|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ «ПОЛТАВСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ  НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ  «ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ» | | | | | | |
| Циклова комісія дисциплін програмної інженерії | | | | | | |
| **ЗВІТ** з навчальної практики | | | | | | |
| «Практика з об’єктно-орієнтованого програмування» | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | Виконав: здобувач освіти 3 курсу, | | | | | |
|  | групи | 35 | |  | | |
|  | напрям підготовки: 121 «Інженерія | | | | | |
|  | програмного забезпечення» | | | | | |
|  | спеціалізації «Розробка програмного | | | | | |
|  | забезпечення» | | | | | |
|  | Кандрін Е.В. | | | | | |
|  | (прізвище та ініціали) | | | | | |
|  | Керівник | |  | |  | В.В. Оліник |
|  |  | | (підпис) | |  | (прізвище та ініціали) |
|  | | | | | | |
| Полтава – 2022 | | | | | | |

**ЗМІСТ**

[ВСТУП 3](#_Toc66647158)

[1. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №1 4](#_Toc66647159)

[1.1. Постановка завдання 4](#_Toc66647160)

[1.2. Опис використаних методів програмування 4](#_Toc66647161)

[1.3. Текст програми з коментарями 4](#_Toc66647162)

[1.4. Результати роботи програми 5](#_Toc66647163)

[2. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №2 6](#_Toc66647164)

[2.1. Постановка завдання 6](#_Toc66647165)

[2.2. Текст програми з коментарями 6](#_Toc66647166)

[2.3. Результати роботи програми 8](#_Toc66647167)

[3. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №3 9](#_Toc66647168)

[3.1. Постановка завдання 9](#_Toc66647169)

[3.2. Текст програми з коментарями 9](#_Toc66647170)

[3.3. Результати роботи програми 11](#_Toc66647171)

[4. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №4 11](#_Toc66647172)

[4.1. Постановка завдання 12](#_Toc66647173)

[4.2. Текст програми з коментарями 12](#_Toc66647174)

[4.3. Результати роботи програми 14](#_Toc66647175)

[5. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №5 15](#_Toc66647176)

[5.1. Постановка завдання 15](#_Toc66647177)

[5.2. Текст програми з коментарями 15](#_Toc66647178)

[5.3. Результати роботи програми 16](#_Toc66647179)

[ВИСНОВКИ 17](#_Toc66647180)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 18](#_Toc66647181)

**ВСТУП**

Метою практики є формування професійного вміння прийняття самостійних рішень на певних ділянках роботи (або з конкретних питань) в реальних виробничих умовах шляхом виконання окремих функцій і завдань, властивих майбутній професії.

Практика з ООП є невід’ємною складовою частиною процесу підготовки молодших спеціалістів за напрямом «Розробка програмного забезпечення» і проводиться на базі навчального закладу з використанням командної роботи і проектного підходу. Впродовж практики здобувачі освіти виконують завдання для закріплення навичок об’єктно-орієнтованого програмування.

Завданнями практики з «Об’єктно-орієнтованого програмування» є:

* Створити програму на мові Java для визначення кількості чергувань 0 и 1 у двійковому представленні заданого десяткового числа.
* Розробити клас, що серіалізується, для зберігання параметрів і результатів обчислень.
* Розробити клас для демонстрації в діалоговому режимі збереження та відновлення стану об'єкта, використовуючи серіалізацію.
* Забезпечити розміщення результатів обчислень уколекції з можливістю збереження/відновлення.
* Використовуючи шаблон проектування Factory Method (Virtual Constructor), розробити ієрархію, що передбачає розширення рахунок додавання нових відображуваних класів.
* Розширити ієрархію інтерфейсом "фабрикованих" об'єктів, що представляє набір методів для відображення результатів обчислень.
* Розширити ієрархію похідними класами, реалізують методи для подання результатів у вигляді текстової таблиці.
* Розширити ієрархію інтерфейсом "фабрикованих" об'єктів, що представляє набір методів для відображення результатів обчислень.

# 1. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №1

## 1.1. Постановка завдання

* Створити програму на мові Java для визначення кількості чергувань 0 и 1 у двійковому представленні заданого десяткового числа.
* Розробити клас, що серіалізується, для зберігання параметрів і результатів обчислень.
* Розробити клас для демонстрації в діалоговому режимі збереження та відновлення стану об'єкта, використовуючи серіалізацію.

## 1.2. Опис використаних методів програмування

В ході розробки було використано наступні методи програмування:

* Об’єктно Орієнтоване програмування
* Візуальне проектування графічного інтерфейсу

## 1.3. Текст програми з коментарями

import progect.Calc;

import java.io.IOException;

import java.io.BufferedReader;

import java.io.InputStreamReader;

/\*\* Вычисление и отображение результатов.

\* Содержит реализацию статического метода main().

\* @author Егор

\* @see Main#main

\*/

public class Main {

/\*\* Объект класса {@linkplain Calc}.<br>Решает задачу инд. задания. \*/

private final Calc calc = new Calc();

/\*\* Отображает меню. \*/

private void menu() {

String s = null;

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

do {

do {

System.out.println("Enter command...");

System.out.print("'e'xit, 'v'iew, 'g'enerate, 's'ave, 'r'estore: ");

try {

s = in.readLine();

} catch(IOException e) {

System.out.println("Error: " + e);

System.exit(0);

}

} while (s.length() != 1);

switch (s.charAt(0)) {

case 'e' -> System.out.println("Exit.");

case 'v' -> {

System.out.println("View current.");

calc.show();

}

case 'g' -> {

System.out.println("Random generation.");

calc.init((int)(0 + Math.random()\*1000));

calc.show();

}

case 's' -> {

System.out.println("Save current.");

try {

calc.save();

} catch (IOException e) {

System.out.println("Serialization error: " + e);

}

calc.show();

}

case 'r' -> {

System.out.println("Restore last saved.");

try {

calc.restore();

} catch (Exception e) {

System.out.println("Serialization error: " + e);

}

calc.show();

}

default -> System.out.print("Wrong command. ");

}

} while(s.charAt(0) != 'e');

}

/\*\* Выполняется при запуске программы.

\* Вычисляется значение количество чередований 0 и 1 в двоичном представлении

\* заданного десятичного числа.

\* Результаты вычислений выводятся на экран.

\* @param args - параметры запуска программы.

\*/

public static void main(String[] args) {

Main main = new Main();

main.menu();

}

}

Рисунок 1.1 – Опис головної форми.

private void menu() {

String s = null;

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

do {

do {

System.out.println("Enter command...");

System.out.print("'e'xit, 'v'iew, 'g'enerate, 's'ave, 'r'estore: ");

try {

s = in.readLine();

} catch(IOException e) {

System.out.println("Error: " + e);

System.exit(0);

}

} while (s.length() != 1);

switch (s.charAt(0)) {

case 'e' -> System.out.println("Exit.");

case 'v' -> {

System.out.println("View current.");

calc.show();

}

case 'g' -> {

System.out.println("Random generation.");

calc.init((int)(0 + Math.random()\*1000));

calc.show();

}

case 's' -> {

System.out.println("Save current.");

try {

calc.save();

} catch (IOException e) {

System.out.println("Serialization error: " + e);

}

calc.show();

}

case 'r' -> {

System.out.println("Restore last saved.");

try {

calc.restore();

} catch (Exception e) {

System.out.println("Serialization error: " + e);

}

calc.show();

}

default -> System.out.print("Wrong command. ");

}

} while(s.charAt(0) != 'e');

}

Рисунок 1.2 – Опис функціоналу кнопок на головній формі

package progect;

import java.io.Serializable;

/\*\* Хранит исходные данные и результат вычислений.

