Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



Звіт

з лабораторної роботи № 6

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Параметризоване програмування»

**Виконав:**

студент групи КІ-306

Фодор Андрій

**Прийняв:**

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Мета роботи:** оволодіти навиками параметризованого програмування мовою Java.

**Завдання (варіант № 21)**

1. Створити параметризований клас, що реалізує предметну область задану варіантом. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

2. Автоматично згенерувати документацію до розробленого пакету.

3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

5. Дати відповідь на контрольні запитання.

Предмента область 21 варіатну – **БАК ДЛЯ СМІТТЯ**

**Вихідний код програми**

package KI306.Lab6.Fodor;

import java.util.ArrayList;

/\*\*

\* Class driver is written to demonstrate code of lab6

\*/

public class GarbageCanDriver{

public static void main(String[] args) {

GarbageCan <? super Rubbish> garbageCan = new GarbageCan<Rubbish>();

garbageCan.throwAwayGarbage(new GlassCan(30));

garbageCan.throwAwayGarbage(new PaperCan(25));

garbageCan.throwAwayGarbage(new PlasticCan(40));

garbageCan.throwAwayGarbage(new PlasticCan(15));

garbageCan.throwAwayGarbage(new PlasticCan(10));

garbageCan.throwAwayGarbage(new PaperCan(5));

System.out.println("---------------------------------------\nRECYCLING START\n\*&^$&\*#($\*$&#\*$($(\*#");

garbageCan.recycleRubbish();

System.out.println("RECYCLING END\n---------------------------------------");

int max = garbageCan.findMaxURMPack();

int [] arr = new int[2];

arr = garbageCan.findMaxTotalURM();

if(max != -1){

System.out.println("The biggest useful raw material part was got from this part of garbage:");

garbageCan.getInfo(max);

switch (arr[1]) {

case 1 -> System.out.println("The biggest useful raw material part in total was got from GLASS");

case 2 -> System.out.println("The biggest useful raw material part in total was got from PLASTIC");

case 3 -> System.out.println("The biggest useful raw material part in total was got from PAPER");

}

System.out.println("This weight is " + arr[0] + " kg");

}else{

System.out.println("There are no recycled rubbish");

}

System.out.println("\n\n\nGarbage can state: ");

garbageCan.getInfo();

}

}

/\*\*

\* Generic class GarbageCan implements garbage can

\* @param <T> is type for ArrayList, that can contains any Rubbish inherited classes

\*/

class GarbageCan <T extends Rubbish>{

private ArrayList<T> arr;

private int GlassURM = 0;

private int PaperURM = 0;

private int PlasticURM = 0;

/\*\*

\* Default constructor

\*/

public GarbageCan(){

arr = new ArrayList<T>();

}

/\*\*

\* Method is used to find index of arr element that has the biggest value of URM (useful raw materials)

\* @return index of arr element or -1 if there are no elements

\*/

public int findMaxURMPack()

{

if (!arr.isEmpty())

{

T max = arr.get(0);

int maxIdx = 0;

for (int i=1; i< arr.size(); i++)

{

if ( arr.get(i).compareTo(max) > 0 ){

max = arr.get(i);

maxIdx = i;

}

}

return maxIdx;

}

return -1;

}

/\*\*

\* Method returns the biggest value of URM in total

\* @return array, where first element is URM kg, second is value that indicate whether this URM of glass, plastic or paper

\*/

public int[] findMaxTotalURM(){

int[] resArr = new int[2];

int max = this.GlassURM;

resArr[1] = 1;

if(max < this.PlasticURM){

max = this.PlasticURM;

resArr[1] = 2;

}

else if(max < this.PaperURM){

max = this.PaperURM;

resArr[1] = 3;

}

resArr[0] = max;

return resArr;

}

/\*\*

\* Method is implements rubbish throwing to can

\* @param g is some type of rubbish, generic type

\*/

public void throwAwayGarbage(T g){

arr.add(g);

System.out.println("Rubbish thrown:");

g.getInfo();

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

}

/\*\*

\* Method implements rubbish recycling and counting of total URM

\*/

public void recycleRubbish(){

if (!arr.isEmpty()) {

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

arr.get(i).recycle();

switch (arr.get(i).getRubbishName()) {

case "GLASS" -> this.GlassURM += arr.get(i).getUsefulRawMaterial();

case "PAPER" -> this.PaperURM += arr.get(i).getUsefulRawMaterial();

case "PLASTIC" -> this.PlasticURM += arr.get(i).getUsefulRawMaterial();

}

}

}

}

/\*\*

\* Method allow to get information about any part of garbage

\* @param index is index of this part in arr

\*/

public void getInfo(int index){

if(!arr.isEmpty()){

arr.get(index).getInfo();

