|  |
| --- |
|  |
| 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** | ***Исаков Л.А.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

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

Москва 2024

**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.

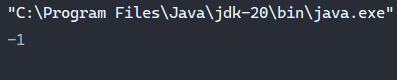
Постановка задачи: имплементировать интерфейс Comparator, сравнивающий два массива с одинаковыми типами элементов по количеству элементов в данных массивах.

1. **Код**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task1

package Task1;  
  
import java.util.Comparator;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 Comparator<Object[]> comparator = (arr1, arr2) -> arr1.length - arr2.length;  
 var newArr1 = new Integer[] {1, 2};  
 var newArr2 = new Integer[] {2, 3, 4};  
 System.out.println(comparator.compare(newArr1, newArr2));  
 }  
}

1. **Результат работы программы**



**Практическая работа №2**

1. **Постановка задачи и персональный вариант**

Тема: Работа со Stream API в Java 8.

Постановка задачи: фильтрация по возрасту больше, чем 20, сортировка по последней букве имени, увеличение возраста каждого на 3, вычисление среднего возраста всех элементов.

1. **Код**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task2

package Task2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Comparator;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 var list = new ArrayList<Human>();  
 list.add(new Human(18, "Alex", "Alexandrov", LocalDate.of(2005, 7, 1), 60));  
 list.add(new Human(19, "Sveta", "Oblakova", LocalDate.of(2004, 7, 1), 55));  
 list.add(new Human(20, "Lesha", "Terekov", LocalDate.of(2003, 7, 1), 50));  
 list.add(new Human(21, "Timur", "Ivanov", LocalDate.of(2002, 7, 1), 45));  
 list.add(new Human(22, "Denis", "Petrov", LocalDate.of(2001, 7, 1), 40));  
 list.add(new Human(23, "Nastya", "Sidorova", LocalDate.of(2000, 7, 1), 35));  
  
 var stream = list.stream();  
 var av = stream.filter(human -> human.age > 20)  
 .sorted(new HumanComparator())  
 .peek(human -> human.age += 3)  
 .mapToInt(Human::getAge).average();  
  
 System.out.println(av);  
 }  
}  
  
class HumanComparator implements Comparator<Human>  
{  
 @Override  
 public int compare(Human x1, Human x2) {  
 return x1.firstName.substring(x1.firstName.length() - 1)  
 .compareTo(x2.firstName.substring(x2.firstName.length() - 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: List с использованием Semaphore и Map с использованием ключевого слова synchronized.

1. **Код**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task3\_1

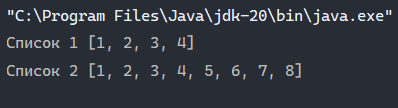
package Task3.Task3\_1;  
  
import java.util.ArrayList;  
import java.util.concurrent.Semaphore;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 var semaphore = new Semaphore(1);  
 var list = new ArrayList<Integer>();  
  
 var writer1 = new Thread(() -> {  
 try  
 {  
 semaphore.acquire();  
 list.add(1);  
 list.add(2);  
 list.add(3);  
 list.add(4);  
 System.out.println("Список 1 " + list);  
 } catch (InterruptedException e)  
 {  
 throw new RuntimeException(e);  
 }  
 finally  
 {  
 semaphore.release();  
 }  
 });  
  
 var writer2 = new Thread(() -> {  
 try  
 {  
 semaphore.acquire();  
 list.add(5);  
 list.add(6);  
 list.add(7);  
 list.add(8);  
 System.out.println("Список 2 " + list);  
 } catch (InterruptedException e)  
 {  
 throw new RuntimeException(e);  
 }  
 finally  
 {  
 semaphore.release();  
 }  
 });  
  
 writer1.start();  
 writer2.start();  
 }  
}

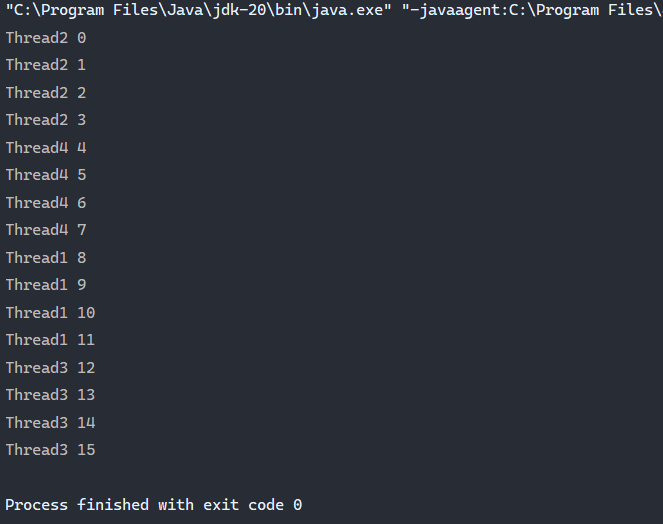
https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task3\_2

