|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
|  | |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ № 1 — 24** | |
| **по дисциплине** | |
| **«Шаблоны программных платформ языка Джава»** | |
| Выполнил студент группы **ИКБО-20-22** | ***Шумахер М.Е.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2024г. |  |
| «Зачтено» | « » 2024г. |  |

Москва 2024

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

Цель: Знакомство со встроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

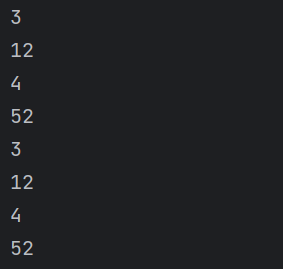
Постановка задачи: Имплементировать интерфейс Consumer, принимающий на вход массив чисел и выводящий в консоль в порядке возрастания.

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac1

package prac1;  
  
import java.util.function.Consumer;  
  
public class Main implements Consumer<int[]> {  
 @Override  
 public void accept(int[] num){  
 for (int n : num){  
 System.*out*.println(n + " ");  
 }  
 }  
 public static void main(String[] args) {  
 int[] num ={3,12,4,52};  
 Main cons = new Main();  
 cons.accept(num);  
 Consumer<int[]> printer = arr -> {  
 for (int n : arr) {  
 System.*out*.println(n + " ");  
 }  
 };  
 printer.accept(num);  
 }  
}

1. **Результат работы программы**



**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Цель: Работа со Stream API в Java 8

Постановка задачи: Увеличение веса каждого объекта на 3, сортировка по весу в обратном порядке, фильтрация по дате рождения меньшей, чем 01.01.2000, сумма всех весов.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac2>

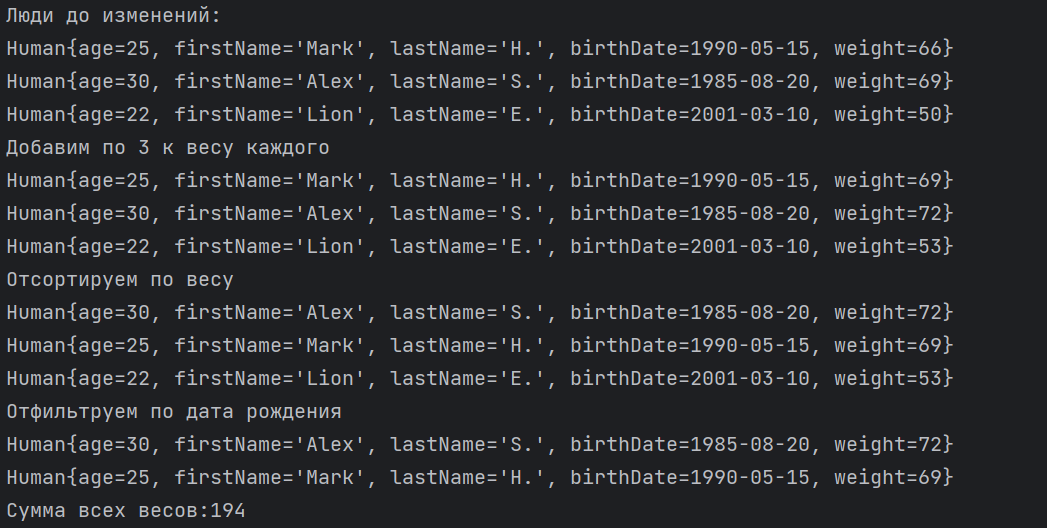
Класс Human:

package prac2;  
  
import java.time.LocalDate;  
  
public class Human {  
 private int age;  
 private String firstName;  
 private String lastName;  
 private LocalDate birthDate;  
 private int weight;  
  
 public Human(int age, String firstName, String lastName, LocalDate birthDate, int weight) {  
 this.age = age;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthDate = birthDate;  
 this.weight = weight;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public LocalDate getBirthDate() {  
 return birthDate;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public void setWeight(int weight) {  
 this.weight = weight;  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "age=" + age +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", weight=" + weight +  
 '}';  
 }  
}

