Міністерство Освіти і Науки України

Національний технічний університет

«Харківський політехнічний інститут»

Кафедра «Програмної інженерії та інформаційних технологій управління»

Лабораторна робота № 1

# Робота з узагальненнями та колекціями в Java

**Виконав:**

Студент групи КН-36А

Кулик В.В.

**Перевірила:**

Козуля М.М.

Харків 2017

**Завданная**

### 1.1 Індивідуальне завдання

Розробити ієрархію класів для представлення сутностей індивідуального завдання. Базовий абстрактний клас, який представляє першу з сутностей індивідуального завдання [лабораторної роботи № 5 курсу "Алгоритмізація та програмування" (друга частина)](http://iwanoff.96.lt/algorithmization_2/LabTraining05.html), повинен містити абстрактні функції для доступу до елементів типу другого класу попередньої роботи та сортування цих елементів за певною ознакою. Похідні класи від створеного абстрактного класу повинні відповідно представляти послідовність елементів у вигляді масиву та списку. Здійснити тестування обох реалізацій. Тест повинен включати виконання завдання попередньої лабораторної роботи, а також сортування за визначеними ознаками відповідно до завдання [лабораторної роботи № 6 курсу "Алгоритмізація та програмування" (друга частина)](http://iwanoff.96.lt/algorithmization_2/LabTraining06.html).

Для сортування слід використовувати методи sort() класів Arrays та Collections відповідно. Для визначення ознак сортування використати лямбда-вирази.

### 1.2 Мінімум функції

Реалізувати програму, що дозволяє знайти мінімум деякої функції на заданому інтервалі. Алгоритм знаходження мінімуму полягає в послідовному переборі з певним кроком точок інтервалу і порівнянні значень функції в поточній точці з раніше знайденим мінімумом.

Знайти та застосувати відповідний стандартний функціональний інтерфейс для опису функції.

Реалізувати два підходи - через використання лямбда-виразів і через використання вказівників на методи.

### 1.3 Узагальнений клас

Створити узагальнений клас для зберігання довільних даних у масиві. Реалізувати функцію додавання елементу в кінець масиву, видалення елементу, додавання групи (іншого масиву) елементів.

### 1.4 Створення бібліотеки узагальнених функцій для роботи з масивами та списками

Реалізувати клас зі статичними узагальненими методами, які реалізують таку функціональність:

* обмін місцями двох груп елементів
* обмін місцями усіх пар сусідніх елементів (з парним і непарним індексом)
* заміна групи елементів іншим масивом (списком) елементів
* вставлення у масив (список) іншого масиву (списку) елементів у вказане місце

Реалізувати наведені функції для масивів і для списків. Здійснити демонстрацію роботи усіх методів з використанням даних різних типів (Integer, Double, String) .

### 1.5 Реалізація інтерфейсу Comparable

Створити клас Circle, який реалізує інтерфейс Comparable. Більшим вважається коло з більшим радіусом. Здійснити сортування списку об'єктів типу Circle.

### 1.6 Реалізація інтерфейсу Comparator

Створити клас Triangle. Трикутник визначати довжинами сторін. Площа трикутника в цьому випадку може бути обчислена за формулою Герона:

http://iwanoff.96.lt/oop_kn/Images/07_Heron.png

де *a*, *b* і *c* - довжини сторін трикутника. Здійснити сортування списку трикутників за зменшенням площі. Для визначення ознаки сортування використовувати об'єкт, який реалізує інтерфейс Comparator.

