|  |
| --- |
|  |
| 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**

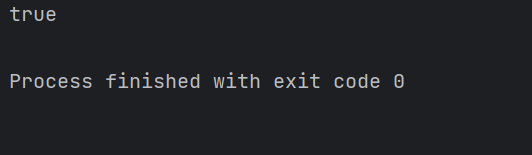
Тема: Знакомство со встроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

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**

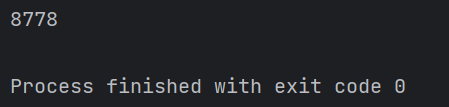
Тема: Работа со Stream API в Java 8.

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.

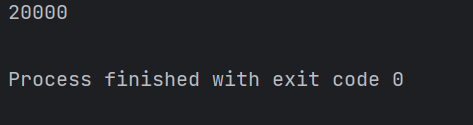
Постановка задачи: Map с использованием ключевого слова synchronized, Set с использованием Semaphore.

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**

