Розробник:

Огонькова Наталія Максимівна

Група КІТ-118б

Варіант №11

**Лабораторна робота № 10**

Обробка параметризованих контейнерів

**Мета:**

* Розширення функціональності параметризованих класів.

**Вимоги:**

1. Використовуючи програму рішення завдання лабораторної роботи №9:
2. Розробити параметризовані методи (Generic Methods) для обробки колекцій об’єктів згідно прикладної задачі.
3. Продемонструвати розроблену функціональність (створення, управління та обробку власних контейнерів) в діалоговому та автоматичному режимах.
4. Автоматичний режим виконання програми задається параметром командного рядка -auto. Наприклад, java ClassName -auto.
5. В автоматичному режимі діалог з користувачем відсутній, необхідні данні генеруються, або зчитуються з файлу.
6. Забороняється використання алгоритмів з Java Collections Framework.
7. ОПИС ПРОГРАМИ
   1. Опис змінних

**private** String name;

**private** String unit;

**private** **int** count;

**private** **int** unit\_price;

**private** String data\_of\_receipt;

**private** String attribute;

**private** String value;

* 1. Ієрархія та структура класів

**class** Lab10 – точка входу в програму

**class** Demo – точка входу в програму

**class** UI – клас с описом двох режимів входу в програму

**class** Until – клас с описом параметризованих методів

**class** Container – клас, що містить функції

**class** Store – клас, що містить дані, введені користувачем

**class** Linked – клас, що містить інтерфейс з прототипами функцій

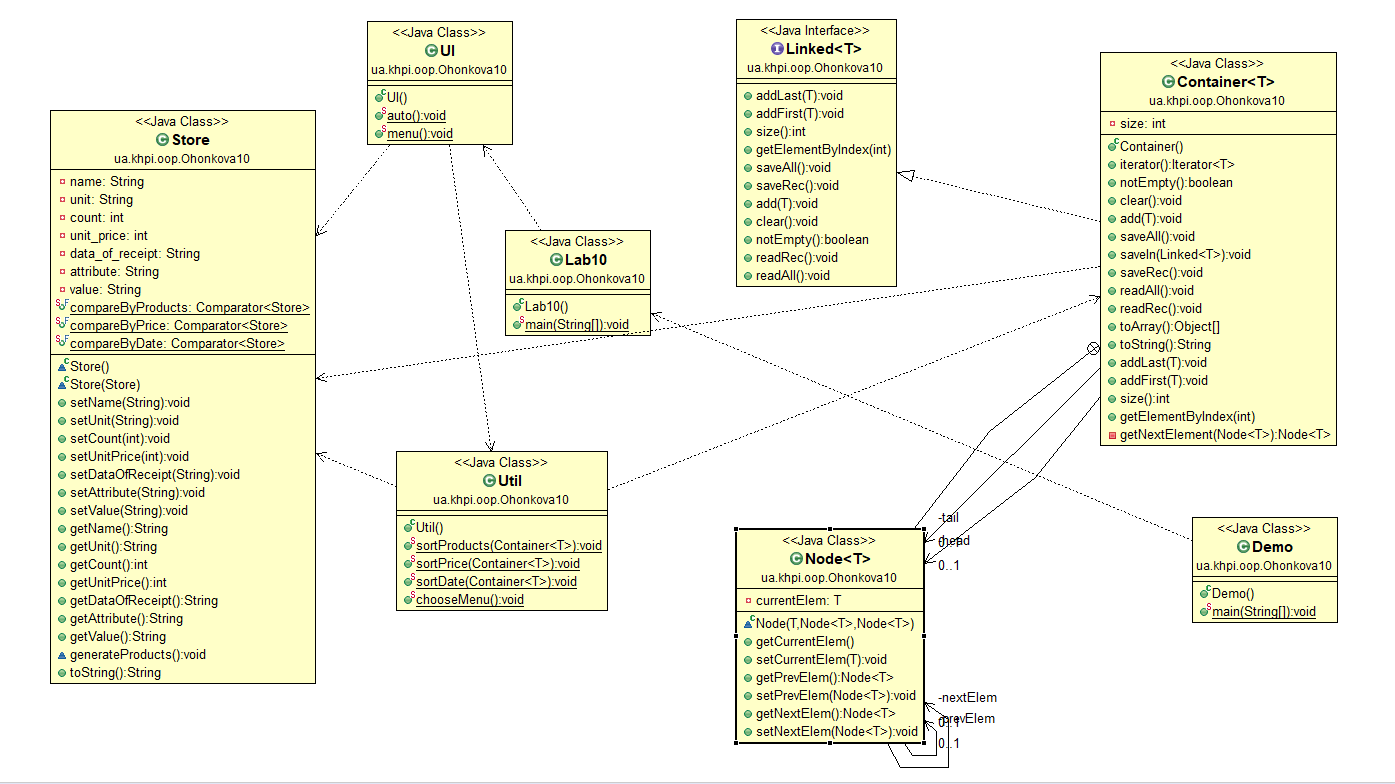


Рисунок 1 – Класи програми та їх зв’язки

* 1. Текст програми

**class** Lab10

package ua.khpi.oop.Ohonkova10;

import java.io.IOException;

public class Lab10 {

public static void main(String[] args) throws IOException {

if (args.length != 0) {

if (args[0].equals("-auto")) {

UI.*auto*();

} else {

System.*out*.println("Repeat entered with params -auto");

}

} else {

UI.*menu*();

}

}

}

**class** Demo

package ua.khpi.oop.Ohonkova10;

import java.io.IOException;

public class Demo {

public static void main(String args[]) throws IOException {

// Lab10.main(new String[]{"-auto"});

Lab10.*main*(args);

}

}

**class** UI

package ua.khpi.oop.Ohonkova10;

import java.io.IOException;

import java.util.Scanner;

import ua.khpi.oop.Ohonkova10.Container;

import ua.khpi.oop.Ohonkova10.Store;

import ua.khpi.oop.Ohonkova10.Util;

public class UI {

public static void auto() throws IOException {

Store temp = new Store();

Container<Store> container = new Container<>();

System.out.println(container.size());

