|  |
| --- |
|  |
| 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** | ***Королев А.Д.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

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

Москва 2022

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

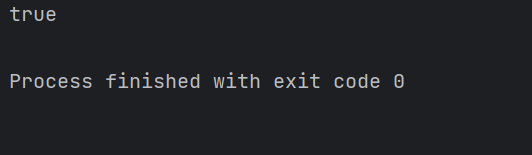
Тема: Знакомство со встроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

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

1. **Код и пояснение к коду**

package First;  
  
import java.util.function.Predicate;  
  
//10) Имплементировать интерфейс Predicate, определяющий, является ли данная строка email-адресом, используя регулярное выражение.  
  
public class FirstTask {  
 public static void main(String[] args) {  
 Predicate<String> predicate = email ->  
 email.matches("^[\\w-\\.]+@[\\w-]+(\\.[\\w-]+)\*\\.[a-z]{2,}$");  
  
 System.*out*.println(predicate.test("user@example.com"));  
 }  
}

1. **Результат работы программы**



**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

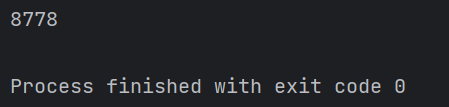
Тема: Работа со Stream API в Java 8.

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 Second;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
  
public class Second {  
 public static void main(String[] args) {  
 List<Human> people = new ArrayList<>(List.*of*(  
 new Human(19, "Andrey", "Korolev", LocalDate.*now*(), 66),  
 new Human(20, "Egor", "Bekasov", LocalDate.*now*(), 10),  
 new Human(21, "Vanya", "Terentev", LocalDate.*now*(), 100),  
 new Human(22, "Gosha", "Aleks", LocalDate.*now*(), 87),  
 new Human(23, "Stepa", "Gorohov", LocalDate.*now*(), 0)  
 ));  
  
 Integer ans = people.stream()  
 .sorted(Comparator.*comparing*(human ->  
 human.getFirstName().charAt(human.getFirstName().length() - 2)))  
 .filter(human -> human.getWeight() > 60)  
 .sorted(Comparator.*comparing*(human -> human.getAge()))  
 .mapToInt(Human::getAge)  
 .reduce((a, b) -> a \* b).getAsInt();  
  
 System.*out*.println(ans);  
 }  
}

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.

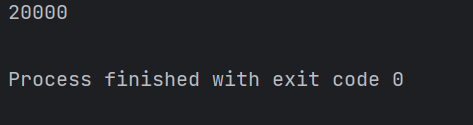
Постановка задачи: Map с использованием ключевого слова synchronized, Set с использованием Semaphore.

1. **Код и пояснение к коду**

package Third;  
  
import java.util.Collection;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Set;  
  
public class MySyncMap<K, V> implements Map<K, V> {  
 private HashMap<K, V> map = new HashMap<>();  
  
 @Override  
 public synchronized int size() {  
 return map.size();  
 }  
  
 @Override  
 public synchronized boolean isEmpty() {  
 return map.isEmpty();  
 }  
  
 @Override  
 public synchronized boolean containsKey(Object key) {  
 return map.containsKey(key);  
 }  
  
 @Override  
 public synchronized boolean containsValue(Object value) {  
 return map.containsValue(value);  
 }  
  
 @Override  
 public synchronized V get(Object key) {  
 return map.get(key);  
 }  
  
 @Override  
 public synchronized V put(K key, V value) {  
 return map.put(key, value);  
 }  
  
 @Override  
 public synchronized V remove(Object key) {  
 return map.remove(key);  
 }  
  
 @Override  
 public synchronized void putAll(Map<? extends K, ? extends V> m) {  
 map.putAll(m);  
 }  
  
 @Override  
 public synchronized void clear() {  
 map.clear();  
 }  
  
 @Override  
 public synchronized Set<K> keySet() {  
 return map.keySet();  
 }  
  
 @Override  
 public synchronized Collection<V> values() {  
 return map.values();  
 }  
  
 @Override  
 public synchronized Set<Entry<K, V>> entrySet() {  
 return map.entrySet();  
 }  
}

package Third;  
  
import java.util.Collection;  
import java.util.HashSet;  
import java.util.Iterator;  
import java.util.Set;  
import java.util.concurrent.Semaphore;  
  
public class MySyncSet<T> implements Set<T> {  
 private final static Semaphore *semaphore* = new Semaphore(1);  
 private HashSet<T> set = new HashSet<>();  
  
 @Override  
 public int size() {  
 try {  
 *semaphore*.acquire();  
 return set.size();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
  
