Marco Faella

Classi e metodi parametrici

Lezione n. 13

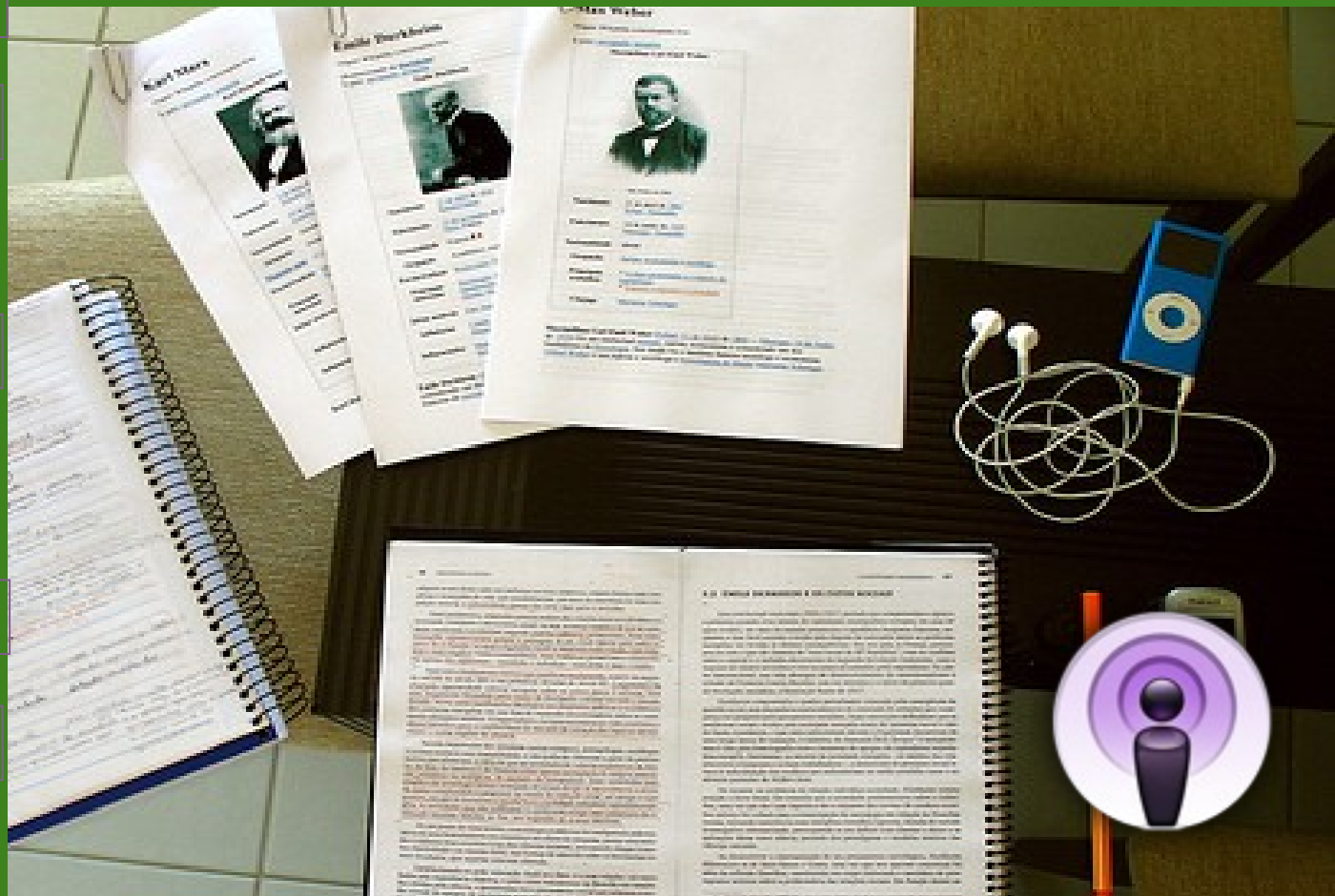
Parole chiave:
Java

Corso di Laurea:
Informatica

Insegnamento:
Linguaggi di Programmazione II

Email Docente:
faella.didattica@gmail.com

A.A. 2009-2010



La programmazione parametrica

- La versione 1.5 ha introdotto una nuova, importante funzionalità nel linguaggio: la programmazione parametrica, in Inglese anche detta *generics*

- Si tratta della possibilità di dotare classi, interfacce e metodi di *parametri di tipo*
- Simili ai normali parametri dei metodi, questi parametri hanno come possibili valori i tipi (non primitivi) del linguaggio

- Questo meccanismo consente di scrivere codice più robusto dal punto di vista dei tipi di dati, evitando in molti casi il ricorso alle conversioni forzate (cast)