}else{

System.out.println("There are no element with index " + index);

}

}

/\*\*

\* Method is used to get information about all instances of three rubbish classes in arr

\*/

public void getInfo(){

if (!arr.isEmpty()) {

for (T t : arr) {

t.getInfo();

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

}

}

}

/\*\*

\* Getter for GlassURM

\* @return GlassURM

\*/

public int getGlassURM() {

return GlassURM;

}

/\*\*

\* Getter for PaperURM

\* @return PaperURM

\*/

public int getPaperURM() {

return PaperURM;

}

/\*\*

\* Getter for PlasticURM

\* @return PlasticURM

\*/

public int getPlasticURM() {

return PlasticURM;

}

}

/\*\*

\* Interface Rubbish is describing necessary methods for inherited classes and has Comparable methods

\*/

interface Rubbish extends Comparable <Rubbish>{

public double recycle();

public void getInfo();

public int getUsefulRawMaterial();

public String getRubbishName();

}

/\*\*

\* GlassCan implements Rubbish interface and glass garbage

\*/

class GlassCan implements Rubbish {

private static int ID = 1;

private final int id = ID;

private final String rubbishName = "GLASS";

private int rubbishSize;

private int usefulRawMaterial = 0;

/\*\*

\* Constructor that identifies each instance of this class and initialize weight of garbage

\* @param garbageWeight is weight of garbage for this type

\*/

public GlassCan(int garbageWeight){

ID++;

this.rubbishSize = garbageWeight;

}

/\*\*

\* Method allow to get rubbish name

\* @return name of this rubbish type

\*/

public String getRubbishName(){

return this.rubbishName;

}

/\*\*

\* Getter for rubbish size

\* @return rubbish size in kg

\*/

public int getRubbishSize(){

return this.rubbishSize;

}

/\*\*

\* Getter for URM

\* @return URM of current pert of this type garbage

\*/

public int getUsefulRawMaterial(){

return this.usefulRawMaterial;

}

/\*\*

\* method implements parent interface which allow to recycle this part of rubbish and get URM

\* @return count of URM

\*/

@Override

public double recycle() {

this.usefulRawMaterial += (int)(this.rubbishSize\*0.5);

this.rubbishSize = 0;

return this.usefulRawMaterial;

}

/\*\*

\* Method implements parent interface which allow to get information about current part of rubbish (instance of this class)

\*/

@Override

public void getInfo() {

System.out.println("ID: " + this.id +

"\nRubbish name is " + this.rubbishName +

"\nFrom this part you get " + this.usefulRawMaterial + " kg useful raw materials" +

"\nRubbish thrown here is " + this.rubbishSize + " kg");

}

/\*\*

\* Method implements Comparable interface

\* @param o the object to be compared.

\* @return value that identify comparing result

\*/

@Override

public int compareTo(Rubbish o) {

Integer s = this.usefulRawMaterial;

return s.compareTo(o.getUsefulRawMaterial());

}

}

/\*\*

\* PaperCan implements Rubbish interface and paper garbage

\*/

class PaperCan implements Rubbish {

private static int ID = 1;

private final int id = ID;

private final String rubbishName = "PAPER";

private int rubbishSize;

private int usefulRawMaterial= 0;

/\*\*

\* Constructor that identifies each instance of this class and initialize weight of garbage

\* @param garbageWeight is weight of garbage for this type

\*/

public PaperCan(int garbageWeight){

ID++;

this.rubbishSize = garbageWeight;

}

/\*\*

\* Method allow to get rubbish name

\* @return name of this rubbish type

\*/

public String getRubbishName(){

return this.rubbishName;

}

/\*\*

\* Getter for rubbish size

\* @return rubbish size in kg

\*/

public int getRubbishSize(){

return this.rubbishSize;

}

/\*\*

\* Getter for URM

\* @return URM of current pert of this type garbage

\*/

public int getUsefulRawMaterial(){

return this.usefulRawMaterial;

}

/\*\*

\* method implements parent interface which allow to recycle this part of rubbish and get URM

\* @return count of URM

\*/

@Override

public double recycle() {

this.usefulRawMaterial += (int)(this.rubbishSize\*0.2);

this.rubbishSize = 0;

return this.usefulRawMaterial;

}

/\*\*

\* Method implements parent interface which allow to get information about current part of rubbish (instance of this class)

\*/

@Override

public void getInfo() {

System.out.println("ID: " + this.id +

"\nRubbish name is " + this.rubbishName +

"\nFrom this part you get " + this.usefulRawMaterial + " kg useful raw materials" +

"\nLast rubbish thrown here was " + this.rubbishSize + " kg");

}

/\*\*

\* Method implements Comparable interface

\* @param o the object to be compared.