 }  
  
 @Override  
 public boolean isEmpty() {  
 try {  
 *semaphore*.acquire();  
 return set.isEmpty();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 try {  
 *semaphore*.acquire();  
 return set.contains(o);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
  
 }  
  
 @Override  
 public Iterator<T> iterator() {  
 try {  
 *semaphore*.acquire();  
 return set.iterator();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public Object[] toArray() {  
 try {  
 *semaphore*.acquire();  
 return set.toArray();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public <T1> T1[] toArray(T1[] a) {  
 try {  
 *semaphore*.acquire();  
 return (T1[]) set.toArray();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public boolean add(T t) {  
 try {  
 *semaphore*.acquire();  
 return set.add(t);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 try {  
 *semaphore*.acquire();  
 return set.remove(o);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public boolean containsAll(Collection<?> c) {  
 try {  
 *semaphore*.acquire();  
 return set.contains(c);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public boolean addAll(Collection<? extends T> c) {  
 try {  
 *semaphore*.acquire();  
 return set.add((T) c);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public boolean retainAll(Collection<?> c) {  
 try {  
 *semaphore*.acquire();  
 return set.retainAll(c);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public boolean removeAll(Collection<?> c) {  
 try {  
 *semaphore*.acquire();  
 return set.removeAll(c);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
  
 @Override  
 public void clear() {  
 try {  
 *semaphore*.acquire();  
 set.clear();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } finally {  
 *semaphore*.release();  
 }  
 }  
}

1. **Результат работы программы**



**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

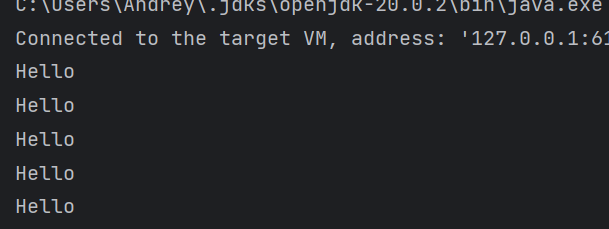
Тема: Работа с ExecutorService, CompletableFuture.

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

1. **Код и пояснение к коду**

package Fourth;  
  
import java.util.Collection;  
import java.util.List;  
import java.util.concurrent.\*;  
  
public class MyExecutoreService implements ExecutorService {  
 ExecutorService executorService;  
  
 public MyExecutoreService(int n) {  
 executorService = Executors.*newFixedThreadPool*(n);  
 }  
  
 @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.isShutdown();  
 }  
  
 @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);  
 }  
}

1. **Результат работы программы**



**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**

Singleton.java

package \_5;  
  
public enum Singleton {  
 *INSTANCE*;  
 public Singleton getInstance() {  
 return *INSTANCE*;  
 }  
}

Singleton\_.java

package \_5;  
  
public class Singleton\_ {  
 private static Singleton\_ *instance*;  
 public static synchronized Singleton\_ getInstance() {  
 if(*instance* == null) {  
 *instance* = new Singleton\_();  
 return *instance*;  
 }  
 return *instance*;  
 }  
  
}

}

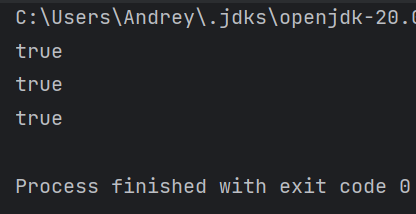
Singleton\_\_.java

package \_5;  
  
  
public class Singleton\_\_ {  
 private Singleton\_\_ (){}  
 public final static Singleton\_\_ *instance* = new Singleton\_\_();  
}

Main.java

package \_5;  
  
public class Main {  
 public static void main(String[] args) {  
 Singleton singleton1 = Singleton.*INSTANCE*;  
 Singleton singleton2 = Singleton.*INSTANCE*;  
 System.*out*.println(singleton1 == singleton2);  
  
 Singleton\_\_ singleton\_\_1 = Singleton\_\_.*instance*;  
 Singleton\_\_ singleton\_\_2 = Singleton\_\_.*instance*;  
 System.*out*.println(singleton\_\_1 == singleton\_\_2);  
  
 Singleton\_\_ value = Singleton\_\_.*instance*;  
 Singleton\_\_ value2 = Singleton\_\_.*instance*;  
 System.*out*.println(value == value2);  
 }  
}

1. **Результат выполнения программы**



Ссылка на гит: <https://github.com/grannnsacker/java4sem>.

