Ricerca e ordinamento su array di oggetti

Corso di Programmazione 2

Esercitazione 5

Sommario

- Ricercare in array di oggetti
  - Interfaccia comparable
- Ordinare array di oggetti

Problema

- Come ordinare, ricercare array di oggetti?

Ordinare un array Integer []

Definiamo la classe OrdinaArrayObject che include i metodi:

static int bubblesort(Integer A[]);
static void inizializzaCasuale(Integer A[]);
static void stampaArray(Integer A[]);
public static void main(String args[]);

Cosa cambia?

Ordinare un array String[]

Aggiungere nella classe OrdinaArrayObject i metodi:
static int bubblesort(String A[]);
static void inizializzaCasuale(String A[]);
static void stampaArray(String A[]);

Modificare
public static void main(String args[]);

Generazione casuale di una stringa

```
public String randomString(int length) {
  String ssource =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ" +
"abcdefghijklmnopqrstuvwxyz";
char[] src = ssource.toCharArray();
char[] buf = new char[length];
for(int i = 0; i < length; i++)
   buf[i] = src[new
     Random().nextInt(src.length)];
return new String(buf);
}
```

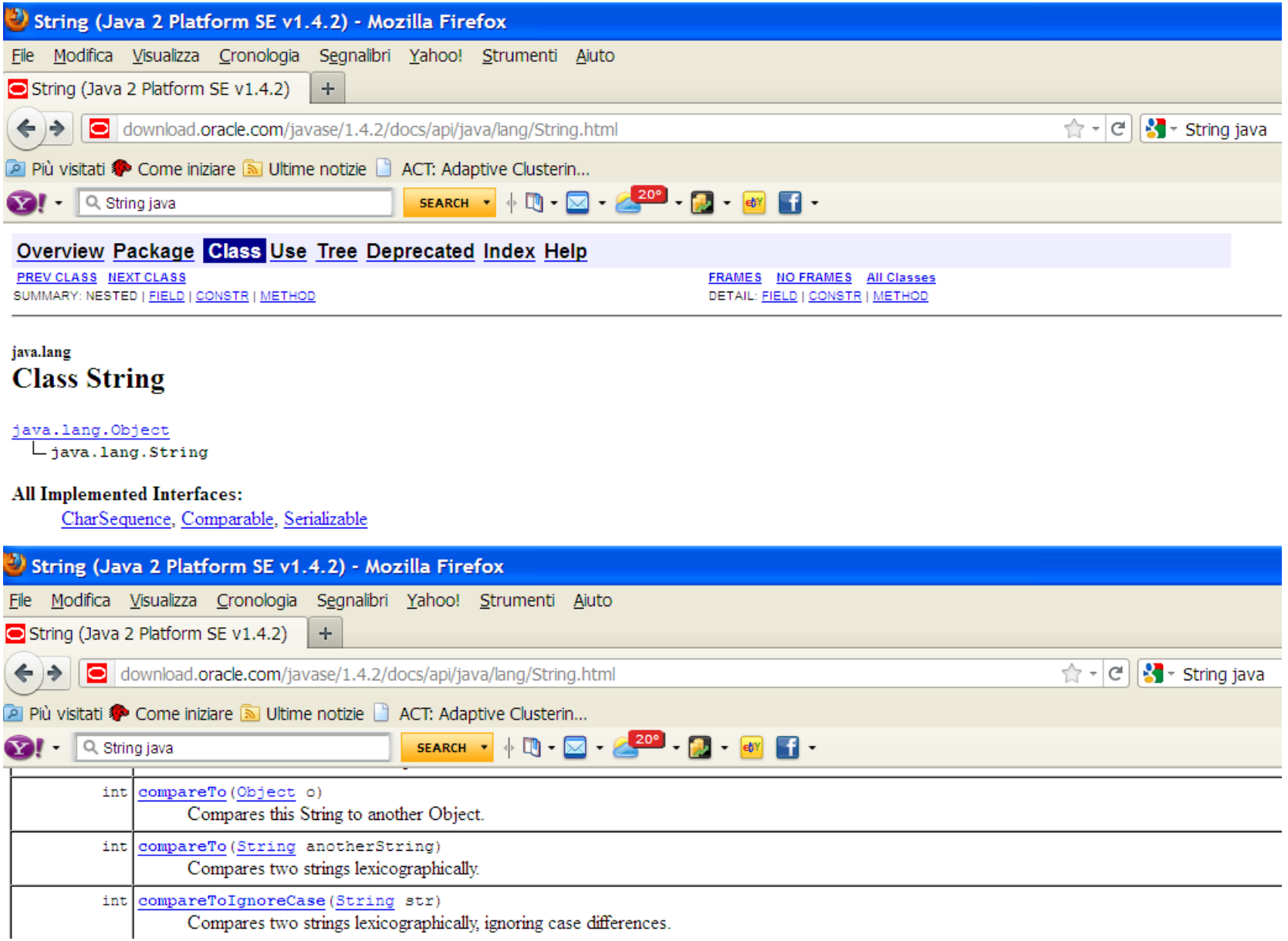
“<” versus compareTo()

