|  |
| --- |
|  |
| 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** | ***Шумахер М.Е.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

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

Москва 2024

**Практическая работа №1**

1. **Постановка задачи и персональный вариант**

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

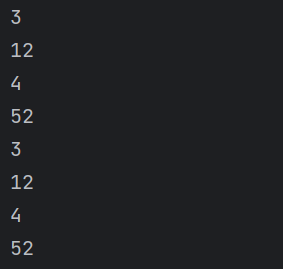
Постановка задачи: Имплементировать интерфейс Consumer, принимающий на вход массив чисел и выводящий в консоль в порядке возрастания.

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac1

package prac1;  
  
import java.util.function.Consumer;  
  
public class Main implements Consumer<int[]> {  
 @Override  
 public void accept(int[] num){  
 for (int n : num){  
 System.*out*.println(n + " ");  
 }  
 }  
 public static void main(String[] args) {  
 int[] num ={3,12,4,52};  
 Main cons = new Main();  
 cons.accept(num);  
 Consumer<int[]> printer = arr -> {  
 for (int n : arr) {  
 System.*out*.println(n + " ");  
 }  
 };  
 printer.accept(num);  
 }  
}

1. **Результат работы программы**



**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Цель: Работа со Stream API в Java 8

Постановка задачи: Увеличение веса каждого объекта на 3, сортировка по весу в обратном порядке, фильтрация по дате рождения меньшей, чем 01.01.2000, сумма всех весов.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac2>

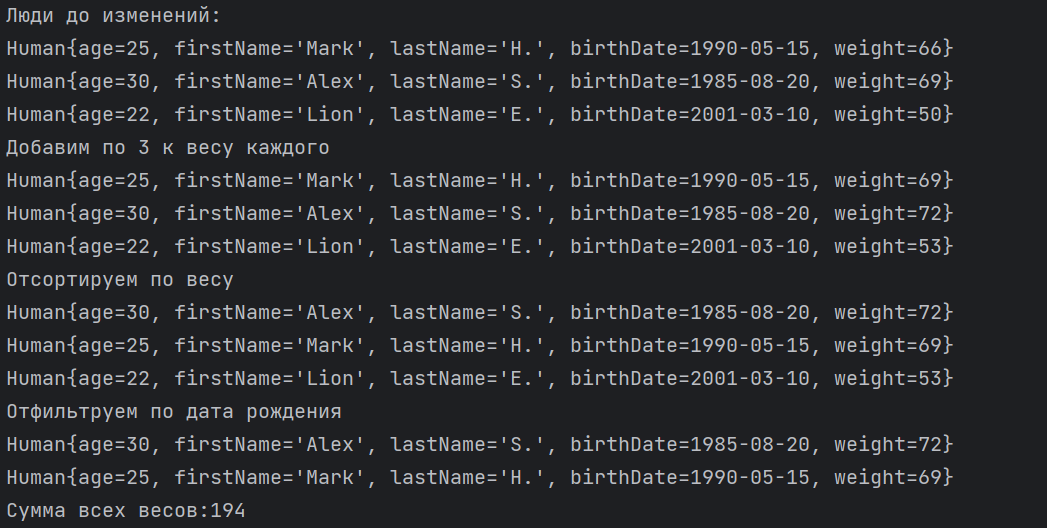
Класс Human:

package prac2;  
  
import java.time.LocalDate;  
  
public class Human {  
 private int age;  
 private String firstName;  
 private String lastName;  
 private 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;  
 }  
  
 public void setWeight(int weight) {  
 this.weight = weight;  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "age=" + age +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", weight=" + weight +  
 '}';  
 }  
}

Класс Main:

package prac2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class Main {  
 public static void main(String[] args) {  
 List<Human> h\_list = new ArrayList<>();  
 h\_list.add(new Human(25, "Mark", "H.", LocalDate.*of*(1990, 5, 15), 66));  
 h\_list.add(new Human(30, "Alex", "S.", LocalDate.*of*(1985, 8, 20), 69));  
 h\_list.add(new Human(22, "Lion", "E.", LocalDate.*of*(2001, 3, 10), 50));  
 System.*out*.println("Люди до изменений:");  
 h\_list.forEach(System.*out*::println);  
 System.*out*.println("Добавим по 3 к весу каждого");  
 h\_list = h\_list.stream()  
 .map(human -> {  
 human.setWeight(human.getWeight()+3);  
 return human;  
 }).collect(Collectors.*toList*());  
 h\_list.forEach(System.*out*::println);  
  
  
 System.*out*.println("Отсортируем по весу");  
 h\_list=h\_list.stream().sorted((human1,human2) -> human2.getWeight() - human1.getWeight()).collect(Collectors.*toList*());  
 h\_list.forEach(System.*out*::println);  
  
  
 System.*out*.println("Отфильтруем по дата рождения");  
 h\_list.stream().filter(human -> human.getBirthDate().isBefore(LocalDate.*of*(2000,1,1))).forEach(System.*out*::println);  
  
  
 int sum = h\_list.stream().mapToInt(Human::getWeight).sum();  
 System.*out*.println("Сумма всех весов:" + sum);  
 }  
}

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: Map с использованием Lock, Set с использованием Semaphore.

