Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Факультет электронно-информационных систем

Кафедра ИИТ

Лабораторная работа №3

По дисциплине «Современные платформы программирования»

Выполнила:

Студентка 3 курса

Группы ПО-11

Лозицкая М.А.

Проверил:

Козик И.Д.

Брест 2025

**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 (порождающий паттерн)**Вариант 3

Проект «Бургер-закусочная». Реализовать возможность формирования заказа из определенных позиций (тип бургера (веганский, куриный и т.д.)), напиток (холодный – пепси, кока-кола и т.д.; горячий – кофе, чай и т.д.), тип упаковки – с собой, на месте. Должна формироваться итоговая стоимость заказа.

Код программы:  
from enum import Enum

class BurgerType(Enum):

VEGAN = "Веганский"

CHICKEN = "Куриный"

BEEF = "Говяжий"

CHEESEBURGER = "Чизбургер"

class ColdDrinkType(Enum):

PEPSI = "Пепси"

COCA\_COLA = "Кока-Кола"

SPRITE = "Спрайт"

FANTA = "Фанта"

class HotDrinkType(Enum):

COFFEE = "Кофе"

TEA = "Чай"

HOT\_CHOCOLATE = "Горячий шоколад"

class PackagingType(Enum):

TAKEAWAY = "С собой"

ON\_SITE = "На месте"

class DrinkCategory(Enum):

COLD = "Холодный"

HOT = "Горячий"

NONE = "Без напитка"

class Product:

def \_\_init\_\_(self, name, price):

self.name = name

self.price = price

class Order:

def \_\_init\_\_(self):

self.burger = None

self.drink = None

self.packaging = None

def \_\_str\_\_(self):

items = []

if self.burger:

items.append(f"Бургер: {self.burger.name} - {self.burger.price} руб.")

if self.drink:

items.append(f"Напиток: {self.drink.name} - {self.drink.price} руб.")

if self.packaging:

items.append(f"Упаковка: {self.packaging.name}")

total = sum(item.price for item in [self.burger, self.drink] if item)

items.append(f"Итого: {total} руб.")

return "\n".join(items)

class OrderBuilder:

def \_\_init\_\_(self):

self.order = Order()

def add\_burger(self, burger\_type):

prices = {

BurgerType.VEGAN: 150,

BurgerType.CHICKEN: 180,

BurgerType.BEEF: 200,

BurgerType.CHEESEBURGER: 190

}

self.order.burger = Product(burger\_type.value, prices[burger\_type])

return self

def add\_drink(self, drink):

if drink:

self.order.drink = drink

return self

def set\_packaging(self, packaging\_type):

self.order.packaging = Product(packaging\_type.value, 0)

return self

def build(self):

return self.order

def get\_user\_choice(options, title):

print(f"\n{title}:")

for i, option in enumerate(options, 1):

print(f"{i}. {option.value}")

while True:

try:

choice = int(input("Ваш выбор: "))

if 1 <= choice <= len(options):

return options[choice - 1]

print("Неверный выбор, попробуйте снова")

except ValueError:

print("Пожалуйста, введите число")

def select\_drink():

drink\_category = get\_user\_choice(

[DrinkCategory.COLD, DrinkCategory.HOT, DrinkCategory.NONE],

"Выберите тип напитка"

)

if drink\_category == DrinkCategory.NONE:

return None

if drink\_category == DrinkCategory.COLD:

drinks = list(ColdDrinkType)

else:

drinks = list(HotDrinkType)

selected = get\_user\_choice(drinks, "Выберите напиток")

prices = {

ColdDrinkType.PEPSI: 80,

ColdDrinkType.COCA\_COLA: 90,

ColdDrinkType.SPRITE: 80,

ColdDrinkType.FANTA: 80,

HotDrinkType.COFFEE: 120,

HotDrinkType.TEA: 60,

HotDrinkType.HOT\_CHOCOLATE: 110

}

return Product(selected.value, prices[selected])

def create\_order():

builder = OrderBuilder()

# Выбор бургера

burger = get\_user\_choice(list(BurgerType), "Выберите бургер")

builder.add\_burger(burger)

# Выбор напитка

drink = select\_drink()

builder.add\_drink(drink)

# Выбор упаковки

packaging = get\_user\_choice(list(PackagingType), "Выберите упаковку")

builder.set\_packaging(packaging)

return builder.build()

def main():

while True:

print("\n=== Создание нового заказа ===")

order = create\_order()

print("\nВаш заказ:")

print(order)

# Цикл проверки корректного ответа

while True:

choice = input("\nСоздать еще один заказ? (да/нет): ").lower()

if choice in ('да', 'нет'):

break

print("Ошибка: пожалуйста, введите 'да' или 'нет'")

if choice != 'да':

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

Результат работы программы:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Задание 2 (структурный паттерн)**

Вариант 3

Проект «IT-компания». В проекте должен быть реализован класс «Сотрудник» c субординацией (т.е. должна быть возможность определения кому подчиняется сотрудник и кто находится в его подчинении). Для каждого сотрудника помимо сведений о субординации хранятся другие данные (ФИО, отдел, должность, зарплата). Предусмотреть возможность удаления и добавления сотрудника.