container.add(temp);

System.out.println(container.size());

container.readRec();

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

int counter = 0;

for (Store s : container) {

System.out.println("#" + ++counter);

System.out.println(s);

}

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

System.out.println(container.size());

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

container.saveIn(container);

System.out.println(container.getElementByIndex(0));

container.clear();

System.out.println(container.size());

}

public static void menu() throws IOException {

System.out.println("It is menu mode");

Container<Store> linkedContainer = new Container<>();

Store rec1 = new Store();

Scanner scan = new Scanner(System.in);

boolean loop = true;

while (loop) {

Util.chooseMenu();

int choose = scan.nextInt();

switch (choose) {

case 1:

rec1.generateProducts();

linkedContainer.add(rec1);

System.out.println("Done!");

break;

case 2:

if (linkedContainer.notEmpty()) {

int count = 0;

for (Store s : linkedContainer) {

System.out.println();

System.out.println(++count);

System.out.println(s);

}

} else {

System.out.println("List is empty!");

}

break;

case 3:

if (linkedContainer.notEmpty()) {

linkedContainer.clear();

System.out.println("Successfully!");

} else {

System.out.println("Array is empty!");

}

break;

case 4:

if (linkedContainer.notEmpty()) {

System.out.println("Your container have data.");

} else {

System.out.println("Your container doesn`t have data.");

}

break;

case 5:

System.out.println("Size of container: " + linkedContainer.size());

break;

case 6:

if (linkedContainer.notEmpty()) {

System.out.print("Entered index from 0 to " + (linkedContainer.size() - 1) + ": ");

int choose1 = scan.nextInt();

if (choose1 > (linkedContainer.size() - 1) || choose1 < 0) {

System.out.println("Repeat enter");

} else {

System.out.println(linkedContainer.getElementByIndex(choose1));

}

break;

}

case 7:

if (linkedContainer.notEmpty()) {

linkedContainer.saveRec();

} else {

System.out.println("Your container is empty");

}

break;

case 8:

if (linkedContainer.notEmpty()) {

linkedContainer.clear();

}

linkedContainer.readRec();

break;

case 9:

if (linkedContainer.notEmpty()) {

System.out.println("What field do you want to sort by?");

System.out.println("1. Sort by products");

System.out.println("2. Sort by price");

System.out.println("3. Sort by date");

int choose2 = scan.nextInt();

boolean loop2 = true;

while (loop2) {

switch (choose2) {

case 1:

