Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

Generics e sottotipi (cenni)

Generics e sottotipi

- I meccanismi di subtyping si estendono alle classi generiche:

  `class C<T> implements / extends D<T> {…}`

- `C<T>` è sottotipo di `D<T>` per qualunque `T`
- Analogamente:

  `class C<T> implements / estends D {…}`

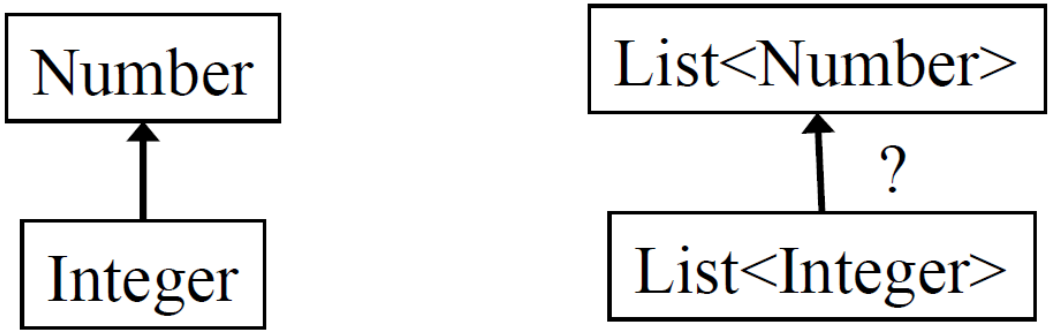
- `C<T>` è sottotipo di `D` per qualunque `T`

Generics e sottotipi: esempi

- `Integer` è un sottotipo di `Number`
- `ArrayList<E>` è un sottotipo di `List<E>`
- `List<E>` è un sottotipo di `Collection<E>`

Ma…

Generics e sottotipi



- **`Integer`** è un sottotipo di **`Number`**
- **`List<Integer>`** è un sottotipo di **`List<Number>`**?

Generics e sottotipi

In generale:

- se `Type1` è sottotipo di `Type2`, allora `C<Type1>` *non* è un sottotipo di `C<Type2>`

  - `Set<Integer>` non è sottotipo di `Set<Object>`
  - `Collection<Object>` non è supertipo di `Collection<T>` per nessun `T != Object`

Generics e sottotipi

- Un operatore sui tipi F è:
  - **covariante** se F<T'> ≤ F<T> quando T' ≤ T
    (F conserva la relazione di sottotipo).
  - **controvariante** se F<T> ≤ F<T'> quando T' ≤ T
    (F inverte la relazione di sottotipo)
  - **invariante** se non è né covariante né controvariante
- Formalmente: la nozione di sottotipo usata in Java è **invariante** per le classi generiche

Wildcard

Wildcard = una variabile di tipo anonima

- **?** indica un qualche tipo, non specificato
  - **`Collection<?>`** è supertipo di qualunque **`Collection<T>`**

Sintassi delle wildcard:

- `? extends Type`, sottotipo non specificato del tipo `Type`
  - Nota: `GenType<Integer>` non è sottotipo di `GenType<Number>` MA `GenType<Integer>` è sottotipo di `GenType<? extends Number>`.
  - **?** notazione semplificata per `? extends Object`
- `? super Type`, supertipo non specificato del tipo `Type`

Wildcard

Quale differenza c'è tra

1) `List<T>`
2) `List<? extends T>` ?

- Nel caso 2) il tipo anonimo è un sottotipo sconosciuto di `T`
- Es: `List<? extends Animal>` può memorizzare `Gatti` ma non anche `Cavalli`

wildcard

```
interface Set<E> {
     void addAll(Collection<? extends E> c);
}
```

- maggiormente flessibile rispetto a

```
void addAll(Collection<E> c);
```

- espressiva come

```
<T extends E> void addAll(Collection<T> c);
```

Wildcard

Quando si usano le wildcard?

