|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | | |
|  | Институт информационных технологий (ИТ) |
|  | Кафедра инструментального и прикладного программного обеспечения (ИиППО) |

|  |  |  |  |
| --- | --- | --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №1 - 13** | | | |
| **по дисциплине** | | | |
| **«Шаблоны программных платформ языка Java»**  **Вариант 21** | | | |
| Выполнил студент группы ИКБО-20-19 | | Николаев-Аксенов И. С. | |
|  | |  | |
| Принял  *Ассистент* | | Батанов А. О. | |
| Практические работы выполнены | «\_\_\_»\_\_\_\_\_\_\_2021 г. | | (подпись студента) | |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_2021 г. | | (подпись руководителя) | |
|  |  | |  | |

Москва 2021

**Содержание**

[Практическая работа №1 3](#_Toc67350870)

[Практическая работа №2 5](#_Toc67350871)

[Практическая работа №3 8](#_Toc67350872)

[Практическая работа №4 14](#_Toc67350873)

[Практическая работа №5 16](#_Toc67350874)

[Практическая работа №6 18](#_Toc67350875)

[Практическая работа №7 30](#_Toc67350876)

[Практическая работа №8 35](#_Toc67350877)

[Практическая работа №9 41](#_Toc67350878)

[Практическая работа №10 45](#_Toc67350879)

[Практическая работа №11 48](#_Toc67350880)

[Практическая работа №12 51](#_Toc67350881)

[Практическая работа №13 54](#_Toc67350882)

[Вывод 56](#_Toc67350883)

[Список использованных источников 56](#_Toc67350884)

Практическая работа №1

***Цель работы***

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

Постановка задачи: Имплементировать интерфейс Comparator, сравнивающий двух студентов по набранным за семестр баллов.

***Листинг программы***

*Student.java*

public class Student {

private double gpa = 0;

public Student(double gpa) {

this.gpa = gpa;

}

public double getGPA() {

return gpa;

}

public void setGPA(double gpa) {

this.gpa = gpa;

}

}

*startStudent.java*

import java.util.Comparator;

import java.util.Scanner;

public class startStudent {

public static void main(String[] args) {

System.out.print("Введите два балла GPA для двух студентов через пробел: ");

Scanner sc = new Scanner(System.in);

String[] arr = sc.nextLine().split(" ");

Student a = new Student(Double.parseDouble(arr[0]));

Student b = new Student(Double.parseDouble(arr[1]));

Comparator<Student> compGPA = (o1, o2) -> Double.compare(o1.getGPA(), o2.getGPA());

double result = compGPA.compare(a, b);

if(result >= 1)

System.out.println("У первого студента GPA больше чем у второго.");

else if(result == 0)

System.out.println("Оба студента имеют одиннаковые баллы.");

else

System.out.println("У второго студента GPA больше чем у первого");

}

}

***Результат выполнения программы***



Рисунок 1.1 – Демонстрация работы программы

Практическая работа №2

***Цель работы***

Тема: Работа со Stream API в Java 8.

Постановка задачи: Уменьшение веса каждого объекта на 5, фильтрация по дате рождения меньшей, чем 3 февраля 1999, конкатенация фамилий в строку через пробел.

***Листинг программы***

*Human.java*

import java.time.LocalDate;

public class Human {

private int age, weight;

private String firstName, lastName;

private LocalDate birthDate;

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 void setAge(int age) {

this.age = age;

}

public int getWeight() {

return weight;

}

public void setWeight(int weight) {

this.weight = weight;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public LocalDate getBirthDate() {

return birthDate;

}

public void setBirthDate(LocalDate birthDate) {

this.birthDate = birthDate;

}

@Override

public String toString() {

return firstName + ' ' + lastName + ", " + age + " years, " + birthDate + ", " + weight + "kg";

}

}

*HumanBuilder.java*

import java.time.LocalDate;

import java.util.\*;

import java.util.stream.Stream;

public class HumanBuilder {

private String[] firstNames = {"James", "Mary", "John", "Patricia", "Robert", "Jennifer", "Michael", "Linda", "William", "Elizabeth", "David", "Barbara", "Richard", "Susan", "Joseph", "Jessica", "Thomas", "Sarah", "Charles", "Karen", "Christopher", "Nancy", "Daniel", "Lisa", "Matthew", "Margaret", "Anthony", "Betty", "Donald", "Sandra", "Mark", "Ashley", "Paul", "Dorothy", "Steven", "Kimberly", "Andrew", "Emily", "Kenneth", "Donna", "Joshua", "Michelle", "Kevin", "Carol", "Brian", "Amanda"};

private String[] lastNames = {"Smith", "Johnson", "Williams", "Brown", "Jones", "Garcia", "Miller", "Davis", "Rodriguez", "Martinez", "Hernandez", "Lopez", "Gonzalez", "Wilson", "Anderson", "Thomas", "Taylor", "Moore", "Jackson", "Martin", "Lee", "Perez", "Thompson", "White", "Harris", "Sanchez", "Clark", "Ramirez", "Lewis", "Robinson"};

private List<Human> generateList(int size) {

List<Human> result = new ArrayList<>();

Random rand = new Random();

for(int i = 0; i < size; i++) {

LocalDate ld = LocalDate.of(rand.nextInt(70) + 1950, rand.nextInt(12) + 1, rand.nextInt(27) + 1);

result.add(new Human(rand.nextInt(99), firstNames[rand.nextInt(firstNames.length)], lastNames[rand.nextInt(lastNames.length)], ld, rand.nextInt(100) + 20));

}

return result;

}

public void humanTask(int size) {

List<Human> harr = generateList(size);

System.out.println("Original list:");

harr.forEach(System.out::println);

System.out.println("\nList after applying stream methods:");

Stream<Human> s1 = harr.stream();

Stream<Human> s2 = harr.stream();

Stream<Human> s3 = harr.stream();

System.out.println("\nFirst stream: ");

s1.peek(o -> o.setWeight(o.getWeight() - 5)).forEach(System.out::println);

System.out.println("\nSecond stream: ");

s2.filter(o -> o.getBirthDate().isBefore(LocalDate.of(1999, 2, 3))).forEach(System.out::println);

System.out.println("\nThird stream: ");

s3.map(o -> o.getLastName() + ' ').forEach(System.out::print);

}

}

*Main.java*

public class main {

public static void main(String[] args) {

HumanBuilder hb = new HumanBuilder();

hb.humanTask(10);

}

}

***Результат выполнения программы***

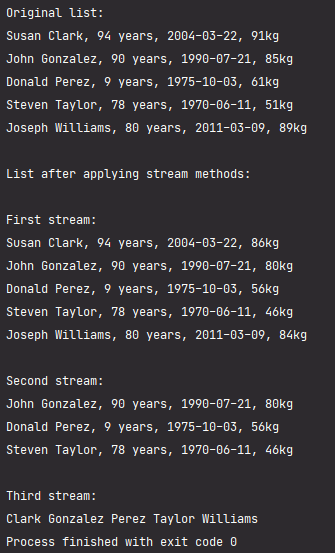


Рисунок 2.1 – Демонстрация работы программы

Практическая работа №3

***Цель работы***

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

Постановка задачи: Создать свои потокобезопасные имплементации интерфейсов в соответствии с вариантом индивидуального задания. Set с использованием Lock, Map с использованием Semaphore.

