|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №1-24**

**по дисциплине «Шаблоны программных платформ языка Джава»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил:  Студент группы ИКБО-20-22 | « 27 » мая 2024г. | | (подпись) | | Фридрих А.В. | |
|  | |  | |  | |  | |
| Принял:  Преподаватель кафедры ИиППО ИИТ | | « 27 » мая 2024г. | | (подпись) | | Ермаков С.Р. | |

Москва 2023 г.

**Содержание**

[1 Практическая работа №1 7](#_Toc152328307)

[1.1 Цель работы. 7](#_Toc152328308)

[1.2 Задание. 7](#_Toc152328309)

[1.3 Тестирование. 9](#_Toc152328310)

[1.4 Выводы. 9](#_Toc152328311)

[2 Практическая работа №2 10](#_Toc152328312)

[2.1 Цель работы. 10](#_Toc152328313)

[2.2 Задание. 10](#_Toc152328314)

[2.3 Код программы. 10](#_Toc152328315)

[2.4 Тестирование. 12](#_Toc152328316)

[2.5 Вывод. 12](#_Toc152328317)

[3 Практическая работа №3 13](#_Toc152328318)

[3.1 Цель работы. 13](#_Toc152328319)

[3.2 Задание. 13](#_Toc152328320)

[3.3 Код программы. 13](#_Toc152328321)

[3.4 Тестирование. 14](#_Toc152328322)

[3.5 Вывод. 15](#_Toc152328323)

[4 Практическая работа №4 16](#_Toc152328324)

[4.1 Цель работы. 16](#_Toc152328325)

[4.2 Задание. 16](#_Toc152328326)

[4.3 Код программы. 16](#_Toc152328327)

[4.4 Тестирование. 19](#_Toc152328328)

[4.5 Вывод. 19](#_Toc152328329)

[5 Практическая работа №5 20](#_Toc152328330)

[5.1 Цель работы. 20](#_Toc152328331)

[5.2 Задание. 20](#_Toc152328332)

[5.3 Код программы. 22](#_Toc152328333)

[5.4 Тестирование. 36](#_Toc152328334)

[5.5 Вывод. 36](#_Toc152328335)

[6 Практическая работа №6 37](#_Toc152328336)

[6.1 Цель работы. 37](#_Toc152328337)

[6.2 Задание. 37](#_Toc152328338)

[6.3 Код программы. 37](#_Toc152328339)

[6.4 Тестирование. 41](#_Toc152328340)

[6.5 Вывод. 41](#_Toc152328341)

[7 Практическая работа №7 42](#_Toc152328342)

[7.1 Цель. 42](#_Toc152328343)

[7.2 Задание. 42](#_Toc152328344)

[7.3 Код программы. 42](#_Toc152328345)

[7.4 Тестирование. 44](#_Toc152328346)

[7.5 Вывод. 44](#_Toc152328347)

[8 Практическая работа №8 45](#_Toc152328348)

[8.1 Цель работы. 45](#_Toc152328349)

[8.2 Задание. 45](#_Toc152328350)

[8.3 Код программы. 45](#_Toc152328351)

[8.4 Тестирование. 51](#_Toc152328352)

[8.5 Вывод. 51](#_Toc152328353)

[9 Практические работы №9 52](#_Toc152328354)

[9.1 Цель работы. 52](#_Toc152328355)

[9.2 Задание. 52](#_Toc152328356)

[9.3 Тестирование. 54](#_Toc152328357)

[9.4 Вывод. 54](#_Toc152328358)

[10 Практическая работа №10 55](#_Toc152328359)

[10.1 Цель работы. 55](#_Toc152328360)

[10.2 Задание. 55](#_Toc152328361)

[10.3 Код программы. 55](#_Toc152328362)

[10.4 Тестирование. 58](#_Toc152328363)

[10.5 Вывод. 58](#_Toc152328364)

[11 Практическое задание №11 59](#_Toc152328365)

[11.1 Цель работы. 59](#_Toc152328366)

[11.2 Задание. 59](#_Toc152328367)

[11.3 Код программы. 59](#_Toc152328368)

[11.4 Тестиривание. 66](#_Toc152328369)

[11.5 Вывод. 67](#_Toc152328370)

[12 Практическая работа №12 68](#_Toc152328371)

[12.1 Цель работы. 68](#_Toc152328372)

[12.2 Задание. 68](#_Toc152328373)

[12.3 Код программы. 68](#_Toc152328374)

[12.4 Тестирование. 75](#_Toc152328375)

[12.5 Вывод. 75](#_Toc152328376)

[13 Практическая работа №13 76](#_Toc152328377)

[13.1 Цель работы. 76](#_Toc152328378)

[13.2 Задание. 77](#_Toc152328379)

[13.3 Код программы. 77](#_Toc152328380)

[13.4 Тестирование. 77](#_Toc152328381)

[13.5 Вывод. 78](#_Toc152328382)

[14 Практическая работа №14 78](#_Toc152328383)

[14.1 Цель работы. 78](#_Toc152328384)

[14.2 Задание. 78](#_Toc152328385)

[14.3 Код программы. 79](#_Toc152328386)

[14.4 Тестирование. 86](#_Toc152328387)

[14.5 Вывод. 86](#_Toc152328388)

[15 Практическая работа №15 87](#_Toc152328389)

[15.1 Цель работы. 87](#_Toc152328390)

[15.2 Задание. 87](#_Toc152328391)

[15.3 Код программы. 87](#_Toc152328392)

[15.4 Тестирование. 92](#_Toc152328393)

[15.5 Вывод. 92](#_Toc152328394)

[16 Практическая работа №16 93](#_Toc152328395)

[16.1 Цель работы. 93](#_Toc152328396)

[16.2 Задание. 93](#_Toc152328397)

[16.3 Код программы. 93](#_Toc152328398)

[16.4 Тестирование. 97](#_Toc152328399)

[16.5 Вывод. 97](#_Toc152328400)

[17 Практическая работа №17 98](#_Toc152328401)

[17.1 Цель работы. 98](#_Toc152328402)

[17.2 Задание. 98](#_Toc152328403)

[17.3 Код программы. 98](#_Toc152328404)

[17.4 Тестирование. 100](#_Toc152328405)

[17.5 Вывод. 100](#_Toc152328406)

[18 Практическая работа №18 101](#_Toc152328407)

[18.1 Цель работы. 101](#_Toc152328408)

[18.2 Задание. 101](#_Toc152328409)

[18.3 Код программы. 101](#_Toc152328410)

[18.4 Тестирование. 105](#_Toc152328411)

[18.5 Вывод. 105](#_Toc152328412)

[19 Практическая работа №19 106](#_Toc152328413)

[19.1 Цель работы. 106](#_Toc152328414)

[19.2 Задание. 106](#_Toc152328415)

[19.3 Код программы. 106](#_Toc152328416)

[19.4 Тестирование. 107](#_Toc152328417)

[19.5 Вывод. 107](#_Toc152328418)

[20 Практическая работа №20 108](#_Toc152328419)

[20.1 Цель работы. 108](#_Toc152328420)

[20.2 Задание. 108](#_Toc152328421)

[20.3 Код программы. 108](#_Toc152328422)

[20.4 Тестирование. 112](#_Toc152328423)

[20.5 Вывод. 112](#_Toc152328424)

[21 Практическая работа №21 113](#_Toc152328425)

[21.1 Цель работы. 113](#_Toc152328426)

[21.2 Задание. 113](#_Toc152328427)

[21.3 Код программы. 113](#_Toc152328428)

[21.4 Тестирование. 123](#_Toc152328429)

[21.5 Вывод. 123](#_Toc152328430)

[22 Практическая работа №22 124](#_Toc152328431)

[22.1 Цель работы. 124](#_Toc152328432)

[22.2 Задание. 124](#_Toc152328433)

[22.3 Код программы. 124](#_Toc152328434)

[22.4 Тестирование. 130](#_Toc152328435)

[22.5 Вывод. 130](#_Toc152328436)

[23 Практическая работа №23 131](#_Toc152328437)

[23.1 Цель работы. 131](#_Toc152328438)

[23.2 Задание. 131](#_Toc152328439)

[23.3 Код программы. 132](#_Toc152328440)

[23.4 Тестирование. 147](#_Toc152328441)

[23.5 Вывод. 147](#_Toc152328442)

[24 Практическая работа №24 148](#_Toc152328443)

[24.1 Цель работы. 148](#_Toc152328444)

[24.2 Задание. 148](#_Toc152328445)

[24.3 Код программы. 149](#_Toc152328446)

[24.4 Тестирование. 179](#_Toc152328447)

[24.5 Вывод. 179](#_Toc152328448)

[25 Список литературы 180](#_Toc152328449)

# Практическая работа №1

## Цель работы.

Знакомство со втроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыквания. Предикаты. Функции. Компараторы.

## Задание.

Имплементировать интерфейс Predicate, определяющий, содержит

ли массив студентов студента с максимальным количеством баллов

(максимальное значение – 100)

Код программы.

package pra1;  
  
public class Student {  
 public Integer id;  
 public Integer ball;  
  
 public Student(Integer id, Integer ball) {  
 this.id = id;  
 this.ball = ball;  
 }  
}

Листинг 1.1 – класс Student.

package pra1;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.function.Predicate;  
  
public class Main {  
  
 public static void main(String[] args) {  
 Predicate<List<Student>> hasMax = (students)->{  
 for(Student s : students){  
 if(s.ball==100) return true;  
 }  
 return false;  
 };  
 var st1 = new Student(1, 50);  
 var st2 = new Student(2, 100);  
 var st3 = new Student(3, 40);  
 var without = new ArrayList<Student>();  
 without.addAll(List.*of*(st1, st3));  
 var with = new ArrayList<Student>();  
 with.addAll(List.*of*(st1,st2,st3));  
 System.out.println(hasMax.test(without));  
 System.out.println(hasMax.test(with));  
 }  
}

Листинг 1.2 – код класса Main.

## Тестирование.



Рисунок 1 - результат работы программы.

## Выводы.

Освоили работу с функциональными интерфейсами в Java.

# Практическая работа №2

## Цель работы.

Работа со Stream API в Java 8.

## Задание.

Сортировка по последней букве фамилии, фильтрация по признаку

«возраст больше, чем вес», сортировка по дате рождения, произведение всех возрастов.

## Код программы.

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

Листинг 2.1 – код класса Human.

import java.time.LocalDate;  
import java.time.Month;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
public class HumanWorker {  
 private List<Human> humans;  
 public HumanWorker(){  
 this.humans = new ArrayList<>();  
 var h1 = new Human(20, "Aleksander", "Fridrikh", LocalDate.*of*(2004, Month.AUGUST, 13), 85);  
 var h2 = new Human(40, "Ivan", "Pupkin", LocalDate.*of*(1984, Month.FEBRUARY, 17), 65);  
 var h3 = new Human(13, "Katerina", "Melkaya", LocalDate.*of*(2011, Month.JULY, 3), 48);  
 var h4 = new Human(70, "Vladislav", "Rakov", LocalDate.*of*(1954, Month.MARCH, 27), 60);  
 var h5 = new Human(66, "Vladislava", "Rakova", LocalDate.*of*(1958, Month.JANUARY, 1), 49);  
 var h6 = new Human(34, "Artemon", "Bogomolov", LocalDate.*of*(1990, Month.NOVEMBER, 20), 140);  
 var h7 = new Human(34, "Slava", "Lapov", LocalDate.*of*(1990, Month.MAY, 9), 80);  
 var h8 = new Human(15, "Aleksandr", "Pubertat", LocalDate.*of*(2009, Month.APRIL, 5), 60);  
 var h9 = new Human(18, "Anton", "Checkov", LocalDate.*of*(2006, Month.SEPTEMBER, 14), 75);  
 var h10 = new Human(23, "Maria", "Chernova", LocalDate.*of*(2001, Month.JUNE, 25), 55);  
 this.humans.addAll(List.*of*(h1,h2,h3,h4,h5,h6,h7,h8,h9,h10));  
 }  
 public List<Human> sortByLastSymb(){  
 return this.humans.stream().sorted(Comparator.*comparingInt*(h -> h.lastName.charAt(h.lastName.length() - 1))).collect(Collectors.*toList*());  
 }  
 public List<Human> filterByWeight(){  
 return this.humans.stream().filter(h->h.age>h.weight).collect(Collectors.*toList*());  
 }  
 public List<Human> sortByDate(){  
 return this.humans.stream().sorted(Comparator.*comparing*(h -> h.birthDate)).collect(Collectors.*toList*());  
 }  
 public Integer getMultiplication() throws Exception{  
 return this.humans.stream().map(h->h.age).reduce((h1,h2)->h1\*h2).orElseThrow(()->new Exception("can't get multiplication"));  
 }  
}

Листинг 2.2 – код класса HumanWorker.

public class Main {  
 public static void main(String[] args) {  
 var test = new HumanWorker();  
 System.out.println("SORTED");  
 test.sortByLastSymb().forEach(System.out::println);  
 System.out.println("FILTERING");  
 test.filterByWeight().forEach(System.out::println);  
 System.out.println("SORTED BY DATE");  
 test.sortByDate().forEach(System.out::println);  
 System.out.println("GET MULTIPLICATION");  
 try {  
 System.out.println(test.getMultiplication());  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }  
 }  
}

Листинг 2.3 – код класса Main.

## Тестирование.

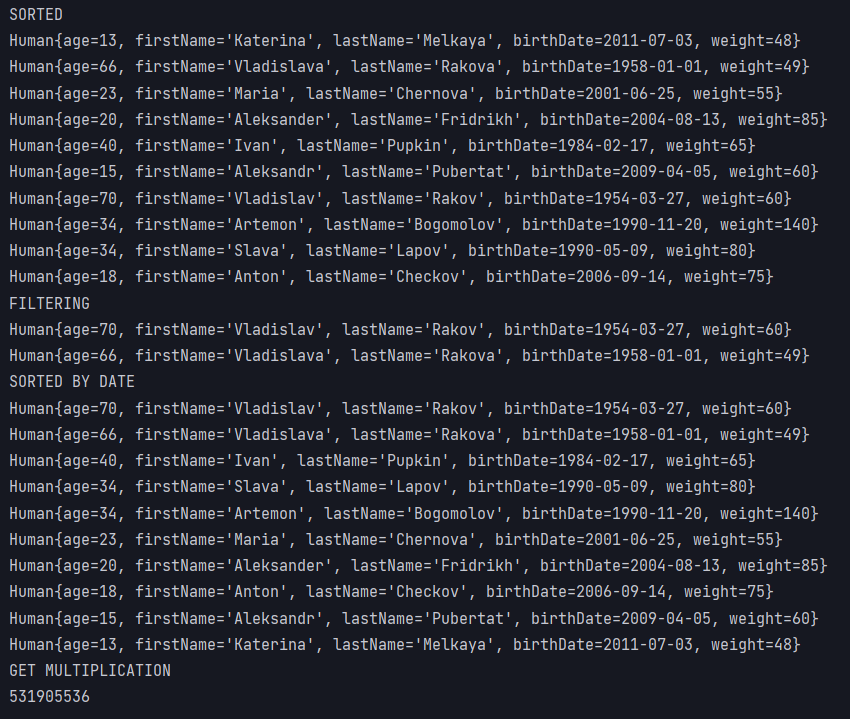


Рисунок 2 - результат работы программы.