**Хід роботи**

### 1.1 Індивідуальне завдання

*Код програми 1.1:*

**BaseAbstract**

package F\_ex;

public abstract class BaseAbstract {

private String name;

private String spec;

abstract public Reception getReception(int i);

abstract public void setReception(int i, Reception recep);

abstract public int receptionsCount();

abstract public void addReception(Reception recep);

abstract public void clearReception();

abstract public void sortByVisitors();

abstract public void sortByDate();

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;}

public String getSpec() {

return spec;

}

public void setSpec(String spec) {

this.spec = spec;

}

public int allVisitors(){

int s=0;

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

s+=getReception(i).getNumOfVisit();

}

return s;

}

public int minVisitors() {

Reception visit=getReception(0);

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

if (visit.getNumOfVisit() > getReception(i).getNumOfVisit()) {

visit = getReception(i);

}

}

return visit.getNumOfVisit();

}

public int SurnameLength(){

return name.length();

}

public void showData(String title) {

System.out.println(title);

System.out.println(getName() + " " + getSpec());

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

System.out.println(getReception(i).getDay() + " " + getReception(i).getShift() + " "

+ getReception(i).getNumOfVisit());

}

System.out.println();

}

protected void test(){

setName("Курченко");

setSpec("Терапевт");

addReception(new Reception(30,"15.03.17","Утро"));

addReception(new Reception(12,"31.03.17","Обед"));

addReception(new Reception(21,"01.03.17","Вечер"));

showData("Вывод:" + '\n');

System.out.println("Минимальное количество посетителей: " + minVisitors());

System.out.println("Длинна фамиилии врача: " + SurnameLength());

sortByVisitors();

showData('\n' + "Сортировка по посетителям:");

sortByDate();

showData("Соритровка по дате:");