\* @return value that identify comparing result

\*/

@Override

public int compareTo(Rubbish o) {

Integer s = this.usefulRawMaterial;

return s.compareTo(o.getUsefulRawMaterial());

}

}

/\*\*

\* PlasticCan implements Rubbish interface and plastic garbage

\*/

class PlasticCan implements Rubbish {

private static int ID = 1;

private final int id = ID;

private final String rubbishName = "PLASTIC";

private int rubbishSize;

private int usefulRawMaterial = 0;

/\*\*

\* Constructor that identifies each instance of this class and initialize weight of garbage

\* @param garbageWeight is weight of garbage for this type

\*/

public PlasticCan(int garbageWeight){

ID++;

this.rubbishSize = garbageWeight;

}

/\*\*

\* Method allow to get rubbish name

\* @return name of this rubbish type

\*/

public String getRubbishName(){

return this.rubbishName;

}

/\*\*

\* Getter for rubbish size

\* @return rubbish size in kg

\*/

public int getRubbishSize(){

return this.rubbishSize;

}

/\*\*

\* Getter for URM

\* @return URM of current pert of this type garbage

\*/

public int getUsefulRawMaterial(){

return this.usefulRawMaterial;

}

/\*\*

\* method implements parent interface which allow to recycle this part of rubbish and get URM

\* @return count of URM

\*/

@Override

public double recycle() {

this.usefulRawMaterial += (int)(this.rubbishSize\*0.3);

this.rubbishSize = 0;

return this.usefulRawMaterial;

}

/\*\*

\* Method implements parent interface which allow to get information about current part of rubbish (instance of this class)

\*/

@Override

public void getInfo() {

System.out.println("ID: " + this.id +

"\nRubbish name is " + this.rubbishName +

"\nFrom this part you get " + this.usefulRawMaterial + " kg useful raw materials" +

"\nLast rubbish thrown here was " + this.rubbishSize + " kg");

}

/\*\*

\* Method implements Comparable interface

\* @param o the object to be compared.