## Вывод.

Получил знания и практические навыки по работе с Stream API в Java 8.

# Практическая работа №3

## Цель работы.

Знакомство с конкурентным программированием в Java. Потокобезопасность, ключевое слово syncrhonized, мьютексы, семафоры, мониторы, барьеры.

## Задание.

Реализовать потокобезопасную коллекцию Мap с использованием семафора, а также потокобезопасный Set с использованием ключевого слова synchronized.

## Код программы.

import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.Semaphore;  
  
public class MapSemaphore {  
 public static void main(String[] args) {  
 Map<Integer, String> syncMap = new HashMap<>();  
 var semaphore = new Semaphore(1);  
 var first = new Thread(()->{  
 try{  
 semaphore.acquire();  
 System.out.println("First thread");  
 syncMap.put(1, "Clynic");  
 syncMap.put(2, "K");  
 syncMap.put(3, "A");  
 syncMap.put(4, "D");  
 syncMap.keySet().stream().forEach(k->{  
 System.out.println(k+":"+syncMap.get(k));  
 });  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }finally{  
 semaphore.release();  
 }  
 });  
 var second = new Thread(()->{  
 try{  
 semaphore.acquire();  
 System.out.println("Second thread");  
 syncMap.put(1, "NIKNOJIsfbh;Op");  
 syncMap.put(2, "A");  
 syncMap.put(3, "L");  
 syncMap.put(4, "O");  
 syncMap.keySet().stream().forEach(k->{  
 System.out.println(k+":"+syncMap.get(k));  
 });  
 }catch (Exception e){  
 System.out.println(e.getMessage());  
 }finally{  
 semaphore.release();  
 }  
 });  
 first.start();  
 second.start();  
 }  
}

Листинг 3.1 – код класса MapSemaphore.

import java.util.HashSet;  
import java.util.Set;  
  
public class SetSync {  
 public static void main(String[] args) {  
 Set<Integer> syncSet = new HashSet<>();  
 var first = new Thread(()->{  
 synchronized (syncSet){  
 System.out.println("First thread");  
 syncSet.add(1);  
 syncSet.add(2);  
 syncSet.add(3);  
 System.out.println(syncSet);  
 }  
 });  
 var second = new Thread(()->{  
 synchronized (syncSet){  
 System.out.println("Second thread");  
 syncSet.add(4);  
 syncSet.add(5);  
 syncSet.add(6);  
 System.out.println(syncSet);  
 }  
 });  
 var third = new Thread(()->{  
 synchronized (syncSet){  
 System.out.println("Third thread");  
 syncSet.add(7);  
 syncSet.add(8);  
 syncSet.add(9);  
 System.out.println(syncSet);  
 }  
 });  
 first.start();  
 second.start();  
 third.start();  
  
 }  
}

Листинг 3.2 – код класса SyncSet.

## Тестирование.

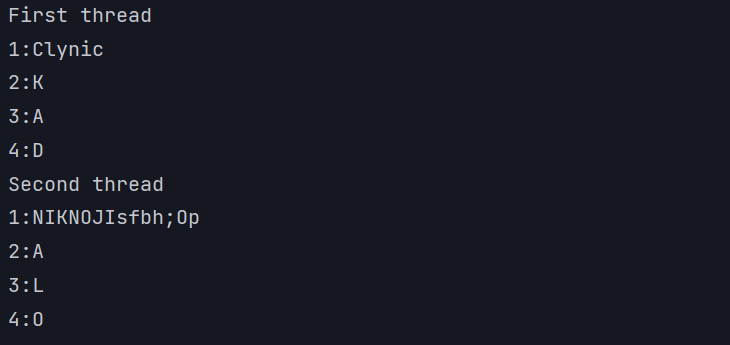


Рисунок 3 - результат работы MapSemaphore.

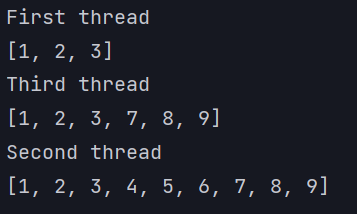


Рисунок 4 - результат работы SyncSet.

## Вывод.

Изучены основы работы с потоками. Реализованы потокобезопасные коллекции.

# Практическая работа №4

## Цель работы.

Работа с ExecutorService, CompletableFuture.

## Задание.

Реализовать собственную имплементацию ExecutorService с единственным параметром конструктора – количеством потоков.

## Код программы.

import java.util.Collection;  
import java.util.List;  
import java.util.concurrent.\*;  
  
public class Realization implements ExecutorService {  
 private ExecutorService fs;  
 public Realization(int count){  
 fs = Executors.newFixedThreadPool(count);  
 }  
 @Override  
 public void shutdown() {  
 fs.shutdown();  
 }  
  
 @Override  
 public List<Runnable> shutdownNow() {  
 return fs.shutdownNow();  
 }  
  
 @Override  
 public boolean isShutdown() {  
 return fs.isShutdown();  
 }  
  
 @Override  
 public boolean isTerminated() {  
 return fs.isTerminated();  
 }  
  
 @Override  
 public boolean awaitTermination(long timeout, TimeUnit unit) throws InterruptedException {  
 return fs.awaitTermination(timeout, unit);  
 }  
  
 @Override  
 public <T> Future<T> submit(Callable<T> task) {  
 return fs.submit(task);  
 }  
  
 @Override  
 public <T> Future<T> submit(Runnable task, T result) {  
 return fs.submit(task, result);  
 }  
  
 @Override  
 public Future<?> submit(Runnable task) {  
 return fs.submit(task);  
 }  
  
 @Override  
 public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws InterruptedException {  
 return fs.invokeAll(tasks);  
 }  
  
 @Override  
 public <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit) throws InterruptedException {  
 return invokeAll(tasks, timeout, unit);  
 }  
  
 @Override  
 public <T> T invokeAny(Collection<? extends Callable<T>> tasks) throws InterruptedException, ExecutionException {  
 return fs.invokeAny(tasks);  
 }  
  
 @Override  
 public <T> T invokeAny(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException {  
 return fs.invokeAny(tasks, timeout, unit);  
 }  
  
 @Override  
 public void execute(Runnable command) {  
 fs.execute(command);  
 }  
}

Листинг 4.1 – код класса Realization.

public class Main {  
 public static void main(String[] args) {  
 var result = new Realization(5);  
 result.submit(() -> {  
 try {  
 Thread.*sleep*(200);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.out.println("We run it");  
 });  
 result.submit(() -> System.out.println("Start"));  
 }  
}

Листинг 4.2 – код класса Main.

## Тестирование.

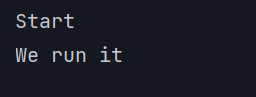


Рисунок 5 – результат тестирования программы.

## Вывод.

Изучена работа с ExecutorService, CompletableFuture.

# Практическая работа №5

## Цель работы.

Познакомиться с паттернами проектирования, их определением и

классификацией. Обзор паттернов GoF. Паттерн Синглтон.

## Задание.

Реализовать паттерн Singleton как минимум 3-мя способами.

## Код программы.

public class Singleton1 {  
 private static Singleton1 INSTANCE = new Singleton1();  
 private Singleton1(){}  
 public void printMessage(){  
 System.out.println("Message from singleton 1");  
 }  
 public static Singleton1 getInstance(){  
 return Singleton1.INSTANCE;  
 }  
}

Листинг 5.1 – код класса Singleton1.

public class Singleton2{  
 private static Singleton2 INSTANCE;  
  
 private Singleton2() {}  
  
 public static Singleton2 getInstance() {  
 if (INSTANCE == null) {  
 INSTANCE = new Singleton2();  
 }  
 return INSTANCE;  
 }  
 public void printMessage(){  
 System.out.println("Message from singleton2");  
 }  
}

Листинг 5.2 – код класса Singleton2.

public class Singleton3 {  
  
 private Singleton3() {  
 }  
 public void printMessage(){  
 System.out.println("Message from singleton 3");  
 }  
 private static class SingletonHolder {  
 public static final Singleton3 HOLDER\_INSTANCE = new Singleton3();  
 }  
  
 public static Singleton3 getInstance() {  
 return SingletonHolder.HOLDER\_INSTANCE;  
 }  
}

Листинг 5.3 – код класса Singleton3.

public class Main {  
 public static void main(String[] args) {  
 var s1 = Singleton1.*getInstance*();  
 s1.printMessage();  
 var s2 = Singleton2.*getInstance*();  
 s2.printMessage();  
 var s3 = Singleton3.*getInstance*();  
 s3.printMessage();  
 }  
}

Листинг 5.4 – код класса Main.

## Тестирование.

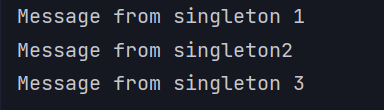


Рисунок 6 – результат тестирования программы.

## Вывод.

Освоен паттерн Singleton.

# Практическая работа №6

## Цель работы.

Изучить понятие наследования, и научиться реализовывать наследование в Java.

## Задание.

Создать абстрактный класс, описывающий собак(Dog). С помощью наследования реализовать различные виды посуды, имеющие свои свойства и методы. Протестировать работу классов.

## Код программы.

package ru.tuganov.task6;

public abstract class Dog {

protected String name;

protected int years;

abstract String getName();

abstract int getYears();

abstract void getWoof();

}

Листинг 6.1 – код абстрактного класса Dog.

package ru.tuganov.task6;

public class Cat extends Dog{

public Cat(String name, int years){

this.name = name;

this.years = years;

}

@Override

String getName() {

return name;

}

@Override

int getYears() {

return years/2;

}

@Override

void getWoof() {

System.out.println("Кошки не гавкают.");

}

public String toString(){

return "name: " + name + "; years: " + years;

}

}

Листинг 6.2 – код класса Cat.

package ru.tuganov.task6;

public class Dolphin extends Dog{

public Dolphin(String name, int years) {

this.name = name;

this.years = years;

}

@Override

String getName() {

return name;

}

@Override

int getYears() {

return years/3;

}

@Override

void getWoof() {

System.out.println("Гав-гав");

}

public String toString(){

return "name: " + name + "; years: " + years;

}

}

Листинг 6.3 – код класса Dolphin.

package ru.tuganov.task6;

public class Taksa extends Dog{

public Taksa(String name, int years){

this.name = name;

this.years = years;

}

@Override

String getName() {

return name;

}

@Override

int getYears() {

return years;

}

@Override

void getWoof() {

System.out.println("Мяу-Мяу");

}

public String toString(){

return "name: " + name + "; years: " + years;

}

}

Листинг 6.4 – код класса Taksa.

package ru.tuganov.task6;

public class Main {

public static void main(String... args){

Taksa taksa = new Taksa("Бобик", 2);

Cat cat = new Cat("Шарик", 6);

Dolphin dolphin = new Dolphin("Перчатка", 6);

System.out.println(taksa + "\n" + cat + "\n" + dolphin);

taksa.getWoof();

cat.getWoof();

dolphin.getWoof();

}

}

Листинг 6.5 – код класса Main.

## Тестирование.

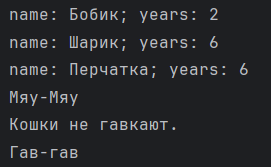


Рисунок 6 – результат тестирования программы.

## Вывод.

Изучены понятия наследования, и изучены реализации наследования в Java.

# Практическая работа №7

## Цель.

Введение в событийное программирование на языке Java.

## Задание.

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Создайте JFrame приложение, у которого есть следующие компоненты GUI:

• одна кнопка JButton labeled “AC Milan”

• другая JButton подписана “Real Madrid”

• надпись JLabel содержит текст “Result: 0 X 0”

• надпись JLabel содержит текст “Last Scorer: N/A”

• надпись Label содержит текст “Winner: DRAW”;

Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

## Код программы.

package ru.tuganov.task7;

import javax.swing.\*;

import java.awt.\*;

public class Main extends JFrame {

int RMil = 0;

int RMad = 0;

JButton Milan = new JButton("AC Milan");

JButton Madrid = new JButton("Real Madrid");

Label Result = new Label("Result:" + RMil + "X" + RMad);

Label LastScorer = new Label("Last Scorer: N/A");

Label Winner = new Label("Winner: DRAW");

public Main(){

super("Task 4");

setLayout(new FlowLayout());

setLocationRelativeTo(null);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

add(Milan);

add(Madrid);

add(Result);

add(LastScorer);

add(Winner);

setSize(500, 100);

Milan.addActionListener(e -> {

RMil ++;

LastScorer.setText("Last Soccer: AC Milan");

Result.setText("Result:" + RMil + "X" + RMad);

update();

});

Madrid.addActionListener(e -> {

RMad ++;

LastScorer.setText("Last Soccer: Real Madrid");

Result.setText("Result:" + RMil + "X" + RMad);

update();

});

}

public void update(){

if (RMad > RMil){

Winner.setText("Winner: Real Madrid");

} else {

Winner.setText("Winner: AC Milan");

}

}

public static void main(String... args){

new Main().setVisible(true);

}

}

Листинг 7.1 – код класса Main.

## Тестирование.

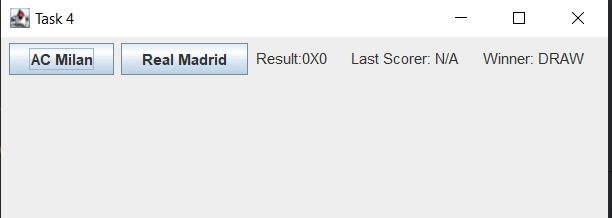


Рисунок 7 – результат тестирование программы.

## Вывод.

Изучено событийное программирование на Java.

# Практическая работа №8

## Цель работы.

Научиться создавать графический интерфейс пользователя, освоить на практике работу с различными объектами для создания ГИП, менеджерами размещения компонентов.

## Задание.

Создать окно, нарисовать в нем 20 случайных фигур, случайного цвета. Классы фигур должны наследоваться от абстрактного класса Shape, в котором описаны свойства фигуры: цвет, позиция.