da bubblesort():

```
…
for (int j=1;j<= A.length-i;j++)
   {
       if(A[j] < A[j-1]) //if(A[j-1].compareTo(A[j])>0)
       {
          …
```

operator < cannot be applied to java.lang.String

“<” versus compareTo()



compareTo

public int **compareTo**(String anotherString)

- Compares two strings lexicographically. […]. The result is a positive integer if this String object lexicographically follows the argument string. The result is zero if the strings are equal; compareTo returns 0 exactly when the equals(Object) method would return true. […]

- **Parameters:**
  - anotherString - the String to be compared.
- **Returns:**
  - the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

Ordinare un Array Studente[]

- Definire la classe Studente

Studente
-int : matricola -float : mediaVoti -int : nroEsamiSostenuti -int : nroLodi
int compareTo(Studente)

```
int compareTo(Studente o)
{
 if(o.getMediaVoti()<this.mediaVoti) return +1;
        else
 if(o.getMediaVoti()>this.mediaVoti) return -1;
 return 0;
}
```

Ordinare un Array Studente[]

Dotare la classe **`Studente`** di un costruttore definito come segue:

```
private static int matricolaCount=0;
Studente(){
 matricola=matricolaCount++;
 mediaVoti=(float)(new Random().nextDouble()*30.0);
 nroEsamiSostenuti=new Random().nextInt(20)+1;
 nroLodi=new Random().nextInt(nroEsamiSostenuti)+1;
}
```

Ordinare un Array Studente[]

Aggiungere nella classe **`OrdinaArrayObject`** i metodi:

```
static int bubblesort(Studente A[]);
static void stampaArray(Studente A[]);
```

Modificare al fine di inizializzare, stampare, ordinare un array **`Studente[10]`**

Problema

E se volessimo ordinare

```
Carta A[] =new Carta[10];
Persona A[]= new Persona[100];
Libro A[]= new Libro[25]
```

Dovremmo definire:

```
static int bubblesort(Carta A[]);
static int bubblesort(Persona A[]);
static int bubblesort(Libro A[]);
```

```
static void bubblesort(Carta A[]){
    for (int i=1; i<=A.length-1; i++){
        boolean scambiAvvenuti = false;
        for (int j=1;j<= A.length-i;j++)
        {
            if( A[j-1].compareTo(A[j])>0)
            {
                Carta temp=A[j];
                A[j]=A[j-1]; A[j-1]=temp; scambiAvvenuti=true;
            }
        }
        if(!scambiAvvenuti) break;
    }
}
```

Codice replicato!

```
static void bubblesort(Studente A[]){
    for (int i=1; i<=A.length-1; i++){
        boolean scambiAvvenuti = false;
        for (int j=1;j<= A.length-i;j++)
        {
            if( A[j-1].compareTo(A[j])>0)
            {
                Studente  temp=A[j];
                A[j]=A[j-1]; A[j-1]=temp; scambiAvvenuti=true;
            }
        }
        if(!scambiAvvenuti) break;
    }
}
```

Interfaccia *Comparable*

```
interface Comparable
{
    public int compareTo(Object o)
}
```

- Se una classe implementa l'interfaccia **`Comparable`** allora sarà definita una relazione d'ordine totale sui suoi oggetti, che saranno "*confrontabili*".
- L'intero restituito dal metodo **`compareTo`** deve essere
  1. positivo se l'oggetto passato è minore
  2. negativo se l'oggetto passato è maggiore
  3. zero se l'oggetto passato è uguale

  all'oggetto sul quale si invoca **`compareTo`**.

