



Metodologie di Programmazione

Lezione 28: I tipi generici (parte 2)

Lezione 28:

Sommario



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Jolly come tipi generici
- Cancellazione del tipo
- Parola chiave super nei generici
- Overloading di metodi generici
- Tipi raw

- Nel caso in cui non sia necessario utilizzare il tipo generico **T** nel corpo della classe o del metodo, è possibile utilizzare il **jolly** ?

```
public static void mangia(ArrayList<? extends Mangiabile> frutta)
{
    // mangia la frutta
}
```

- Equivalente a:

```
public static <T extends Mangiabile> void mangia2(ArrayList<T> frutta)
{
    // mangia la frutta
}
```

- ma senza la nozione del tipo utilizzato

Esercizio:

Successioni numeriche



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Scrivere una classe **SuccessioniNumeriche**, i cui oggetti sono inizialmente vuoti
- La classe espone un metodo **addSuccessione** che, preso in input un nome e una sequenza numerica, consente di aggiungere una successione (di lunghezza finita) del tipo numerico generico specificato (Integer, Float, Long, ecc.)
- La classe ha inoltre un metodo **getSuccessione**(String nome) con cui è possibile recuperare una successione a partire dal nome mnemonico
- Prevedere le seguenti successioni:
 - «Fibonacci»: 1, 1, 2, 3, 5, 8, 13, 21, 34
 - «1/n»: 1, 1/2, 1/3, 1/4, 1/5, 1/6
 - «RandomLong»: contiene oggetti Long costruiti casualmente
- **Hint:** utilizzare la classe **Number** come superclasse del tipo generico

Esempio: metodo generico di somma (1)

- Implementare un metodo generico **somma** che calcoli la somma di tutti i numeri contenuti in una collezione
- Implementazione **non generica**:

```
public class SommaNumeri
{
    public static void main(String[] args)
    {
        // interi e double
        Number[] numeri = { 1, 1.5, 2, 2.5 };
        ArrayList<Number> listaDiNumeri = new ArrayList<Number>();

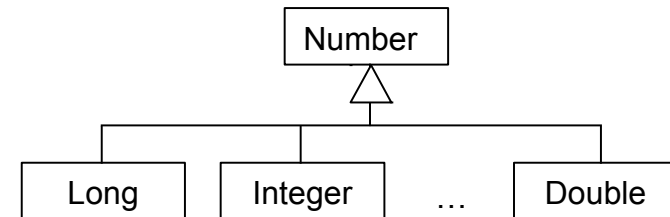
        for (Number n : numeri)
            listaDiNumeri.add(n);

        System.out.printf("La lista contiene: %s\n", listaDiNumeri);
        System.out.printf("La somma dei numeri nella lista è: %.1f\n", somma(listaDiNumeri));
    }

    public static double somma(ArrayList<Number> lista)
    {
        double totale = 0.0;

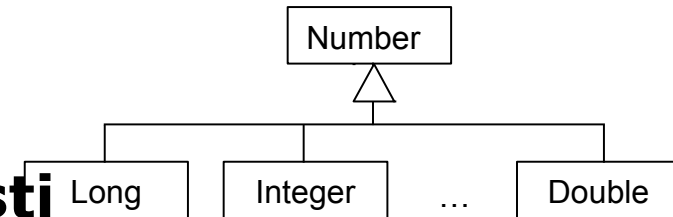
        for (Number n : lista)
            totale += n.doubleValue();

        return totale;
    }
}
```



Esempio: metodo generico di somma (1)

- Implementare un metodo generico **somma** che calcoli la somma di tutti i numeri contenuti in una collezione
- Implementazione **non generica**
- Posso creare **ArrayList<Number>** contenenti interi, double, ecc. **misti**
- Tuttavia, non posso passare in input un **ArrayList<Integer>**:



```
public static void main(String[] args)
{
    // interi
    Integer[] numeri = { 1, 2, 3, 4 };
    ArrayList<Integer> listaDiNumeri = new ArrayList<Integer>();

    for (Integer n : numeri)
        listaDiNumeri.add(n);

    System.out.printf("La lista contiene: %s\n", listaDiNumeri);
    System.out.printf("La somma dei numeri nella lista è: %.1f\n", somma(listaDiNumeri));
}
```

Esempio: metodo generico di somma (2)

- Implementare un metodo generico **somma** che calcoli la somma di tutti i numeri contenuti in una collezione
- Implementazione **generica**:

```
public class SommaNumeriGenerico
{
    public static void main(String[] args)
    {
        // interi
        Integer[] numeri = { 1, 2, 3, 4 };
        ArrayList<Integer> listaDiNumeri = new ArrayList<Integer>();

        for (Integer n : numeri)
            listaDiNumeri.add(n);

        System.out.printf("La lista contiene: %s\n", listaDiNumeri);
        System.out.printf("La somma dei numeri nella lista è: %.1f\n", somma(listaDiNumeri));
    }

    public static double somma(ArrayList<? extends Number> lista)
    {
        double totale = 0.0;

        for (Number n : lista)
            totale += n.doubleValue();

        return totale;
    }
}
```

Come funziona dietro le quinte?