\* @author Егор

\*

\*/

public class Item2d implements Serializable {

/\*\* Вычисляемое значение. \*/

// transient

private int x;

/\*\* Результат вычисления. \*/

private int y;

/\*\* Автоматически сгенерированная константа \*/

private static final long serialVersionUID = 1L;

/\*\* Инициализирует поля {@linkplain Item2d#x}, {@linkplain Item2d#y} \*/

public Item2d() {

}

/\*\* Устанавливает значения полей: десятичного числа

\* и результата вычисления .

\* @param x - значение для инициализации поля {@linkplain Item2d#x}

\* @param y - значение для инициализации поля {@linkplain Item2d#y}

\*/

public Item2d(int x, int y) {

this.x = x;

this.y = y;

}

/\*\* Установка значения поля {@linkplain Item2d#x}

\* @param x - значение для {@linkplain Item2d#x}

\* @return Значение {@linkplain Item2d#x}

\*/

public double setX(int x) {

return this.x = x;

}

/\*\* Получение значения поля {@linkplain Item2d#x}

\* @return Значение {@linkplain Item2d#x}

\*/

public int getX() {

return x;

}

/\*\* Установка значения поля {@linkplain Item2d#y}

\* @param y - значение для {@linkplain Item2d#y}

\* @return Значение {@linkplain Item2d#y}

\*/

public int setY(int y) {

return this.y = y;

}

/\*\* Получение значения поля {@linkplain Item2d#y}

\* @return значение {@linkplain Item2d#y}

\*/

public int getY() {

return y;

}

/\*\* Установка значений {@linkplain Item2d#x} и {@linkplain Item2d#y}

\* @param x - значение для {@linkplain Item2d#x}

\* @param y - значение для {@linkplain Item2d#y}

\* @return this

\*/

public Item2d setXY(int x, int y) {

this.x = x;

this.y = y;

return this;

}

/\*\* Представляет результат вычислений в виде строки.<br>{@inheritDoc} \*/

@Override

public String toString() {

return "Cгенерируемое десятичное число = " + x + "\n Количество чередований 0 и 1 в двоичном представлении\n" +

"заданного десятичного числа. = " + y;

}

}

Рисунок 1.3 – Опис класу для зберігання даних

package progect;

import java.io.IOException;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

/\*\*

\* Содержит реализацию методов для вычисления и отображения результатов.

\* @author Егор

\*/

public class Calc {

/\*\* Имя файла, используемое при сериализации. \*/

private static final String FNAME = "Item2d.bin";

/\*\* Сохраняет результат вычислений. Объект класса {@linkplain Item2d} \*/

private Item2d result;

/\*\* Инициализирует {@linkplain Calc#result} \*/

public Calc() {

result = new Item2d();

}

/\*\* Установить значение {@linkplain Calc#result}

\* @param result - новое значение ссылки на объект {@linkplain Item2d}

\*/

public void setResult(Item2d result) {

this.result = result;

}

/\*\* Получить значение {@linkplain Calc#result}

\* @return текущее значение ссылки на объект {@linkplain Item2d}

\*/

public Item2d getResult() {

return result;

}

/\*\* Вычисляет значение чередований.

\* @param x - десятичное число.

\* @return результат вычисления.

\*/

private int calc(int x) {

String BinaryS = Integer.toBinaryString((int) x);

int binary = Integer.parseInt (Integer.toBinaryString((int) x));

int y =0;

System.out.println("Binary = "+binary);

int buff = binary%10;

binary = binary/10;

for(int i=0;i<BinaryS.length()-2;i++) {

if(buff != binary%10) {

y++;

}

buff = binary%10;

binary = binary/10;

}

return y;

}

/\*\* Вычисляет значение чередований и сохраняет

\* результат в объекте {@linkplain Calc#result}

\* @param x - десятичное число.

\* @return

\*/

public int init(int x ) {

result.setX(x);

return result.setY(calc(x));

}

/\*\* Выводит результат вычислений. \*/

public void show() {

System.out.println(result);

}

/\*\* Сохраняет {@linkplain Calc#result} в файле {@linkplain Calc#FNAME}

\* @throws IOException

\*/

public void save() throws IOException {

ObjectOutputStream os = new ObjectOutputStream(new

FileOutputStream(FNAME));

os.writeObject(result);

os.flush();

os.close();

}

/\*\* Восстанавливает {@linkplain Calc#result} из файла {@linkplain Calc#FNAME}

\* @throws Exception

\*/

public void restore() throws Exception {

ObjectInputStream is = new ObjectInputStream(new FileInputStream(FNAME));

result = (Item2d)is.readObject();

is.close();

}

}

Рисунок 1.4 – Опис класу обчислення

## 1.4. Результати роботи програми

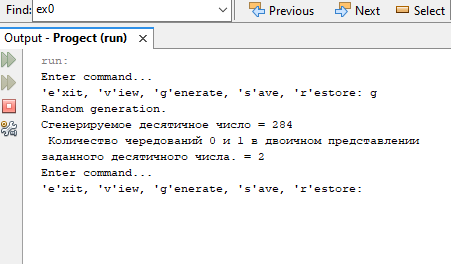


Рисунок 1.5 – Виведення на екран головної форми.

# 2. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №2

## 2.1. Постановка завдання

* Забезпечити розміщення результатів обчислень у колекції з можливістю збереження/відновлення.
* Використовуючи шаблон проектування Factory Method (Virtual Constructor), розробити ієрархію, що передбачає розширення рахунок додавання нових відображуваних класів.
* Розширити ієрархію інтерфейсом "фабрикованих" об'єктів, що представляє набір методів для відображення результатів обчислень.

## 2.2. Текст програми з коментарями

package progect;

import java.io.IOException;

import java.io.BufferedReader;

import java.io.InputStreamReader;

/\*\* Вычисление и отображение результатов.

\* Содержит реализацию статического метода main().

\* @author Егор

\* @see Main#main

\*/

public class Main {

/\*\* Объект класса {@linkplain Calc}.<br>Решает задачу инд. задания. \*/

/\* private final Calc calc = new Calc();\*/

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

public final View view;

/\*\* Инициализирует поле {@linkplain Main#view view}.

