|  |
| --- |
|  |
| 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** | ***Горохов С.В.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

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

Москва 2022

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

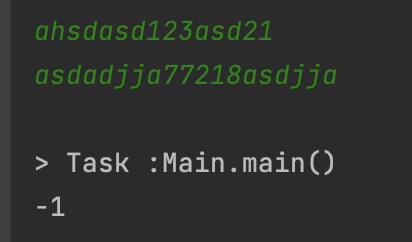
Тема: Знакомство со встроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

Постановка задачи: Имплементировать интерфейс Predicate, определяющий, является ли данная строка email-адресом, используя регулярное выражение.

1. **Код и пояснение к коду**

package org.prac\_1;  
  
import java.util.Comparator;  
import java.util.Scanner;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class Main {  
 public static void main(String[] args) {  
 Comparator<String> myComparator = (String str, String str2) -> {  
 Pattern pattern = Pattern.*compile*("[0-9]+");  
 int str1NumCounter = 0;  
 Matcher matcherString1 = pattern.matcher(str);  
 while (matcherString1.find()) {  
 str1NumCounter += Integer.*parseInt*(matcherString1.group(0));  
 }  
 int str2NumCounter = 0;  
 Matcher matcherString2 = pattern.matcher(str2);  
 while (matcherString2.find()) {  
 str2NumCounter += Integer.*parseInt*(matcherString2.group(0));  
 }  
 System.*out*.println(str1NumCounter);  
 return Integer.*compare*(str1NumCounter, str2NumCounter);  
 };  
 Scanner scanner = new Scanner(System.*in*);  
 String input1 = scanner.nextLine();  
 String input2 = scanner.nextLine();  
 System.*out*.println(myComparator.compare(input1, input2));  
 }  
}

1. **Результат работы программы**



**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

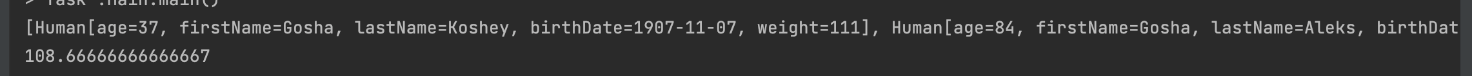
Тема: Работа со Stream API в Java 8.

Постановка задачи: Сортировка по второй букве имени в обратном порядке, фильтрация по весу больше, чем 60, сортировка по возрасту, произведение всех возрастов

1. **Код и пояснение к коду**

package org.prac2;  
  
import java.time.LocalDate;  
import java.util.\*;  
  
public class Main {  
  
 static List<Human> generateList(int n) {  
 Random random = new Random();  
 List<String> names = Arrays.*asList*("Stepa", "Gosha", "Loh", "Nikita", "Andrey");  
 List<String> lastNames = Arrays.*asList*("Goroh", "Tilt", "Aleks", "Korol", "Koshey");  
 List<Human> humans = new ArrayList<>();  
 for (int i = 0; i < n; i++) {  
 String currentName = names.get(random.nextInt(0, 5));  
 String currentLastName = lastNames.get(random.nextInt(0,5));  
 int age = random.nextInt(1, 100);  
 LocalDate birthDate = LocalDate.*of*(random.nextInt(1900, 2000), random.nextInt(1, 12), random.nextInt(1, 28));  
 int weight = random.nextInt(100, 120);  
 humans.add(new Human(  
 age,  
 currentName,  
 currentLastName,  
 birthDate,  
 weight  
 ));  
 }  
 return humans;  
 }  
 public static void main(String[] args) {  
 List<Human> humanList = *generateList*(10);  
 System.*out*.println(humanList);  
 OptionalDouble value = humanList.stream().sorted((o1, o2) -> -Integer.*compare*(o1.age(), o2.age()))  
 .filter((h1) -> h1.firstName().startsWith("A"))  
 .sorted(Comparator.*comparing*(Human::birthDate))  
 .mapToDouble(Human::weight)  
 .average();  
 if (value.isPresent()) {  
 System.*out*.println(value.getAsDouble());  
 }  
 }  
}

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 в соответствии с вариантом индивидуального задания: Set с использованием Semaphore, List с использованием ключевого слова synchronized.

1. **Код и пояснение к коду**

package org.prac3;  
  
import java.util.\*;  
import java.util.concurrent.ConcurrentHashMap;  
import java.util.concurrent.Semaphore;  
  
public class MySet implements Set<Integer> {  
 private Integer[] values = new Integer[5];  
 private final Semaphore semaphore = new Semaphore(1);  
 private int lastIndex = 0;  
  
 @Override  
 public int size() {  
 return values.length;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return lastIndex == 0;  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 try {  
 semaphore.acquire();  
 for (int i = 0; i < lastIndex; i++) {  
 if (o == values[i]) {  
 semaphore.release();  
 return true;  
 }  
 }  
 semaphore.release();  
 return false;  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 @Override  
 public Iterator<Integer> iterator() {  
 return new Iterator<>() {  
 private int current = 0;  
 @Override  
 public boolean hasNext() {  
 return current < lastIndex;  
 }  
  
 @Override  
 public Integer next() {  
 try {  
 semaphore.acquire();  
 if (current < lastIndex) {  
 Integer nextValue = values[current];  
 current++;  
 return nextValue;  
 } else {  
 throw new RuntimeException("Ты тронул это зря!!!");  
 }  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 semaphore.release();  
 }  
 }  
 };  
 }  
  
 @Override  
 public Object[] toArray() {  
 return values.clone();  
 }  
  
 @Override  
 public <T> T[] toArray(T[] a) {  
 return a;  
 }  
  
 @Override  
 public boolean add(Integer integer) {  
 try {  
 semaphore.acquire();  
 if (values.length <= lastIndex) {  
 extendArray();  
 }  
 if (!contains(integer)) {  
 values[lastIndex] = integer;  
 return true;  
 }  
 return false;  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 semaphore.release();  
 }  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 try {  
 semaphore.acquire();  
 int currentIndex = -1;  
 boolean found = false;  
 for (int i = 0; i < lastIndex; i++) {  
 if (values[i] != o && !found) {  
 currentIndex+=1;  
 found = true;  
 }  
 if (currentIndex >= 0) {  
 values[i] = values[currentIndex];  
 }  
 }  
 return found;  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 semaphore.release();  
 }  
 }  
  
 @Override  
 public boolean containsAll(Collection<?> c) {  
 try {  
 semaphore.acquire();  
 for (Object o: c) {  
 if (!contains(o)) {  
 return false;  
 }  
 }  
 return true;  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 semaphore.release();  
 }  
 }  
  
 @Override  
 public boolean addAll(Collection<? extends Integer> c) {  
 for (Integer i: c) {  
 add(i);  
 }  
 return true;  
 }  
  
 @Override  
 public boolean retainAll(Collection<?> c) {  
 try {  
 semaphore.acquire();  
 int removedCounter = 0;  
 for (int i = 0; i < values.length; i++) {  
 if (!c.contains(values[i-removedCounter])) {  
 remove(values[i-removedCounter]);  
 removedCounter++;  
 }  
 }  
 return true;  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 semaphore.release();  
  
 }  
 }  
  
 @Override  
 public boolean removeAll(Collection<?> c) {  
 for (Object o: c) {  
 remove(o);  
 }  
 return true;  
 }  
  