}

}

**CompareByDate**

**package F\_ex;**

**import java.util.Comparator;**

**public class CompareByDate implements Comparator<Reception> {**

**public int compare(Reception c1, Reception c2) {**

**return c2.getDay().compareTo(c1.getDay());**

**}**

**}**

**CompareByVisitors**

**package F\_ex;**

**import java.util.Comparator;**

**public class CompareByVisitors implements Comparator<Reception> {**

**public int compare(Reception c1, Reception c2) {**

**return Double.compare(c1.getNumOfVisit(), c2.getNumOfVisit());**

**}**

**}**

**DataOfRec**

**package F\_ex;**

**import java.util.Arrays;**

**import java.util.Comparator;**

**public class DataOfRec extends BaseAbstract{**

**private Reception [] recept=null;**

**@Override**

**public Reception getReception(int i){**

**return recept[i];**

**}**

**@Override**

**public void setReception(int i, Reception recep) {**

**recept[i]=recep;**

**}**

**@Override**

**public int receptionsCount(){**

**return recept==null?0:recept.length;**

**}**

**@Override**

**public void addReception(Reception recep){**

**Reception[] temp=new Reception[receptionsCount()+1];**

**if(recept!=null)**

**System.arraycopy(recept, 0, temp, 0, recept.length);**

**temp[receptionsCount()]=recep;**

**recept=temp;**

**}**

**@Override**

**public void clearReception(){**

**recept=null;**

**}**

**@Override**

**public void sortByVisitors(){**

**Arrays.sort(recept,new CompareByVisitors());**

**}**

**@Override**

**public void sortByDate(){**

**Arrays.sort(recept, new CompareByDate());**

**}**

**}**

**Reception**

**package F\_ex;**

**public class Reception {**

**private int numOfVisit;**

**private String day;**

**private String shift;**

**public Reception(int numOfVisit, String day, String shift) {**

**super();**

**this.numOfVisit = numOfVisit;**

**this.day = day;**

**this.shift = shift;**

**}**

**public int getNumOfVisit() {**

**return numOfVisit;**

**}**

**public void setNumOfVisit(int numOfVisit) {**

**this.numOfVisit = numOfVisit;**

**}**

**public String getDay() {**

**return day;**

**}**

**public void setDay(String day) {**

**this.day = day;**

**}**

**public String getShift() {**

**return shift;**

**}**

**public void setShift(String shift) {**

**this.shift = shift;**

**}**

**}**

**ReceptionWithList**

**package F\_ex;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.Collections;**

**import java.util.List;**

**public class ReceptionWithList extends BaseAbstract{**

**List<Reception> recept = new ArrayList<>();**

**@Override**

**public Reception getReception(int i) {**

**return recept.get(i);**

**}**

**@Override**

**public void setReception(int i, Reception recep) {**

**recept.set(i, recep);**

**}**

**@Override**

**public int receptionsCount() {**

**return recept.size();**

**}**

**@Override**

**public void addReception(Reception recep) {**

**recept.add(recep);**

**}**

**@Override**

**public void clearReception() {**

**recept = new ArrayList<Reception>();**

**}**

**@Override**

**public void sortByVisitors(){**

**Collections.sort(recept,new CompareByVisitors());**

**}**

**@Override**

**public void sortByDate(){**

**Collections.sort(recept, new CompareByDate());**

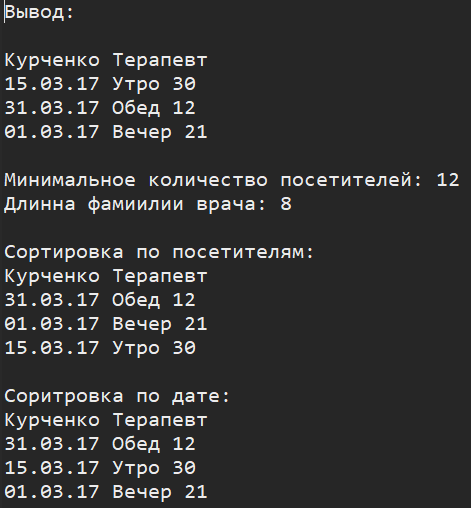
**}**

**public static void main(String[] args) {**

**new ReceptionWithList().test();**

**}**

**}**



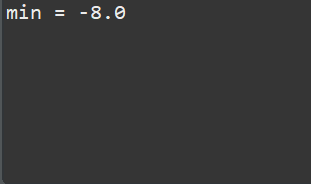
### 1.2 Мінімум функції

*Код програми 1.2:*

**Лямбда-вираз**

**package** lab1.minlamda;  
  
**public interface** MinFun {  
 **public double** f(**double** x);  
 **default double** min(**double** x) {  
 **return** f(x);  
 }  
 **static void** printTable(**double** from, **double** to, **double** step, MinFun func) {  
 **double** min = func.min(from);  
 **for** (**double** x = from + step; x <= to; x += step) {  
 **if**(min > func.min(x)){  
 min = func.min(x);  
 }  
 }  
 System.***out***.println(**"min = "** + min);  
 }  
}

**package** lab1.minlamda;  
  
**import** lab1.minlamda.MinFun;  
  
**public class** PrintWithLambda {  
 **public static void** main(String[] args) {  
 MinFun.*printTable*(-2.0, 2.0, 0.5, x -> x \* x \* x);  
 }  
}



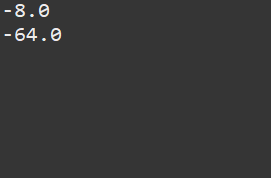
**Вказівники на методи**

**package** lab1.minvkazonmethod;  
  
**public interface** Func {  
 **double** f(**double** x);  
}

**package** lab1.minvkazonmethod;  
  
**public class** MinMethod {  
 **public double** min(**double** from, **double** to, **double** step, Func c) {  
 **if**(from <= to){  
 **double** min = c.f(from);  
 **for** (**double** x = from + step; x <= to; x += step) {  
 **if** (min > c.f(x)) {  
 min = c.f(x);  
 }  
 }  
 **return** min;  
 }  
 **else**{  
 System.***out***.println(**"Ошибка "**);  
 System.***out***.println(**"Начало отрезка меньше его конца "**);  
 **return** -1;  
 }  
 }  
  