Класс Main:

package prac2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class Main {  
 public static void main(String[] args) {  
 List<Human> h\_list = new ArrayList<>();  
 h\_list.add(new Human(25, "Mark", "H.", LocalDate.*of*(1990, 5, 15), 66));  
 h\_list.add(new Human(30, "Alex", "S.", LocalDate.*of*(1985, 8, 20), 69));  
 h\_list.add(new Human(22, "Lion", "E.", LocalDate.*of*(2001, 3, 10), 50));  
 System.*out*.println("Люди до изменений:");  
 h\_list.forEach(System.*out*::println);  
 System.*out*.println("Добавим по 3 к весу каждого");  
 h\_list = h\_list.stream()  
 .map(human -> {  
 human.setWeight(human.getWeight()+3);  
 return human;  
 }).collect(Collectors.*toList*());  
 h\_list.forEach(System.*out*::println);  
  
  
 System.*out*.println("Отсортируем по весу");  
 h\_list=h\_list.stream().sorted((human1,human2) -> human2.getWeight() - human1.getWeight()).collect(Collectors.*toList*());  
 h\_list.forEach(System.*out*::println);  
  
  
 System.*out*.println("Отфильтруем по дата рождения");  
 h\_list.stream().filter(human -> human.getBirthDate().isBefore(LocalDate.*of*(2000,1,1))).forEach(System.*out*::println);  
  
  
 int sum = h\_list.stream().mapToInt(Human::getWeight).sum();  
 System.*out*.println("Сумма всех весов:" + sum);  
 }  
}

1. **Результат работы программы**



**Практическая работа №3**

1. **Постановка задачи и персональный вариант**

Тема: знакомство с конкурентным программированием в Java. Потокобезопасность, ключевое слово syncrhonized, мьютексы, семафоры, мониторы, барьеры.

Постановка задачи: создать свои потокобезопасные имплементации интерфейсов: Map с использованием Lock, Set с использованием Semaphore.

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac3

package prac3.part1;  
  
public class Main {  
 public static void main(String[] args) {  
 ConcurrentMap<String, Integer> concurrentMap = new ConcurrentMap<>();  
  
 Thread thread1 = new Thread(() -> {  
 try {  
 for (int i = 0; i < 5; i++) {  
 concurrentMap.put("Key" + i, i);  
 System.*out*.println("Thread 1 added: Key" + i + " -> " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 try {  
 for (int i = 5; i < 10; i++) {  
 concurrentMap.put("Key" + i, i);  
 System.*out*.println("Thread 2 added: Key" + i + " -> " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 }  
}

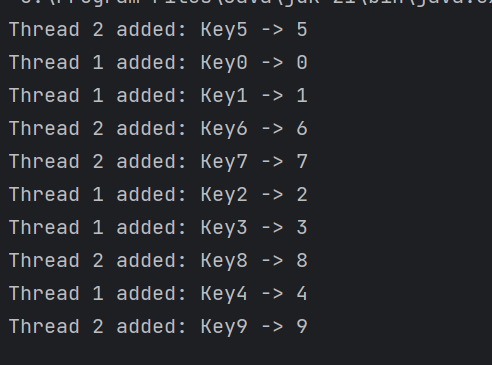
package prac3.part1;  
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class ConcurrentMap<K, V> {  
 private final Map<K, V> map;  
 private final Lock lock;  
  
 public ConcurrentMap() {  
 map = new HashMap<>();  
 lock = new ReentrantLock();  
 }  
  
 public void put(K key, V value) {  
 lock.lock();  
 try {  
 map.put(key, value);  
 } finally {  
 lock.unlock();  
 }  
 }  
}

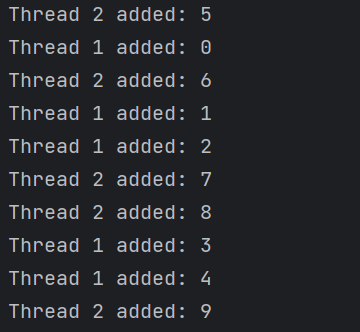
2-я часть:

package prac3;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.concurrent.Semaphore;  
  
public class ConcurrentSet<T> {  
 private Set<T> set;  
 private Semaphore semaphore;  
  
 public ConcurrentSet() {  
 set = new HashSet<>();  
 semaphore = new Semaphore(1); // Одна разрешающая семафор  
 }  
  
 public boolean add(T element) throws InterruptedException {  
 semaphore.acquire(); // Захватываем разрешение  
 boolean added = set.add(element);  
 semaphore.release(); // Освобождаем разрешение  
 return added;  
 }  
}

package prac3;  
  
public class Main {  
 public static void main(String[] args) {  
 ConcurrentSet<Integer> concurrentSet = new ConcurrentSet<>();  
  
