

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ФАКУЛЬТЕТ

ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

**Отчёт по лабораторной работе №3**

Специальность ПО11

Выполнил  
Лесько М.И.  
студент группы ПО11

Проверил  
А. А. Крощенко ст.  
преп. кафедры ИИТ,  
11.04.2025 г.

Брест 2025

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

**Задание 1. Заводы по производству автомобилей. Реализовать возможность создавать автомобили различных типов на различных заводах.**

**Первая группа заданий (порождающий паттерн)**

Выполнение:

**Код программы:**

```
from abc import ABC, abstractmethod
from datetime import datetime

class Car(ABC):
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    @abstractmethod
    def get_info(self):
        pass

    def calculate_age(self):
        current_year = datetime.now().year
        age = current_year - self.year
        if age == 1:
            return f"{age} год"
        elif 2 <= age <= 4:
            return f"{age} года"
        else:
            return f"{age} лет"

class Sedan(Car):
    def get_info(self):
        return f"Седан {self.brand} {self.model} {self.year} года ({self.calculate_age()})"

class SUV(Car):
    def get_info(self):
        return f"Внедорожник {self.brand} {self.model} {self.year} года ({self.calculate_age()})"

class Truck(Car):
    def get_info(self):
        return f"Грузовик {self.brand} {self.model} {self.year} года ({self.calculate_age()})"

class CarFactory(ABC):
    def __init__(self, brand):
        self.brand = brand

    @abstractmethod
    def create_sedan(self, model, year) -> Sedan:
        pass

    @abstractmethod
    def create_suv(self, model, year) -> SUV:
        pass

    @abstractmethod
    def create_truck(self, model, year) -> Truck:
        pass

class ToyotaFactory(CarFactory):
    def __init__(self):
        super().__init__("Toyota")

    def create_sedan(self, model, year) -> Sedan:
        return Sedan(self.brand, model if model else "Camry", year)

    def create_suv(self, model, year) -> SUV:
        return SUV(self.brand, model if model else "RAV4", year)

    def create_truck(self, model, year) -> Truck:
        return Truck(self.brand, model if model else "Hilux", year)
```

```

class FordFactory(CarFactory):
    def __init__(self):
        super().__init__("Ford")

    def create_sedan(self, model, year) -> Sedan:
        return Sedan(self.brand, model if model else "Focus", year)

    def create_suv(self, model, year) -> SUV:
        return SUV(self.brand, model if model else "Explorer", year)

    def create_truck(self, model, year) -> Truck:
        return Truck(self.brand, model if model else "F-150", year)

class VolkswagenFactory(CarFactory):
    def __init__(self):
        super().__init__("Volkswagen")

    def create_sedan(self, model, year) -> Sedan:
        return Sedan(self.brand, model if model else "Passat", year)

    def create_suv(self, model, year) -> SUV:
        return SUV(self.brand, model if model else "Tiguan", year)

    def create_truck(self, model, year) -> Truck:
        return Truck(self.brand, model if model else "Amarok", year)

def get_user_choice(options, prompt):
    print(prompt)
    for key, value in options.items():
        print(f"{key}. {value}")

    while True:
        choice = input("Ваш выбор: ")
        if choice in options:
            return choice
        print("Некорректный ввод. Попробуйте снова.")

def get_valid_year():
    current_year = datetime.now().year
    while True:
        try:
            year = int(input(f"Введите год выпуска (1886-{current_year}): "))
            if 1886 <= year <= current_year:
                return year
            print(f"Год должен быть между 1886 и {current_year}")
        except ValueError:
            print("Пожалуйста, введите число.")

def main():
    factories = {
        '1': ToyotaFactory(),
        '2': FordFactory(),
        '3': VolkswagenFactory()
    }

    car_types = {
        '1': ('Седан', 'create_sedan'),
        '2': ('Внедорожник', 'create_suv'),
        '3': ('Грузовик', 'create_truck')
    }

    print("\nСистема создания автомобилей с полными характеристиками")

    while True:
        # Выбор марки
        factory_options = {k: v.brand for k, v in factories.items()}
        factory_choice = get_user_choice(factory_options, "\nВыберите марку автомобиля:")
        factory = factories[factory_choice]

        # Выбор типа
        type_choice = get_user_choice(
            {k: v[0] for k, v in car_types.items()},
            "\nВыберите тип автомобиля:"
        )
        car_type, create_method = car_types[type_choice]

        # Ввод модели