 **public double** function(**double** x) {  
 **return** x \* x \* x;  
 }  
}

**package** lab1.minvkazonmethod;  
  
**public interface** MinOperation {  
 **double** min(**double** from, **double** to, **double** step, Func c);  
}

**package** lab1.minvkazonmethod;  
  
**public class** TestMethod {  
 **static void** print(MinOperation op, **double** from, **double** to, **double** step, Func c) {  
 System.***out***.println(op.min(from, to, step, c));  
 }  
  
 **static double** ret\_min(Func op, **double** p) {  
 **return** op.f(p);  
 }  
  
 **public static void** main(String[] args) {  
 MinMethod dm = **new** MinMethod();  
 *print*(dm::min, 2, 4,0.5, dm::function);  
 *print*(dm::min, 6, 4,0.5, dm::function);  
 }  
}

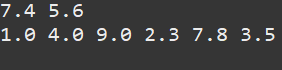


### 1.3 Узагальнений клас

*Код програми 1.3:*

**package** lab1.Array;  
  
**public class** MyArray<T> {  
 **public** Object[] **arr** = {};  
  
 **public** MyArray(T... arr) {  
 **this**.**arr** = arr;  
 }  
  
 **public** MyArray(**int** size) {  
 **arr** = **new** Object[size];  
 }  
  
 **public int** size() {  
 **return arr**.**length**;  
 }  
  
 **public** T get(**int** i) {  
 **return** (T) **arr**[i];  
 }  
  
 **public void** set(**int** i, T t) {  
 **arr**[i] = t;  
 }  
  
 **public void** add(T t) {  
 Object[] temp = **new** Object[**arr**.**length** + 1];  
 System.*arraycopy*(**arr**, 0, temp, 0, **arr**.**length**);  
 **arr** = temp;  
 **arr**[**arr**.**length** - 1] = t;  
 }  
  
 **public** <T> **void** addGroup(T[] a) {  
  
 Object[] temp = **new** Object[**arr**.**length** + a.**length**];  
 System.*arraycopy*(**arr**, 0, temp, 0, **arr**.**length**);  
 **arr** = temp;  
 **for** (**int** i = 1; i <= a.**length**; i++)  
 **arr**[**arr**.**length** - i] = a[a.**length**-i];  
  
  
 }  
  
 **public void** printAll() {  
 **for** (**int** i = 0; i < **arr**.**length**; i++) {  
 System.***out***.print(**arr**[i] + **" "**);  
 }  
 System.***out***.println();  
 }  
  
 **public void** delElem(**int** t) {  
 **arr**[t] = **null**;  
 **for** (**int** k = t; k < **arr**.**length** - 1; k++)  
 **arr**[k] = **arr**[k + 1];  
  
 Object[] temp = **new** Object[**arr**.**length** - 1];  
 System.*arraycopy*(**arr**, 0, temp, 0, **arr**.**length** - 1);  
 **arr** = temp;  
 }  
}