 Thread thread1 = new Thread(() -> {  
 try {  
 for (int i = 0; i < 5; i++) {  
 concurrentSet.add(i);  
 System.*out*.println("Thread 1 added: " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 try {  
 for (int i = 5; i < 10; i++) {  
 concurrentSet.add(i);  
 System.*out*.println("Thread 2 added: " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 }  
}

1. **Результат работы программы**





**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Цель: работа с ExecutorService, CompletableFuture

Постановка задачи: реализовать собственную имплементацию ExecutorService с единственным параметром конструктора – количеством потоков.

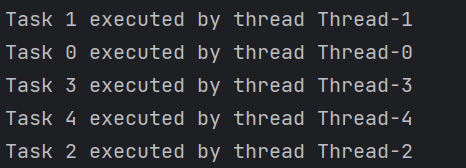
1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac4>

package prac4;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
class CustomExecutorService  
{  
 private final List<WorkerThread> threads;  
 private final BlockingQueue<Runnable> taskQueue;  
  
 public CustomExecutorService(int threadPoolSize)  
 {  
 this.threads = new ArrayList<>(threadPoolSize);  
 this.taskQueue = new LinkedBlockingQueue<>();  
  
 for (int i = 0; i < threadPoolSize; i++)  
 {  
 WorkerThread workerThread = new WorkerThread();  
 threads.add(workerThread);  
 workerThread.start();  
 }  
 }  
  
 public void submit(Runnable task)  
 {  
 try  
 {  
 taskQueue.put(task);  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
  
 public void shutdown()  
 {  
 for (WorkerThread workerThread : threads)  
 {  
 workerThread.interrupt();  
 }  
 }  
  
 private class WorkerThread extends Thread  
 {  
 @Override  
 public void run()  
 {  
 while (!Thread.*currentThread*().isInterrupted())  
 {  
 try  
 {  
 Runnable task = taskQueue.take();  
 task.run();  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }  
 }  
}

package prac4;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 CustomExecutorService executorService = new CustomExecutorService(5);  
  
 for (int i = 0; i < 10; i++)  
 {  
 final int taskNumber = i;  
 executorService.submit(() -> {  
 System.*out*.println("Task " + taskNumber + " executed by thread " + Thread.*currentThread*().getName());  
 });  
 }  
  
 executorService.shutdown();  
 }  
}

1. **Результат работы программы**



**Практическая работа №5**

1. **Постановка задачи и персональный вариант**

Цель: познакомиться с паттернами проектирования, их определением и классификацией. Обзор паттернов GoF. Паттерн Синглтон.

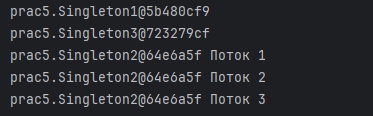
Постановка задачи: реализовать паттерн Singleton как минимум 3-мя способами.

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac5

package prac5;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println(Singleton1.*getInstance*());  
 System.*out*.println(Singleton3.*getInstance*());  
 for (int i = 1; i < 4; i++) {  
 var t = new Thread(() -> {  
 System.*out*.println(Singleton2.*getInstance*() + " " + Thread.*currentThread*().getName());  
 });  
 t.setName("Поток " + i);  
 t.start();  
 }  
 }  
}  
  
class Singleton1{//ленивая инициализация  
 private static Singleton1 *instance*;  
 private Singleton1(){}  
 public static Singleton1 getInstance(){  
 if (*instance* == null){  
 *instance* = new Singleton1();  
 }  
 return *instance*;  
 }  
}  
class Singleton2 {//потокобезопасный  
 private static Singleton2 *instance*;  
 private Singleton2(){}  
 public static synchronized Singleton2 getInstance(){  
 if (*instance* == null){  
 *instance* = new Singleton2();  
 }  
 return *instance*;  
 }  
}  
class Singleton3 {//холодная инициализация  
 private static final Singleton3 *instance* = new Singleton3();  
 private Singleton3() {}  
 public static Singleton3 getInstance() {  
 return *instance*;  
 }  
}

1. **Результат работы программы**



**Практическая работа №6**

1. **Постановка задачи и персональный вариант**

Цель: знакомство с реализацией порождающих паттернов проектирования.