***Листинг программы***

*MapSemaphore.java*

import java.util.Collection;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.Semaphore;

public class MapSemaphore<K, V> extends HashMap<K, V> implements Map<K, V> {

private static final Semaphore sem = new Semaphore(1);

@Override

public int size() {

return super.size();

}

@Override

public boolean isEmpty() {

return super.isEmpty();

}

@Override

public boolean containsKey(Object key) {

boolean b = false;

try {

sem.acquire();

b = super.containsKey(key);

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return b;

}

@Override

public boolean containsValue(Object value) {

boolean b = false;

try {

sem.acquire();

b = super.containsValue(value);

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return b;

}

@Override

public V get(Object key) {

V k = null;

try {

sem.acquire();

k = super.get(key);

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return k;

}

@Override

public V put(K key, V value) {

V k = null;

try {

sem.acquire();

k = super.put(key, value);

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return k;

}

@Override

public V remove(Object key) {

V k = null;

try {

sem.acquire();

k = super.remove(key);

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return k;

}

@Override

public void putAll(Map<? extends K, ? extends V> m) {

try {

sem.acquire();

super.putAll(m);

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public void clear() {

try {

sem.acquire();

super.clear();

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public Set<K> keySet() {

Set<K> k = null;

try {

sem.acquire();

k = super.keySet();

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return k;

}

@Override

public Collection<V> values() {

Collection<V> k = null;

try {

sem.acquire();

k = super.values();

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

return k;

}

@Override

public Set<Entry<K, V>> entrySet() {

Set<Entry<K, V>> k = null;

try {

sem.acquire();

k = super.entrySet();

sem.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

assert k != null;

return k;

}

}

*SetLock.java*

import java.util.ArrayList;

import java.util.Collection;

import java.util.Iterator;

import java.util.Set;

import java.util.concurrent.locks.ReentrantLock;

public class SetLock<E> implements Set {

private ArrayList<E> arr = new ArrayList<>();

private static final ReentrantLock re = new ReentrantLock();

@Override

public boolean add(Object o) {

re.lock();

boolean b = arr.add((E) o);

re.unlock();

return b;

}

@Override

public boolean remove(Object o) {

re.lock();

boolean b = arr.remove(o);

re.unlock();

return b;

}

@Override

public boolean addAll(Collection c) {

re.lock();

boolean b = arr.addAll(c);

re.unlock();

return b;

}

@Override

public void clear() {

re.lock();

arr.clear();

re.unlock();

}

@Override

public boolean removeAll(Collection c) {

re.lock();

boolean b = arr.removeAll(c);

re.unlock();

return b;

}

@Override

public boolean retainAll(Collection c) {

re.lock();

boolean b = arr.retainAll(c);

re.unlock();

return b;

}

@Override

public boolean containsAll(Collection c) {

re.lock();

boolean b = arr.containsAll(c);

re.unlock();

return b;

}

@Override

public boolean contains(Object o) {

re.lock();

boolean b = arr.contains((E) o);

re.unlock();

return b;

}

@Override

public int size() {

return arr.size();

}

@Override

public boolean isEmpty() {

return arr.isEmpty();

}

@Override

public Iterator iterator() {

return arr.iterator();

}

@Override

public Object[] toArray() {

return arr.toArray();

}

@Override

public Object[] toArray(Object[] a) {

return arr.toArray(a);

}

@Override

public String toString() {

return "SetLock{" +

"arr=" + arr +

'}';

}

}

*TypesTester.java*

public class TypesTester {

static class t1 extends Thread {

@Override

public void run() {

SetLock<Integer> sl = new SetLock<>();

sl.add(5);

sl.add(15);

sl.add(25);

System.out.println(sl);

System.out.println(sl.contains(25));

sl.remove(15);

System.out.println(sl.size());

}

}

static class t2 extends Thread {

@Override

public void run() {

MapSemaphore<Integer, String> ms = new MapSemaphore<>();

ms.put(1, "Ivan");

ms.put(2, "Petr");

ms.put(3, "Mikhail");

System.out.println(ms);

System.out.println(ms.values());

System.out.println(ms.keySet());

ms.remove(2);

System.out.println(ms.size());

}

}

public static void main(String[] args) {

t1 th11 = new t1();

t1 th12 = new t1();

th11.start();

th12.start();

try {

Thread.sleep(900);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("---------------------");

t2 th21 = new t2();

t2 th22 = new t2();

th21.start();

th22.start();

}

}

***Результат выполнения программы***

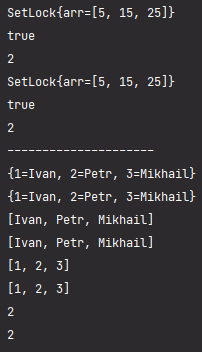


Рисунок 3.1 – Демонстрация работы программы

Практическая работа №4

***Цель работы***

Тема: Работа с ExecutorService, CompletableFuture.

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

***Листинг программы***

*RandomWordThread.java*

import java.util.Random;

public class RandomWordThread implements Runnable {

private String[] words = {"capricious", "roasted", "check", "box", "day", "debonair", "coordinated", "observe", "beds", "jail", "wide-eyed", "anger", "attraction", "slimy", "thoughtless", "time", "drab", "pushy", "smiling", "helpful", "understood", "truck", "hobbies", "delight", "launch", "acoustic", "troubled", "thick", "cattle", "explode", "large", "melt", "release", "bashful", "hurried", "animal", "bite-sized", "kneel", "unaccountable", "squealing", "show", "drown", "telling", "aunt", "low", "superficial", "wave", "breath", "succeed", "complain"};

private Random random = new Random();

@Override

public void run() {

try {

System.out.println(words[random.nextInt(words.length)]);

Thread.sleep(random.nextInt(5001) + 500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

*ExecutorServiceHandler.java*

import java.util.Random;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class ExecutorServiceHandler {

private ExecutorService exec;

private Random random = new Random();

public ExecutorServiceHandler(int number) {

exec = Executors.newFixedThreadPool(number);

int bound = random.nextInt(30) + 1;