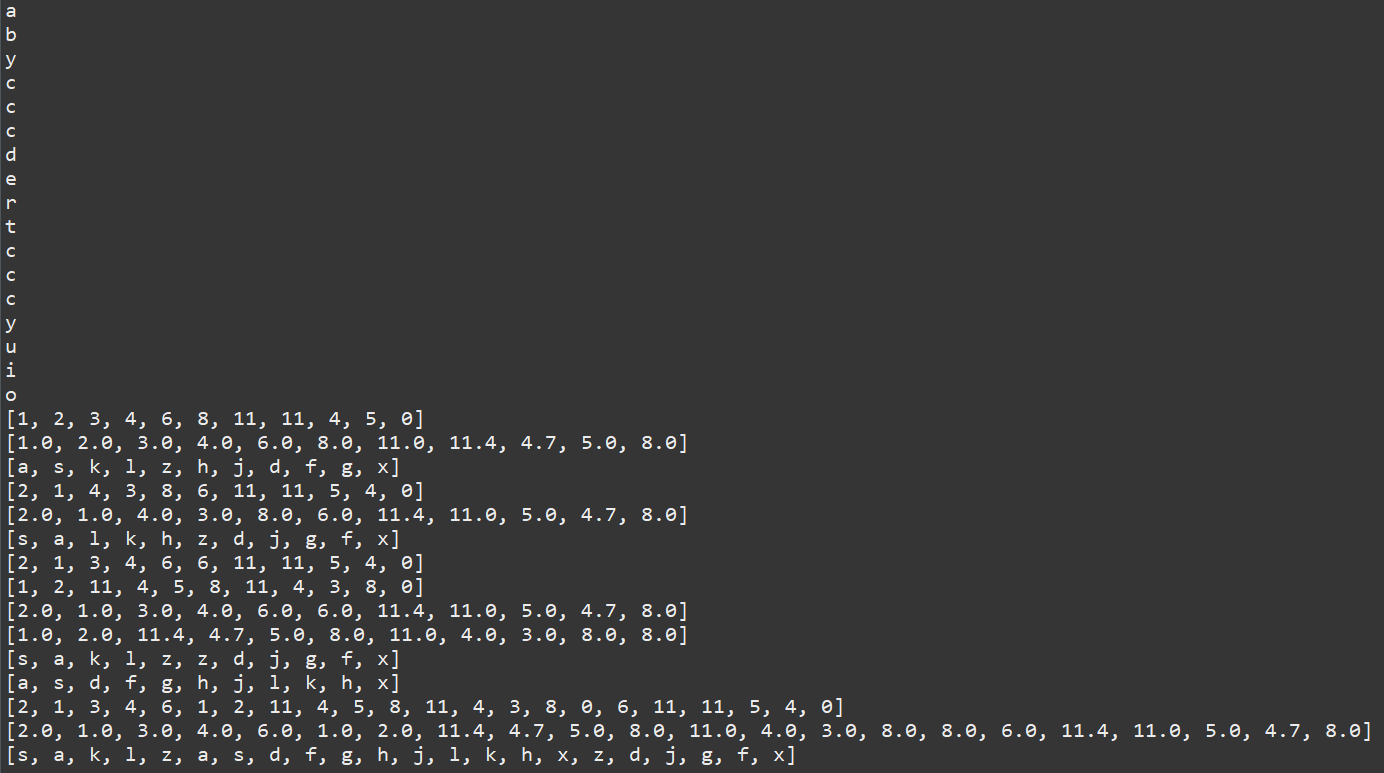
**package** lab1.Array;  
  
**public class** TestArray {  
 **public static void** main(String[] args) {  
 MyArray<Double> a = **new** MyArray<>(4.0, 5.6);  
 a.set(0, 7.4);  
 a.printAll();  
 MyArray<Double> b = **new** MyArray<>(3);  
 b.set(0, 1.0);  
 b.set(1, 2.0);  
 b.set(2, 4.0);  
 b.add(8.0);  
 b.delElem(1);  
 Double[] j = {2.5, 7.8, 3.0};  
 b.addGroup(j);  
 b.printAll();  
 }  
}



### 1.4 Створення бібліотеки узагальнених функцій для роботи з масивами та списками

*Код програми 1.4:*

**package** lab1.uzagal;  
  
**import** java.util.Arrays;  
**import** java.util.\*;  
  
**public class** UzagArray {  
 **public static** <E> **void** ZamenaGroup(List<E> arr, **int** ot1, **int** do1, **int** ot2, **int** do2) {  
 **for** (**int** i = 0; i <= do1 - ot1; i++) {  
 E temp = arr.get(ot2 + i);  
 arr.add(ot2 + i,arr.get(ot1 + i));  
 arr.add(ot1 + i,temp);  
 }  
 **for** (**int** i = 0; i < arr.size(); i++) {  
 System.***out***.println(arr.get(i));  
 }  
 }  
 **public static** <E> **void** ZamenaGroup(E[] arr, **int** ot1, **int** do1, **int** ot2, **int** do2) {  
 **for** (**int** i = 0; i <= do1 - ot1; i++) {  
 E temp = arr[ot2 + i];  
 arr[ot2 + i] = arr[ot1 + i];  
 arr[ot1 + i] = temp;  
 }  
 System.***out***.println(Arrays.*toString*(arr));  
 }  
  