 @Override  
 public void clear() {  
 values = new Integer[5];  
 lastIndex = 0;  
 }  
  
 private void extendArray() {  
 values = Arrays.*copyOf*(values, values.length\*2);  
 }  
}

package org.prac3;  
  
import org.jetbrains.annotations.NotNull;  
  
import java.util.\*;  
  
public class MyList<T> implements List<T> {

private List<T> innerList = new ArrayList<>();  
 @Override  
 public int size() {  
 return innerList.size();  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return innerList.isEmpty();  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 return innerList.contains(o);  
 }  
  
 @NotNull  
 @Override  
 public Iterator<T> iterator() {  
 return innerList.iterator();  
 }  
  
 @NotNull  
 @Override  
 public Object[] toArray() {  
 return innerList.toArray();  
 }  
  
 @NotNull  
 @Override  
 public <T1> T1[] toArray(@NotNull T1[] a) {  
 return innerList.toArray(a);  
 }  
  
 @Override  
 public boolean add(T t) {  
 return innerList.add(t);  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 return innerList.remove(o);  
 }  
  
 @Override  
 public boolean containsAll(@NotNull Collection<?> c) {  
 return innerList.containsAll(c);  
 }  
  
 @Override  
 public boolean addAll(@NotNull Collection<? extends T> c) {  
 return innerList.addAll(c);  
 }  
  
 @Override  
 public boolean addAll(int index, @NotNull Collection<? extends T> c) {  
 return innerList.addAll(index, c);  
 }  
  
 @Override  
 public boolean removeAll(@NotNull Collection<?> c) {  
 return innerList.removeAll(c);  
 }  
  
 @Override  
 public boolean retainAll(@NotNull Collection<?> c) {  
 return innerList.removeAll(c);  
 }  
  
 @Override  
 public void clear() {  
 innerList.clear();  
 }  
  
 @Override  
 public T get(int index) {  
 return innerList.get(index);  
 }  
  
 @Override  
 public T set(int index, T element) {  
 return innerList.set(index, element);  
 }  
  
 @Override  
 public void add(int index, T element) {  
 innerList.add(index, element);  
 }  
  
 @Override  
 public T remove(int index) {  
 return innerList.remove(index);  
 }  
  
 @Override  
 public int indexOf(Object o) {  
 return innerList.indexOf(o);  
 }  
  
 @Override  
 public int lastIndexOf(Object o) {  
 return innerList.lastIndexOf(o);  
 }  
  
 @NotNull  
 @Override  
 public ListIterator<T> listIterator() {  
 return innerList.listIterator();  
 }  
  
 @NotNull  
 @Override  
 public ListIterator<T> listIterator(int index) {  
 return innerList.listIterator(index);  
 }  
  
 @NotNull  
 @Override  
 public List<T> subList(int fromIndex, int toIndex) {  
 return innerList.subList(fromIndex, toIndex);  
 }  
}

**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Тема: Работа с ExecutorService, CompletableFuture.

Постановка задачи:

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

1. **Код и пояснение к коду**

package org.prac4;  
  
import java.util.Collection;  
import java.util.List;  
import java.util.concurrent.\*;  
class MyExecutorService implements ExecutorService {  
 private ExecutorService executorService;  
  
 public MyExecutorService(int threadCount) {  
 executorService = Executors.*newFixedThreadPool*(threadCount);  
 }  
  
 @Override  
 public void shutdown() {  
 executorService.shutdown();  
 }  
  
 @Override  
 public List<Runnable> shutdownNow() {  
 return executorService.shutdownNow();  
 }  
  
 @Override  
 public boolean isShutdown() {  
 return executorService.isShutdown();  
 }  
  
 @Override  
 public boolean isTerminated() {  
 return executorService.isTerminated();  
 }  
  
 @Override  
 public boolean awaitTermination(long timeout, TimeUnit unit) throws InterruptedException {  
 return executorService.awaitTermination(timeout, unit);  
 }  
  
 @Override  
 public <T> Future<T> submit(Callable<T> task) {  
 return executorService.submit(task);  
 }  
  
 @Override  
 public <T> Future<T> submit(Runnable task, T result) {  
 return executorService.submit(task, result);  
 }  
  
 @Override  
 public Future<?> submit(Runnable task) {  
 return executorService.submit(task);  
 }  
  
 @Override  
 public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws InterruptedException {  
 return executorService.invokeAll(tasks);  
 }  
  
 @Override  
 public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit) throws InterruptedException {  
 return executorService.invokeAll(tasks, timeout, unit);  
 }  
  
 @Override  
 public <T> T invokeAny(Collection<? extends Callable<T>> tasks) throws InterruptedException, ExecutionException {  
 return executorService.invokeAny(tasks);  
 }  
  
 @Override  
 public <T> T invokeAny(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException {  
 return executorService.invokeAny(tasks, timeout, unit);  
 }  
  
 @Override  
 public void execute(Runnable command) {  
 executorService.execute(command);

}

**Практическая работа №5**

1. **Постановка задачи и персональный вариант**

Тема: 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 и пояснение к коду**

package org.prac5;  
  
public class MySingleton1 {  
 private static MySingleton1 *instance*;  
  
 public static MySingleton1 getInstance() {  
 if (*instance* == null){  
 synchronized (MySingleton1.class) {  
 if (*instance* != null) {  
 *instance* = new MySingleton1();  
 }  
 }  
 }  
 return *instance*;  
 }  
}

package org.prac5;  
  
public enum MySingleton2 {  
 *INSTANCE*  
}

package org.prac5;  
  
public class MySingleton3 {  
 private static final MySingleton3 *instance* = new MySingleton3();  
  
 public static MySingleton3 getInstance() {  
 return *instance*;  
 }  
}

**Практическая работа №6**

1. **Постановка задачи и персональный вариант**

Тема: знакомство с реализацией порождающих паттернов проектирования

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

1. **Код и пояснение к коду**

AbstractFactory

Main.java

package org.prac6.abstractFactory;  
  
public class Main {  
 public static void main(String[] args) {  
 AbstractFactory factory = new GoshaFactory();  
 System.*out*.println(factory.getCat());  
 System.*out*.println(factory.getCat());  
 System.*out*.println(factory.getDog());  
 }  
}

Builder

Main.java

package org.prac6.builder;  
  
public class Main {  
 public static void main(String[] args) {  
 GoshaDogBuilder builder = new GoshaDogBuilder();  
 builder.setMmr(123);  
 builder.setPts(123);  
 System.*out*.println(builder.build().toString());  
 }  
}

FactoryMethod

Main.java

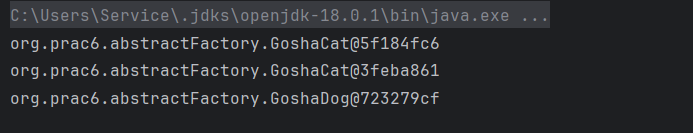
package org.prac6.factory;  
  
  
public class Main {  
 public static void main(String[] args) {  
 DogFactory factory = new DogFactory();  
 System.*out*.println(factory.createDog("Gosha"));  
 }  
}