## Код программы.

package ru.tuganov.task8;

import java.awt.\*;

public abstract class Shape {

int x, y;

Color color;

public Shape(int x, int y, Color color) {

this.x = x;

this.y = y;

this.color = color;

}

public void setColor(Color color){

this.color = color;

}

public void setCoord(int x, int y){

this.x = x;

this.y = y;

}

public int getX(){

return x;

}

public int getY(){

return y;

}

public Color getColor(){

return color;

}

}

Листинг 8.1 – код абстрактного класса Shape.

package ru.tuganov.task8;

import java.awt.\*;

public class Circle extends Shape{

int radius;

public Circle(int x, int y, Color color, int radius) {

super(x, y, color);

this.radius = radius;

}

public void setRadius(int radius){

this.radius = radius;

}

public int getRadius(){

return radius;

}

}

Листинг 8.2 – код класса Circle.

package ru.tuganov.task8;

import java.awt.\*;

import javax.swing.\*;

public class Figure extends JFrame{

Triangle triangle;

Circle circle;

Square square;

public Figure(){

super("Task8");

setSize(500, 500);

setLocationRelativeTo(null);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g){

Graphics2D graph = (Graphics2D) g;

int[] coordsX = new int[3];

int[] coordsY = new int[3];

for (int i = 0; i < 7; i++){

circle = new Circle((int)(Math.random() \* 400), (int)(Math.random() \* 400), new Color((int)(Math.random() \* 0x1000000)), (int)(Math.random() \* 100));

square = new Square((int)(Math.random() \* 400), (int)(Math.random() \* 400), new Color((int)(Math.random() \* 0x1000000)), (int)(Math.random() \* 100));

triangle = new Triangle((int)(Math.random() \* 400), (int)(Math.random() \* 400), new Color((int)(Math.random() \* 0x1000000)), (int)(Math.random() \* 100));

graph.setColor(square.getColor());

graph.fillRect(square.getX(), square.getY(), square.getSide(), square.getSide());

graph.setColor(circle.getColor());

graph.fillOval(circle.getX(), circle.getY(), 2\*circle.getRadius(), 2\*circle.getRadius());

triangleCoords(triangle, coordsX, coordsY);

graph.setColor(triangle.getColor());

graph.fillPolygon(coordsX, coordsY, 3);

}

}

public void triangleCoords(Triangle triangle, int[] coordsX, int[] coordsY){

coordsX[0] = triangle.getX();

coordsY[0] = triangle.getY();

coordsX[1] = triangle.getX() + triangle.getSide();

coordsY[1] = triangle.getY();

coordsX[2] = triangle.getX() + (triangle.getSide() / 2);

coordsY[2] = triangle.getY() - (int)(Math.sqrt(3) \* triangle.getSide() / 2);

}

public static void main(String... args) {

new Figure().setVisible(true);

}

}

Листинг 8.3 – код класса Figure.

package ru.tuganov.task8;

import java.awt.\*;

public class Square extends Shape{

int side;

public Square(int x, int y, Color color, int side) {

super(x, y, color);

this.side = side;

}

public int getSide(){

return side;

}

public void setSide(int side){

this.side = side;

}

}

Листинг 8.4 – код класса Square.

package ru.tuganov.task8;

import java.awt.\*;

public class Triangle extends Shape{

int side;

public Triangle(int x, int y, Color color, int side) {

super(x, y, color);

this.side = side;

}

public int getSide(){

return side;

}

public void setSide(int side){

this.side = side;

}

}

Листинг 8.5 – код класса Triangle.

## Тестирование.

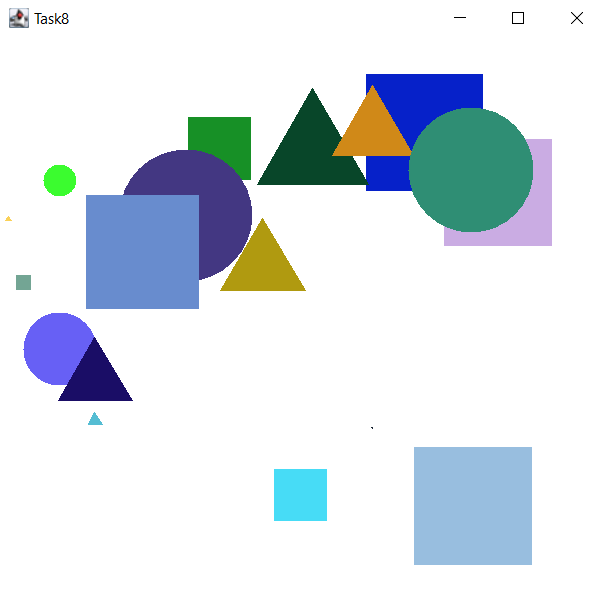


Рисунок 8 – результат тестирования программы.

## Вывод.

Изучено создание графического интерфейса, освоена работа на практике с различными объектами для создания ГИП, менеджерами размещения компонентов.

# Практические работы №9

## Цель работы.

Изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

## Задание.

Реализовать интерфеййс Priceable, имеющий метод getPrice(), возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену

Код программы.

package ru.tuganov.task9;

public interface Priceable {

void setPrice(int price);

int getPrice();

}

Листинг 9.1 – код интерфейса Priceable.

package ru.tuganov.task9;

public class Computer implements Priceable{

int price;

public Computer(){}

@Override

public void setPrice(int price) {

this.price = price;

}

@Override

public int getPrice() {

return price;

}

}

Листинг 9.2 – код класса Computer.

package ru.tuganov.task9;

public class Dog implements Priceable{

int price;

public Dog(){}

@Override

public void setPrice(int price) {

this.price = price;

}

@Override

public int getPrice() {

return price;

}

}

Листинг 9.3 – код класса Dog.

package ru.tuganov.task9;

public class Main {

public static void main(String... args){

Priceable dog = new Dog();

Priceable computer = new Computer();

dog.setPrice(100);

computer.setPrice(200);

System.out.println("Computer price: " + computer.getPrice() + "; Dog price: " + dog.getPrice());

}

}

Листинг 9.4 – код класса Main.

## Тестирование.



Рисунок 9 – результат тестирования программы.

## Вывод.

Изучены понятия интерфейса, получены навыки создания интерфейсов в Java и их применения в программах.

# Практическая работа №10

## Цель работы.

Разработка и программирование рекурсивных алгоритмов на языке Java.

## Задание.

8. Палиндром

Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO. При решении этой задачи нельзя пользоваться циклами, в решениях на питоне нельзя использовать срезы с шагом, отличным от 1.

10. Разворот числа

Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке. При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика. Функция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

11. Количество единиц

Дана последовательность натуральных чисел (одно число в строке), завершающаяся двумя числами 0 подряд. Определите, сколько раз в этой последовательности встречается число 1. Числа, идущие после двух нулей, необходимо игнорировать.

В этой задаче нельзя использовать глобальные переменные и параметры, передаваемые в функцию. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметров.

## Код программы.

package ru.tuganov.task10;

import java.util.Scanner;

public class Main {

public static boolean palindrome(String s) {

if (s.length() == 1) {

return true;

} else {

if (s.substring(0, 1).equals(s.substring(s.length() - 1))) {

if (s.length() == 2) {

return true;

}

return palindrome(s.substring(1, s.length() - 1));

} else {

return false;

}

}

}

public static int revNumbers(int c, int numbers){

if (c == 0) {

return numbers;

}

else {

return revNumbers(c/10, numbers \* 10 + c % 10);

}

}

public static int numbers(int numb, boolean zero){

Scanner scanner = new Scanner(System.in);

int stroka = scanner.nextInt();

if (stroka == 0 && zero) {

return numb;

} else zero = stroka == 0;

if (stroka == 1) {

numb++;

}

return numbers(numb, zero);

}

public static void main(String... args){

Scanner scanner = new Scanner(System.in);

System.out.println("Задание 8. Введите строку: ");

String s = scanner.next();

System.out.println(palindrome(s) ? "YES" : "NO");

System.out.println("Задание 10. Введите число: ");

int c = scanner.nextInt();

System.out.println("Реверс числа: " + revNumbers(c, 0));

System.out.println("Задание 11. Вводите числа: ");

System.out.println("Количество единиц: " + numbers(0, false));

}

}

Листинг 10.1 – код класса Main.

## Тестирование.

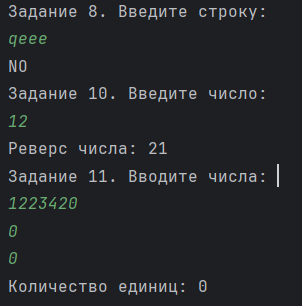


Рисунок 10 – результат тестирование программы.

## Вывод.

Изучена разработка и программирование рекурсивных алгоритмов на языке Java.

# Практическое задание №11

## Цель работы.

Освоение на практике методов сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

## Задание.

Упражнение 1.

Написать тестовый класс, который создает массив класса Student и сортирует массив iDNumber и сортирует его вставками.

Упражнение 2.

Напишите класс SortingStudentsByGPA который реализует интерфейс Comparator таким образом, чтобы сортировать список студентов по их итоговым баллам в порядке убывания с использованием алгоритма быстрой сортировки.

Упражнение 3.

Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках с использованием алгоритма сортировки слиянием.

## Код программы.

package ru.tuganov.task11;

import java.util.Comparator;

public class SortingStudentsByRating implements Comparator {

@Override

public int compare(Object o1, Object o2) {

return ((Student) o1).rating - ((Student) o2).rating;

}

}

Листинг 11.1 – код класса SortingStudentsByRating.

package ru.tuganov.task11;

public class Student {

public String name;

public int rating;

}

Листинг 11.2 – код класса Student.

package ru.tuganov.task11;

public class Main {

public static void main(String... args) {

Student[] arrayOfClass1 = new Student[5];

Student[] arrayOfClass2 = new Student[6];

Student[] arrayOfClass3 = new Student[11];

String[] arrayOfNames = {"Michael", "Nikita", "Maksim", "Makima", "Gennadi", "Lev"};

for (int i = 0; i < 5; i++) {

arrayOfClass1[i] = new Student();

arrayOfClass1[i].name = arrayOfNames[(int) Math.floor(Math.random() \* 10 % 6)];

arrayOfClass1[i].rating = (int) (Math.random() \* 20);

}

for (int i = 0; i < 6; i++) {

arrayOfClass2[i] = new Student();

arrayOfClass2[i].name = arrayOfNames[(int) Math.floor(Math.random() \* 10 % 6)];

arrayOfClass2[i].rating = (int) (Math.random() \* 10);

}

System.out.println("Ex 1. Before: ");

print(arrayOfClass1);

System.out.println("After: ");

print(selectionSort(arrayOfClass1));

System.out.println("\nEx 2. Before: ");

print(arrayOfClass2);

quickSort(arrayOfClass2, 0, arrayOfClass2.length - 1);

System.out.println("Ex 2. After: ");

print(arrayOfClass2);

for (int i = 0; i < 5; i++) {

arrayOfClass1[i] = new Student();

arrayOfClass1[i].name = arrayOfNames[(int) Math.floor(Math.random() \* 10 % 6)];

arrayOfClass1[i].rating = (int) (Math.random() \* 20);

}

for (int i = 0; i < 6; i++) {

arrayOfClass2[i] = new Student();

arrayOfClass2[i].name = arrayOfNames[(int) Math.floor(Math.random() \* 10 % 6)];

arrayOfClass2[i].rating = (int) (Math.random() \* 10);

}

System.arraycopy(arrayOfClass1, 0, arrayOfClass3, 0, arrayOfClass1.length);

System.arraycopy(arrayOfClass2, 0, arrayOfClass3, arrayOfClass1.length, arrayOfClass2.length);

System.out.println("\nEx 3. Before: ");

print(arrayOfClass3);

System.out.println("After: ");

mergeSort(arrayOfClass3, 0, arrayOfClass3.length - 1);

print(arrayOfClass3);

}

public static void print(Student[] list){

for (Student student : list) {

System.out.println("name: " + student.name + "; rating: " + student.rating);

}

}

public static Student[] selectionSort(Student[] list) {

int n = list.length;

for (int i = 1; i < n; i++) {

Student key = list[i];

int j = i - 1;

while (j >= 0 && list[j].rating > key.rating) {

list[j + 1] = list[j];

j = j - 1;

}

list[j + 1] = key;

}

return list;

}

static SortingStudentsByRating comp = new SortingStudentsByRating();

public static void quickSort(Student[] array, int low, int high) {

if (low < high) {

int pivotIndex = low + (high - low) / 2;

Student pivot = array[pivotIndex];

int i = low;

int j = high;

while (i <= j) {

while (comp.compare(array[i], pivot) < 0) {

i++;

}

while (comp.compare(array[j], pivot) > 0) {

j--;

}

if (i <= j) {

Student temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

}

quickSort(array, low, j);

quickSort(array, i, high);

}

}

public static void mergeSort(Student[] array, int low, int high) {

if (low < high) {

int mid = low + (high - low) / 2;

mergeSort(array, low, mid);

mergeSort(array, mid + 1, high);

merge(array, low, mid, high);

}

}

private static void merge(Student[] array, int low, int mid, int high) {

int n1 = mid - low + 1;

int n2 = high - mid;

Student[] left = new Student[n1];

Student[] right = new Student[n2];

System.arraycopy(array, low, left, 0, n1);

for (int j = 0; j < n2; ++j) {

right[j] = array[mid + 1 + j];

}

int i = 0;

int j = 0;

int k = low;

while (i < n1 && j < n2) {

if (comp.compare(left[i], right[j]) <= 0) {

array[k] = left[i];

i++;

} else {

array[k] = right[j];

j++;

}

k++;

}

while (i < n1) {

array[k] = left[i];

i++;

k++;

}

while (j < n2) {

array[k] = right[j];

j++;

k++;

}

}

}

Листинг 11.3 – код класса Main.

## Тестиривание.

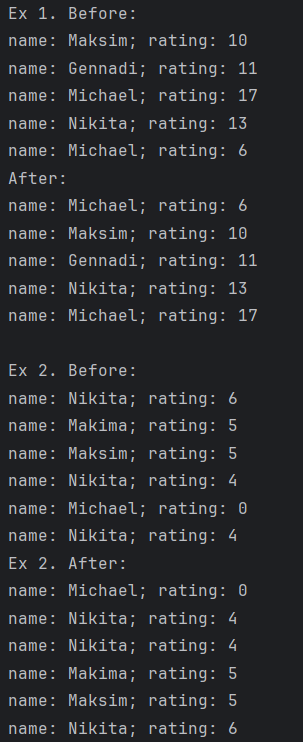


Рисунок 11.1 – результат тестирования программы.

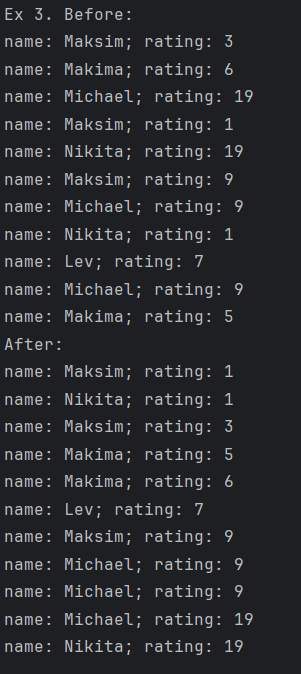


Рисунок 11.2 – результат тестирования программы.