```

```

model = input(f"\nВведите модель {factory.brand} (или нажмите Enter для модели по
умолчанию): ").strip()

# Ввод года
year = get_valid_year()

# Создание автомобиля      car = getattr(factory, create_method)(model,
year)

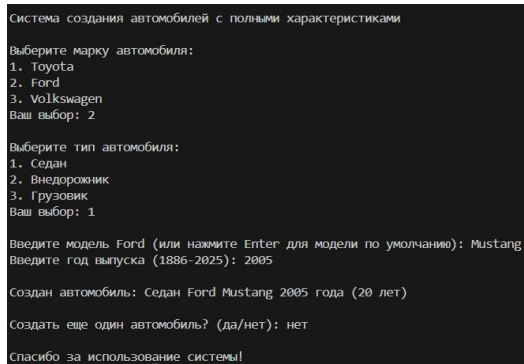
# Вывод результата
print(f"\nСоздан автомобиль: {car.get_info()}")

# Повтор      if input("\nСоздать еще один автомобиль? (да/нет): ").lower() != 'да':
print("\nСпасибо за использование системы!")
break

if __name__ == "__main__":    main()

```

## Рисунок с результатом работы программы:



```

Система создания автомобилей с полными характеристиками
Выберите марку автомобиля:
1. Toyota
2. Ford
3. Volkswagen
Ваш выбор: 2

Выберите тип автомобиля:
1. Седан
2. Внедорожник
3. Грузовик
Ваш выбор: 1

Введите модель Ford (или нажмите Enter для модели по умолчанию): Mustang
Введите год выпуска (1886-2025): 2005

Создан автомобиль: Седан Ford Mustang 2005 года (20 лет)

Создать еще один автомобиль? (да/нет): нет

спасибо за использование системы!

```

**Задание 2. Проект «Универсальная электронная карта». В проекте должна быть реализована универсальная электронная карта, в которой есть функции паспорта, страхового полиса, банковской карты и т. д.**

## Вторая группа заданий (структурный паттерн)

Выполнение:

### Код программы:

```

from abc import ABC, abstractmethod
from datetime import date
from typing import List, Dict, Type

class CardComponent(ABC):
    @abstractmethod def
show_info(self):
    pass
    @abstractmethod def is_valid(self) -
> bool:
    pass
    @abstractmethod def
edit(self):
    pass
class PassportComponent(CardComponent):
    def __init__(self):
self.full_name = ""
self.birth_date = None
self.passport_number = ""
self.issue_date = None
self.expiry_date = None

    def show_info(self):
        print("\n== Паспортные данные ==")
        print(f"1. ФИО: {self.full_name}")
        print(f"2. Дата рождения: {self.birth_date}")
        print(f"3. Номер паспорта:

