МАСИВИ И КОНТЕЙНЕРИ

гл.ас. д-р Мария Евтимова

https://github.com/marias83837/JavaPresentations

Масив

- начин на съхраняване на група от данни от един и същ тип, като позволява произволен достъп до последователност от обекти
- масивите са обекти могат да бъдат присвоени на променливи от тип Object
- всички методи на класа Object могат да бъдат извикани за масив

Декларация на масив

<тип > []<идентификатор>

<тип ><идентификатор>[]

Примери:

int [] array; - масив от примитиви

Triangle [] triangles;- масив от обекти

Инициализация на масив

- при създаване на обект- масив неговите референции автоматично се инициализират с NULL

Примери:

array= new int [10];

triangles= new Triangles [3];

Инициализиране на масив със списък от стойности

- int[] intList = {2, 3, 6, 7, 10, 13};

  double[] dList = {13.12, 0.12, 45.36};

  String[] sList = {“Ivan", “Petar", “Asen"};

- int[][] array2 = {{1, 6, 3}, {4, 5, 8}};

Пример

```
public class ArrayExample {

    public static void main(String[] args) {
        int[] Array = {1, 2, 3, 4};

        //извеждане на всички елементи от масива
        for (int i = 0; i < Array.length; i++) {
            System.out.println(Array[i] + " ");
        }

        // сума на елементите на масива
        double totalEl = 0;
        for (int i = 0; i < Array.length; i++) {
            totalEl += Array[i];
        }
        System.out.println("Сума " + totalEl);

        // най- големия елемент от масива
        double maxValue = Array[0];
        for (int i = 1; i < Array.length; i++) {
            if (Array[i] > maxValue) maxValue = Array[i];
        }
        System.out.println("Максимум " + maxValue);
    }
}
```

Резултат

```
1
2
3
4
Сума 10.0
Максимум 4.0
```

Примери за многомерни масиви for()

```
public class TwoDArray {
    public static void main(String[] args) {

        int[][] array2D = {
                {1, -5, 3},
                {3, 2, -7, 4},
                {9},
        };

        for (int i = 0; i < array2D.length; i++) {
            System.out.println("дължина на реда: " + array2D[i].length);
            for(int j = 0; j < array2D[i].length; j++) {

                System.out.println(array2D[i][j]);
            }

        }
    }
}
```

```
дължина на реда: 3
1
-5
3
дължина на реда: 4
3
2
-7
4
дължина на реда: 1
9
```

Примери за многомерни масиви for...each

```java
public class TwoDArray {
   public static void main(String[] args) {

      int[][] array2D = {
            {1, -5, 3},
            {3, 2, -7, 4},
            {9},
      };
      for (int[] innerArray: array2D) {
         System.out.println("дължина на реда: " + innerArray.length);
        for(int data: innerArray) {
           System.out.println(data);


        }

     }
   }
}
```

```
дължина на реда: 3
1
-5
3
дължина на реда: 4
3
2
-7
4
дължина на реда: 1
9
```

масивите се представят чрез обекти, които представляват масиви от други масиви.

Пример с тримерен масив

```
public class ThreeDArray {
   public static void main(String[] args) {


      int[][][] array3D = {
                  {
                   {1, -2, 3},
                   {3, 4, 5}
                  },
                  {
                   {-4, -5, 6, 8},
                   {9},
                   {2, 3}
                  }
      };


      for (int[][] array2D: array3D) {
         for (int[] innerArray: array2D) {
            for(int data: innerArray) {
               System.out.println(data);
            }
         }
      }
   }
}
```

```
1
-2
3
3
4
5
-4
-5
6
8
9
2
3
```

Пример с масив от обекти

```
public class DayGen {
    private int day,month,year;
    public DayGen (int d, int m, int y){
        this.day = d;
        this.month=m;
        this.year = y;
    }
    public String toString(){
        return "("+day+","+month+","+year+")";
    }

    public static void main(String arg[]){
        DayGen random[][]=new DayGen[3][4];
        init(random);
        for(DayGen line[]:random ){
            for (DayGen k : line){
                System.out.print("\t"+k);
            }
            System.out.println();
        }
    }
    public static void init(DayGen tb[][]){
        for(int i=0;i<tb.length;i++){
            for(int j =0; j<tb[i].length;j++){
                tb[i][j]=new DayGen((int)(Math.random()*7+1),
                        (int)(Math.random()*30+1),
                        (int)(Math.random()*100+1930));
            }
        }
    }
}
```

```
(6,19,1957)     (1,24,2022)     (4,22,2023)     (4,27,2026)
(3,8,1975)      (1,26,1969)     (4,7,1988)      (1,27,1930)
(7,28,1963)     (7,3,1949)      (1,16,1985)     (6,30,1953)
```