\* @return value that identify comparing result

\*/

@Override

public int compareTo(Rubbish o) {

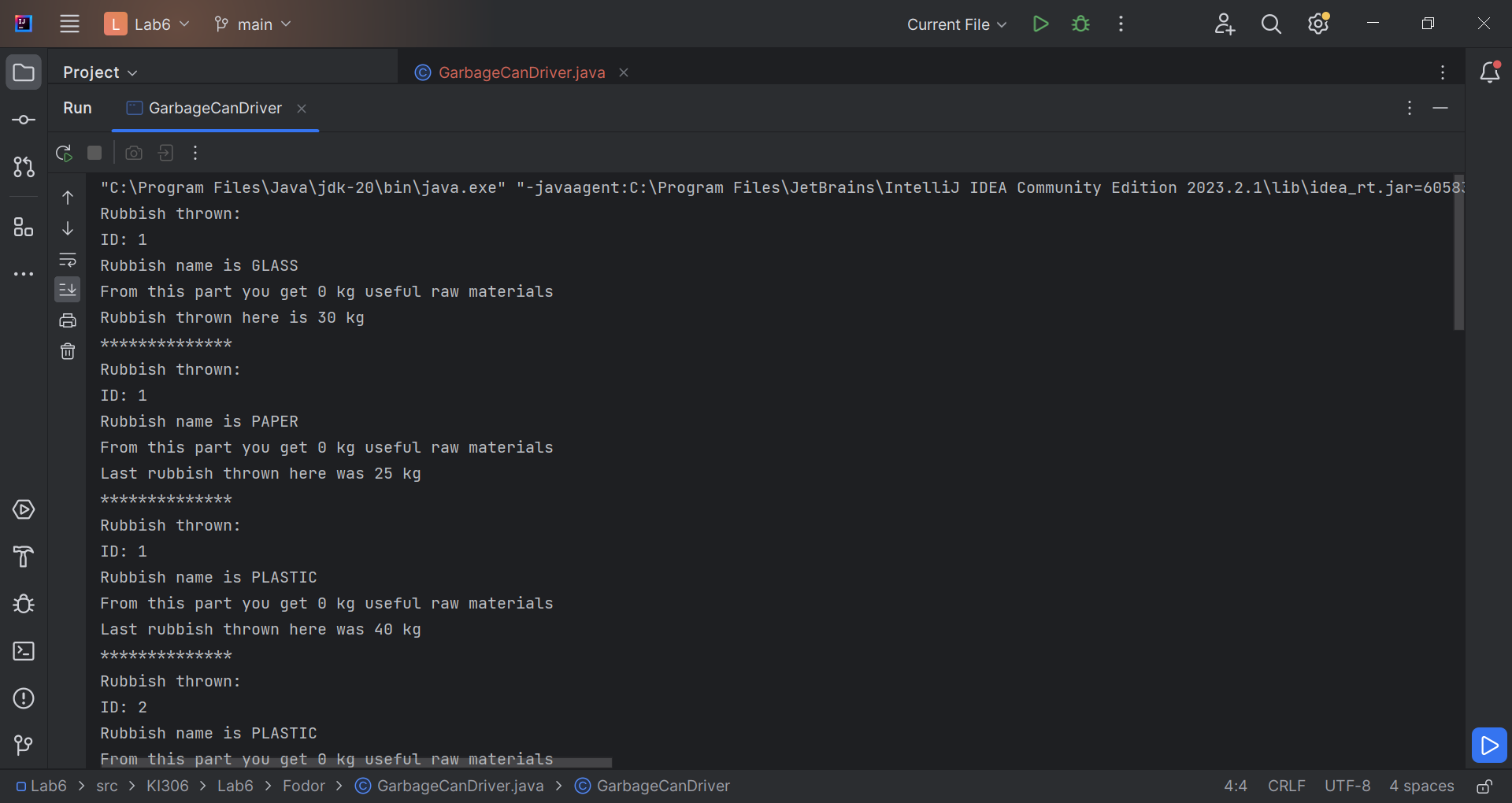
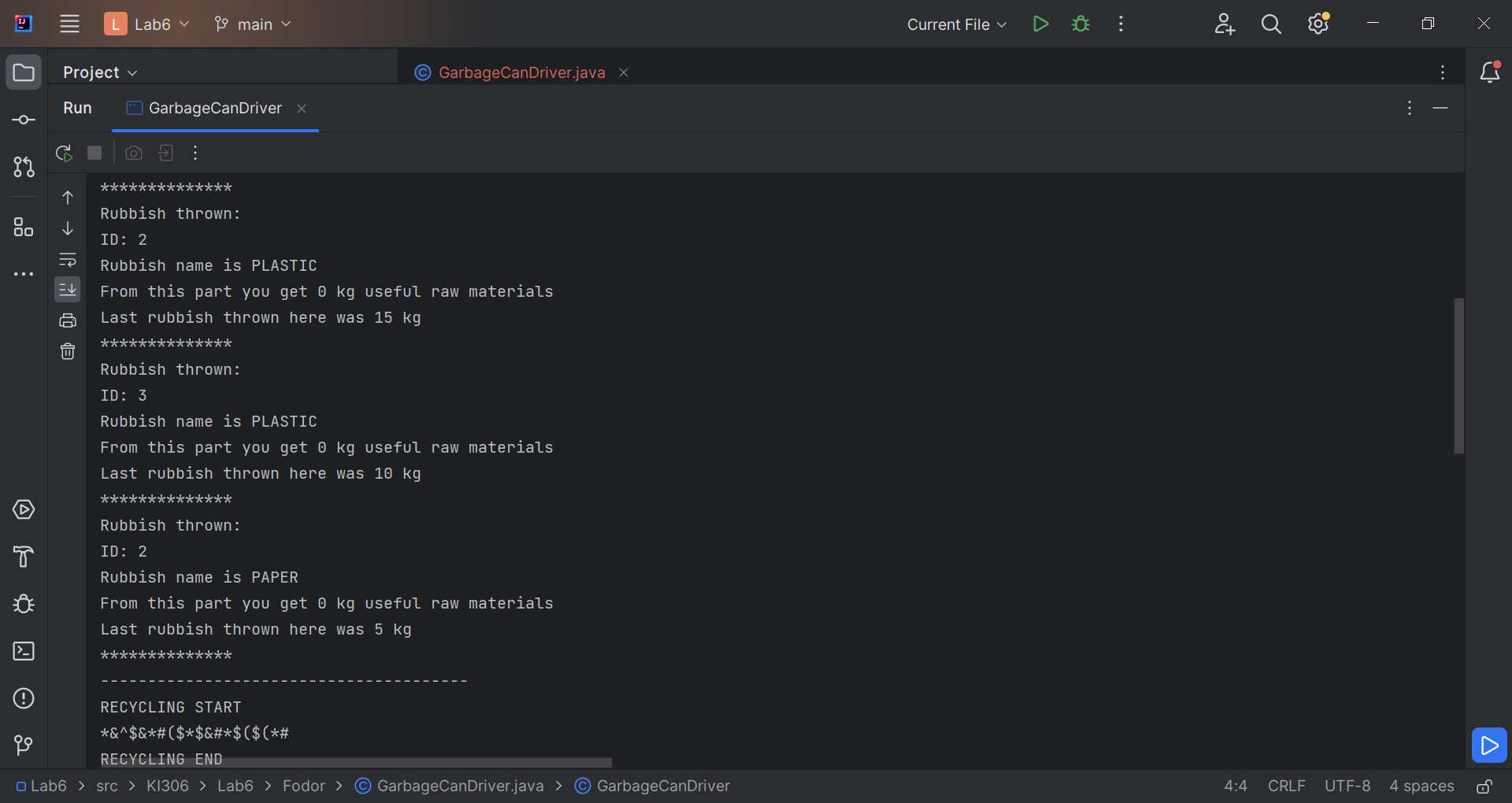
Integer s = this.usefulRawMaterial;

