|  |
| --- |
|  |
| 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 — 4** | |
| **по дисциплине** | |
| **«Шаблоны программных платформ языка Джава»** | |
| Выполнил студент группы **ИКБО-20-22** | ***Омельяненко К.Д.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

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

Москва 2024

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

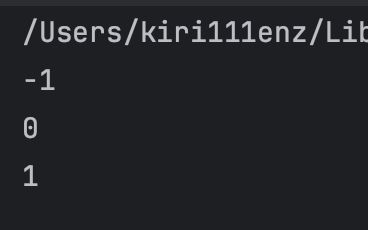
Тема: Знакомство со встроенными функциональными интерфейсами Java.

Постановка задачи: имплементировать интерфейс Comparator, сравнивающий две строки по сумме всех чисел, представленных в строке.

1. **Код и пояснение к коду**

package ru.mirea.lab1;  
  
import java.util.Comparator;  
  
public class StringNumberSumComparator implements Comparator<String> {  
 @Override  
 public int compare(String s1, String s2) {  
 int sum1 = sumOfNumbers(s1);  
 int sum2 = sumOfNumbers(s2);  
  
 return Integer.*compare*(sum1, sum2);  
 }  
  
 private int sumOfNumbers(String s) {  
 int sum = 0;  
 StringBuilder temp = new StringBuilder("0");  
 for (char c : s.toCharArray()) {  
 if (Character.*isDigit*(c)) {  
 temp.append(c);  
 } else {  
 sum += Integer.*parseInt*(temp.toString());  
 temp = new StringBuilder("0");  
 }  
 }  
 return sum + Integer.*parseInt*(temp.toString());  
 }  
  
 public static void main(String[] args) {  
 StringNumberSumComparator comp = new StringNumberSumComparator();  
 System.*out*.println(comp.compare("123", "321"));  
 System.*out*.println(comp.compare("1a2", "2b1"));  
 System.*out*.println(comp.compare("1111", "999"));  
 }  
}

1. **Результат тестирования программы**



**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Тема: Работа со Stream API в Java 8.

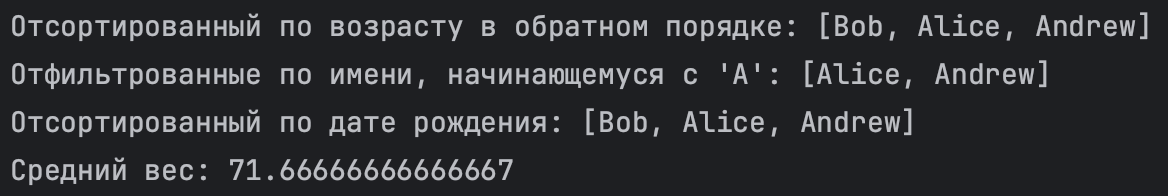
В ходе выполнения практической работы должно быть реализовано:

* класс Human (int age, String firstName, String lastName, LocalDate birthDate, int weight);
* приложение, которое создает список из объектов класса Human, а затем производит действия в соответствии с вариантом индивидуального задания (список после каждого этапа должен выводиться в консоль).

1. **Код и пояснение к коду**

package ru.mirea.lab2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
import java.util.stream.Collectors;  
  
class Human {  
 private final int age;  
 private final String firstName;  
 private final String lastName;  
 private final LocalDate birthDate;  
 private final 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 LocalDate getBirthDate() { return birthDate; }  
 public int getWeight() { return weight; }  
  
 @Override  
 public String toString() {  
 return firstName;  
 }  
}  
  
class AgeComparator implements Comparator<Human> {  
 @Override  
 public int compare(Human h1, Human h2) {  
 return Integer.*compare*(h2.getAge(), h1.getAge()); // обратный порядок  
 }  
}  
class Test {  
 public static void main(String[] args) {  
 List<Human> humans = new ArrayList<>();  
 humans.add(new Human(25, "Alice", "Johnson", LocalDate.*of*(1995, 5, 15), 60));  
 humans.add(new Human(30, "Bob", "Smith", LocalDate.*of*(1990, 3, 10), 70));  
 humans.add(new Human(20, "Andrew", "White", LocalDate.*of*(2000, 4, 20), 85));  
  
 // Сортировка по возрасту в обратном порядке  
 humans.sort(new AgeComparator());  
 System.*out*.println("Отсортированный по возрасту в обратном порядке: " + humans);  
  
 // Фильтрация по имени, которое начинается с "А"  
 List<Human> filteredByName = humans.stream()  
 .filter(h -> h.getFirstName().startsWith("A"))  
 .collect(Collectors.*toList*());  
 System.*out*.println("Отфильтрованные по имени, начинающемуся с 'A': " + filteredByName);  
  
 // Сортировка по дате рождения  
 humans.sort(Comparator.*comparing*(Human::getBirthDate));  
 System.*out*.println("Отсортированный по дате рождения: " + humans);  
  
 // Расчет среднего веса  
 double averageWeight = humans.stream()  
 .mapToInt(Human::getWeight)  
 .average()  
 .orElse(0);  
 System.*out*.println("Средний вес: " + averageWeight);  
 }  
}

1. **Результат тестирования программы**



**Практическая работа №3**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с конкурентным программированием в Java

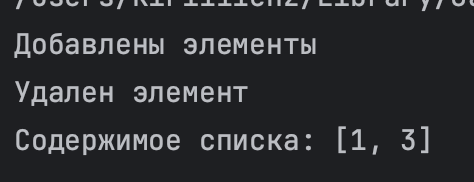
Постановка задачи: создать свои потокобезопасные имплементации интерфейсов List с использованием ключевого слова synchronized, Set с использованием Semaphore.

