



Cenni sulla programmazione generica



Variabili di tipo

- Programmazione generica:

Creazione di strutture dati e algoritmi che possono essere utilizzati con tipi di dati diversi

- Es. `ArrayList<String>`,
`ArrayList<Rectangle>`, etc.

- Permette riutilizzo del codice

- Si può realizzare con:

- **ereditarietà** (variabili di tipo `Object`)
- **variabili di tipo** (variabili a cui si può assegnare un tipo *non primitivo* e che possono essere usate come tipi nelle dichiarazioni)



Realizzazione di classi generiche

- Le variabili di tipo di una classe generica
 - sono dichiarate tra parentesi angolari dopo il nome della classe
 - di solito sono indicate con una lettera maiuscola
 - sono utilizzate per dichiarare le variabili, i parametri dei metodi e il valore di restituzione nel codice della classe

```
public class ArrayList <E>
{
    public ArrayList() { . . . }
    public void add(E element) { . . . }
}
```



Variabili di Tipo

- Le variabili di tipo rendono più sicuro e di più facile comprensione il codice generico.
- E' impossibile aggiungere un oggetto di tipo `String` ad un esemplare di `ArrayList<BankAccount>` mentre è possibile aggiungere un oggetto di tipo `String` ad un esemplare di `ArrayList` che sia stato creato con l'intenzione di usarlo per contenere conti correnti



Variabili di Tipo

```
ArrayList<BankAccount> accounts1 =  
    new ArrayList<BankAccount>();  
  
ArrayList accounts2 = new ArrayList();  
// Dovrebbe contenere oggetti di tipo BankAccount  
  
accounts1.add("my savings"); // errore durante la  
                             compilazione  
accounts2.add("my savings"); // errore non individuato  
                             dal compilatore  
  
BankAccount account = (BankAccount) accounts2.getFirst();  
// errore durante l'esecuzione
```



Realizzazione di classi generiche

```
public class Pair<S,T>{  
    public Pair(S primoEl, T secondoEl) {  
        primo = primoEl;  
        secondo = secondoEl;  
    }  
    public S getFirst() { return primo; }  
    public T getSecond() { return secondo; }  
    private S primo;  
    private T secondo;  
}
```



Classe tester per Pair

```
public class PairTester {  
    public static void main(String[] args) {  
        Pair<Double,Integer> p =  
            new Pair<Double,Integer>(3.0,3);  
        double x = p.getFirst();  
        int y = p.getSecond();  
    }  
}
```

- L'effetto ottenuto è come se le variabili di tipo venissero assegnate con i tipi indicati al momento dell'istanziazione
 - in questo caso, **S** con **Double** e **T** con **Integer**



Metodi generici

- Possono appartenere anche a classi non generiche
- Considera l'esempio

```
public class ArrayUtil{  
    public static void print(String[] a) {  
        for(String e : a)  
            System.out.print(e+" ");  
        System.out.println();  
    }  
    .....  
}
```

- L'algoritmo può essere riutilizzato per stampare array di oggetti di qualsiasi tipo



Metodi generici

- Stampa array di oggetti di tipo arbitrario:

```
public class ArrayUtil{  
    public static <E> void print(E[] a) {  
        for(E e : a)  
            System.out.print(e+" ");  
        System.out.println();  
    }  
    .....  
}
```

- Utilizzo metodo:

```
Rectangle[ ] rectangles= .....;  
ArrayUtil.print(rectangles);
```



Osservazioni

- Per utilizzare un metodo generico, non occorre specificare il tipo effettivo da assegnare alle variabili di tipo
- Il tipo del parametro è dedotto dal compilatore dall'uso che ne facciamo
 - Nell'esempio, il compilatore deduce che **Rectangle** è il tipo effettivo da usare per **E**
- Si possono definire metodi generici sia in una classe normale (non generica) che in una classe generica.