\* @param view \*/

public Main(View view) {

this.view = view;

}

/\*\* Отображает меню. \*/

private void menu() {

String s = null;

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

do {

do {

System.out.println("Enter command...");

System.out.print("'e'xit, 'v'iew, 'g'enerate, 's'ave, 'r'estore: ");

try {

s = in.readLine();

} catch(IOException e) {

System.out.println("Error: " + e);

System.exit(0);

}

} while (s.length() != 1);

switch (s.charAt(0)) {

case 'e' -> System.out.println("Exit.");

case 'v' -> {

System.out.println("View current.");

view.viewShow();

}

case 'g' -> {

System.out.println("Random generation.");

view.viewInit();

view.viewShow();

}

case 's' -> {

System.out.println("Save current.");

try {

view.viewSave();

} catch (IOException e) {

System.out.println("Serialization error: " + e);

}

view.viewShow();

}

case 'r' -> {

System.out.println("Restore last saved.");

try {

view.viewRestore();

} catch (Exception e) {

System.out.println("Serialization error: " + e);

}

view.viewShow();

}

default -> System.out.println("Wrong command.");

}

} while(s.charAt(0) != 'e');

}

/\*\* Выполняется при запуске программы;

\* вызывает метод {@linkplain Main#menu() menu()}

\* @param args - параметры запуска программы.

\*/

public static void main(String[] args) {

Main main = new Main(new ViewableResult().getView());

main.menu();

}

}

Рисунок 2.1 – Покращення основного меню

package progect;

import java.io.IOException;

/\*\* Product

\* (шаблон проектирования

\* Factory Method)<br>

\* Интерфейс "фабрикуемых"

\* объектов<br>

\* Объявляет методы

\* отображения объектов

\* @author Егор

\*

\*/

public interface View {

/\*\* Отображает заголовок \*/

public void viewHeader();

/\*\* Отображает основную часть \*/

public void viewBody();

/\*\* Отображает окончание \*/

public void viewFooter();

/\*\* Отображает объект целиком \*/

public void viewShow();

/\*\* Выполняет инициализацию \*/

public void viewInit();

/\*\* Сохраняет данные для последующего восстановления

\* @throws java.io.IOException \*/

public void viewSave() throws IOException;

/\*\* Восстанавливает ранее сохранённые данные

\* @throws java.lang.Exception \*/

public void viewRestore() throws Exception;

}

Рисунок 2.2 – Опис Factory Method(View)

package progect;

/\*\* Creator

\* (шаблон проектирования

\* Factory Method)<br>

\* Объявляет метод,

\* "фабрикующий" объекты

\* @author Егор

\* @see Viewable#getView()

\*/

public interface Viewable {

/\*\* Создаёт объект, реализующий {@linkplain View} \*/

public View getView();

}

Рисунок 2.3 – Опис Factory Method(Viewable)

package progect;

/\*\* ConcreteCreator

\* (шаблон проектирования

\* Factory Method)<br>

\* Объявляет метод,

\* "фабрикующий" объекты

\* @author Егор

\* @see Viewable

\* @see ViewableResult#getView()

\*/

public class ViewableResult implements Viewable {

/\*\* Создаёт отображаемый объект {@linkplain ViewResult}

\* @return \*/

@Override

public View getView() {

return new ViewResult();

}

}

Рисунок 2.4 – Опис Factory Method(ViewableResult)

package progect;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.util.ArrayList;

/\*\* ConcreteProduct

\* (Шаблон проектирования

\* Factory Method)<br>

\* сохранение и отображение

\* результатов

\* @author Егор

\* @see View

\*/

public class ViewResult implements View {

/\*\* Имя файла, используемое при сериализации \*/

private static final String FNAME = "items.bin";

/\*\* Определяет количество значений для вычисления по умолчанию \*/

private static final int DEFAULT\_NUM = 10;

/\*\* Коллекция десятичных чисел и результатов вычислений \*/

private ArrayList<Item2d> items = new ArrayList<>();

/\*\* Вызывает {@linkplain ViewResult#ViewResult(int n) ViewResult(int n)}

\* с параметром {@linkplain ViewResult#DEFAULT\_NUM DEFAULT\_NUM}

\*/

public ViewResult() {

this(DEFAULT\_NUM);

}

/\*\* Инициализирует коллекцию {@linkplain ViewResult#items}

\* @param n начальное количество элементов

\*/

public ViewResult(int n) {

for(int ctr = 0; ctr < n; ctr++) {

items.add(new Item2d());

}

}

/\*\* Получить значение {@linkplain ViewResult#items}

\* @return текущее значение ссылки на объект {@linkplain ArrayList}

\*/

public ArrayList<Item2d> getItems() {

return items;

}

/\*\* Вычисляет значение чередований.

\* @param x - десятичное число.

\* @return результат вычисления.

\*/

private int calc(int x) {

String BinaryS = Integer.toBinaryString((int) x);

int binary = Integer.parseInt (Integer.toBinaryString((int) x));

int y =0;

int buff = binary%10;

binary = binary/10;

for(int i=0;i<BinaryS.length()-2;i++) {

if(buff != binary%10) {

y++;

}

buff = binary%10;

binary = binary/10;

}

return y;

}

/\*\* Вычисляет значение чередований и сохраняет

\* результат в коллекции {@linkplain ViewResult#items}

\* @param stepX шаг приращения десятичного числа

\*/

public void init(int stepX) {

int x =(int)(0 + Math.random()\*1000);

for(Item2d item : items) {

item.setXY(x, calc(x));

x = stepX +(int)(0 + Math.random()\*100) ;

}

}

/\*\* Вызывает <b>init(int stepX)</b> со случайным значением десятичного числа<br>

\* {@inheritDoc}

\*/

@Override

public void viewInit() {

init((int)(0 + Math.random()\*1000));

}

/\*\* Реализация метода {@linkplain View#viewSave()}<br>

\* {@inheritDoc}

\*/

@Override

public void viewSave() throws IOException {

ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream(FNAME));

os.writeObject(items);

os.flush();

os.close();

}

/\*\* Реализация метода {@linkplain View#viewRestore()}<br>

\* {@inheritDoc}

\*/

@SuppressWarnings("unchecked")

@Override

public void viewRestore() throws Exception {

ObjectInputStream is = new ObjectInputStream(new FileInputStream(FNAME));

items = (ArrayList<Item2d>) is.readObject();

is.close();

}

/\*\* Реализация метода {@linkplain View#viewHeader()}<br>

\* {@inheritDoc}

\*/

@Override

public void viewHeader() {

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

}

/\*\* Реализация метода {@linkplain View#viewBody()}<br>

\* {@inheritDoc}

\*/

@Override

public void viewBody() {

for(Item2d item : items) {

System.out.printf("Десятичное число: %d; \nКоличество чередований 0 и 1 в двоичном представлении: %d \n\n",

item.getX(), item.getY());

}

System.out.println();

}

/\*\* Реализация метода {@linkplain View#viewFooter()}<br>

\* {@inheritDoc}

\*/

@Override

public void viewFooter() {

System.out.println("End.");

}

/\*\* Реализация метода {@linkplain View#viewShow()}<br>

\* {@inheritDoc}

\*/

@Override

public void viewShow() {

viewHeader();

viewBody();

viewFooter();

}

}

Рисунок 2.5 – Опис Factory Method(ViewResult)