- Mediante la **cancellazione del tipo** (type erasure)
- Quando il **compilatore** traduce il metodo/la classe generica in bytecode Java:
 1. **Elimina la sezione del tipo parametrico** e sostituisce il tipo parametrico con quello reale
 2. Per default **il tipo generico viene sostituito** con il tipo **Object** (a meno di vincoli sul tipo)
- **Solo una copia** del metodo o della classe viene creata!

Cancellazione del tipo: la classe Coppia

```
public class Coppia<T, S>
{
    private T a;
    private S b;

    public Coppia(T a, S b)
    {
        this.a = a;
        this.b = b;
    }

    public T getPrimo() { return a; }
    public S getSecondo() { return b; }
}
```



```
public class Coppia
{
    private Object a;
    private Object b;


    public Coppia(Object a, Object b)
    {
        this.a = a;
        this.b = b;
    }

    public Object getPrimo() { return a; }
    public Object getSecondo() { return b; }
}
```

Cancellazione del tipo: il metodo massimo

```
public class Massimo
{
    static public <T extends Comparable<T>> T getMassimo(T a, T b, T c)
    {
        if (a.compareTo(b) > 0) return a.compareTo(c) >= 0 ? a : c;
        else return b.compareTo(c) >= 0 ? b : c;
    }

    public static void main(String[] args)
    {
        int max = getMassimo(10, 20, 30);
        String s = getMassimo("abc", "def", "ghi");
    }
}
```



```
public class Massimo
{
    static public Comparable getMassimo(Comparable a, Comparable b, Comparable c)
    {
        if (a.compareTo(b) > 0) return a.compareTo(c) >= 0 ? a : c;
        else return b.compareTo(c) >= 0 ? b : c;
    }

    public static void main(String[] args)
    {
        int max = (Integer)getMassimo(10, 20, 30);
        String s = (String)getMassimo("abc", "def", "ghi");
    }
}
```

Esercizio: cancellazione del tipo nella classe Pila

- Mostrare come viene cancellato il tipo dal compilatore nella classe **Pila** creata in un precedente esercizio
- Mostrare inoltre come viene trasformato il seguente codice:

```
Pila<Integer> p = new Pila<Integer>(10);  
p.push(1);  
p.push(2);  
Integer k = p.pop();
```

Le parole chiave `extends` e `super` nei generici

```
static public void esamina(ArrayList<Frutto> frutti)
{
    // ...
}
```

- Si può imporre un **vincolo** sul tipo generico `T` mediante la parola chiave:
 - **`extends`**: `T` deve essere un sottotipo della classe specificata o la classe stessa
 - **`super`**: `T` deve essere una superclasse della classe specificata o la classe stessa

Esempio di utilizzo della parola chiave **super**



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Implementare un metodo statico per l'ordinamento di liste utilizzando il metodo statico per l'ordinamento degli array

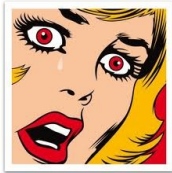
```
public static <T extends Comparable<? super T>> void sort(List<T> list)
{
    Object a[] = list.toArray();
    Arrays.sort(a);

    list.clear();
    for (Object o : a) list.add((T)o);
}
```

A volte “super” nei generici è necessario...



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA



- Ma perché non posso scrivere semplicemente `<T extends Comparable<T>>`?
- Immaginiamo questa situazione:
 - `public class Frutto implements Comparable<Frutto>`
 - `public class Pera extends Frutto implements Comparable<Pera>` **non si può fare!**
 - ❖ Perché non si può implementare due volte la stessa interfaccia
 - `public class Pera extends Frutto` **sì**
 - Se volessi `ordinare una collezione di Pera` non potrei, perché `Pera` non estende `Comparable<Pera>`, ma `Comparable<Frutto>`

Overloading dei metodi generici



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Un metodo generico può essere sovraccaricato, come ogni altro metodo
- Anche da un metodo **non generico con lo stesso nome e numero di parametri**
- Quando il compilatore traduce una chiamata di metodo cerca il metodo più specifico
- **Prima** il **non** generico e **poi**, eventualmente, il metodo generico

