

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ”
КАФЕДРА ИИТ

ОТЧЁТ
по лабораторной работе №6

Выполнил:
студент 3 курса
группы ПО-9
Мельничук В.М.
Проверил:
Крощенко А.А.

Брест 2024

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

Номер зачетной книжки 210663

Вариант 3

Задание 1

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.

Выходные данные:

```
C:\Users\vladi\.jdk\openjdk-21.0.2\bin\java.exe
Order details:
Burger: Vegan
Patties: 2
Toppings: []
Burger Price: $20.0
Drink type: Cold
Drink: Cola
Drink Price: $10.0
Packaging: ToGo
-----
Total Price: $30.0

Burger: Meat
Patties: 3
Toppings: [onion, tomato]
Burger Price: $32.0
Drink: N/A
Drink Price: $0.0
Packaging: InPlace
-----
Total Price: $32.0
```

Код программы:

```
import java.util.ArrayList;
import java.util.List;

enum drinkType{
    Cold,
    Hot
}

enum burgerType{
    Vegan,
    Meat
}

enum packingType{
    ToGo,
    InPlace
}

class Burger {
    private String type; // Тип
    private int patties = 1; // Кол-во котлет
    private List<String> toppings; // Топпинги
    private double price; // Цена

    public Burger(String type, double price, int patties) {
        this.type = type;
        this.price = price;
        this.patties = patties;
        this.toppings = new ArrayList<>();
    }
}
```

```

    public void addTopping(String topping) {
        this.toppings.add(topping);
    }

    public String getType() {
        return type;
    }

    public int getPatties() {
        return patties;
    }

    public List<String> getToppings() {
        return toppings;
    }

    public double getPrice() {
        return price;
    }
}

class Drink {
    private String type; // Тип
    private String name; // Название
    private double price; // Цена

    public Drink(String type, String name, double price) {
        this.type = type;
        this.name = name;
        this.price = price;
    }

    public String getType() {
        return type;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

class Packaging {
    private String type;

    public Packaging(String type) {
        this.type = type;
    }

    public String getType() {
        return type;
    }
}

class Order {
    private String orderDetails;

    public Order(String orderDetails) {
        this.orderDetails = orderDetails;
    }

    public String getOrderDetails() {
        return orderDetails;
    }
}

class OrderBuilder {
    private Burger burger;
    private Drink drink;
    private Packaging packaging;

    public OrderBuilder() {
    }

    public OrderBuilder addBurger(burgerType type, double price, int patty) {
        this.burger = new Burger(type.toString(), price, patty);
        return this;
    }
}

```

```

    public OrderBuilder addTopping(String topping) {
        this.burger.addTopping(topping);
        return this;
    }

    public OrderBuilder addDrink(drinkType type, String name, double price) {
        this.drink = new Drink(type.toString(), name, price);
        return this;
    }

    public OrderBuilder addPackaging(packagingType type) {
        this.packaging = new Packaging(type.toString());
        return this;
    }

    public Order getOrderDetails() {
        double burgerPrice = (burger != null) ? burger.getPrice() : 0.0;
        double drinkPrice = (drink != null) ? drink.getPrice() : 0.0;
        double totalPrice = burgerPrice + drinkPrice;
        StringBuilder details = new StringBuilder();
        if (burger != null) {
            details.append("Burger: ").append(burger.getType()).append("\n");
            details.append("Patties: ")
                .append(burger.getPatties()).append("\n");
            details.append("Toppings: ")
                .append(burger.getToppings()).append("\n");
            details.append("Burger Price: $")
                .append(burger.getPrice()).append("\n");
        } else {
            details.append("Burger: ").append("N/A").append("\n");
            details.append("Burger Price: $").append("0.0").append("\n");
        }
        if (drink != null) {
            details.append("Drink type: ").append(drink.getType()).append("\n");
            details.append("Drink: ").append(drink.getName()).append("\n");
            details.append("Drink Price: $").append(drinkPrice).append("\n");
        } else {
            details.append("Drink: ").append("N/A").append("\n");
            details.append("Drink Price: $").append("0.0").append("\n");
        }
        details.append("Packaging: ").append(packaging.getType()).append("\n");
        details.append("-----\n");
        details.append("Total Price: $").append(totalPrice).append("\n");
        return new Order(details.toString());
    }
}

public class task1 {
    public static void main(String[] args) {
        // Создаем заказ
        OrderBuilder orderBuilder = new OrderBuilder();
        Order order = orderBuilder
            .addBurger(burgerType.Vegan, 20, 2)
            .addDrink(drinkType.Cold, "Cola", 10)
            .addPackaging(packagingType.ToGo)
            .getOrderDetails();

        OrderBuilder orderBuilder1 = new OrderBuilder();
        Order order1 = orderBuilder1
            .addBurger(burgerType.Meat, 32, 3)
            .addTopping("onion, tomato")
            .addPackaging(packagingType.InPlace)
            .getOrderDetails();

        System.out.println("Order details:");
        System.out.println(order.getOrderDetails());
        System.out.println(order1.getOrderDetails());
    }
}