## 2.3. Результати роботи програми

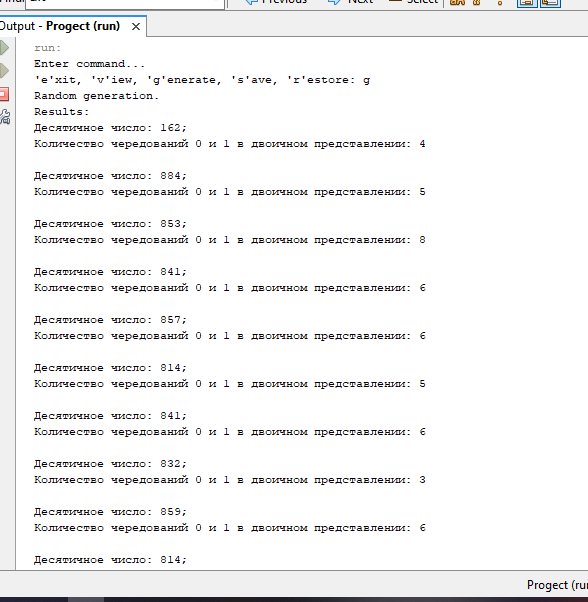


Рисунок 2.6 – Виведення на екран результат роботи програми

# 3. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №3

## 3.1. Постановка завдання

* Розширити ієрархію похідними класами, реалізують методи для подання результатів у вигляді текстової таблиці.
* Розширити ієрархію інтерфейсом "фабрикованих" об'єктів, що представляє набір методів для відображення результатів обчислень.

## 3.2. Текст програми з коментарями

package Polimorphism;  
  
import progect.View;  
  
*/\*\* Вычисление и отображение результатов<br>*

*\* Содержит реализацию статического метода main()*

*\* @author Егор*

*\* @see Main#main*

*\*/*public class Main extends ViewInterface.Main {  
  
 public Main(View view) {  
 super(view);  
 }  
  
 public static void main(String[] args) {  
 Main main = new Main(new ViewableTable().getView());  
 main.menu();  
 }  
}

Рисунок 3.1 – Меню

package progect\_2;

import progect.ViewableResult;

import progect.View;

/\*\* ConcreteCreator

\* (шаблон проектирования

\* Factory Method)<br>

\* Объявляет метод,

\* "фабрикующий" объекты

\* @author Егор

\* @see ViewableResult

\* @see ViewableTable#getView()

\*/

public class ViewableTable extends ViewableResult {

/\*\* Создаёт отображаемый объект {@linkplain ViewTable} \*/

@Override

public View getView() {

return new ViewTable();

}

}

Рисунок 3.2 – Опис класу ViewableTable

package progect\_2;

import java.util.Formatter;

import progect.Item2d;

import progect.ViewResult;

/\*\* ConcreteProduct

\* (шаблон проектирования

\* Factory Method)<br>

\* Вывод в виде таблицы

\* @author Егор

\* @see ViewResult

\*/

public class ViewTable extends ViewResult {

/\*\* Определяет ширину таблицы по умолчанию \*/

private static final int DEFAULT\_WIDTH = 20;

/\*\* Текущая ширина таблицы \*/

private int width;

/\*\* Устанавливает {@linkplain ViewTable#width width}

\* значением {@linkplain ViewTable#DEFAULT\_WIDTH DEFAULT\_WIDTH}<br>

\* Вызывается конструктор суперкласса {@linkplain ViewResult#ViewResult() ViewResult()}

\*/

public ViewTable() {

width = DEFAULT\_WIDTH;

}

/\*\* Устанавливает {@linkplain ViewTable#width} значением <b>width</b><br>

\* Вызывается конструктор суперкласса {@linkplain ViewResult#ViewResult() ViewResult()}

\* @param width определяет ширину таблицы

\*/

public ViewTable(int width) {

this.width = width;

}

/\*\* Устанавливает {@linkplain ViewTable#width} значением <b>width</b><br>

\* Вызывается конструктор суперкласса {@linkplain ViewResult#ViewResult(int n) ViewResult(int

n)}

\* @param width определяет ширину таблицы

\* @param n количество элементов коллекции; передаётся суперконструктору

\*/

public ViewTable(int width, int n) {

super(n);

this.width = width;

}

/\*\* Устанавливает поле {@linkplain ViewTable#width} значением <b>width</b>

\* @param width новая ширина таблицы

\* @return заданная параметром <b>width</b> ширина таблицы

\*/

public int setWidth(int width) {

return this.width = width;

}

/\*\* Возвращает значение поля {@linkplain ViewTable#width}

\* @return текущая ширина таблицы

\*/

public int getWidth() {

return width;

}

/\*\* Выводит вертикальный разделитель шириной {@linkplain ViewTable#width} символов \*/

private void outLine() {

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

System.out.print("--");

}

}

/\*\* Вызывает {@linkplain ViewTable#outLine()}; завершает вывод разделителем строки \*/

private void outLineLn() {

outLine();

System.out.println();

}

/\*\* Выводит заголовок таблицы шириной {@linkplain ViewTable#width} символов \*/

private void outHeader() {

Formatter fmt = new Formatter();

fmt.format("%s%d%s%2$d%s", "%", (width-3)/2, "s | %", "s\n");

System.out.printf(fmt.toString(), "Десятичное число ", "количевство чередований ");

}

/\*\* Выводит тело таблицы шириной {@linkplain ViewTable#width} символов \*/

private void outBody() {

Formatter fmt = new Formatter();

fmt.format("%s%d%s%2$d%s", "%", (width-3)/2, "d | %", "d\n");

for(Item2d item : getItems()) {

System.out.printf(fmt.toString(), item.getX(), item.getY());

}

}

/\*\* Перегрузка метода суперкласса;

\* устанавливает поле {@linkplain ViewTable#width} значением <b>width</b><br>

\* Для объекта {@linkplain ViewTable} вызывает метод {@linkplain ViewTable#init(int

stepX)}

\* @param width новая ширина таблицы.

\* @param stepX передается методу <b>init(int)</b>

\*/

public final void init(int width, int stepX) { // method overloading

this.width = width;

init(stepX);

}

/\*\* Переопределение (замещение, overriding) метода суперкласса;

\* выводит информационное сообщение и вызывает метод суперкласса

\* {@linkplain ViewResult#init(int stepX) init(int stepX)}<br>

\* {@inheritDoc}

\*/

public void init(double stepX) { // method overriding

System.out.print("Initialization... ");

super.init((int) stepX);

System.out.println("done. ");

}

/\*\* Вывод элемента таблицы<br>{@inheritDoc} \*/

@Override

public void viewHeader() {

outHeader();

outLineLn();

}

/\*\* Вывод элемента таблицы<br>{@inheritDoc} \*/

@Override

public void viewBody() {

outBody();

}

/\*\* Вывод элемента таблицы<br>{@inheritDoc} \*/

@Override

public void viewFooter() {

outLineLn();

}

}

Рисунок 3.3 – Опис класу ViewTable

## 3.3. Результати роботи програми

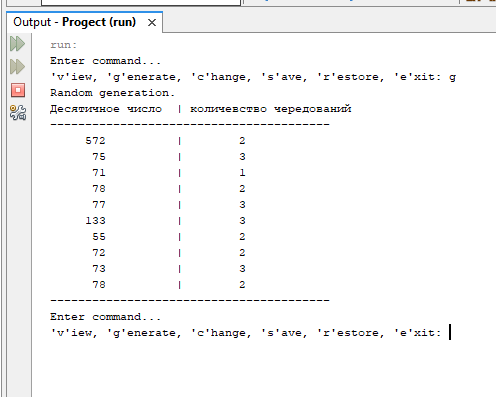


Рисунок 3.4 – Виведення результату роботи на екран

4. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №4

## 4.1. **Постановка завдання**

* Реалізувати можливість скасування (undo) операцій (команд).
* Продемонструвати поняття "макрокоманда".

