UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Informatica
@UnivAQ

Laboratorio di Algoritmi e Strutture Dati
a.a. 2022/2023

CONFRONTO TRA OGGETTI:
le interfacce Comparable e Comparator

Giovanna Melideo
Università degli Studi dell'Aquila
DISIM

Richiami: La libreria standard

- Java possiede un'enorme e lussuosa libreria di classi e interfacce standard, che costituisce uno dei punti di forza del linguaggio.
- Essa è organizzata in vari **package** e **subpackage** che raccolgono le classi secondo un'organizzazione basata sul campo d'utilizzo.

La libreria standard

- I principali package sono:
  - **java.io** contiene classi per realizzare l'input – output in Java
  - **java.awt** contiene classi per realizzare interfacce grafiche, come `Button`
  - **java.net** contiene classi per realizzare connessioni, come `Socket`
  - **java.applet** contiene un'unica classe: `Applet`. Questa permette di realizzare applet
  - **java.util** raccoglie classi d'utilità, come `Date`
  - **java.lang** è il package che contiene le classi nucleo del linguaggio, come `Object`, `System`, `String`, l'interfaccia `Comparable`, ecc

Il comando `import` (1 di 5)

- In qualunque programma Java ci si può riferire alle classi contenute nei package tramite il loro nome "lungo", cioè avente come prefisso anche il nome del package e del subpackage.
- Tuttavia nomi “lunghi" sono scomodissimi!

Es:
```
java.util.Scanner tastiera =
        new java.util.Scanner(System.in);
```

Il comando `import` (2 di 5)

- Le dichiarazioni **`import`** poste all'inizio di un file .java, prima di ogni dichiarazione di classe, permettono di usare in quel file i nomi "corti" delle classi importate, senza il prefisso del package.
- Per utilizzare il nome “corto” di una classe della libreria all’interno di una nuova classe bisogna dunque importarla.

Il comando **`import`** (3 di 5)

- Supponiamo di voler utilizzare la classe **`Date`** del package `java.util`. Prima di dichiarare la classe in cui abbiamo intenzione di utilizzare `Date` dobbiamo scrivere:

```
import java.util.Date;
```

- oppure, per importare tutte le classi del package java.util:
- `import java.util.*; //` uso di wild card

Il comando **`import`** (4 di 5)

- Di default in ogni file Java è importato automaticamente tutto il package **`java.lang`**, senza il quale non potremmo utilizzare classi fondamentali quali **`System`**, **`String`**, **`Math`**.
- Notiamo che questa è una delle caratteristiche che rende Java definibile come "semplice". Quindi, nel momento in cui compiliamo una classe Java, il compilatore anteporrà alla dichiarazione della nostra classe il comando:

```
import java.lang.*;
```

Il comando `import` (5 di 5)

- Le dichiarazioni di **`import`** NON copiano programmi né in formato sorgente né in formato compilato; rendono semplicemente utilizzabili nomi corti invece di nomi lunghi. Pertanto, importare tutte le classi di un package non è penalizzante rispetto a importarne una sola.
- L'asterisco non implica l'importazione delle classi appartenenti ai "sottopackage"

  `import java.awt.*` non importa **`java.awt.event.*`**

- Quindi l'istruzione **`import java.*`** non importa tutti i package fondamentali.

La documentazione del JDK

- Per conoscere tutte le classi (che sono in continua evoluzione), basta consultare la documentazione del JDK-Java Development Kit (da installare a parte).
- Aprire il file "index.html" che si trova nella cartella “API” della cartella "Docs" del J.D.K.
- Se non si trova la cartella fare una ricerca sul disco rigido.
- Se la ricerca fallisce procurarsi la documentazione (www.java.sun.com)

L'interfaccia `java.lang.Comparable`

- L'interfaccia **`Comparable`** impone un **criterio di ordinamento** sugli oggetti della classe che la implementa (**ordinamento naturale** della classe).
- **`Comparable`** contiene il solo metodo di confronto naturale:

```
public interface Comparable<T> {
 int compareTo(T obj);
}
```

- Il metodo **`compareTo`** confronta l'oggetto corrente **`this`** con l'oggetto specificato **`obj`**

L'interfaccia `Comparable` (1 di 5)

`x.compareTo(y)` restituisce:

- un valore negativo se `x` è minore di `y`
  - se `x` «precede» `y`
- 0 se `x` è uguale a `y`
- un valore positivo se `x` è maggiore di `y`
  - se `x` «segue» `y`

L'interfaccia `Comparable` (2 di 5)

`x.compareTo(y)` deve rappresentare una **relazione d'ordine totale** tra oggetti:

- sia *sgn(a) = 1 se a>0, 0 se a=0, -1 se a<0* (funzione segno)
- dati `x, y, z` appartenenti ad una classe che implementa `Comparable`, deve valere:

