Programmazione Java
Variabili membro, Metodi
La parola chiave final

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Sommario

» Variabili membro

» Metodi

» La parola chiave `final`

Variabili membro

» Sintassi

```
[modificatoriAccesso] [static|final|transient|volatile]*
                        tipo      nome

modificatoriAccesso = [ public |  private | protected ]
```

» Non possono apparire più di una volta

» Esempio

- `public static final String HELLO = "Hello";`

Metodi

» Sintassi

```
[modificatoriAccesso]
     [ static | abstract | final | native | synchronized ]*
          tipoRitorno nomeMetodo(listaparametri)
                    [ throws exceptions]

modificatoriAccesso = [ public |  private | protected ]
```

» Non possono apparire più di una volta

» Esempio

```
public static final String getHello() {
  return "Hello World";
}
```

La parola chiave `final`

- Ha significati diversi a seconda del contesto, ma in generale dice: “Questo non può essere cambiato”
- Può essere utilizzato in tre casi:
  - Dati
  - Metodi
  - Classi

La parola chiave `final` > Dati

- Modo per dire al compilatore che un dato è costante
- Una costante è utile per due ragioni:
  - Può essere una costante nota in fase di compilazione che non cambierà mai
    - Il compilatore può scrivere direttamente il valore costante in qualsiasi calcolo dove esso è utilizzato
  - Può essere un valore inizializzato in fase di esecuzione che non si vuole cambiare

La parola chiave `final` > Dati

- Indica una variabile (di istanza o di classe) che può essere assegnata una sola volta ovvero **costante**
- Una volta assegnata non può essere più modificata
  - Tipo reference non può essere modificato il riferimento
  - Valori all'interno dell'oggetto possono essere modificati (Java non fornisce un modo per rendere costante un oggetto arbitrario. Questa limitazione comprende anche gli array)
- Generalmente le costanti scritte in maiuscolo
- Generalmente vengono utilizzate in congiunzione con parola chiave `static`
- `const` parola chiave riservata ma non utilizzata

La parola chiave `final` > Dati


```
public class FinalData {

  private static Random rand = new Random();
  private String id;
  public FinalData(String id) { this.id = id; }

  // Can be compile-time constants:
  private final int VAL_ONE = 9;
  private static final int VAL_TWO = 99;
  // Typical public constant:
  public static final int VAL_THREE = 39;
  // Cannot be compile-time constants:
  private final int i4 = rand.nextInt(20);
  static final int i5 = rand.nextInt(20);
  private Value v1 = new Value(11);
  private final Value v2 = new Value(22);
  private static final Value v3 = new Value(33);
  // Arrays:
  private final int[] a = { 1, 2, 3, 4, 5, 6 };
  public String toString() {
    return id + ": " + "i4 = " + i4 + ", i5 = " + i5;
  }
```

Entrambi possono essere utilizzati come costanti in fase di compilazione

...

La parola chiave `final` > Dati


```
public class FinalData {

  private static Random rand = new Random();
  private String id;
  public FinalData(String id) { this.id = id; }

  // Can be compile-time constants:
  private final int VAL_ONE = 9;
  private static final int VAL_TWO = 99;
  // Typical public constant:
  public static final int VAL_THREE = 39;
  // Cannot be compile-time constants:
  private final int i4 = rand.nextInt(20);
  static final int i5 = rand.nextInt(20);
  private Value v1 = new Value(11);
  private final Value v2 = new Value(22);
  private static final Value v3 = new Value(33);
  // Arrays:
  private final int[] a = { 1, 2, 3, 4, 5, 6 };
  public String toString() {
    return id + ": " + "i4 = " + i4 + ", i5 = " + i5;
  }
```

Modo tipico di definire costanti, in questo modo è possibile utilizzarle anche fuori dal package. `Static` per mettere in rilievo che ce n'è una sola e `final` per dire che è una costante

...

La parola chiave `final` > Dati


```
public class FinalData {

  private static Random rand = new Random();
  private String id;
  public FinalData(String id) { this.id = id; }

  // Can be compile-time constants:
  private final int VAL_ONE = 9;
  private static final int VAL_TWO = 99;
  // Typical public constant:
  public static final int VAL_THREE = 39;
  // Cannot be compile-time constants:
  private final int i4 = rand.nextInt(20);
  static final int i5 = rand.nextInt(20);
  private Value v1 = new Value(11);
  private final Value v2 = new Value(22);
  private static final Value v3 = new Value(33);
  // Arrays:
  private final int[] a = { 1, 2, 3, 4, 5, 6 };
  public String toString() {
    return id + ": " + "i4 = " + i4 + ", i5 = " + i5;
  }
```

- Il fatto di essere `final` non significa essere noto in fase di compilazione
- Tale differenza si ha solo quando i valori vengono inizializzati in fase di esecuzione

...

La parola chiave `final` > Dati


```
public class FinalData {

  private static Random rand = new Random();
  private String id;
  public FinalData(String id) { this.id = id; }

  // Can be compile-time constants:
  private final int VAL_ONE = 9;
  private static final int VAL_TWO = 99;
  // Typical public constant:
  public static final int VAL_THREE = 39;
  // Cannot be compile-time constants:
  private final int i4 = rand.nextInt(20);
  static final int i5 = rand.nextInt(20);
  private Value v1 = new Value(11);
  private final Value v2 = new Value(22);
  private static final Value v3 = new Value(33);
  // Arrays:
  private final int[] a = { 1, 2, 3, 4, 5, 6 };
  public String toString() {
    return id + ": " + "i4 = " + i4 + ", i5 = " + i5;
  }
```

In questo caso emerge la differenza tra `final static` e solo `final`

- il valore di `i4` cambia per ogni istanza di `FinalData`
- Il valore di `i5` rimane sempre uguale a quello dell'inizializzazione avvenuta con la creazione della prima istanza di `FinalData`

...