 **public static** <E> **void** ParAndNepar(List<E> arr) {  
 **int** x = arr.size();  
 **if** (arr.size() % 2 != 0) {  
 x -= 1;  
 }  
 **for** (**int** i = 0; i < x; i += 2) {  
 E temp = arr.get(i);  
 arr.add(i, arr.get(i + 1));  
 arr.add(i + 1, temp);  
 }  
 **for** (**int** i = 0; i < arr.size(); i++) {  
 System.***out***.println(arr.get(i));  
 }  
 }  
 **public static** <E> **void** ParAndNepar(E[] arr) {  
 **int** x = arr.**length**;  
 **if** (arr.**length** % 2 != 0) {  
 x -= 1;  
 }  
 **for** (**int** i = 0; i < x; i += 2) {  
 E temp = arr[i];  
 arr[i] = arr[i + 1];  
 arr[i + 1] = temp;  
 }  
 System.***out***.println(Arrays.*toString*(arr));  
 }  
  
 **public static** <E> **void** ZamenaGroupArray(List<E> arr1, List<E> arr2, **int** ot1, **int** do1, **int** ot2, **int** do2) {  
 **for** (**int** i = 0; i <= do1 - ot1; i++) {  
 E temp = arr2.get(ot2 + i);  
 arr2.add(ot2 + i, arr1.get(ot1 + i));  
 arr1.add(ot1 + i, temp);  
  
 }  
 **for** (**int** i = 0; i < arr1.size(); i++) {  
 System.***out***.println(arr1.get(i));  
 }  
 **for** (**int** i = 0; i < arr2.size(); i++) {  
 System.***out***.println(arr2.get(i));  
 }  
 }  
 **public static** <E> **void** ZamenaGroupArray(E[] arr1, E[] arr2, **int** ot1, **int** do1, **int** ot2, **int** do2) {  
 **for** (**int** i = 0; i <= do1 - ot1; i++) {  
 E temp = arr2[ot2 + i];  
 arr2[ot2 + i] = arr1[ot1 + i];  
 arr1[ot1 + i] = temp;  
  
 }  
 System.***out***.println(Arrays.*toString*(arr1));  
 System.***out***.println(Arrays.*toString*(arr2));  
 }  
  
