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

Антоненко В.В.

**Перевірив:**

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)
* вставлення у масив (список) іншого масиву (списку) елементів у вказане місце
* заміна групи елементів іншим масивом (списком) елементів

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:*

**День**

**package** lab1.individual;  
  
**import** java.util.StringTokenizer;  
  
**public class** Day **implements** Comparable<Day> {  
 **private** String **data**;  
 **private double temperature**;  
 **private** String **comments**;  
  
 **public** Day(String data, **double** temperature, String comments) {  
 **this**.**data** = data;  
 **this**.**temperature** = temperature;  
 **this**.**comments** = comments;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Дата: "** + getData() + **"\nТемпература равна "** + getTemperature() +  
 **" градусов\nКомментарий: "** + getComments();  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (**this** == obj) {  
 **return true**;  
 }  
 **if** (obj == **null** || !(obj **instanceof** Day)) {  
 **return false**;  
 }  
 Day c = (Day) obj;  
 **return** c.getData() == getData() &&  
 c.getTemperature() == getTemperature() &&  
 c.getComments().equals(getComments());  
 }  
  
 **public** String getComments() {  
 **return comments**;  
 }  
  
 **public void** setComments(String comments) {  
 **this**.**comments** = comments;  
 }  
  
 **public double** getTemperature() {  
 **return temperature**;  
 }  
  
 **public void** setTemperature(**double** temperature) {  
 **this**.**temperature** = temperature;  
 }  
  
 **public** String getData() {  
 **return data**;  
 }  
  
 **public void** setDatd(String data) {  
 **this**.**data** = data;  
 }  
  
  
 **public int** kolvosim() {  
 **int** m = **comments**.length();  
 **return** m;  
 }  
  
 **public boolean** containsWord(String word) {  
 StringTokenizer st = **new** StringTokenizer(getComments());  
 String s;  
 **while** (st.hasMoreTokens()) {  
 s = st.nextToken();  
 **if** (s.equalsIgnoreCase(word)) {  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 **public boolean** containsSubstring(String substring) {  
 **return** getComments().toUpperCase().indexOf(substring.toUpperCase()) >= 0;  
 }  
  
 **private void** testWord(String word) {  
 **if** (containsWord(word)) {  
 System.***out***.println(**"Слово \""** + word + **"\" міститься у коментарі"**);  
 } **else** {  
 System.***out***.println(**"Слово \""** + word + **"\" не міститься у коментарі"**);  
 }  
 **if** (containsSubstring(word)) {  
 System.***out***.println(**"Текст \""** + word + **"\" міститься у коментарі"**);  
 } **else** {  
 System.***out***.println(**"Текст \""** + word + **"\" не міститься у коментарі"**);  
 }  
 }  
  
 @Override  
 **public int** compareTo(Day c) {  
 **return** Double.*compare*(getTemperature(), c.getTemperature());  
 }  
  
 **public static void** main(String[] args) {  
 Day days = **new** Day(**"20.05.17"**, 22.5,  
 **"Облачно с прояснениями"**);  
 days.testWord(**"Облачно"**);  
 days.testWord(**"Облако"**);  
 days.testWord(**"Ясно"**); }}

**Погода**

**package** lab1.individual;  
  
**import** java.util.Arrays;  
**import** java.util.Comparator;  
  
  
  
**public abstract class** AbstractPogoda {  
 **private** String **sezon**;  
 **private** String **com**;  
  
 **abstract public void** addDays(Day day);  
 **abstract public void** sortByTemp();  
 **abstract public void** sortByComments();  
 **abstract public void** clearDay();  
 **abstract public int** daysCount();  
 **abstract public** Day getDays(**int** index);  
 **abstract public void** setDay(**int** i, Day day);  
  
  
 **public** String getCom() {  
 **return com**;  
 }  
  
 **public void** setCom(String com) {  
 **this**.**com** = com;  
 }  
  
 **public** String getSezon() {  
 **return sezon**;  
 }  
  
 **public void** setSezon(String sezon) {  
 **this**.**sezon** = sezon;  
 }  
  
 **public double** sered\_temp() {  
 **double** sum = 0;  
 **for** (**int** i = 0; i < daysCount(); i++) {  
 sum += getDays(i).getTemperature();  
 }  
 **return** sum / daysCount();  
 }  
  
  
 **public** String maxTemp() {  
 Day max = getDays(0);  
 **for** (**int** i = 1; i < daysCount(); i++) {  
 **if** (max.getTemperature() < getDays(i).getTemperature()) {  
 max = getDays(i);  
 }  
 }  
 **return** max.getData();  
 }  
  
 **public** String maxComment() {  
 Day max = getDays(0);  
 **for** (**int** i = 1; i < daysCount(); i++) {  
 **if** (max.kolvosim() < getDays(i).kolvosim()) {  
 max = getDays(i);  
 }  
 }  
 **return** max.getData();  
 }  
  