La parola chiave `final` > Dati


```
  …
  public static void main(String[] args) {

    FinalData fd1 = new FinalData("fd1");

    //! fd1.VAL_ONE++; // Error: can't change value
    fd1.v2.i++; // Object isn't constant!
    fd1.v1 = new Value(9); // OK -- not final
    for(int i = 0; i < fd1.a.length; i++)
      fd1.a[i]++; // Object isn't constant!
    //! fd1.v2 = new Value(0); // Error: Can't
    //! fd1.v3 = new Value(1); // change reference
    //! fd1.a = new int[3];
    System.out.println(fd1);
    System.out.println("Creating new FinalData");
    FinalData fd2 = new FinalData("fd2");
    System.out.println(fd1);
    System.out.println(fd2);

  }

}
```

- `v2` è `final` ma il suo valore (l'oggetto Value) può essere cambiato
- Non si può però cambiare il riferimento `v2` ad un nuovo oggetto
- Stessa cosa anche per gli array

(Vedere FinalData.java)

La parola chiave `final` > Dati

- E’ possibile assegnare il valore della variabile **NON** contestualmente alla dichiarazione (*blank final*)
- E’ necessario inizializzare le variabili final prima di poter essere utilizzato
  - Variabile di istanza
    - E’ necessario inizializzarla in tutti i costruttori oppure all’interno di un inizializzatore di istanza
  - Variabile di classe
    - E’ necessario inizializzarla all’interno di un inizializzatore statico
- Un errore in fase di compilazione viene restituito in caso contrario

La parola chiave `final` > Dati

- I *blank final* forniscono molta più flessibilità nell'utilizzo
  - Ad esempio un campo final di una classe può così essere diverso per ciascun oggetto

(Vedere BlankFinal.java)

La parola chiave `final` > Dati


```
class Poppet {

  private int i;
  Poppet(int ii) { i = ii; }

}

public class BlankFinal {
  private final int i = 0;    // Initialized final
  private final int j;        // Blank final
  private final Poppet p;     // Blank final reference

  // Blank finals MUST be initialized in the constructor:
  public BlankFinal() {
    j = 1;                    // Initialize blank final
    p = new Poppet(1);        // Initialize blank final reference
  }

  public BlankFinal(int x) {
    j = x;                    // Initialize blank final
    p = new Poppet(x);        // Initialize blank final reference
  }
  public static void main(String[] args) {
    new BlankFinal();
    new BlankFinal(47);
  }

  }
```

(vedere BlankFinal.java)

La parola chiave `final` > Dati


```
public class Constants {

  public static void main(String[] args) {
      final double CM_PER_INCH = 2.54;
      System.out.println("Centimetri per inch: " + CM_PER_INCH);
      CM_PER_INCH = 10.30;  //ERRORE IN COMPILAZIONE
  }

  //OPPURE
  public static final double CM_PER_INCH = 2.54;

}
```

La parola chiave `final` > Dati


```
class Point {
        int x,y;
        final Point root;
        static final Point ORIGIN;

        static {
                ORIGIN = new Point(0,0);
        }

        public Point(int x, int y ){
                this.x = x;
                this.y = y;
                root = null;
        }
       public Point(int x, int y, Point root ){
                this.x = x;
                this.y = y;
                this.root = root;
        }
}
```

La parola chiave `final` > Dati


```
class Test {

   public static void main(String[] args) {
      Point p1 = new Point(100,100);
      Point p2 = new Point(200,200,p1);

      p1.root = new Point(300,300); //ERRORE
      p2.root.x = 300;              //OK
      System.out.println("ORIGIN.x:" + Point.ORIGIN.x);

      Point.ORIGIN = new Point(1000, 1000); //ERRORE
      Point.ORIGIN.x = 400;                 //OK
   }
}
```

La parola chiave `final` > Argomenti

- Java permette di rendere final gli argomenti dichiarandoli come tali nell'elenco degli argomenti
- All'interno del metodo si può cambiare ciò a cui punta il riferimento all'argomento

(vedere FinalArguments.java)

La parola chiave `final` > Argomenti


```
class Gizmo {
    public void spin() {}
  }

public class FinalArguments {
  void with(final Gizmo g) {
  //! g = new Gizmo(); // Illegal -- g is final
  }

  void without(Gizmo g) {
   g = new Gizmo(); // OK -- g not final
   g.spin();
  }
  // void f(final int i) { i++; } // Can't change


  // You can only read from a final primitive:
  int g(final int i) { return i + 1; }
  public static void main(String[] args) {
      FinalArguments bf = new FinalArguments();
      bf.without(null);
      bf.with(null);
    }
  }
```

(vedere FinalArguments.java)

La parola chiave `final` > Metodi

» Un metodo può essere definito final:

- Per evitare che una classe che eredita ne cambi il significato
- Per ragioni di efficienza

```
public class A {
   final void m1(){}
   void m2(){}
   }


public class B extends A{
   //! void m1() {} Illegal
   void m2() {}

}
```

La parola chiave `final` > Metodi

» Tutti i metodi private di una classe sono implicitamente final

– Poiché non si può accedere ad un metodo private, non si può ridefinirlo

(Vedere FinalOverridingIllusion.java)

La parola chiave `final` > Classi

- Definendo una class final si stabilisce che non si vuole ereditare da quella classe o permettere a qualcun altro di farlo

- I campi di una classe final possono essere o non essere final

- Poiché con final si impedisce l'ereditarietà, tutti I metodi di una classe final sono implicitamente final, dato che non c'è modo di ridefinirli

(Vedere Jurassic.java)