Util.sortProducts(linkedContainer);

loop2 = false;

break;

case 2:

Util.sortPrice(linkedContainer);

loop2 = false;

break;

case 3:

Util.sortDate(linkedContainer);

loop2 = false;

break;

default:

System.out.println("Wrong number!");

break;

}

}

} else {

System.out.println("Your container is empty");

}

break;

case 0:

System.out.println("Thanks for working!");

loop = false;

break;

default:

System.out.println("Wrong number!");

break;

}

}

}

}

**class** Until

package ua.khpi.oop.Ohonkova10;

import java.util.Arrays;

import ua.khpi.oop.Ohonkova10.Container;

//параметризованные методы

public class Util {

public static <T extends Store> void sortProducts(Container<T> obj) {

Store[] array = new Store[obj.size()];

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

array[i] = obj.getElementByIndex(i);

}

Arrays.sort(array, Store.compareByProducts);

obj.clear();

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

obj.add((T) array[i]);

}

}

public static <T extends Store> void sortPrice(Container<T> obj) {

Store[] array = new Store[obj.size()];

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

array[i] = obj.getElementByIndex(i);

}

Arrays.sort(array, Store.compareByPrice);

obj.clear();

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

obj.add((T) array[i]);

}

}

public static <T extends Store> void sortDate(Container<T> obj) {

Store[] array = new Store[obj.size()];

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

array[i] = obj.getElementByIndex(i);

}

Arrays.sort(array, Store.compareByDate);

obj.clear();

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

obj.add((T) array[i]);

}

}

public static void chooseMenu() {

System.out.println();

System.out.println("1. Add product.");

System.out.println("2. Show all products.");

System.out.println("3. Clear container.");

System.out.println("4. Check elements in container.");

System.out.println("5. Size of container.");

System.out.println("6. Get element by index.");

System.out.println("7. Save data to file.");

System.out.println("8. Read data from file.");

System.out.println("9. Sorting data in container.");

System.out.println("0. End of work.");

System.out.print("Write your choose there: ");

}

}

**class** Container

package ua.khpi.oop.Ohonkova10;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.PrintWriter;

import java.io.Serializable;

import java.util.Arrays;

import java.util.Iterator;

import java.util.NoSuchElementException;

import ua.khpi.oop.Ohonkova10.Store;

public class Container<T extends Store> implements Linked<T>, Serializable {

public Container() {

head = new Node<>(null, null, tail);

tail = new Node<>(null, head, null);

head = new Node<>(null, null, tail);

}

@Override

public Iterator<T> iterator() {

return new Iterator<T>() {

private int position = 0;

@Override

public boolean hasNext() {

return position < size;

}

@Override

public T next() {

if (this.hasNext()) {

return getElementByIndex(position++);

} else {

throw new NoSuchElementException();

}

}

};

}

@Override

public boolean notEmpty() {

return size > 0;

}

@Override

public void clear() {

for (Node<T> x = head; x != null; ) {

Node<T> next = x.nextElem;

x.currentElem = null;

x.nextElem = null;

x.prevElem = null;

x = next;

}

head = null;

tail = null;

tail = new Node<>(null, head, null);

head = new Node<>(null, null, tail);

size = 0;

}

@Override

public void add(final T obj) {

addLast(obj);

}

@Override

public void saveAll() {

try {

File file = new File("save.txt");

if (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = new PrintWriter(file);

System.out.println();

pw.println(size);

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

pw.println(getElementByIndex(i));

}

pw.close();

} catch (IOException e) {

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

}

}

public void saveIn(Linked<T> obj) throws IOException {

FileOutputStream file = new FileOutputStream("save.natacha");

ObjectOutputStream object = new ObjectOutputStream(file);

object.writeObject(obj);

object.close();

}

@Override

public void saveRec() {

try {

File file = new File("save.txt");

if (!file.exists()) {

file.createNewFile();

}

PrintWriter pw = new PrintWriter(file);

Store temp;

System.out.println();

pw.println(size);

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

temp = (Store) getElementByIndex(i);

pw.println(temp.getName());

pw.println(temp.getUnit());

pw.println(temp.getCount());

pw.println(temp.getUnitPrice());

pw.println(temp.getDataOfReceipt());

pw.println(temp.getAttribute());

pw.println(temp.getValue());