Методи за обработка на масиви java.util.Arrays

- **За сортиране на масив във възходящ ред**

public static void sort(Object[] obj)

public static void sort(<тип>[] data)

public static void sort(<тип>[]a, int fromIndex, int toIndex)

- **Търси в определен сортиран масив за зададена стойност (двойчен алгоритъм)**

public static int binarySearch(Object[] a, Object key)

public static int binarySearch(<тип>[] a, int fromIndex, int toIndex, <тип> key)

връща стойност >=0 , при откриване на търсения елемент

Методи за обработка на масиви java.util.Arrays

- **Сравняване на масиви- връща true, ако двата зададени масива имат еднакъв брой елементи**

public static boolean equals(<тип>[] a, <тип>[] a2)

- **Запълване на елементите на масив- присвоява зададена стойност на всеки елемент от зададения масив**

public static void fill(<тип>[] a, <тип> val)

public static void fill(<тип>[]a, int fromIndex, int toIndex,<тип>val)

Пример за сортиране

```
  public static void main(String[] args)
  {
    // създаване на масив
    String[] a = {"Ivan", "Petar", "Atanas", "Kiril"};

    // сортиране на масива
    Arrays.sort(a);
    System.out.println("Сортиран масив");
    // извеждане на резултата
    for (String element : a)
    {
      System.out.println(element);
    }
  }
}
```

```
Сортиран масив
Atanas
Ivan
Kiril
Petar
```

Пример за двойчно търсене

```
public static void binarySearch(int a[], int first, int last, int key)
  int middle = (first + last)/2;
 while( first <= last ){
      if ( a[middle] < key ){
        first = middle + 1;
      }else if ( a[middle] == key ){
        System.out.println("Елемента е намерен на индекс: " + middle);
        break;
      }else{
         last = middle - 1;
      }
      middle = (first + last)/2;
   }
   if ( first > last ){
      System.out.println("Елемента не е намерен");
   }
 }
 public static void main(String args[]){
       int a[] = {10,20,30,40,50};
        int key = 30;
        int last=a.length-1;
        binarySearch(a,0,last,key);
 }
}
```

```
run:
Елемента е намерен на индекс: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Шаблони в Java
Generics

- това са класове, интерфейси и методи, които имат типизирани параметри за един или повече типове, които се съхраняват или използват
- Формалните и типизираните параметри осигуряват многократно използване на даден код с различен вход
- входът при формалните параметри се състои от стойности
- входът при типизираните параметри се състои от типове

Преимущества

- прехвърля отговорността за сигурността на типа от потребителя към компилатора
- отпада необходимостта от писане на код за проверка на коректността на типа на данните, защото се прилага по време на компилация;
- отпада необходимостта от принудително преобразуване на типа и се намалява възможността за възникване на грешки по време на изпълнение;
- позволява на програмистите да реализират генетични алгоритми

```
public class Generic<T>
{
private T field;
public void setField(T value) {
field=value;
}
}
Generic<String> genstr=new Generic<String>();
```

или

```
Generic<String> genstr=new Generic<>();
genstr.setField(“Низ”);
Generic<Integer> genint=new Generic<Integer>();
```

или