 **public static** <E> **void** InsertArray(List<E> arr1, List<E> arr2, **int** psn) {  
 List<E> temp = **new** ArrayList<E>();  
 **for** (**int** i = 0; i < psn; i++) {  
 temp.add(i, arr1.get(i));  
 }  
 **int** t = 0;  
 **for** (**int** i = psn; i < arr2.size() + psn; i++) {  
 temp.add(i, arr2.get(t));  
 t++;  
 }  
 t = psn;  
 **for** (**int** i = arr2.size() + psn; i < arr1.size() + arr2.size(); i++) {  
 temp.add(i, arr1.get(t));  
 t++;  
 }  
 arr1 = temp;  
 **for** (**int** i = 0; i < arr1.size(); i++) {  
 System.***out***.println(arr1.get(i));  
 }  
 }  
 **public static** <E> **void** InsertArray(E[] arr1, E[] arr2, **int** psn) {  
 Object[] temp = **new** Object[arr1.**length** + arr2.**length**];  
 **for** (**int** i = 0; i < psn; i++) {  
 temp[i] = arr1[i];  
 }  
 **int** t = 0;  
 **for** (**int** i = psn; i < arr2.**length** + psn; i++) {  
 temp[i] = arr2[t];  
 t++;  
 }  
 t = psn;  
 **for** (**int** i = arr2.**length** + psn; i < arr1.**length** + arr2.**length**; i++) {  
 temp[i] = arr1[t];  
 t++;  
 }  
 arr1 = (E[]) temp;  
 System.***out***.println(Arrays.*toString*(arr1));  
 }  
 **public static void** main(String[] args) {  
 Integer[] a = {1, 2, 11, 4, 5, 8, 11, 3, 4, 6, 0};  
 Integer[] d = {1, 2, 11, 4, 5, 8, 11, 3, 4, 6, 0};  
 List<String> list = **new** ArrayList<String>();  
 list.add(**"a"**);  
 list.add(**"b"**);  
 list.add(**"c"**);  
 list.add(**"d"**);  
 list.add(**"e"**);  
 list.add(**"r"**);  
 list.add(**"t"**);  
 list.add(**"y"**);  
 list.add(**"u"**);  
 list.add(**"i"**);  
 list.add(**"o"**);  
 Double[] b = {1.0, 2.0, 11.4, 4.7, 5.0, 8.0, 11.0, 3.0, 4.0, 6.0, 8.0};  
 Double[] f = {1.0, 2.0, 11.4, 4.7, 5.0, 8.0, 11.0, 3.0, 4.0, 6.0, 8.0};  
 String[] c = {**"a"**, **"s"**, **"d"**, **"f"**, **"g"**, **"h"**, **"j"**, **"k"**, **"l"**, **"z"**, **"x"**};  
 String[] e = {**"a"**, **"s"**, **"d"**, **"f"**, **"g"**, **"h"**, **"j"**, **"k"**, **"l"**, **"z"**, **"x"**};  
 *ZamenaGroup*(list, 2, 4, 7, 9);  
 *ZamenaGroup*(a, 2, 4, 7, 9);  
 *ZamenaGroup*(b, 2, 4, 7, 9);  
 *ZamenaGroup*(c, 2, 4, 7, 9);  
 *ParAndNepar*(a);  
 *ParAndNepar*(b);  
 *ParAndNepar*(c);  
 *ZamenaGroupArray*(a, d, 2, 4, 7, 9);  
 *ZamenaGroupArray*(b, f, 2, 4, 7, 9);  
 *ZamenaGroupArray*(c, e, 2, 4, 7, 9);  
 *InsertArray*(a, d, 5);  
 *InsertArray*(b, f, 5);  
 *InsertArray*(c, e, 5);  
 }  
}



### 1.5 Реалізація інтерфейсу Comparable

*Код програми 1.5:*

**package** lab1.Radius;  
  
**class** Cicle **implements** Comparable<Cicle>{  
  
 **private int radius**;  
  
 **public** Cicle(**int** radius) {  
 **super**();  
 **this**.**radius** = radius;  
 }  
  
 **public int** getRadius() {  
 **return radius**;  
 }  
 **public void** setRadius(**int** radius) {  
 **this**.**radius** = radius;  
 }  
  
 **public int** compareTo(Cicle compareCircle) {  
  
 **int** compareQuantity = ((Cicle) compareCircle).getRadius();  
  
 **return** compareQuantity - **this**.**radius**;  
  
 }  
}

**package** lab1.Radius;  
  
**import** lab1.Array.MyArray;  
  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.Collections;  
**import** java.util.List;  
  
**public class** Test {  
 **public static void** main(String args[]) {  
 List<Cicle> al = **new** ArrayList<Cicle>();  
 Cicle nameArray[] = {  
 **new** Cicle(8),  
 **new** Cicle(13),  
 **new** Cicle(5),  
 **new** Cicle(6),  
 };  
 List<Cicle> names = Arrays.*asList*(nameArray);  
 Collections.*sort*(names);  
 System.***out***.println(names.get(0).getRadius());  
 System.***out***.println(names.get(1).getRadius());  
 System.***out***.println(names.get(2).getRadius());  
 System.***out***.println(names.get(3).getRadius());  
  