return s.compareTo(o.getUsefulRawMaterial());

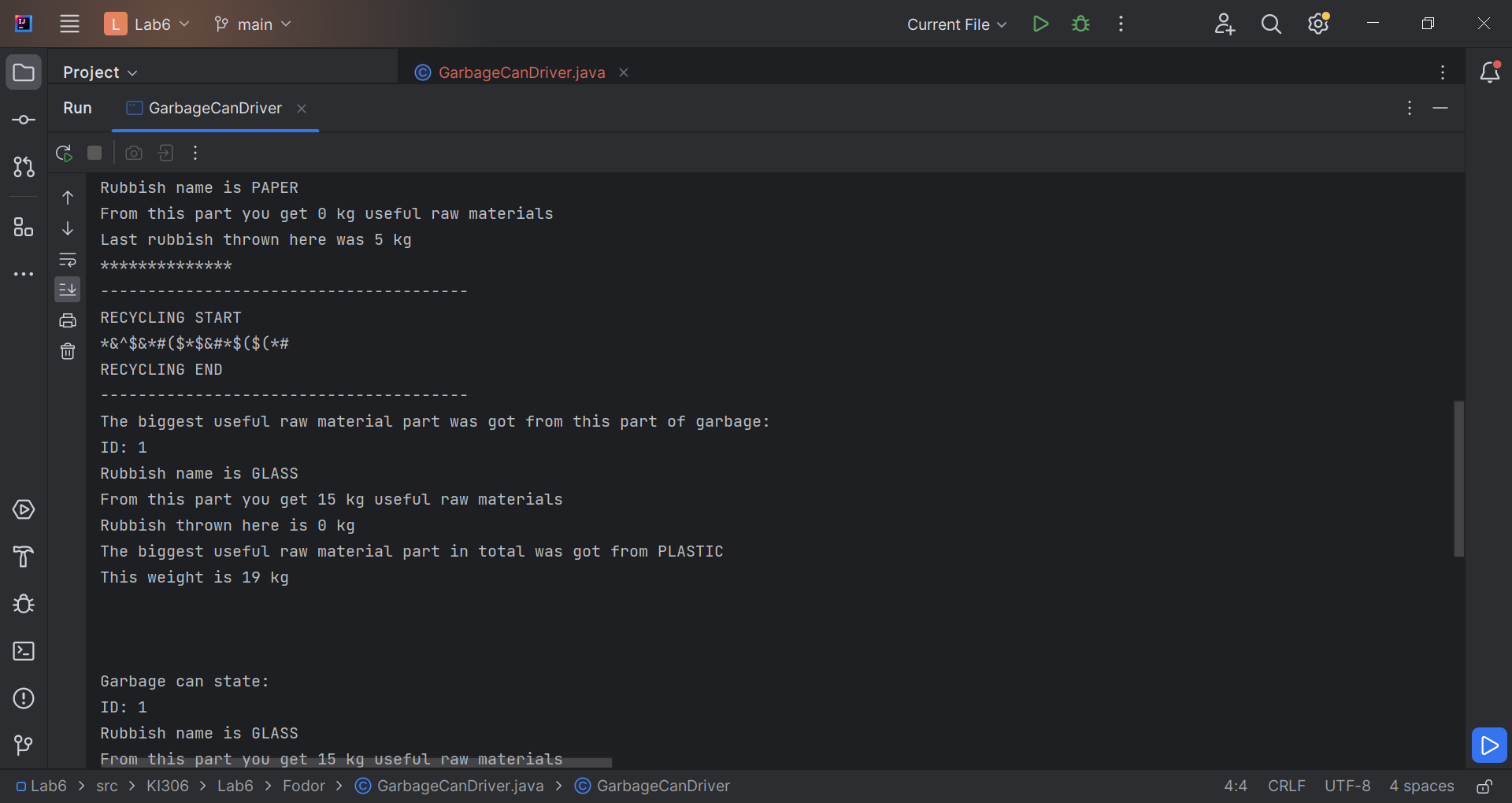
}

}