```
Generic<Integer> genint=new Generic<>();
genint.setField(123);
```

Ограничения

?- заместващ символ за типизирания параметър

- ограничение отгоре

<? extends A> Типизирания параметър е от тип А или наследник на типа А

- ограничение отдолу

<? Super A> Типизирания параметър е от тип А или супер клас на типа А

- неизвестно ограничение

<?> Типизирания параметър е от неизвестен тип

Контейнери в Java

Контейнера е обект, който групира множество елементи в едно цяло

Приложение:

за съхранение и обработка на агрегатни данни (телефонен бележник- съответствие между имена и телефонни номера), папка с писма, ръка в игра на карти- колекция от карти).

https://docs.oracle.com/javase/6/docs/technotes/guides/collections/reference.html

Архитектура за обработка на колекции

- **интерфейси**- абстрактни типове данни, които представляват колекции

- **реализации**- конкретни реализации на интерфейсите (структури от данни)

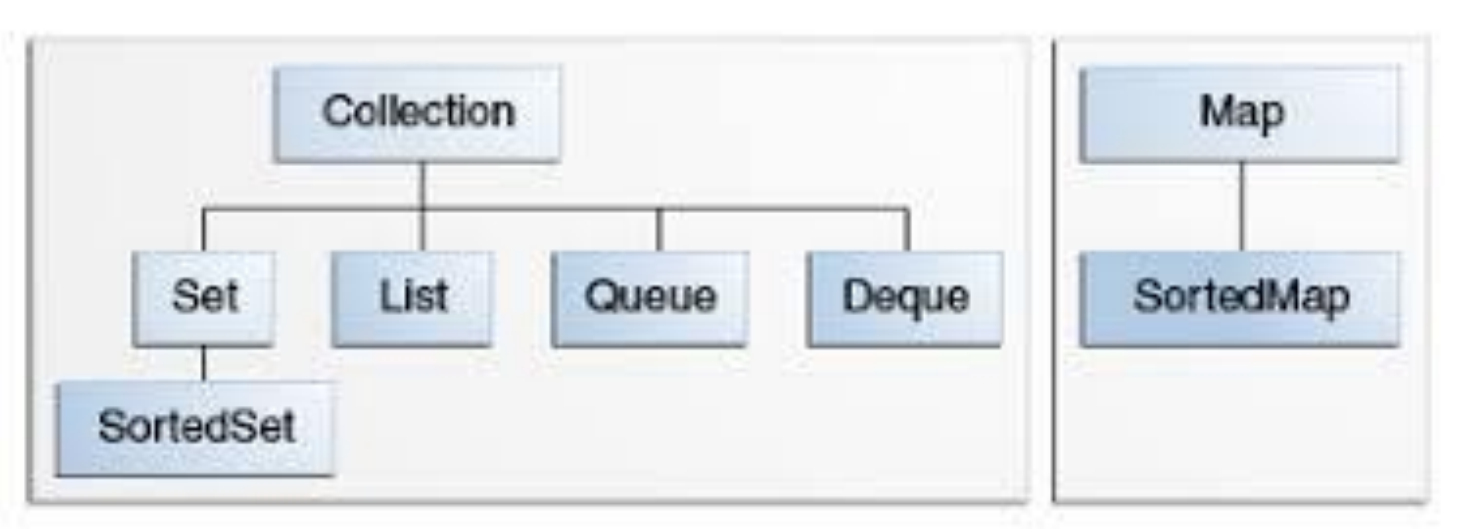
- **алгоритми**- методи, които изпълняват операции като търсене и сортиране за обекти, използващи много различни реализации на даден интерфейс на колекциите.

Наричат се полиморфни алгоритми

Предимства

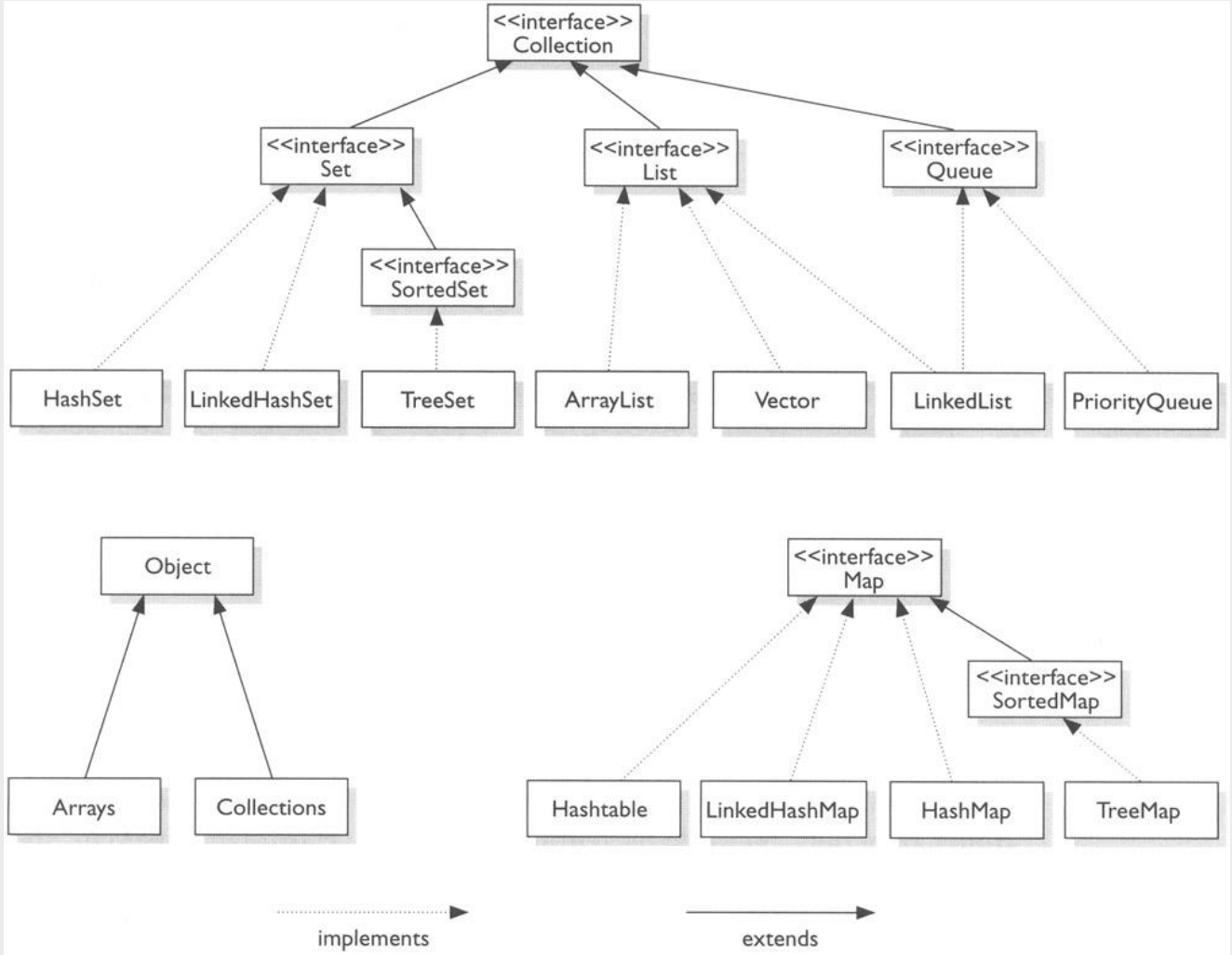
- намалява усилията на програмистите- използват структури от данни и алгоритми
- увеличава скоростта и качеството на програмите
- позволява взаимна работа на несвързани интерфейси
- намалява усилията за изучаване и използване на нови интерфейси
- намалява усилието за проектиране на нови интерфейси
- подпомага многократно използване на софтуера- създават се нови алгоритми, които оперират върху обекти, които реализират тези интерфейси

Интерфейси- два основни интерфейса за всички видове колекции в Java



-Collection<E>
-MAP<K,V>

Класове реализиращи интерфейсите в Java



- ArrayList- за произволен достъп
- LinkedList- ако често вмъквате и изтривате елементи от средата на списъка
- Map- начин на асоцииране на обекти с други обекти
- HAshMap- бърз достъп
- Set- приема само по един от всеки тип обект
- HashSet- осигурява максимално бързо търсене
- TreeSet-поддържа елементите сортирани

Vector

Vector е синхронизиран динамично нарастващ масив от обекти с ефективен достъп по индекс

⇔ ArrayList- несинхронизиран

