Reti e Laboratorio 3
Modulo Laboratorio 3
a.a. 2024-2025

docente: Laura Ricci

laura.ricci@unipi.it

Assignment 2:
compareTo e Bounded Types

14/10/2024

L'INTERFACCIA COMPARABLE

- un'operazione tipica è confrontare elementi di una collezione, ad esempio per ordinarli, utilizzando gli algoritmi offerti da Java
- l'interfaccia `Comparable` è utilizzata per definire come gli oggetti istanze di una certa classe devono essere confrontati

```
package java.lang;
public interface Comparable<T> {
    int compareTo(T);
    }
```

- il metodo `compareTo()` accetta un singolo oggetto come parametro e restituisce un valore di tipo `int`, e la sua implementazione deve restituire
  - un valore positivo indica che l'oggetto su cui viene chiamato `compareTo()` è maggiore dell'oggetto parametro.
  - un valore pari a zero indica che i due oggetti sono uguali.
  - un valore negativo indica che l'oggetto su cui viene chiamato `compareTo()` è minore dell'oggetto parametro.

L'INTERFACCIA COMPARABLE

- una classe deve implementare l'interfaccia per consentire il confronto tra le sue istanze
- è possibile poi ordinare gli elementi come si fa con interi o stringhe

```
class MyClass implements Comparable<MyClass> {
// Corpo della classe
...
public int compareTo(MyClass value)
 {
 // Logica del confronto
 ...
 return result;
}
}
```

- il risultato deve essere restituito considerando le regole viste nella slide precedente
- l'implementazione del metodo determina come gli elementi saranno ordinati da metodi come `Arrays.sort()` e `Collections.sort()`

IMPLEMENTARE L'INTERFACCIA

```
package SortExample;

public class Employee implements Comparable<Employee> {
  String firstName;
  String lastName;

  public Employee(String first, String last)
  {
    this.firstName = first;
    this.lastName = last;
  }

  // User-friendly output when printed.
  public String toString()
   { return "( " + lastName + ", " + firstName + " )";}

  public int compareTo(Employee value)
    {
       if (this.lastName.compareTo(value.lastName) != 0) {
           // if lastNames are different, compare lastName
       return this.lastName.compareTo(value.lastName);
    } else {
          // if lastNames are the same, compare firstName
      return this.firstName.compareTo(value.firstName);
    }
  }}
```

IMPLEMENTARE L'INTERFACCIA

```
package SortExample;

import java.util.*;

public class SortExample {
   public static void main(String[] args)
     {
      // creaiamo un array con alcuni Employee
      Employee a[] = new Employee[5];
      a[0] = new Employee("Laura","Ricci");
      a[1] = new Employee("Giovanni","Rossi");
      a[2] = new Employee("Maria","Verdi");
      a[3] = new Employee("Antonio","Bianchi");
      a[4] = new Employee("Mario","Gialli");

      // il metodo .sort() usa l'interfaccia Comparable
      Arrays.sort(a);
      // Print out the sorted Employees
     for (int i=0; i < a.length; i++) {
         System.out.println(a[i]);
    }
  }
```

```
OUTPUT

( Bianchi, Antonio )
( Gialli, Mario )
( Ricci, Laura )
( Rossi, Giovanni )
( Verdi, Maria )
```

JAVA BOUNDED TYPES

- Java ha introdotto i Generics che consentono il riuso del codice e permettono un controllo forte dei tipi da parte del compilatore
- bounded types
  - permettono di imporre dei vincoli sui tipi generici che possono essere utilizzati come parametri di classi, interfacce e metodi
  - definiti con la seguente sintassi

```
class MyClass<T extends MyClassType> {
// ...
}
```

- esempio

```
class Shape {  // ...  }
class Circle extends Shape {  // ...  }

class ShapeContainer<T extends Shape> {
   private T shape;
   public void addShape(T shape) {
       this.shape = shape;
   }
  public T getShape() {
        return shape;  }  }
```

USO DEI BOUNDED TYPES

```
public static <T> T findMax( T[] array)
   {
     int maxIndex = 0;
     for ( int i = 1; i < array.length; i++) {
          if ( array[i].compareTo(array[maxIndex]) > 0 )
             maxIndex = i;
     }
     return array[maxIndex];
   }
```

- il compilatore restitusice il seguente errore

```
The method CompareTo(T) is undefined for type T
```

- quale è il problema?
  - il compilatore non può sapere se il tipo generico T verrà istanziato con un tipo che implementa il metodo `compareTo` usato nel ciclo
  - per questo solleva un errore

USO DEI BOUNDED TYPES

```
public static <T extends Comparable<T> > T findMax( T[] array) {

    int maxIndex = 0;
    for ( int i = 1; i < array.length; i++) {
          if ( array[i].compareTo(array[maxIndex]) > 0 )
          maxIndex = i;
     }
     return array[maxIndex];
  }
```

- abbiamo ristretto il parametro di tipo T a un tipo che implementi l'interfaccia `Comparable <T>`, garantendo così che la chiamata alla `compareTo` è valida
- il compilatore non solleva errori