}

pw.close();

} catch (IOException e) {

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

}

}

@Override

public void readAll() {

try (BufferedReader br = new BufferedReader(new FileReader("save.txt"))) {

Object temp;

String line;

line = br.readLine();

int count = Integer.parseInt(line);

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

line = br.readLine();

temp = line;

add((T) temp);

}

} catch (IOException ex) {

ex.printStackTrace();

}

}

@Override

public void readRec() throws IOException {

BufferedReader br = null;

Store temp = new Store();

try {

br = new BufferedReader(new FileReader("1.txt"));

String line;

line = br.readLine();

int count = Integer.parseInt(line);

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

line = br.readLine();

temp.setName(line);

line = br.readLine();

temp.setUnit(line);

line = br.readLine();

temp.setCount(Integer.parseInt(line));

line = br.readLine();

temp.setUnitPrice(Integer.parseInt(line));

line = br.readLine();

temp.setDataOfReceipt(line);

line = br.readLine();

temp.setAttribute(line);

line = br.readLine();

temp.setValue(line);

add((T) new Store(temp));

}

} catch (IOException ex) {

ex.printStackTrace();

} finally {

br.close();

}

}

public Object[] toArray() {

Object[] result = new Object[size];

int i = 0;

for (Node<T> temp = head; i < size; temp = temp.nextElem) {

result[i++] = temp.currentElem;

}

return result;

}

@Override

public String toString() {

return Arrays.toString(toArray());

}

private Node<T> head; //первый элемент

private Node<T> tail; //последний элемент

private int size = 0; //размер списка

@Override

public void addLast(final T obj) {

Node<T> prev = tail; //сохранение данных хвоста

prev.setCurrentElem(obj); //установка значения

tail = new Node<>(null, prev, null); //изменение указателя хвоста

prev.setNextElem(tail); //установка указателя на хвост

size++; //увелечение размера списка

}

@Override

public void addFirst(final T obj) {

Node<T> next = head;

next.setCurrentElem(obj);

head = new Node<>(null, null, next);

next.setPrevElem(head);

size++;

}

@Override

public int size() {

return size;

}

@Override

public T getElementByIndex(final int index) {

Node<T> target = head.getNextElem(); //следующий элемент первого узла

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

target = getNextElement(target);

}

return target.getCurrentElem();

}

//head-null&tail-null

//null<-prevElem[head(t=null)]nextElem->&prevElem[head(t=null)]nextElem->null

private Node<T> getNextElement(final Node<T> index) {

return index.getNextElem();

}

// head -> null & tail -> null

// null <- prevElem [head(t = null)] nextElem-> & <- prevElem [head(t = null)] nextElem -> null

private class Node<T> implements Serializable {

private T currentElem; //текущий

private Node<T> prevElem; //предыдущий

private Node<T> nextElem; //следующий

//конструктор копирования

Node(final T currentElem, final Node<T> prevElem, final Node<T> nextElem) {

this.currentElem = currentElem;

this.prevElem = prevElem;

this.nextElem = nextElem;

}

public T getCurrentElem() {

return currentElem;

}

public void setCurrentElem(final T currentElem) {

this.currentElem = currentElem;

}

public Node<T> getPrevElem() {

return prevElem;

}

public void setPrevElem(final Node<T> prevElem) {

this.prevElem = prevElem;

}

public Node<T> getNextElem() {

return nextElem;

}

public void setNextElem(final Node<T> nextElem) {

this.nextElem = nextElem;

}

}

}

**class** Store

package ua.khpi.oop.Ohonkova10;

import java.io.Serializable;

import java.util.Comparator;

import java.util.Scanner;

import ua.khpi.oop.Ohonkova10.Store;

public class Store implements Serializable{

private String name;

private String unit;

private int count;

private int unit\_price;

private String data\_of\_receipt;

private String attribute;

private String value;

Store() {

name = null;

unit = null;

count = 0;

unit\_price = 0;

data\_of\_receipt = null;

attribute = null;

value = null;

}

Store (final Store obj) {

name = obj.name;

unit = obj.unit;

count = obj.count;

unit\_price = obj.unit\_price;

data\_of\_receipt = obj.data\_of\_receipt;

attribute = obj.attribute;

value = obj.value;

}

public void setName(String name) {

this.name = name;