Постановка задачи: написать реализацию паттернов «Фабричный метод», «Абстрактная фабрика», «Строитель», «Прототип».

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac6

package prac6;  
  
public class Main {  
 public static void main(String[] args) {  
 // «Фабричный метод»  
 System.*out*.println("Паттерн «Фабричный метод»:");  
 new FactoryMethod();  
 System.*out*.println();  
  
 // «Абстрактная фабрика»  
 System.*out*.println("Паттерн «Абстрактная фабрика»:");  
 new AbstractFactory();  
 System.*out*.println();  
  
 // «Строитель»  
 System.*out*.println("Паттерн «Строитель»:");  
 new Builder();  
 System.*out*.println();  
  
 // «Прототип»  
 System.*out*.println("Паттерн «Прототип»:");  
 new Prototype();  
  
 }  
}

import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;

}  
}

FactoryMethod

package prac6;  
  
public class FactoryMethod{  
 public FactoryMethod() {  
 Work();  
 }  
  
 private void Work() {  
 Factory factory = new PanelFactory("Завод панельных домов");  
 factory.BuildHouse();  
 factory = new WoodFactory("Завод деревянных домов");  
 factory.BuildHouse();  
 }  
}  
  
// Фабрики  
abstract class Factory{  
 public String Name;  
  
 public Factory(String name) {  
 Name = name;  
 }  
  
 abstract House BuildHouse();  
}  
  
class PanelFactory extends Factory{  
 public PanelFactory(String name) {  
 super(name);  
 }  
  
 @Override  
 House BuildHouse() {  
 return new PanelHouse();  
 }  
}  
  
class WoodFactory extends Factory {  
 public WoodFactory(String name) {  
 super(name);  
 }  
  
 @Override  
 House BuildHouse() {  
 return new WoodHouse();  
 }  
}  
  
// Виды домов  
interface House { }  
  
class PanelHouse implements House {  
 public PanelHouse() {  
 System.*out*.println("Panel house!");  
 }  
}  
  
class WoodHouse implements House{  
 public WoodHouse() {  
 System.*out*.println("Wooden house!");  
 }  
}

AbstractFactory

package prac6;  
  
public class AbstractFactory{  
 public AbstractFactory(){  
 Work();  
 }  
  
 private void Work(){  
 var hero1 = new Hero(new ElfFactory());  
 var hero2 = new Hero(new TankFactory());  
 hero1.Attack();  
 hero2.Attack();  
 }  
}  
  
// Оружие  
abstract class Weapon{  
 abstract void Hit();  
}  
class Sword extends Weapon{  
 public void Hit(){  
 System.*out*.println("Удар мечом!");  
 }  
}  
class Bow extends Weapon{  
 public void Hit(){  
 System.*out*.println("Выстрел из лука!");  
 }  
}  
  
// Герой  
class Hero{  
 private Weapon HeroWeapon;  
  
 public Hero(HeroFactory factory){  
 HeroWeapon = factory.CreateWeapon();  
 }  
  
 public void Attack(){  
 HeroWeapon.Hit();  
 }  
}  
  
// Фабрики  
abstract class HeroFactory{  
 abstract Weapon CreateWeapon();  
}  
class TankFactory extends HeroFactory{  
 public Weapon CreateWeapon(){  
 return new Sword();  
 }  
}  
class ElfFactory extends HeroFactory{  
 public Weapon CreateWeapon(){  
 return new Bow();  
 }  
}

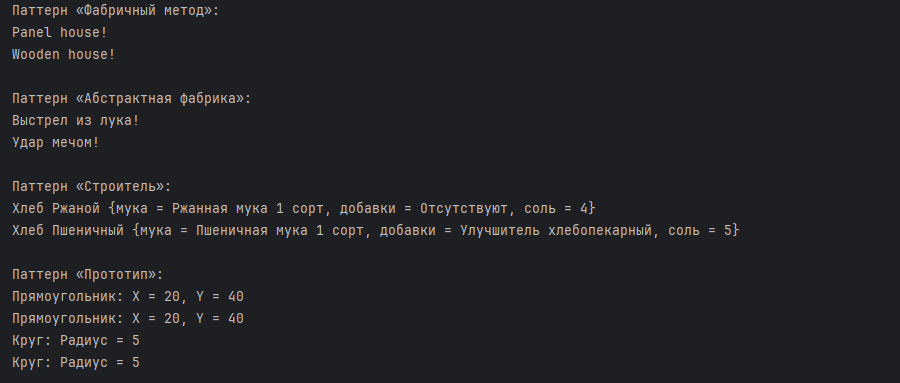
Builder

package prac6;  
  
public class Builder{  
 public Builder(){  
 Work();  
 }  
  
 private void Work(){  
 var baker = new Baker();  
 var bread1 = baker.CreateBread(new RyeBreadBuilder());  
 System.*out*.println(bread1);  
 var bread2 = baker.CreateBread(new WheatBreadBuilder());  
 System.*out*.println(bread2);  
 }  
}  
  