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac3

package prac3.part1;  
  
public class Main {  
 public static void main(String[] args) {  
 ConcurrentMap<String, Integer> concurrentMap = new ConcurrentMap<>();  
  
 Thread thread1 = new Thread(() -> {  
 try {  
 for (int i = 0; i < 5; i++) {  
 concurrentMap.put("Key" + i, i);  
 System.*out*.println("Thread 1 added: Key" + i + " -> " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 try {  
 for (int i = 5; i < 10; i++) {  
 concurrentMap.put("Key" + i, i);  
 System.*out*.println("Thread 2 added: Key" + i + " -> " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 }  
}

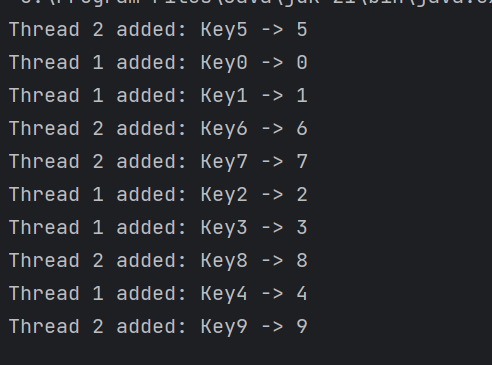
package prac3.part1;  
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class ConcurrentMap<K, V> {  
 private final Map<K, V> map;  
 private final Lock lock;  
  
 public ConcurrentMap() {  
 map = new HashMap<>();  
 lock = new ReentrantLock();  
 }  
  
 public void put(K key, V value) {  
 lock.lock();  
 try {  
 map.put(key, value);  
 } finally {  
 lock.unlock();  
 }  
 }  
}

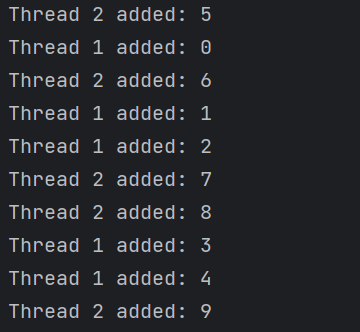
2-я часть:

package prac3;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.concurrent.Semaphore;  
  
public class ConcurrentSet<T> {  
 private Set<T> set;  
 private Semaphore semaphore;  
  
 public ConcurrentSet() {  
 set = new HashSet<>();  
 semaphore = new Semaphore(1); // Одна разрешающая семафор  
 }  
  
 public boolean add(T element) throws InterruptedException {  
 semaphore.acquire(); // Захватываем разрешение  
 boolean added = set.add(element);  
 semaphore.release(); // Освобождаем разрешение  
 return added;  
 }  
}

package prac3;  
  
public class Main {  
 public static void main(String[] args) {  
 ConcurrentSet<Integer> concurrentSet = new ConcurrentSet<>();  
  
 Thread thread1 = new Thread(() -> {  
 try {  
 for (int i = 0; i < 5; i++) {  
 concurrentSet.add(i);  
 System.*out*.println("Thread 1 added: " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 try {  
 for (int i = 5; i < 10; i++) {  
 concurrentSet.add(i);  
 System.*out*.println("Thread 2 added: " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 }  
}

1. **Результат работы программы**





**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Цель: работа с ExecutorService, CompletableFuture

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

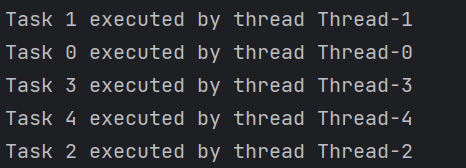
1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac4>

package prac4;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
class CustomExecutorService  
{  
 private final List<WorkerThread> threads;  
 private final BlockingQueue<Runnable> taskQueue;  
  
 public CustomExecutorService(int threadPoolSize)  
 {  
 this.threads = new ArrayList<>(threadPoolSize);  
 this.taskQueue = new LinkedBlockingQueue<>();  
  