## 4.2. Текст програми з коментарями

package progect\_3;

import progect.View;

import progect\_2.ViewableTable;

/\*\* Формирует и отображает

\* меню; реализует шаблон

\* Singleton

\* @author Еор

\*/

public class Application {

/\*\* Ссылка на экземпляр класса Application; шаблон Singleton

\* @see Application

\*/

private static Application instance = new Application();

/\*\* Закрытый конструктор; шаблон Singleton

\* @see Application

\*/

private Application() {}

/\*\* Возвращает ссылку на экземпляр класса Application;

\* шаблон Singleton

\* @see Application

\*/

public static Application getInstance() {

return instance;

}

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d};

\* инициализируется с помощью Factory Method

\*/

private View view = new ViewableTable().getView();

/\*\* Объект класса {@linkplain Menu};

\* макрокоманда (шаблон Command)

\*/

private Menu menu = new Menu();

/\*\* Обработка команд пользователя

\* @see Application

\*/

public void run() {

menu.add(new ViewConsoleCommand(view));

menu.add(new GenerateConsoleCommand(view));

menu.add(new ChangeConsoleCommand(view));

menu.add(new SaveConsoleCommand(view));

menu.add(new RestoreConsoleCommand(view));

menu.execute();

}

}

Рисунок 4.1 – Опис класу Application

package progect\_3;

import progect.Item2d;

import progect.View;

import progect.ViewResult;

/\*\* Консольная команда

\* Change item;

\* шаблон Command

\* @author Егор

\*/

public class ChangeConsoleCommand

extends ChangeItemCommand

implements ConsoleCommand {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

private View view;

/\*\* Возвращает поле {@linkplain ChangeConsoleCommand#view}

\* @return значение {@linkplain ChangeConsoleCommand#view}

\*/

public View getView() {

return view;

}

/\*\* Устанавливает поле {@linkplain ChangeConsoleCommand#view}

\* @param view значение для {@linkplain ChangeConsoleCommand#view}

\* @return новое значение {@linkplain ChangeConsoleCommand#view}

\*/

public View setView(View view) {

return this.view = view;

}

/\*\* Инициализирует поле {@linkplain ChangeConsoleCommand#view}

\* @param view объект, реализующий интерфейс {@linkplain View}

\*/

public ChangeConsoleCommand(View view) {

this.view = view;

}

@Override

public char getKey() {

return 'c';

}

@Override

public String toString() {

return "'c'hange";

}

@Override

public void execute() {

System.out.println("Change item: scale factor " + setOffset((int) (Math.random() \* 100.0)));

for (Item2d item : ((ViewResult)view).getItems()) {

super.setItem(item);

super.execute();

}

view.viewShow();

}

}

Рисунок 4.2 – Опис класу ChangeConsoleCommand

package progect\_3;

import progect.Item2d;

/\*\* Команда

\* Change item;

\* шаблон Command

\* @author Егор

\*/

public class ChangeItemCommand implements Command {

/\*\* Обрабатываемый объект; шаблон Command \*/

private Item2d item;

/\*\* Параметр команды; шаблон Command \*/

private double offset;

/\*\* Устанавливаент поле {@linkplain ChangeItemCommand#item}

\* @param item значение для {@linkplain ChangeItemCommand#item}

\* @return новое значение {@linkplain ChangeItemCommand#item}

\*/

public Item2d setItem(Item2d item) {

return this.item = item;

}

/\*\* Возвращает поле {@linkplain ChangeItemCommand#item}

\* @return значение {@linkplain ChangeItemCommand#item}

\*/

public Item2d getItem() {

return item;

}

/\*\* Устанавливаент поле {@linkplain ChangeItemCommand#offset}

\* @param offset значение для {@linkplain ChangeItemCommand#offset}

\* @return новое значение {@linkplain ChangeItemCommand#offset}

\*/

public double setOffset(int offset) {

return this.offset = offset;

}

/\*\* Возвращает поле {@linkplain ChangeItemCommand#offset}

\* @return значение {@linkplain ChangeItemCommand#offset}

\*/

public int getOffset() {

return (int) offset;

}

@Override

public void execute() {

item.setX((int) (item.getX() + offset));

}

}

Рисунок 4.3 – Опис класу ChangeItemCommand

package progect\_3;

/\*\* Интерфейс команды

\* или задачи;

\* шаблоны: Command,

\* Worker Thread

\* @author Егор

\*/

public interface Command {

/\*\* Выполнение команды; шаблоны: Command, Worker Thread \*/

public void execute();

}

Рисунок 4.3 – Опис класу Command

package progect\_3;

/\*\* Интерфейс

\* консольной команды;

\* шаблон Command

\* @author Егор

\*/

public interface ConsoleCommand extends Command {

/\*\* Горячая клавиша команды;

\* шаблон Command

\* @return символ горячей клавиши

\*/

public char getKey();

}

Рисунок 4.3 – Опис класу ConsoleCommand

package progect\_3;

import progect.View;

/\*\* Консольная команда

\* Generate;

\* шаблон Command

\* @author Егор

\*/

public class GenerateConsoleCommand implements ConsoleCommand {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

private View view;

/\*\* Инициализирует поле {@linkplain GenerateConsoleCommand#view}

\* @param view объект, реализующий интерфейс {@linkplain View}

\*/

public GenerateConsoleCommand(View view) {

this.view = view;

}

@Override

public char getKey() {

return 'g';

}

@Override

public String toString() {

return "'g'enerate";

}

@Override

public void execute() {

System.out.println("Random generation.");

view.viewInit();

view.viewShow();

}

}

Рисунок 4.3 – Опис класу GenerateConsoleCommand

package progect\_3;

/\*\* Вычисление и отображение

\* результатов; cодержит реализацию

\* статического метода main()

\* @author Егор

\* @see Main#main

\*/

public class Main {

/\*\* Выполняется при запуске программы;

\* вызывает метод {@linkplain Application#run()}

\* @param args параметры запуска программы

\*/

public static void main(String[] args) {

Application app = Application.getInstance();

app.run();

}

}

Рисунок 4.3 – Опис класу Main

package progect\_3;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

/\*\* Макрокоманда

\* (шаблон Command);

\* Коллекция объектов

\* класса ConsoleCommand

\* @see ConsoleCommand

\*/

public class Menu implements Command {

/\*\* Коллекция консольных команд;

\* @see ConsoleCommand

\*/

private List<ConsoleCommand> menu = new ArrayList<ConsoleCommand>();

/\*\* Добавляет новую команду в коллекцию

\* @param command реализует {@linkplain ConsoleCommand}

\* @return command

\*/

public ConsoleCommand add(ConsoleCommand command) {

menu.add(command);

return command;