## Вывод.

Освоено на практике методов сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

# Практическая работа №12

## Цель работы.

Изучение на практике приемов работы со стандартными контейнерными классами Java Collection Framework.

## Задание.

Напишите программу в виде консольного приложения, которая моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую; карта «0» побеждает карту «9».

Карточная игра “ В пьяницу”. В этой игре карточная колода раздается поровну двум игрокам. Далее они открывают по одной верхней карте, и тот, чья карта старше, забирает себе обе открытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт, - проигрывает.

Для простоты будем считать, что все карты различны по номиналу и что самая младшая карта побеждает самую старшую карту (“шестерка берет туз”).

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

## Код программы.

package ru.tuganov.task12;

import java.util.\*;

public class Main {

public static void main(String... args){

Scanner scan= new Scanner(System.in);

int[] firstDeck = new int[5];

int[] secondDeck = new int[5];

System.out.println("Введите первую колоду: ");

for (int i = 0; i < 5 ; i++){

firstDeck[i] = scan.nextInt();

}

System.out.println("Введите вторую колоду: ");

for (int i = 0; i < 5 ; i++){

secondDeck[i] = scan.nextInt();

}

deckStack(firstDeck, secondDeck);

deskQueue(firstDeck, secondDeck);

deskDeque(firstDeck, secondDeck);

deskLinkedList(firstDeck, secondDeck);

}

public static void deckStack(int[] firstDeck, int[] secondDeck) {

int steps = 0;

Stack<Integer> firstStack = new Stack<>();

Stack<Integer> firstHold = new Stack<>();

Stack<Integer> secondStack = new Stack<>();

Stack<Integer> secondHold = new Stack<>();

for (int i = 0; i < 5; i++) {

firstStack.add(firstDeck[i]);

secondStack.add(secondDeck[i]);

}

int a, b;

while ((!firstStack.isEmpty() && !secondStack.isEmpty()) || (!firstHold.isEmpty() && !secondHold.isEmpty())){

if (steps >= 106) {

System.out.println("Ботва\n");

return;

}

a = firstStack.pop();

b = secondStack.pop();

if ((a == 0 && b == 9)){

firstHold.add(a);

firstHold.add(b);

} else if (a == 9 && b == 0){

secondHold.add(b);

secondHold.add(a);

} else if (a > b){

firstHold.add(a);

firstHold.add(b);

} else {

secondHold.add(b);

secondHold.add(a);

}

if (firstStack.isEmpty()){

changeStack(firstStack, firstHold);

} else if (secondStack.isEmpty()){

changeStack(secondStack, secondHold);

}

steps ++;

}

if (firstStack.isEmpty()){

System.out.println("second: " + steps);

} else {

System.out.println("first: " + steps);

}

}

public static void changeStack(Stack<Integer> stackInto, Stack<Integer> stackOut){

while (!stackOut.isEmpty()){

stackInto.add(stackOut.pop());

}

}

public static void deskQueue(int[] firstDeck, int[] secondDeck) {

int steps = 0;

Queue<Integer> firstQueue = new ArrayDeque<>();

Queue<Integer> secondQueue = new ArrayDeque<>();

for (int i = 0; i < 5; i++){

firstQueue.add(firstDeck[i]);

secondQueue.add(secondDeck[i]);

}

int a, b;

while (!firstQueue.isEmpty() && !secondQueue.isEmpty()){

if (steps > 106){

System.out.println("Ботва: ");

return;

}

a = firstQueue.remove();

b = secondQueue.remove();

if ((a == 0 && b == 9)){

firstQueue.add(a);

firstQueue.add(b);

} else if (a == 9 && b == 0){

secondQueue.add(b);

secondQueue.add(a);

} else if (a > b){

firstQueue.add(a);

firstQueue.add(b);

} else {

secondQueue.add(b);

secondQueue.add(a);

}

steps++;

}

if (firstQueue.isEmpty()){

System.out.println("second: " + steps);

} else {

System.out.println("first: " + steps);

}

}

public static void deskDeque(int[] firstDeck, int[] secondDeck) {

int steps = 0;

Queue<Integer> firstDeque = new ArrayDeque<>();

Queue<Integer> secondDeque = new ArrayDeque<>();

for (int i = 0; i < 5; i++){

firstDeque.add(firstDeck[i]);

secondDeque.add(secondDeck[i]);

}

int a, b;

while (!firstDeque.isEmpty() && !secondDeque.isEmpty()){

if (steps > 106){

System.out.println("Ботва: ");

return;

}

a = firstDeque.remove();

b = secondDeque.remove();

if ((a == 0 && b == 9)){

firstDeque.add(a);

firstDeque.add(b);

} else if (a == 9 && b == 0){

secondDeque.add(b);

secondDeque.add(a);

} else if (a > b){

firstDeque.add(a);

firstDeque.add(b);

} else {

secondDeque.add(b);

secondDeque.add(a);

}

steps++;

}

if (firstDeque.isEmpty()){

System.out.println("second: " + steps);

} else {

System.out.println("first: " + steps);

}

}

public static void deskLinkedList(int[] firstDeck, int[] secondDeck) {

int steps = 0;

LinkedList<Integer> firstList = new LinkedList<>();

LinkedList<Integer> secondList = new LinkedList<>();

for (int i = 0; i < 5; i++){

firstList.add(firstDeck[i]);

secondList.add(secondDeck[i]);

}

int a, b;

while (!firstList.isEmpty() && !secondList.isEmpty()){

if (steps > 106){

System.out.println("Ботва: ");

return;

}

a = firstList.remove();

b = secondList.remove();

if ((a == 0 && b == 9)){

firstList.add(a);

firstList.add(b);

} else if (a == 9 && b == 0){

secondList.add(b);

secondList.add(a);

} else if (a > b){

firstList.add(a);

firstList.add(b);

} else {

secondList.add(b);

secondList.add(a);

}

steps++;

}

if (firstList.isEmpty()){

System.out.println("second: " + steps);

} else {

System.out.println("first: " + steps);

}

}

}

Листинг 12.1 – код класса Main.

## Тестирование.

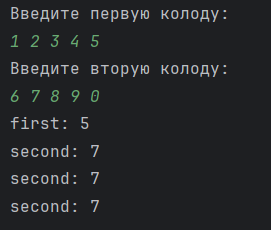


Рисунок 12 – результат тестирования программы.

## Вывод.

Изучены на практике приемы работы со стандартными контейнерными классами Java Collection Framework.

# Практическая работа №13

## Цель работы.

Освоить на практике работу с файлами на языке Java. Получить практические навыки по чтению и записи данных в файл.

## Задание.

Реализовать запись в файл введённой с клавиатуры информации.

## Код программы.

package ru.tuganov.task13;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class Main {

public static void main(String... args){

System.out.println("Введите что-нибудь, а я запишу в файл: ");

try(FileWriter writer = new FileWriter("C:\\Users\\user\\IdeaProjects\\JavaProgramming\\src\\ru\\tuganov\\task13\\newFile.txt", true)){

writer.write((new Scanner(System.in)).nextLine());

} catch (IOException ex){

System.out.println(ex.getMessage());

}

}

}

Листинг 13.1 – код класса Main.

## Тестирование.



Рисунок 13.1 – результат тестирования программы.

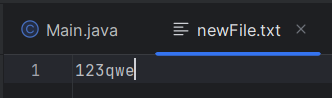


Рисунок 13.2 – результат тестирования программы.

## Вывод.

Освоена на практике работа с файлами на языке Java. Получены практические навыки по чтению и записи данных в файл.

# Практическая работа №14

## Цель работы.

Изучить различные списки ожидания.

## Задание.

* Исследуйте UML диаграмму классов на рисунке 1 и понаблюдайте, как она выражает то, что мы говорили выше в словах. Убедитесь, что вы понимаете все аспекты диаграммы.
* Расширить и модифицировать исходный код WaitList, как необходимо, чтобы полностью реализовать всю схему UML. Включить комментарии Javadoc. Обратите внимание на переключение ролей после реализации каждого интерфейса / класса!
* Изучение работу метода main(), которая использует ваши новые классы и интерфейс.

## Код программы.

package ru.tuganov.task14;

import java.util.Collection;

public interface IWaitList<E> {

void add(E element);

E remove();

boolean contains(E element);

boolean containsAll(Collection c);

boolean isEmpty();

}

Листинг 14.1 – код интерфейса IWaitList.

package ru.tuganov.task14;

import java.util.Collection;

import java.util.concurrent.ConcurrentLinkedDeque;

public class WaitList<E> implements IWaitList<E> {

protected ConcurrentLinkedDeque<E> content = new ConcurrentLinkedDeque<>();

public WaitList(){}

public WaitList(Collection c){

content.addAll(c);

}

@Override

public void add(E element){

content.add(element);

}

@Override

public E remove(){

return content.remove();

}

@Override

public boolean contains(E element){

return content.contains(element);

}

@Override

public boolean containsAll(Collection c){

return content.containsAll(c);

}

@Override

public boolean isEmpty(){

return content.isEmpty();

}

@Override

public String toString() {

return "WaitList{" +

"components=" + content +

'}';

}

}

Листинг 14.2 – код абстрактного класса WaitList.

package ru.tuganov.task14;

import java.util.Collection;

public class BoundedWaitList<E> extends WaitList<E> {

private int capacity;

public BoundedWaitList(int capacity){

this.capacity = capacity;

}

public BoundedWaitList(Collection<E> e){

super(e);

this.capacity = e.size();

}

public int getCapacity(){

return capacity;

}

@Override

public void add(E element){

if (content.size() == capacity){

return;

}

content.add(element);

}

}

Листинг 14.3 – код класса BoundedList.

package ru.tuganov.task14;

import java.util.concurrent.ConcurrentLinkedDeque;

public class UnfairWaitList<E> extends WaitList<E>{

protected ConcurrentLinkedDeque<E> removedContent = new ConcurrentLinkedDeque<>();

public UnfairWaitList(){}

@Override

public void add(E element){

if (removedContent.contains(element)){return;}

content.add(element);

}

public void remove(E element, boolean is\_add) {

boolean removed = false;

for (int i =0; i < content.size(); i++){

E el = content.remove();

if (el == element){

if (removed || is\_add){continue;}

removedContent.add(el);

continue;

}

else {

content.add(el);

}

}

/\*

for(int i = 0; i < components.size(); i++){

E el = components.remove();

if(!removed && el.equals(element)){

removed = true; // i--

}else{

components.add(el);

}

}

if(removed){

components.add(components.remove());

}

\*/

}

public void moveToBack(E element){

remove(element, true);

content.add(element);

}

}

Листинг 14.4 – код класса UnfairList.

package ru.tuganov.task14;

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

ArrayList<String> al = new ArrayList<>();

al.add("Hi2");

al.add("Hi3");

WaitList<String> wl = new WaitList<>();

wl.add("Hi");

wl.add("Hi2");

wl.add("Hi3");

System.out.println(wl);

System.out.println(wl.remove());

System.out.println(wl);

System.out.println(wl.isEmpty());

System.out.println(wl.contains("Hi"));

System.out.println(wl.containsAll(al));

System.out.println();

BoundedWaitList<String> bwl = new BoundedWaitList<>(3);

bwl.add("foo");

bwl.add("foo2");

bwl.add("bar");

System.out.println(bwl);

bwl.add("bar2");

System.out.println();

System.out.println(bwl);

System.out.println(bwl.getCapacity());

System.out.println();

bwl = new BoundedWaitList<String>(al);

System.out.println(bwl.getCapacity());

bwl.add("bar2");

System.out.println();

UnfairWaitList<Integer> uwl = new UnfairWaitList<>();

uwl.add(1);

uwl.add(2);

uwl.add(23);

uwl.add(5);

uwl.add(5);

uwl.add(6);

System.out.println(uwl);

uwl.remove(5, false);

System.out.println(uwl);

uwl.add(5);

System.out.println(uwl);

uwl.moveToBack(6);

System.out.println(uwl);

}

}

Листинг 14.5 – код класса Main.

## Тестирование.

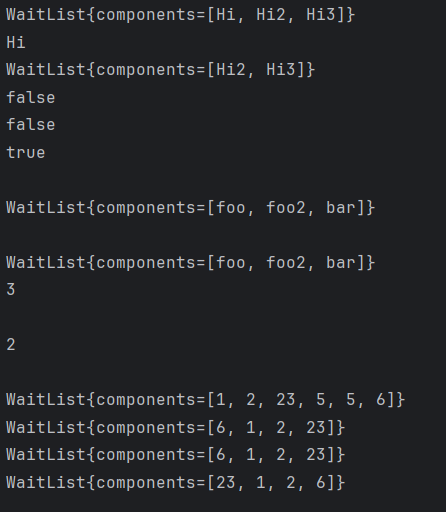


Рисунок 14 – результат тестирования программы.

## Вывод.

Изучены техники написания WaitList’ов.

# Практическая работа №15

## Цель работы.

Введение в разработку программ c с использованием событийного программирования на языке программирования Джава с использованием паттерна MVC.

## Задание.

Напишите реализацию программного кода, с использованием паттерна MVC для расчета заработной платы сотрудника предприятия. Предлагается использовать следующие классы.

• Класс Employee – сотрудник будет выступать в качестве слоя модели.

• Класс EmployeeView будет действовать как слой представления.

• Класс EmployeeContoller будет действовать как уровень контроллера.