package Task3.Task3\_2;

import java.util.HashMap;  
  
public class Main  
{  
 public static HashMap<String, Integer> synchronizedMap = new HashMap<>();  
 public static int iterator = 0;  
  
 public static void main(String[] args)  
 {  
 // Map с использованием ключевого слова synchronized  
 for (int i = 1; i < 5; i++)  
 {  
 var t = new Thread(() -> {  
 synchronized (synchronizedMap) {  
 for (int j = iterator; j < iterator + 4; j++)  
 {  
 synchronizedMap.put("key" + j, j);  
 System.out.println(Thread.currentThread().getName() + " " + j);  
 }  
 iterator+=4;  
 }  
 });  
 t.setName("Thread" + i);  
 t.start();  
 }  
 }  
}

1. **Результат работы программы**





**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Тема: работа с ExecutorService, CompletableFuture.

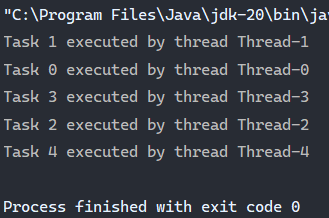
Постановка задачи: реализовать собственную имплементацию ExecutorService с единственным параметром конструктора – количеством потоков.

1. **Код**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4

package Task4;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
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();  
 }  
}  
  
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();  
 }  
 }  
 }  
 }  
}

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.

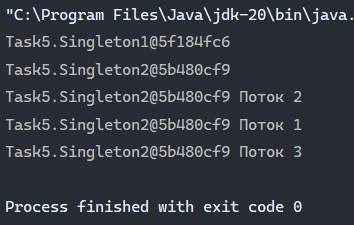
Постановка задачи: реализовать паттерн Singleton как минимум 3-мя способами.

1. **Код**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task5

package Task5;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 // Реализовать паттерн Singleton как минимум 3-мя способами  
 System.out.println(Singleton1.getInstance());  
 System.out.println(Singleton2.getInstance());  
 for (int i = 1; i < 4; i++)  
 {  
 var t = new Thread(() -> {  
 System.out.println(Singleton3.getInstance() + " " + Thread.currentThread().getName());  
 });  
 t.setName("Поток " + i);  
 t.start();  
 }  
 }  
}  
  
// Прстой пример  
class Singleton1  
{  
 private static final Singleton1 Instance = new Singleton1();  
  
 private Singleton1() { }  
  
 public static Singleton1 getInstance()  
 {  
 return Instance;  
 }  
}  
  
// Ленивая инициализация  
class Singleton2  
{  
 private static Singleton2 Instance;  
  
 private Singleton2() { }  
  
 public static Singleton2 getInstance()  
 {  
 if (Instance == null)  
 {  
 Instance = new Singleton2();  
 }  
 return Instance;  
 }  
}  
  
// Потокобезопасный  
class Singleton3  
{  
 private Singleton3() { }  
  
 private static final class InstanceHolder  
 {  
 private static final Singleton3 Instance = new Singleton3();  
 }  
  
 public static synchronized Singleton3 getInstance()  
 {  
 return InstanceHolder.Instance;  
 }  
}

1. **Результат работы программы**



**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**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4/Main

package Task6;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 // «Фабричный метод»  
 System.out.println("Паттерн «Фабричный метод»:");  
 new FactoryMethod();  
 System.out.println();  
  
 // «Абстрактная фабрика»  
 System.out.println("Паттерн «Абстрактная фабрика»:");  
 new AbstractFactory();  
 System.out.println();  
  
 // «Строитель»  
 System.out.println("Паттерн «Строитель»:");  
 new Builder();  
 System.out.println();  
  
 // «Прототип»  
 System.out.println("Паттерн «Прототип»:");  
 new Prototype();  
 }  
}