System.out.println("Запуск " + bound + " потоков, при возможных " + number + " потоках..\n");

for(int i = 0; i < bound; i++) {

System.out.println("Запуск потока №" + (i + 1));

exec.execute(new RandomWordThread());

}

exec.shutdown();

System.out.println("Завершение работы потоков...");

}

}

*Main.java*

public class Main {

public static void main(String[] args) {

ExecutorServiceHandler esh = new ExecutorServiceHandler(3);

}

}

***Результат выполнения программы***

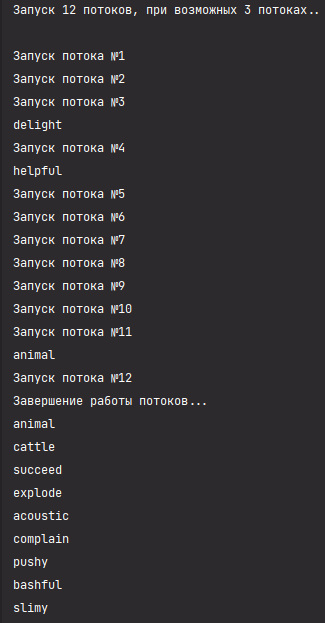


Рисунок 4.1 – Демонстрация работы программы

Практическая работа №5

***Цель работы***

Тема: Познакомиться с паттернами проектирования, их определением и классификацией. Обзор паттернов GoF. Паттерн Синглтон.

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

***Листинг программы***

*FirstSingletonImpml.java*

public class FirstSingletonImpl {

private static FirstSingletonImpl instance;

private FirstSingletonImpl() {

System.out.println("FirstSingletonImpl");

}

public static FirstSingletonImpl getInstance() {

if(instance == null)

instance = new FirstSingletonImpl();

return instance;

}

}

*SecondSingletonImpml.java*

public class SecondSingletonImpl {

private static SecondSingletonImpl instance = new SecondSingletonImpl();

private SecondSingletonImpl() {

System.out.println("SecondSingletonImpl");

}

public static SecondSingletonImpl getInstance() {

return instance;

}

}

*ThirdSingletonImpml.java*

public class ThirdSingletonImpl {

private ThirdSingletonImpl() {

System.out.println("ThirdSingletonImpl");

}

private static class ThirdSingletonImplHolder {

private final static ThirdSingletonImpl instance = new ThirdSingletonImpl();

}

public static ThirdSingletonImpl getInstance() {

return ThirdSingletonImplHolder.instance;

}

}

*FourthSingletonImpml.java*

public class FourthSingletonImpl {

private static volatile FourthSingletonImpl instance;

private FourthSingletonImpl() {

System.out.println("FourthSingletonImpl");

}

public static FourthSingletonImpl getInstance() {

if (instance == null)

synchronized (FourthSingletonImpl.class) {

if (instance == null)

instance = new FourthSingletonImpl();

}

return instance;

}

}

*Main.java*

public class Main {

public static void main(String[] args) {

FirstSingletonImpl.getInstance();

SecondSingletonImpl.getInstance();

ThirdSingletonImpl.getInstance();

FourthSingletonImpl.getInstance();

}

}

***Результат выполнения программы***

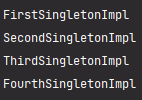


Рисунок 5.1 – Демонстрация работы программы

Практическая работа №6

***Цель работы***

Тема: Знакомство с реализацией порождающих паттернов проектирования.

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

***Листинг программы***

***AbstractFacrory:***

*AfricanGarden.java*

package AbstractFactory;

public class AfricanGarden implements GardenFactory {

public AfricanGarden() {

System.out.println("Создан африканский сад!");

}

@Override

public Tree plantTree() {

return new Baobab();

}

@Override

public Flower plantFlower() {

return new Gloriosa();

}

}

*Baobab.java*

package AbstractFactory;

public class Baobab implements Tree {

public Baobab() {

System.out.println("Посажено Дерево - Баобаб");

}

}

*EuropeanGarden.java*

package AbstractFactory;

public class EuropeanGarden implements GardenFactory {

public EuropeanGarden() {

System.out.println("Создан европейский сад!");

}

@Override

public Tree plantTree() {

return new Oak();

}

@Override

public Flower plantFlower() {

return new Rose();

}

}

*Flower.java*

package AbstractFactory;

public interface Flower {

}

*GardenFactory.java*

package AbstractFactory;

public interface GardenFactory {

Tree plantTree();

Flower plantFlower();

}

*Gloriosa.java*

package AbstractFactory;

public class Gloriosa implements Flower {

public Gloriosa() {

System.out.println("Посажен Цветок - Глориоза");

}

}

*HomeGarden.java*

package AbstractFactory;

public class HomeGarden implements GardenFactory {

public HomeGarden() {

System.out.println("Создан домашний сад");

}

@Override

public Tree plantTree() {

System.out.println("Вы не можете посадить дома дерево!");

return null;

}

@Override

public Flower plantFlower() {

return new Orchid();

}

}

*MainAbstractFactory.java*

package AbstractFactory;

public class MainAbstractFactory {

public static void main(String[] args) {

AfricanGarden ag = new AfricanGarden();

ag.plantFlower();

ag.plantTree();

System.out.print("\n");

EuropeanGarden eg = new EuropeanGarden();

eg.plantFlower();

eg.plantTree();

System.out.print("\n");

HomeGarden hg = new HomeGarden();

hg.plantFlower();

hg.plantTree();

}

}

*Oak.java*

package AbstractFactory;

public class Oak implements Tree {

public Oak() {

System.out.println("Посажено Дерево - Дуб");

}

}

*Orchid.java*

package AbstractFactory;

public class Orchid implements Flower {

public Orchid() {

System.out.println("Посажен Цветок - Орхидея");

}

}

*Rose.java*

package AbstractFactory;

public class Rose implements Flower {

public Rose() {

System.out.println("Посажен Цветок - Роза");

}

}

*Tree.java*

package AbstractFactory;

public interface Tree {

}

***Builder:***

*Animal.java*

package Builder;

public class Animal {

private final String species;

private final int weight;

private final String habitat;

private final int lifespan;

public static class AnimalBuilder {

private final String species;

private int weight = 0;

private String habitat = "";

private int lifespan = 0;

public AnimalBuilder(String species) {

this.species = species;

}

public AnimalBuilder weight(int weight) {

this.weight = weight;

return this;

}

public AnimalBuilder habitat(String habitat) {

this.habitat = habitat;

return this;

}

public AnimalBuilder lifespan(int lifespan) {

this.lifespan = lifespan;

return this;

}

public Animal create() {

return new Animal(this);