- Questa funzionalità è una forma di *polimorfismo parametrico*

Esempio: la classe Pair

- La programmazione parametrica dimostra tutta la sua utilità nella realizzazione di *collezioni*, ovvero classi deputate a contenere altri oggetti
- Quindi, come primo esempio supponiamo di voler realizzare una classe, chiamata Pair, che rappresenta una coppia di oggetti dello stesso tipo
- In mancanza della programmazione parametrica (ad esempio, in Java 1.4) la classe si sarebbe dovuta realizzare secondo il seguente schema:

```
class Pair {
     private Object first, second;
     public Pair(Object a, Object b) { ... }
     public Object getFirst() { ... }
     public void setFirst(Object a) { ... }
     ...
}
```

- Un'implementazione come questa comporta che gli utenti della classe debbano necessariamente ricorrere al **cast** perché gli elementi estratti dalla coppia riacquistino il loro tipo originario, come nel seguente esempio:

```
Pair p = new Pair(“uno”, “due”);
String a = (String) p.getFirst();
```

La versione parametrica di Pair

- Vediamo ora come ovviare a questo problema rendendo la classe Pair parametrica:

```
class Pair<T> {
     private T first, second;
     public Pair(T a, T b) {
          first = a;
          second = b;
     }
     public T getFirst() { return first; }
     public void setFirst(T a) { first = a; }
     ...
}
```

- In questo caso, la classe Pair ha un parametro di tipo, chiamato T
- I parametri di tipo vanno dichiarati dopo il nome della classe, racchiusi tra parentesi angolari
- Se vengono dichiarati più parametri di tipo, questi vanno separati da virgole
- All'interno della classe, un parametro di tipo si comporta **come un tipo di dati vero e proprio**, tranne che per alcune eccezioni che vedremo in seguito
- In particolare, come si vede dall'esempio, un parametro di tipo si può usare come tipo di un campo, tipo di un parametro formale di un metodo e tipo di ritorno di un metodo

La versione parametrica di Pair

- La nuova versione di Pair permette agli utenti della classe di specificare di che tipo di coppia si tratta e, così facendo, di evitare i cast:

```
Pair<String> p = new Pair<String>(“uno”, “due”);
String a = p.getFirst();
```

- Nella dichiarazione della variabile “p”, è **obbligatorio** indicare il parametro attuale di tipo

- Nell'instanziazione dell'oggetto Pair, tale indicazione è **facoltativa** (in questo contesto)

```
Pair<String> p = new Pair<>(“uno”, “due”);
String a = p.getFirst();

Pair<Employee> q = new Pair<>(…,…);
Employee e = q.getFirst();
```

La sintassi <> si chiama “operatore diamond”

- Come per i normali parametri dei metodi, “String” è il **parametro attuale**, che prende il posto del **parametro formale** “T” di Pair

La versione grezza delle classi parametriche

- Per compatibilità con le versioni precedenti di Java, è possibile usare una classe parametrica come se non lo fosse
- Quando utilizziamo una classe parametrica senza specificare i parametri di tipo, si dice che stiamo usando la versione *grezza* di quella classe

Perché è stata data questa possibilità?

- Per *retro-compatibilità*
- Con l'introduzione dei generics, molte classi della libreria standard Java sono diventate parametriche
- La versione grezza di queste classi permette alla nuova versione della libreria standard di essere compatibile con i programmi scritti con le versioni precedenti del linguaggio

La versione grezza delle classi parametriche

- Ad esempio, se Pair è la classe parametrica descritta nelle slide precedenti, è anche possibile utilizzarla così:

```
Pair p = new Pair(“uno”, “due”);
String a = (String) p.getFirst();
```

- La prima riga provoca un warning in compilazione
- Il cast nella seconda riga è indispensabile

- Le classi grezze esistono solo per retro-compatibilità
  - il codice nuovo **dovrebbe sempre specificare** i parametri di tipo delle classi parametriche

La classe Pair con due parametri di tipo

- Esaminiamo anche un'ulteriore versione di Pair, in grado di contenere due oggetti di tipo diverso
- In questo caso, la classe avrà **due parametri di tipo**, che rappresentano rispettivamente il tipo del primo e del secondo elemento della coppia

```
class Pair<T, U> {
     private T first;
     private U second;
     public Pair(T a, U b) {
          first = a;
          second = b;
     }
     public T getFirst() { return first; }
     public void setFirst(T a) { first = a; }
     public U getSecond() { return second; }
     public void setSecond(U a) { second = a; }
}

Pair<String,Employee> p = new Pair<>(“Pippo”, new Employee(…));
```

Esercizio:

Scrivere un metodo statico *getMedian* che accetta un array qualsiasi e restituisce **l'elemento mediano (di posto intermedio)**

Metodi parametrici

- Si dice che una classe è parametrica se ha almeno un parametro di tipo
- Anche i **singoli metodi** e costruttori possono avere parametri di tipo, indipendentemente dal fatto che la classe cui appartengono sia parametrica o meno
- Un caso tipico è rappresentato dai metodi statici
  - i metodi statici non possono utilizzare i parametri di tipo della classe in cui sono contenuti