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4/FactoryMethod

package Task6;  
  
public class FactoryMethod  
{  
 public FactoryMethod()  
 {  
 Work();  
 }  
  
 private void Work()  
 {  
 Factory factory = new PanelFactory("Завод панельных домов");  
 factory.BuildHouse();  
 factory = new WoodFactory("Завод деревянных домов");  
 factory.BuildHouse();  
 }  
}  
  
// Фабрики  
abstract class Factory  
{  
 public String Name;  
  
 public Factory(String name)  
 {  
 Name = name;  
 }  
  
 abstract House BuildHouse();  
}  
  
class PanelFactory extends Factory  
{  
 public PanelFactory(String name)  
 {  
 super(name);  
 }  
  
 @Override  
 House BuildHouse()  
 {  
 return new PanelHouse();  
 }  
}  
  
class WoodFactory extends Factory  
{  
 public WoodFactory(String name)  
 {  
 super(name);  
 }  
  
 @Override  
 House BuildHouse()  
 {  
 return new WoodHouse();  
 }  
}  
  
// Виды домов  
interface House { }  
  
class PanelHouse implements House  
{  
 public PanelHouse()  
 {  
 System.out.println("Panel house!");  
 }  
}  
  
class WoodHouse implements House  
{  
 public WoodHouse()  
 {  
 System.out.println("Wooden house!");  
 }  
}

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4/AbstractFactory

package Task6;  
  
public class AbstractFactory  
{  
 public AbstractFactory()  
 {  
 Work();  
 }  
  
 private void Work()  
 {  
 var hero1 = new Hero(new ElfFactory());  
 var hero2 = new Hero(new TankFactory());  
 hero1.Attack();  
 hero2.Attack();  
 }  
}  
  
// Оружие  
abstract class Weapon  
{  
 abstract void Hit();  
}  
class Sword extends Weapon  
{  
 public void Hit()  
 {  
 System.out.println("Удар мечом!");  
 }  
}  
class Bow extends Weapon  
{  
 public void Hit()  
 {  
 System.out.println("Выстрел из лука!");  
 }  
}  
  
// Герой  
class Hero  
{  
 private Weapon HeroWeapon;  
  
 public Hero(HeroFactory factory)  
 {  
 HeroWeapon = factory.CreateWeapon();  
 }  
  
 public void Attack()  
 {  
 HeroWeapon.Hit();  
 }  
}  
  
// Фабрики  
abstract class HeroFactory  
{  
 abstract Weapon CreateWeapon();  
}  
class TankFactory extends HeroFactory  
{  
 public Weapon CreateWeapon()  
 {  
 return new Sword();  
 }  
}  
class ElfFactory extends HeroFactory  
{  
 public Weapon CreateWeapon()  
 {  
 return new Bow();  
 }  
}

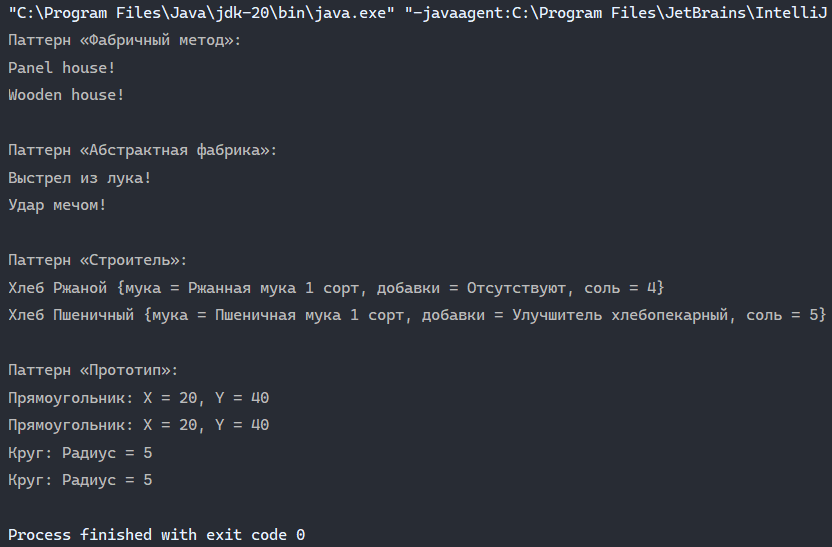
https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4/Builder

package Task6;  
  
public class Builder  
{  
 public Builder()  
 {  
 Work();  
 }  
  
 private void Work()  
 {  
 var baker = new Baker();  
 var bread1 = baker.CreateBread(new RyeBreadBuilder());  
 System.out.println(bread1);  
 var bread2 = baker.CreateBread(new WheatBreadBuilder());  
 System.out.println(bread2);  
 }  
}  
  