- si usa `? extends T` nei casi in cui si vogliono ottenere dei valori (da un produttore di valori)
- si usa `? super T` nei casi in cui si vogliono inserire valori (in un consumatore)
- non vanno usate (basta `T`) quando si ottengono e si producono valori

```
<T> void copy(List<? super T> dst,
              List<? extends T> src);
```

Esempi

Stampa gli elementi di <u>un qualunque collection:</u>

1. Sia `static void printSet(Set<Object> els) {…}`

```
Set<String> s = new TreeSet<String>();
…
printSet(s); //errore
```

- `Set<Object>` non è il supertipo di `Set<T>` per nessun `T != Object`

  "The method printSet(Set<Object>) in the type Test is not applicable for the arguments (Set<String>)»

esempi

Stampa gli elementi di un qualunque collection:

2. Sia `static void printSet(Set<?> els) {…}`

```
Set<String> s = new TreeSet<String>();
…
printSet(s); //ok
```

- `Set<?>` è il supertipo di `Set<T>` per qualunque `T`

esempi

```
Collection<?> c = new ArrayList<String>();
c.add(new String()); // errore
```

- Poichè non sappiamo esattamente quale tipo indica **?**, non possiamo inserire elementi nella collezione.
- In generale, non possiamo modificare valori che hanno tipo **?**

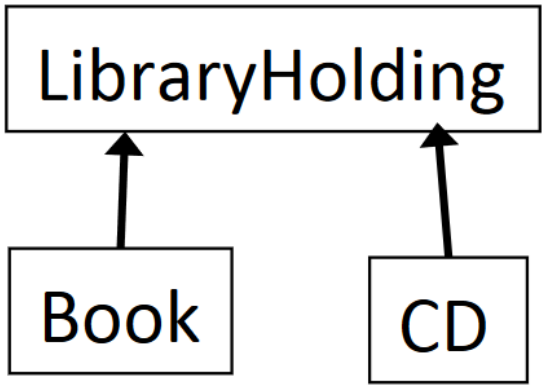
Covarianza degli array

- Domanda: se `Type1` è un sottotipo di `Type2`, che relazione esiste tra `Type1[]` e `Type2[]`?
  - Sappiamo che se `Type1` è un sottotipo di `Type2` i tipi `Type1[]` e `Type2[]` non dovrebbero essere correlati. Invece…

Peculiarità di Java:

- se `Type1` è un sottotipo di `Type2`, allora `Type1[]` è un sottotipo di `Type2[]`

Gli array: esempio



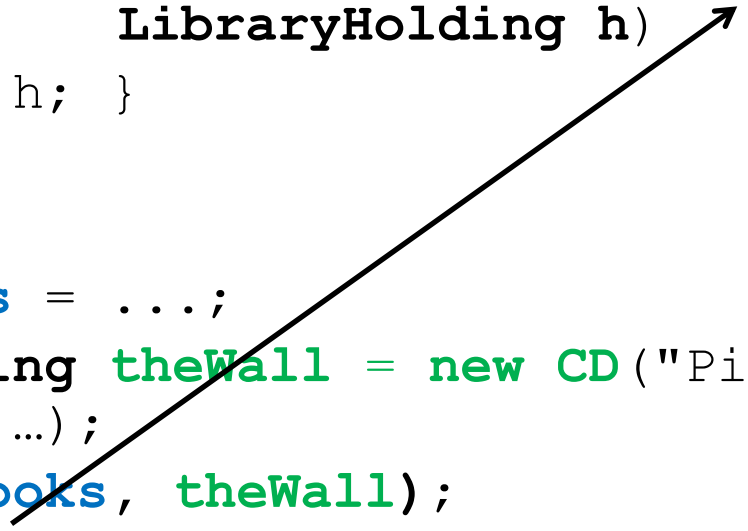

```
void maybeSwap(LibraryHolding[ ] arr) {
      if(arr[17].dueDate( ) < arr[34].dueDate( ))
            // ... swap arr[17] and arr[34]
}

// cliente
Book[ ] books = ...;
maybeSwap(books); // usa la covarianza degli array
```

Gli array: esempio

```
void replace17(LibraryHolding[ ] arr,
               LibraryHolding h)
{ arr[17] = h; }

// cliente
Book[] books = ...;
LibraryHolding theWall = new CD("Pink Floyd",
"The Wall", …);
replace17(books, theWall);
Book b = books[17]; // contiene un CD
b.getChapters( );   // problema!!
```



- Attenzione agli array in Java!