**Министерство образования Республики Беларусь Учреждение Образования**

**«Брестский Государственный Технический Университет» Кафедра ИИТ**

**Лабораторная работа №6**

**По дисциплине СПП за 5 семестр Тема:** «Java»

**Выполнил:**

Студент 3-го курса Группы ПО-5

Брич М.Н.

**Проверил:**

Крощенко А.А.

**Брест 2021**

Лабораторная работа №6

# Цель работы:

приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Java

# Вариант: 3 Согласно зачетной книжке Задания и выполненные решения:

* Прочитать задания, взятые из каждой группы.
* Определить паттерн проектирования, который может использоваться при реализации задания. Пояснить свой выбор.
* Реализовать фрагмент программной системы, используя выбранный паттерн. Реализовать все необходимые дополнительные классы.

Варианты работ определяются по последней цифре в зачетной книжке (190363).

***Первая группа заданий***

***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.***

Для решения данного задания был использован **паттерн Builder (Строитель),** который позволяет создавать сложные объекты пошагово. Строитель даёт возможность

использовать один и тот же код строительства для получения разных представлений объектов.

# Код:

package lab3\_1;

import lab3\_1.enums.LocationType;

import lab3\_1.enums.BurgerType;

import lab3\_1.enums.DrinkType;

import lab3\_1.enums.SideType;

public class Lab6\_1 {

public static void main(String[] args) {

Order order = new Builder("Пищик А.В.")

.setBurger(BurgerType.BURGER\_WITH\_SALAMI)

.setDrink(DrinkType.TEA)

.setSide(SideType.MOZZARELLA\_STICKS)

.setLocation(LocationType.IN\_RESTAURANT)

.build();

Order order2 = new Builder("Брич М.Н.")

.setBurger(BurgerType.BEEF\_BURGER)

.setDrink(DrinkType.COFFEE)

.setSide(SideType.FRENCH\_FRIES)

.setLocation(LocationType.DELIVERY)

.build();

Order order3 = new Builder("Нерода А.А.")

.setBurger(BurgerType.CHICKEN\_BURGER)

.setDrink(DrinkType.COCA\_COLA)

.setSide(SideType.CHICKEN\_NUGGETS)

.setLocation(LocationType.TAKEOUT)

.build();

order.show();

order2.show();

order3.show();

}

}

package lab3\_1;

import lab3\_1.enums.BurgerType;

import lab3\_1.enums.DrinkType;

import lab3\_1.enums.LocationType;

import lab3\_1.enums.SideType;

public class Builder {

private final Order order;

public Builder(String customer) {

order = new Order(customer);

order.burger = null;

order.drink = null;

order.side = null;

order.location = null;

}

private Builder(

String customer,

BurgerType burger,

DrinkType drink,

SideType side,

LocationType location)

{

order = new Order(customer);

order.burger = burger;

order.drink = drink;

order.side = side;

order.location = location;

}

public Builder setBurger(BurgerType burger) {

return new Builder(order.customer, burger, order.drink, order.side, order.location);

}

public Builder setDrink(DrinkType drink) {

return new Builder(order.customer, order.burger, drink, order.side, order.location);

}

public Builder setSide(SideType side) {

return new Builder(order.customer, order.burger, order.drink, side, order.location);

}

public Builder setLocation(LocationType location) {

return new Builder(order.customer, order.burger, order.drink, order.side, location);

}

public Order build() {

return order;

}

}

package lab3\_1;

import lab3\_1.enums.LocationType;

import lab3\_1.enums.BurgerType;

import lab3\_1.enums.DrinkType;

import lab3\_1.enums.SideType;

public class Order {

public final String customer;

public BurgerType burger;

public DrinkType drink;

public SideType side;

public LocationType location;

public Order(String customer) {

this.customer = customer;

}

public void show()

{

System.out.printf(

"Описание заказа:\n Заказчик: %s\n Бургер: %s\n Напиток: %s\n Дополнение: %s\n Упаковка: %s%n",

customer,

burger.name(),

drink.name(),

side.name(),

location.name()

);

System.out.println();

}

}

package lab3\_1.enums;

public enum BurgerType {

BEEF\_BURGER,

CHICKEN\_BURGER,

BURGER\_WITH\_SALAMI

}

package lab3\_1.enums;

public enum DrinkType {

COCA\_COLA,

TEA,

COFFEE

}

package lab3\_1.enums;

public enum LocationType {

IN\_RESTAURANT,

TAKEOUT,

DELIVERY

}

package lab3\_1.enums;

public enum SideType {

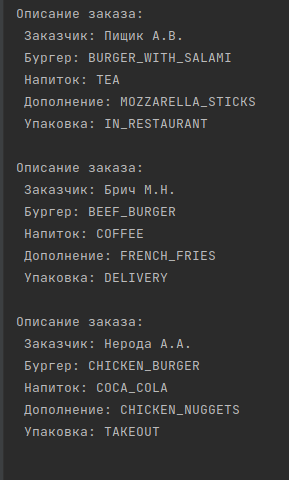
FRENCH\_FRIES,

CHICKEN\_NUGGETS,

MOZZARELLA\_STICKS

}

# Результат:

****

Выполнение происходит согласно поставленной задачи. Работает корректно!