}

public void setUnit(String unit) {

this.unit = unit;

}

public void setCount(int count) {

this.count = count;

}

public void setUnitPrice(int unit\_price) {

this.unit\_price = unit\_price;

}

public void setDataOfReceipt(String data\_of\_receipt) {

this.data\_of\_receipt = data\_of\_receipt;

}

public void setAttribute(String attribute) {

this.attribute = attribute;

}

public void setValue(String value) {

this.value = value;

}

public String getName() {

return name;

}

public String getUnit() {

return unit;

}

public int getCount() {

return count;

}

public int getUnitPrice() {

return unit\_price;

}

public String getDataOfReceipt() {

return data\_of\_receipt;

}

public String getAttribute() {

return attribute;

}

public String getValue() {

return value;

}

void generateProducts() {

Scanner scan = new Scanner(System.in);

Scanner scan2 = new Scanner(System.in);

int choose = 0;

System.out.print("Введите название товара: ");

name = scan2.nextLine();

System.out.print("Введите единицу измирения: ");

unit = scan2.nextLine();

System.out.print("Введите количество товара: ");

count = scan.nextInt();

System.out.print("Введите цену за единицу товара: ");

unit\_price = scan.nextInt();

System.out.print("Введите дату получения: ");

data\_of\_receipt = scan2.nextLine();

System.out.print("Свойства: ");

attribute = scan2.nextLine();

System.out.print("Значение: ");

value = scan2.nextLine();

}

@Override

public String toString() {

return "Name = " + name + "\n" +"Unit = " + unit +"\n" + "Count = "

+ count + "\n" +"Unit\_price = " + unit\_price + "\n" +"Data\_of\_receipt = " + data\_of\_receipt + "\n" + "Attribute = "

+ attribute + "\n" +"Value = " + value;

}

//компоратор - это функция для сравнения обьектов по определенному полю в контейнере

//используется в встроенном методе sort

public static final Comparator<Store> compareByProducts = new Comparator<Store>() {

@Override

public int compare(Store o1, Store o2) {

return o1.getName().compareTo(o2.getName());

}

};

public static final Comparator<Store> compareByPrice = new Comparator<Store>() {

@Override

public int compare(Store o1, Store o2) {

return o1.getUnitPrice() - o2.getUnitPrice();

}

};

public static final Comparator<Store> compareByDate = new Comparator<Store>() {

@Override

public int compare(Store o1, Store o2) {

return o1.getDataOfReceipt().compareTo(o2.getDataOfReceipt());

}

};

}

**class** Linked

package ua.khpi.oop.Ohonkova10;

import java.io.IOException;

import java.io.Serializable;

//создаем интерфейс

//обьявление прототипов функций

//Т обобщение

public interface Linked<T> extends Serializable, Iterable<T> {

void addLast(T obj);

void addFirst(T obj);

int size();

T getElementByIndex(int index);

void saveAll();

void saveRec();

void add(T obj);

void clear();

boolean notEmpty();

void readRec() throws IOException;

void readAll();

}

1. ВАРІАНТИ ВИКОРИСТАННЯ

Програму можна використовувати для створення контейнеру з даними, що вводить користувач, стосовно теми «магазин».