Esempio di overloading dei metodi generici

```
public class Massimo
{
    static public <T extends Comparable<T>> T getMassimo(T a, T b, T c)
    {
        if (a.compareTo(b) > 0) return a.compareTo(c) >= 0 ? a : c;
        else return b.compareTo(c) >= 0 ? b : c;
    }

    static public String getMassimo(String a, String b, String c)
    {
        // implementazione specifica per la stringa
        // ...
    }

    public static void main(String[] args)
    {
        // chiama l'implementazione generica
        int max = getMassimo(10, 20, 30);
        // chiama l'implementazione specifica per le stringhe
        String s = getMassimo("abc", "def", "ghi");
    }
}
```


I tipi raw

- I tipi **generici** sono stati introdotti con Java 5
- Per retrocompatibilità, è possibile istanziare una classe generica **senza specificare il tipo parametrico**

```
// istanza con tipo PARAMETRIZZATO
Pila<Integer> p1 = new Pila<Integer>(10);
```

```
// istanza con tipo RAW
Pila p2 = new Pila(10);
```

- È possibile assegnare un'istanza con tipo parametrico a una con tipo raw:

```
// è possibile assegnare un'istanza
// con tipo parametrico a una con tipo raw
p2 = p1;
```

- — e viceversa :
- ```
p1 = p2;
```

# Esercizio: MultiInsieme



SAPIENZA  
UNIVERSITÀ DI ROMA  
DIPARTIMENTO DI INFORMATICA

- Definire una classe generica per gestire multiinsiemi, ovvero un insieme che può contenere più copie dello stesso elemento (ad esempio,  $\{ 1, 1, 2, 3, 3 \}$ ).
- Oltre a definire un costruttore che crea un multiinsieme vuoto, la classe deve definire i seguenti metodi:
  - Un metodo **add** che, dato un valore, lo aggiunge al multiinsieme, restituendo true se il valore non era già contenuto nel multiinsieme, false altrimenti.
  - Un metodo **get** che, preso in input un valore, restituisce il numero di copie di tale valore nel multiinsieme (0 se il valore non è contenuto).
  - Un metodo **contains** che, preso in input un valore, restituisce true se il valore è contenuto nel multiinsieme, false altrimenti.
  - Un metodo **toSet** che restituisce l'insieme dei valori contenuti nel multiinsieme, ovvero senza duplicati
  - Un metodo **intersect** che, preso in input un multiinsieme set dello stesso tipo dell'oggetto su cui il metodo è invocato, modifica il multiinsieme di quest'ultimo in modo da contenere l'intersezione tra se stesso e set. Ad esempio, dato il multiinsieme  $\{ 1, 1, 1, 2, 4, 4, 5 \}$ , l'intersezione con il multiinsieme  $\{ 1, 1, 2, 4, 7, 7 \}$  modifica il primo in  $\{ 1, 1, 2, 4 \}$

- Definire una classe generica per gestire multimappe. Una multimappa mantiene associazioni (chiave, insieme di valori) tali che ad ogni chiave è associato un insieme di valori. Il tipo delle chiavi può essere diverso da quello dei valori.
- Oltre a definire un costruttore che crea una multimappa vuota, la classe deve definire i seguenti metodi:
  - Un metodo **put** che, presa in input una chiave e un valore, aggiunge l'associazione alla multimappa, restituendo true se il valore non era già contenuto nell'insieme associato alla chiave, false altrimenti. Il metodo gestisce la situazione in cui la chiave specificata non esista, creando la nuova associazione.
  - Un metodo **get** che, presa in input una chiave, se la chiave è presente restituisce l'insieme ad essa associato, altrimenti restituisce null.

- Un metodo **contains** che, presa in input una chiave e un valore, restituisce true se l'associazione tra la chiave e il valore è contenuta nella multimappa, false altrimenti
- Un metodo **intersect** che, presa in input una chiave k e un insieme di valori set, se la chiave è presente rende l'insieme dei valori associato alla chiave uguale all'intersezione tra l'insieme originale e l'insieme set preso in input. Se set è pari a null, la chiave k viene rimossa dalla multimappa. Se la chiave non è presente, il metodo lancia l'eccezione `IllegalArgumentException`.
- Un metodo **intersectMultiMappa** che, presa in input una multimappa dello stesso tipo, rende la multimappa dell'oggetto su cui il metodo è invocato uguale all'intersezione delle due multimappe. In altre parole, rimarranno in questa multimappa solamente le chiavi che sono presenti anche nella multimappa presa in input e, per ognuna di queste chiavi, l'insieme dei valori diventerà uguale all'intersezione degli insiemi dei valori associati alla chiave nelle due multimappe.