}

}

private Animal(AnimalBuilder builder) {

species = builder.species;

weight = builder.weight;

habitat = builder.habitat;

lifespan = builder.lifespan;

}

@Override

public String toString() {

return species + " обычно живет в " + habitat

+ ", весит примерно " + weight

+ " и живет " + lifespan + " лет.";

}

}

*MainBuilder.java*

package Builder;

public class MainBuilder {

public static void main(String[] args) {

Animal lion = new Animal

.AnimalBuilder("Лев")

.weight(150)

.habitat("Саванна")

.lifespan(12)

.create();

Animal polarBear = new Animal

.AnimalBuilder("Белый медведь")

.weight(400)

.habitat("Арктика")

.lifespan(25)

.create();

Animal redFox = new Animal

.AnimalBuilder("Обыкновенная лисица")

.weight(10)

.habitat("тундра, и степи, и леса разного типа, и пустыни и высокогорья")

.lifespan(4)

.create();

System.out.println(lion);

System.out.println(polarBear);

System.out.println(redFox);

}

}

***FactoryMethod:***

*AfricanCow.java*

package FactoryMethod;

public class AfricanCow extends Cow {

@Override

public String getType() {

return "Cow from Africa";

}

}

*AfricanLion.java*

package FactoryMethod;

public class AfricanLion extends Lion {

@Override

public String getType() {

return "Lion from Africa";

}

}

*AfricanOwl.java*

package FactoryMethod;

public class AfricanOwl extends Owl {

@Override

public String getType() {

return "Owl from Africa";

}

}

*AfricanZebra.java*

package FactoryMethod;

public class AfricanZebra extends Zebra {

@Override

public String getType() {

return "Zebra from Africa";

}

}

*AfricanZoo.java*

package FactoryMethod;

public class AfricanZoo extends Zoo {

@Override

protected Animal createAnimal(AnimalType type) {

Animal animal = null;

switch (type) {

case ZEBRA:

animal = new AfricanZebra();

break;

case LION:

animal = new AfricanLion();

break;

case COW:

animal = new AfricanCow();

break;

case OWL:

animal = new AfricanOwl();

break;

case BEAR:

animal = new RussianBear();

break;

}

return animal;

}

}

*Animal.java*

package FactoryMethod;

public interface Animal {

String getType();

}

*AnimalType.java*

package FactoryMethod;

public enum AnimalType {

ZEBRA,

LION,

COW,

OWL,

BEAR

}

*Bear.java*

package FactoryMethod;

public class Bear implements Animal {

@Override

public String getType() {

return "Bear";

}

}

*Cow.java*

package FactoryMethod;

public class Cow implements Animal {

@Override

public String getType() {

return "Cow";

}

}

*Lion.java*

package FactoryMethod;

public class Lion implements Animal {

@Override

public String getType() {

return "Lion";

}

}

*MainFactoryMethod.java*

package FactoryMethod;

public class MainFactoryMethod {

public static void main(String[] args) {

System.out.println("Zoo in Cape Town, Republic of South Africa");

Zoo capeTown = new AfricanZoo();

capeTown.buyNewAnimal(AnimalType.ZEBRA);

capeTown.buyNewAnimal(AnimalType.OWL);

capeTown.buyNewAnimal(AnimalType.COW);

capeTown.buyNewAnimal(AnimalType.LION);

capeTown.buyNewAnimal(AnimalType.BEAR);

System.out.println("\nZoo in Moscow, Russian Federation");

Zoo moscow = new RussianZoo();

moscow.buyNewAnimal(AnimalType.BEAR);

moscow.buyNewAnimal(AnimalType.OWL);

moscow.buyNewAnimal(AnimalType.COW);

moscow.buyNewAnimal(AnimalType.LION);

moscow.buyNewAnimal(AnimalType.ZEBRA);

}

}

*Owl.java*

package FactoryMethod;

public class Owl implements Animal {

@Override

public String getType() {

return "Owl";

}

}

*RussianBear.java*

package FactoryMethod;

public class RussianBear extends Bear {

@Override

public String getType() {

return "Bear from Russia";

}

}

*RussianCow.java*

package FactoryMethod;

public class RussianCow extends Cow {

@Override

public String getType() {

return "Cow from Russia";

}

}

*RussianLion.java*

package FactoryMethod;

public class RussianLion extends Lion {

@Override

public String getType() {

return "Lion from Russia";

}

}

*RussianOwl.java*

package FactoryMethod;

public class RussianOwl extends Owl {

@Override

public String getType() {

return "Owl from Russia";

}

}

*RussianZoo.java*

package FactoryMethod;

public class RussianZoo extends Zoo {

@Override

protected Animal createAnimal(AnimalType type) {

Animal animal = null;

switch (type) {

case BEAR:

animal = new RussianBear();

break;

case COW:

animal = new RussianCow();

break;

case OWL:

animal = new RussianOwl();

break;

case LION:

animal = new RussianLion();

break;

case ZEBRA:

animal = new AfricanZebra();

break;

}

return animal;

}

}

*Zebra.java*

package FactoryMethod;

public class Zebra implements Animal {

@Override

public String getType() {

return "Zebra from Africa";

}

}

*Zoo.java*

package FactoryMethod;

public abstract class Zoo {

public void buyNewAnimal(AnimalType type) {

Animal animal = createAnimal(type);

System.out.println("You just bought " + animal.getType() + " at the zoo!");

}

protected abstract Animal createAnimal(AnimalType type);

}

***Prototype:***

*Life.java*

package Prototype;

public class Life {

private Type type;

public Life() {

}

public Life(Type type) {

this.type = type;

}

public Life copy() {

return new Life();

}

public Type getType() {

return type;

}

public void setType(Type type) {

this.type = type;

}

@Override

public String toString() {

return "Life{" +

"type=" + type +

'}';

}

}

*Type.java*

package Prototype;

public enum Type {

BACTERIA,

FUNGUS,

ANIMAL,

PLANT

}

*MainPrototype.java*

package Prototype;

public class MainPrototype {

public static void main(String[] args) {

Life bacteria = new Life(Type.BACTERIA);

Life fungus = bacteria.copy();

fungus.setType(Type.FUNGUS);

System.out.println(bacteria);

System.out.println(fungus);

}

}

***Результат выполнения программы***

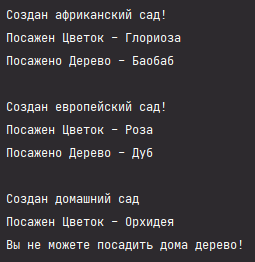


Рисунок 6.1 – Демонстрация работы программы Abstract Factory

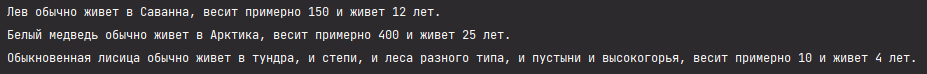


Рисунок 6.2 – Демонстрация работы программы Main Builder



Рисунок 6.3 – Демонстрация работы программы Factory Method



Рисунок 6.4 – Демонстрация работы программы Prototype

Практическая работа №7

***Цель работы***

Тема: Реализация структурных паттернов проектирования.