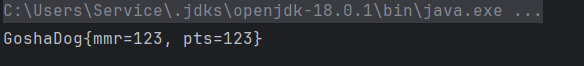
Prototype

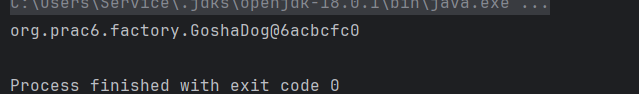
Main.java

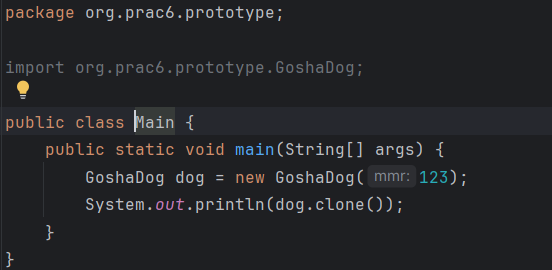
package org.prac6.prototype;  
  
import org.prac6.prototype.GoshaDog;  
  
public class Main {  
 public static void main(String[] args) {  
 GoshaDog dog = new GoshaDog(123);  
 System.*out*.println(dog.clone());  
 }  
}

1. **Результаты вывода**









Ссылка на репозиторий: https://github.com/Stepoik/java-prac

**Практическая работа №7**

1. **Постановка задачи и персональный вариант**

Тема: реализация структурных паттернов проектирования

Постановка задачи: написать реализацию паттерна в соответствии с вариантом индивидуального задания. Индивидуальное задание: Фасад, Легковес.

1. **Код и пояснение к коду**

Фасад

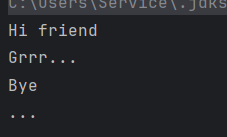
package org.prac7.facade;  
  
public class Facade {

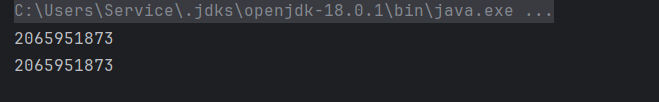
private Friend friend = new Friend();  
 private Enemy enemy = new Enemy();  
  
 public void sayHelloToPeople() {  
 friend.sayHello();  
 enemy.sayHello();  
 }  
  
 public void sayByeToPeople() {  
 friend.sayBye();  
 enemy.sayBye();  
 }  
}  
  
class Friend {  
 public void sayHello() {  
 System.*out*.println("Hi friend");  
 }  
  
 public void sayBye() {  
 System.*out*.println("Bye");  
 }  
}  
  
class Enemy {  
 public void sayHello() {  
 System.*out*.println("Grrr...");  
 }  
  
 public void sayBye() {  
 System.*out*.println("...");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Facade facade = new Facade();  
 facade.sayHelloToPeople();  
 facade.sayByeToPeople();  
 }  
}

Легковес

package org.prac7.flyweight;  
  
import java.util.HashMap;  
  
public class Flyweight {  
 private String information;  
 public Flyweight(String someInformation) {  
 this.information = someInformation;  
 }  
  
 @Override  
 public String toString() {  
 return "Flyweight{" +  
 "information='" + information + '\'' +  
 '}';  
 }  
}  
  
class Factory {  
 private final HashMap<String, Flyweight> cash = new HashMap<>();  
 public Flyweight getFlyweight(String information) {  
 if (!cash.containsKey(information)) {  
 cash.put(information, new Flyweight(information));  
 }  
 return cash.get(information);  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Factory factory = new Factory();  
 System.*out*.println(System.*identityHashCode*(factory.getFlyweight("stepa")));  
 System.*out*.println(System.*identityHashCode*(factory.getFlyweight("stepa")));  
 }  
}

**Результаты выполнения программы**





Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №8**

1. **Постановка задачи и персональный вариант**

Тема: реализация поведенческих паттернов проектирования

Постановка задачи: написать реализацию паттерна в соответствии с вариантом индивидуального задания. Индивидуальное задание: Снимок, Наблюдатель.

1. **Код и пояснение к коду**

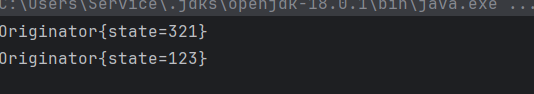
Снимок

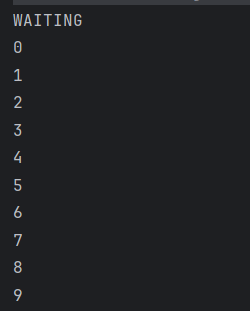
package org.prac8.memento;  
  
class Originator {  
 private int state;  
  
 public Originator(int startState) {  
 state = startState;  
 }  
  
 public void setMemento(Memento m) {  
 state = m.state;  
 }  
  
 public Memento createMemento() {  
 Memento memento = new Memento();  
 memento.state = state;  
 return memento;  
 }  
  
 @Override  
 public String toString() {  
 return "Originator{" +  
 "state=" + state +  
 '}';  
 }  
  
 public static class Memento {  
 private int state;  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Originator originator = new Originator(123);  
 Originator.Memento memento = originator.createMemento();  
 Originator otherOriginator = new Originator(321);  
 System.*out*.println(otherOriginator);  
 otherOriginator.setMemento(memento);  
 System.*out*.println(otherOriginator);  
 }  
}

Наблюдатель

package org.prac8.observer;  
  
import java.util.HashSet;  
import java.util.regex.Pattern;  
  
interface Observer<T> {  
 void update(T value);  
}  
  
class ConcreteObserver implements Observer<Integer> {  
 private Integer value;  
  
 @Override  
 public void update(Integer value) {  
 System.out.println(value);  
 this.value = value;  
 }  
}  
  
interface Observable<T> {  
 void attach(Observer<T> observer);  
  
 void detach(Observer<T> observer);  
  
 void notify(T value);  
}  
  
  
class ConcreteObservable implements Observable<Integer> {  
 private HashSet<Observer<Integer>> observers = new HashSet<>();  
  
 @Override  
 public void attach(Observer<Integer> observer) {  
 observers.add(observer);  
 }  
  
 @Override  
 public void detach(Observer<Integer> observer) {  
 observers.remove(observer);  
 }  
  
 @Override  
 public void notify(Integer value) {  
 for (Observer<Integer> observer : observers) {  
 observer.update(value);  
 }  
 }  
  
 public Thread startThread() {  
 Thread thread = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 notify(i);  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 });  
 thread.start();  
 return thread;  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 ConcreteObservable observable = new ConcreteObservable();  
 observable.attach(new ConcreteObserver());  
 observable.startThread();  
 System.out.println("WAITING");  
 }  
}

1. **Результат выполнения программы**





Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №9**

1. **Постановка задачи и персональный вариант**

Тема: знакомство с системой сборки приложения. Gradle

Постановка задачи: cоздать 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 и пояснение к коду**

build.gradle

plugins **{**  
id 'java'  
**}**  
  
group 'org.prac\_1'  
version '1.0-SNAPSHOT'  
  
repositories **{**  
mavenCentral()  
**}**  
  
dependencies **{**  
testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'  
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'  
**}**  
  
plugins **{** id 'java'  
**}**group 'org.prac\_1'  
version '1.0-SNAPSHOT'  
  
repositories **{** mavenCentral()  
**}**dependencies **{** testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'  
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'  
**}**test **{** useJUnitPlatform()  
**}**jar **{** archiveName "stepa.jar"  
 manifest **{** attributes(  
 "Main-Class": "org.prac\_1.Main"  
 )  
 **}  
}**tasks.register("copyDockerJar",Copy) **{** dependsOn jar  
 from jar  
 into "docker"  
**}**tasks.register("runDocker") **{** dependsOn "copyDockerJar"  
 doLast **{** exec **{** commandLine "docker", "build", "-t", "haha", "./docker"  
 **}** exec **{** commandLine "docker", "run", "--rm", "haha"  
 **}  
 }  
}**runDocker **{  
  
}**

Main.java

package org.prac\_1;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Stepa horoshi");  
 }  
}

Dockerfile

from openjdk:17  
ADD stepa.jar /  
CMD java -jar stepa.jar

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №10**

1. **Постановка задачи и персональный вариант**

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.

