Riuso di Codice in Java e in OCaml

Sommario: 18 Maggio, 2016

- Ragioni e Contesti di Riuso
- Meccanismi di Riuso e Riuso Sistematico
- Rivediamo alcuni Lucidi
- Studio di un Caso

Ragioni e Contesti di Riuso

- **Ragioni**
  - Risparmiare Tempo e Risorse nello Sviluppo
  - Ridurre "duplicati" ed errori

- **Contesti**
  - Riuso Interno: Sviluppatori riusano, estendono, riconfigurano propri componenti
    - Sia, durante lo sviluppo vero e proprio
    - Sia, terminato lo sviluppo, durante:
      - Manutenzione
      - Aggiornamento e Ri-Adattamento
  - Riuso Esterno: Sviluppatori usano componenti di terze-parti

  - Interessati al **Riuso Interno**

Meccanismi di Riuso e Riuso Sistematico

- Librerie, API, Framework:
  - Risorse già pronte per applicazioni "generiche"
- Procedure, Funzioni, Moduli, Oggetti, e Classi:
  - forniscono componenti
  - integrabili e configurabili
  - attraverso le interfacce di uso:
    - trasmissioni, import-export,
    - ereditarietà e gerarchia
- Metodologie che supportano riuso:
  - Higher Order Programming
  - Generic Programming
  - Inheritance Programming

Principi: Riuso di Codice

Quando la realizzazione di un sistema A è stata completata e la sua esecuzione è a regime, la scrittura del codice di A entra in una nuova fase: La manutenzione.

- È una fase di programmazione molto impegnativa e costosa.
- **Riuso** significa contenere il codice che deve essere prodotto e gli errori conseguenti
- Nell'esempio sotto, A è esteso con una nuova classe di contatori che riusano il codice dei contatori già presenti, ma

```
public class NewCounter extends AbstractCounter{
/* Classe NewCounter per contatori AbstractCounter con operazione
   alive che indica quanti NewCounter sono stati prodotti.
*/
    private static int counterCount = 0;
    public static int alive(){return counterCount;}
//metodi
    public NewCounter(){counterCount++;}
}
```

- sono in grado di dire quanti contatori sono in uso.

Principi: Riuso di Codice/2

- A è esteso con una nuova classe di contatori che riusano il codice dei contatori già presenti, ma hanno delle proprietà aggiuntive

```
public class NewCounter extends AbstractCounter{
/* Classe NewCounter per contatori AbstractCounter con operazione
   alive che indica quanti NewCounter sono stati prodotti.
*/
    private static int counterCount = 0;
    public static int alive(){return counterCount;}
//metodi
    public NewCounter(){counterCount++;}
}
```

- **vecchie e nuove parti** di A **convivono perfettamente**.

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter e NewCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
        {//uso di NewCounter
            NewCounter x;
            for (int i=0; i+myCounterA.get()<10; i++) {//i due tipi di contatore usati insiemee New
                NewCounter y = new NewCounter();
                x=y;
                x.inc();
            }
        }
        System.out.println("sono stati usati "+NewCounter.alive()+" NewCounter");
    }
}
```

Studio di un Caso in OCaml e in Java

- Si assuma di operare su un sistema OCaml [Java] contenente:
  - Un tipo astratto per relazioni binarie IMMUTABLE
  - Un codice che opera su tali valori: operazione sim, operazione incl, ...

  Si specializzi la relazione per operare con relazioni simmetriche, ovvero si definisca un nuovo tipo astratto specifico per relazioni simmetriche ed equipaggiato delle operazioni del vecchio tipo astratto.
  Fatto questo, si dica se i nuovi valori possono essere utilizzati nel codice del vecchio sistema.

- Vediamo che in entrambi i linguaggi ciò è realizzato facendo RIUSO del vecchio tipo astratto:
  - OCaml: vedi Esercizio3 In AltriEsercizi8 e soluzione
    - stato concreto del nuovo tipo implementato con il vecchio tipo
  - Java: vedi Esercizio4 In AltriEsercizi8 e soluzione
    - stato concreto del nuovo tipo **eredita ed estende** vecchio tipo
- **MA**:
  - OCaml: il vecchio codice non è riusabile sui nuovi valori e deve essere riscritto completamente.
  - Java: Il vecchio codice è pienamente usabile sui nuovi valori che si integrano completamente con il vecchio sistema.