```

```

{self.passport_number}")    print(f'4. Дата выдачи: {self.issue_date}')
print(f'5. Срок действия: {self.expiry_date}')
    print(f'Действителен: {'Да' if self.is_valid() else 'Нет'}')

def is_valid(self) -> bool:
    return self.expiry_date and date.today() <= self.expiry_date

def edit(self):
    print("\nРедактирование паспортных данных:")    self.full_name = input("Введите ФИО: ") or self.full_name    self.birth_date =
self._input_date("Дата рождения (ГГГГ-ММ-ДД): ", self.birth_date)    self.passport_number = input("Номер паспорта: ") or
self.passport_number    self.issue_date = self._input_date("Дата выдачи (ГГГГ-ММ-ДД): ", self.issue_date)    self.expiry_date =
self._input_date("Срок действия (ГГГГ-ММ-ДД): ", self.expiry_date)

def _input_date(self, prompt, current_value=None):    while True:
    date_str = input(prompt) or (str(current_value) if current_value else "")    if not date_str:
        return current_value    try:
        return date.fromisoformat(date_str)    except ValueError:
        print("Неверный формат даты. Используйте ГГГГ-ММ-ДД")

class InsuranceComponent(CardComponent):    def __init__(self):
    self.policy_number = ""    self.insurance_company = ""
    self.expiry_date = None

def show_info(self):
    print("\n==== Страховой полис ====")
    print(f'1. Номер полиса: {self.policy_number}')    print(f'2. Страховая компания:
{self.insurance_company}')    print(f'3. Срок действия: {self.expiry_date}')
    print(f'Действителен: {'Да' if self.is_valid() else 'Нет'}')

def is_valid(self) -> bool:
    return self.expiry_date and date.today() <= self.expiry_date
def edit(self):
    print("\nРедактирование страхового полиса:")
    self.policy_number = input("Номер полиса: ") or self.policy_number    self.insurance_company = input("Страховая компания: ") or
self.insurance_company    self.expiry_date = self._input_date("Срок действия (ГГГГ-ММ-ДД): ", self.expiry_date)

def _input_date(self, prompt, current_value=None):    while True:
    date_str = input(prompt) or (str(current_value) if current_value else "")    if not date_str:
        return current_value    try:
        return date.fromisoformat(date_str)    except ValueError:
        print("Неверный формат
даты. Используйте ГГГГ-ММ-ДД")

class BankCardComponent(CardComponent):    def __init__(self):
    self.card_number = ""    self.bank_name = ""
    self.expiry_date = None
    self.balance = 0.0

def show_info(self):
    print("\n==== Банковская карта ====")    print(f'1. Номер карты: **** *{self.card_number[-4]} if self.card_number else '****')")
print(f'2. Банк: {self.bank_name}')    print(f'3. Срок действия: {self.expiry_date}')    print(f'4. Баланс: {self.balance:.2f} руб.")
print(f'Действительна: {'Да' if self.is_valid() else 'Нет'}')

def is_valid(self) -> bool:
    return self.expiry_date and date.today() <= self.expiry_date

def edit(self):
    print("\nРедактирование банковской карты:")    self.card_number = input("Номер карты (16 цифр): ") or self.card_number
self.bank_name = input("Название банка: ") or self.bank_name    self.expiry_date = self._input_date("Срок действия (ГГГГ-ММ-ДД): ",
self.expiry_date)    self._input_balance()

def _input_date(self, prompt, current_value=None):    while True:
    date_str = input(prompt) or (str(current_value) if current_value else "")
    if not date_str:
        return current_value    try:
        return date.fromisoformat(date_str)    except ValueError:
        print("Неверный формат даты.
Используйте ГГГГ-ММ-ДД")

def _input_balance(self):
    while True:
        balance_str = input(f'Баланс ( {self.balance:.2f}): ') or str(self.balance)    try:

```

```

        self.balance = float(balance_str)
        except ValueError:
            print("Неверный формат суммы. Используйте число (например 1000.50)")

class UniversalElectronicCard:
    def __init__(self):
        self.components: Dict[str, CardComponent] = {
            "1": PassportComponent(),
            "2": InsuranceComponent(),
            "3": BankCardComponent()
        }

    def show_info(self):
        print("\n=== Универсальная электронная карта ===")
        for name, component in self.components.items():
            component.show_info()

    def edit_component(self):
        print("\nВыберите компонент для редактирования:")
        print("1. Паспортные данные")
        print("2. Страховой полис")
        print("3. Банковская карта")
        print("0. Назад")

        choice = input("Ваш выбор: ")
        if choice in self.components:
            self.components[choice].edit()
        elif choice != "0":
            print("Неверный выбор")

    def is_valid(self) -> bool:
        return all(component.is_valid() for component in self.components.values())

def main():
    card = UniversalElectronicCard()

    while True:
        print("\n=== Меню управления электронной картой ===")
        print("1. Просмотреть данные карты")
        print("2. Редактировать данные")
        print("3. Проверить валидность карты")
        print("0. Выход")

        choice = input("Ваш выбор: ")

        if choice == "1":
            card.show_info()
        elif choice == "2":
            card.edit_component()
        elif choice == "3":
            valid = card.is_valid()
            print(f"Карта {'действительна' if valid else 'недействительна'}")
            if not valid:
                print("Проверьте сроки действия компонентов")
            elif choice == "0":
                print("До свидания!")
                break
            else:
                print("Неверный выбор, попробуйте снова")

if __name__ == "__main__":
    main()