// Для хлеба  
class Flour{  
 public String FlourType;  
  
 public Flour(String flourType){  
 FlourType = flourType;  
 }  
}  
class Additive{  
 public String AdditiveName;  
  
 public Additive(String additiveName){  
 AdditiveName = additiveName;  
 }  
}  
class Salt{  
 public int saltCount;  
  
 public Salt(int count){  
 saltCount = count;  
 }  
}  
  
class Bread{  
 public String name;  
 public Flour flour;  
 public Additive additive;  
 public Salt salt;  
  
 @Override  
 public String toString(){  
 return "Хлеб " + name + " {" +  
 "мука = " + flour.FlourType +  
 ", добавки = " + additive.AdditiveName +  
 ", соль = " + salt.saltCount + "}";  
 }  
}  
  
// Получение хлеба  
class Baker{  
 public Bread CreateBread(BreadBuilder builder){  
 builder.SetName();  
 builder.SetFlour();  
 builder.SetAdditive();  
 builder.SetSalt();  
 return builder.bread;  
 }  
}  
  
abstract class BreadBuilder{  
 public Bread bread;  
  
 public BreadBuilder(){  
 bread = new Bread();  
 }  
  
 abstract void SetName();  
 abstract void SetFlour();  
 abstract void SetAdditive();  
 abstract void SetSalt();  
}  
  
class RyeBreadBuilder extends BreadBuilder{  
 void SetName(){  
 this.bread.name = "Ржаной";  
 }  
 void SetFlour(){  
 this.bread.flour = new Flour("Ржанная мука 1 сорт");  
 }  
 void SetAdditive(){  
 this.bread.additive = new Additive("Отсутствуют");  
 }  
 void SetSalt(){  
 this.bread.salt = new Salt(4);  
 }  
}  
class WheatBreadBuilder extends BreadBuilder{  
 void SetName(){  
 this.bread.name = "Пшеничный";  
 }  
 void SetFlour(){  
 this.bread.flour = new Flour("Пшеничная мука 1 сорт");  
 }  
 void SetAdditive(){  
 this.bread.additive = new Additive("Улучшитель хлебопекарный");  
 }  
 void SetSalt(){  
 this.bread.salt = new Salt(5);  
 }  
}

Prototype

package prac6;  
  
public class Prototype{  
 public Prototype(){  
 Work();  
 }  
  
 private void Work(){  
 var figure1 = new Rectangle(20, 40);  
 var figure1\_1 = figure1.Clone();  
 figure1.GetInfo();  
 figure1\_1.GetInfo();  
  
 var figure2 = new Circle(5);  
 var figure2\_1 = figure2.Clone();  
 figure2.GetInfo();  
 figure2\_1.GetInfo();  
 }  
}  
  
interface IFigure{  
 IFigure Clone();  
 void GetInfo();  
}  
  
class Rectangle implements IFigure{  
 private int X;  
 private int Y;  
  
 public Rectangle(int x, int y){  
 X = x;  
 Y = y;  
 }  
  
 public IFigure Clone(){  
 return new Rectangle(this.X, this.Y);  
 }  
  
 public void GetInfo(){  
 System.*out*.println("Прямоугольник: X = " + X + ", Y = " + Y);  
 }  
}  
  
class Circle implements IFigure {  
 private int Radius;  
  
 public Circle(int radius) {  
 Radius = radius;  
 }  
  
 public IFigure Clone() {  
 return new Circle(this.Radius);  
 }  
  
 public void GetInfo() {  
 System.*out*.println("Круг: Радиус = " + Radius);  
 }  
}

1. **Результат работы программы**



**Практическая работа №7**

1. **Постановка задачи и персональный вариант**

Цель: Реализация структурных паттернов проектирования.