```

Задание 2

Проект «ИТ-компания». В проекте должен быть реализован класс «Сотрудник» с 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 добавле-

ния сотрудника.

Выходные данные:

```
C:\Users\vladi\.jdk\openjdk-21.0.2\bin\java.exe "-javaagent:E:\So
Manager: John (CEO)
Manager: Alice (Manager)
Name: Tom, Department: Sales, Position: Salesman, Salary: 5000.0

Manager: Bob (Manager)
Name: Jerry, Department: IT, Position: Developer, Salary: 6000.0
```

Код программы

```
import java.util.ArrayList;
import java.util.List;

interface EmployeeComponent {
    void add(EmployeeComponent employee);
    void remove(EmployeeComponent employee);
    List<EmployeeComponent> getSubordinates();
    String getInfo();
}

class Employee implements EmployeeComponent {
    private String name;
    private String department;
    private String position;
    private double salary;

    public Employee(String name, String department, String position, double salary) {
        this.name = name;
        this.department = department;
        this.position = position;
        this.salary = salary;
    }

    public void add(EmployeeComponent employee) {
    }

    public void remove(EmployeeComponent employee) {
    }

    public List<EmployeeComponent> getSubordinates() {
        return new ArrayList<>();
    }

    public String getInfo() {
        return "Name: " + name + ", Department: " + department + ", Position: " + position + ", Salary: "
+ salary;
    }
}

class Manager implements EmployeeComponent {
    private String name;
    private List<EmployeeComponent> subordinates;

    public Manager(String name) {
        this.name = name;
        this.subordinates = new ArrayList<>();
    }

    public void add(EmployeeComponent employee) {
        subordinates.add(employee);
    }

    public void remove(EmployeeComponent employee) {
        subordinates.remove(employee);
    }

    public List<EmployeeComponent> getSubordinates() {
        return subordinates;
    }
}
```

```

        public String getInfo() {
            StringBuilder info = new StringBuilder("Manager: " + name + "\n");
            for (EmployeeComponent employee : subordinates) {
                info.append(employee.getInfo()).append("\n");
            }
            return info.toString();
        }
    }
}

public class task2 {
    public static void main(String[] args) {
        EmployeeComponent ceo = new Manager("John (CEO)");
        EmployeeComponent manager1 = new Manager("Alice (Manager)");
        EmployeeComponent manager2 = new Manager("Bob (Manager)");
        EmployeeComponent employee1 = new Employee("Tom", "Sales", "Salesman", 5000);
        EmployeeComponent employee2 = new Employee("Jerry", "IT", "Developer", 6000);

        manager1.add(employee1);
        manager2.add(employee2);
        ceo.add(manager1);
        ceo.add(manager2);

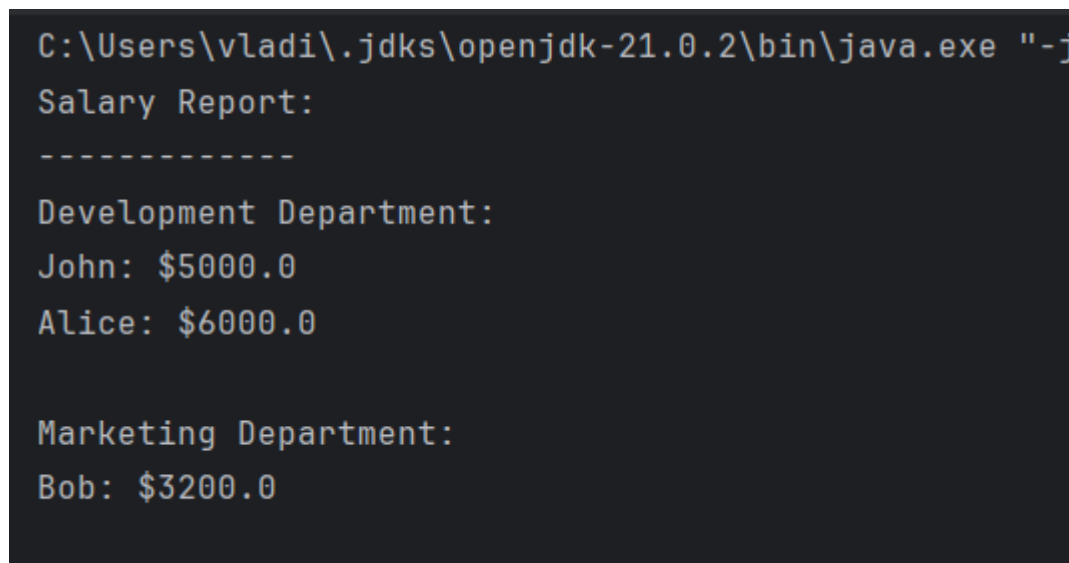
        System.out.println(ceo.getInfo());
    }
}

