|  |
| --- |
|  |
| 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 ПРАКТИЧЕСКОЙ РАБОТЕ № 1-4** | |
| **по дисциплине** | |
| **«Шаблоны программных платформ языка Джава»** | |
| Выполнил студент группы **ИКБО-36-22** | ***Утенков Ю. Ю.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « 18 » февраля 2024 г. |  |
| «Зачтено» | « » 2024 г. |  |

Москва 2024

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

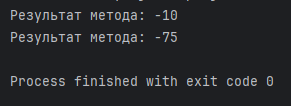
Тема: Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

Постановка задачи: Имплементировать интерфейс Comparator, сравнивающий два числа по модулю.

1. **Код и пояснение к коду**

package SPPYAD;  
  
import java.util.Comparator;  
@FunctionalInterface  
interface MyComparator extends Comparator<Integer>  
{  
 @Override  
 int compare(Integer o1, Integer o2);  
}  
public class PR1  
{  
 public static void main(String[] args) {  
 MyComparator comparator = (a, b) -> a - b;  
 System.*out*.println("Результат метода: " + comparator.compare(10, 20));  
 MyComparator comparator1 = (x, y) -> x \* x - y \* y;  
 System.*out*.println("Результат метода: " + comparator1.compare(5, 10));  
 }  
}

1. **Результат работы программы**

****

**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Тема: Работа со Stream API в Java 8

Постановка задачи:

1) класс Human (int age, String firstName, String lastName, LocalDate birthDate, int weight);

2) приложение, которое создает список из объектов класса Human, а

затем производит действия в соответствии с вариантом индивидуального

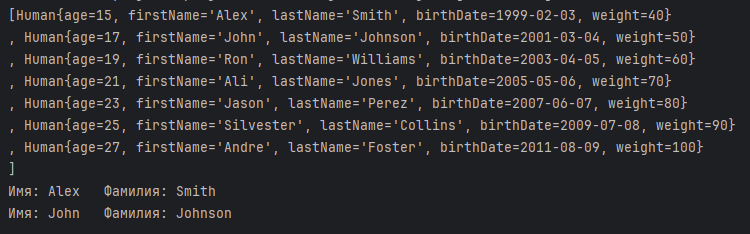
задания.

**Код и пояснение к коду**

****

package SPPYAD;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
import java.util.Scanner;  
import java.util.stream.Stream;  
  
public class PR2  
{  
 static class Human implements Comparable<Human>  
 {  
 private int age;  
 private String firstName;  
 private String lastName;  
 private final LocalDate birthDate;  
 private int weight;  
  
 public Human(int age, String firstName, String lastName, LocalDate birthDate, int weight)  
 {  
 this.age = age;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthDate = birthDate;  
 this.weight = weight;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public LocalDate getBirthDate() {  
 return birthDate;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Human human = (Human) o;  
 return age == human.age && weight == human.weight && Objects.*equals*(firstName, human.firstName) &&  
 Objects.*equals*(lastName, human.lastName) && Objects.*equals*(birthDate, human.birthDate);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(age, firstName, lastName, birthDate, weight);  
 }  
  
 @Override  
 public int compareTo(Human o) {  
 return this.weight - o.weight;  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "age=" + age +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", weight=" + weight +  
 '}' + "\n";  
 }  
 }  
 public static void main(String[] args)  
 {  
 try (FileReader reader = new FileReader("src/SPPYAD/Names.txt"))  
 {  
 Scanner sc = new Scanner(reader);  
 List<Human> array = new ArrayList<>();  
 for (int i = 0; i < 7 || sc.hasNext(); i++)  
 array.add(new Human((i << 1) + 15, sc.next(), sc.next(), LocalDate.*of*(1999 + 2 \* i, i + 2, i + 3), 40 + 10 \* i));  
 System.*out*.println(array);  
 Stream<Human> str = array.stream();  
 str.limit(5).sorted().filter(e -> e.getWeight() < 60).forEach(e -> System.*out*.println("Имя: " + e.getFirstName() + " Фамилия: " + e.getLastName()));  
 }  
 catch (IOException exc)  
 {  
 System.*exit*(1);  
 }  
 }  
}

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.

Вариант 7: List с использованием ключевого слова synchronized, Set с использованием Semaphore.