```
Vector<Integer> vec= new Vector<Integer>(10/*начален капацитет*/);
vec.add(5);
```

ArrayList

ArrayList е несинхронизиран динамично нарастващ масив от обекти с ефективен достъп по индекс

ArrayList<Integer>arr=new ArrayList<Integer>(10/*начален капацитет*/)

arr.add(5);

Колекция java.util.ArrayList<E>- реализира интерфейса List<E>

public class **ArrayList<E>**
extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, Serializable

//Създава празен списък с начален размер 10

public ArrayList()

//Създава празен списък с начален размер initialCapacity

public ArrayList(int initialCapacity)

//Връща броя на елементите в списъка

public int size()

//Добавя елемента в края на списъка

public boolean add(E e)

//Премахва елемента в определена позиция index от списъка

public E remove(int index)

//Премахва първото срещане на обекта от списъка

public boolean remove(Object o)

//Премахва всички елементи от списъка

public void clear()

//Връща итератор за итериране на елементите в списъка

public Iterator<E>iterator()

//Връща списъчен итератор на елементите в списъка

public ListIterator<E>listIterator()

//Връща последователен поток Stream за колекцията като неин източниик

public Stream<E>stream()

//Търси обект в списъка

public boolean contains(Object o)

//Връща индекса на първото срещане на дадения обект в списъка или -1 в противен случай

public int indexOf(Object o)

//Връща елемент, намиращ се в позиция index на списъка

public E get(int index)

//Замества елемента в позиция index на списъка с element

public E set(int index, E element)

//Сортира списъка според реда, определен чрез дадения компаратор с

public void sort(Comparator<?super E> c)

HashMap е несинхронизирана по ключ- стойност хеш-таблица