Персональный вариант: интерфейс Musician с методом void doCoding(), его имплементации: Drummer, Guitarist, Trombonist.

1. **Код и пояснение к коду**

Musician.java

package com.example.prac10;  
  
public interface Musician {  
 void doCoding();  
}

Guitarist.java

package com.example.prac10;  
  
import org.springframework.stereotype.Component;  
  
@Component(value = "Guitarist")  
public class Guitarist implements Musician {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Guitarist coding");  
 }  
}

Drummer.java

package com.example.prac10;  
  
import org.springframework.stereotype.Component;  
  
@Component(value = "Drummer")  
public class Drummer implements Musician {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Drummer coding");  
 }  
}

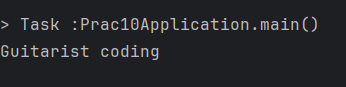
Trombonist.java

package com.example.prac10;  
  
import org.springframework.stereotype.Component;  
  
@Component(value = "Trombonist")  
public class Trombonist implements Musician {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Trombonist coding");  
 }  
}

Prac10Application.java

package com.example.prac10;  
  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
@SpringBootApplication  
public class Prac10Application {  
  
 public static void main(String[] args) {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
 ((Musician)context.getBean(args[0])).doCoding();  
 }  
  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №11**

1. **Постановка задачи и персональный вариант**

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.

1. **Код и пояснение к коду**

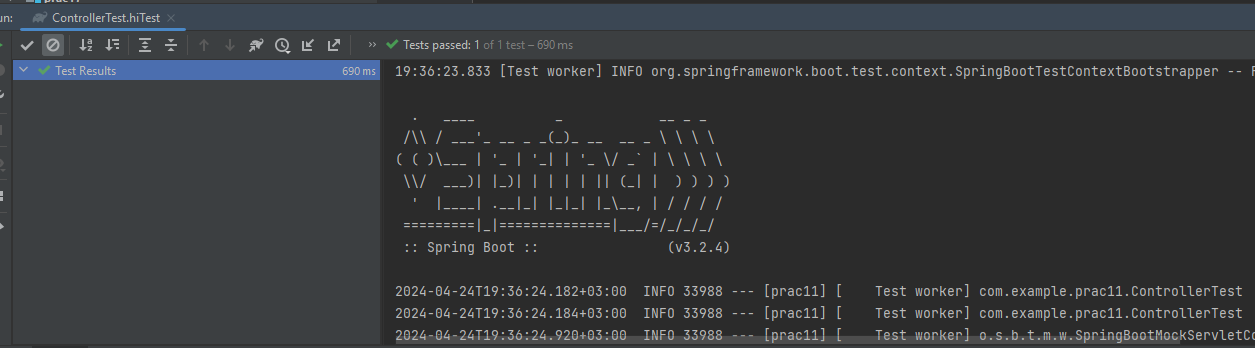
Conroller.java

@RestController  
public class Controller {  
 @GetMapping("/hi")  
 public String hi() {  
 return "privet";  
 }  
}

ControllerTest.java

@WebMvcTest  
public class ControllerTest {  
 @Autowired  
 MockMvc mock;  
  
 @Test  
 public void hiTest() throws Exception {  
 mock.perform(MockMvcRequestBuilders.*get*("/hi")).andExpect(result ->  
 Assertions.*assertEquals*( "privet", result.getResponse().getContentAsString()));  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №12**

1. **Постановка задачи и персональный вариант**

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

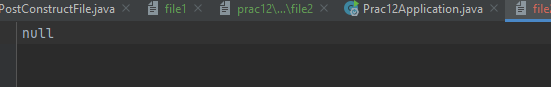
Постановка задачи: создать приложение, которое при запуске берет данные из одного файла, хеширует, а при остановке приложения удаляет исходный файл, оставляя только файл с захешированными данными. Названия первого и второго файла передаются в качестве аргументов при запуске. При отсутствии первого файла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

1. **Код и пояснение к коду**

PostConstructFile.java

@Component(value = "deleter")  
public class PostConstructFile {  
  
 @PostConstruct  
 public void init() {  
 String firstPath = FilePathStorage.*getFirstPath*();  
 String secondPath = FilePathStorage.*getSecondPath*();  
 try {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(new FileInputStream(firstPath)));  
 StringBuilder data = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 data.append(line);  
 }  
 Writer writer = new BufferedWriter(new OutputStreamWriter(  
 new FileOutputStream(secondPath), "utf-8"));  
 writer.write(data.hashCode());  
 reader.close();  
 writer.close();  
 } catch (IOException e) {  
 try {  
 Writer writer = new BufferedWriter(new OutputStreamWriter(  
 new FileOutputStream(secondPath), "utf-8"));  
 writer.write("null");  
 writer.close();  
 } catch (Exception er) {  
 System.*out*.println("Вообще ничего не получилось");  
 }  
 }  
 }  
  
 @PreDestroy  
 public void destroy() {  
 String firstPath = FilePathStorage.*getFirstPath*();  
 File fileToDelete = new File(firstPath);  
 System.*out*.println(fileToDelete.delete());  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №13**

1. **Постановка задачи и персональный вариант**

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.

1. **Код и пояснение к коду**

Student.java

import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Student {  
 @Value("${student.name}")  
 private String name;  
  
 @Value("${student.last\_name}")  
 private String lastName;  
  
 @Value("${student.group}")  
 private String group;  
 @Autowired  
 Environment env;  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", group='" + group + '\'' +  
 '}';  
 }  
  
 @PostConstruct  
 public void init() {  
 System.*out*.println(this);  
 }  
}

application.yml

program:  
 student:  
 name: Stepa  
 last\_name: Gorokhov  
 group: IKBO-20-22

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №14**

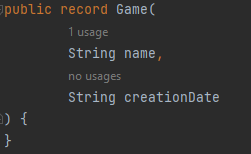
1. **Постановка задачи и персональный вариант**

Тема: знакомство со Spring MVC. Работа с Rest API в Spring.