**Код и пояснение к коду**

package SPPYAD.PR3;  
  
import java.util.\*;  
import java.util.function.UnaryOperator;  
  
class MyArrayList<E> implements List<E>  
{  
 private final List<E> list = new ArrayList<>();  
 @Override  
 public synchronized int size() {  
 return list.size();  
 }  
  
 @Override  
 public synchronized boolean isEmpty() {  
 return list.isEmpty();  
 }  
  
 @Override  
 public synchronized boolean contains(Object o) {  
 return list.contains(o);  
 }  
  
 @Override  
 public synchronized Iterator<E> iterator() {  
 return list.iterator();  
 }  
  
 @Override  
 public synchronized Object[] toArray() {  
 return list.toArray();  
 }  
  
 @Override  
 public synchronized <T> T[] toArray(T[] a) {  
 return list.toArray(a);  
 }  
  
 @Override  
 public synchronized boolean add(E e) {  
 return list.add(e);  
 }  
  
 @Override  
 public synchronized boolean remove(Object o) {  
 return list.remove(o);  
 }  
  
 @Override  
 public synchronized boolean containsAll(Collection<?> c) {  
 return new HashSet<>(list).containsAll(c);  
 }  
  
 @Override  
 public synchronized boolean addAll(Collection<? extends E> c) {  
 return list.addAll(c);  
 }  
  
 @Override  
 public synchronized boolean addAll(int index, Collection<? extends E> c) {  
 return list.addAll(c);  
 }  
  
 @Override  
 public synchronized boolean removeAll(Collection<?> c) {  
 return false;  
 }  
  
 @Override  
 public synchronized boolean retainAll(Collection<?> c) {  
 return list.retainAll(c);  
 }  
  
 @Override  
 public synchronized void clear() {  
 list.clear();  
 }  
  
 @Override  
 public synchronized E get(int index) {  
 return null;  
 }  
  
 @Override  
 public synchronized E set(int index, E element) {  
 return null;  
 }  
  
 @Override  
 public synchronized void add(int index, E element) {  
 list.add(index, element);  
 }  
  
 @Override  
 public synchronized E remove(int index) {  
 return list.remove(index);  
 }  
  
 @Override  
 public synchronized int indexOf(Object o) {  
 return list.indexOf(o);  
 }  
  
 @Override  
 public synchronized int lastIndexOf(Object o) {  
 return list.lastIndexOf(o);  
 }  
  
 @Override  
 public synchronized ListIterator<E> listIterator() {  
 return list.listIterator();  
 }  
  
 @Override  
 public synchronized ListIterator<E> listIterator(int index) {  
 return list.listIterator(index);  
 }  
  
 @Override  
 public synchronized List<E> subList(int fromIndex, int toIndex) {  
 return list.subList(fromIndex, toIndex);  
 }  
  
 @Override  
 public synchronized void replaceAll(UnaryOperator<E> operator) {  
 list.replaceAll(operator);  
 }  
  
 @Override  
 public synchronized void sort(Comparator<? super E> c) {  
 list.sort(c);  
 }  
  
 @Override  
 public synchronized Spliterator<E> spliterator() {  
 return list.spliterator();  
 }  
}  
public class PR3  
{  
 public static void main(String[] args) throws InterruptedException {  
 List<Integer> arrayList = new ArrayList<>();  
 Runnable rn1 = (() -> {  
 for (int i = 0; i < 500; i++)  
 arrayList.add(i);  
 });  
 Runnable rn2 = (() -> {  
 for (int i = 0; i < 500; i++)  
 arrayList.add(i);  
 });  
 Thread thread1 = new Thread(rn1);  
 Thread thread2 = new Thread(rn2);  
 thread1.start();  
 thread2.start();  
 Thread.*sleep*(30);  
 System.*out*.println("Потоконебезопасная коллекция.\nЭлементов добавлено в лист: " + arrayList.size());  
 }  
}

Листинг 1—Тестирование потоконебезопасной коллекции

public class PR3  
{  
 public static void main(String[] args) throws InterruptedException {  
 List<Integer> arrayList = new MyArrayList<>();  
 Runnable rn1 = (() -> {  
 for (int i = 0; i < 500; i++)  
 arrayList.add(i);  
 });  
 Runnable rn2 = (() -> {  
 for (int i = 0; i < 500; i++)  
 arrayList.add(i);  
 });  
 Thread thread1 = new Thread(rn1);  
 Thread thread2 = new Thread(rn2);  
 thread1.start();  
 thread2.start();  
 Thread.*sleep*(30);  
 System.*out*.println("Потокобезопасная коллекция.\nЭлементов добавлено в лист: " + arrayList.size());  
 }  
}

Листинг 2—Тестирование собственной потокобезопасной коллекции

package SPPYAD.PR3;  
  
import java.util.Collection;  
import java.util.HashSet;  
import java.util.Iterator;  
import java.util.Set;  
import java.util.concurrent.Semaphore;  
  
class SemaphoreSet<E> implements Set<E>  
{  
 private final Semaphore semaphore;  
 private final Set<E> set = new HashSet<>();  
  
 public SemaphoreSet(Semaphore semaphore) {  
 this.semaphore = semaphore;  
 }  
  
 @Override  
 public int size() {  
 return set.size();  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return set.isEmpty();  
 }  
  