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом индивидуального задания Легковес, Заместитель.

***Листинг программы***

***Flyweight:***

*FirstHuman.java*

package Flyweight;

public class FirstHuman extends Human {

public FirstHuman() {

firstName = "Иван";

lastName = "Николаев-Аксенов";

age = 19;

}

@Override

public void getInfo() {

System.out.println(firstName + ' ' + lastName + ", возраст " + age + " лет");

}

}

*FlyweightFactory.java*

package Flyweight;

import java.util.HashMap;

public class FlyweightFactory {

private HashMap<Integer, Human> people = new HashMap<>();

public Human getHumanInfo(int number) {

Human human = people.get(number);

if (human == null) {

switch (number) {

case 1: {

human = new FirstHuman();

break;

}

case 2: {

human = new SecondHuman();

break;

}

case 3: {

human = new ThirdHuman();

break;

}

}

people.put(number, human);

}

return human;

}

}

*Human.java*

package Flyweight;

public abstract class Human {

protected String firstName;

protected String lastName;

protected int age;

public abstract void getInfo();

}

*MainFlyweight.java*

package Flyweight;

public class MainFlyweight {

public static void main(String[] args) {

FlyweightFactory factory = new FlyweightFactory();

int[] peopleList = {1, 2, 3, 3, 2};

for(int i: peopleList) {

Human h = factory.getHumanInfo(i);

h.getInfo();

}

}

}

*SecondHuman.java*

package Flyweight;

public class SecondHuman extends Human {

public SecondHuman() {

firstName = "Иван";

lastName = "Иванов";

age = 23;

}

@Override

public void getInfo() {

System.out.println(firstName + ' ' + lastName + ", возраст " + age + " лет");

}

}

*ThirdHuman.java*

package Flyweight;

public class ThirdHuman extends Human {

public ThirdHuman() {

firstName = "Петр";

lastName = "Петров";

age = 48;

}

@Override

public void getInfo() {

System.out.println(firstName + ' ' + lastName + ", возраст " + age + " лет");

}

}

***Proxy:***

*IUserChanger.java*

package Proxy;

import Flyweight.Human;

public interface IUserChanger {

void changeName(User user, String name);

void changeAge(User user, int age);

}

*MainProxy.java*

package Proxy;

public class MainProxy {

public static void main(String[] args) {

User a = new User("Ivan", 19);

User b = new User("Petr", 25);

User c = new User("root", 0);

UserChanger userChanger = new UserChanger();

ProxyUserChanger proxyUserChanger = new ProxyUserChanger();

userChanger.changeAge(a, 21);

userChanger.changeName(b, "Pavel");

proxyUserChanger.changeName(c, "Ivan&Pavel account");

System.out.println(a);

System.out.println(b);

System.out.println(c);

}

}

*ProxyUserChanger.java*

package Proxy;

public class ProxyUserChanger implements IUserChanger {

private UserChanger uc;

@Override

public void changeName(User user, String name) {

System.out.println("Proxy...");

UserChangerInitializer();

uc.changeName(user, name);

}

@Override

public void changeAge(User user, int age) {

System.out.println("Proxy...");

UserChangerInitializer();

uc.changeAge(user, age);

}

private void UserChangerInitializer() {

if(uc == null)

uc = new UserChanger();

}

}

*User.java*

package Proxy;

public class User {

private String name;

private int age;

public User(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return "Пользователь " + name + ", возраст " + age + " лет.";

}

}

*UserChanger.java*

package Proxy;

public class UserChanger implements IUserChanger {

@Override

public void changeName(User user, String name) {

user.setName(name);

}

@Override

public void changeAge(User user, int age) {

user.setAge(age);

}

}

***Результат выполнения программы***

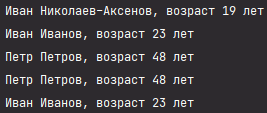


Рисунок 7.1 – Демонстрация работы программы Flyweight

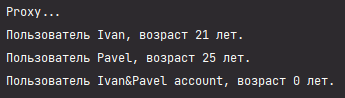


Рисунок 7.2 – Демонстрация работы программы Proxy

Практическая работа №8

***Цель работы***

Тема: Реализация поведенческих паттернов проектирования.

Постановка задачи: Написать реализацию паттерна в соответствии с вариантом индивидуального задания Наблюдатель, Состояние.

***Листинг программы***

***Observer:***

*CollageRating.java*

package Observer;

public class CollageRating implements Observer {

private String name;

private double gpa;

public void printInfo() {

System.out.println("Студент " + name + " имеет средний балл " + gpa);

}

@Override

public void update(String name, double gpa) {

this.name = name;

this.gpa = gpa;

printInfo();

}

}

*MainObserver.java*

package Observer;

public class MainObserver {

public static void main(String[] args) {

Student student = new Student();

Observer studentInfo = new CollageRating();

student.addObserver(studentInfo);

student.setNameAndGPA("Ivan", 7.9);

student.setName("Petr");

student.setGpa(5.2);

}

}

*Observable.java*

package Observer;

public interface Observable {

void addObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

*Observer.java*

package Observer;

interface Observer {

void update(String name, double gpa);

}

*Student.java*

package Observer;

import java.util.LinkedList;

import java.util.List;

public class Student implements Observable {

private String name;

private double gpa;

private List<Observer> obs;

public Student() {

obs = new LinkedList<>();

}

public void setName(String name) {

this.name = name;

notifyObservers();

}

public void setGpa(double gpa) {

this.gpa = gpa;

notifyObservers();

}

public void setNameAndGPA(String name, double gpa) {

this.name = name;

this.gpa = gpa;

notifyObservers();

}

@Override

public void addObserver(Observer o) {

obs.add(o);

}

@Override

public void removeObserver(Observer o) {

obs.remove(o);

}

@Override

public void notifyObservers() {

for(Observer o: obs)

o.update(name, gpa);

}

}

***State:***

*Bread.java*

package State;

public class Bread implements State {

private static final String name = "хлеб";

@Override

public String getName() {

return name;

}

@Override

public void make(StateInfo stateInfo) {

stateInfo.setState(new SandwichWithButter());

}

@Override

public void eat(StateInfo stateInfo) {

System.out.println("Сначала нужно приготовить бутерброд! Пока он на стадии: " + name);