Постановка задачи: написать реализацию паттернов Адаптер и Фасад.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac7>

Adapter.java

package prac7;  
  
public class Adapter{  
 public Adapter(){  
 Driver driver = new Driver();  
 vehicle vehicle = new vehicle();  
 driver.Travel(vehicle);  
  
 Camel camel = new Camel();  
 ITransport camelTransport = new CamelToTransportAdapter(camel);  
 driver.Travel(camelTransport);  
 }  
}  
  
class Driver{  
 public void Travel(ITransport transport){  
 transport.Drive();  
 }  
}  
  
interface ITransport{  
 void Drive();  
}  
class vehicle implements ITransport{  
 public void Drive(){  
 System.*out*.println("vehicle drived");  
 }  
}  
  
interface IAnimal{  
 void Move();  
}  
class Camel implements IAnimal{  
 public void Move(){  
 System.*out*.println("camel moved");  
 }  
}  
  
class CamelToTransportAdapter implements ITransport{  
 Camel camel;  
 public CamelToTransportAdapter(Camel c){  
 camel = c;  
 }  
  
 public void Drive(){  
 camel.Move();  
 }  
}

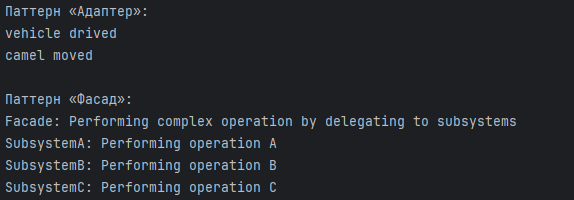
Facade.java

package prac7;  
  
class SubsystemA {  
 public void operationA() {  
 System.*out*.println("SubsystemA: Performing operation A");  
 }  
}  
  
class SubsystemB {  
 public void operationB() {  
 System.*out*.println("SubsystemB: Performing operation B");  
 }  
}  
  
class SubsystemC {  
 public void operationC() {  
 System.*out*.println("SubsystemC: Performing operation C");  
 }  
}  
public class Facade {  
 private SubsystemA subsystemA;  
 private SubsystemB subsystemB;  
 private SubsystemC subsystemC;  
  
 public Facade() {  
 this.subsystemA = new SubsystemA();  
 this.subsystemB = new SubsystemB();  
 this.subsystemC = new SubsystemC();  
 }  
  
 public void performComplexOperation() {  
 System.*out*.println("Facade: Performing complex operation by delegating to subsystems");  
 subsystemA.operationA();  
 subsystemB.operationB();  
 subsystemC.operationC();  
 }  
}

Main.java

package prac7;  
  
class SubsystemA {  
 public void operationA() {  
 System.*out*.println("SubsystemA: Performing operation A");  
 }  
}  
  
class SubsystemB {  
 public void operationB() {  
 System.*out*.println("SubsystemB: Performing operation B");  
 }  
}  
  
class SubsystemC {  
 public void operationC() {  
 System.*out*.println("SubsystemC: Performing operation C");  
 }  
}  
public class Facade {  
 private SubsystemA subsystemA;  
 private SubsystemB subsystemB;  
 private SubsystemC subsystemC;  
  
 public Facade() {  
 this.subsystemA = new SubsystemA();  
 this.subsystemB = new SubsystemB();  
 this.subsystemC = new SubsystemC();  
 }  
  
 public void performComplexOperation() {  
 System.*out*.println("Facade: Performing complex operation by delegating to subsystems");  
 subsystemA.operationA();  
 subsystemB.operationB();  
 subsystemC.operationC();  
 }  
}

1. **Результат работы программы**



**Практическая работа №8**

1. **Постановка задачи и персональный вариант**

Цель: Реализация поведенческих паттернов проектирования.

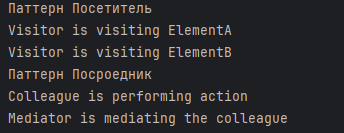
Постановка задачи: написать реализацию паттернов Посетитель и Посредник.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac8>

package prac8;  
  
import java.util.ArrayList;  
import java.util.List;  
  
//ПОСЕТИТЕЛЬ  
interface Visitor {  
 void visit(ElementA element);  
 void visit(ElementB element);  
}  
  
class ConcreteVisitor implements Visitor {  
 @Override  
 public void visit(ElementA element) {  
 System.*out*.println("Visitor is visiting ElementA");  
 }  
  