1. `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` (riflessività e simmetria)
2. `se x.compareTo(y)<0 e y.compareTo(z)<0 allora x.compareTo(z)<0` (transitività)
3. `se x.compareTo(y)==0 allora sgn(x.compareTo(z))==sgn(y.compareTo(z))`

L'interfaccia `Comparable` (3 di 5)

- **`x.compareTo(y)`** dovrebbe lanciare l'eccezione ClassCastException se riceve un oggetto **`y`** che non è confrontabile con **`x`**, a causa del suo tipo effettivo
- Nota che **`null`** non è istanza di nessuna classe, e **`x.compareTo(null)`** dovrebbe sollevare l'eccezione NullPointerException anche se **`e.equals(null)`** returns **`false`**.

L'interfaccia `Comparable` (4 di 5)

- L'implementazione di `Comparable` è **consistente o compatibile con** `equals()` se dati `x` e `y`:

  `x.compareTo(y) == 0` se e solo se `x.equals(y)==true`

- È fortemente raccomandato, ma non strettamente richiesto, che `compareTo`() sia consistente (o compatibile) con `equals`()
- Ogni classe che implementa l'interfaccia `Comparable` e viola questa condizione dovrebbe dichiararlo esplicitamente
  - *"Note: this class has a natural ordering that is inconsistent with equals."*

L'interfaccia `Comparable` (5 di 5)

Classi di Java che implementano **`Comparable`**:

- **`String`**
- **`File`**
- **`Date`**
- **`Byte`**, **`Character`**, **`Short`**, **`Integer`**, **`Long`**, **`Float`**, **`Double`** (classi wrapper)

L'interfaccia `java.util.Comparator`

- In alternativa all'interfaccia **`Comparable`**, si può realizzare una seconda classe che implementa l'interfaccia **`Comparator`**

```
public interface Comparator<T> {

    public int compare(T x, T y);

}
```

L'interfaccia `Comparator` (1 di 2)

`compare(x,y)` restituisce:

- un valore negativo se `x` è minore di `y`
  - se `x` «precede» `y`
- 0 se `x` è uguale a `y`
- un valore positivo se `x` è maggiore di `y`
  - se `x` «segue» `y`

L'interfaccia **`Comparator`** (2 di 2)

L'uso di **`Comparator`** è indicato quando:

- la classe da ordinare non ha un unico criterio di ordinamento naturale, oppure
- la classe da ordinare è già stata realizzata e non si può o non si vuole modificarla
- L'interfaccia **`Comparable`** è definita nel package **`java.lang`** mentre **`Comparator`** in **`java.util`**:
  - sottolinea che il metodo **`compareTo`** dovrebbe essere fornita di default, mentre **`compare`** come un'utility

Uso di comparatori

- Nell'API Java sono presenti metodi che utilizzano le interfacce **`Comparable`** e **`Comparator`** per fornire algoritmi di ordinamento di array e di liste
- Per quanto riguarda gli array, tali metodi si trovano nella classe **`java.util.Arrays`**
- Per quanto riguarda le liste, tali metodi si trovano nella classe **`java.util.Collections`**
- **`Le classi Arrays`** e **`Collections`** contengono solo metodi statici

Arrays.sort

```
static void sort(Object[] a)
```

- ordina l'array a in senso non-decrescente, in base all'ordinamento naturale tra i suoi elementi
- ovvero, suppone che tutti gli elementi contenuti siano confrontabili tra loro tramite l'interfaccia **Comparable**

```
static <T> void sort(T[] a, Comparator<? super T> c)
```

- ordina l'array in senso non-decrescente, in base all'ordinamento indotto dal comparatore c
- L'algoritmo usato è una versione ottimizzata del quicksort

Collections.sort

```
static <T extends Comparable<? super T>> void
    sort(List<T> list)
```

- ordina la lista a in senso non-decrescente, in base all'ordinamento naturale tra i suoi elementi

```
static <T> void    sort(List<T> list, Comparator<? super
T> c)
```

- ordina la lista in senso non-decrescente, in base all'ordinamento indotto dal comparatore c
- L'algoritmo usato è una versione ottimizzata del mergesort

Ordinamento

- In entrambi i casi, l'ordinamento è in-place e stabile:
  - l'array viene modificato senza utilizzare strutture di appoggio (in-place)
  - gli elementi equivalenti secondo l'ordinamento mantengono l'ordine che avevano originariamente (stabile)
- **Homework**: Declare an array whose elements come from the same Student class. Each student has a name and grade point average. Sort the students in alphabetical order. Next, sort the students in decreasing order of GPAs.

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Informatica
@UnivAQ

Domande?

Giovanna Melideo
Università degli Studi dell'Aquila
DISIM