 **public void** findWord(String word) {  
 System.***out***.println();  
 System.***out***.println(**"Слово \""** + word + **"\":"**);  
 **for** (**int** i = 0; i < daysCount(); i++) {  
 **if** (getDays(i).containsWord(word)) {  
 System.***out***.println(**"Дата: "** + getDays(i).getData() + **"\nТемпература равна "** + getDays(i).getTemperature() +  
 **" градусов\nКомментарий: "** + getDays(i).getComments());  
 }  
 }  
 System.***out***.println();  
 }  
  
 **public void** showData(String title) {  
 System.***out***.println(title);  
 System.***out***.println(getSezon() + **" "** + getCom());  
 **for** (**int** i = 0; i < daysCount(); i++) {  
 System.***out***.println(getDays(i).getData() + **" "** + getDays(i).getTemperature() + **" "** + sered\_temp() + **" "** + getDays(i).getComments());  
 }  
 System.***out***.println();  
 }  
  
  
 **protected void** test() {  
 setSezon(**"Лето"**);  
 setCom(**"Жаркое лето с малым количеством осадков"**);  
 addDays(**new** Day(**"20.06.2017"**, 29.2, **"Облачно с прояснениями"**));  
 addDays(**new** Day(**"21.06.2017"**, 34.3, **"Ясно"**));  
 addDays(**new** Day(**"22.06.2017"**, 33.0, **"Ясно"**));  
 addDays(**new** Day(**"23.06.2017"**, 29.1, **"Ясно, возможен дождь"**));  
 addDays(**new** Day(**"24.06.2017"**, 30.6, **"Ясно"**));  
 showData(**"Вихідні дані:"**);  
 System.***out***.println(**"Дата с самым большим комментарием: "** + maxComment());  
 System.***out***.println(**"Дата с самой большой температурой: "** + maxTemp());  
 System.***out***.println(**"Средняя температура состовляет: "** + sered\_temp() + **" градусов"**);  
 findWord(**"дождь"**);  
 sortByTemp();  
 showData(**"\nСортировка за температурой: "**);  
  
 sortByComments();  
 showData(**"\nСортування за алфавітом коментарів:"**);  
 }  
}

**package** lab1.individual;  
  
**import** java.util.Arrays;  
**import** java.util.Comparator;  
  
**public class** PogodaWithArray **extends** AbstractPogoda {  
 **private** Day[] **days** = **null**;  
  
  
 @Override  
 **public void** addDays(Day day) {  
 Day[] temp = **new** Day[daysCount() + 1];  
 **if** (**days** != **null**)  
 System.*arraycopy*(**days**, 0, temp, 0, **days**.**length**);  
 temp[daysCount()] = day;  
 **days** = temp;  
  
 }  
  
 @Override  
 **public void** sortByTemp() {  
 Arrays.*sort*(**days**, **new** CompareByTemp());  
  
 }  
  
 @Override  
 **public void** sortByComments() {  
 Arrays.*sort*(**days**, **new** CompareByComments());  
 }  
  
 @Override  
 **public void** clearDay() {  
 **days** = **null**;  
 }  
  
 @Override  
 **public int** daysCount() {  
 **return days** == **null** ? 0 : **days**.**length**;  
 }  
  
 @Override  
 **public** Day getDays(**int** index) {  
 **return days**[index];  
 }  
  
 @Override  
 **public void** setDay(**int** i, Day day) {  
 **days**[i] = day;  
 }  
  
 **public static void** main(String[] args) {  
 **new** PogodaWithArray().test();  
 }  
  
}  
  
**class** CompareByTemp **implements** Comparator<Day> {  
  
 **public int** compare(Day c1, Day c2) {  
 **return** Double.*compare*(c1.getTemperature(), c2.getTemperature());  
 }  
  
}  
  
**class** CompareByComments **implements** Comparator<Day> {  
  
 **public int** compare(Day c1, Day c2) {  
 **return** c1.getComments().compareTo(c2.getComments());  
 }  
  
}

**package** lab1.individual;  
  
**import** java.util.ArrayList;  
**import** java.util.Collections;  
**import** java.util.List;  
  
**public class** PogodaWithList **extends** AbstractPogoda {  
 List<Day> **days** = **new** ArrayList<>();  
  
  
 @Override  
 **public void** addDays(Day day) {  
 **days**.add(day);  
 }  
  
 @Override  
 **public void** sortByTemp() {  
 Collections.*sort*(**days**, **new** CompareByTemp());  
  
 }  
  
 @Override  
 **public void** sortByComments() {  
 Collections.*sort*(**days**, **new** CompareByComments());  
 }  
  
 @Override  
 **public void** clearDay() {  
 **days** = **new** ArrayList<Day>();  
 }  
  
 @Override  
 **public int** daysCount() {  
 **return days**.size();  
 }  
  
 @Override  
 **public** Day getDays(**int** index) {  
 **return days**.get(index);  
 }  
  
 @Override  
 **public void** setDay(**int** i, Day day) {  
 **days**.set(i, day);  
 }  
  
 **public static void** main(String[] args) {  
 **new** PogodaWithArray().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.