```
HashMap<String, Person>idPerson;
.....
Person p = idPerson.get(“8404128990”);
if(p!=null){
System.out.println(“found:”+p);
}
```

Пример

```
HashMap<String, Integer> frequency(String[] names) {
	HashMap<String, Integer> frequency =
				new HashMap<String, Integer>();
	for(String name : names) {
		Integer currentCount = frequency.get(name);
		if(currentCount == null) {
			currentCount = 0; // auto-boxing
		}
		frequency.put(name, ++currentCount);
	}
	return frequency;
}
```

Пример

```
public static void main(String[] args) {
      System.out.println(
            frequency(new String[]{
                  "Maria", “Ivan", “Petar", “Vasil", “Gosho",
“Silvia"
            }).toString());
}
```

HashMap- не гарантира подредба при извеждане на резултата

Класа функционира подходящо в HashMap - методите equals и hashCode

```
public class Person {
	public String name;
	boolean equals(Object o) {
		return (o instanceof Person &&
					((Person)o).name.equals(name));
	}
	public int hashCode() {
		return name.hashCode();
	}
}
```

equals()- прави проверка, дали вашия ключ е еквивалентен на някой от ключовете в таблицата

hashCode()- за генериране на кода на всеки обект (по подразбиране използва адреса на своя обект)

Обхождане на колекции чрез итератор

Итератора е обект, който се използва за обхождане на колекции и селективно премахване на елементи от колекцията. Използва се метода iterator()

public interface Iterator<E>{

boolean hasNext();//връща true, ако има още елементи

E next() ;//връща следващия елемент

void remove();//премахва, последния елемент, върнат от next()

}

Приложение

- вместо for…..each
- премахване на текущия елемент
- итериране на множество колекции паралелно

Пример:

```
import java.util.*;
public class IteratorExample {
  public static void main(String args[]) {
    // Създаване на списък
    ArrayList list = new ArrayList();
         // add elements to the array list
list.add("Petar");
list.add("Maria");
list.add("Silvia");
list.add("Ivan");
  // Използване на итератор за извеждане на
съдържанието на списъка
System.out.print("Извеждане на списъка: ");
Iterator itr = list.iterator();
            while(itr.hasNext()) {
       Object element = itr.next();
System.out.print(element + " ");      }
System.out.println();
// Модифицирания обект се итерира
      ListIterator litr = list.listIterator();
while(litr.hasNext()) {
    Object element = litr.next();
            litr.set(element + "*");     }
System.out.print("Модифициране на списъка list ");
itr = list.iterator();
           while(itr.hasNext()) {
    Object element = itr.next();
System.out.print(element + " ");      }
System.out.println();
   //Извеждане на обърнатия списък
System.out.print("Обърнат модифициран списък: ");
while(litr.hasPrevious()) {
   Object element = litr.previous();
System.out.print(element + " ");      }
System.out.println();
   }}
```

Резултат

```
Извеждане на списъка: Petar Maria Silvia Ivan
Модифициране на списъка list Petar* Maria* Silvia* Ivan*
Обърнат модифициран списък: Ivan* Silvia* Maria* Petar*
```

Обхождане на колекции чрез агрегиращи операции

Предпочитан метод за итериране на колекции е да се получи поток чрез метода stream() и да се изпълнят агрегиращи операции с ламбда израз