**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**

Фабрика

Andrey.java

package \_6.Factory;  
  
public class Andrey implements Person {  
  
 @Override  
 public void getName() {  
 System.*out*.println("My name is Alexandridi-Shandaevskiy Gosha");  
 }  
}

Person.java

package \_6.Factory;  
  
public interface Person {  
 void getName();  
}

PersonFactory.java

package \_6.Factory;  
  
public class PersonFactory {  
 public Person createPerson(String name) {  
 switch (name) {  
 case "Andrey":  
 return new Andrey();  
 case "Kek":  
 return new Kek();  
 default:  
 return null;  
 }  
 }  
}

Kek.java

package \_6.Factory;  
  
  
public class Kek implements Person {  
  
 @Override  
 public void getName() {  
 System.*out*.println("My name is Stepa");  
 }  
}

Main.java

package \_6.Factory;  
  
public class Main {  
 public static void main(String[] args) {  
 PersonFactory factory = new PersonFactory();  
  
 Person person = factory.createPerson("Andrey");  
 Person person1 = factory.createPerson("Kek");  
 person.getName();  
 person1.getName();  
 }  
}

Абстрактная фабрика

AbsctractFactory.java

package \_6.AbstractFactory;  
  
public interface AbstractFactory {  
 Dog createDog();  
 Person createPerson();  
}

Builder

Builder.java

package \_6.Builder;  
  
public class Builder {  
 private String color;  
 private int age;  
  
 public Builder setColor(String color) {  
 this.color = color;  
 return this;  
 }  
  
 public Builder setAge(int age) {  
 this.age = age;  
 return this;  
 }  
  
 public Dog build() {  
 return new Dog(this.age, this.color);  
 }  
}

Dog.java

package \_6.Builder;  
  
public record Dog(int age, String color) {  
 @Override  
 public String toString() {  
 return "Dog{" +  
 "age=" + age +  
 ", color='" + color + '\'' +  
 '}';  
 }  
}

Main.java

package \_6.Builder;  
  
public class Main {  
 public static void main(String[] args) {  
 Builder builder = new Builder();  
 Dog dog = builder.setColor("dsfs").setAge(18).build();  
 System.*out*.println(dog);  
 }  
}

Прототип

Sheep.java

package \_6.Prototype;  
  
public interface Sheep {  
 Sheep clone();  
 String getName();  
 void setName(String name);  
}

BlackSheep.java

package \_6.Prototype;  
  
public class BlackSheep implements Sheep {  
 private String name;  
  
 public BlackSheep(String name) {  
 this.name = name;  
 }  
  
 public Sheep clone() {  
 return new BlackSheep(this.name);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

WhiteSheep.java

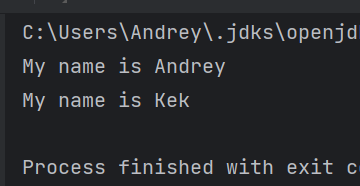
package \_6.Prototype;  
  
public class WhiteSheep implements Sheep {  
 private String name;  
  
 public WhiteSheep(String name) {  
 this.name = name;  
 }  
  
 public Sheep clone() {  
 return new WhiteSheep(this.name);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

Main.java

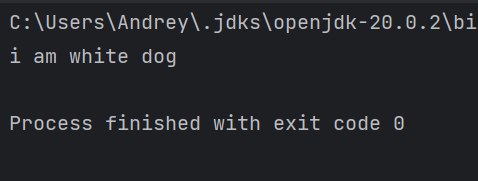
package \_6.Prototype;  
  
public class Main {  
 public static void main(String[] args) {  
 Sheep blackPrototype = new BlackSheep("Baa Baa");  
 Sheep whitePrototype = new WhiteSheep("Fleecy");  
  
 Sheep clonedBlackSheep = blackPrototype.clone();  
 Sheep clonedWhiteSheep = whitePrototype.clone();  
  
 clonedBlackSheep.setName("Midnight");  
 clonedWhiteSheep.setName("Snowball");  
  
 System.out.println("Black sheep: " + clonedBlackSheep.getName());  
 System.out.println("White sheep: " + clonedWhiteSheep.getName());  
 }  
}

1. **Результат выполнения программы**

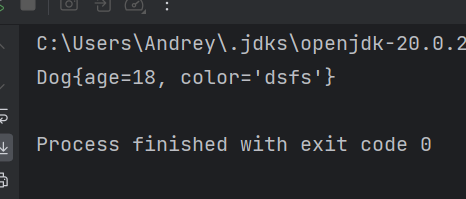
Фабрика



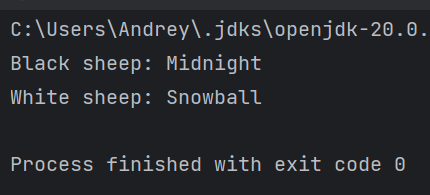
Абстрактная фабрика



Builder



Prototype



Ссылка на репозиторий: <https://github.com/grannnsacker/java4sem>.