  
 }  
}

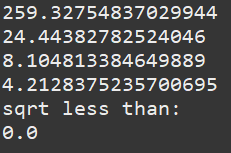


### 1.6 Реалізація інтерфейсу Comparator

*Код програми 1.6:*

**package** lab1.Plosha;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Collections;  
  
**public class** Test {  
 **public static void** main(String[] args) {  
 List<Triangle> list = **new** ArrayList<Triangle>();  
 Triangle a = **new** Triangle(5.3, 2.2, 4.1);  
 list.add(a);  
 list.add(**new** Triangle(7.9, 2.9, 6.2));  
 list.add(**new** Triangle(10.1, 5.3, 9.3));  
 list.add(**new** Triangle(17.5, 37.0, 30.0));  
 list.add(**new** Triangle(1.6, 7.4, 9.2));  
  
 Collections.*sort*(list,**new** Triangle());  
 System.***out***.println(list.get(0).Geron());  
 System.***out***.println(list.get(1).Geron());  
 System.***out***.println(list.get(2).Geron());  
 System.***out***.println(list.get(3).Geron());  
 System.***out***.println(list.get(4).Geron());  
  
  
 }  
}

**package** lab1.Plosha;  
  
**import** java.util.Comparator;  
  
**public class** Triangle **implements** Comparator<Triangle> {  
 **private double a**;  
 **private double b**;  
 **private double c**;  
 Triangle(){  
  
 }  
 **public** Triangle(**double** a, **double** b, **double** c) {  
 **this**.**a** = a;  
 **this**.**b** = b;  
 **this**.**c** = c;  
 }  
  
 **public double** getA() {  
 **return a**;  
 }  
  
 **public void** setA(**double** a) {  
 **this**.**a** = a;  
 }  
  
 **public double** getB() {  
 **return b**;  
 }  
  
 **public void** setB(**double** b) {  
 **this**.**b** = b;  
 }  
  
 **public double** getC() {  
 **return c**;  
 }  
  
 **public void** setC(**double** c) {  
 **this**.**c** = c;  
 }  
  
 **public double** Geron(){  
 **if**((**a**+**b**+**c**)\*(**b**+**c**-**a**)\*(**a**+**c**-**b**)\*(**a**+**b**-**c**)>= 0){  
 **return** 0.25 \* Math.*sqrt*((**a**+**b**+**c**)\*(**b**+**c**-**a**)\*(**a**+**c**-**b**)\*(**a**+**b**-**c**));  
 }  
 **else** {  
 System.***out***.println(**"sqrt less than: "**);  
 **return** 0;  
 }  
 }  
  
 **public double** Geron2(){  
 **if**((**a**+**b**+**c**)\*(**b**+**c**-**a**)\*(**a**+**c**-**b**)\*(**a**+**b**-**c**)>= 0){  
 **return** 0.25 \* Math.*sqrt*((**a**+**b**+**c**)\*(**b**+**c**-**a**)\*(**a**+**c**-**b**)\*(**a**+**b**-**c**));  
 }  
 **else** {  
 **return** -1;  
 }  
 }  
  
 @Override  
 **public int** compare(Triangle o1, Triangle o2) {  
 Double p1 = ((Triangle) o1).Geron2();  
 Double p2 = ((Triangle) o2).Geron2();  
 **if** (p1 < p2) {  
 **return** 1;  
 } **else if** (p1 > p2){  
 **return** -1;  
 } **else** {  
 **return** 0;  
 }  
 }  
}



***Висновок:***

У даній роботі я за допомогою мови програмування Java та використовуючи попередній досвід я розробив різного роду програми, під час розробки яких я навчився працювати з узагальненнями та колекціями в Java.