- Il seguente metodo parametrico restituisce l'elemento mediano (di posto intermedio) di un dato array

```
public static <T> T getMedian(T[] a) {
     int l = a.length;
     return a[l/2];
}
```

- Come si vede, il parametro di tipo va dichiarato **prima del tipo restituito**, racchiuso tra parentesi angolari
- Questo parametro è **visibile solo all'interno del metodo**
- In questo caso, il parametro di tipo permette di restituire un oggetto dello stesso tipo dell'array ricevuto come argomento

Invocare metodi parametrici

- Quando si invoca un metodo parametrico, è opportuno, ma non obbligatorio, specificare il parametro di tipo attuale per quella chiamata

- Ad esempio, supponendo che il metodo getMedian della slide precedente appartenga ad una classe Test, lo si può invocare così:

```
String[] x = {“uno”, “due”, “tre”};
String s = Test.<String>getMedian(x);
```

- Il parametro attuale di tipo va quindi indicato tra il punto e il nome del metodo
- E' anche possibile omettere il parametro attuale di tipo
- In questo caso, il compilatore cercherà di **dedurre** il tipo più appropriato, mediante un meccanismo chiamato *type inference* (inferenza di tipo)

- Per sommi capi, la type inference cerca di individuare *il tipo più specifico che rende la chiamata corretta*

- L'algoritmo di type inference non è né corretto né completo, ovvero:
  - può fallire anche quando una soluzione esiste
  - può individuare una soluzione anche quando questa non esiste; in questo caso, la soluzione individuata sarà segnalata come errore dal type checking

- Le regole precise che il compilatore adotta nella type inference esulano dagli scopi di questo corso
- Ci limiteremo ad esaminare alcuni esempi

- Con riferimento al metodo getMedian, è possibile invocarlo senza specificare il parametro attuale di tipo

```
String[] x = {“uno”, “due”, “tre”};
String s = Test.getMedian(x);
```

- Il compilatore **dedurrà correttamente** che il tipo desiderato è String

Esempio di type inference

- Consideriamo invece il seguente metodo, che assegna il medesimo riferimento a tutte le celle di un array dato

```
public static <T> void fill(T[] a, T x) {
   for (int i=0; i<a.length; i++)
      a[i] = x;
}
```

- L'intenzione del programmatore è chiaramente quella di accettare un array e un oggetto dello stesso tipo, come nel seguente esempio

```
String[] a = new String[10];
fill(a, “ciao”);
```

- In questo caso, il compilatore, durante la fase di type inference, dedurrà correttamente che il tipo desiderato è String
- La prossima slide mostra che la type inference può anche offrire risultati inattesi e **indesiderati**

Esempio di type inference

- Con riferimento al metodo fill della slide precedente, consideriamo il seguente frammento

```
Employee[] a = new Employee[10];
fill(a, new Integer(100));
```

Esempio di type inference

- Con riferimento al metodo fill della slide precedente, consideriamo il seguente frammento

```
Employee[] a = new Employee[10];
fill(a, new Integer(100));
```

- Siccome l'array e l'oggetto passato sono di due tipi differenti, ci aspetteremmo che la type inference **fallisca**, producendo un errore di compilazione

- Invece, la compilazione **va a buon fine**, mentre l'esecuzione del programma produce il lancio di un'**eccezione**
  - la compilazione termina con successo perché la type inference deduce che il tipo richiesto è Object
  - in base alle regole di sottotipo, il tipo Object renderebbe **valida** l'invocazione incriminata
  - l'errore in esecuzione è dovuto al fatto che gli array conservano il tipo con il quale sono stati creati (in questo caso, Employee)
  - ogni scrittura nell'array controlla che l'oggetto assegnato sia di un tipo compatibile con quello dell'array

- Per non incorrere in tali sorprese, ribadiamo il consiglio di **specificare il tipo desiderato** in tutte le invocazioni di metodi parametrici

Costruttori parametrici

- Anche i costruttori possono essere parametrici, indipendentemente dal fatto che la loro classe sia parametrica o meno

```
public class A<T> {
   public <U> A(T x, U y) {
      ...
   }
}
```

- In quest'esempio, il costruttore della classe parametrica A ha a sua volta un parametro di tipo chiamato U
- Mentre il parametro T è visibile in tutta la classe A, il parametro U è visibile solo all'interno di quel costruttore
- Il costruttore in questione può essere invocato con la seguente sintassi

```
A<String> a = new <Integer>A<String>(“ciao”, new Integer(100));
```

- Il parametro di tipo del costruttore (Integer) va specificato prima del nome della classe
- Il parametro di tipo della classe, come abbiamo già visto per la classe Pair, va specificato dopo il nome della classe