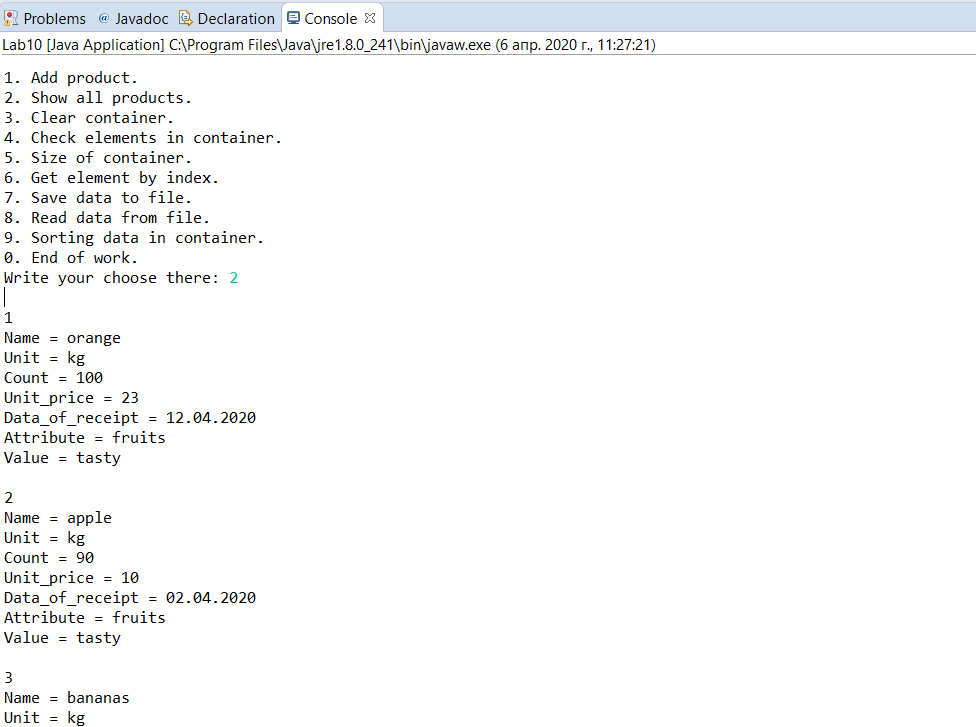


Рисунок 2 – Результат програми в діалоговому режимі

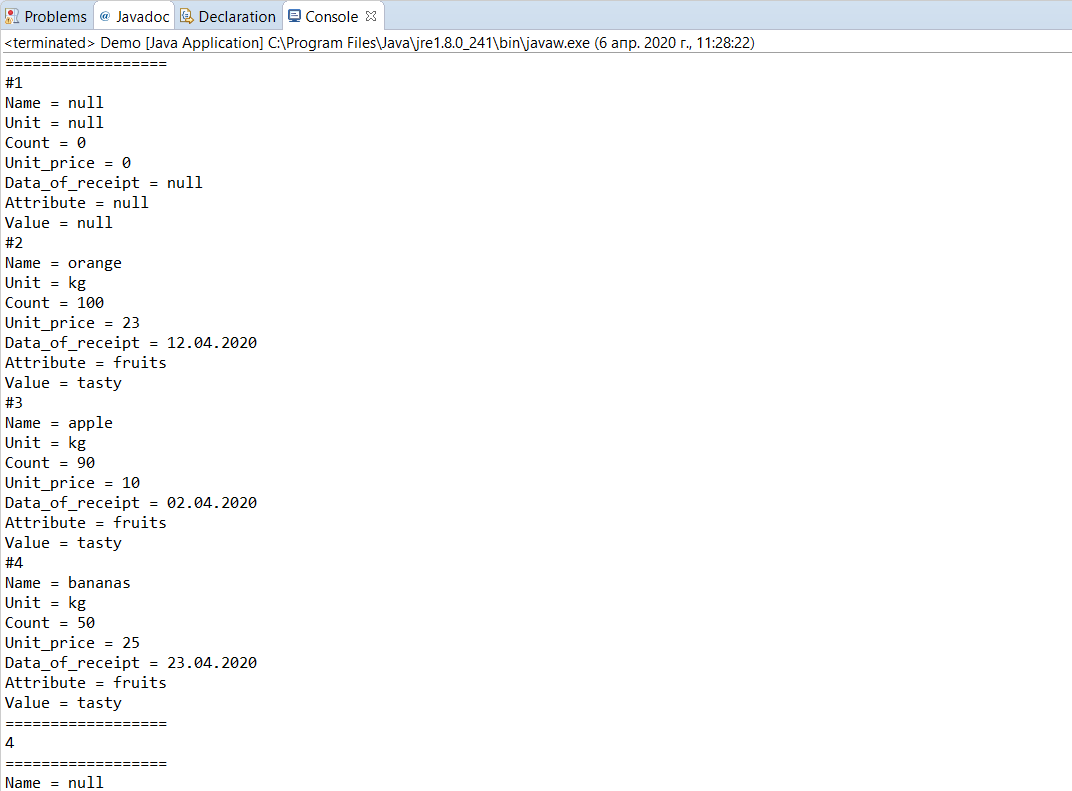


Рисунок 3 – Результат програми в автоматичному режимі

ВИСНОВОК

При виконанні лабораторної роботи набуто практичних навичок, щодо розширення функціональності параметризованих класів і методів. Програма виконується без помилок.