}

}

*HalfASandwich.java*

package State;

public class HalfASandwich implements State {

private static final String name = "половина бутерброда";

@Override

public String getName() {

return name;

}

@Override

public void make(StateInfo stateInfo) {

System.out.println("Вы уже начали есть бутерброд!");

}

@Override

public void eat(StateInfo stateInfo) {

System.out.println("Вы съели бутерброд");

}

}

*MainState.java*

package State;

public class MainState {

public static void main(String[] args) {

StateInfo stateInfo = new StateInfo();

stateInfo.make();

System.out.println();

stateInfo.make();

System.out.println();

stateInfo.make();

System.out.println();

stateInfo.eat();

System.out.println();

stateInfo.eat();

}

}

*SandwichWithButter.java*

package State;

public class SandwichWithButter implements State {

private static final String name = "хлеб с маслом";

@Override

public String getName() {

return name;

}

@Override

public void make(StateInfo stateInfo) {

stateInfo.setState(new SandwichWithButterAndSausage());

}

@Override

public void eat(StateInfo stateInfo) {

System.out.println("Сначала нужно приготовить бутерброд! Пока он на стадии: " + name);

}

}

*SandwichWithButterAndSausage.java*

package State;

public class SandwichWithButterAndSausage implements State {

private static final String name = "бутерброд";

@Override

public String getName() {

return name;

}

@Override

public void make(StateInfo stateInfo) {

System.out.println("Вы уже приготовили бутерброд!");

}

@Override

public void eat(StateInfo stateInfo) {

stateInfo.setState(new HalfASandwich());

}

}

*State.java*

package State;

public interface State {

String getName();

void make(StateInfo stateInfo);

void eat(StateInfo stateInfo);

}

*StateInfo.java*

package State;

public class StateInfo {

private State state = new Bread();

public void make() {

System.out.println("Готовим бутерброд, текущее состояние " + state.getName());

state.make(this);

}

public void eat() {

System.out.println("Едим бутерброд, текущее состояние " + state.getName());

state.eat(this);

}

public void setState(State state) {

System.out.println("Изменяем состояние бутерброда на " + state.getName());

this.state = state;

}

public State getState() {

return state;

}

}

***Результат выполнения программы***

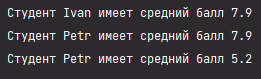


Рисунок 8.1 – Демонстрация работы программы Observer

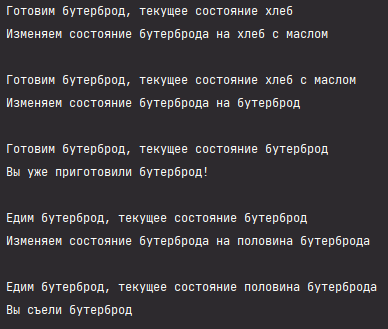


Рисунок 8.2 – Демонстрация работы программы State

Практическая работа №9

***Цель работы***

Тема: 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.

***Листинг программы***

*Main.java*

public class Main {

public static void main(String[] args) {

System.out.println("Hello World from Java from Gradle from Docker!");

}

}

*buld.gradle*

plugins {

id 'java'

}

version '1.0'

repositories {

mavenCentral()

}

jar {

manifest {

attributes(

'Class-Path': configurations.compile.collect { it.getName() }.join(' '),

'Main-Class': 'Main'

)

}

}

task toDocker {

doLast {

def stdout = new ByteArrayOutputStream()

println("Moving jar-file...")

ant.move file: "${buildDir}/libs/HelloWorldProject-1.0.jar", todir: "${projectDir}/Docker"

println("\nCreating Docker container...")

exec {

workingDir "${projectDir}/Docker"

commandLine 'docker', 'build', '-t', 'helloworld', '.'

}

println("\nLaunching Docker container...")

exec {

workingDir "${projectDir}/Docker"

commandLine 'docker', 'run', 'helloworld'

standardOutput = stdout

}

println "\nOutput from Docker container:\n$stdout"

}

}

build.finalizedBy(toDocker)

*Dockerfile*

FROM openjdk:11.0.10

WORKDIR /

ADD HelloWorldProject-1.0.jar HelloWorldProject-1.0.jar

EXPOSE 8080

CMD ["java","-jar","HelloWorldProject-1.0.jar"]