 @Override  
 public void visit(ElementB element) {  
 System.*out*.println("Visitor is visiting ElementB");  
 }  
}  
  
interface Element {  
 void accept(Visitor visitor);  
}  
  
class ElementA implements Element {  
 @Override  
 public void accept(Visitor visitor) {  
 visitor.visit(this);  
 }  
}  
  
class ElementB implements Element {  
 @Override  
 public void accept(Visitor visitor) {  
 visitor.visit(this);  
 }  
}  
  
  
//ПОСРЕДНИК  
interface Mediator {  
 void mediate(Colleague colleague);  
}  
  
class ConcreteMediator implements Mediator {  
 @Override  
 public void mediate(Colleague colleague) {  
 System.*out*.println("Mediator is mediating the colleague");  
 // Perform mediation logic here  
 }  
}  
  
interface Colleague {  
 void action();  
}  
  
class ConcreteColleague implements Colleague {  
 private Mediator mediator;  
  
 public ConcreteColleague(Mediator mediator) {  
 this.mediator = mediator;  
 }  
  
 @Override  
 public void action() {  
 System.*out*.println("Colleague is performing action");  
 // Perform some action  
 mediator.mediate(this);  
 }  
}  
  
// Main class  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Паттерн Посетитель");  
 List<Element> elements = new ArrayList<>();  
 elements.add(new ElementA());  
 elements.add(new ElementB());  
  
 Visitor visitor = new ConcreteVisitor();  
 for (Element element : elements) {  
 element.accept(visitor);  
 }  
  
 System.*out*.println("Паттерн Посроедник");  
 Mediator mediator = new ConcreteMediator();  
 Colleague colleague = new ConcreteColleague(mediator);  
 colleague.action();  
 }  
}

1. **Результат работы программы**



**Практическая работа №9**

1. **Постановка задачи и персональный вариант**

Цель: Знакомство с системой сборки приложения. Gradle.

Постановка задачи: создать приложение, которое выводит какое-то сообщение в консоль. Создать Gradle Task, который создает jar-файл приложения, переносит его в отдельную папку, в которой хранится Dockerfile для jar, а затем создает Docker контейнер из данного jar-файла и запускает его.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task9>

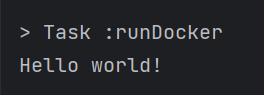
Класс Main:

public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Hello world!");  
 }  
}

Файл: build.gradle:

plugins {  
 id("java")  
}  
  
group = "org.example"  
version = "1.0-SNAPSHOT"  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 testImplementation 'org.junit.jupiter:junit-jupiter-api:5.7.2'  
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.7.2'  
}  
  
jar {  
 manifest {  
 attributes 'Main-Class': 'Main'  
 }  
}  
task buildDocker(type: Exec) {  
 dependsOn jar  
  
 workingDir '.'  
  
 commandLine 'docker', 'build', '-t', 'test', '.'  
}  
  
task runDocker(type: Exec) {  
 dependsOn buildDocker  
  
 workingDir '.'  
  
 commandLine 'docker', 'run', '--rm', 'test'  
}

1. **Результат работы программы**



**Практическая работа №10**

1. **Постановка задачи и персональный вариант**

Цель: Введение в Spring. Container. Bean. Внедрение зависимостей, основанных на конструкторах и сеттерах. Конфигурация бинов. Автоматическое обнаружение и связывание классов.

Постановка задачи: создать приложение, в котором создается ApplicationContext и из него берётся бин с названием, переданным в качестве аргумента к приложению, и вызывается метод интерфейса, который он имплементирует. Нужно создать по одному бину для каждого класса, определить им название. Проверить, что вызывается при вводе названия каждого из бинов. Классы и интерфейс: Интерфейс Programmer с методом doCoding(), его имплементации: Junior, Middle, Senior.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task10>

Класс Task10Application:

import config.BeanConfig;  
import interfaces.Programmer;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
public class Task10Application {  
 public static void main(String[] args)  
 {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
  
 Programmer bean = context.getBean("GetJunior", Programmer.class);  
 bean.doCoding();  
 bean = context.getBean("GetMiddle", Programmer.class);  
 bean.doCoding();  
 bean = context.getBean("GetSenior", Programmer.class);  
 bean.doCoding();  
 }  
}