***Вторая группа заданий***

***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.

Для решения данного задания был использован **паттерн Composite (Компоновщик),** который позволяет сгруппировать объекты в древовидную структуру, а затем работать с ними так, как будто это единичный объект.

# Код:

**Main.java**

package lab6\_2;

public class Lab6\_2 {

public static void main(String[] args) {

Employee ceo = new Employee("Raman Harhun", 5, WorkDepartment.LEAD, WorkField.MANAGEMENT);

ITCompany company = new ITCompany("Harbros Solutions", ceo);

Employee manager = new Employee("Tsimafei Harhun", 4, WorkDepartment.LEAD, WorkField.MANAGEMENT);

ITCompany company2 = new ITCompany("EPAM", manager);

Employee worker = new Employee("Ilya Kulinkovich", 10, WorkDepartment.PROJECTS, WorkField.DEVELOPMENT);

Employee worker2 = new Employee("Yana Danilyuk", 8, WorkDepartment.PROJECTS, WorkField.DESIGN);

ceo.addSubordinate(manager);

manager.addSubordinate(worker);

worker.addSubordinate(worker2);

show(ceo);

show(manager);

show(worker2);

}

private static void show(Employee obj) {

System.out.println(obj);

if (!obj.getSubordinates().isEmpty()) {

System.out.println("Его подчиненные:");

showSubord(obj);

}

System.out.println();

}

private static void showSubord(Employee obj) {

if (!obj.getSubordinates().isEmpty()) {

System.out.println(obj.getSubordinates());

for (Employee employee : obj.getSubordinates()) {

if (!employee.getSubordinates().isEmpty()) {

System.out.println(employee.getSubordinates());

showSubord(employee);

}

}

}

}

}

package lab6\_2;

import java.util.ArrayList;

import java.util.Iterator;

enum WorkDepartment {

LEAD,

RESEARCH,

PROJECTS,

MARKETING

}

enum WorkField {

DESIGN,

DEVELOPMENT,

MANAGEMENT

}

class Employee implements Iterable<Employee> {

public static double MONEY\_PER\_PROJECT = 200;

private String name;

private int numProjects;

private WorkDepartment department;

private WorkField field;

private ArrayList<Employee> subordinates = new ArrayList<>();

public Employee(String name, int numProjects, WorkDepartment department, WorkField field) {

this.name = name;

this.numProjects = numProjects;

this.department = department;

this.field = field;

}

/\* helper methods \*/

public void addSubordinate(Employee employee) {

subordinates.add(employee);

}

public void removeSubordinate(Employee employee) {

subordinates.remove(employee);

employee.removeAllSubordinates();

}

public void removeAllSubordinates() {

for (Employee e: subordinates) {

e.removeAllSubordinates();

e.subordinates.clear();

}

subordinates.clear();

}

public void logSalary() {

System.out.printf("%s%s его зарплата: %f$\n", " ", name, MONEY\_PER\_PROJECT \* numProjects);

}

/\* java.lang.Object \*/

@Override

public String toString() {

return String.format(

"<Employee имя=\"%s\" назв\_проекта=\"%d\" отдел=\"%s\" сфера=\"%s\" подчинение=<arrayList of \"%d\" elements>>",

name, numProjects, department.name(), field.name(), subordinates.size()

);

}

/\* codegen \*/

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getNumProjects() {

return numProjects;

}

public void setNumProjects(int numProjects) {

this.numProjects = numProjects;

}

public WorkDepartment getDepartment() {

return department;

}

public void setDepartment(WorkDepartment department) {

this.department = department;

}

public WorkField getField() {

return field;

}

public void setField(WorkField field) {

this.field = field;

}

public ArrayList<Employee> getSubordinates() {

return subordinates;

}

/\* Iterable \*/

@Override

public Iterator<Employee> iterator() {

return new EmployeeIterator(subordinates);

}

}

# EmployeeIterator.java

package lab6\_2;

import java.util.Iterator;

import java.util.List;

public class EmployeeIterator implements Iterator<Employee> {

private List<Employee> files;

private int position;

public EmployeeIterator(List<Employee> files) {

this.files = files;

position = 0;

}

@Override

public boolean hasNext() {

return position < files.size();

}

@Override

public Employee next() {

return files.get(position++);