}

@Override

public String toString() {

String s = "Enter command...\n";

for (ConsoleCommand c : menu) {

s += c + ", ";

}

s += "'e'xit: ";

return s;

}

@Override

public void execute() {

String s = null;

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

menu: while (true) {

do {

System.out.print(this);

try {

s = in.readLine();

} catch (IOException e) {

System.err.println("Error: " + e);

System.exit(0);

}

} while (s.length() != 1);

char key = s.charAt(0);

if (key == 'e') {

System.out.println("Exit.");

break menu;

}

for (ConsoleCommand c : menu) {

if (s.charAt(0) == c.getKey()) {

c.execute();

continue menu;

}

}

System.out.println("Wrong command.");

continue menu;

}

}

}

Рисунок 4.3 – Покращення основного меню

package progect\_3;

import progect.View;

/\*\* Консольная команда

\* Restore;

\* шаблон Command

\* @author Егор

\*/

public class RestoreConsoleCommand implements ConsoleCommand {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

private View view;

/\*\* Инициализирует поле {@linkplain RestoreConsoleCommand#view}

\* @param view объект, реализующий интерфейс {@linkplain View}

\*/

public RestoreConsoleCommand(View view) {

this.view = view;

}

@Override

public char getKey() {

return 'r';

}

@Override

public String toString() {

return "'r'estore";

}

@Override

public void execute() {

System.out.println("Restore last saved.");

try {

view.viewRestore();

} catch (Exception e) {

System.err.println("Serialization error: " + e);

}

view.viewShow();

}

}

Рисунок 4.3 – Опис класу RestoreConsoleCommand

package progect\_3;

import java.io.IOException;

import progect.View;

/\*\* Консольная команда

\* Save;

\* шаблон Command

\* @author Егор

\*/

public class SaveConsoleCommand implements ConsoleCommand {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

private View view;

/\*\* Инициализирует поле {@linkplain SaveConsoleCommand#view}

\* @param view объект, реализующий интерфейс {@linkplain View}

\*/

public SaveConsoleCommand(View view) {

this.view = view;

}

@Override

public char getKey() {

return 's';

}

@Override

public String toString() {

return "'s'ave";

}

@Override

public void execute() {

System.out.println("Save current.");

try {

view.viewSave();

} catch (IOException e) {

System.err.println("Serialization error: " + e);

}

view.viewShow();

}

}

Рисунок 4.3 – Опис класу SaveConsoleCommand

package progect\_3;

import progect.View;

/\*\* Консольная команда

\* View;

\* шаблон Command

\* @author Егор

\*/

public class ViewConsoleCommand implements ConsoleCommand {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

private View view;

/\*\* Инициализирует поле {@linkplain SaveConsoleCommand#view}

\* @param view объект, реализующий интерфейс {@linkplain View}

\*/

public ViewConsoleCommand(View view) {

this.view = view;

}

@Override

public char getKey() {

return 'v';

}

@Override

public String toString() {

return "'v'iew";

}

@Override

public void execute() {

System.out.println("View current.");

view.viewShow();

}

}

Рисунок 4.3 – Опис класу ViewConsoleCommand

## 4.3. Результати роботи програми

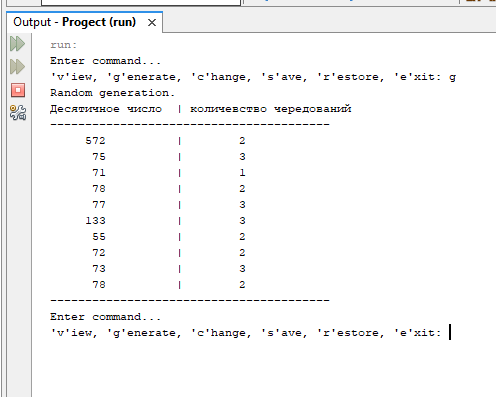


Рисунок 4.4 – Виведення результату роботи на екран

# 5. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №5

## 5.1. **Постановка завдання**

Розробити паралельну обробку елементів колекції (пошук мінімуму, максимуму, обчислення середнього значення, відбір за критерієм, статистична обробка тощо).

## 5.2. Текст програми з коментарями

package progect\_4;

import progect.View;

import progect.ViewableResult;

import progect\_3.ChangeConsoleCommand;

import progect\_3.GenerateConsoleCommand;

import progect\_3.Menu;

import progect\_3.ViewConsoleCommand;

/\*\* Вычисление и отображение

\* результатов; содержит реализацию

\* статического метода main()

\* @author Егор

\* @see Main#main

\*/

public class Main {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d};

\* инициализируется с помощью Factory Method

\*/

private View view = new ViewableResult().getView();

/\*\* Объект класса {@linkplain Menu};

\* макрокоманда (шаблон Command)

\*/

private Menu menu = new Menu();

/\*\* Обработка команд пользователя \*/

public void run() {

menu.add(new ViewConsoleCommand(view));

menu.add(new GenerateConsoleCommand(view));

menu.add(new ChangeConsoleCommand(view));

menu.add(new ExecuteConsoleCommand(view));

menu.execute();

}

/\*\* Выполняется при запуске программы

\* @param args параметры запуска программы

\*/

public static void main(String[] args) {

Main main = new Main();

main.run();

}

}

Рисунок 5.1 – Опис класу Main

package progect\_4;

import java.util.concurrent.TimeUnit;

import progect.Item2d;

import progect.ViewResult;

import progect\_3.Command;

/\*\* Задача, используемая

\* обработчиком потока;

\* шаблон Worker Thread

\* @author Егор

\* @see Command

\* @see CommandQueue

\*/

public class AvgCommand implements Command /\*, Runnable \*/ {

/\*\* Хранит результат обработки коллекции \*/

private int result = 0;

/\*\* Флаг готовности результата \*/

private int progress = 0;

/\*\* Обслуживает коллекцию объектов {@linkplain progect.Item2d} \*/

private ViewResult viewResult;

/\*\* Возвращает поле {@linkplain MaxCommand#viewResult}

\* @return значение {@linkplain MaxCommand#viewResult}

\*/

public ViewResult getViewResult() {

return viewResult;

}

/\*\* Устанавливает поле {@linkplain MaxCommand#viewResult}

\* @param viewResult значение для {@linkplain MaxCommand#viewResult}

\* @return новое значение {@linkplain MaxCommand#viewResult}

\*/

public ViewResult setViewResult(ViewResult viewResult) {

return this.viewResult = viewResult;

}

/\*\* Инициализирует поле {@linkplain MaxCommand#viewResult}

\* @param viewResult объект класса {@linkplain ViewResult}

\*/

public AvgCommand(ViewResult viewResult) {

this.viewResult = viewResult;

}

/\*\* Возвращает результат

\* @return поле {@linkplain MaxCommand#result}

\*/

public int getResult() {

return result;

}

/\*\* Проверяет готовность результата

\* @return false - если результат найден, иначе - true

\* @see MaxCommand#result

\*/

public boolean running() {

return progress < 1000;

}

/\*\* Используется обработчиком потока {@linkplain CommandQueue};

\* шаблон Worker Thread

\*/

@Override

public void execute() {

progress = 0;