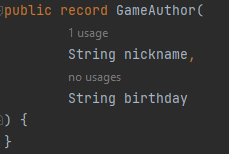
Постановка задачи: Создать класс Game с полями name, creationDate. Создать класс GameAuthor с полями nickname, birthDate. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти

1. **Код пояснение к коду**

Game.java



GameAuthor.java



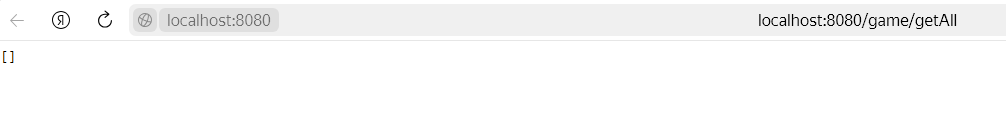
GameAuthorController.java

@Controller()  
@RequestMapping("/gameAuthor")  
public class GameAuthorController {  
 @Autowired  
 private GameAuthorRepository gameAuthorRepository;  
  
 @PostMapping("/create")  
 public @ResponseBody String create(@RequestBody GameAuthor game) {  
 gameAuthorRepository.addGameAuthor(game);  
 return "Success";  
 }  
  
 @DeleteMapping("/delete")  
 public @ResponseBody String remove(@RequestBody DeleteRequestBody deleteRequestBody) {  
 gameAuthorRepository.removeGameAuthorByName(deleteRequestBody.deleteValue());  
 return "Success";  
 }  
  
 @GetMapping("/getAll")  
 public @ResponseBody List<GameAuthor> getAllGames() {  
 return gameAuthorRepository.getGameAuthors();  
 }  
}

GameController.java

@Controller()  
@RequestMapping("/game")  
public class GameController {  
 @Autowired  
 private GameRepository gameRepository;  
  
 @PostMapping("/create")  
 public @ResponseBody String create(@RequestBody Game game) {  
 gameRepository.addGame(game);  
 return "Success";  
 }  
  
 @DeleteMapping("/delete")  
 public @ResponseBody String remove(@RequestBody DeleteRequestBody deleteRequestBody) {  
 gameRepository.removeGameByName(deleteRequestBody.deleteValue());  
 return "Success";  
 }  
  
 @GetMapping("/getAll")  
 public @ResponseBody List<Game> getAllGames() {  
 return gameRepository.getGames();  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №15**

1. **Постановка задачи и персональный вариант**

Тема: использование Hibernate в Spring framework.

Постановка задачи: изменить программу с предыдущего задания так, чтобы объекты хранились в базе данных PostgreSQL вместо памяти компьютераи.

1. **Код пояснение к коду**

Game.java

@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
public class Game {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name="name")  
 private String name;  
 @Column(name="creation\_date")  
 private String creationDate;  
}

GameRepository.java

@Component  
public class GameRepository {  
  
 @Autowired  
 private SessionFactory sessionFactory;  
  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
  
 public void addGame(Game game) {  
 session.beginTransaction();  
 session.persist(game);  
 session.flush();  
 session.getTransaction().commit();  
 }  
  
 public void removeGameByName(String name) {  
 String hql = "delete Game where name=:name";  
 session.createQuery(hql, Game.class).setParameter("name", name);  
 }  
  
 public List<Game> getGames() {  
 return session.createQuery("from Game", Game.class).getResultList();  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №16**

1. **Постановка задачи и персональный вариант**

Тема: использование изучение видов связей между сущностями в Hibernate. Использование транзакций

Постановка задачи: создать связь Один-ко-многим между сущностями из предыдущего задания и проверить работу lazy loading.

1. **Код пояснение к коду**

Game.java

@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
public class Game {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name="name")  
 private String name;  
 @Column(name="creation\_date")  
 private String creationDate;  
  
 @ManyToOne  
 @JsonIgnore  
 private GameAuthor gameAuthor;  
}

GameAuthor.java

@Entity  
@Setter  
@Getter  
@NoArgsConstructor  
public class GameAuthor {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(name = "nickname")  
 private String nickname;  
  
 @Column(name = "birthday")  
 private String birthday;  
  
 @OneToMany(mappedBy = "gameAuthor")  
 private List<Game> games;  
}

GameRequestBody.java

public record GameRequestBody(  
 String name,  
 String creationDate,  
 Long authorId  
) {  
}

GameRepository.java

@Component  
public class GameRepository {  
 @Autowired  
 private SessionFactory sessionFactory;  
  
 @Autowired  
 private GameAuthorRepository gameAuthorRepository;  
  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
  
 public void addGame(GameRequestBody game) {  
 Game newGame = new Game();  
 newGame.setName(game.name());  
 newGame.setCreationDate(game.creationDate());  
 GameAuthor gameAuthor = gameAuthorRepository.getGameAuthorById(game.authorId());  
 if (gameAuthor == null) {  
 throw new RuntimeException("Game author not exist");  
 }  
 newGame.setGameAuthor(gameAuthor);  
 session.beginTransaction();  
 session.persist(newGame);  
 session.flush();  
 session.getTransaction().commit();  
 }  
  
 public void removeGameByName(String name) {  
 String hql = "delete Game where name=:name";  
 session.createQuery(hql, Game.class).setParameter("name", name);  
 }  
  
 public List<Game> getGames() {  
 return session.createQuery("from Game", Game.class).getResultList();  
 }  
}

GameControllers.java

@Controller()  
@RequestMapping("/game")  
public class GameController {  
 @Autowired  
 private GameRepository gameRepository;  
  
 @PostMapping("/create")  
 public @ResponseBody Map<String, String> create(@RequestBody GameRequestBody game) {  
 try {  
 gameRepository.addGame(game);  
 } catch (Exception e) {  
 return Map.*of*("error", e.getMessage());  
 }  
 return Map.*of*("result", "success");  
 }  
  
 @DeleteMapping("/delete")  
 public @ResponseBody String remove(@RequestBody DeleteRequestBody deleteRequestBody) {  
 gameRepository.removeGameByName(deleteRequestBody.deleteValue());  
 return "Success";  
 }  
  
 @GetMapping("/getAll")  
 public @ResponseBody List<Game> getAllGames() {  
 return gameRepository.getGames();  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №17**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с Criteria API в Hibernate.

Постановка задачи: Добавить возможность фильтрации по всем полям всех классов с использованием Criteria API в Hibernate для программы из предыдущего задания. Добавить эндпоинты для каждой фильтрации.

1. **Код пояснение к коду**

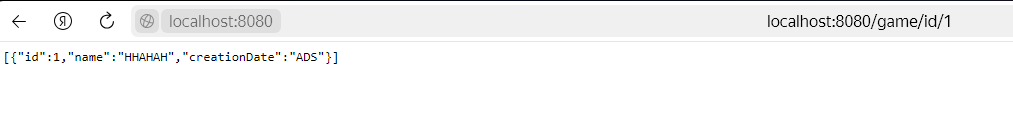
GameRepository.java

@Component  
public class GameRepository {  
 @Autowired  
 private SessionFactory sessionFactory;  
  
 @Autowired  
 private GameAuthorRepository gameAuthorRepository;  
  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
  
 public void addGame(GameRequestBody game) {  
 Game newGame = new Game();  
 newGame.setName(game.name());  
 newGame.setCreationDate(game.creationDate());  
 GameAuthor gameAuthor = gameAuthorRepository.getGameAuthorById(game.authorId());  
 if (gameAuthor == null) {  
 throw new RuntimeException("Game author not exist");  
 }  
 newGame.setGameAuthor(gameAuthor);  
 session.beginTransaction();  
 session.persist(newGame);  
 session.flush();  
 session.getTransaction().commit();  
 }  
  
 public List<Game> getGameByAttr(String attrName, Object attrValue) {  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Game> gameCriteriaQuery = builder.createQuery(Game.class);  
 Root<Game> root = gameCriteriaQuery.from(Game.class);  
 gameCriteriaQuery.select(root).where(builder.equal(root.get(attrName), attrValue));  
 Query<Game> query = session.createQuery(gameCriteriaQuery);  
 return query.getResultList();  
 }  
  
 public void removeGameByName(String name) {  
 String hql = "delete Game where name=:name";  
 session.createQuery(hql, Game.class).setParameter("name", name);  
 }  
  
 public List<Game> getGames() {  
 return session.createQuery("from Game", Game.class).getResultList();  
 }  
}