Класс BeanConfig:

package config;  
  
import component.Junior;  
import component.Middle;  
import component.Senior;  
import interfaces.Programmer;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ComponentScan  
public class BeanConfig {  
 @Bean  
 public Programmer GetJunior(){  
 return new Junior();  
 }  
 @Bean  
 public Programmer GetMiddle(){  
 return new Middle();  
 }  
 @Bean  
 public Programmer GetSenior(){  
 return new Senior();  
 }  
}

Класс Junior:

package component;  
  
import interfaces.Programmer;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Junior implements Programmer {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Junior coding");  
 }  
}

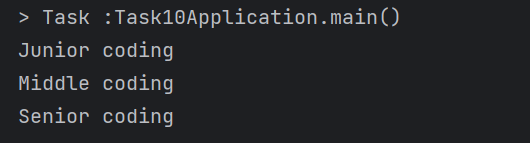
Класс Middle:

package component;  
import interfaces.Programmer;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Middle implements Programmer {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Middle coding");  
 }  
}

Класс Senior:

package component;  
import interfaces.Programmer;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Senior implements Programmer {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Senior coding");  
 }  
}

1. **Результат работы программы**



**Практическая работа №11**

1. **Постановка задачи и персональный вариант**

Цель: разобраться с использованием Spring boot.

Постановка задачи: создать приложение с использованием Spring Boot Starter Initializr (https://start.spring.io/) с такими зависимостями:

– Spring Web;

– Lombok;

– Validation;

– Spring boot Actuator.

Запустить приложение и удостовериться, что не появилось никаких ошибок. Добавить все эндпоинты в Actuator, сделать HTTP-запрос на проверку состояния приложения. Собрать jar-файл приложения, запустить и проверить состояние при помощи REST-запроса.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task11>

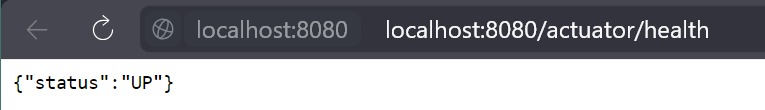
Класс Task11Application:

package com.example.task11;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task11Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Task11Application.class, args);  
 }  
  
}

Класс HealthCheckController:

package com.example.task11.controllers;  
  
import org.springframework.web.bind.annotation.GetMapping;  
  
public class HealthCheckController {  
 @GetMapping("/health")  
 public String checkHealth() {  
 return "Application is up and running!";  
 }  
}

1. **Результат работы программы**



**Практическая работа №12**

1. **Постановка задачи и персональный вариант**

Цель: Работа с жизненным циклом компонентов. Аннотации PostConstruct, PreDestroy.

Постановка задачи: cоздать приложение, которое при запуске берет данные из одного файла, хеширует, а при остановке приложения удаляет исходный файл, оставляя только файл с захешированными данными. Названия первого и второго файла передаются в качестве аргументов при запуске. При отсутствии первого файла создает второй файл и записывает в него строку null. Реализовать с использованием аннотаций PostConstruct, PreDestroy.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task12>

Класс Task12Application:

package com.example.task11;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task11Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Task11Application.class, args);  
 }  
  
}

Класс EditFiles:

package org;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;  
import org.springframework.stereotype.Service;  
  
import java.io.\*;  
  
@Service  
public class EditFiles  
{  
 String file1 = "D:\\code\\\_4\\java\\task12\\f1.txt";  
 String file2 = "D:\\code\\\_4\\java\\task12\\f2.txt";  
  
 @PostConstruct  
 public void PostConstruct() throws IOException{  
 var bufferedWriter = new BufferedWriter(new FileWriter(file2));  
  
 if (new File(file1).exists()){  
 var bufferedReader = new BufferedReader(new FileReader(file1));  
 var strBuilder = new StringBuilder();  
 String line;  
  
 while ((line = bufferedReader.readLine()) != null){  
 strBuilder.append(line);  
 }  
 bufferedWriter.write(String.*valueOf*(strBuilder.hashCode()));  
 System.*out*.println("Текст из 1 файла был передан в виде хеша во 2");  
 bufferedReader.close();  
 }  
 else {  
 System.*out*.println("Первого файла нет");  
 bufferedWriter.write("null");  
 }  
 bufferedWriter.close();  
 }  
  
 @PreDestroy  
 public void PreDestroy(){  
 if(new File(file1).delete()){  
 System.*out*.println("Первый файл удален");  
 }  
 }  
}

1. **Результат работы программы**