System.out.println("Average executed...");

result = 0;

int idx = 1, size = viewResult.getItems().size();

for (Item2d item : viewResult.getItems()) {

result += item.getY();

progress = idx \* 100 / size;

if (idx++ % (size / 2) == 0) {

System.out.println("Average " + progress + "%");

}

try {

TimeUnit.MILLISECONDS.sleep(2000 / size);

} catch (InterruptedException e) {

System.err.println(e);

}

}

result /= size;

System.out.println("Average done. Result = " + String.format("%d",result));

progress = 100;

}

}

Рисунок 5.2 – Опис класу AvgCommand

package progect\_4;

import java.util.Vector;

import progect\_3.Command;

/\*\* Создает обработчик

\* потока, выполняющего

\* объекты с интерфейсом

\* Command; шаблон

\* Worker Thread

\* @author Егор

\* @see Command

\*/

public class CommandQueue implements Queue {

/\*\* Очередь задач \*/

private Vector<Command> tasks;

/\*\* Флаг ожидания \*/

private boolean waiting;

/\*\* Флаг завершения \*/

private boolean shutdown;

/\*\* Устанавливает флаг завершения \*/

public void shutdown() {

shutdown = true;

}

/\*\* Инициализация {@linkplain CommandQueue#tasks}

\* {@linkplain CommandQueue#waiting}

\* {@linkplain CommandQueue#waiting};

\* создает поток для класса {@linkplain CommandQueue.Worker}

\*/

public CommandQueue() {

tasks = new Vector<Command>();

waiting = false;

new Thread(new Worker()).start();

}

@Override

public void put(Command r) {

tasks.add(r);

if (waiting) {

synchronized (this) {

notifyAll();

}

}

}

@Override

public Command take() {

if (tasks.isEmpty()) {

synchronized (this) {

waiting = true;

try {

wait();

} catch (InterruptedException ie) {

waiting = false;

}

}

}

return (Command)tasks.remove(0);

}

/\*\* Обслуживает очередь

\* задач; шаблон

\* Worker Thread

\* @author xone

\* @version 1.0

\* @see Runnable

\*/

private class Worker implements Runnable {

/\*\* Извлекает из очереди

\* готовые к выполнению

\* задачи; шаблон

\* Worker Thread \*/

public void run() {

while (!shutdown) {

Command r = take();

r.execute();

}

}

}

}

Рисунок 5.2 – Опис класу CommandQueue

package progect\_4;

/\*\*

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

/\*\*/

import java.util.concurrent.TimeUnit;

import progect.View;

import progect.ViewResult;

import progect\_3.ConsoleCommand;

/\*\* Консольная команда

\* Execute all threads;

\* шаблон Command

\* @author Егор

\*/

public class ExecuteConsoleCommand implements ConsoleCommand {

/\*\* Объект, реализующий интерфейс {@linkplain View};

\* обслуживает коллекцию объектов {@linkplain progect.Item2d}

\*/

private View view;

/\*\* Возвращает поле {@linkplain ExecuteConsoleCommand#view}

\* @return значение {@linkplain ExecuteConsoleCommand#view}

\*/

public View getView() {

return view;

}

/\*\* Устанавливает поле {@linkplain ExecuteConsoleCommand#view}

\* @param view значение для {@linkplain ExecuteConsoleCommand#view}

\* @return новое значение {@linkplain ExecuteConsoleCommand#view}

\*/

public View setView(View view) {

return this.view = view;

}

/\*\* Инициализирует поле {@linkplain ExecuteConsoleCommand#view}

\* @param view объект, реализующий {@linkplain View}

\*/

public ExecuteConsoleCommand(View view) {

this.view = view;

}

@Override

public char getKey() {

return 'h';

}

@Override

public String toString() {

return "'h'execute";

}

@Override

public void execute() {

/\*\*/

CommandQueue queue1 = new CommandQueue();

CommandQueue queue2 = new CommandQueue();

/\*\*

ExecutorService exec1 = Executors.newSingleThreadExecutor();

ExecutorService exec2 = Executors.newSingleThreadExecutor();

/\*\*/

MaxCommand maxCommand = new MaxCommand((ViewResult)view);

AvgCommand avgCommand = new AvgCommand((ViewResult)view);

MinMaxCommand minMaxCommand = new MinMaxCommand((ViewResult)view);

System.out.println("Execute all threads...");

/\*\*

exec1.execute(minMaxCommand);

exec2.execute(maxCommand);

exec2.execute(avgCommand);

/\*\*/

queue1.put(minMaxCommand);

queue2.put(maxCommand);

queue2.put(avgCommand);

/\*\*/

try {

while (avgCommand.running() ||

maxCommand.running() ||

minMaxCommand.running()) {

TimeUnit.MILLISECONDS.sleep(100);

}

/\*\*

exec1.shutdown();

exec2.shutdown();

/\*\*/

queue1.shutdown();

queue2.shutdown();

/\*\*/

TimeUnit.SECONDS.sleep(1);

} catch (InterruptedException e) {

System.err.println(e);

}

System.out.println("All done.");

}}

Рисунок 5.2 – Опис класу ExecuteConsoleCommand

package progect\_4;

import java.util.concurrent.TimeUnit;

import progect.ViewResult;

import progect\_3.Command;

/\*\* Задача, используемая

\* обработчиком потока;

\* шаблон Worker Thread

\* @author Егор

\* @see Command

\* @see CommandQueue

\*/

public class MaxCommand implements Command /\*, Runnable \*/ {

/\*\* Хранит результат обработки коллекции \*/

private int result = -1;

/\*\* Флаг готовности результата \*/

private int progress = 0;

/\*\* Обслуживает коллекцию объектов {@linkplain progect.Item2d} \*/

private ViewResult viewResult;

/\*\* Возвращает поле {@linkplain MaxCommand#viewResult}

\* @return значение {@linkplain MaxCommand#viewResult}

\*/

public ViewResult getViewResult() {

return viewResult;

}

/\*\* Устанавливает поле {@linkplain MaxCommand#viewResult}

\* @param viewResult значение для {@linkplain MaxCommand#viewResult}

\* @return новое значение {@linkplain MaxCommand#viewResult}

\*/

public ViewResult setViewResult(ViewResult viewResult) {

return this.viewResult = viewResult;

}

/\*\* Инициализирует поле {@linkplain MaxCommand#viewResult}

\* @param viewResult объект класса {@linkplain ViewResult}

\*/

public MaxCommand(ViewResult viewResult) {

this.viewResult = viewResult;

}

/\*\* Возвращает результат

\* @return поле {@linkplain MaxCommand#result}

\*/

public int getResult() {

return result;

}

/\*\* Проверяет готовность результата

\* @return false - если результат найден, иначе - true

\* @see MaxCommand#result

\*/

public boolean running() {

return progress < 1000;

}

/\*\* Используется обработчиком потока {@linkplain CommandQueue};

\* шаблон Worker Thread

\*/

@Override

public void execute() {

progress = 0;

System.out.println("Max executed...");

int size = viewResult.getItems().size();

result = 0;

for (int idx = 1; idx < size; idx++) {

if (viewResult.getItems().get(result).getY() <

viewResult.getItems().get(idx).getY()) {

result = idx;