***Результат выполнения программы***

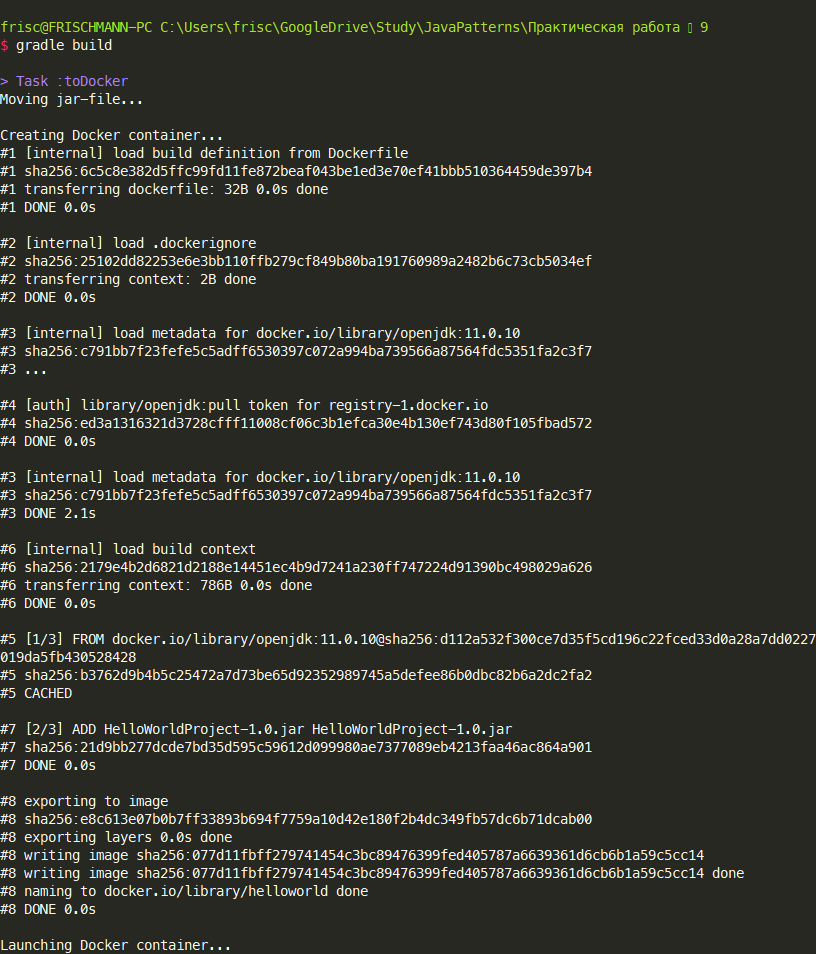


Рисунок 9.1 – Демонстрация работы программы

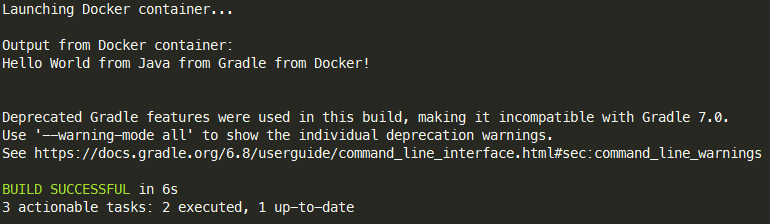


Рисунок 9.2 – Демонстрация работы программы

Практическая работа №10

***Цель работы***

Тема: 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 определяются в соответствии с вариантом индивидуального задания Интерфейс Fighter с методом doFight(), его имплементации: StreetFighter, Boxer, Judoka.

***Листинг программы***

*Boxer.java*

package App;

import org.springframework.stereotype.Component;

@Component

public class Boxer implements Fighter {

@Override

public void doFight() {

System.out.println("Boxer enters the ring");

}

}

*Fighter.java*

package App;

public interface Fighter {

void doFight();

}

*Judoka.java*

package App;

import org.springframework.stereotype.Component;

@Component

public class Judoka implements Fighter {

@Override

public void doFight() {

System.out.println("Judoka fighter enters the ring");

}

}

*StreetFighter.java*

package App;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext ac = new ClassPathXmlApplicationContext("fighters.xml");

Fighter f = (Fighter) ac.getBean(args[0]);

f.doFight();

}

}

*MainApp.java*

package App;

import org.springframework.stereotype.Component;

@Component

public class StreetFighter implements Fighter {

@Override

public void doFight() {

System.out.println("Street fighter enters the ring");

}

}

*Fighters.xml*

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="streetFighter" class="App.StreetFighter"/>

<bean id="boxer" class="App.Boxer"/>

<bean id="judoka" class="App.Judoka"/>

</beans>

***Результат выполнения программы***

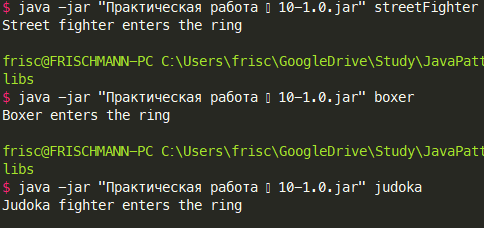


Рисунок 10.1 – Демонстрация работы программы

Практическая работа №11

***Цель работы***

Тема: 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.

***Листинг программы***

*TestRest.java*

package App.AppMain;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class TestRest {

@RequestMapping("/test")

public String open(){

return "Hello, World!";

}

}

*AppMainApplication.java*

package App.AppMain;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class AppMainApplication {

public static void main(String[] args) {

SpringApplication.run(AppMainApplication.class, args);

TestRest tr = new TestRest();

tr.open();

}

}

*Application.properties*

management.endpoint.auditevents.enabled**=**true

management.endpoint.beans.enabled**=**true

management.endpoint.caches.enabled**=**true

management.endpoint.conditions.enabled**=**true

management.endpoint.configprops.enabled**=**true

management.endpoint.env.enabled**=**true

management.endpoint.flyway.enabled**=**true

management.endpoint.health.enabled**=**true

management.endpoint.httptrace.enabled**=**true

management.endpoint.info.enabled**=**true

management.endpoint.integrationgraph.enabled**=**true

management.endpoint.loggers.enabled**=**true

management.endpoint.liquibase.enabled**=**true

management.endpoint.metrics.enabled**=**true

management.endpoint.mappings.enabled**=**true

management.endpoint.scheduledtasks.enabled**=**true

management.endpoint.sessions.enabled**=**true

management.endpoint.shutdown.enabled**=**true

management.endpoint.threaddump.enabled**=**true

***Результат выполнения программы***

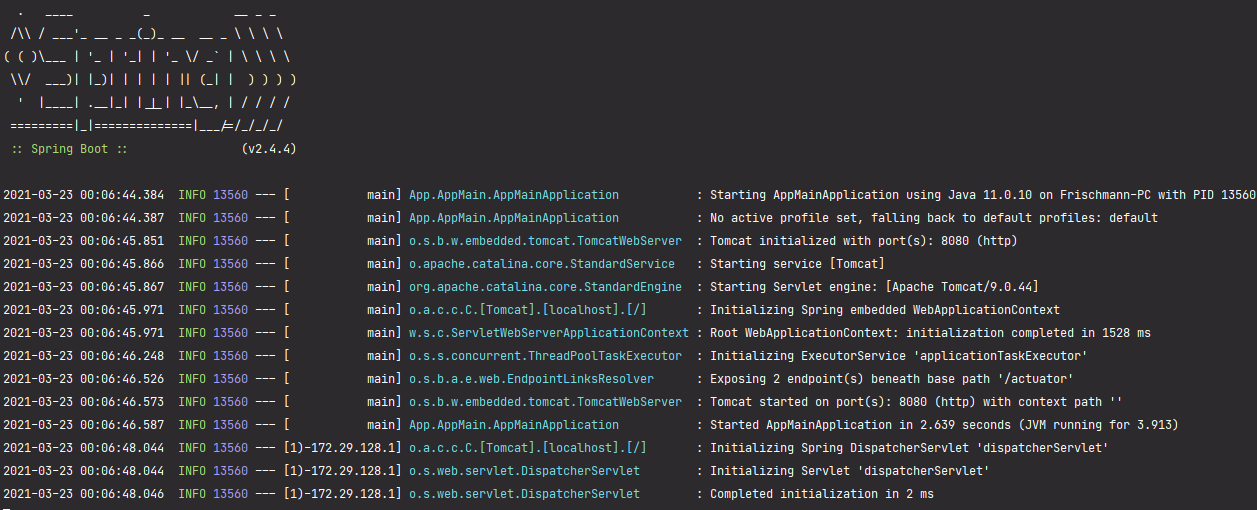


Рисунок 11.1 – Демонстрация работы программы

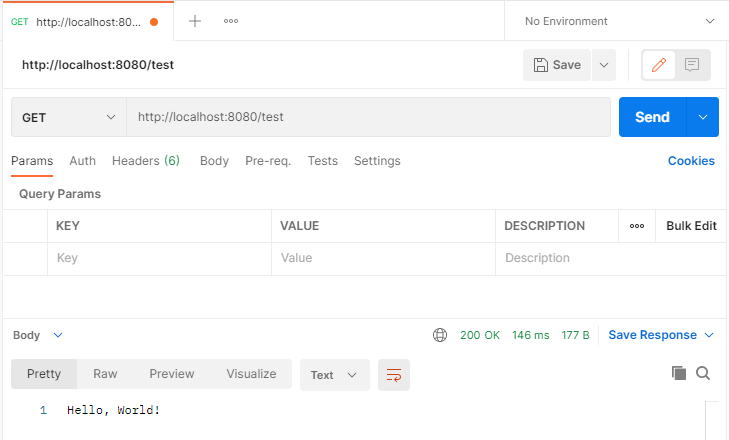


Рисунок 11.2 – Демонстрация работы программы

Практическая работа №12

***Цель работы***

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

Постановка задачи: Создать 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.

