов (включение/выключение), массового производства по заказам, статистики производства, открытия второго завода и производства на нём, отображения всех произведённых смартфонов.

Рисунки с результатами работы программы

```
ТехноСмарт произвел: Samsung Galaxy S23
ТехноСмарт произвел: iPhone 15 Pro
ТехноСмарт произвел: Xiaomi 13 Pro

Модель 'nokia' не производится на заводе

Тестирование смартфонов:
Samsung Galaxy S23 включен
Samsung Galaxy S23 уже включен
Samsung Galaxy S23 выключен

iPhone 15 Pro включен
iPhone 15 Pro выключен


Производство '{'samsung': 2, 'iphone': 1, 'google': 2, 'oneplus': 1}' на заводе 'ТехноСмарт':
ТехноСмарт произвел: Samsung Galaxy S23
ТехноСмарт произвел: Samsung Galaxy S23
ТехноСмарт произвел: iPhone 15 Pro
ТехноСмарт произвел: Google Pixel 8 Pro
ТехноСмарт произвел: Google Pixel 8 Pro
ТехноСмарт произвел: OnePlus 11


Статистика 'ТехноСмарт':
Всего произведено: 9 смартфонов
 - Samsung Galaxy S23: 3 шт.
 - iPhone 15 Pro: 2 шт.
 - Xiaomi 13 Pro: 1 шт.
 - Google Pixel 8 Pro: 2 шт.
 - OnePlus 11: 1 шт.
```

```
Смартфоны произведённые 'ТехноСмарт':
1. Samsung Galaxy S23: экран 6.1", 8ГБ ОЗУ, 256ГБ память, камера 50МП, батарея 3900мАч (выключен)
2. iPhone 15 Pro: экран 6.1", 6ГБ ОЗУ, 256ГБ память, камера 48МП, батарея 3274мАч (выключен)
3. Xiaomi 13 Pro: экран 6.73", 12ГБ ОЗУ, 512ГБ память, камера 50МП, батарея 4820мАч (выключен)
4. Samsung Galaxy S23: экран 6.1", 8ГБ ОЗУ, 256ГБ память, камера 50МП, батарея 3900мАч (выключен)
5. Samsung Galaxy S23: экран 6.1", 8ГБ ОЗУ, 256ГБ память, камера 50МП, батарея 3900мАч (выключен)
6. iPhone 15 Pro: экран 6.1", 6ГБ ОЗУ, 256ГБ память, камера 48МП, батарея 3274мАч (выключен)
7. Google Pixel 8 Pro: экран 6.7", 12ГБ ОЗУ, 128ГБ память, камера 50МП, батарея 5050мАч (выключен)
8. Google Pixel 8 Pro: экран 6.7", 12ГБ ОЗУ, 128ГБ память, камера 50МП, батарея 5050мАч (выключен)
9. OnePlus 11: экран 6.7", 16ГБ ОЗУ, 256ГБ память, камера 50МП, батарея 5000мАч (выключен)

Завод 'ЭкономСмарт' (произведено смартфонов: 0)

Производство на втором заводе:
ЭкономСмарт произвел: Xiaomi 13 Pro
ЭкономСмарт произвел: Xiaomi 13 Pro
ЭкономСмарт произвел: OnePlus 11


Статистика 'ЭкономСмарт':
Всего произведено: 3 смартфонов
 - Xiaomi 13 Pro: 2 шт.
 - OnePlus 11: 1 шт.

Смартфоны произведённые 'ЭкономСмарт':
1. Xiaomi 13 Pro: экран 6.73", 12ГБ ОЗУ, 512ГБ память, камера 50МП, батарея 4820мАч (выключен)
2. Xiaomi 13 Pro: экран 6.73", 12ГБ ОЗУ, 512ГБ память, камера 50МП, батарея 4820мАч (выключен)
3. OnePlus 11: экран 6.7", 16ГБ ОЗУ, 256ГБ память, камера 50МП, батарея 5000мАч (выключен)
```

Задание 2.

5) 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).