## Код программы.

package ru.tuganov.task15;

public class Employee {

String name;

String post;

double taxes;

int salary;

public void setName(String name) {

this.name = name;

}

public void setPost(String post) {

this.post = post;

}

public void setTaxes(double taxes) {

this.taxes = taxes;

}

public void setSalary(int salary) {

this.salary = salary;

}

public String getName() {

return name;

}

public String getPost() {

return post;

}

public int getSalary() {

return salary;

}

public double getTaxes() {

return taxes;

}

}

Листинг 15.1 – код класса Employee.

package ru.tuganov.task15;

public class EmployeeController {

Employee employee;

EmployeeView view;

public EmployeeController(EmployeeView view, Employee employee){

this.employee = employee;

this.view = view;

}

public void updateView(){

view.printEmployeeDetails(employee.getName(), employee.getPost(), employee.getTaxes(), employee.getSalary());

}

public void setName(String name) {

employee.name = name;

}

public void setPost(String post) {

employee.post = post;

}

public void setTaxes(double taxes) {

employee.taxes = taxes;

}

public void setSalary(int salary) {

employee.salary = salary;

}

public String getName() {

return employee.name;

}

public String getPost() {

return employee.post;

}

public int getSalary() {

return employee.salary;

}

public double getTaxes() {

return employee.taxes;

}

}

Листинг 15.2 – код класса EmployeeController.

package ru.tuganov.task15;

public class EmployeeView {

public void printEmployeeDetails(String name, String post, double taxes, int salary){

System.out.println("Паспорт рабочего:");

System.out.println("Имя - " + name);

System.out.println("Должность - " + post);

System.out.println("Зарплата - " + salary);

System.out.println("Зарплата после вычета налогов - " + (salary \* (1 - taxes)));

}

}

Листинг 15.3 – код класса EmployeeView.

package ru.tuganov.task15;

public class MVCEmployee {

public static void main(String... args){

Employee employeeModel = retrieveEmployeeFromDatabase();

EmployeeView employeeView = new EmployeeView();

EmployeeController employeeController = new EmployeeController(employeeView, employeeModel);

employeeController.updateView();

employeeController.setPost("Зав. Кафедры");

System.out.println("\nОбновление!\n");

employeeController.updateView();

}

static Employee retrieveEmployeeFromDatabase(){

Employee employee = new Employee();

employee.setName("Влад");

employee.setPost("Старший преподаватель");

employee.setSalary(2000);

employee.setTaxes(0.02);

return employee;

}

}

Листинг 15.4 – код класса MVCEmployee.

## Тестирование.

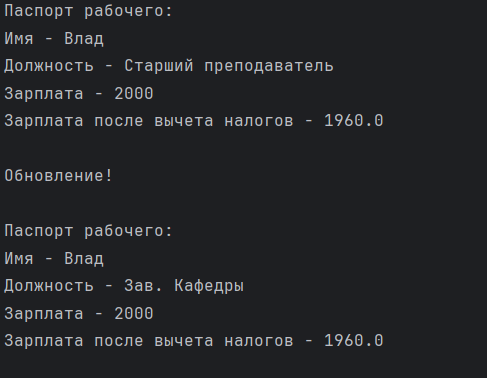


Рисунок 15 – результат тестирования программы.

## Вывод.

Произведено введение в разработку программ c использованием событийного программирования на языке программирования Джава с использованием паттерна MVC.

# Практическая работа №16

## Цель работы.

Получение практических навыков разработки программ, изучение синтаксиса языка Java, освоение основных конструкций языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

## Задание.

Измените код из предыдущего примера следующим образом:

Удалите throws Exception из метода getKey().

Измените метод getKey(), добавив try-catch блок для обработки исключений.

Добавьте цикл к getKey() таким образом, чтобы пользователь получил еще один шанс на ввод значение ключа

Замечания: Инструкция throw очень аналогична инструкции return – она прекращает выполнение метода, дальше мы не идем. Если мы нигде не ставим catch, то у нас выброс Exception очень похож на System.exit() – система завершает процесс. Но мы в любом месте можем поставить catch и, таким образом, предотвратить поломку кода.

## Код программы.

package ru.tuganov.task16;

public class Exception1 {

public void exceptionDemo() {

try{

System.out.println( 2 / 0 );}

catch (ArithmeticException e){System.out.println("Attempt divizion by zero");}

}

public static void main(String[] args) {

new Exception1().exceptionDemo();

}

}

Листинг 16.1 – код класса Exception1.

package ru.tuganov.task16;

import java.util.Scanner;

public class Exception2 {

public static void main(String... args) {

Scanner myScanner = new Scanner( System.in);

System.out.print( "Enter an integer ");

try {

String intString = myScanner.next();

int i = Integer.parseInt(intString);

System.out.println(2 / i);

}

//catch (Exception a){System.out.println("Exception");}

catch(ArithmeticException e){System.out.println("На ноль не дели");}

catch (NumberFormatException e){System.out.println("Введите чиселки");}

finally {

System.out.println("Что-то произошло");

}

}

}

Листинг 16.2 – код класса Exception2.

package ru.tuganov.task16;

public class ThrowsDemo1 {

public void getDetails(String key) {

try{

if(key == null) {

throw new NullPointerException("null key in getDetails" );

}}

catch (NullPointerException e){System.out.println(e.getMessage());}

}

public static void main(String[] args) {

new ThrowsDemo1().getDetails(null);

}

}

Листинг 16.2 – код класса ThrowsDemo1.

package ru.tuganov.task16;

public class ThrowsDemo2 {

public void printMessage(String key) {

String message;

try{

message = getDetails(key);}

catch (NullPointerException e){message = e.getMessage();}

System.out.println( message );

}

public String getDetails(String key) { if(key == null) {

throw new NullPointerException( "null key in getDetails" );

}

return "data for" + key;

}

public static void main(String[] args) {

new ThrowsDemo2().printMessage(null);

}

}

Листинг 16.2 – код класса ThrowsDemo2.

package ru.tuganov.task16;

import java.util.Objects;

import java.util.Scanner;

public class ThrowsDemo3 {

public void getKey() {

Scanner myScanner = new Scanner( System.in );

while (true) {

String key = myScanner.nextLine();

try {

printDetails(key);

return;

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

public void printDetails(String key) throws Exception {

String message = getDetails(key);

System.out.println( message );

}

private String getDetails(String key) throws Exception {

if(Objects.equals(key, "")) {

throw new Exception( "Key set to empty string" );

}

return "data for " + key;

}

public static void main(String[] args) {

new ThrowsDemo3().getKey();

}

}

Листинг 16.2 – код класса ThrowsDemo3.

## Тестирование.



Рисунок 16.1 – результат работы программы.

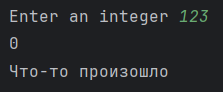


Рисунок 16.2 – результат работы программы.



Рисунок 16.3 – результат работы программы.

## Вывод.

Получены практические навыки разработки программ, изучен синтаксис языка Java, освоены основные конструкции языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

# Практическая работа №17

## Цель работы.

Научиться создавать собственные исключения.

## Задание.

Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

## Код программы.

package ru.tuganov.task17;

public class NotINNException extends Exception{

public NotINNException(String name, int INN){

super("Имя: " + name + "\nИНН: " + INN + "\nНе найден");

}

}

Листинг 17.1 – код класса NotINNException.

package ru.tuganov.task17;

import java.util.Scanner;

public class Main{

public static void main(String... args) {

int[] massINN = new int[20];

System.out.println("Все ИНН: ");

for (int i = 0; i < 20; i++){

massINN[i] = (int)(Math.random() \* 100000);

System.out.println(massINN[i]);

}

Scanner scanner = new Scanner(System.in);

System.out.println("Введите имя и ИНН: ");

String name = scanner.nextLine();

int INN = scanner.nextInt();

try{

notINN(massINN, INN, name);

} catch (NotINNException e){

System.out.println(e.getLocalizedMessage());

return;

}

catch (Exception e) {

System.out.println(e);

}

System.out.println("Имя: " + name + "\nИНН:" + INN + "\nНайден\n");

}

public static void notINN(int[] massINN, int INN, String name) throws NotINNException{

for (int i : massINN) {

if (INN == i) {

return;

}

}

throw new NotINNException(name, INN);

}

}

Листинг 17.2 – код класса Main.

## Тестирование.

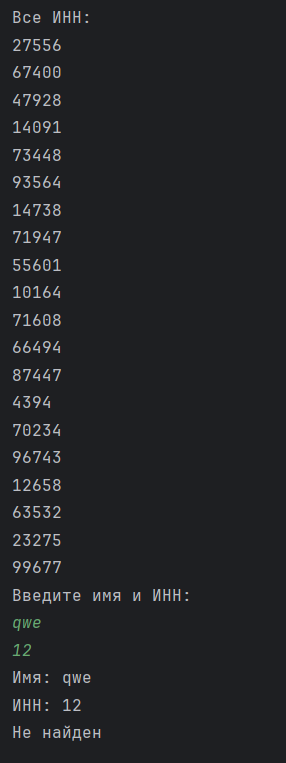


Рисунок 17 – результат работы программы.

## Вывод.

Получен навык создания собственных исключений.

# Практическая работа №18

## Цель работы.

Научиться работать с обобщенными типами в Java и применять их в программах.

## Задание.

Написать класс Matrix, на основе обобщенного типа, реализовать операции с матрицами

## Код программы.

package ru.tuganov.task18;

public class MatrixBox<N> {

private final N[][] matrix;

int rows;

int cols;

public MatrixBox(N[][] matrix){

this.matrix = matrix;

}

public Integer[][] MatrixPlus(N[][] matrixPlus){

rows = matrix.length;

cols = matrix[0].length;

Integer[][] matrixFinal = new Integer[rows][cols];

for (int i = 0; i < rows; i ++){

for (int j = 0; j < cols; j++){

matrixFinal[i][j] = (int)matrixPlus[i][j] + (int)matrix[i][j];

}

}

return matrixFinal;

}

public Integer[][] MatrixMult(N[][] matrixMult){

rows = matrix.length;

cols = matrix[0].length;

int rowsMult = matrixMult.length;

int colsMult = matrixMult[0].length;

Integer[][] matrixFinal = new Integer[cols][rowsMult];

for (int i = 0; i < cols; i ++){

for (int j = 0; j < rowsMult; j ++){

matrixFinal[i][j] = 0;

for (int k = 0; k < rows; k++){

matrixFinal[i][j] += (int)matrix[i][k] \* (int)matrixMult[k][j];

}

}

}

return matrixFinal;

}

public Integer[][] MatrixCoeff(int coeff){

rows = matrix[0].length;

cols = matrix.length;

Integer[][] matrixCoeff = new Integer[rows][cols];

for (int i = 0; i < rows; i ++){

for (int j = 0; j < cols; j++){

matrixCoeff[i][j] = (int)matrix[i][j] \* coeff;

}

}

return matrixCoeff;

}

public void printMatrix(Integer[][] matrix){

rows = matrix.length;

cols = matrix[0].length;

for (int i = 0; i < cols; i ++){

for (int j = 0; j < rows; j++){

System.out.print(matrix[i][j] + " ");

}

System.out.println("\n");

}

}

}

Листинг 18.1 – код класса MatrixBox.

package ru.tuganov.task18;

public class Main {

public static void main(String... args){

Integer[][] matrix = new Integer[4][4];

for (int i = 0; i < 4; i++){

for (int j = 0; j < 4; j++){

matrix[i][j] = (int)(Math.random() \* 10);

}

}

MatrixBox<Integer> matrixBox = new MatrixBox<Integer>(matrix);

Integer[][] matrixTwo = new Integer[4][4];

Integer[][] matrixTest;

for (int i = 0; i < 4; i++){

for (int j = 0; j < 4; j++){

matrixTwo[i][j] = (int)(Math.random() \* 10);

}

}

System.out.println("Первая матрица: \n");

matrixBox.printMatrix(matrix);

System.out.println("Вторая матрица: \n");

matrixBox.printMatrix(matrixTwo);

System.out.println("Сумма матриц: ");

matrixTest = matrixBox.MatrixPlus(matrixTwo);

matrixBox.printMatrix(matrixTest);

System.out.println("Умножение матриц: ");

matrixTest = matrixBox.MatrixMult(matrixTwo);

matrixBox.printMatrix(matrixTest);

System.out.println("Умножение на коэфф. k = 2: ");

matrixTest = matrixBox.MatrixCoeff(2);

matrixBox.printMatrix(matrixTest);

}

}

Листинг 18.2 – код класса Main.

## Тестирование.

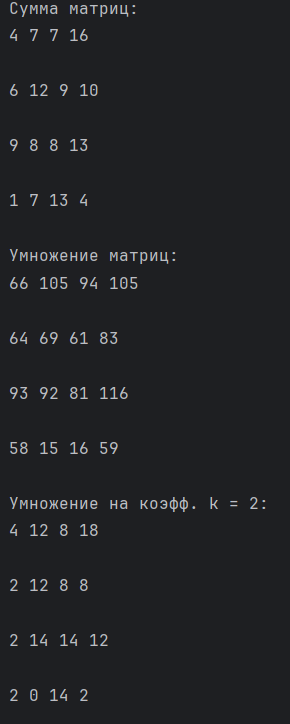


Рисунок 18 – результат работы программы.

## Вывод.

Изучена работа с обобщенными типами в Java и их применение в программах.

# Практическая работа №19

## Цель работы.

Написать метод для конвертации массива строк/чисел в список

## Задание.

Реализовать метод, который возвращает любой элемент массива по индексу.

## Код программы.

package ru.tuganov.task19;

import java.util.ArrayList;

public class Main {

public static <E> ArrayList<E> arrayToList(E[] arr){

ArrayList<E> l = new ArrayList<>(arr.length);

for (E element : arr) {

l.add(element);

}

return l;

}

public static void main(String... args){

String[] a = new String[]{"Привет ", " о дивный ", "мир."};

Integer[] b = new Integer[]{1, 2, 3};

ArrayList<String> l = arrayToList(a);

ArrayList<Integer> m = arrayToList(b);

System.out.println(l);

System.out.println(m);

}

}

Листинг 19.1 – код класса Main.

## Тестирование.

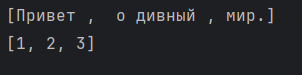


Рисунок 19 – результат работы программы.

## Вывод.

Изучена работа с обобщенными типами в Java и их применением, прием стирания типов в разработке программ на Джаве.

# Практическая работа №20

## Цель работы.

Научиться разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

## Задание.

Напишите программу-калькулятор арифметических выражений записанных в обратной польской нотации (RPN-калькулятор).

## Код программы.

package ru.tuganov.task20;

import java.util.Stack;

public class RPN {

Stack<Double> numbers = new Stack<>();

String postfix;

static String infix;

public RPN(String infix){

this.infix = infix;

postfix = convertToRPN();

}

public double calc(){

String[] tokens = postfix.split(" ");

for (String token : tokens) {

if (isNumeric(token)) {

double num = Double.parseDouble(token);

numbers.push(num);

} else {

double operand2 = numbers.pop();

double operand1 = numbers.pop();

double result = evaluate(token, operand1, operand2);

numbers.push(result);

}

}

return numbers.pop();

}

public static boolean isNumeric(String str) {

try {

Double.parseDouble(str);

return true;

} catch (Exception e) {

return false;

}

}

public static double evaluate(String operator, double operand1, double operand2) {

return switch (operator) {

case "+" -> operand1 + operand2;

case "-" -> operand1 - operand2;

case "\*" -> operand1 \* operand2;

case "/" -> operand1 / operand2;

default -> 0.0;

};

}

public static String convertToRPN() {

StringBuilder postfixExpression = new StringBuilder();

Stack<Character> simbols = new Stack<>();

char[] infixChar = infix.toCharArray();

for (int i = 0; i < infix.length(); i++) {

char c = infixChar[i];

if (Character.isDigit(c)) {

postfixExpression.append(c);

postfixExpression.append(" ");

} else if (c == '(') {

simbols.push(c);

} else if (c == ')') {

while (!simbols.isEmpty() && simbols.peek() != '(') {

postfixExpression.append(simbols.pop());

postfixExpression.append(" ");

}

simbols.pop();

} else if (c == '+' || c == '-' || c == '\*' || c == '/') {

while (!simbols.isEmpty() && priority(simbols.peek()) >= priority(c)) {

postfixExpression.append(simbols.pop());

postfixExpression.append(" ");

}

simbols.push(c);

}

}

while (!simbols.isEmpty()) {

postfixExpression.append(simbols.pop());

postfixExpression.append(" ");

}

return postfixExpression.toString().trim();

}

public static int priority(char operator) {

return switch (operator) {

case '+', '-' -> 1;

case '\*', '/' -> 2;

default -> 0;

};

}

}

Листинг 20.1 – код класса RPN.

package ru.tuganov.task20;

import java.util.Scanner;

public class Main {

public static void main(String... args){

System.out.println("Введите выражение: ");

RPN rpn = new RPN(new Scanner(System.in).nextLine());

System.out.println("Ответ: " + rpn.calc());

}

}

Листинг 20.2 – код класса Main.