**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: Адаптер и мост.

1. **Код и пояснение к коду**

Adapter

CardReader.java

package \_7.adapter;

public class CardReader implements USB{

private final MemoryCard memoryCard;

public CardReader(MemoryCard memoryCard) {

this.memoryCard = memoryCard;

}

@Override

public void connectWithUsbCable() {

this.memoryCard.insert();

this.memoryCard.copyData();

}

}

MemoryCard.java

package \_7.adapter;

public class MemoryCard {

public void insert() {

System.*out*.println("Memory card connects!");

}

public void copyData() {

System.*out*.println("Data copies!");

}

}

USB.java

package \_7.adapter;

public interface USB {

void connectWithUsbCable();

}

Main.java

package \_7.adapter;

public class Main {

public static void main(String[] args) {

USB cardReader = new CardReader(new MemoryCard());

cardReader.connectWithUsbCable();

}

}

Most

Color.java

package \_7.most;

public interface Color {

void fillColor();

}

Shape.java

package \_7.most;

public abstract class Shape {

protected Color color;

public Shape(Color color) {

this.color = color;

}

public abstract void draw();

}

Color.java

package \_7.most;

public class Circle extends Shape{

public Circle(Color color) {

super(color);

}

@Override

public void draw() {

System.out.println("Drawing circle");

color.fillColor();

}

}

Rectangle.java

package \_7.most;

public class Rectangle extends Shape {

public Rectangle(Color color) {

super(color);

}

@Override

public void draw() {

System.out.println("Drawing rectangle");

color.fillColor();

}

}

BlueColor.java

package \_7.most;

public class BlueColor implements Color{

@Override

public void fillColor() {

System.out.println("Filling in blue color");

}

}

RedColor.java

package \_7.most;

public class RedColor implements Color{

@Override

public void fillColor() {

System.out.println("Filling in red color");

}

}

Main.java

package \_7.most;

public class Main {

public static void main(String[] args) {

Color red = new RedColor();

Color blue = new BlueColor();

Rectangle rectangle = new Rectangle(red);

rectangle.draw();

Circle circle = new Circle(blue);

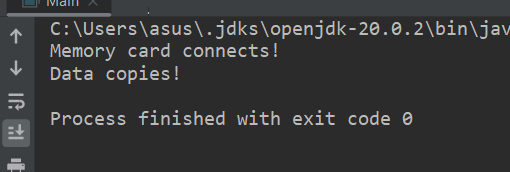
circle.draw();

}

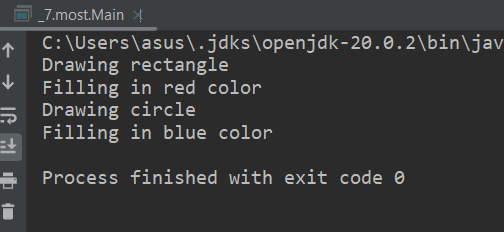
}

1. **Результаты выполнения программы**

Adapter



Most



Ссылка на репозиторий: <https://github.com/botmadness111>

**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: Цепочка обязанностей, Команда.

1. **Код и пояснение к коду**

Chain

MessageSender.java

package \_8.chain;

public abstract class MessageSender {

private final Integer priorityLevel;

private MessageSender nextMessageSender;

public MessageSender(Integer priorityLevel) {

this.priorityLevel = priorityLevel;

}

public void setNextMessageSender(MessageSender nextMessageSender) {

this.nextMessageSender = nextMessageSender;

}

public void messageSenderManager(String message, Integer level){

if(level >= priorityLevel){

write(message);

}

if(nextMessageSender != null){

nextMessageSender.messageSenderManager(message, level);

}

}

public abstract void write(String message);

}

EmailMessageSender.java

package \_8.chain;

public class EmailMessageSender extends MessageSender{

public EmailMessageSender(Integer priorityLevel) {

super(priorityLevel);

}

@Override

public void write(String message) {

System.*out*.println("Sending email: " + message);

}

}

LogReportMessageSender.java

package \_8.chain;

public class LogReportMessageSender extends MessageSender{

public LogReportMessageSender(Integer priorityLevel) {

super(priorityLevel);

}

@Override

public void write(String message) {

System.*out*.println("Message sender using simple log report: " + message);

}

}

SMSMessageSender.java

package \_8.chain;

public class SMSMessageSender extends MessageSender{

public SMSMessageSender(Integer priorityLevel) {

super(priorityLevel);