```

## Рисунок с результатом работы программы:

```

=== Универсальная электронная карта ===

=== Паспортные данные ===
1. ФИО: Лесько Максим Игоревич
2. Дата рождения: 2005-10-04
3. Номер паспорта: 123456789
4. Дата выдачи: 2020-04-05
5. Срок действия: 2028-04-05
Действителен: Да

=== Страховой полис ===
1. Номер полиса: 12345678
2. Страховая компания: ОАО
3. Срок действия: 2028-04-05
Действителен: Да

=== Банковская карта ===
1. Номер карты: **** * 5678
2. Банк: ББ
3. Срок действия: 2028-04-05
4. Баланс: 150.28 руб.
Действительна: Да

=== Меню управления электронной картой ===
1. Просмотреть данные карты
2. Редактировать данные
3. Проверить валидность карты
0. Выход
Ваш выбор: █

```

### Задание 3. Проект «Принтеры». В проекте должны быть реализованы разные модели принтеров, которые выполняют разные виды печати.

#### Третья группа заданий (поведенческий паттерн)

Выполнение:

#### Код программы:

```
from abc import ABC, abstractmethod

# Базовый класс для печати class
PrintStrategy(ABC): @abstractmethod
    def print(self, text: str):
        pass

# Конкретные типы печати
class LaserPrint(PrintStrategy):
    def print(self, text: str):
        print(f"[Лазерная печать] {text}")

class InkjetPrint(PrintStrategy):
    def print(self, text: str):
        print(f"[Струйная печать] {text}")

class MatrixPrint(PrintStrategy):
    def print(self, text: str):
        print(f"[Матричная печать] {text}")

# Класс принтера class Printer:
    def __init__(self, name: str):
        self.name = name
        self.strategy = None

    def set_print_type(self, strategy: PrintStrategy):
        self.strategy = strategy

    def do_print(self, text: str):
        print(f"\nПринтер '{self.name}' готов к печати:")
        if self.strategy:
            self.strategy.print(text)
        else:
            print("Ошибка: не выбран тип печати!")

# Главное меню def main():
    # Создаем список принтеров
    printers = [
        Printer("Офисный HP"),
        Printer("Домашний Canon"),
        Printer("Магазинный Epson")
    ]
    # Создаем типы печати
    print_types = {
        1: LaserPrint(),
        2: InkjetPrint(),
        3: MatrixPrint()
    }

    current_printer = None

    while True:
        print("\n=== Меню управления принтерами ===")
        print("1. Выбрать принтер")
        print("2. Выбрать тип печати")
        print("3. Напечатать текст")
        print("0. Выход")

        choice = input("Выберите действие: ")

        if choice == "1":
            print("\nДоступные принтеры:")
            for i, printer in enumerate(printers, 1):
                print(f"{i}. {printer.name}")
```

```

try:
    num = int(input("Номер принтера: ")) - 1
    current_printer = printers[num]
    print(f"Выбран: {current_printer.name}")
except:
    print("Ошибка выбора!")

elif choice == "2":
    if not current_printer:
        print("Сначала выберите принтер!")
        continue

    print("\nТипы печати:")
    print("1. Лазерная")
    print("2. Струйная")
    print("3. Матричная")

try:
    num = int(input("Номер типа печати: "))
    current_printer.set_print_type(print_types[num])
    print("Тип печати установлен")
except:
    print("Ошибка выбора!")

elif choice == "3":
    if not current_printer:
        print("Сначала выберите принтер!")
        continue

    text = input("Введите текст для печати: ")
    current_printer.do_print(text)

elif choice == "0":
    print("До свидания!")
    break
else:
    print("Неверный ввод!")

if __name__ == "__main__":
    main()

```

**Рисунок с результатом работы программы:**

```

=== Меню управления принтерами ===
1. Выбрать принтер
2. Выбрать тип печати
3. Напечатать текст
0. Выход
Выберите действие: 1

Доступные принтеры:
1. Офисный HP
2. Домашний Canon
3. Магазинный Epson
Номер принтера: 1
Выбран: Офисный HP

=== Меню управления принтерами ===
1. Выбрать принтер
2. Выбрать тип печати
3. Напечатать текст
0. Выход
Выберите действие: 2

Типы печати:
1. Лазерная
2. Струйная
3. Матричная
Номер типа печати: 1
Тип печати установлен

=== Меню управления принтерами ===
1. Выбрать принтер
2. Выбрать тип печати
3. Напечатать текст
0. Выход
Выберите действие: 3
Введите текст для печати: печатный текст

Принтер 'Офисный HP' готов к печати:
[Лазерная печать] печатный текст

```

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