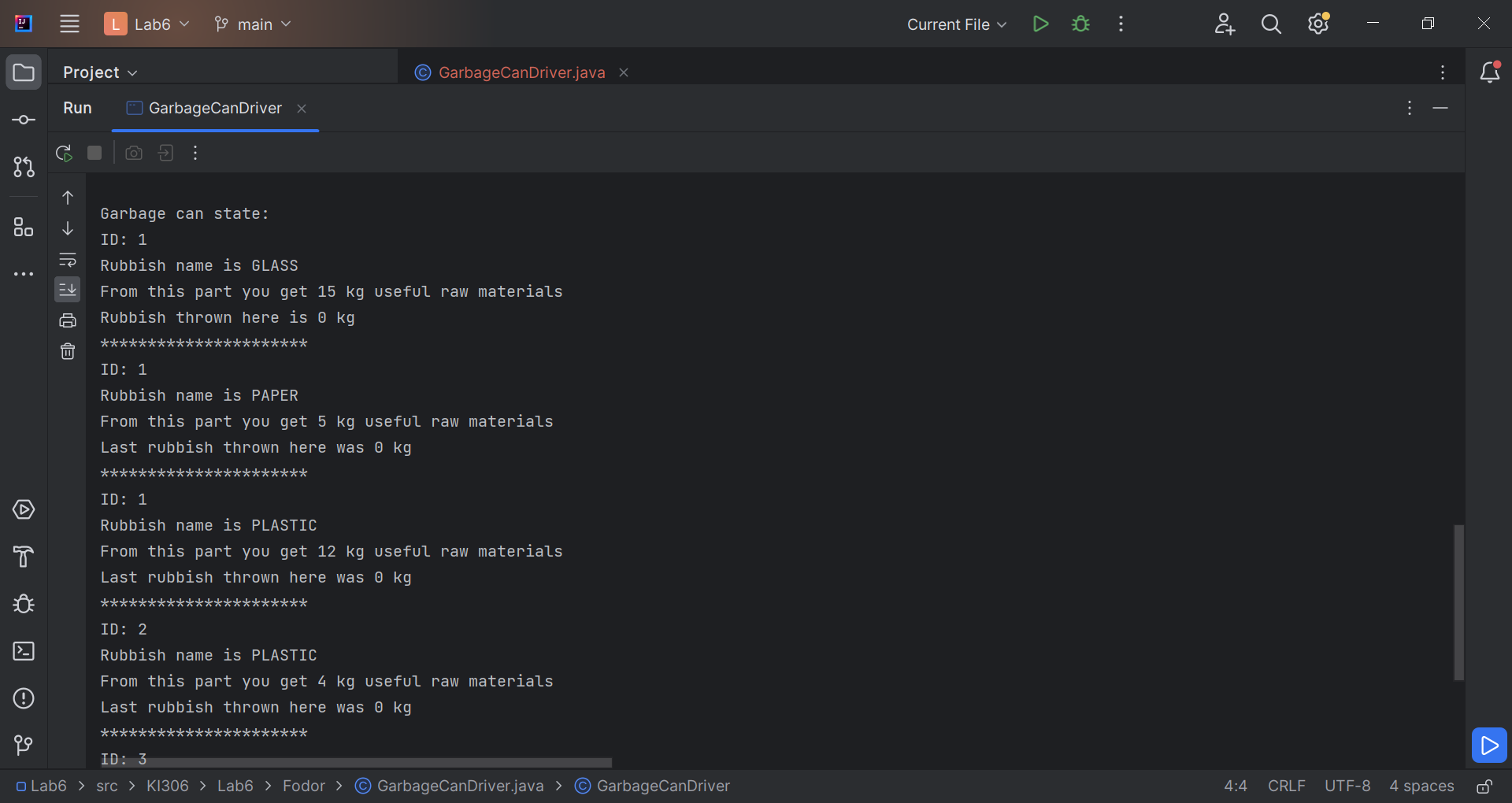
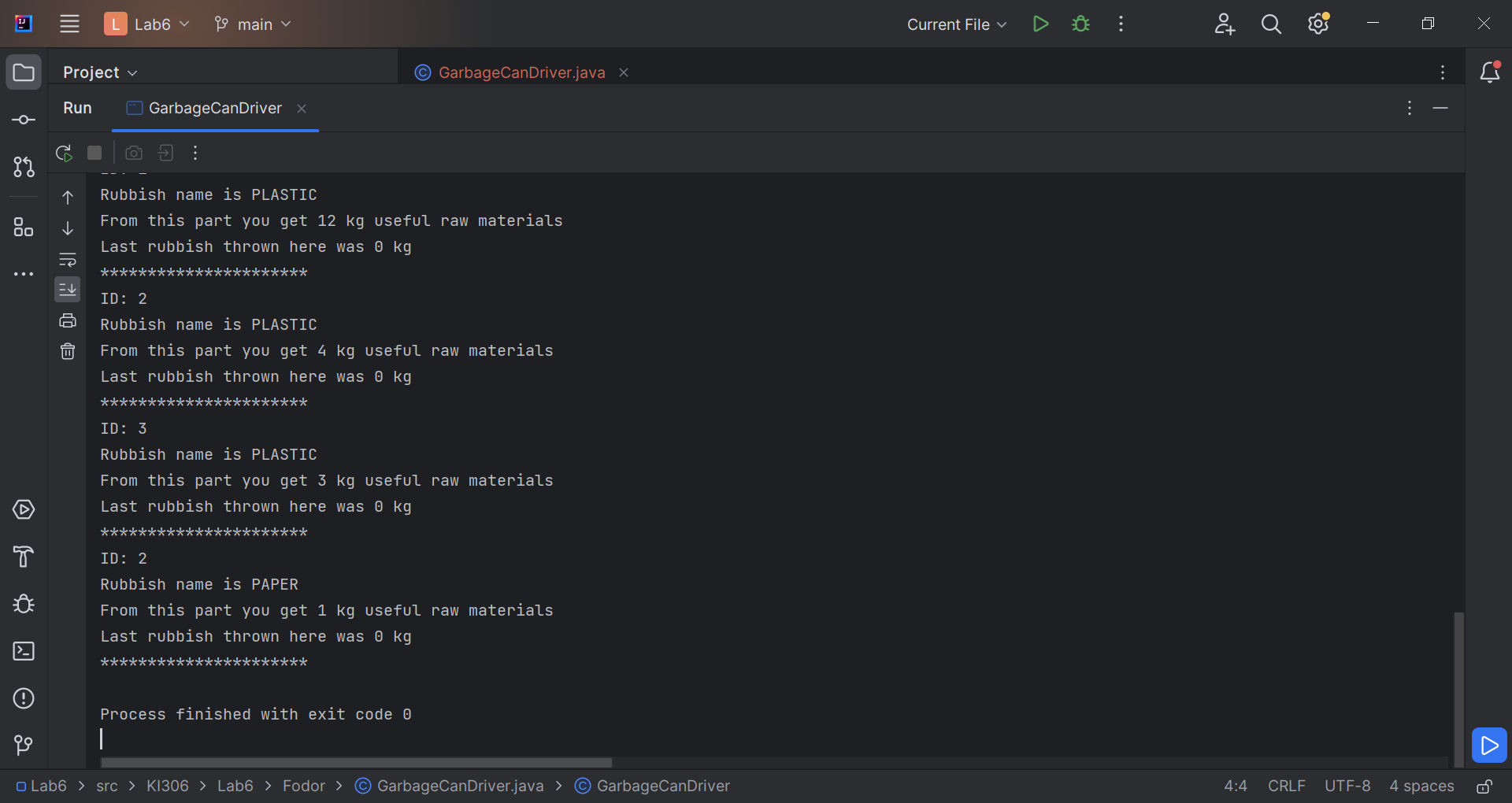
**Результат виконання програми**