}

@Override

public void write(String message) {

System.*out*.println("Sending SMS to manager: " + message);

}

}

Main.java

package \_8.chain;

public class Main {

public static void main(String[] args) {

MessageSender reportMessageSender = new LogReportMessageSender(1);

MessageSender emailMessageSender = new EmailMessageSender(2);

MessageSender smsMessageSender = new SMSMessageSender(3);

reportMessageSender.setNextMessageSender(emailMessageSender);

emailMessageSender.setNextMessageSender(smsMessageSender);

reportMessageSender.messageSenderManager("First level", 1);

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

reportMessageSender.messageSenderManager("Second level", 2);

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

reportMessageSender.messageSenderManager("Third level", 3);

}

}

Comand

Command.java

package \_8.comand;

public interface Command{

void execute();

}

Light.java

package \_8.comand;

public class Light{

public void turnOn(){

System.*out*.println("The light is on");

}

public void turnOff(){

System.*out*.println("The light is off");

}

}

TurnOffLightCommand.java

package \_8.comand;

public class TurnOffLightCommand implements Command{

private final Light theLight;

public TurnOffLightCommand(Light light){

this.theLight=light;

}

@Override

public void execute(){

theLight.turnOff();

}

}

TurnOnLightCommand.java

package \_8.comand;

public class TurnOnLightCommand implements Command{

private final Light theLight;

public TurnOnLightCommand(Light light){

this.theLight=light;

}

@Override

public void execute() {

theLight.turnOn();

}

}

Switcher.java

package \_8.comand;

public class Switcher {

private Command flipUpCommand;

private Command flipDownCommand;

public Switcher(Command flipUpCommand,Command flipDownCommand){

this.flipUpCommand=flipUpCommand;

this.flipDownCommand=flipDownCommand;

}

public void flipUp(){

flipUpCommand.execute();

}

public void flipDown(){

flipDownCommand.execute();

}

}

Main.java

package \_8.comand;

public class Main {

public static void main(String[] args) {

Light light = new Light();

Command switchUp = new TurnOnLightCommand(light);

Command switchDown = new TurnOffLightCommand(light);

Switcher switcher = new Switcher(switchUp, switchDown);

switcher.flipUp();

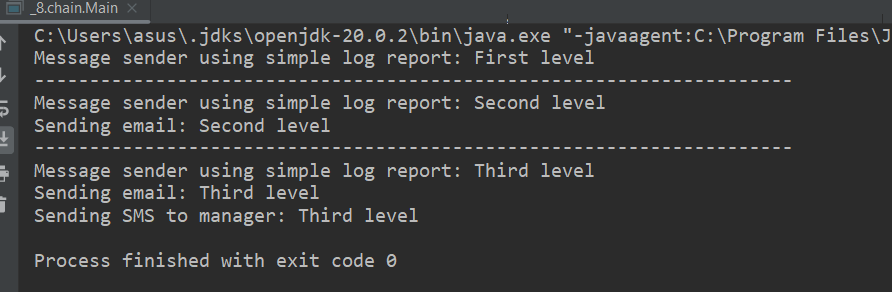
switcher.flipDown();

}

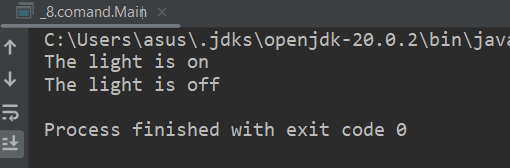
}

1. **Результат выполнения программы**

Chain



Command



Ссылка на репозиторий: <https://github.com/botmadness111>

**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.example'  
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'  
}  
  
java {  
 sourceCompatibility = JavaVersion.toVersion("8")  
 targetCompatibility = JavaVersion.toVersion("8")  
}  
  
test {  
 useJUnitPlatform()  
}  
  
jar {  
 manifest{  
 attributes (  
 "Main-Class" : "org.example.Main"  
 )  
 }  
}  
  
task createJar {  
 createJar.dependsOn jar  
 copy {  
 from jar  
 into "$projectDir/docker"  
 rename { String name ->  
 name.replace(name, "project.jar")  
 }  
 }  
}  
  
task docker(dependsOn: createJar, type: Exec) {  
 commandLine "bash", "./docker/script.sh"  
}

script.sh

docker build -t java\_project ./docker

docker run java\_project

Dockerfile

FROM openjdk:17  
COPY project.jar project.jar  
CMD ["java", "-jar", "project.jar"]

Main.java

package org.example;

public class Main {

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

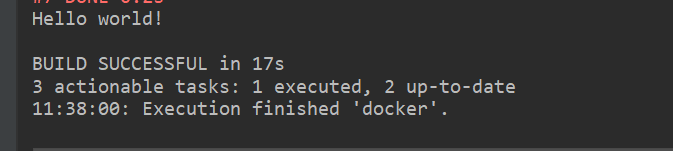
System.*out*.println("Hello world!");

Thread.*sleep*(10000);

}

}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/botmadness111>

**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 Knight с методом void fight(), его имплементации: StrongKnight, WeakKnight, KingOfKnights.