***Листинг программы***

*FileWorker.java*

package App.Main;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.ApplicationArguments;

import org.springframework.stereotype.Component;

import org.springframework.util.DigestUtils;

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

import java.io.\*;

@Component

public class FileWorker {

@Autowired

private ApplicationArguments arguments;

private String hashed;

public FileWorker(){

hashed = "";

}

@PostConstruct

public void init() throws IOException {

try(InputStream file = new FileInputStream(arguments.getNonOptionArgs().get(0))) {

hashed = DigestUtils.md5DigestAsHex(file);

File secondFile = new File(arguments.getNonOptionArgs().get(1));

if(!secondFile.exists())

secondFile.createNewFile();

FileWriter writer = new FileWriter(secondFile);

writer.write(hashed);

writer.close();

} catch (FileNotFoundException e) {

File file = new File(arguments.getNonOptionArgs().get(1));

if(!file.exists())

file.createNewFile();

FileWriter writer = new FileWriter(file);

writer.write("null");

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

}

@PreDestroy

public void deleteFirst(){

File file = new File(arguments.getNonOptionArgs().get(0));

if(file.exists()) {

file.delete();

}

}

}

*MainApplication.java*

package App.Main;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MainApplication {

private final FileWorker worker;

@Autowired

public MainApplication(FileWorker worker) {

this.worker = worker;

}

public static void main(String[] args) throws Exception {

SpringApplication.run(MainApplication.class, args);

}

}

***Результат выполнения программы***



Рисунок 12.1 – Демонстрация работы программы



Рисунок 12.2 – Демонстрация работы программы



Рисунок 12.2 – Демонстрация работы программы



Рисунок 12.3 – Демонстрация работы программы

Практическая работа №13

***Цель работы***

Тема: Конфигурирование приложения. Environment.

Постановка задачи: Создать файл application.yml в папке resources, добавить в него такие свойства:

* student.name – имя студента;
* student.last\_name – фамилия студента;
* student.group – название группы студента.

При запуске приложения выведите данные свойства в консоль при помощи интерфейса Environment или аннотации Value.

***Листинг программы***

*Student.java*

package App.Main;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

@Component

public class Student {

@Value("${student.name}")

private String name;

@Value("${student.last\_name}")

private String last\_name;

@Value("${student.group}")

private String group;

@PostConstruct

public void init() {

System.out.println("First name: " + name + "\nLast name: " + last\_name + "\nGroup: " + group);

}

}

*MainApplication.java*

package App.Main;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MainApplication {

public static void main(String[] args) {

SpringApplication.run(MainApplication.class, args);

}

}

*application.yml*

**student**:

**name**: Ivan

**last\_name**: Nikolaev-Axenov

**group**: ИКБО-20-19

***Результат выполнения программы***

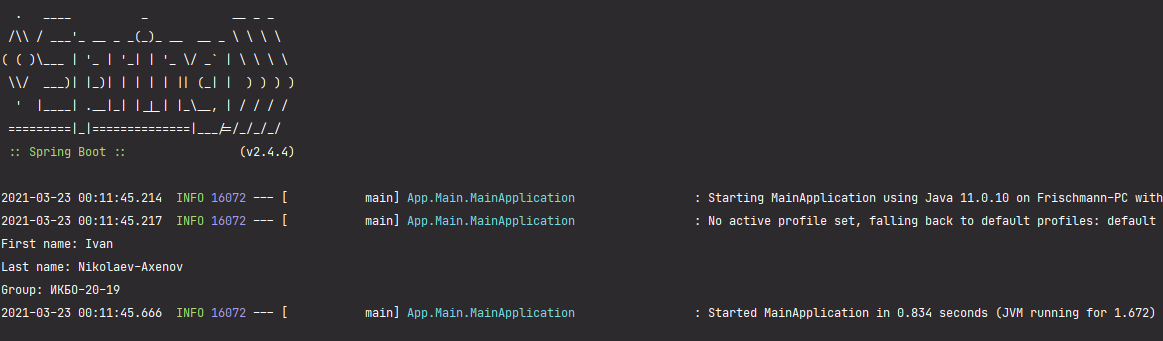


Рисунок 13.1 – Демонстрация работы программы

Вывод

В ходе выполнения данных практических работ были получены навыки работы с основными технологиями, необходимыми для создания клиент-серверных приложений. Также были получены навыки работы с фреймворком Spring.

Список использованных источников

1. Стелтинг С., Маасен О. Применение шаблонов Java. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 576 с.: ил. — Парал. тит. англ.
2. Functional Interfaces in Java: Fundamentals and Examples 1st ed. Edition, Kindle Edition [Электронный ресурс]. URL: https://www.amazon.com/Functional-Interfaces-Java-Fundamentals-Examples-ebook/dp/B07NRHQSCW (дата обращения: 29.01.21). Заголовок с экрана.
3. Hibernate Search 6.0.0.Final: Reference Documentation [Электронный ресурс]. URL: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html\_single/ (дата обращения: 29.01.21). Заголовок с экрана.
4. Паттерны проектирования на Java. Каталог Java-примеров. [Электронный ресурс]. URL: https://refactoring.guru/ru/design-patterns/java (дата обращения: 29.01.21). Заголовок с экрана.
5. Руководство по Spring [Электронный ресурс]. URL: https://proselyte.net/tutorials/spring-tutorial-full-version/ (дата обращения: 29.01.21). Заголовок с экрана.
6. The Reactive Manifesto [Электронный ресурс]. URL: https://www.reactivemanifesto.org/ (дата обращения: 29.01.21). Заголовок с экрана.
7. Spring Framework Documentation [Электронный ресурс]. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения: 29.01.21). Заголовок с экрана.
8. Hibernate Search 6.0.0. Final: Reference Documentation [Электронный ресурс]. URL: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html\_single/ (дата обращения: 29.01.21). Заголовок с экрана.