## Тестирование.

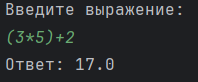


Рисунок 20 – результат работы программы.

## Вывод.

Получены навыки разработки программ с абстрактными типами данных на языке Джава и применения паттерна MVC при разработке программ.

# Практическая работа №21

## Цель работы.

Научиться разрабатывать программы с абстрактными типами данных на языке Джава.

## Задание.

Задание 1. Реализовать очередь на массиве

Найдите инвариант структуры данных «очередь». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.

Реализуйте классы, представляющие циклическую очередь с применением массива.

Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.

Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).

Класс ArrayQueue должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).

Должны быть реализованы следующие функции(процедуры)/методы:

• enqueue – добавить элемент в очередь; • element – первый элемент в очереди;

• dequeue – удалить и вернуть первый элемент в очереди;

• size – текущий размер очереди;

• isEmpty – является ли очередь пустой;

• clear – удалить все элементы из очереди.

Инвариант, пред- и постусловия записываются в исходном коде в виде комментариев.

Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.

Напишите тесты реализованным классам.

Задание 2. Очередь на связанном списке

Определите интерфейс очереди Queue и опишите его контракт.

Реализуйте класс LinkedQueue — очередь на связном списке.

Выделите общие части классов LinkedQueue и ArrayQueue в базовый класс AbstractQueue.

## Код программы.

package ru.tuganov.task21;

public class ArrayQueue {

int size;

int head;

int tail;

Object[] elements = new Object[10];

private void fixCapacity(int capacity)

{

int len = elements.length;

if (capacity > len)

{

Object[] newElements = new Object[elements.length \* 2];

int i = 0;

while (tail != head)

{

newElements[i] = elements[head];

head = (head + 1) % len;

i++;

}

head = 0;

tail = len - 1;

elements = newElements;

}

}

public void enqueue(Object element)

{

fixCapacity(size + 2);

elements[tail] = element;

tail = (tail + 1) % elements.length;

size++;

}

public Object element()

{

return elements[head];

}

public Object dequeue()

{

Object ret = element();

elements[head] = null;

head = (head + 1) % elements.length;

size--;

return ret;

}

public int size()

{

return size;

}

public boolean isEmpty()

{

return (size == 0);

}

public void clear()

{

head = 0;

tail = 0;

size = 0;

}

}

Листинг 21.1 – код интерфейса ArrayQueue.

package ru.tuganov.task21;

public class ArrayQueueADT {

private int size;

private int head;

private int tail;

private Object[] elements = new Object[10];

static void fixCapacity(ArrayQueueADT queue, int capacity)

{

int len = queue.elements.length;

if (capacity > len)

{

Object[] newElements = new Object[queue.elements.length \* 2];

int i = 0;

while (queue.tail!=queue.head)

{

newElements[i] = queue.elements[queue.head];

queue.head = (queue.head + 1) % len;

i++;

}

queue.head = 0;

queue.tail = len - 1;

queue.elements = newElements;

}

}

public void enqueue(ArrayQueueADT queue, Object element)

{

fixCapacity(queue, queue.size + 2);

queue.elements[queue.tail] = element;

queue.tail = (queue.tail + 1) % queue.elements.length;

queue.size++;

}

public Object element(ArrayQueueADT queue)

{

return queue.elements[queue.head];

}

public Object dequeue(ArrayQueueADT queue)

{

Object ret = element(queue);

queue.elements[queue.head] = null;

queue.head = (queue.head + 1) % queue.elements.length;

queue.size--;

return ret;

}

public int size(ArrayQueueADT queue)

{

return queue.size;

}

public boolean isEmpty(ArrayQueueADT queue)

{

return (queue.size == 0);

}

public void clear(ArrayQueueADT queue)

{

queue.head = 0;

queue.tail = 0;

queue.size = 0;

}

}

Листинг 21.2 – код класса ArrayQueueADT.

package ru.tuganov.task21;

public class ArrayQueueModule {

private static int size;

private static int head;

private static int tail;

private static Object[] elements = new Object[10];

static void fixCapacity(int capacity)

{

int len = elements.length;

if (capacity > len)

{

Object[] newElements = new Object[elements.length \* 2];

int i = 0;

while (tail != head)

{

newElements[i] = elements[head];

head = (head + 1) % len;

i++;

}

head = 0;

tail = len - 1;

elements = newElements;

}

}

public void enqueue(Object element)

{

fixCapacity(size + 2);

elements[tail] = element;

tail = (tail + 1) % elements.length;

size++;

}

public Object element()

{

return elements[head];

}

public Object dequeue()

{

Object ret = element();

elements[head] = null;

head = (head + 1) % elements.length;

size--;

return ret;

}

public int size()

{

return size;

}

public boolean isEmpty()

{

return (size == 0);

}

public void clear()

{

head = 0;

tail = 0;

size = 0;

}

}

Листинг 21.3 – код класса ArrayQueueModule.

package ru.tuganov.task21;

public class Main {

public static void main(String... args) {

ArrayQueue arrayQueue = new ArrayQueue();

for (int i = 0; i < 10; i ++){

arrayQueue.enqueue(i);

}

System.out.println("Первый элемент: " + arrayQueue.element());

System.out.println("Удаление первого элемента: " + arrayQueue.dequeue());

System.out.println("Текущий размер: " + arrayQueue.size());

arrayQueue.clear();

System.out.println("Удалил все, проверка на пустоту: " + arrayQueue.isEmpty() + "\n");

ArrayQueueADT arrayQueueADT = new ArrayQueueADT();

for (int i = 10; i > 0; i --){

arrayQueueADT.enqueue(arrayQueueADT, i);

}

System.out.println("Первый элемент: " + arrayQueueADT.element(arrayQueueADT));

System.out.println("Удаление первого элемента: " + arrayQueueADT.dequeue(arrayQueueADT));

System.out.println("Текущий размер: " + arrayQueueADT.size(arrayQueueADT));

arrayQueueADT.clear(arrayQueueADT);

System.out.println("Удалил все, проверка на пустоту: " + arrayQueueADT.isEmpty(arrayQueueADT) + "\n");

ArrayQueueModule arrayQueueModule = new ArrayQueueModule();

for (int i = 20; i > 0; i = i -2){

arrayQueueModule.enqueue(i);

}

System.out.println("Первый элемент: " + arrayQueueModule.element());

System.out.println("Удаление первого элемента: " + arrayQueueModule.dequeue());

System.out.println("Текущий размер: " + arrayQueueModule.size());

arrayQueueModule.clear();

System.out.println("Удалил все, проверка на пустоту: " + arrayQueueModule.isEmpty() + "\n");

}

}

Листинг 21.4 – код класса Main.

## Тестирование.

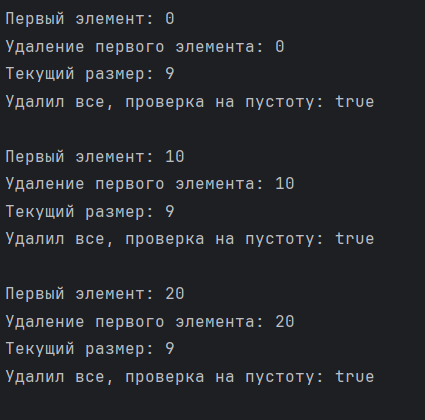


Рисунок 21 – результат работы программы.

## Вывод.

Получены навыки разработки программ с абстрактными типами данных на языке Джава.

# Практическая работа №22

## Цель работы.

Научиться применять порождающие паттерны при разработке программ на Java. В данной практической работе рекомендуется использовать следующие паттерны: Абстрактная фабрика и фабричный метод.

## Задание.

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

## Код программы.

package ru.tuganov.task22;

public abstract interface Chair {

int getHeigh();

String getNameChair();

int getArea();

}

Листинг 22.1 – код интерфейса Chair.

package ru.tuganov.task22;

public class FunctionalChair implements Chair{

int height;

String name;

int area;

public FunctionalChair(int height, String name, int area){

this.height = height;

this.area = area;

this.name = name;

}

@Override

public int getHeigh() {

return height;

}

@Override

public String getNameChair() {

return name;

}

@Override

public int getArea() {

return area;

}

public int sum (int a, int b){return a + b;}

public String toString(){

return "Стул " + this.getNameChair() + " с высотой спинки " + this.getHeigh() + " имеет площадь сидения " + this.getArea();

}

}

Листинг 22.2 – код класса FunctionalChair.

package ru.tuganov.task22;

public class MagicChair implements Chair {

int height;

String name;

int area;

public MagicChair(int height, String name, int area){

this.height = height;

this.area = area;

this.name = name;

}

@Override

public int getHeigh() {

return height;

}

@Override

public String getNameChair() {

return name;

}

@Override

public int getArea() {

return area;

}

public String toString(){

return "Стул " + this.getNameChair() + " с высотой спинки " + this.getHeigh() + " имеет площадь сидения " + this.getArea();

}

}

Листинг 22.3 – код класса MagicChair.

package ru.tuganov.task22;

public class VictorianChair implements Chair{

int height;

String name;

int area;

int age;

public VictorianChair(int height, String name, int area, int age){

this.height = height;

this.area = area;

this.name = name;

this.age = age;

}

public VictorianChair(int age){

this.age = age;

}

public int getAge(){

return age;

}

@Override

public int getHeigh() {

return height;

}

@Override

public String getNameChair() {

return name;

}

@Override

public int getArea() {

return area;

}

@Override

public String toString(){

return "Стул " + this.getNameChair() + " с высотой спинки " + this.getHeigh() + " имеет площадь сидения " + this.getArea() + " и ему (лет) " + this.age;

}

}

Листинг 22.4 – код класса VictorianChair.

package ru.tuganov.task22;

public interface ChairAbstractFactory {

public MagicChair createMagicChair();

public VictorianChair createVictorianChair();

public FunctionalChair createFunctionalChair();

}

Листинг 22.5 – код интерфейса ChairAbstractFactory.

package ru.tuganov.task22;

public class ChairFactory {

public static MagicChair getMagicChair (ChairAbstractFactory factory){

return factory.createMagicChair();

}

public static VictorianChair getVictorianChair (ChairAbstractFactory factory){

return factory.createVictorianChair();

}

public static FunctionalChair getFunctionalChair (ChairAbstractFactory factory){

return factory.createFunctionalChair();

}

}

Листинг 22.6 – код класса ChairFactory.

package ru.tuganov.task22;

public class Client {

static Chair chair;

public static void main(String... args) {

Chair magicChair = new MagicChair(100, "магический", 100);

Chair functionalChair = new FunctionalChair(100, "калькулятор", 100);

Chair victorianChair = new VictorianChair(100, "олдскул", 100, 100);

setChair(magicChair);

sit();

setChair(functionalChair);

sit();

setChair(victorianChair);

sit();

}

public static void sit(){

System.out.println(chair.toString());

}

public static void setChair(Chair chair){

Client.chair = chair;

}

}

Листинг 22.7 – код класса Client.

## Тестирование.

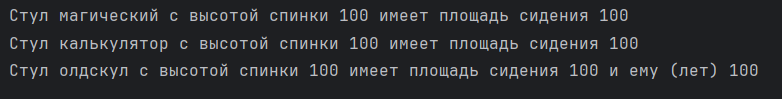


Рисунок 22 – результат работы программы.

## Вывод.

Получены навыки применения порождающих паттернов при разработке программ на Java. Использован фабричный метод.

# Практическая работа №23

## Цель работы.

В процессе написания тестовых заданий ознакомьтесь с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов.

## Задание.

Задание 1

Создайте класс Drink – напитка. Класс описывает сущность – напиток и характеризуется следующими свойствами - стоимостью, названием и описанием. Класс должен быть определен как неизменяемый (Immutable class).

Конструкторы:

• принимающий два параметра – название и описание. Стоимость

при этом инициализируется значением 0;

• принимающий три параметра – стоимость, название и описание.

Методы:

• возвращающий̆стоимость;

• возвращающий̆название;

• возвращающий̆описание.

Дополнительные требования:

Вместо литералов в коде (магических констант) необходимо использовать константы класса, содержащие эти значения. Пояснение: в этом случае вы локализуете изменения этих значений в одном месте, а имя константы скажет нам о сути литерала. Этот класс должен быть неизменяемым (правила проектирования таких классов приводятся в лекциях).

Задание 2

Создайте интерфейс Item для работы с позициями заказа. Интерфейс определяет 3 метода:

− возвращает стоимость;

− возвращает название;

− возвращает описание позиции.

Класс Drink и Dish должны реализовывать этот интерфейс. Класс Dish сделайте неизменяемым (аналогично Drink). Order должен хранить (удалять и добавлять) не только экземпляры класса Dish, но и Drink (Для этого разработайте классы Order и TablesOrderManager).

Задание 3

Создайте класс InternetOrder, который моделирует сущность интернетзаказ в ресторане или кафе. Класс основан на циклическом двусвязном списке с выделенной головой и может хранить как блюда, так и напитки. Внимание: список реализуется самостоятельно.

Конструкторы:

− не принимающий параметров (для списка создается только головной элемент, сам список пуст). − принимающий массив позиций заказа (создаем список из N позиций.)

Методы:

• добавляющий позицию в заказ (принимает ссылку типа Item). Пока этот метод возвращает истину после выполнения операции добавления элемента.

• удаляющий позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Если позиций с заданным названием несколько, всегда удаляются последние. Возвращает логическое значение (true, если элемент был удален).

• удаляющий все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов. возвращающий общее число позиций заказа (повторяющиеся тоже считаются) в заказе.

• Возвращающий массив заказанных блюд и напитков (значений null в массиве быть не должно).

• возвращающий общую стоимость заказа.