Код программы:

```
import random


class ElectronicThermometer:
    def __init__(self, name):
        self.name = name
        self.is_on = False
        self.current_temperature = 0.0

    def __str__(self):
        status = "включен" if self.is_on else "выключен"
        return f"Электронный градусник '{self.name}':
{self.current_temperature}°C ({status})"

    def turn_on(self):
        if not self.is_on:
            self.is_on = True
            print(f"Электронный градусник '{self.name}' включен")
        else:
            print(f"Электронный градусник '{self.name}' уже включен")

    def turn_off(self):
        if self.is_on:
            self.is_on = False
            print(f"Электронный градусник '{self.name}' выключен")
        else:
            print(f"Электронный градусник '{self.name}' уже выключен")

    def measure_temperature(self):
        if not self.is_on:
            print(f"Ошибка: градусник '{self.name}' выключен!")
            return None

        self.current_temperature = round(random.uniform(35.0, 42.0), 1)
        return self.current_temperature

    def get_temperature(self):
        return self.current_temperature if self.is_on else None


class AnalogThermometer:
    def __init__(self, name, min_temp, max_temp, max_height_mm=100):
        self.name = name
        self.min_temp = min_temp
        self.max_temp = max_temp
        self.max_height_mm = max_height_mm
        self.mercury_height = 0.0

    def __str__(self):
        return (f"Аналоговый градусник '{self.name}': "
                f"диапазон [{self.min_temp}°C, {self.max_temp}°C], "
                f"столб {self.mercury_height}мм")

    def shake(self):
        self.mercury_height = 0
        print(f"Аналоговый градусник '{self.name}' встряхнут")
```

```
    def measure_with_mercury(self, temperature_celsius):
        if temperature_celsius < self.min_temp or temperature_celsius >
self.max_temp:
            print(f"Температура {temperature_celsius}°C вне диапазона
измерений!")
            return False

        self.mercury_height = round(((temperature_celsius - self.min_temp) / (
                self.max_temp - self.min_temp)) * self.max_height_mm, 1)

        print(f"Аналоговый градусник '{self.name}' измерил: "
              f"столб поднялся на {self.mercury_height:.1f}мм")
        return True

    def get_mercury_height(self):
        return self.mercury_height


class ThermometerAdapter(ElectronicThermometer):
    def __init__(self, analog_thermometer):
        super().__init__(f"Адаптер для {analog_thermometer.name}")

        self.analog = analog_thermometer
        self.last_measured_temp = 0.0

    def __str__(self):
        return (f"Адаптер: {self.analog.name} -> {self.name}\n"
                f"Состояние: {'включен' if self.is_on else 'выключен'}\n"
                f"Текущие показания: {self.last_measured_temp}°C")

    def turn_on(self):
        if not self.is_on:
            self.is_on = True
            print(f"Адаптер для '{self.analog.name}' активирован")
        else:
            print(f"Адаптер для '{self.analog.name}' уже активирован")

    def turn_off(self):
        if self.is_on:
            self.is_on = False
            print(f"Адаптер для '{self.analog.name}' деактивирован")
        else:
            print(f"Адаптер для '{self.analog.name}' уже деактивирован")

    def measure_temperature(self):
        if not self.is_on:
            print(f"Ошибка: адаптер для '{self.analog.name}' выключен!")
            return None

        self.analog.shake()

        temp_to_measure = round(random.uniform(
            self.analog.min_temp + 1,
            self.analog.max_temp - 1
        ), 1)

        success = self.analog.measure_with_mercury(temp_to_measure)

        if success:
            temp_range = self.analog.max_temp - self.analog.min_temp
            height_range = self.analog.max_height_mm
            mercury_height = self.analog.get_mercury_height()

            self.last_measured_temp = round(
                (mercury_height / height_range) * temp_range +
self.analog.min_temp,
                1
```

```
            )

            print(f"Адаптер преобразовал: {mercury_height:.1f}мм ->
{self.last_measured_temp}°C")
            return self.last_measured_temp
        print("Не удалось измерить температуру...")
        return None

    def get_temperature(self):
        return self.last_measured_temp if self.is_on else None


class MedicalCabinet:
    def __init__(self, name):
        self.name = name
        self.thermometers = []

    def __str__(self):
        return f"Медицинский кабинет '{self.name}' (градусников:
{len(self.thermometers)})"

    def add_thermometer(self, thermometer):
        self.thermometers.append(thermometer)
        print(f"В кабинет добавлен: {thermometer.name}")

    def measure_patient(self, patient_name):
        print(f"\nИзмерение температуры пациента '{patient_name}':")

        results = {}
        for thermo in self.thermometers:
            print(f"\nИспользуем: {thermo.name}")

            if hasattr(thermo, 'turn_on'):
                thermo.turn_on()

            temperature = thermo.measure_temperature()
            if temperature is not None:
                results[thermo.name] = temperature
                print(f"Результат: {temperature}°C")
            else:
                print("Не удалось измерить температуру...")

            if hasattr(thermo, 'turn_off'):
                thermo.turn_off()

        return results


print("Создание градусников:")

electronic = ElectronicThermometer("Eco Thermometer")
print(electronic)

analog = AnalogThermometer("Ртутный", min_temp=34, max_temp=42,
max_height_mm=120)
print(analog)

adapter = ThermometerAdapter(analog)
print(adapter)

print("\nСоздание медицинского кабинета:")
cabinet = MedicalCabinet("Кабинет 142")
print(cabinet)
cabinet.add_thermometer(electronic)
cabinet.add_thermometer(adapter)

print(cabinet.measure_patient("Пациент 1"))
```

```
print(f"\nИсходный {analog}")
analog.measure_with_mercury(38.5)
print(f"После измерения: {analog}")
print(f"Высота столба: {analog.get_mercury_height():.1f}мм")

adapter.turn_on()
temp = adapter.measure_temperature()
if temp:
    print(f"Получена температура: {temp}°C")
print("\nСостояние аналогового градусника после измерения через адаптер:")
print(analog)
adapter.turn_off()
```