 for (int i = 0; i < threadPoolSize; i++)  
 {  
 WorkerThread workerThread = new WorkerThread();  
 threads.add(workerThread);  
 workerThread.start();  
 }  
 }  
  
 public void submit(Runnable task)  
 {  
 try  
 {  
 taskQueue.put(task);  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
  
 public void shutdown()  
 {  
 for (WorkerThread workerThread : threads)  
 {  
 workerThread.interrupt();  
 }  
 }  
  
 private class WorkerThread extends Thread  
 {  
 @Override  
 public void run()  
 {  
 while (!Thread.*currentThread*().isInterrupted())  
 {  
 try  
 {  
 Runnable task = taskQueue.take();  
 task.run();  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }  
 }  
}

package prac4;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 CustomExecutorService executorService = new CustomExecutorService(5);  
  
 for (int i = 0; i < 10; i++)  
 {  
 final int taskNumber = i;  
 executorService.submit(() -> {  
 System.*out*.println("Task " + taskNumber + " executed by thread " + Thread.*currentThread*().getName());  
 });  
 }  
  
 executorService.shutdown();  
 }  
}

1. **Результат работы программы**



**Практическая работа №5**

1. **Постановка задачи и персональный вариант**

Цель: познакомиться с паттернами проектирования, их определением и классификацией. Обзор паттернов GoF. Паттерн Синглтон.

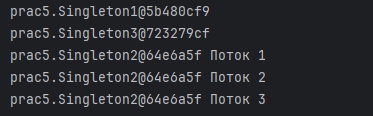
Постановка задачи: реализовать паттерн Singleton как минимум 3-мя способами.

1. **Код**

https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac5

package prac5;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println(Singleton1.*getInstance*());  
 System.*out*.println(Singleton3.*getInstance*());  
 for (int i = 1; i < 4; i++) {  
 var t = new Thread(() -> {  
 System.*out*.println(Singleton2.*getInstance*() + " " + Thread.*currentThread*().getName());  
 });  
 t.setName("Поток " + i);  
 t.start();  
 }  
 }  
}  
  
class Singleton1{//ленивая инициализация  
 private static Singleton1 *instance*;  
 private Singleton1(){}  
 public static Singleton1 getInstance(){  
 if (*instance* == null){  
 *instance* = new Singleton1();  
 }  
 return *instance*;  
 }  
}  
class Singleton2 {//потокобезопасный  
 private static Singleton2 *instance*;  
 private Singleton2(){}  
 public static synchronized Singleton2 getInstance(){  
 if (*instance* == null){  
 *instance* = new Singleton2();  
 }  
 return *instance*;  
 }  
}  
class Singleton3 {//холодная инициализация  
 private static final Singleton3 *instance* = new Singleton3();  
 private Singleton3() {}  
 public static Singleton3 getInstance() {  
 return *instance*;  
 }  
}

1. **Результат работы программы**



**Практическая работа №6**

1. **Постановка задачи и персональный вариант**

Цель: знакомство с реализацией порождающих паттернов проектирования.

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

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac4>

package prac4;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
class CustomExecutorService  
{  
 private final List<WorkerThread> threads;  
 private final BlockingQueue<Runnable> taskQueue;  
  
 public CustomExecutorService(int threadPoolSize)  
 {  
 this.threads = new ArrayList<>(threadPoolSize);  
 this.taskQueue = new LinkedBlockingQueue<>();  
  
 for (int i = 0; i < threadPoolSize; i++)  
 {  
 WorkerThread workerThread = new WorkerThread();  
 threads.add(workerThread);  
 workerThread.start();  
 }  
 }  
  
 public void submit(Runnable task)  
 {  
 try  
 {  
 taskQueue.put(task);  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
  
 public void shutdown()  
 {  
 for (WorkerThread workerThread : threads)  
 {  
 workerThread.interrupt();  
 }  
 }  
  
 private class WorkerThread extends Thread  
 {  
 @Override  
 public void run()  
 {  
 while (!Thread.*currentThread*().isInterrupted())  
 {  
 try  
 {  
 Runnable task = taskQueue.take();  
 task.run();  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }  
 }  
}

package prac4;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 CustomExecutorService executorService = new CustomExecutorService(5);  
  
 for (int i = 0; i < 10; i++)  
 {  
 final int taskNumber = i;  
 executorService.submit(() -> {  
 System.*out*.println("Task " + taskNumber + " executed by thread " + Thread.*currentThread*().getName());  
 });  
 }  
  
 executorService.shutdown();  
 }  
}

1. **Результат работы программы**