```

Задание 3

Проект «Расчет зарплаты». Для задания, указанного во втором пункте («ИТ-компания») реализовать расчет зарплаты с выводом полного отчета. Порядок вывода сотрудников в отчете - по старшинству для каждого отдела.

Выходные данные:



```

C:\Users\vladi\.jdk\openjdk-21.0.2\bin\java.exe "-j
Salary Report:
-----
Development Department:
John: $5000.0
Alice: $6000.0

Marketing Department:
Bob: $3200.0

```

Код программы:

```

import java.util.ArrayList;
import java.util.List;

interface SalaryCalculationStrategy {
    double calculateSalary(Employee1 employee);
}

class StandardSalaryStrategy implements SalaryCalculationStrategy {
    @Override
    public double calculateSalary(Employee1 employee) {
        return employee.getBaseSalary();
    }
}

```

```

class HourlySalaryStrategy implements SalaryCalculationStrategy {
    private double hourlyRate;

    public HourlySalaryStrategy(double hourlyRate) {
        this.hourlyRate = hourlyRate;
    }

    @Override
    public double calculateSalary(Employee1 employee) {
        return hourlyRate * employee.getHoursWorked();
    }
}

class SalaryCalculator {
    private SalaryCalculationStrategy calculationStrategy;

    public SalaryCalculator(SalaryCalculationStrategy calculationStrategy) {
        this.calculationStrategy = calculationStrategy;
    }

    public double calculate(Employee1 employee) {
        return calculationStrategy.calculateSalary(employee);
    }
}

class Employee1 {
    private String name;
    private double baseSalary;
    private double hoursWorked;

    public Employee1(String name, double baseSalary) {
        this.name = name;
        this.baseSalary = baseSalary;
    }

    public String getName() {
        return name;
    }

    public double getBaseSalary() {
        return baseSalary;
    }

    public double getHoursWorked() {
        return hoursWorked;
    }

    public void setHoursWorked(double hoursWorked) {
        this.hoursWorked = hoursWorked;
    }
}

class Department {
    private String name;
    private List<Employee1> employees;

    public Department(String name) {
        this.name = name;
        this.employees = new ArrayList<>();
    }

    public void addEmployee(Employee1 employee) {
        employees.add(employee);
    }

    public List<Employee1> getEmployees() {
        return employees;
    }
}

public class task3 {
    public static void main(String[] args) {
        Department development = new Department("Development");
        Department marketing = new Department("Marketing");

        development.addEmployee(new Employee1("John", 5000));
        development.addEmployee(new Employee1("Alice", 6000));
        marketing.addEmployee(new Employee1("Bob", 4500));
    }
}

```

```

SalaryCalculationStrategy standardStrategy = new StandardSalaryStrategy();
SalaryCalculationStrategy hourlyStrategy = new HourlySalaryStrategy(20);

SalaryCalculator standardCalculator = new SalaryCalculator(standardStrategy);
SalaryCalculator hourlyCalculator = new SalaryCalculator(hourlyStrategy);

System.out.println("Salary Report:");
System.out.println("-----");

System.out.println("Development Department:");
for (Employee1 employee : development.getEmployees()) {
    System.out.println(employee.getName() + ": $" + standardCalculator.calculate(employee));
}

System.out.println("\nMarketing Department:");
for (Employee1 employee : marketing.getEmployees()) {
    employee.setHoursWorked(160);
    System.out.println(employee.getName() + ": $" + hourlyCalculator.calculate(employee));
}
}
}

```