 @Override  
 public boolean contains(Object o) {  
 return set.contains(o);  
 }  
  
 @Override  
 public Iterator<E> iterator() {  
 return set.iterator();  
 }  
  
 @Override  
 public Object[] toArray() {  
 return set.toArray();  
 }  
  
 @Override  
 public <T> T[] toArray(T[] a) {  
 return set.toArray(a);  
 }  
  
 @Override  
 public boolean add(E e) {  
 if (set.contains(e)) {  
 return false;  
 }  
 try {  
 semaphore.acquire();  
 set.add(e);  
 System.*out*.println("Добавлено " + e + " " + Thread.*currentThread*().getName());  
 Thread.*sleep*(100);  
 semaphore.release();  
 return true;  
 } catch (InterruptedException ex) {  
 System.*exit*(1);  
 }  
 return false;  
 }  
  
 @Override  
 public boolean remove(Object o) {  
 return set.remove(o);  
 }  
  
 @Override  
 public boolean containsAll(Collection<?> c) {  
 return set.containsAll(c);  
 }  
  
 @Override  
 public boolean addAll(Collection<? extends E> c) {  
 return set.addAll(c);  
 }  
  
 @Override  
 public boolean retainAll(Collection<?> c) {  
 return set.retainAll(c);  
 }  
  
 @Override  
 public boolean removeAll(Collection<?> c) {  
 return set.removeAll(c);  
 }  
  
 @Override  
 public void clear() {  
 set.clear();  
 }  
}  
public class SemaphoreTest  
{  
 public static void main(String[] args) throws InterruptedException {  
 Semaphore sm = new Semaphore(1);  
 SemaphoreSet<String> strings = new SemaphoreSet<>(sm);  
 Thread tr1 = new Thread(() -> {  
 for (int i = 0; i < 30; i++)  
 strings.add(i + "");  
 });  
 Thread tr2 = new Thread(() -> {  
 for (int i = 0; i < 30; i++)  
 strings.add(i + "");  
 });  
 tr1.start();  
 tr2.start();  
 Thread.*sleep*(300);  
 System.*out*.println("Элементов в Set: " + strings.size() + " " + Thread.*currentThread*().getName());  
 strings.forEach(System.*out*::println);  
 }  
}

Листинг 3 — Коллекция Set с семафором

1. **Результат работы программы**

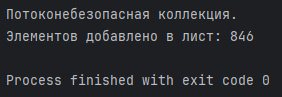


Рисунок 1 — Результат работы потоконебезопасной коллекции

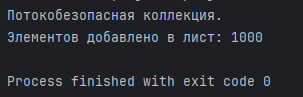


Рисунок 2 — Результат работы потокобезопасной коллекции

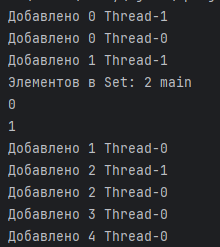


Рисунок 3 — Результат работы программы с семафором для 1 потока коллекции Set

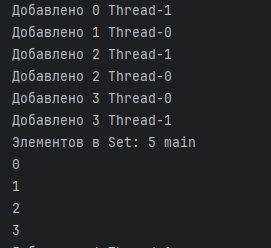


Рисунок 4 — Результат работы программы с семафором для 2 потоков коллекции Set

**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Тема: Работа с ExecutorService, CompletableFuture.

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

**Код и пояснение к коду**

package SPPYAD.PR4;  
  
import java.util.ArrayDeque;  
import java.util.Queue;  
  
class SimpleExecutorService {  
 private final Thread[] threads;  
 private final Queue<Runnable> taskQueue;  
  
 public SimpleExecutorService(int numThreads) {  
 this.threads = new Thread[numThreads];  
 this.taskQueue = new ArrayDeque<>();  
  
 for (int i = 0; i < numThreads; i++) {  
 threads[i] = new WorkerThread();  
 threads[i].start();  
 }  
 }  
  
 public void execute(Runnable task) {  
 synchronized (taskQueue) {  
 taskQueue.add(task);  
 taskQueue.notify();  
 }  
 }  
  
 private class WorkerThread extends Thread {  
 @Override  
 public void run() {  
 Runnable task;  
 while (true) {  
 synchronized (taskQueue) {  
 while (taskQueue.isEmpty()) {  
 try {  
 taskQueue.wait();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 return;  
 }  
 }  
 task = taskQueue.poll();  
 }  
 try {  
 task.run();  
 } catch (RuntimeException e) {  
 System.*exit*(1);  
 }  
 }  
 }  
 }  
}  
  
public class PR4  
{  
 public static void main(String[] args) {  
 SimpleExecutorService executorService = new SimpleExecutorService(3);  
 // Поступают задачи на выполнение  
 for (int i = 0; i < 10; i++) {  
 int taskNumber = i;  
 executorService.execute(() -> {  
 System.*out*.println("Задача " + taskNumber + " выполняется потоком " + Thread.*currentThread*().getName());  
 try {  
 Thread.*sleep*(1000); // Имитируем выполнение задачи  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Задача " + taskNumber + " была выполнена");  
 });  
 }  
 }  
}

Листинг 4 – Реализация ExecutorService

1. **Результат работы программы**