Спецификация ввода

Ввод не требуется – основная программа приводит тестовое исполнение функциональности.

Спецификация вывода

В консоль выводятся результаты создания электронного и аналогового градусников, создания адаптера, создания медицинского кабинета, добавления градусников в кабинет, измерения температуры пациента разными градусниками, прямой работы с аналоговым градусником (измерение, высота столба), работы с аналоговым градусником через адаптер, сравнения интерфейсов градусников.

Рисунки с результ атами работы программы

```
Создание градусников:
Электронный градусник 'Eco Thermometer': 0.0°C (выключен)
Аналоговый градусник 'Ртутный': диапазон [34°C, 42°C], столб 0.0мм
Адаптер: Ртутный -> Адаптер для Ртутный
Состояние: выключен
Текущие показания: 0.0°C

Создание медицинского кабинета:
Медицинский кабинет 'Кабинет 142' (градусников: 0)
В кабинет добавлен: Eco Thermometer
В кабинет добавлен: Адаптер для Ртутный

Измерение температуры пациента 'Пациент 1':

Используем: Eco Thermometer
Электронный градусник 'Eco Thermometer' включен
Результат: 37.5°C
Электронный градусник 'Eco Thermometer' выключен

Используем: Адаптер для Ртутный
Адаптер для 'Ртутный' активирован
Аналоговый градусник 'Ртутный' встряхнут
Аналоговый градусник 'Ртутный' измерил: столб поднялся на 21.0мм
Адаптер преобразовал: 21.0мм -> 35.4°C
Результат: 35.4°C
Адаптер для 'Ртутный' деактивирован
{'Eco Thermometer': 37.5, 'Адаптер для Ртутный': 35.4}

Исходный Аналоговый градусник 'Ртутный': диапазон [34°C, 42°C], столб 21.0мм
Аналоговый градусник 'Ртутный' измерил: столб поднялся на 67.5мм
```

Задание 3.

5) 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.