1. **Код и пояснение к коду**

Knight.java

public interface Knight {

void fight();

}

StrongKnight.java

import org.springframework.stereotype.Component;

@Component

public class StrongKnight implements Knight{

public void fight(){

System.*out*.println("Strong");

}

}

WeakKnight.java

import org.springframework.stereotype.Component;

@Component

public class WeakKnight implements Knight{

@Override

public void fight() {

System.*out*.println("weak(");

}

}

KingOfKnights

import org.springframework.stereotype.Component;

@Component

public class KingOfKnights implements Knight{

@Override

public void fight() {

System.*out*.println("King");

}

}

KnightsTest.java

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

@SpringBootTest

public class KnightsTest {

@Autowired

public void KnightsTest(List<Knight> knights) {

this.knights = knights;

}

List<Knight> knights;

@Test

public void run(){

for (Knight k:

knights) {

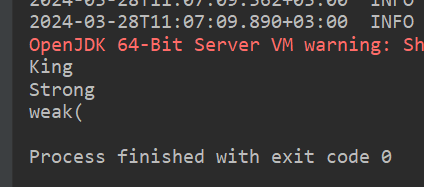
k.fight();

}

}

}

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/botmadness111>

**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**

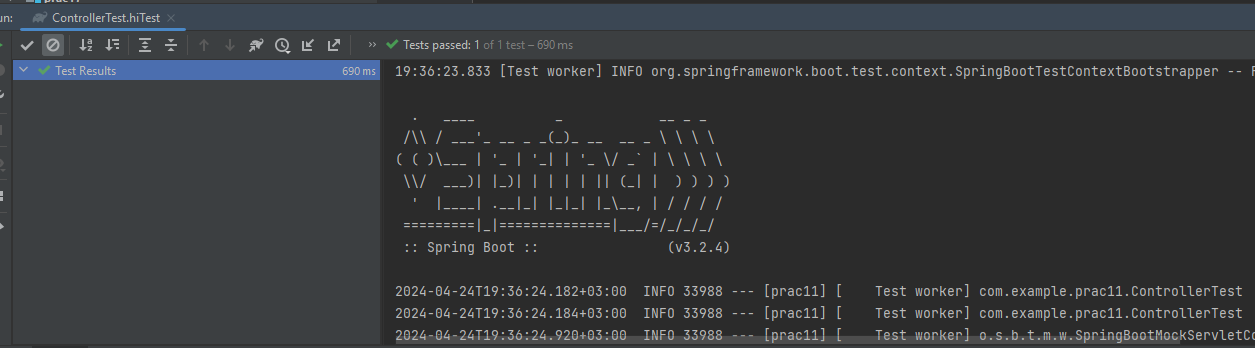
Conroller.java

@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/botmadness111>

**Практическая работа №12**

1. **Постановка задачи и персональный вариант**

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

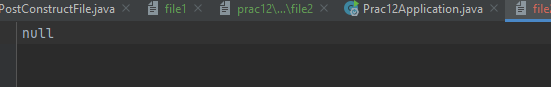
Постановка задачи: создать приложение, которое при запуске берет данные из одного файла, хеширует, а при остановке приложения удаляет исходный файл, оставляя только файл с захешированными данными. Названия первого и второго файла передаются в качестве аргументов при запуске. При отсутствии первого файла 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/botmadness111>

**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: Andrey  
 last\_name: Korolev  
 group: IKBO-20-22

Ссылка на репозиторий: <https://github.com/botmadness111>

**Практическая работа №14**

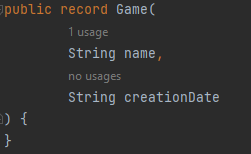
1. **Постановка задачи и персональный вариант**

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

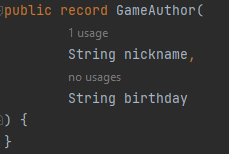
Постановка задачи: Создать класс Game с полями name, creationDate. Создать класс GameAuthor с полями nickname, birthDate. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти

1. **Код пояснение к коду**

Game.java



GameAuthor.java



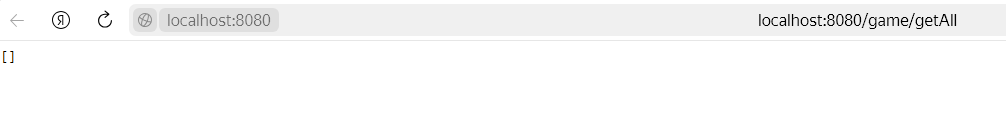
GameAuthorController.java

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

GameController.java

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

1. **Результат выполнения программы**



Ссылка на репозиторий: <https://github.com/botmadness111>

**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 компьютера.

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/botmadness111>

**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;  
  
 @OneToMany  
 @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;  
  
 @ManyToOne(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/botmadness111>

**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**

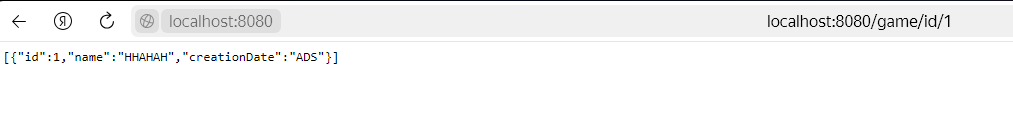
GameRepository.java

@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/botmadness111>

**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**

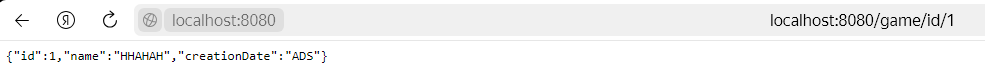
GameRepository.java

@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/botmadness111>

**Практическая работа №19**

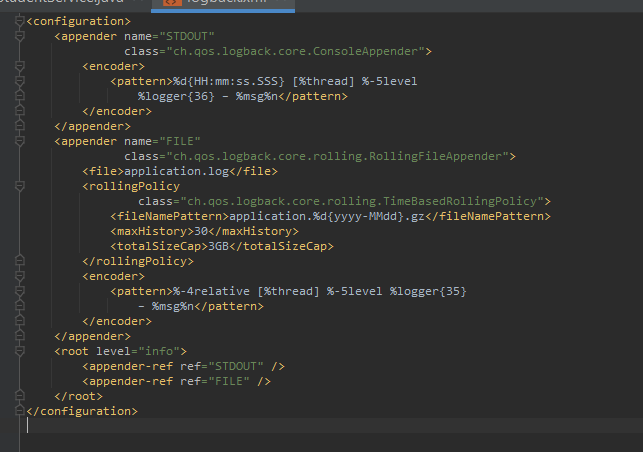
1. **Постановка задачи и персональный вариант**

Тема: Знакомство с логированием с использованием Logback в Spring.

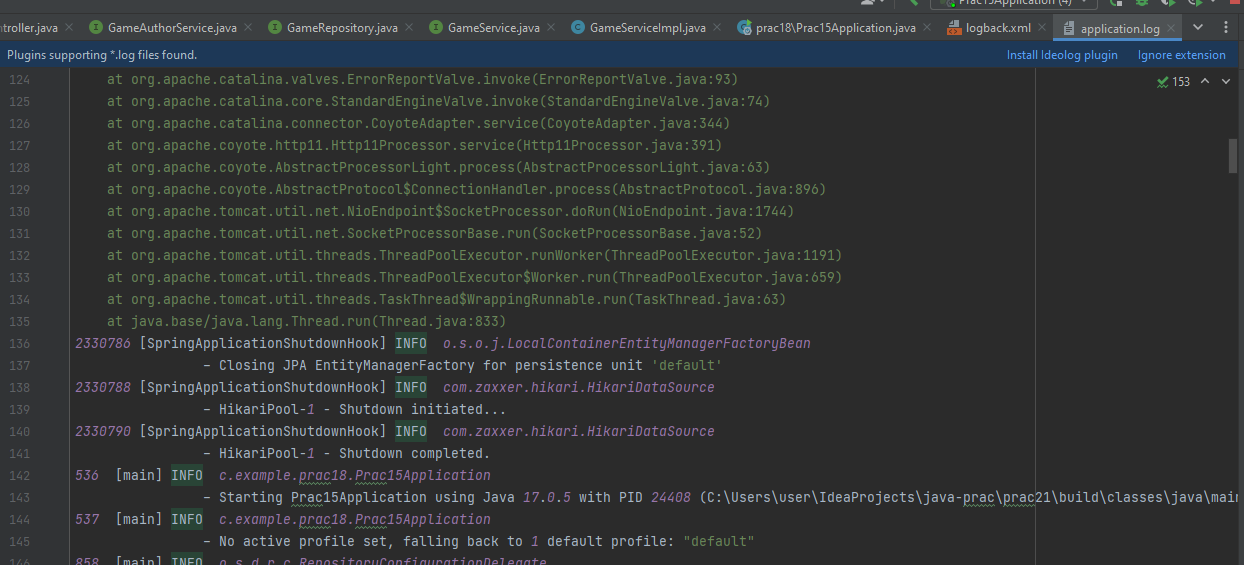
Постановка задачи: Создать файл logback.xml, добавить логирование во все методы классов сервисов