• возвращающий число заказанных блюд или напитков (принимает название блюда или напитка в качестве параметра).

• возвращающий массив названий заказанных блюд и напитков (без повторов).

• возвращающий массив позиций заказа, отсортированный по убыванию цены.

## Код программы.

package ru.tuganov.task23;

public class Dish {

int cost;

String name;

String description;

public Dish(String name, String description, int cost){

this.cost = cost;

this.name = name;

this.description = description;

}

public Dish(String name, String description){

this.cost = 0;

this.name = name;

this.description = description;

}

public int getCost() {

return cost;

}

public String getName() {

return name;

}

public String getDescription() {

return description;

}

public void setCost(int cost) {

this.cost = cost;

}

public void setName(String name) {

this.name = name;

}

public void setDescription(String description) {

this.description = description;

}

@Override

public String toString(){

return "Блюдо: " + getName() + "; Описание: " + getDescription() + "; Цена: " + getCost() + "\n";

}

}

Листинг 23.1 – код класса Dish.

package ru.tuganov.task23;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Objects;

public class Manager {

private Node head;

private int size;

public Manager() {

head = new Node(null);

head.setNext(head);

head.setPrevious(head);

size = 0;

}

private static class Node {

int tableNumber;

private Object data;

private Node next;

private Node previous;

public Node(Object data) {

this.data = data;

}

public Object getData() {

return data;

}

public Node getNext() {

return next;

}

public void setNext(Node next) {

this.next = next;

}

public Node getPrevious() {

return previous;

}

public void setPrevious(Node previous) {

this.previous = previous;

}

public void setTableNumber(int tableNumber){this.tableNumber = tableNumber;}

public int getTableNumber(){return this.tableNumber;}

}

public void add(Object element, int tableNumber) {

Node newNode = new Node(element);

newNode.setNext(head.getNext());

newNode.setPrevious(head);

newNode.setTableNumber(tableNumber);

head.getNext().setPrevious(newNode);

head.setNext(newNode);

size++;

}

public int removeALl(String element, int tableNumber) {

int table;

int summ = 0;

Node currentNode = head.getNext();

while (currentNode != head) {

table = currentNode.getTableNumber();

if (table == tableNumber) {

return ((Order)currentNode.getData()).removeALl(element);

}

currentNode = currentNode.getNext();

}

return 0;

}

public boolean remove(String element, int tableNumber) {

int table;

Node currentNode = head.getNext();

while (currentNode != head) {

table = currentNode.getTableNumber();

if (table == tableNumber) {

return ((Order)currentNode.getData()).remove(element);

}

currentNode = currentNode.getNext();

}

return false;

}

public int totalCost(int tableNumber) {

Node currentNode = head.getPrevious();

while (currentNode != head) {

if (currentNode.getTableNumber() == tableNumber){

return ((Order)currentNode.getData()).totalCost();

}

currentNode = currentNode.getNext();

}

return 0;

}

public int totalDishes(String element) {

int summ = 0;

Node currentNode = head.getNext();

while (currentNode != head) {

summ +=((Order)currentNode.getData()).totalDishes(element);

currentNode = currentNode.getNext();

}

return summ;

}

public Dish[] uniqueDishes(int tableNumber) {

Node currentNode = head.getPrevious();

while (currentNode != head) {

if (currentNode.getTableNumber() == tableNumber){

return ((Order)currentNode.getData()).uniqueDishes();

}

}

return null;

}

public Dish[] getSorted(int tableNumber){

Node currentNode = head.getPrevious();

while (currentNode != head) {

if (currentNode.getTableNumber() == tableNumber){

return ((Order)currentNode.getData()).getSorted();

}

}

return null;

}

}

Листинг 23.2 – код класса Manager.

package ru.tuganov.task23;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Objects;

public class Order {

private Node head;

private int size;

public Order() {

head = new Node(null);

head.setNext(head);

head.setPrevious(head);

size = 0;

}

private static class Node {

private Object data;

private Node next;

private Node previous;

public Node(Object data) {

this.data = data;

}

public Object getData() {

return data;

}

public Node getNext() {

return next;

}

public void setNext(Node next) {

this.next = next;

}

public Node getPrevious() {

return previous;

}

public void setPrevious(Node previous) {

this.previous = previous;

}

}

public void add(Object element) {

Node newNode = new Node(element);

newNode.setNext(head.getNext());

newNode.setPrevious(head);

head.getNext().setPrevious(newNode);

head.setNext(newNode);

size++;

}

public int removeALl(String element) {

String dish;

int summ = 0;

Node currentNode = head.getNext();

while (currentNode != head) {

dish = ((Dish)currentNode.getData()).getName();

if (element.equals(dish)) {

currentNode.getPrevious().setNext(currentNode.getNext());

currentNode.getNext().setPrevious(currentNode.getPrevious());

size--;

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

public boolean remove(String element) {

String dish;

Node currentNode = head.getPrevious();

while (currentNode != head) {

dish = ((Dish)currentNode.getData()).getName();

if (element.equals(dish)) {

currentNode.getPrevious().setNext(currentNode.getNext());

currentNode.getNext().setPrevious(currentNode.getPrevious());

size--;

return true;

}

currentNode = currentNode.getPrevious();

}

return false;

}

public int totalCost() {

int summ = 0;

Node currentNode = head.getPrevious();

while (currentNode != head) {

summ += ((Dish)currentNode.getData()).getCost();

currentNode = currentNode.getPrevious();

}

return summ;

}

public int totalDishes(String element) {

String dish;

int summ = 0;

Node currentNode = head.getPrevious();

while (currentNode != head) {

dish = ((Dish)currentNode.getData()).getName();

if (element.equals(dish)) {

summ++;

}

currentNode = currentNode.getPrevious();

}

return summ;

}

boolean unique(Order dishes, String dish){

Node currentNode = dishes.getHead().getNext();

while (currentNode != dishes.getHead()) {

String oneDish;

oneDish = ((Dish)currentNode.data).getName();

if (oneDish == null) break;

if (oneDish.equals(dish)){

return false;

}

currentNode = currentNode.getNext();

}

return true;

}

public Dish[] uniqueDishes() {

Order dishes = new Order();

Node currentNode = head.getNext();

int i = 0;

while (currentNode != head){

String dish = ((Dish)currentNode.getData()).getName();

if (unique(dishes, dish)){

dishes.add(((Dish)currentNode.getData()));

i++;

}

currentNode = currentNode.getNext();

}

Dish[] uniqueDishes = new Dish[i];

currentNode = dishes.getHead().getNext();

for (int j = 0; j < i; j ++){

uniqueDishes[j] = ((Dish)currentNode.getData());

currentNode = currentNode.getNext();

}

return uniqueDishes;

}

public Dish[] getSorted(){

Dish[] dishes = new Dish[size];

int i = 0;

Node currentNode = head.getNext();

while (currentNode != head){

dishes[i] = (Dish)currentNode.getData();

i++;

currentNode = currentNode.getNext();

}

coolSort(dishes);

return dishes;

}

public void coolSort(Dish[] dishes){

for (int i = 0; i < size; i++){

for (int j = i; j < size; j++){

if (dishes[j].getCost() > dishes[i].getCost()){

Dish tmp = dishes[i];

dishes[i] = dishes[j];

dishes[j] = tmp;

}

}

}

}

public int getSize() {

return size;

}

public Node getHead(){

return head;

}

}

Листинг 23.3 – код класса Order.

package ru.tuganov.task23;

import java.util.Arrays;

public class Main {

public static void main(String... args) {

Manager manager = new Manager();

Order order = new Order();

Dish dish = new Dish("Рыба", "крутая", 20);

order.add(dish);

dish = new Dish("Мясо", "крутая", 20);

order.add(dish);

dish = new Dish("Рыба", "ыыы", 40);

order.add(dish);

dish = new Dish("Что-то", "смак", 10);

order.add(dish);

manager.add(order, 1);

System.out.println("Общая стоимость заказа первого столика: " + manager.totalCost(1));

System.out.println("Все блюда первого столика: \n" + Arrays.toString(manager.getSorted(1)));

System.out.println("Все уникальные блюда первого столика: \n" + Arrays.toString(manager.uniqueDishes(1)));

System.out.println("Удалил все Рыба: " + manager.removeALl("Рыба", 1));

System.out.println("Удалил Рыба: " + manager.remove("Рыба", 1));

System.out.println("Все Мясо: " + manager.totalDishes("Мясо"));

}

}

Листинг 23.4 – код интерфейса Main.

## Тестирование.

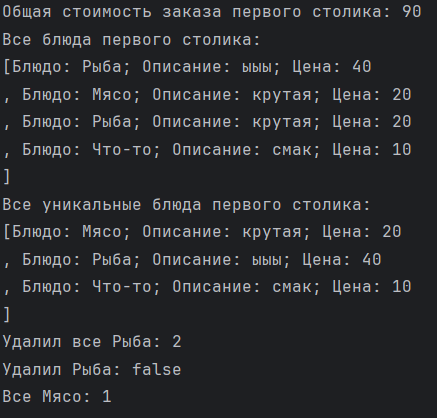


Рисунок 23 – результат работы программы.

## Вывод.

Ознакомились с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов.

# Практическая работа №24

## Цель работы.

Продолжение 23 работы.

## Задание.

Задание 1

Переименуйте класс Order из предыдущего задания в RestaurantOrder.

Создайте интерфейс Order – позиции заказа.

• Интерфейс должен определять следующие методы: добавления позиции в заказ (принимает ссылку типа Item), при этом возвращает логическое значение;

• удаляет позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Возвращает логическое значение;

• удаляет все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов. −возвращает общее число позиций заказа в заказе;

• возвращает массив позиций заказа. −возвращает общую стоимость заказа;

• возвращает число заказанных блюд или напитков (принимает название в качестве параметра). −возвращает массив названий заказанных блюд и напитков (без повторов);

• возвращает массив позиций заказа, отсортированный по убыванию. цены.

Замечание: Классы InternetOrder и RestaurantOrder должны реализовывать интерфейс Order.

Задание 2

Переименуйте класс TablesOrderManager в OrderManager. Добавьте ему еще одно поле типа java.util.HasMap<String, Order>, которое содержит пары адрес-заказ, и методы (работающие с этим полем):

Методы класса

• перегрузка метода добавления заказа. В качестве параметров принимает строку – адрес и ссылку на заказ.

• перегрузка метода получения заказа. В качестве параметра принимает строку – адрес.

• перегрузка метода удаления заказа. В качестве параметра принимает строку – адрес заказа.

• перегрузка метода добавления позиции к заказу. В качестве параметра принимает адрес и Item.

• возвращающий массив имеющихся на данный момент интернет-заказов.

• возвращающий суммарную сумму имеющихся на данный момент интернет-заказов.

• возвращающий общее среди всех интернет-заказов количество заказанных порций заданного блюда по его имени. Принимает имя блюда в качестве параметра.

Методы должны работать с интерфейсными ссылками Order и Item.

Задание 3

Создайте объявляемое исключение OrderAlreadyAddedException, выбрасываемое при попытке добавить заказ столику или по адресу, если со столиком или адресатом уже связан заказ.

Конструктор классов Drink и Dish должен выбрасывать исключение java.lang.IllegalArgumentException при попытке создать блюдо или напиток со стоимостью меньше 0, без имени или описания (если параметры имя и описание - пустые строки).

Создайте не объявляемое исключение IllegalTableNumber, выбрасываемое в методах, принимающих номер столика в качестве параметра, если столика с таким номером не существует.

## Код программы.

package ru.tuganov.task24;

public class Address {

final String cityName;

final int zipCode;

final String streetName;

final int buildingNumber;

final char buildingLetter;

final int apartmentNumber;

//Address EMPTY\_ADDRESS = new Address("", 0, "", 0, ' ', 0);

public Address(String cityName, int zipCode, String streetName, int buildingNumber, char buildingLetter, int apartmentNumber) {

this.cityName = cityName;

this.zipCode = zipCode;

this.streetName = streetName;

this.buildingNumber = buildingNumber;

this.buildingLetter = buildingLetter;

this.apartmentNumber = apartmentNumber;

}

/\*public Address(){

this.cityName = EMPTY\_ADDRESS.cityName;

this.zipCode = EMPTY\_ADDRESS.zipCode;

this.streetName = EMPTY\_ADDRESS.streetName;

this.buildingNumber = EMPTY\_ADDRESS.buildingNumber;

this.buildingLetter = EMPTY\_ADDRESS.buildingLetter;

this.apartmentNumber = EMPTY\_ADDRESS.apartmentNumber;

}\*/

public String getCityName() {

return cityName;

}

public int getZipCode() {

return zipCode;

}

public String getStreetName() {

return streetName;

}

public int getBuildingNumber() {

return buildingNumber;

}

public char getBuildingLetter() {

return buildingLetter;

}

public int getApartmentNumber() {

return apartmentNumber;

}

}

Листинг 24.1 – код класса Address.

package ru.tuganov.task24;

public interface Alcoholable {

boolean isAlcoholicDrink();

double getAlcoholVol();

}

Листинг 24.2 – код интерфейса Alcoholable.

package ru.tuganov.task24;

import java.util.Comparator;

public class CompareItems implements Comparator<MenuItem> {

@Override

public int compare(MenuItem o1, MenuItem o2) {

return Double.compare(o1.getCost(), o2.getCost());

}

}

Листинг 24.3 – код класса CompareItems.

package ru.tuganov.task24;

public class Customer {

final String firstName;

final String secondName;

final int age;

final Address address;

//private final Customer MATURE\_UNKNOWN\_CUSTOMER = new Customer("","", 21,new Address());

//private final Customer NOT\_MATURE\_UNKNOWN\_CUSTOMER = new Customer("","", 0,new Address());

public Customer(String firstName, String secondName, int age, Address address) {

this.firstName = firstName;

this.secondName = secondName;

this.age = age;

this.address = address;

}

/\*public Customer (int age) {

this.age = age;

if (age < 21) {

this.firstName = NOT\_MATURE\_UNKNOWN\_CUSTOMER.firstName;

this.secondName = NOT\_MATURE\_UNKNOWN\_CUSTOMER.secondName;

this.address = NOT\_MATURE\_UNKNOWN\_CUSTOMER.address;

}

else {

this.firstName = MATURE\_UNKNOWN\_CUSTOMER.firstName;

this.secondName = MATURE\_UNKNOWN\_CUSTOMER.secondName;

this.address = MATURE\_UNKNOWN\_CUSTOMER.address;

}

}\*/

public String getFirstName() {

return firstName;

}

public String getSecondName() {

return secondName;

}

public int getAge() {

return age;

}

public Address getAddress() {

return address;

}

}

Листинг 24.4 – код класса Customer.

package ru.tuganov.task24;

public class Dish extends MenuItem{

public Dish(String name, String description, int cost) {

super(cost, name, description);