}

progress = idx \* 100 / size;

if (idx % (size / 3) == 0) {

System.out.println("Max " + progress + "%");

}

try {

TimeUnit.MILLISECONDS.sleep(3000 / size);

} catch (InterruptedException e) {

System.err.println(e);

}

}

System.out.println("Max done. Item #" + result +

" found: " + viewResult.getItems().get(result));

progress = 100;

}

}

Рисунок 5.2 – Опис класу MaxCommand

package progect\_4;

import java.util.concurrent.TimeUnit;

import progect.Item2d;

import progect.ViewResult;

import progect\_3.Command;

/\*\* Задача, используемая

\* обработчиком потока;

\* шаблон Worker Thread

\* @author Егор

\* @see Command

\* @see CommandQueue

\*/

public class MinMaxCommand implements Command /\*, Runnable \*/ {

/\*\* Хранит результат обработки коллекции \*/

private int resultMin = -1;

/\*\* Хранит результат обработки коллекции \*/

private int resultMax = -1;

/\*\* Флаг готовности результата \*/

private int progress = 0;

/\*\* Обслуживает коллекцию объектов {@linkplain progect.Item2d} \*/

private ViewResult viewResult;

/\*\* Возвращает поле {@linkplain MinMaxCommand#viewResult}

\* @return значение {@linkplain MinMaxCommand#viewResult}

\*/

public ViewResult getViewResult() {

return viewResult;

}

/\*\* Устанавливает поле {@linkplain MinMaxCommand#viewResult}

\* @param viewResult значение для {@linkplain MinMaxCommand#viewResult}

\* @return новое значение {@linkplain MinMaxCommand#viewResult}

\*/

public ViewResult setViewResult(ViewResult viewResult) {

return this.viewResult = viewResult;

}

/\*\* Инициализирует поле {@linkplain MinMaxCommand#viewResult}

\* @param viewResult объект класса {@linkplain ViewResult}

\*/

public MinMaxCommand(ViewResult viewResult) {

this.viewResult = viewResult;

}

/\*\* Возвращает результат

\* @return поле {@linkplain MinMaxCommand#resultMin}

\*/

public int getResultMin() {

return resultMin;

}

/\*\* Возвращает результат

\* @return поле {@linkplain MinMaxCommand#resultMax}

\*/

public int getResultMax() {

return resultMax;

}

/\*\* Проверяет готовность результата

\* @return false - если результат найден, иначе - true

\*/

public boolean running() {

return progress < 100;

}

/\*\* Используется обработчиком потока {@linkplain CommandQueue};

\* шаблон Worker Thread

\*/

@Override

public void execute() {

progress = 0;

System.out.println("MinMax executed...");

int idx = 0, size = viewResult.getItems().size();

for (Item2d item : viewResult.getItems()) {

if (item.getY() < 0) {

if ((resultMax == -1) ||

(viewResult.getItems().get(resultMax).getY() <

item.getY())) {

resultMax = idx;

}

} else {

if ((resultMin == -1) ||

(viewResult.getItems().get(resultMin).getY() >

item.getY())) {

resultMin = idx;

}

}

idx++;

progress = idx \* 100 / size;

if (idx % (size / 5) == 0) {

System.out.println("MinMax " + progress + "%");

}

try {

TimeUnit.MILLISECONDS.sleep(5000 / size);

} catch (InterruptedException e) {

System.err.println(e);}}

System.out.print("MinMax done.\n ");

if (resultMin > -1) {

System.out.print("Min positive #" + resultMin + " found: " +

String.format("%d", viewResult.getItems().get(resultMin).getY()));

progress = 100;

} else {

System.out.print("\n Min positive not found.");}

if (resultMax > -1) {

System.out.println("\n Max negative #" + resultMax + " found: " +

String.format("%d",

viewResult.getItems().get(resultMax).getY()));} else {

System.out.println("\n Max negative item not found.");}

progress = 100;}}

Рисунок 5.2 – Опис класу MinMaxCommand

package progect\_4;

import progect\_3.Command;

/\*\* Представляет

\* методы для помещения

\* и извлечения задач

\* обработчиком потока;

\* шаблон Worker Thread

\* @author Егор

\* @see Command

\*/

public interface Queue {

/\*\* Добавляет новую задачу в очередь;

\* шаблон Worker Thread

\* @param cmd задача

\*/

void put(Command cmd);

/\*\* Удаляет задачу из очереди;

\* шаблон Worker Thread

\* @return удаляемая задача

\*/

Command take();

}

Рисунок 5.2 – Опис класу Queue

## 5.3. Результати роботи програми

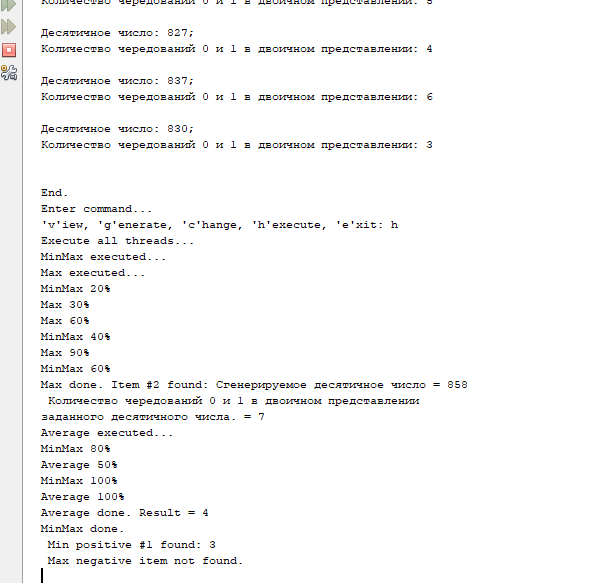


Рисунок 5.3 – Результат роботи програми

# ВИСНОВКИ

В ході практики з «Об’єктно-орієнтованого програмування» я сформував професійні вміння, навички, приймав самостійні рішення на певних ділянках роботи в реальних виробничих умовах шляхом виконання окремих функцій і завдань, властивих майбутній професії. Також, закріпив теоретичні знання із спеціальних і орієнтованих на організацію професійної діяльності дисциплін, набув практичних навичок і досвіду відповідно до профілю обраної спеціальності, виховання потреби систематично поновлювати свої знання та творчо застосовувати їх у практичній діяльності.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Kathy Sierra, Bert Bates. “Head First Java”: Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
2. Raoul-Gabriel Urma,  Mario Fusco, Alan Mycroft. “Java in action”: ©2015 by Manning Publications Co.
3. Блох, Джошуа Б70 Java: эффективное программирование, 3-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2019. — 464 с.: ил. — Парал. тит. англ.
4. GeeksforGeeks [Електронний ресурс]. – Режим доступу: URL: <https://www.geeksforgeeks.org/array-class-in-java/> (дата звернення 25.03.2022). – Назва з екрану.
5. JavaRush [Електронний ресурс]. – Режим доступу: URL: <https://javarush.ru/> (дата звернення 24.03.2022). – Назва з екрану.
6. Вікіпідручник [Електронний ресурс]. – Режим доступу: URL: https://uk.wikibooks.org/ (дата звернення 24.03.2022). – Назва з екрану.