1. **Код пояснение к коду**

logback.xml



1. **Результат выполнения программы**

****

Ссылка на репозиторий: <https://github.com/botmadness111>

**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**

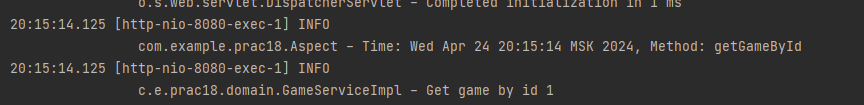
Aspect.java

@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/botmadness111>

**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/botmadness111>

**Практическая работа №22**

1. **Постановка задачи и персональный вариант**

Тема: Планирование заданий. Scheduler в Spring.

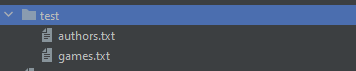
Постановка задачи: Для приложения из предыдущего задания создать класс-сервис с методом, который будет вызываться каждые 30 минут и очищать определённую директорию, а затем создавать по файлу для каждой из сущностей и загружать туда все данные из базы данных. Также добавить возможность вызывать данный 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/botmadness111>

**Практическая работа №23**

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring Security для аутентификации и авторизации пользователей.

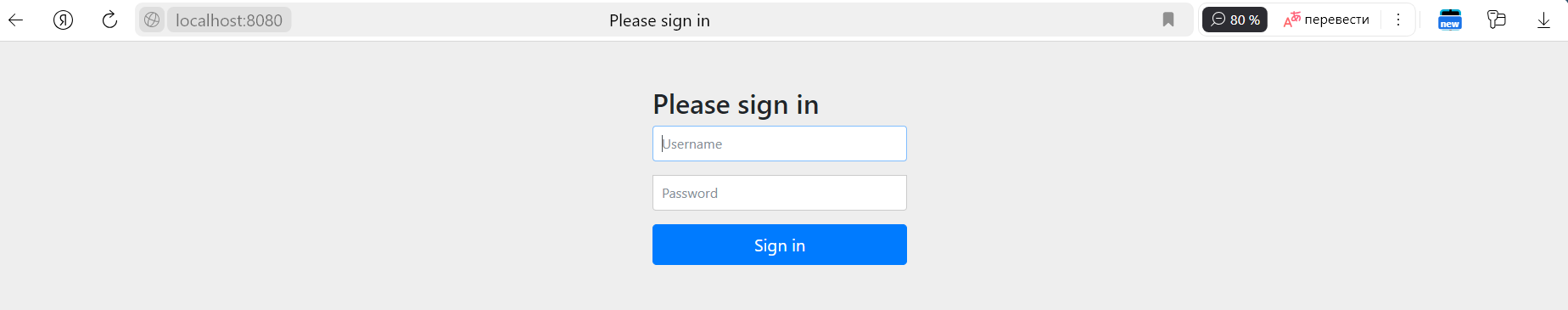
Постановка задачи: В приложении из предыдущего задания добавить возможность регистрации и авторизации пользователей, хранение cookie сессий в базе данных PostgreSQL, хеширование паролей алгоритмом Bcrypt, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

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/botmadness111>

**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**

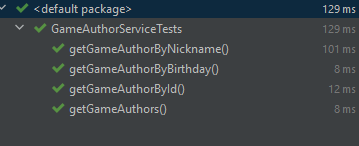
GameAuthorServiceTest.java

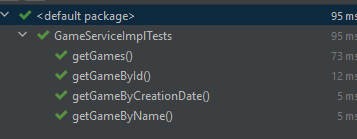
@SpringBootTest  
public class GameAuthorServiceTests {  
 @Mock  
 private GameAuthorRepository gameAuthorRepository;  
 @Test  
 void getGameAuthorByNickname() {  
 GameAuthor gameAuthor = new GameAuthor(  
 0L,  
 "stepa",  
 "hype"  
 );  
 Mockito.*when*(gameAuthorRepository.getGameAuthorsByNickname("stepa")).thenReturn(List.*of*(gameAuthor, 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/botmadness111>