Linguaggi di Programmazione

Programmazione ad Oggetti

Note

Le seguenti slide introducono la programmazione ad oggetti utilizzando uno pseudocodice al fine di porre l'attenzione sugli aspetti concettuali della programmazione ad oggetti piuttosto che sulla sintassi di uno specifico linguaggio di programmazione.

Classi

Esempio di classe:

```
Classe Rettangolo {
     intero base, altezza
     void impostaBase (intero b) {
          base = b
     }

     void impostaAltezza (intero a) {
          altezza = a
     }
     intero area() {
          return base * altezza
     }
     intero stampa () {
          stampa(«la base è», base, « - l'altezza è», altezza)
          }
}
```

Classi

- Una **classe** definisce uno schema di **oggetti**
- In una definizione di classe troviamo due componenti:
  - **Attributi** (detti anche *campi* o *variabili di istanza*)
  - **Metodi**
- Definita la classe rettangolo è possibile creare **oggetti** di tipo rettangolo, detti **istanze della classe**.

Oggetti

```
Rettangolo r
R = nuovo Rettangolo()
r.impostaBase(4)
r.impostaAltezza(2)
r.stampa()
```

L’esecuzione darà:

la base è 4 – l’altezza è 2

Oggetti, attributi e metodi

- Ogni oggetto è dotato di un proprio **stato**
- Lo stato non dovrebbe essere visibile dall'esterno ma modificato e ispezionato attraverso i metodi
- Gli **attributi** possono essere **nascosti** o **visibili**
- La notazione `r.impostaBase()` evidenzia che il metodo *appartiene* all'oggetto *rivevitore* r

Universo di oggetti

Durante l'esecuzione si ha un universo di oggetti in evoluzione.

L'universo può cambiare per tre ragioni:

- Creazione di nuovi oggetti
- Modifica dello stato di oggetti esistenti
- Cancellazione di oggetti esistenti

Uguaglianza tra oggetti

```
Rettangolo r1,r2,r3
r1 = nuovo Rettangolo
r2 = nuovo Rettangolo
r3 = r1
```

Dopo l'esecuzione esistono due oggetti di tipo Rettangolo.

r1 e r3 denotano lo stesso rettangolo

Uguaglianza tra oggetti

Nei linguaggi OO (object-oriented) si distinguono due diverse nozioni di uguaglianza:

1. Due espressioni sono uguali se e solo se denotano lo stesso oggetto
2. Due oggetti sono uguali se e solo se lo sono in qualche modo i loro stati

La seconda uguaglianza deve essere implementata dall'utente.

Uguaglianza tra oggetti

L'uguaglianza tra gli stati può essere implementata in diversi modi:

1. Due oggetti sono uguali se i loro campi corrispondenti sono identici, cioè denotano lo stesso valore (se di tipo primitivo) o oggetto
2. Due oggetti sono uguali se i loro campi corrispondenti denotano lo stesso valore (se di tipo primitivo) o oggetti *ricorsivamente* uguali.

Copia di oggetti

r3 = r1

Dopo l'assegnazione r3 e r1 denotano lo stesso oggetto.

Se si vuole creare una copia di un oggetto esistente è necessario utilizzare un metodo opportuno.

Copia di oggetti

```
Rettangolo copia(){
    Rettangolo r = nuovo rettangolo()
    r.base = base
    r.altezza = altezza
    return r
}

r3 = r1.copy()
```

Copia di oggetti

Due tipi di copia:

1. **Shallow copy**: nuovo oggetto i cui campi denotano gli stessi oggetti denotati dai campi dell'oggetto ricevitore (una shallow copy è uguale in senso (1) al ricevitore)
2. **Deep copy**: nuovo oggetto i cui campi sono ricorsivamente deep copy dei campi corrispondenti dell'oggetto ricevitore (una deep copy è uguale in senso (2) al ricevitore)

Clientship e ricorsione tra classi

Un oggetto può avere campi che sono a loro volta oggetti.

Si dice che una classe A è cliente di (o usa) una classe B se il codice di A utilizza B come tipo o come generatore di oggetti.

La relazione di clientship può essere direttamente o mutuamente ricorsiva.

Visibilità delle componenti

Almeno due livelli di visibilità:

- pubblica
- privata

Due possibili modi di intendere la visibilità *privata*:

- per classe
- per oggetto

Visibilità per classe e per oggetto

Visibilità per oggetto:

un oggetto ha accesso a tutte le proprie componenti ma non può accedere a componenti private di oggetti della sua stessa classe.

Visibilità per classe:

non vi è modo di nascondere alcuna componente ad oggetti della stessa classe.

Uso di *this*, *self*, …

Nei linguaggio OO esiste una parola chiave (*this*, *self*, *current*, …) che denota, durante l'esecuzione di un metodo, l'oggetto su cui è stato invocato il metodo.

Alcuni linguaggi permettono di riferirsi a tale oggetto anche implicitamente.

Uso di *this*, *self*, …

```
booleano eUguale(rettangolo r){
    return altezza == r.altezza AND base == r.base
}
```

Equivale a:

```
booleano eUguale(rettangolo r){
    return this.altezza == r.altezza AND this.base == r.base
}
```

Uso di *this*, *self*, …

In alcuni casi è necessario un utilizzo esplicito:

```
Classe Rettangolo {
      intero base, altezza
      void impostaBase (intero base) {
            this.base = base
      }

      void impostaAltezza (intero altezza) {
            this.altezza = altezza
      }
      intero area() {
            return base * altezza
      }
      intero stampa () {
            stampa(«la base è», base, « - l'altezza è», altezza)
            }
      Rettandolo areaMaggiore (Rettangolo r) {
            if (area() > r.area()) return this
            else return r
      }
}
```

Costruttori

I costruttori permettono di inizializzare gli oggetti. Vengono invocati al momento della creazione di un oggetto.

```
Classe Rettangolo {
    intero base, altezza
    …
    Rettangolo (intero a, intero b)
        altezza = a
        base = b
    }
}

Rettangolo r = new Rettangolo (5,8)
```