// Для хлеба  
class Flour  
{  
 public String FlourType;  
  
 public Flour(String flourType)  
 {  
 FlourType = flourType;  
 }  
}  
class Additive  
{  
 public String AdditiveName;  
  
 public Additive(String additiveName)  
 {  
 AdditiveName = additiveName;  
 }  
}  
class Salt  
{  
 public int saltCount;  
  
 public Salt(int count)  
 {  
 saltCount = count;  
 }  
}  
  
class Bread  
{  
 public String name;  
 public Flour flour;  
 public Additive additive;  
 public Salt salt;  
  
 @Override  
 public String toString()  
 {  
 return "Хлеб " + name + " {" +  
 "мука = " + flour.FlourType +  
 ", добавки = " + additive.AdditiveName +  
 ", соль = " + salt.saltCount + "}";  
 }  
}  
  
// Получение хлеба  
class Baker  
{  
 public Bread CreateBread(BreadBuilder builder)  
 {  
 builder.SetName();  
 builder.SetFlour();  
 builder.SetAdditive();  
 builder.SetSalt();  
 return builder.bread;  
 }  
}  
  
abstract class BreadBuilder  
{  
 public Bread bread;  
  
 public BreadBuilder()  
 {  
 bread = new Bread();  
 }  
  
 abstract void SetName();  
 abstract void SetFlour();  
 abstract void SetAdditive();  
 abstract void SetSalt();  
}  
  
class RyeBreadBuilder extends BreadBuilder  
{  
 void SetName()  
 {  
 this.bread.name = "Ржаной";  
 }  
 void SetFlour()  
 {  
 this.bread.flour = new Flour("Ржанная мука 1 сорт");  
 }  
 void SetAdditive()  
 {  
 this.bread.additive = new Additive("Отсутствуют");  
 }  
 void SetSalt()  
 {  
 this.bread.salt = new Salt(4);  
 }  
}  
class WheatBreadBuilder extends BreadBuilder  
{  
 void SetName()  
 {  
 this.bread.name = "Пшеничный";  
 }  
 void SetFlour()  
 {  
 this.bread.flour = new Flour("Пшеничная мука 1 сорт");  
 }  
 void SetAdditive()  
 {  
 this.bread.additive = new Additive("Улучшитель хлебопекарный");  
 }  
 void SetSalt()  
 {  
 this.bread.salt = new Salt(5);  
 }  
}

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task4/Prototype

package Task6;  
  
public class Prototype  
{  
 public Prototype()  
 {  
 Work();  
 }  
  
 private void Work()  
 {  
 var figure1 = new Rectangle(20, 40);  
 var figure1\_1 = figure1.Clone();  
 figure1.GetInfo();  
 figure1\_1.GetInfo();  
  
 var figure2 = new Circle(5);  
 var figure2\_1 = figure2.Clone();  
 figure2.GetInfo();  
 figure2\_1.GetInfo();  
 }  
}  
  
interface IFigure  
{  
 IFigure Clone();  
 void GetInfo();  
}  
  
class Rectangle implements IFigure  
{  
 private int X;  
 private int Y;  
  
 public Rectangle(int x, int y)  
 {  
 X = x;  
 Y = y;  
 }  
  
 public IFigure Clone()  
 {  
 return new Rectangle(this.X, this.Y);  
 }  
  
 public void GetInfo()  
 {  
 System.out.println("Прямоугольник: X = " + X + ", Y = " + Y);  
 }  
}  
  
class Circle implements IFigure  
{  
 private int Radius;  
  
 public Circle(int radius)  
 {  
 Radius = radius;  
 }  
  
 public IFigure Clone()  
 {  
 return new Circle(this.Radius);  
 }  
  
 public void GetInfo()  
 {  
 System.out.println("Круг: Радиус = " + Radius);  
 }  
}

1. **Результат работы программы**



**Практическая работа №7**

1. **Постановка задачи и персональный вариант**

Тема:

Постановка задачи:

1. **Код**

https://github.com/Peygy/JavaPatterns/tree/main/Tasks/src/Task7

1. **Результат работы программы**