GameAuthorRepository.java

@Repository  
public class GameAuthorRepository {  
 @Autowired  
 private SessionFactory sessionFactory;  
  
 private Session session;  
  
 @PostConstruct  
 void init() {  
 session = sessionFactory.openSession();  
 }  
  
 public void addGameAuthor(GameAuthor gameAuthor) {  
 session.beginTransaction();  
 session.persist(gameAuthor);  
 session.flush();  
 session.getTransaction().commit();  
 }  
  
 public void removeGameAuthorByName(String name) {  
 String hql = "delete GameAuthor where nickname=:name";  
 session.createQuery(hql, GameAuthor.class).setParameter("name", name);  
 }  
  
 public List<GameAuthor> getGameAuthorByAttr(String attrName, Object attrValue) {  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<GameAuthor> gameCriteriaQuery = builder.createQuery(GameAuthor.class);  
 Root<GameAuthor> root = gameCriteriaQuery.from(GameAuthor.class);  
 gameCriteriaQuery.select(root).where(builder.equal(root.get(attrName), attrValue));  
 Query<GameAuthor> query = session.createQuery(gameCriteriaQuery);  
 return query.getResultList();  
 }  
  
 public GameAuthor getGameAuthorById(Long id) {  
 return session.createQuery("from GameAuthor where id="+id.toString(), GameAuthor.class).getSingleResult();  
 }  
  
 public List<GameAuthor> getGameAuthors() {  
 return session.createQuery("from GameAuthor", GameAuthor.class).getResultList();  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №18**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с Criteria API в Hibernate.

Постановка задачи: Переписать код предыдущего задания с использованием сервисов и отделения логики контроллера от логики сервиса и репозитория. В программе всё взаимодействие с базой данных должно быть реализовано через репозитории Spring Data Jpa.

1. **Код пояснение к коду**

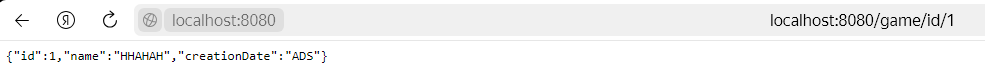
GameRepository.java

@Repository  
public interface GameRepository extends JpaRepository<Game, Long> {  
 List<Game> findAllByCreationDate(String creationDate);  
  
 List<Game> findAllByName(String name);  
  
 void deleteByName(String name);  
}

GameServiceImpl.java

@Service  
public class GameServiceImpl implements GameService {  
 @Autowired  
 private GameRepository gameRepository;  
  
 @Autowired GameAuthorService gameAuthorService;  
  
 @Override  
 public void addGame(GameRequestBody game) {  
 GameAuthor gameAuthor = gameAuthorService.getGameAuthorById(game.authorId());  
 if (gameAuthor == null) {  
 return;  
 }  
 Game newGame = new Game();  
 newGame.setGameAuthor(gameAuthor);  
 newGame.setCreationDate(game.creationDate());  
 newGame.setName(game.name());  
 gameRepository.save(newGame);  
 }  
  
 @Override  
 public List<Game> getGameByName(String name) {  
 return gameRepository.findAllByName(name);  
 }  
  
 @Override  
 public List<Game> getGameByCreationDate(String creationDate) {  
 return gameRepository.findAllByCreationDate(creationDate);  
 }  
  
 @Override  
 public Game getGameById(Long id) {  
 return gameRepository.findById(id).orElse(null);  
 }  
  
  
 @Override  
 public void removeGameByName(String name) {  
 gameRepository.deleteByName(name);  
 }  
  
 @Override  
 public List<Game> getGames() {  
 return gameRepository.findAll();  
 }  
}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №19**

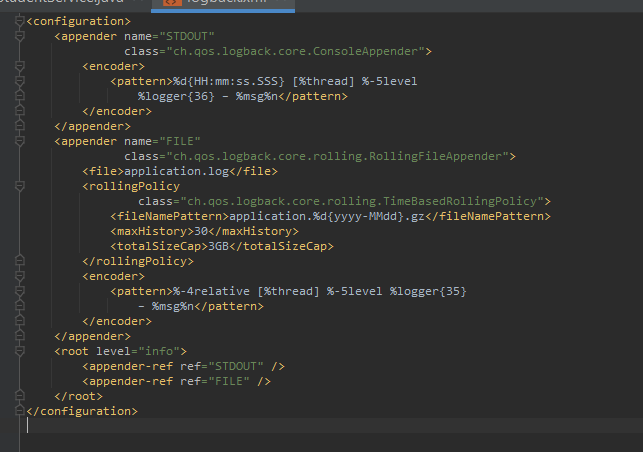
1. **Постановка задачи и персональный вариант**

Тема: Знакомство с логированием с использованием Logback в Spring.

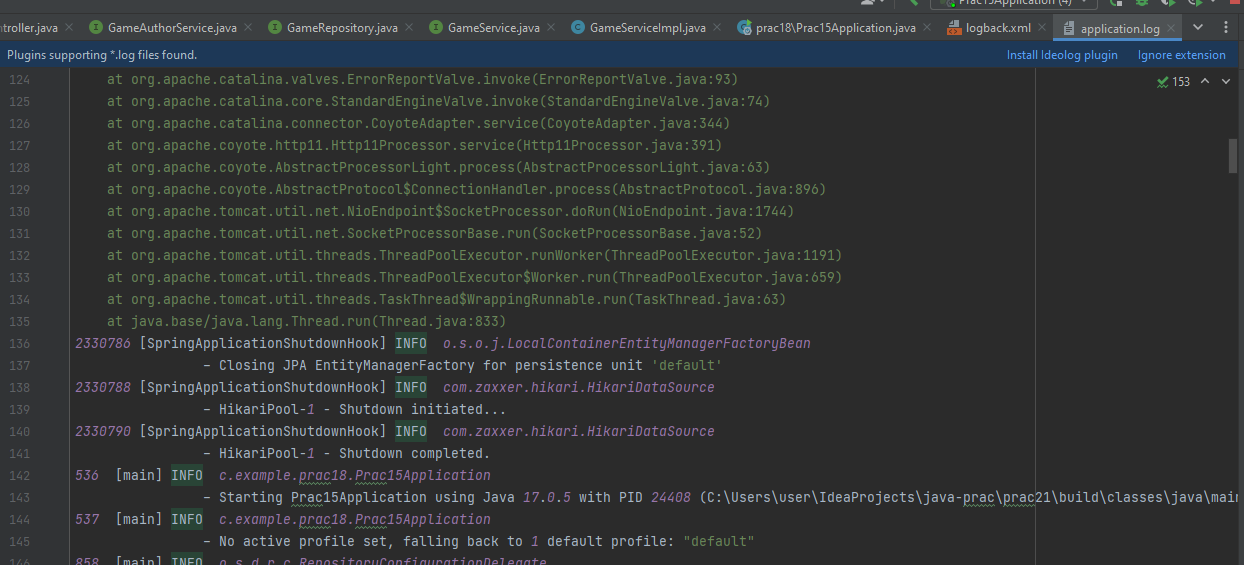
Постановка задачи: Создать файл logback.xml, добавить логирование во все методы классов сервисов

1. **Код пояснение к коду**

logback.xml



1. **Результат выполнения программы**

****

Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №20**

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring AOP. Pointcut, JoinPoint. Advice.

