

JAVA GENERICS

Pietro Di Lena

Università di Bologna

Credits: basate su slide create dal Prof. Angelo Di Iorio

Tipi parametrici

- In diversi casi è utile definire classi che operano su tipi di dato diverso e implementano un comportamento generalizzabile
- Ad esempio, si vuole progettare un classe per una coppia ordinata di oggetti, su cui è possibile:
 - Leggere/scrivere il primo o il secondo oggetto
 - Scambiare di posizione gli oggetti
- Questi comportamenti (metodi) non dipendono dal tipo specifico di dato ma si possono generalizzare
- Possono essere cioè **parametrizzati** rispetto ad un **tipo generico**

Java Generics

- Java (dalla versione 5) permette di dichiarare classi e metodi *generici* che possono *operare* su *tipi di dato diversi*
- E' possibile definire classi parametriche rispetto a *formal type parameters* che possono essere usati come tipi dichiarati per variabili, parametri e valori di ritorno
- La definizione di una classe generica si esprime con la sintassi: `ClassName<T>`
- `T` è un tipo utilizzabile nella classe e nei suoi metodi
- Quando un programmatore usa la classe, deve specificare esplicitamente il tipo `T`

```
public class Coppia<T> {  
    private T primo;  
    private T secondo;
```

Generics

NON esiste la definizione
della classe T

```
    public Coppia(T primo, T secondo) {  
        this.primo = primo;  
        this.secondo = secondo;  
    }
```

```
    public T getPrimo() { return primo;}
```

```
    public T getSecondo() { return secondo;}
```

```
    public void inverti(){  
        T tmp = this.primo;  
        this.primo = this.secondo;  
        this.secondo = tmp;  
    }
```

```
    @Override
```

```
    public String toString() {  
        return "Coppia [primo="+primo+",secondo="+secondo+"]";  
    }
```

```
}
```

classe Coppia

Uso di generics

```
public class CoppiaTest {  
    public doTest() {  
        Coppia<Integer> ci = new Coppia<Integer>(12, 30);  
        Coppia<String> cs = new Coppia<String>("abc", "def");  
  
        System.out.println(ci);  
        ci.inverti();  
        cs.inverti();  
        System.out.println(ci);  
        System.out.println(cs);  
    }  
}
```

Coppia di Interi

Coppia di Stringhe

classe TestCoppia

Tipi parametrici

- Si può usare un qualsiasi identificativo per il tipo parametrico (tranne ovviamente le keyword del linguaggio)
- Per convenzione i tipi parametrici iniziano con una lettera maiuscola e solitamente si usa una singola lettera
- Un tipo parametrico può essere usato in qualunque posizione in cui si può usare un tipo di classe
- Può essere sostituito anche con le classi definite dal programmatore, non solo con `Integer`, `String`, ecc.

Tipi parametrici multipli

- La definizione di una classe generica può avere anche più di un tipo parametrico
- Si usa la stessa sintassi, separando i tipi con una virgola
- Esempio: scriviamo il codice di una classe generica, `CoppiaMista`, per gestire una coppia di oggetti eventualmente di tipo diverso
- La classe `CoppiaMista`, oltre al costruttore, espone un metodo per verificare i tipi:
 - `checkOmogenei()`: ritorna `true` se i due oggetti sono dello stesso tipo, `false` altrimenti

```
public class CoppiaMista<A, B> {  
    protected A primo;  
    protected B secondo;  
  
    public CoppiaMista(A primo, B secondo) {  
        this.primo = primo;  
        this.secondo = secondo;  
    }  
  
    public A getPrimo() { return primo;}  
  
    public B getSecondo() {return secondo;}  
  
    public Boolean checkOmogenei(){  
        return (this.primo.getClass() == this.secondo.getClass());  
    }  
}
```

classe CoppiaMista

Precisazioni

- Un tipo generico NON può essere usato insieme a `new` per
 - creare un nuovo oggetto
 - allocare memoria per un array
- Versioni più recenti di Java permettono di inferire il tipo parametrico in una chiamata al costruttore e non richiedono di specificarlo
- Entrambe le seguenti dichiarazioni sono valide:

```
Coppia<Integer> coppia = new Coppia<>();
```

```
Coppia<Integer> coppia = new Coppia<Integer>();
```

Vincoli sui tipi parametrici

- Un tipo parametrico può essere sostituito con qualunque tipo
- In molte situazioni tuttavia è utile imporre dei vincoli sui tipi che sostituiranno il tipo parametrico, per evitare errori e comportamenti non attesi
- Immaginiamo ad esempio di volere aggiungere alla classe `Coppia<T>` (di oggetti omogenei) un metodo per indicare se il primo oggetto è “minore” del secondo
- Come garantire che gli oggetti siano confrontabili?

```
public class Coppia<T extends Comparable<T>> {  
    private T primo;  
    private T secondo;
```

Vincolo

```
    public Coppia(T primo, T secondo) {  
        this.primo = primo;  
        this.secondo = secondo;  
    }
```

```
    public T getPrimo() { return primo;}
```

```
    public T getSecondo() { return secondo;}
```

```
    ... // metodi inverti() e toString() omessi
```

```
    public T minore(){  
        if (primo.compareTo(secondo) <= 0)  
            return primo;  
        else  
            return secondo;  
    }
```

Sicuramente
implementato in T

```
}
```

classe Coppia

Tipi generici ed `extends`

- La keyword `extends` permette di imporre che la classe che sostituisce il tipo generico deve implementare l'interfaccia `Comparable`
- **Attenzione: si usa `extends` e non `implements`**
- Questo perché è possibile esprimere un vincolo anche in termini di una classe e non solo di un'interfaccia. In questo esempio solo le sottoclassi di `Person` possono essere usate con la classe generica `Coppia`

```
public class Coppia<T extends Person>
```

- **Se il vincolo imposto da `extends` non è soddisfatto Java produce un errore in fase di compilazione**

Esercizio

- Implementiamo in Java una classe generica per gestire una coppia di oggetti (anche di tipo diverso) **colorati**
- La classe espone i seguenti metodi:
 - Costruttore: prende in input i due oggetti
 - `coloraTutti(Color c)`: setta ad entrambi gli oggetti il colore `c`
- La classe estende la classe `CoppiaMista<A, B>` vista in precedenza

classe `CoppiaMistaColorabile`

```
public class CoppiaMista<A, B> {  
    protected A primo;  
    protected B secondo;  
  
    public CoppiaMista(A primo, B secondo) {  
        this.primo = primo;  
        this.secondo = secondo;  
    }  
  
    public A getPrimo() { return primo;}  
  
    public B getSecondo() {return secondo;}  
  
    public Boolean checkOmogenei(){  
        return (this.primo.getClass() == this.secondo.getClass());  
    }  
}
```

classe CoppiaMista

Interfaccia IColorabile

```
public interface IColorabile {  
  
    public void setColor(Color c);  
  
    public Colore getColor();  
  
}
```

package colori

Coppia Mista Colorabile

```
public class CoppiaMistaColorabile<A extends IColorabile, B
extends IColorabile> extends CoppiaMista<A,B>{

    public CoppiaMistaColorabile(A primo, B secondo) {
        super(primo,secondo);
    }

    public void coloraTutti(Color c){
        this.primo.setColor(c);
        this.secondo.setColor(c);
    }

}
```

Sicuramente
implementato
in A e B

classe CoppiaMistaColorabile