```
System.out.println("\n Списък");

list.stream().forEach(element->System.out.println(element));

//Сортира елементите в списъка

System.out.println("Сортиране");

Collections.sort(list);

list.stream().forEach(element->System.out.println(element));
```

Списък:
Petar
Maria
Silvia
Ivan

Сортиране:
Ivan
Silvia
Maria
Petar

```
//Изтрива Мария от списъка
list.remove(“Мария”);
list.stream().forEach(element->System.out.println(element);
//Изтрива всички елементи от списъка
list.clear();
```

Извеждане на списъка:
Ivan
Silvia
Petar

Сортиране на елементи от колекция- класа Collections

- **Comparable-** позволява търсене базирано на един елемент (естествена последователност на елементите)

public interface Comparable<T>

int compare(T o)- сравнява този обект със специфичен обект //връща <0,=0 или >0

- **Comparator-** позволява търсене базирано на няколко елемента (последователността определена от определен компаратор)

public interface Comparator<T>

int compare(T o1, T o2)- сравнява два аргумента за подредба //връща <0,=0 или >0

boolean equals(Object obj)- определя дали има друг обект еднакъв със сравнявания

Пример с Comparable

```
class Student implements Comparable<Student>{
int id;
String name;
int mark;
Student(int id,String name,int mark){
this.id=id;
this.name=name;
this.mark=mark;  }
public int compareTo(Student st){
if(mark==st.mark)
return 0;
else if(mark>st.mark)
return 1;
else
return -1;
}  }
//Създаване на клас за сортиране на елементи
public class ComparableSorting{
public static void main(String args[]){
ArrayList<Student> list=new ArrayList<Student>();
list.add(new Student(108,"Viktor",5));
list.add(new Student(110,"Ana",6));
list.add(new Student(109,"Joan",4));
 Collections.sort(list);
for(Student st:list){
System.out.println(st.id+" "+st.name+" "+st.mark);
}}}
```

Резултат:

```
109 Joan  4
108 Viktor  5
110 Ana  6
```

Пример с Comparator

```
class Student{
int id;
String name;
int mark;
Student(int id,String name,int mark){
this.id=id;
this.name=name;
this.mark=mark;
}
}

class MarkComparator implements Comparator<Student>{
public int compare(Student s1,Student s2){
if(s1.mark==s2.mark)
return 0;
else if(s1.mark>s2.mark)
return 1;
else
return -1;
}
}
class NameComparator implements Comparator<Student>{
public int compare(Student s1,Student s2){
return s1.name.compareTo(s2.name);
}
}
```

```
class ExampleComparator{
public static void main(String args[]){
//Създаване на списък със студенти
ArrayList<Student> list=new ArrayList<Student>();
list.add(new Student(108,"Viktor",5));
list.add(new Student(110,"Ana",6));
list.add(new Student(109,"Joan",4));



System.out.println("Sorting by Name");
//Използване на  NameComparator за сортиране на елементите
Collections.sort(list, new NameComparator());



//Извеждане на списъка
for(Student st: list){
System.out.println(st.id+" "+st.name+" "+st.mark);
}
  System.out.println(“Сортиране по успех");
//Използване на  MarkComparator за сортиране на елементи
Collections.sort(list, new MarkComparator());
//Извеждане на списъка
for(Student st: list){
System.out.println(st.id+" "+st.name+" "+st.mark);
}
  }
}
```

Резултат:

```
Сортиране по име:
110 Ana 6
109 Joan 4
108 Viktor 5
Сортиране по успех:
109 Joan  4
108 Viktor  5
110 Ana  6
```

Търсене в колекции

- Търсене на даден елемент в колекция, използвайки алгоритъма за двоично търсене и естествена последователност на елементите ва колекцията- елементите трябва да реализират интерфейса Comparable, списъкът трябва да бъде сортиран в нарастващ ред – чрез метода sort(List)

static <T> int binarySearch(List<? extends Comparable<? super T>>list, T key)

- Търсене на даден елемент в колекция, използвайки алгоритъм за двоично търсене според последователнотта, определена чрез определен компаратор- елементите трябва да реализират интерфейса Comparator, списъкът трябва да бъде сортиран в нарастващ ред (чрез метода sort(List, Comparator)).

public static <T> int binarySearch(List<? Extends T> list, T key, Comparator<? Super T> c)