Постановка задачи: Для приложения из предыдущего задания добавить логирование времени выполнения каждого метода сервиса с использованием Spring AOP.

1. **Код пояснение к коду**

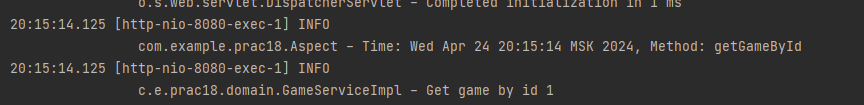
Aspect.java

@Slf4j  
@Component  
@org.aspectj.lang.annotation.Aspect  
public class Aspect {  
 @Before("allServiceMethods()")  
 public void logParameters(JoinPoint joinPoint) {  
 *log*.info("Time: {}, Method: {}", new Date(), joinPoint.getSignature().getName());  
 }  
  
 @Pointcut("within(com.example.prac18.domain.\*)")  
 public void allServiceMethods() {  
 }  
}

GameService.java

@Service  
@Slf4j  
public class GameServiceImpl implements GameService {  
 @Autowired  
 private GameRepository gameRepository;  
  
 @Autowired GameAuthorService gameAuthorService;  
  
 @Override  
 public void addGame(GameRequestBody game) {  
 *log*.info("Add game");  
 GameAuthor gameAuthor = gameAuthorService.getGameAuthorById(game.authorId());  
 if (gameAuthor == null) {  
 return;  
 }  
 Game newGame = new Game();  
 newGame.setGameAuthor(gameAuthor);  
 newGame.setCreationDate(game.creationDate());  
 newGame.setName(game.name());  
 gameRepository.save(newGame);  
 }  
  
 @Override  
 public List<Game> getGameByName(String name) {  
 *log*.info("Get game by name {}", name);  
 return gameRepository.findAllByName(name);  
 }  
  
 @Override  
 public List<Game> getGameByCreationDate(String creationDate) {  
 *log*.info("Get game by creation date {}", creationDate);  
 return gameRepository.findAllByCreationDate(creationDate);  
 }  
  
 @Override  
 public Game getGameById(Long id) {  
 *log*.info("Get game by id {}", id);  
 return gameRepository.findById(id).orElse(null);  
 }  
  
  
 @Override  
 public void removeGameByName(String name) {  
 *log*.info("Remove game by name {}", name);  
 gameRepository.deleteByName(name);  
 }  
  
 @Override  
 public List<Game> getGames() {  
 *log*.info("Get games");  
 return gameRepository.findAll();  
 }  
}

1. **Результат выполнения программы**

****

Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №21**

1. **Постановка задачи и персональный вариант**

Тема: Планирование заданий. Scheduler в Spring.

Постановка задачи: Для приложения из предыдущего задания пометить все классы сервисов, в которых происходит взаимодействие с базой данных, как Transactional. Добавить отправку информации о сохранении каждого объекта по электронной почте, создав отдельный класс EmailService с асинхронными методами отправки сообщений. Для асинхронности методов используйте аннотацию Async.

1. **Код пояснение к коду**

EmailService.java

@Service  
@RequiredArgsConstructor  
public class EmailService {  
 private final JavaMailSender mailSender;  
 @Async  
 public void sendMessage(String to, String text) {  
 SimpleMailMessage message = new SimpleMailMessage();  
 message.setFrom("holop300@gmail.com");  
 message.setTo(to);  
 message.setText(text);  
 mailSender.send(message);  
 }  
}

Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №22**

1. **Постановка задачи и персональный вариант**

Тема: Планирование заданий. Scheduler в Spring.

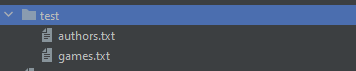
Постановка задачи: Для приложения из предыдущего задания создать класс-сервис с методом, который будет вызываться каждые 30 минут и очищать определённую директорию, а затем создавать по файлу для каждой из сущностей и загружать туда все данные из базы данных. Также добавить возможность вызывать данный метод с использованием Java Management Extensions (JMX).

1. **Код пояснение к коду**

ShedulerService.java

@Service  
@Slf4j  
@RequiredArgsConstructor  
public class SchedulerService {  
 private final GameAuthorService gameAuthorService;  
 private final GameService gameService;  
 @Scheduled(fixedDelay = 30, timeUnit = TimeUnit.*SECONDS*)  
 public void doScheduledTask(){  
 try {  
 FileUtils.*cleanDirectory*(new File("test"));  
 } catch (IOException e) {  
 *log*.info("Error while clean directory");  
 }  
 List<String> authors = gameAuthorService.getGameAuthors().stream().map((gameAuthor) -> gameAuthor.getId() +  
 " " +  
 gameAuthor.getBirthday() +  
 " " +  
 gameAuthor.getNickname()).toList();  
 List<String> games = gameService.getGames().stream().map((game) -> game.getId() +  
 " " +  
 game.getName() +  
 " " +  
 game.getCreationDate()).toList();  
 Path authorsFile = Paths.*get*("test/authors.txt");  
 Path gamesFile = Paths.*get*("test/games.txt");  
 try {  
 Files.*write*(authorsFile, authors, StandardCharsets.*UTF\_8*);  
 Files.*write*(gamesFile, games, StandardCharsets.*UTF\_8*);  
 } catch (IOException e) {  
 *log*.info("Error while write files");  
 }  
 }  
}

1. **Результат выполнения программы**

****

Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №23**

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring Security для аутентификации и авторизации пользователей.

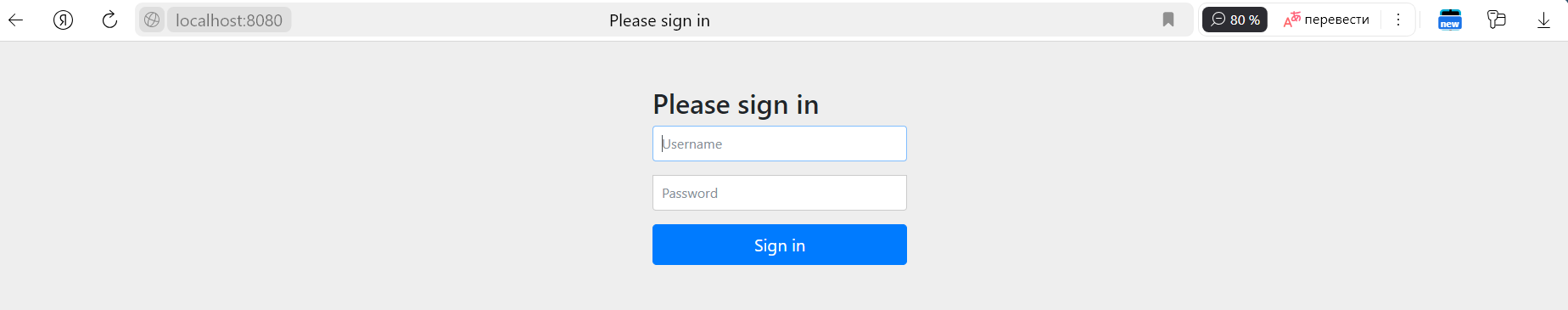
Постановка задачи: В приложении из предыдущего задания добавить возможность регистрации и авторизации пользователей, хранение cookie сессий в базе данных PostgreSQL, хеширование паролей алгоритмом Bcrypt, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

1. **Код пояснение к коду**

AppConfig.java

@Configuration  
@EnableJpaRepositories  
@EnableWebSecurity  
@EnableMethodSecurity  
public class AppConfig {  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return new UserDetailsServiceImpl();  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http.csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(auth -> auth.requestMatchers("/\*\*").authenticated())  
 .formLogin(AbstractAuthenticationFilterConfigurer::permitAll)  
 .build();  
 }  
  
 @Bean  
 public AuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService());  
 provider.setPasswordEncoder(passwordEncoder());  
 return provider;  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
}

1. **Результат выполнения программы**

****

Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>

**Практическая работа №24**

1. **Постановка задачи и персональный вариант**

Тема: Тестирование в Spring Framework с использованием Junit.