Limiti al tipo delle variabili di tipo

- Può essere necessario limitare variabili di tipo
- Es: un metodo generico **min** va bene per oggetti che possono essere confrontati
`public static <E> E min(E[] a)`
- Non pone alcun vincolo sul tipo che possiamo assegnare ad **E**



Uso di extends per tipi generici

```
public static <E extends Comparable> E min(E[] a) {  
    E smallest = a[0];  
    for (int i=1; i<a.length;i++)  
        if (a[i].compareTo(smallest)<0)  
            smallest = a[i];  
    return smallest;  
}
```

- In questo caso **E** può essere assegnata con un qualsiasi tipo che estende **Comparable**
- Per esprimere più di un vincolo si usa “&”

```
public static <E extends Comparable & Cloneable> E  
    min(E[] a)
```

.....




Type erasure

- Le variabili di tipo non sono tipi di Java
 - Ad esempio, se **T** è una variabile di tipo come in **Pair**, non troveremo mai **T.java** oppure **T.class** nel file system
 - **T** non fa parte del nome della classe **Pair**
- Nella compilazione di **Pair** si genera il file **Pair.class** dove non vi è traccia dei parametri **T** e **S** (type erasure)
 - La classe viene trasformata nella classe "grezza" corrispondente



Type erasure

- Consente alle applicazioni Java che usano tipi generici di essere compatibili con le librerie e le applicazioni create prima dell'avvento dei tipi generici (Java 5.0) .
- Tutta l'informazione relativa ai tipi generici viene rimossa
- Ciascuna variabile di tipo viene sostituita con un tipo effettivo di Java opportuno (**Object** oppure il tipo che delimita lo scope della variabile attraverso **extends**)
- I tipi generici nel codice vengono rimpiazzati con il tipo grezzo corrispondente
- Vengono aggiunti i casting dove è necessario (deducendoli dalle istanziazioni degli oggetti)
 - prima della compilazione in senso stretto avviene una traduzione code-to-code



Classe `Pair<S, T>` dopo type erasure

```
public class Pair{
    public Pair(Object primoEl, Object secondoEl)
    {
        primo = primoEl;
        secondo = secondoEl;
    }
    public Object getFirst() { return primo; }
    public Object getSecond() { return secondo; }
    private Object primo;
    private Object secondo;
}
```

- `Pair` è il tipo grezzo corrispondente a `Pair<S, T>`



Classe PairTester dopo type erasure

```
public class PairTester {  
    public static void main(String[] args) {  
        Pair p = new Pair(3.0,3);  
  
        double x = (Double) p.getFirst();  
        int y = (Integer) p.getSecond();  
    }  
}
```

- Cancellazione dei tipi generici e aggiunta di operatori casting in maniera opportuna



Metodo `min` dopo type erasure

```
public static Comparable min(Comparable[] a) {  
    Comparable smallest = a[0];  
    for (int i=0; i<a.length;i++)  
        if (a[i].compareTo(smallest)<0)  
            smallest = a[i];  
    return smallest;  
}
```

- **Comparable** è il tipo meno generale che è compatibile con tutti i tipi che possono essere utilizzati con il metodo



Decompilazione bytecode

- Esistono dei tool per risalire dal bytecode al codice sorgente
 - Ad esempio JAD, DJ Java Decompiler, JReverse Pro
- Si possono verificare gli effetti della type erasure seguendo questi passi:
 - Scrivere un sorgente con generics
 - Generare il bytecode
 - Decompilare



Esercizio

- Modificare la classe `Pair` come segue
 - Il tipo dei due valori deve essere lo stesso
 - Aggiungere un metodo `swap` che scambia i due

```
public class Pair<T>
{
    public Pair(T firstElement, T secondElement)
    {
        first = firstElement;
        second = secondElement;
    }

    public T getFirst()
    {
        return first;
    }

    public T getSecond()
    {
        return second;
    }
}
```

```
public void swap()
{
    T temp = first;
    first = second;
    second = temp;
}

private T first;
private T second;
}
```