UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

5. Generics in Java

2 esercizi risolti + 1 extra

Tutorato di
Programmazione Avanzata

RELATORE
Imberti Federico

SEDE
Dalmine, BG

Key-points della teoria: Perchè usare i generics?

1. Nel momento in cui scriviamo un codice **ad-hoc** per la nostra codebase possiamo non preoccuparci di quanto generale sia un metodo o una classe;
    a. Tradotto: se ho a che fare solo con interi, che per esempio voglio salvare in una lista e poi ordinare con un algoritmo di sort, non sarà un problema se la lista potrà contenere solo interi e l'algoritmo eseguire solo confronti tra interi.
2. Se invece voglio scrivere una **libreria** o del codice che sia maggiormente **riutilizzabile** dovrò invece scrivere metodi e classi più generiche;
    a. Tradotto: anche se nella mia applicazione avrò a che fare solo con interi magari altri sviluppatori che useranno il mio codice dovranno gestire dei float. Per questo motivo sviluppo il mio codice in modo che possa gestire tutti i sottotipi di "Number" anziché solo gli "Integer".

Key-points della teoria: Invarianza, Covarianza, Controvarianza

- **Covarianza**: accetto un tipo T e tutti i suoi sottotipi
  - Type safe solo in **output**: non posso aggiungere nulla a una collection covariante
  - Java:
    - Array sono covarianti
    - Metodi sono covarianti nel tipo di ritorno (posso ritornare un oggetto più specifico)
    - Implementato con `<? extends T>`

- **Controvarianza**: accetto un tipo T e tutti i suoi supertipi
  - Type safe solo in **input**: non posso ottenere nulla da una collection controvariante
  - Java:
    - Implementato con `<? super T>`

- **Invarianza**: accetto un tipo T e solo T
  - Type safe sia in lettura che in scrittura
  - Java:
    - Implementato con `<T>`

Key-points della teoria: Invarianza, Covarianza, Controvarianza (Java)

Posso usare una `List<Gatto>` laddove è richiesta una `List<Animale>`?

- **No**, perché i Type parameters in java sono invarianti.

Posso usare `List<Gatto>` laddove è richiesta una `List<Animale extends Gatto>`?

- **Si**, grazie alla covarianza, ma non potrò aggiungere né oggetti Gatto né oggetti Animale alla lista. Potrò tuttavia aggiungere “null”.

Posso usare `List<Gatto>` laddove è richiesta una `List<Animale super Gatto>`?

- **Si**, grazie alla controvarianza, ma non potrò leggere né oggetti Gatto né oggetti Animale dalla lista. Potrò tuttavia leggere oggetti di tipo Object.

Key-points della teoria: Further Readings (pt. 1)

- Covariance and Contravariance, YouTube, spiegata molto bene;
- Generics and Wildcards in Java | Part 2, YouTube, focus su Java.

Key-points della teoria: Tipi generici in Java

Java dispone di 3 modi per dichiarare dei tipi generici:

1. Usando “**Object**”, cioè il supertipo di ogni classe in Java
   a. Non possiamo dare vincoli (accettiamo qualunque cosa).

```
Object arrayOfObjects[10]; //Un array di oggetti e relativi sottotipi con 10 campi
```

2. Usando le “**wildcard**”
   a. Possiamo dare sia un limite superiore che inferiore;
   b. Ma ammettono solo un supertipo o sottotipo.

```
List<? Extends Number> lNums = new ArrayList<>() //ArrayList di Numeri e tutti i
suoi sottotipi (Integer, Float …)

List<? Super Integer> lInts = new ArrayList<>() //ArrayList di Interi e tutti i
suoi supertipi (Number e Object)
```

Key-points della teoria: Tipi generici in Java

Java dispone di 3 modi per dichiarare dei tipi generici:

3. Usando i “Type Parameters” (**Generics**)
    a. Possiamo vincolarli solo superiormente;
    b. Ma concatenare più supertipi;
    c. Implementano anche invarianza.

```
List<T Extends Number> lNums = new ArrayList<>() //ArrayList di Numeri e tutti i
suoi sottotipi (Integer, Float …)


List<T Super Integer> lInts = new ArrayList<>() //ERRORE di compilazione


List<T Extends A & Comparable<T>> lNums = new ArrayList<>() //ArrayList di istanze
di A e dei suoi sottotipi a patto che siano comparabili
```

Key-points della teoria: Generics Vs Wildcards

1. Non possiamo usare le wildcard per definire **classi generiche** o il tipo dei parametri di una funzione
   a. Dobbiamo ricorrere ai generics

```
Class ListaDiNumeri <T extends Numbers> { . . . }

public static void <E> doStuffTo(E elem) { . . . }
```

2. Quando abbiamo un **parametro generico unbounded** possiamo usare sia generics che wildcards
   a. Convenzionalmente si preferiscono le wildcards
   b. Specialmente quando il parametro appare solo una volta

```
public static <E> void swap(List<E> list, int src, int des);

public static void swap(List<?> list, int src, int des);
```

Key-points della teoria: Further Readings (pt. 2)

- Type Parameters vs Wildcards (Baeldung, deep-dive rispetto alle slide)
- What's the purpose behind wildcards and how are they different from generics? (Stack Overflow, interessante corollario del precedente)
- Wildcards in Java (Geeks for Geeks, molti esempi carini)
- Generics in Java (Geeks for Geeks, ma che ve lo dico a fare?)
- Class Object (R.T.F.M.)

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Domandine?

Esercizio 1/2

Usando i generics in Java, scrivi una classe **Dizionario** che rappresenta una mappa da un insieme di un tipo T (chiave) ad un altro tipo S (valore). Il primo tipo T è un comparable (cioè estende in modo opportuno Comparable). Come sottostante usa due arraylist (uno per le chiavi di tipo T e uno per i valori di tipo S tali che chiave e valore stiano nella stessa posizione nei due array. Devi definire i seguenti metodi:

- un metodo per inserire una relazione tra due valori (T e S).
- un metodo che restituisce il valore (S) per una certa chiave (T).

Scrivi un main in cui fai un po' di prove. Se riesci sfrutta il fatto che T sia comparable.
Se ho Persona che è Comparable, Studente estende Persona (ma non ha un suo metodo compareTo), si può fare una **Dizionario** di Studenti e loro voto intero in programmazione avanzata? Fai qualche prova.

Esercizio 2/2

Implementa la struttura dati “StringBuffer” dell'esercizio 2 in Java, che però sia generica in modo che si possa memorizzare ogni tipo che estenda Object in questo array variabile. Come sottostante NON usare un array list o altre collezioni, ma usa un array puro []. Implementa i metodi dell'esercizio 2.

Nel main fai un paio di esempi con interi o char.

Esercizio Extra

Implementa la struttura dati “List” dell’esercizio 3 in Java, che però sia generica in modo che si possa memorizzare ogni tipo che estenda Object. Come tipo utilizza una lista. Crea un “main” che illustri l’utilizzo di List creando una lista di interi ed una lista di stringhe.