}

}

**ITCompany.java**

package lab6\_2;

import java.util.Iterator;

class ITCompany {

private String name;

private Employee ceo;

public ITCompany(String name, Employee ceo) {

this.name = name;

this.ceo = ceo;

}

/\* helper methods \*/

private void logSalaries(Employee employee) {

Iterator<Employee> iterator = employee.iterator();

while (iterator.hasNext()) {

Employee next = iterator.next();

next.logSalary();

logSalaries(next);

}

}

public void logSalaries() {

System.out.println("=============== SALARY LOG BEGIN =============== ");

ceo.logSalary();

logSalaries(ceo);

System.out.println("=============== SALARY LOG END =============== ");

}

/\* codegen \*/

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Employee getCeo() {

return ceo;

}

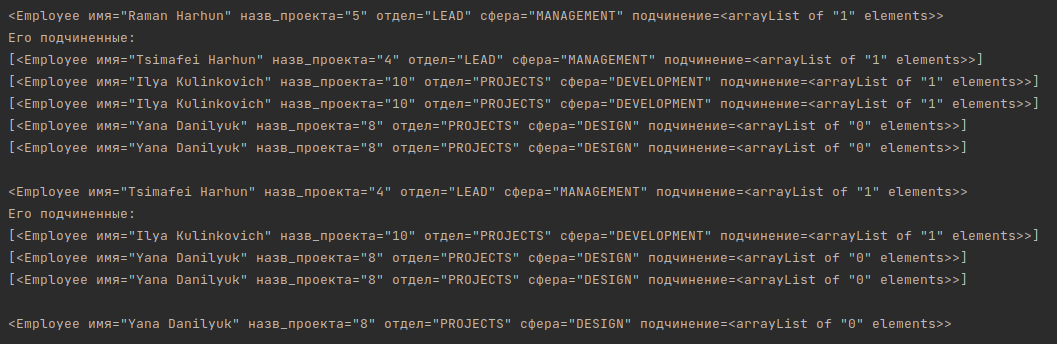
public void setCeo(Employee ceo) {

this.ceo = ceo;

}

}

# Результат:

****

Исходя из полученных результатов, все работает корректно, согласно поставленному заданию.

***Третья группа заданий***

***Проект* «*Расчет зарплаты». Для задания, указанного во втором пункте ((IT-компанияя) реализовать расчет зарплаты с выводом полного отчета.***

## Порядок вывода сотрудников в отчете по старшинству для каждого отдела.

Для решения данного задания был использован **паттерн Iterator (Итератор),** который даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

# Код:

**Main.java**

# package lab6\_3;

# public class Lab6\_3 {

# public static void main(String[] args) {

# Employee ceo = new Employee("Raman Harhun", 5, WorkDepartment.LEAD, WorkField.MANAGEMENT);

# ITCompany company = new ITCompany("Harbros Solutions", ceo);

# Employee manager = new Employee("Tsimafei Harhun", 4, WorkDepartment.LEAD, WorkField.MANAGEMENT);

# ITCompany company2 = new ITCompany("EPAM", manager);

# Employee worker = new Employee("Ilya Kulinkovich", 10, WorkDepartment.PROJECTS, WorkField.DEVELOPMENT);

# Employee worker2 = new Employee("Yana Danilyuk", 8, WorkDepartment.PROJECTS, WorkField.DESIGN);

# ceo.addSubordinate(manager);

# manager.addSubordinate(worker);

# worker.addSubordinate(worker2);

# show(ceo);

# show(manager);

# show(worker2);

# //сдесь вывод суммы

# company.logSalaries();

# System.out.println();

# company2.logSalaries();

# }

# private static void show(Employee obj) {

# System.out.println(obj);

# if (!obj.getSubordinates().isEmpty()) {

# System.out.println("Его подчиненные:");

# showSubord(obj);

# }

# System.out.println();

# }

# private static void showSubord(Employee obj) {

# if (!obj.getSubordinates().isEmpty()) {

# System.out.println(obj.getSubordinates());

# for (Employee employee : obj.getSubordinates()) {

# if (!employee.getSubordinates().isEmpty()) {

# System.out.println(employee.getSubordinates());

# showSubord(employee);

# }

# }

# }

# }

# }

# Employee.java

package lab6\_3;

import java.util.ArrayList;

enum WorkDepartment {

LEAD,

RESEARCH,

PROJECTS,

MARKETING

}

enum WorkField {

DESIGN,

DEVELOPMENT,

MANAGEMENT

}

class Employee implements EmployeeIterator {

public static double MONEY\_PER\_PROJECT = 200;

private String name;

private int numProjects;

private WorkDepartment department;

private WorkField field;

int position = 0;

private ArrayList<Employee> subordinates = new ArrayList<>();

public Employee(String name, int numProjects, WorkDepartment department, WorkField field) {

this.name = name;

this.numProjects = numProjects;

this.department = department;

this.field = field;