Interfaccia *Comparable*

Alcune classi che implementano l'interfaccia *Comparable*

Byte	Numerico con segno
Character	Numerico senza segno
Long	Numerico con segno
Integer	Numerico con segno
Short	Numerico con segno
Double	Numerico con segno
Float	Numerico con segno
BigInteger	Numerico con segno
BigDecimal	Numerico con segno
File	Dipende dal sistema, lessicografico
String	Lessicografico
Date	Ordinamento cronologico

Ordinare array di oggetti

```
ordinaArray(int A[]) { …}
ordinaArray(float A[]) { …}

…
ordinaArray(Comparable A[]) { …}
```

Esercizio

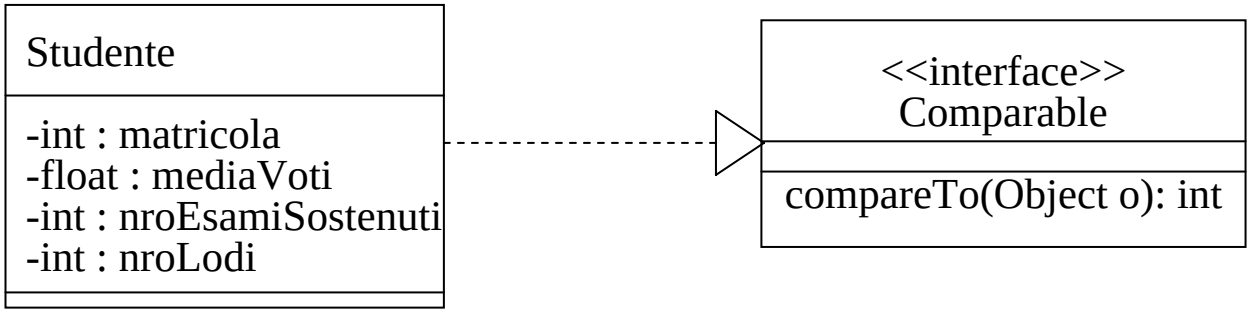
Modificare *bubblesort* per ordinare array di oggetti, testare con una delle classi *wrapper* dei tipi primitivi

```
static void bubblesort (int A[]){
   for (int i=1; i<=A.length-1; i++)
   {
       boolean scambiAvvenuti = false;
       for (int j=1;j<= A.length-i;j++) {
           if(A[j]< A[j-1]){
               int temp=A[j];
               A[j]=A[j-1];
               A[j-1]=temp;
               scambiAvvenuti=true;
           }
       }
       if(!scambiAvvenuti) break;
    }
   }
```

Per svolgere l'esercizio completare l'implementazione della classe *OrdinaArrayObject* (vedi ***OrdinaArrayObject.java***)

Esercizio

Si consideri la classe *Studente* fornita. Essa implementa l'interfaccia *Comparable* e ordina gli studenti in base alla media dei voti.




```
public int compareTo(Object o) {
 if(((Studente)o).getMediaVoti()<this.mediaVoti) return +1;
      else
 if(((Studente)o).getMediaVoti()>this.mediaVoti) return -1;
return 0;
}
```

Esercizio

```
class Studente implements Comparable{
   …
    public static void main(String rgs[]){
      Studente s[]=new Studente[4];
      s[0]=new Studente((float)28.5,10,3);
      s[1]=new Studente((float)24.5,12,5);
      s[2]=new Studente((float)26.5,15,1);
      s[3]=new Studente((float)22.5,14,4);
      int tempI=0; // cerca minimo
      for(int i=1;i<s.length;i++) {
          if(s[i].compareTo(s[tempI])<0)
             tempI=i;
      }
    System.out.println(s[tempI]);
   }
}
```

MATRICOLA: 4 MEDIA VOTI: 22.5 NUMERO ESAMI SOSTENUTI: 14 NUMERO LODI: 4

Esercizi

1. Creare un array di 10 elementi di classe *Studente*, avvalorarlo usando il costruttore standard di Studente (quello che genera casualmente i dati) e ordinarlo con il metodo *bubblesort*.
2. Modificare il metodo **`compareTo`** di *Comparable* in modo da ordinare gli studenti in base al numero delle lodi.
3. Modificare i metodi:
   - *sequentialSearch*
   - *binarySearch*

   di ***RicercaArray.java*** in modo da ricercare degli elementi in *array* ordinati di oggetti.