Код программы:

```
class ATM:
    def __init__(self, atm_id, initial_money):
        self.atm_id = atm_id
        self.total_money = initial_money
        self.state = WaitingState(self)
        self.current_card = None
        self.failed_attempts = 0

    def __str__(self):
        return f"Банкомат {self.atm_id}: {self.total_money} руб., состояние:
{self.state}"

    def set_state(self, state):
        self.state = state
        print(f"Банкомат {self.atm_id} перешел в режим: {self.state}")
```

```
    def insert_card(self, card):
        self.state.insert_card(card)

    def enter_pin(self, pin):
        self.state.enter_pin(pin)

    def withdraw_money(self, amount):
        self.state.withdraw_money(amount)

    def cancel(self):
        self.state.cancel()

    def has_enough_money(self, amount):
        return self.total_money >= amount

    def dispense_money(self, amount):
        if self.has_enough_money(amount):
            self.total_money -= amount
            print(f"Выдано {amount} руб.")
            return True
        return False


class ATMState:
    def __init__(self, atm):
        self.atm = atm

    def insert_card(self, card):
        print("Операция не поддерживается в текущем режиме")

    def enter_pin(self, pin):
        print("Операция не поддерживается в текущем режиме")

    def withdraw_money(self, amount):
        print("Операция не поддерживается в текущем режиме")

    def cancel(self):
        print("Операция не поддерживается в текущем режиме")


class WaitingState(ATMState):
    def __str__(self):
        return "Ожидание карты"

    def insert_card(self, card):
        print(f"Карта {card} вставлена")
        self.atm.current_card = card
        self.atm.set_state(PinEntryState(self.atm))


class PinEntryState(ATMState):
    def __str__(self):
        return "Ввод PIN-кода"

    def enter_pin(self, pin):
        correct_pin = "1234"
        if pin == correct_pin:
            print("PIN-код принят")
            self.atm.failed_attempts = 0
            self.atm.set_state(OperationState(self.atm))
        else:
            self.atm.failed_attempts += 1
            print(f"Неверный PIN-код. Попытка {self.atm.failed_attempts}/3")
            if self.atm.failed_attempts >= 3:
                print("Превышено количество попыток. Карта заблокирована")
                self.atm.set_state(BlockedState(self.atm))
```

```
    def cancel(self):
        print("Операция отменена")
        self.atm.current_card = None
        self.atm.set_state(WaitingState(self.atm))


class OperationState(ATMState):
    def __str__(self):
        return "Выполнение операции"

    def withdraw_money(self, amount):
        if amount <= 0:
            print("Некорректная сумма")
            return

        if not self.atm.has_enough_money(amount):
            print("Недостаточно средств в банкомате")
            self.atm.set_state(NoMoneyState(self.atm))
            return

        if amount > 10000:
            print("Превышен лимит на снятие (10000 руб.)")
            return

        if self.atm.dispense_money(amount):
            print(f"Снято {amount} руб. Остаток: {self.atm.total_money} руб.")
            print("Заберите карту")
            self.atm.current_card = None
            self.atm.set_state(WaitingState(self.atm))

    def cancel(self):
        print("Операция отменена")
        self.atm.current_card = None
        self.atm.set_state(WaitingState(self.atm))


class NoMoneyState(ATMState):
    def __str__(self):
        return "Блокировка (нет денег)"

    def insert_card(self, card):
        print("Банкомат временно не обслуживает. Нет денег")

    def cancel(self):
        print("Возврат к ожиданию...")
        self.atm.set_state(WaitingState(self.atm))


class BlockedState(ATMState):
    def __str__(self):
        return "Блокировка (неверный PIN)"

    def insert_card(self, card):
        print("Карта заблокирована. Обратитесь в банк")

    def cancel(self):
        print("Возврат к ожиданию...")
        self.atm.failed_attempts = 0
        self.atm.current_card = None
        self.atm.set_state(WaitingState(self.atm))


class Card:
    def __init__(self, card_number, owner):
        self.card_number = card_number
        self.owner = owner
```

```
    def __str__(self):
        return f"{self.card_number} ({self.owner})"


demo_atm = ATM("ATM-001", 50000)
print(demo_atm)

demo_card = Card("1234-5678-9012-3456", "Владелец 1")
print(demo_card)

print("СЦЕНАРИЙ 1: Успешное снятие денег")
demo_atm.insert_card(demo_card)
demo_atm.enter_pin("1234")
demo_atm.withdraw_money(5000)
print(f"\nИтог: {demo_atm}")

print("СЦЕНАРИЙ 2: Неверный PIN-код")
demo_atm.insert_card(demo_card)
demo_atm.enter_pin("0000")
demo_atm.enter_pin("1111")
demo_atm.enter_pin("2222")
print(f"\nИтог: {demo_atm}")

demo_atm.cancel()
print(f"После сброса: {demo_atm}")

print("СЦЕНАРИЙ 3: Недостаточно средств в банкомате")
atm2 = ATM("ATM-002", 1000)
print(atm2)

atm2.insert_card(demo_card)
atm2.enter_pin("1234")
atm2.withdraw_money(2000)
print(f"\nИтог: {atm2}")

atm2.cancel()
print(f"После сброса: {atm2}")

print("СЦЕНАРИЙ 4: Отмена операции")
demo_atm.insert_card(demo_card)
demo_atm.enter_pin("1234")
demo_atm.cancel()
demo_atm.insert_card(demo_card)
print(f"\nИтог: {demo_atm}")

print("ДЕМОНСТРАЦИЯ ВСЕХ СОСТОЯНИЙ")
states_demo = ATM("DEMO-001", 5000)
print("\n1. Ожидание карты:")
print(states_demo)
print("\n2. Ввод PIN-кода:")
states_demo.insert_card(demo_card)
print(states_demo)
print("\n3. Выполнение операции:")
states_demo.enter_pin("1234")
print(states_demo)
print("\n4. Блокировка (нет денег):")
states_demo.total_money = 0
states_demo.withdraw_money(1000)
print(states_demo)
print("\n5. Возврат в ожидание:")
states_demo.cancel()
print(states_demo)
```

Спецификация ввода

Ввод не требуется – основная программа приводит тестовое исполнение функциональности.