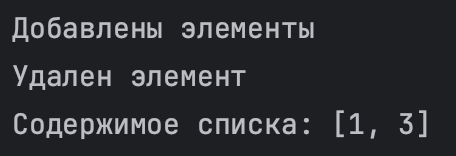
1. **Код и пояснение к коду**

package ru.mirea.lab3;  
  
import java.util.\*;  
import java.util.concurrent.Semaphore;  
  
public class SemaphoreSet<T> {  
 private final Set<T> set = Collections.*synchronizedSet*(new HashSet<>());  
 private final Semaphore semaphore = new Semaphore(1);  
  
 public void add(T item) throws InterruptedException {  
 semaphore.acquire();  
 try {  
 set.add(item);  
 } finally {  
 semaphore.release();  
 }  
 }  
  
 public void remove(T item) throws InterruptedException {  
 semaphore.acquire();  
 try {  
 set.remove(item);  
 } finally {  
 semaphore.release();  
 }  
 }  
  
 public boolean contains(T item) throws InterruptedException {  
 semaphore.acquire();  
 try {  
 return set.contains(item);  
 } finally {  
 semaphore.release();  
 }  
 }  
  
 public List<T> toList() {  
 return set.stream().toList();  
 }  
  
 public static void main(String[] args) throws InterruptedException {  
 SemaphoreSet<Integer> semaphoreSet = new SemaphoreSet<>();  
  
 Thread addThread = new Thread(() -> {  
 try {  
 semaphoreSet.add(1);  
 semaphoreSet.add(2);  
 semaphoreSet.add(3);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 System.*out*.println("Добавлены элементы");  
 });  
  
 Thread removeThread = new Thread(() -> {  
 try {  
 semaphoreSet.remove(2);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 System.*out*.println("Удален элемент");  
 });  
  
 addThread.start();  
 removeThread.start();  
  
 addThread.join();  
 removeThread.join();  
  
 System.*out*.println(semaphoreSet.toList());  
 }  
}

package ru.mirea.lab3;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class SynchronizedList<T> {  
 private final List<T> list = new ArrayList<>();  
  
 public synchronized void add(T item) {  
 list.add(item);  
 }  
  
 public synchronized void remove(T item) {  
 list.remove(item);  
 }  
  
 public synchronized boolean contains(T item) {  
 return list.contains(item);  
 }  
  
 public synchronized T get(int index) {  
 return list.get(index);  
 }  
  
 public synchronized int size() {  
 return list.size();  
 }  
  
 public static void main(String[] args) throws InterruptedException {  
 SynchronizedList<Integer> synchronizedList = new SynchronizedList<>();  
  
 Thread addThread = new Thread(() -> {  
 synchronizedList.add(1);  
 synchronizedList.add(2);  
 synchronizedList.add(3);  
 System.*out*.println("Добавлены элементы");  
 });  
  
 Thread removeThread = new Thread(() -> {  
 synchronizedList.remove(2);  
 System.*out*.println("Удален элемент");  
 });  
  
 addThread.start();  
 removeThread.start();  
  
 addThread.join();  
 removeThread.join();  
  
 System.*out*.println("Содержимое списка: " + synchronizedList.list);  
 }  
}

1. **Результат тестирования 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 ru.mirea.lab4;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
  
public class CustomServiceExecutor {  
 private final BlockingQueue<Runnable> taskQueue;  
 private final List<Thread> workerThreads;  
 private volatile boolean isShutDown = false;  
  
 public CustomServiceExecutor(int threadCount) {  
 taskQueue = new LinkedBlockingQueue<>();  
 workerThreads = new ArrayList<>(threadCount);  
  
 // инициализация рабочих потоков  
 for (int i = 0; i < threadCount; i++) {  
 Thread workerThread = new Worker("Worker-" + i);  
 workerThreads.add(workerThread);  
 workerThread.start();  
 }  
 }  
  
 public void execute(Runnable task) {  
 if (isShutDown) {  
 throw new IllegalStateException("Executor service is shutdown");  
 }  
  
 try {  
 taskQueue.put(task);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
  
 public void shutdown() {  
 isShutDown = true;  
 }  
  
 public List<Runnable> shutdownNow() {  
 shutdown();  
 List<Runnable> remainingTasks = new ArrayList<>();  
 taskQueue.drainTo(remainingTasks);  
 for (Thread worker : workerThreads) {  
 worker.interrupt();  
 }  
 return remainingTasks;  
 }  
  
 private class Worker extends Thread {  
 public Worker(String name) {  
 super(name);  
 }  
  
 public void run() {  
 while (!isShutDown || !taskQueue.isEmpty()) {  
 try {  
 Runnable task = taskQueue.take();  
 task.run();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }  
 }  
  
 public boolean isShutDown() {  
 return isShutDown;  
 }  
  
 public boolean isTerminated() {  
 if (!isShutDown) {  
 return false;  
 }  
  
 for (Thread thread : workerThreads) {  
 if (thread.isAlive()) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public static void main(String[] args) {  
 CustomServiceExecutor executorService = new CustomServiceExecutor(3);  
  
 Runnable task1 = () -> System.*out*.println("Задача 1 выполняется: " + Thread.*currentThread*().getName());  
 Runnable task2 = () -> System.*out*.println("Задача 2 выполняется: " + Thread.*currentThread*().getName());  
 Runnable task3 = () -> System.*out*.println("Задача 3 выполняется: " + Thread.*currentThread*().getName());  
  
 executorService.execute(task1);  
 executorService.execute(task2);  
 executorService.execute(task3);  
  
 executorService.shutdown();  
  
 System.*out*.println("Executor остановлен: " + executorService.isShutDown());  
 System.*out*.println("Все задачи выполнены: " + executorService.isTerminated());  
 }  
}

1. **Результат тестирования программы**