Постановка задачи: Написать модульное тестирование для всех классов сервисов приложения из предыдущего задания.

1. **Код пояснение к коду**

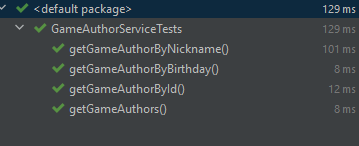
GameAuthorServiceTest.java

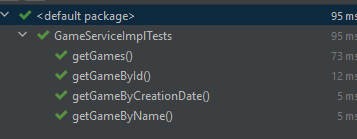
@SpringBootTest  
public class GameAuthorServiceTests {  
 @Mock  
 private GameAuthorRepository gameAuthorRepository;  
 @Test  
 void getGameAuthorByNickname() {  
 GameAuthor gameAuthor = new GameAuthor(  
 0L,  
 "stepa",  
 "hype"  
 );  
 Mockito.*when*(gameAuthorRepository.getGameAuthorsByNickname("stepa")).thenReturn(List.*of*(gameAuthor, gameAuthor));  
 GameAuthorService gameAuthorService = new GameAuthorServiceImpl(gameAuthorRepository);  
 Assertions.*assertEquals*(2, gameAuthorService.getGameAuthorByNickname("stepa").size());  
 }  
  
 @Test  
 void getGameAuthorByBirthday() {  
 GameAuthor gameAuthor = new GameAuthor(  
 0L,  
 "stepa",  
 "hype"  
 );  
 Mockito.*when*(gameAuthorRepository.getGameAuthorsByBirthday("hype")).thenReturn(List.*of*(gameAuthor, gameAuthor, gameAuthor));  
 GameAuthorService gameAuthorService = new GameAuthorServiceImpl(gameAuthorRepository);  
 Assertions.*assertEquals*(3, gameAuthorService.getGameAuthorByBirthday("hype").size());  
 }  
  
 @Test  
 void getGameAuthorById() {  
 GameAuthor gameAuthor = new GameAuthor(  
 0L,  
 "stepa",  
 "hype"  
 );  
 Mockito.*when*(gameAuthorRepository.findById(Mockito.*anyLong*())).thenReturn(Optional.*empty*());  
 Mockito.*when*(gameAuthorRepository.findById(0L)).thenReturn(Optional.*of*(gameAuthor));  
 GameAuthorService gameAuthorService = new GameAuthorServiceImpl(gameAuthorRepository);  
 Assertions.*assertNotNull*(gameAuthorService.getGameAuthorById(0L));  
 Assertions.*assertNull*(gameAuthorService.getGameAuthorById(2L));  
 }  
  
 @Test  
 void getGameAuthors() {  
 GameAuthor gameAuthor = new GameAuthor(  
 0L,  
 "stepa",  
 "hype"  
 );  
 Mockito.*when*(gameAuthorRepository.findAll()).thenReturn(List.*of*(gameAuthor));  
 GameAuthorService gameAuthorService = new GameAuthorServiceImpl(gameAuthorRepository);  
 Assertions.*assertEquals*(1, gameAuthorService.getGameAuthors().size());  
 }  
}

GameServiceTest.java

@SpringBootTest  
public class GameServiceImplTests {  
 @Mock  
 private GameRepository gameRepository;  
 @Mock  
 private GameAuthorService gameAuthorService;  
  
 @Test  
 void getGames() {  
 Game game = new Game(  
 0L,  
 "assasin",  
 "vchera"  
 );  
 Mockito.*when*(gameRepository.findAll()).thenReturn(List.*of*(game));  
 GameService gameService = new GameServiceImpl(gameRepository, gameAuthorService);  
 Assertions.*assertEquals*(1, gameService.getGames().size());  
 Assertions.*assertEquals*("vchera", gameService.getGames().get(0).getCreationDate());  
 }  
  
 @Test  
 void getGameById() {  
 Game game1 = new Game(  
 0L,  
 "assasin",  
 "vchera"  
 );  
 Game game2 = new Game(  
 1L,  
 "stepa",  
 "now"  
 );  
 Mockito.*when*(gameRepository.findById(Mockito.*anyLong*())).thenReturn(Optional.*empty*());  
 Mockito.*when*(gameRepository.findById(0L)).thenReturn(Optional.*of*(game1));  
 Mockito.*when*(gameRepository.findById(1L)).thenReturn(Optional.*of*(game2));  
 GameService gameService = new GameServiceImpl(gameRepository, gameAuthorService);  
 Assertions.*assertEquals*("assasin",gameService.getGameById(0L).getName());  
 Assertions.*assertEquals*("stepa",gameService.getGameById(1L).getName());  
 Assertions.*assertNull*(gameService.getGameById(2L));  
 }  
  
 @Test  
 void getGameByCreationDate() {  
 Game game1 = new Game(  
 0L,  
 "assasin",  
 "vchera"  
 );  
 Game game2 = new Game(  
 1L,  
 "stepa",  
 "now"  
 );  
 Mockito.*when*(gameRepository.findAllByCreationDate(Mockito.*anyString*())).thenReturn(List.*of*());  
 Mockito.*when*(gameRepository.findAllByCreationDate("vchera")).thenReturn(List.*of*(game1));  
 Mockito.*when*(gameRepository.findAllByCreationDate("now")).thenReturn(List.*of*(game2));  
 GameService gameService = new GameServiceImpl(gameRepository, gameAuthorService);  
 Assertions.*assertEquals*(1,gameService.getGameByCreationDate("vchera").size());  
 Assertions.*assertEquals*(1,gameService.getGameByCreationDate("now").size());  
 Assertions.*assertEquals*(0,gameService.getGameByCreationDate("2").size());  
 }  
  
 @Test  
 void getGameByName() {  
 Game game1 = new Game(  
 0L,  
 "assasin",  
 "vchera"  
 );  
 Mockito.*when*(gameRepository.findAllByName(Mockito.*anyString*())).thenReturn(List.*of*());  
 Mockito.*when*(gameRepository.findAllByName("assasin")).thenReturn(List.*of*(game1));  
 GameService gameService = new GameServiceImpl(gameRepository, gameAuthorService);  
 Assertions.*assertEquals*(1, gameService.getGameByName("assasin").size());  
 Assertions.*assertEquals*(0, gameService.getGameByName("assasin2").size());  
 }  
}

1. **Результат выполнения программы**

****

****

Ссылка на репозиторий: <https://github.com/Stepoik/java-prac>