Спецификация вывода

В консоль выводятся результаты создания банкомата и карты, успешного снятия денег, ввода неверного PIN-кода (с подсчётом попыток), блокировки карты при превышении попыток, попытки снятия суммы больше доступной в банкомате, отмены операции, демонстрации переключения между всеми состояниями банкомата (ожидание, ввод PIN, выполнение операции, блокировка), итоговое состояние банкомата после каждой операции.

Рисунки с результатами работы программы

```
Банкомат ATM-001: 50000 руб., состояние: Ожидание карты
1234-5678-9012-3456 (Владелец 1)
СЦЕНАРИЙ 1: Успешное снятие денег
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат ATM-001 перешел в режим: Ввод PIN-кода
PIN-код принят
Банкомат ATM-001 перешел в режим: Выполнение операции
Выдано 5000 руб.
Снято 5000 руб. Остаток: 45000 руб.
Заберите карту
Банкомат ATM-001 перешел в режим: Ожидание карты

Итог: Банкомат ATM-001: 45000 руб., состояние: Ожидание карты
СЦЕНАРИЙ 2: Неверный PIN-код
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат ATM-001 перешел в режим: Ввод PIN-кода
Неверный PIN-код. Попытка 1/3
Неверный PIN-код. Попытка 2/3
Неверный PIN-код. Попытка 3/3
Превышено количество попыток. Карта заблокирована
Банкомат ATM-001 перешел в режим: Блокировка (неверный PIN)

Итог: Банкомат ATM-001: 45000 руб., состояние: Блокировка (неверный PIN)
Возврат к ожиданию...
Банкомат ATM-001 перешел в режим: Ожидание карты
После сброса: Банкомат ATM-001: 45000 руб., состояние: Ожидание карты
СЦЕНАРИЙ 3: Недостаточно средств в банкомате
Банкомат ATM-002: 1000 руб., состояние: Ожидание карты
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат ATM-002 перешел в режим: Ввод PIN-кода
```

```
Итог: Банкомат ATM-001: 45000 руб., состояние: Блокировка (неверный PIN)
Возврат к ожиданию...
Банкомат ATM-001 перешел в режим: Ожидание карты
После сброса: Банкомат ATM-001: 45000 руб., состояние: Ожидание карты
СЦЕНАРИЙ 3: Недостаточно средств в банкомате
Банкомат ATM-002: 1000 руб., состояние: Ожидание карты
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат ATM-002 перешел в режим: Ввод PIN-кода
PIN-код принят
Банкомат ATM-002 перешел в режим: Выполнение операции
Недостаточно средств в банкомате
Банкомат ATM-002 перешел в режим: Блокировка (нет денег)

Итог: Банкомат ATM-002: 1000 руб., состояние: Блокировка (нет денег)
Возврат к ожиданию...
Банкомат ATM-002 перешел в режим: Ожидание карты
После сброса: Банкомат ATM-002: 1000 руб., состояние: Ожидание карты
СЦЕНАРИЙ 4: Отмена операции
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат ATM-001 перешел в режим: Ввод PIN-кода
PIN-код принят
Банкомат ATM-001 перешел в режим: Выполнение операции
Операция отменена
Банкомат ATM-001 перешел в режим: Ожидание карты
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат ATM-001 перешел в режим: Ввод PIN-кода

Итог: Банкомат ATM-001: 45000 руб., состояние: Ввод PIN-кода
```

```
Итог: Банкомат ATM-001: 45000 руб., состояние: Ввод PIN-кода
ДЕМОНСТРАЦИЯ ВСЕХ СОСТОЯНИЙ

1. Ожидание карты:
Банкомат DEMO-001: 5000 руб., состояние: Ожидание карты

2. Ввод PIN-кода:
Карта 1234-5678-9012-3456 (Владелец 1) вставлена
Банкомат DEMO-001 перешел в режим: Ввод PIN-кода
Банкомат DEMO-001: 5000 руб., состояние: Ввод PIN-кода

3. Выполнение операции:
PIN-код принят
Банкомат DEMO-001 перешел в режим: Выполнение операции
Банкомат DEMO-001: 5000 руб., состояние: Выполнение операции

4. Блокировка (нет денег):
Недостаточно средств в банкомате
Банкомат DEMO-001 перешел в режим: Блокировка (нет денег)
Банкомат DEMO-001: 0 руб., состояние: Блокировка (нет денег)

5. Возврат в ожидание:
Возврат к ожиданию...
Банкомат DEMO-001 перешел в режим: Ожидание карты
Банкомат DEMO-001: 0 руб., состояние: Ожидание карты
```

Вывод:

Я закрепил навыки применения паттернов проектирования при решении практических задач с использованием языка Python.