На цьому етапі передані в об’єкт класу баку для сміття об’єкти різного типу сміття (трьох різних класів, що представляють паперове, пластикове та скляне сміття), були перероблені, в результаті чого отримано корисну сировину, яка використовується для порівняння об’єктів класів.

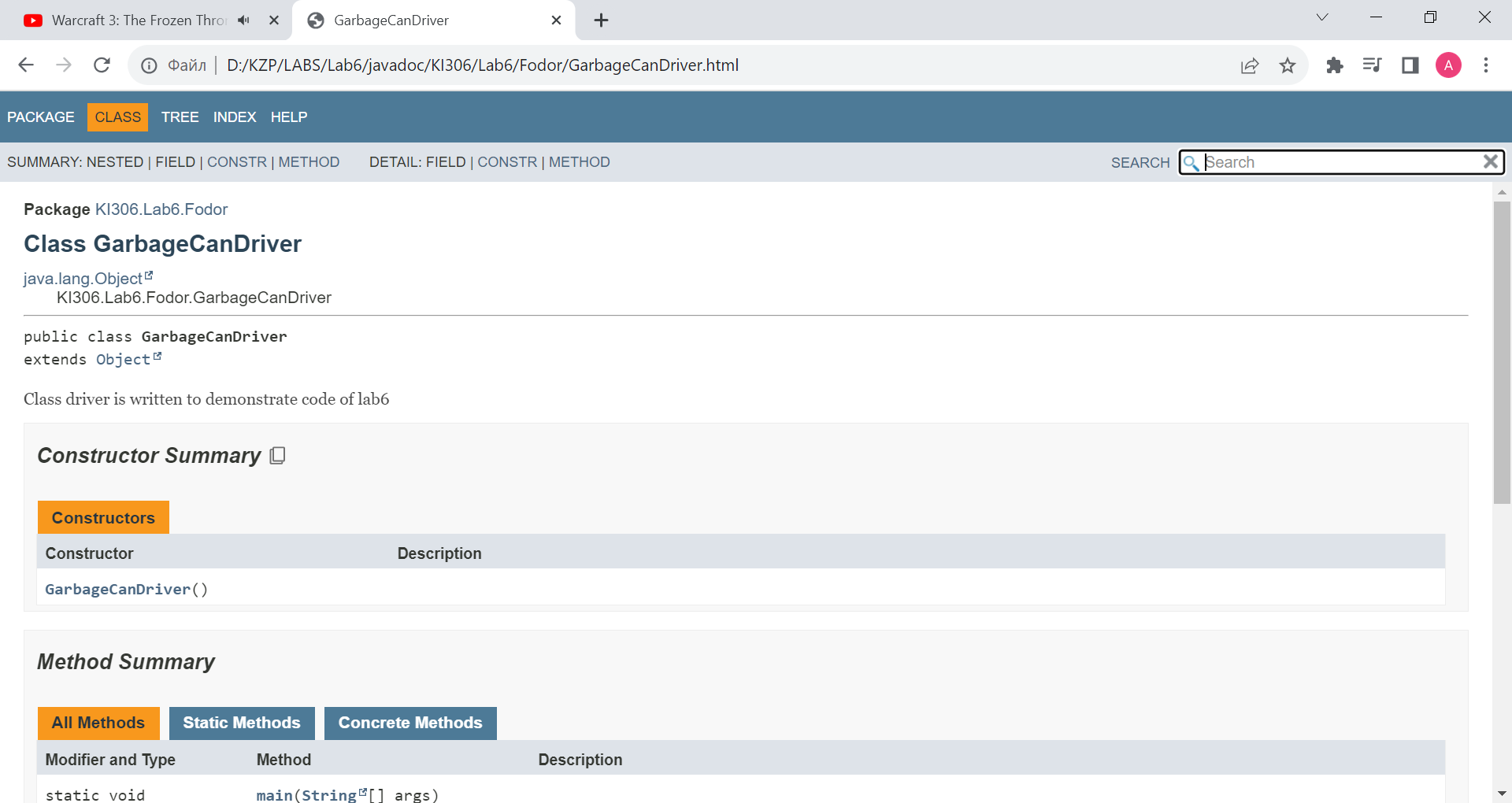


На цьому етапі, коли вже все сміття перероблено, екземпляр класу смітника може підрахувати, з якої порції сміття було отримано найбільше сировини. Також він підраховує кількість отриманої сировини кожного з трьох типів сміття в загальному. Ми можемо побачити це значення в останніх двох рядочках.

Щоб переконатися в правильності підрахунків методів класу зверху, ми можемо переглянути інформацію про кожну порцію сміття окремо і власноруч все просумувати і порівняти. Уважно перечитавши це, ми можемо з легкістю побачити, що з першої порції скляного сміття було отримано 15 кілограм корисної сировини, в той час, як з першої порції пластику було отримано 12 кг сировини. Але оскільки порції пластикового сміття було 3 по 12, 4 і 3 кг, то сумарна кількість сировини з пластику становить 19 кг, що є більше за сумарну кількість сировини зі скла (15 кг) та паперу (1 + 5 = 6 кг). Отже, можна спокійно стверджувати, що програма працює коректно і в той же час виконує всі умови завдання, зв’язаної з параметризацією, wild card та розширення можливостей generic типу Т.

**Фрагмент згенерованої документації**



**Висновок:** на цій лабораторній роботі я оволодів навиками параметризованого програмування мовою Java. Закріпив отримані знання на прикладі виконання індивідуального завдання, переконався в перевагах параметризованого програмування в мові програмування Java