Код программы:  
class Employee:

def \_\_init\_\_(self, name: str, position: str, salary: float, department: str = None):

self.name = name

self.position = position

self.salary = salary

self.department = department

self.\_supervisor = None

self.subordinates = []

@property

def supervisor(self):

return self.\_supervisor

@supervisor.setter

def supervisor(self, value):

if self.\_supervisor == value:

return

if self.\_supervisor:

self.\_supervisor.subordinates.remove(self)

self.\_supervisor = value

if value:

value.subordinates.append(self)

def get\_hierarchy(self, level=0):

result = "\t" \* level + f"{self.name} ({self.position}, {self.salary}₽)\n"

for sub in sorted(self.subordinates, key=lambda x: x.name):

result += sub.get\_hierarchy(level + 1)

return result

def \_\_repr\_\_(self):

return f"{self.name} ({self.position})"

class Company:

def \_\_init\_\_(self):

self.employees = []

self.departments = {}

self.\_ceo = None

@property

def ceo(self):

return self.\_ceo

@ceo.setter

def ceo(self, value):

if self.\_ceo:

for sub in self.\_ceo.subordinates[:]:

sub.supervisor = value

self.\_ceo = value

def add\_employee(self, employee):

if employee not in self.employees:

self.employees.append(employee)

if employee.department and employee.department != "Без отдела":

if employee.department not in self.departments:

self.departments[employee.department] = []

if employee not in self.departments[employee.department]:

self.departments[employee.department].append(employee)

def remove\_employee(self, employee):

if employee not in self.employees:

return

if employee == self.ceo:

print("\nНельзя удалить директора без замены!")

self.create\_new\_ceo()

return

for sub in employee.subordinates[:]:

sub.supervisor = employee.supervisor

self.employees.remove(employee)

if employee.department and employee.department in self.departments:

self.departments[employee.department].remove(employee)

if not self.departments[employee.department]:

del self.departments[employee.department]

def create\_new\_ceo(self):

print("\nСоздание нового директора:")

name = input\_with\_validation("Введите ФИО нового директора: ")

salary = input\_with\_validation("Введите зарплату: ", float)

new\_ceo = Employee(name, "Директор", salary)

if self.ceo:

for sub in self.ceo.subordinates[:]:

sub.supervisor = new\_ceo

self.ceo = new\_ceo

self.add\_employee(new\_ceo)

print("Новый директор назначен!")

def print\_structure(self):

if not self.ceo:

print("Нет директора в компании!")

return

print("\nСтруктура компании:")

print(f"{self.ceo.name} (Директор, {self.ceo.salary}₽)")

no\_dept = [e for e in self.employees if not e.department and e != self.ceo]

for emp in sorted(no\_dept, key=lambda x: x.name):

print(f"\t{emp.name} ({emp.position}, {emp.salary}₽)")

for dept in sorted(self.departments.keys()):

print(f"\nОтдел {dept}:")

head = next((e for e in self.departments[dept] if e.supervisor == self.ceo), None)

if head:

print(head.get\_hierarchy(level=1))

def input\_with\_validation(prompt, type\_func=str, allowed=None, back=False):

while True:

choice = input(prompt)

if back and choice in ('0', '-1'):

return choice

try:

converted = type\_func(choice)

if allowed is not None and converted not in allowed:

raise ValueError

return converted

except ValueError:

print("Некорректный ввод! Попробуйте снова.")