}

}

Листинг 24.5 – код класса Dish.

package ru.tuganov.task24;

public class Drink extends MenuItem implements Alcoholable{

final double alcoholVol;

final DrinkTypeEnum type;

public Drink(double alcoholVol, DrinkTypeEnum type, int cost, String name, String description) {

super(cost, name, description);

this.alcoholVol = alcoholVol;

this.type = type;

}

@Override

public boolean isAlcoholicDrink() {

return (alcoholVol > 0);

}

@Override

public double getAlcoholVol() {

return alcoholVol;

}

}

Листинг 24.6 – код класса Drink.

package ru.tuganov.task24;

enum DrinkTypeEnum {

BEER,

WINE,

VODKA,

BRANDY,

CHAMPAGNE,

WHISKEY,

TEQUILA,

RUM,

VERMUTH,

LIQUOR,

JAGERMEISTER,

JUICE,

COFEE,

GREEN\_TEA,

BLACK\_TEA,

MILK,

WATER,

SODA

}

Листинг 24.7 – код класса DrinkTypeEnum.

package ru.tuganov.task24;

import java.util.ArrayList;

import java.util.List;

public class InternetOrder implements Order{

private ListNode head;

private int size;

//Customer customer = new Customer(23);

public InternetOrder() {

head = new ListNode(null);

head.setNext(head);

head.setPrevious(head);

size = 0;

}

private static class ListNode {

private Object data;

private ListNode next;

private ListNode previous;

public ListNode(Object data) {

this.data = data;

}

public Object getData() {

return data;

}

public ListNode getNext() {

return next;

}

public void setNext(ListNode next) {

this.next = next;

}

public ListNode getPrevious() {

return previous;

}

public void setPrevious(ListNode previous) {

this.previous = previous;

}

}

@Override

public boolean add(MenuItem item) {

ListNode newNode = new ListNode(item);

newNode.setNext(head.getNext());

newNode.setPrevious(head);

head.getNext().setPrevious(newNode);

head.setNext(newNode);

size++;

return true;

}

@Override

public List<String> itemsNames() {

List<String> names = new ArrayList<>();

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (!names.contains(((MenuItem)currentNode.getData()).getName())){

names.add(((MenuItem)currentNode.getData()).getName());

}

currentNode = currentNode.getNext();

}

return names;

}

@Override

public int itemsQuantity() {

return size;

}

@Override

public int itemsQuantity(String itemName) {

int summ = 0;

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (((MenuItem)currentNode.getData()).getName().equals(itemName)){

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public int itemsQuantity(MenuItem itemName) {

int summ = 0;

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (currentNode.getData() == itemName){

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public List<MenuItem> getItems() {

List<MenuItem> items = new ArrayList<>();

ListNode currentNode = head.getNext();

while (currentNode != head) {

items.add((MenuItem)currentNode.getData());

currentNode = currentNode.getNext();

}

return items;

}

@Override

public boolean remove(String itemName) {

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (((MenuItem)currentNode.getData()).getName().equals(itemName)){

currentNode.getPrevious().setNext(currentNode.getNext());

currentNode.getNext().setPrevious(currentNode.getPrevious());

size--;

return true;

}

currentNode = currentNode.getNext();

}

return false;

}

@Override

public boolean remove(MenuItem item) {

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (currentNode.getData() == item){

currentNode.getPrevious().setNext(currentNode.getNext());

currentNode.getNext().setPrevious(currentNode.getPrevious());

size--;

return true;

}

currentNode = currentNode.getNext();

}

return false;

}

@Override

public int removeALl(String itemName) {

int summ = 0;

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (((MenuItem)currentNode.getData()).getName().equals(itemName)){

currentNode.getPrevious().setNext(currentNode.getNext());

currentNode.getNext().setPrevious(currentNode.getPrevious());

size--;

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public int removeALl(MenuItem item) {

int summ = 0;

ListNode currentNode = head.getNext();

while (currentNode != head) {

if (currentNode.getData() == item){

currentNode.getPrevious().setNext(currentNode.getNext());

currentNode.getNext().setPrevious(currentNode.getPrevious());

size--;

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public List<MenuItem> sortedItemsByCostDesc() {

List<MenuItem> sorted = new ArrayList<>();

ListNode currentNode = head.getNext();

while (currentNode != head) {

sorted.add((MenuItem)currentNode.getData());

currentNode = currentNode.getNext();

}

sorted.sort(new CompareItems());

return sorted;

}

@Override

public int costTotal() {

int summ = 0;

ListNode currentNode = head.getNext();

while (currentNode != head) {

summ += ((MenuItem)currentNode.getData()).getCost();

currentNode = currentNode.getNext();

}

return summ;

}

/\*@Override

public Customer getCustomer() {

return customer;

}

@Override

public void setCustomer(Customer customer) {

this.customer = customer;

}\*/

}

Листинг 24.8 – код классов InternetOrder и ListNode.

package ru.tuganov.task24;

import java.util.ArrayList;

import java.util.List;

public class InternetOrderManager implements OrdersManager{

private QueueNode head;

private int size;

//Customer customer = new Customer(23);

public InternetOrderManager() {

head = new QueueNode(null);

head.setNext(head);

head.setPrevious(head);

size = 0;

}

private static class QueueNode {

private Object data;

private QueueNode next;

private QueueNode previous;

public QueueNode(Object data) {

this.data = data;

}

public Object getData() {

return data;

}

public QueueNode getNext() {

return next;

}

public void setNext(QueueNode next) {

this.next = next;

}

public QueueNode getPrevious() {

return previous;

}

public void setPrevious(QueueNode previous) {

this.previous = previous;

}

}

@Override

public List<Order> getOrders() {

List<Order> orders = new ArrayList<>();

QueueNode currentNode = head.getNext();

while (currentNode != head) {

orders.add((Order)currentNode.getData());

currentNode = currentNode.getNext();

}

return orders;

}

@Override

public int itemsQuantity(String itemName) {

int summ = 0;

QueueNode currentNode = head.getNext();

while (currentNode != head) {

if (((MenuItem)currentNode.getData()).getName().equals(itemName)){

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public int itemsQuantity(MenuItem item) {

int summ = 0;

QueueNode currentNode = head.getNext();

while (currentNode != head) {

if (currentNode.getData() == item){

summ++;

}

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public int ordersCostSummary() {

int summ = 0;

QueueNode currentNode = head.getNext();

while (currentNode != head) {

summ += ((MenuItem)currentNode.getData()).getCost();

currentNode = currentNode.getNext();

}

return summ;

}

@Override

public int ordersQuantity() {

int summ = 0;

QueueNode currentNode = head.getNext();

while (currentNode != head) {

summ += ((Order)currentNode.getData()).itemsQuantity();

currentNode = currentNode.getNext();

}

return summ;

}

}

Листинг 24.9 – код классов InternetOrderManager и QueueNode.

package ru.tuganov.task24;

public class Main {

public static void main(String... args) {

TableOrdersManager tableOrdersManager = new TableOrdersManager();

TableOrder tableOrder = new TableOrder();

MenuItem menuItem = new MenuItem(20, "рыба", "крутая");

tableOrder.add(menuItem);

MenuItem menuItem1 = new MenuItem(30, "рыба", "еще круче");

tableOrder.add(menuItem1);

MenuItem menuItem2 = new MenuItem(10, "мясо", "вкусное");

tableOrder.add(menuItem2);

MenuItem menuItem3 = new MenuItem(40, "котлета", "крутая");

tableOrder.add(menuItem3);

tableOrder.add(menuItem);

tableOrdersManager.add(tableOrder, 1);

System.out.println("Общая стоимость: " + tableOrdersManager.ordersCostSummary());

System.out.println("Сколько рыбы: " + tableOrdersManager.itemsQuantity("рыба"));

System.out.println("Свободные столики: " + tableOrdersManager.freeTableNumbers());

tableOrdersManager.removeAll(tableOrder);

System.out.println("Общая стоимость: " + tableOrdersManager.ordersCostSummary());

}

}

Листинг 24.10 – код класса Main.

package ru.tuganov.task24;

public class MenuItem {

protected int cost;

protected String name;

protected String description;

public MenuItem(int cost, String name, String description) {

this.cost = cost;

this.name = name;

this.description = description;

}

public int getCost() {

return cost;

}

public String getName() {

return name;

}

public String getDescription() {

return description;

}

}

Листинг 24.11 – код класса MenuItem.

package ru.tuganov.task24;

import java.util.List;

public interface Order {

boolean add(MenuItem item);

List<String> itemsNames();

int itemsQuantity();

int itemsQuantity(String itemName);

int itemsQuantity(MenuItem itemName);

List <MenuItem> getItems();

boolean remove(String itemName);

boolean remove(MenuItem item);

int removeALl(String itemName);

int removeALl(MenuItem item);

List <MenuItem> sortedItemsByCostDesc();

int costTotal();

/\*Customer getCustomer();

void setCustomer(Customer customer);\*/

}

Листинг 24.12 – код интерфейса Order.

package ru.tuganov.task24;

import java.util.List;

public interface OrdersManager {

List<Order> getOrders();

int itemsQuantity(String itemName);

int itemsQuantity(MenuItem item);

int ordersCostSummary();

int ordersQuantity();

}

Листинг 24.13 – код интерфейса OrdersManager.

package ru.tuganov.task24;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

public class TableOrder implements Order{

int size = 0;

List<MenuItem> items = new ArrayList<>();

//Customer customer = new Customer(0);

@Override

public boolean add(MenuItem item) {

for (MenuItem i: items){

if (i == item){

return false;

}

}

items.add(item);

size++;

return true;

}

@Override

public List<String> itemsNames() {

List <String> names = new ArrayList<>();

for (MenuItem i: items){

if (!names.contains(i.getName())){

names.add(i.getName());

}

}

return names;

}

@Override

public int itemsQuantity() {

return size;

}

@Override

public int itemsQuantity(String itemName) {

int summ = 0;

for (MenuItem i: items){

if (Objects.equals(i.getName(), itemName)){

summ++;

}

}

return summ;

}

@Override

public int itemsQuantity(MenuItem itemName) {

int summ = 0;

for (MenuItem i: items){

if (i == itemName){

summ++;

}

}

return summ;

}

@Override

public List<MenuItem> getItems() {

return items;

}

@Override

public boolean remove(String itemName) {

for (MenuItem i: items){

if (Objects.equals(i.getName(), itemName)){

items.remove(i);

size--;

return true;

}

}

return false;

}

@Override

public boolean remove(MenuItem item) {

for (MenuItem i: items){

if (i == item){

items.remove(i);

size--;

return true;

}

}

return false;

}

@Override

public int removeALl(String itemName) {

int summ = 0;

for (MenuItem i: items){

if (Objects.equals(i.getName(), itemName)){

items.remove(i);

size--;

summ++;

}

}

return summ;

}

@Override

public int removeALl(MenuItem item) {

int summ = 0;

for (MenuItem i: items){

if (i == item){

items.remove(i);

size--;

summ++;

}

}

return summ;

}

@Override

public List<MenuItem> sortedItemsByCostDesc() {

items.sort(new CompareItems());

return items;

}

@Override

public int costTotal() {

int summ = 0;

for (MenuItem i: items){

summ += i.getCost();

}

return summ;

}

/\*@Override

public Customer getCustomer() {

return customer;

}

@Override

public void setCustomer(Customer customer) {

this.customer = customer;

}\*/

}

Листинг 24.14 – код класса TablerOrder.

package ru.tuganov.task24;

import java.util.ArrayList;

import java.util.List;

public class TableOrdersManager implements OrdersManager{

int free = 20;

Order[] orders = new Order[20];

public void add(Order order, int tableNumber){

orders[tableNumber - 1] = order;

free--;

}

public void addItem(MenuItem item, int tableNumber){

orders[tableNumber - 1].add(item);

}

public int freeTableNumber(){

return free;

}

public List<Integer> freeTableNumbers(){

List<Integer> freeTables = new ArrayList<>();

for (int i = 0; i < 20; i++){

if (orders[i] == null){

freeTables.add(i + 1);

}

}

return freeTables;

}

public Order getOrder(int tableNumber){

return orders[tableNumber - 1];

}

public void remove(int tableNumber){

orders[tableNumber - 1] = null;

}

public int remove(Order order){

for (int i = 0; i < 20; i++){

if (orders[i] == order){

orders[i] = null;

return i + 1;

}

}

return 0;

}

public int removeAll(Order order){

int summ = 0;

for (int i = 0; i < 20; i++){

if (orders[i] == order){

orders[i] = null;

summ++;

}

}

return summ;

}

@Override

public List<Order> getOrders() {

return new ArrayList<>(List.of(orders));

}

@Override

public int itemsQuantity(String itemName) {

int summ = 0;

for (Order i: orders){

if (i != null){

for (MenuItem j: i.getItems()) {

if (j.getName().equals(itemName)) {

summ++;

}

}

}

}

return summ;

}

@Override

public int itemsQuantity(MenuItem item) {

int summ = 0;

for (Order i: orders){

for (MenuItem j: i.getItems()){

if (j == item){

summ++;

}

}

}

return summ;

}

@Override

public int ordersCostSummary() {

int summ = 0;

for (Order i: orders){

if (i != null) {

for (MenuItem j : i.getItems()) {

summ += j.getCost();

}

}

}

return summ;

}

@Override

public int ordersQuantity() {

return orders.length;

}

}

Листинг 24.15 – код класса TableOrdersManager.

## Тестирование.

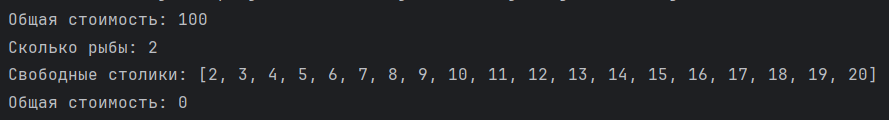


Рисунок 24 – результат работы программы.

## Вывод.

Ознакомились с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов.

# Список литературы

1. Зорина Н.В. Курс лекций по Объектно-ориентированному программированию на Java, МИРЭА, Москва, 2016 61

2. Программирование на языке Java: работа со строками и массивами. Методические указания. [Электронный ресурс] : Учебно-методические пособия — Электрон. дан. — СПб. : ПГУПС, 2015. — 24 с.

3. Кожомбердиева, Г.И. Программирование на языке Java: создание графического интерфейса пользователя: учеб. пособие. [Электронный ресурс] : Учебные пособия / Г.И. Кожомбердиева, М.И. Гарина. — Электрон. дан. — СПб.: ПГУПС, 2012. — 67 с.

4. Вишневская, Т.И. Технология программирования. Часть 1. [Электронный ресурс] / Т.И. Вишневская, Т.Н. Романова. — Электрон. дан. — М. : МГТУ им. Н.Э. Баумана, 2007. — 59 с.