}

/\* helper methods \*/

public void addSubordinate(Employee employee) {

subordinates.add(employee);

}

public void removeSubordinate(Employee employee) {

subordinates.remove(employee);

employee.removeAllSubordinates();

}

public void removeAllSubordinates() {

for (Employee e: subordinates) {

e.removeAllSubordinates();

e.subordinates.clear();

}

subordinates.clear();

}

public void logSalary() {

System.out.printf("%s%s его зарплата: %f$\n", " ", name, MONEY\_PER\_PROJECT \* numProjects);

}

/\* java.lang.Object \*/

@Override

public String toString() {

return String.format(

"<Employee имя=\"%s\" назв\_проекта=\"%d\" отдел=\"%s\" сфера=\"%s\" подчинение=<arrayList of \"%d\" elements>>",

name, numProjects, department.name(), field.name(), subordinates.size()

);

}

/\* codegen \*/

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getNumProjects() {

return numProjects;

}

public void setNumProjects(int numProjects) {

this.numProjects = numProjects;

}

public WorkDepartment getDepartment() {

return department;

}

public void setDepartment(WorkDepartment department) {

this.department = department;

}

public WorkField getField() {

return field;

}

public void setField(WorkField field) {

this.field = field;

}

public ArrayList<Employee> getSubordinates() {

return subordinates;

}

@Override

public boolean hasNext() {

return position < subordinates.size();

}

@Override

public Employee next() {

if (!hasNext()) {

return null;

}

Employee employee = subordinates.get(position);

position++;

return employee;

}

@Override

public void reset() {

position = 0;

}

/\* Iterable \*/

}

# EmployeeIterator.java

package lab6\_3;

public interface EmployeeIterator {

boolean hasNext();

Employee next();

void reset();

}

**ITCompany.java** package lab6\_3;

class ITCompany {

private String name;

private Employee ceo;

public ITCompany(String name, Employee ceo) {

this.name = name;

this.ceo = ceo;

}

/\* helper methods \*/

private void logSalaries(Employee employee) {

while (employee.hasNext()) {

Employee next = employee.next();

next.logSalary();

logSalaries(next);

}

employee.reset();

}

public void logSalaries() {

System.out.println("=============== SALARY LOG BEGIN =============== ");

ceo.logSalary();

logSalaries(ceo);

System.out.println("=============== SALARY LOG END =============== ");

}

/\* codegen \*/

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Employee getCeo() {

return ceo;

}

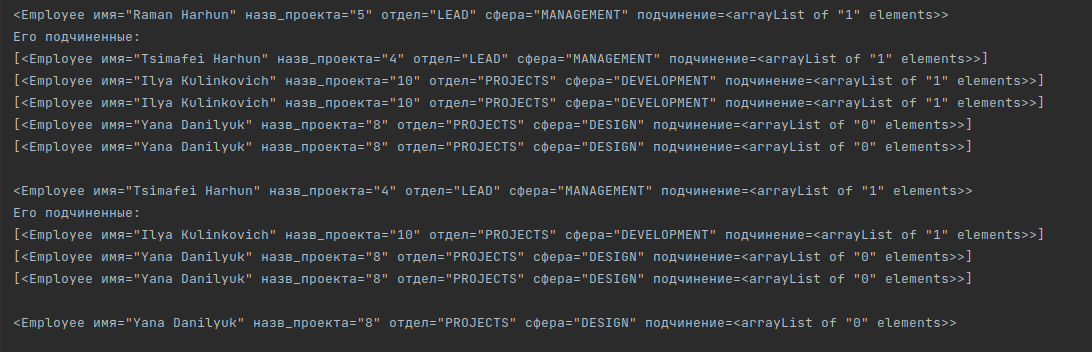
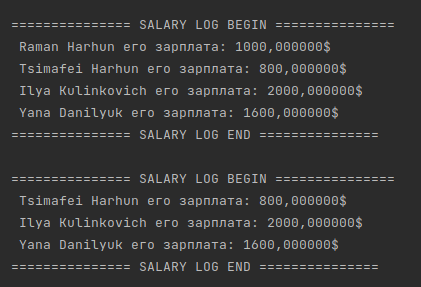
public void setCeo(Employee ceo) {

this.ceo = ceo;

}

}

# Результат:

**  
**

Исходя из полученных результатов, все работает корректно, согласно поставленному заданию.

**Вывод:** приобрел навыки применения паттернов проектирования при решении практических задач с использованием языка Java. В процессе были использованы

следующие паттерны: Builder (Строитель), Composite (Компоновщик), Iterator (Итератор).