def create\_employee(company):

name = input\_with\_validation("Введите ФИО сотрудника: ")

position = input\_with\_validation("Введите должность: ")

salary = input\_with\_validation("Введите зарплату: ", float)

print("\nВыберите отдел:")

print("1. Без отдела")

print("2. Создать новый отдел")

existing\_departments = sorted(company.departments.keys())

for i, dept in enumerate(existing\_departments, 3):

print(f"{i}. {dept}")

print("0. Назад\n-1. В меню")

max\_choice = len(existing\_departments) + 2

choice = input\_with\_validation("Ваш выбор: ", int, range(0, max\_choice + 2), back=True)

if str(choice) in ('0', '-1'):

return

department = None

if choice == 1:

department = None

elif choice == 2:

dept\_name = input\_with\_validation("Введите название нового отдела: ")

head\_name = input\_with\_validation("Введите ФИО начальника отдела: ")

head\_salary = input\_with\_validation("Введите зарплату начальника: ", float)

head = Employee(head\_name, "Начальник отдела", head\_salary, dept\_name)

head.supervisor = company.ceo

company.add\_employee(head)

department = dept\_name

else:

department = existing\_departments[choice - 3]

employee = Employee(name, position, salary, department)

possible\_supervisors = []

if department:

dept\_employees = company.departments.get(department, [])

head = next((e for e in dept\_employees if e.position == "Начальник отдела"), company.ceo)

possible\_supervisors = [head] + [e for e in dept\_employees if e != head]

else:

possible\_supervisors = [company.ceo] + [e for e in company.employees if not e.department and e != company.ceo]

print("\nВыберите руководителя:")

for i, sup in enumerate(possible\_supervisors, 1):

print(f"{i}. {sup.name} ({sup.position})")

print("0. Назад\n-1. В меню")

max\_sup\_choice = len(possible\_supervisors)

choice\_sup = input\_with\_validation("Ваш выбор: ", int, range(0, max\_sup\_choice + 2), back=True)

if str(choice\_sup) in ('0', '-1'):

return

employee.supervisor = possible\_supervisors[choice\_sup - 1]

company.add\_employee(employee)

print(f"Сотрудник {name} успешно добавлен!")

def delete\_employee(company):

if not company.employees:

print("В компании нет сотрудников!")

return

print("\nСписок сотрудников:")

for i, emp in enumerate(company.employees, 1):

print(f"{i}. {emp.name} ({emp.position})")

print("0. Назад\n-1. В меню")

choice = input\_with\_validation("Выберите сотрудника для удаления: ", int, range(0, len(company.employees) + 2),

back=True)

if str(choice) in ('0', '-1'):

return

employee = company.employees[choice - 1]

if employee.position == "Начальник отдела" and employee.department:

department = employee.department

dept\_employees = [e for e in company.employees if e.department == department]

print(f"\nНеобходимо выбрать нового начальника для отдела {department}:")

print("1. Создать нового сотрудника")

print("2. Выбрать из существующих")

choice = input\_with\_validation("Ваш выбор: ", int, [1, 2])

if choice == 1:

name = input\_with\_validation("Введите ФИО нового начальника: ")

salary = input\_with\_validation("Введите зарплату: ", float)

new\_head = Employee(name, "Начальник отдела", salary, department)

new\_head.supervisor = company.ceo

company.add\_employee(new\_head)

else:

candidates = [e for e in dept\_employees if e != employee]

if not candidates:

print("Нет кандидатов для назначения!")

return

print("Список сотрудников отдела:")

for i, e in enumerate(candidates, 1):

print(f"{i}. {e.name} ({e.position})")

choice\_idx = input\_with\_validation("Выберите сотрудника: ", int, range(1, len(candidates) + 1))

new\_head = candidates[choice\_idx - 1]

new\_head.position = "Начальник отдела"

new\_head.supervisor = company.ceo

for e in dept\_employees:

if e != employee and e.supervisor == employee:

e.supervisor = new\_head

company.departments[department].remove(employee)

company.departments[department].append(new\_head)

company.remove\_employee(employee)

print(f"Сотрудник {employee.name} удален!")

def main\_menu():

company = Company()

company.create\_new\_ceo()

while True:

print("\nГлавное меню:")

print("1. Добавить сотрудника")

print("2. Удалить сотрудника")

print("3. Показать структуру")

print("4. Выход")

choice = input\_with\_validation("Выберите действие: ", int, [1, 2, 3, 4])

if choice == 1:

create\_employee(company)

elif choice == 2:

delete\_employee(company)

elif choice == 3:

company.print\_structure()

elif choice == 4:

break

if \_\_name\_\_ == "\_\_main\_\_":

main\_menu()

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.Результат работы программы:

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

**Задание 3 (поведенческий паттерн)**

Вариант 3

Проект «Расчет зарплаты». Для задания, указанного во втором пункте («IT компания») реализовать расчет зарплаты с выводом полного отчета. Порядок вывода сотрудников в отчете – по старшинству для каждого отдела.  
К коду предыдущего задания добавлено:

class SalaryCalculatorVisitor:

def \_\_init\_\_(self):

self.salaries = {}

self.total\_company = {'sum': 0.0, 'count': 0}

self.no\_dept = {'sum': 0.0, 'count': 0}

self.departments = defaultdict(lambda: {'sum': 0.0, 'count': 0})

self.positions = defaultdict(lambda: {'sum': 0.0, 'count': 0})

def visit(self, employee):

calculated\_salary = employee.salary

self.salaries[employee] = calculated\_salary

self.total\_company['sum'] += calculated\_salary

self.total\_company['count'] += 1

if not employee.department and employee.position != 'Директор':

self.no\_dept['sum'] += calculated\_salary

self.no\_dept['count'] += 1

if employee.department:

self.departments[employee.department]['sum'] += calculated\_salary

self.departments[employee.department]['count'] += 1

self.positions[employee.position]['sum'] += calculated\_salary

self.positions[employee.position]['count'] += 1

def get\_salary(self, employee):

return self.salaries.get(employee, 0)

def get\_company\_stats(self):

return self.total\_company

def get\_no\_dept\_stats(self):

return self.no\_dept

def get\_department\_stats(self):

return self.departments

def get\_position\_stats(self):

return self.positions

в класс Employee:

def accept(self, visitor):

visitor.visit(self)

в класс Company:

def generate\_salary\_report(self):

visitor = SalaryCalculatorVisitor()

departments = self.traverse\_hierarchy()

if 'Без отдела' in departments:

departments['Без отдела'] = [

emp for emp in departments['Без отдела']

if emp.position != 'Директор'

]

for dept in departments:

for emp in departments[dept]:

emp.accept(visitor)

report = [

"\n══════════════════════════════════",

"Полный отчет о зарплатах компании",

"══════════════════════════════════"

]

# Блок руководства

report.append("\n▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬")

report.append("Руководство компании:")

report.append(f" {self.ceo.name} ({self.ceo.position}): {format\_money(self.ceo.salary)}")

# Иерархия с зарплатами

report.append("\n▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬")

report.append("Детальная структура:")

if 'Без отдела' in departments and departments['Без отдела']:

report.append("\nСотрудники без отдела:")

for emp in departments['Без отдела']:

report.append(f" → {emp.name} ({emp.position}): {format\_money(emp.salary)}")

sorted\_depts = sorted([dept for dept in departments if dept != 'Без отдела'])

for dept in sorted\_depts:

report.append(f"\nОтдел {dept}:")

for emp in departments[dept]:

line = f" → {emp.name} ({emp.position}): {format\_money(emp.salary)}"

if emp.position == 'Начальник отдела':

line += f" [Подчиненных: {len(emp.subordinates)}]"

report.append(line)

# Агрегированная статистика

report.append("\n▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬")

report.append("Аналитическая сводка:")

total = visitor.get\_company\_stats()

report.append(

f"\nПо всему предприятию: "

f"Общий фонд - {format\_money(total['sum'])}, "

f"Средняя зарплата - {format\_money(total['sum'] / total['count']) if total['count'] else 0}"

)

no\_dept = visitor.get\_no\_dept\_stats()

if no\_dept['count'] > 0:

report.append(

f"\nСотрудники без отдела: "

f"Суммарно - {format\_money(no\_dept['sum'])}, "

f"Средняя - {format\_money(no\_dept['sum'] / no\_dept['count'])}"

)

report.append("\nПо отделам:")

for dept, stats in visitor.get\_department\_stats().items():

report.append(

f" {dept}: "

f"Сумма - {format\_money(stats['sum'])}, "

f"Средняя - {format\_money(stats['sum'] / stats['count']) if stats['count'] else 0}, "

f"Сотрудников - {stats['count']}"

)

report.append("\nПо должностям:")

for position, stats in visitor.get\_position\_stats().items():

report.append(

f" {position}: "

f"Сумма - {format\_money(stats['sum'])}, "

f"Средняя - {format\_money(stats['sum'] / stats['count']) if stats['count'] else 0}, "

f"Сотрудников - {stats['count']}"

)

print("\n".join(report))

def traverse\_hierarchy(self):

visited = set()

departments = defaultdict(list)

if not self.ceo:

return departments

queue = deque([self.ceo])

visited.add(self.ceo)

if self.ceo.department:

departments[self.ceo.department].append(self.ceo)

while queue:

current = queue.popleft()

for sub in sorted(current.subordinates, key=lambda x: x.name):

if sub not in visited:

visited.add(sub)

queue.append(sub)

dept\_sub = sub.department or 'Без отдела'

departments[dept\_sub].append(sub)

for emp in self.employees:

if emp not in visited:

dept\_emp = emp.department or 'Без отдела'

departments[dept\_emp].append(emp)

return departmentsс

Результат работы программы:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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