Interfacce

Programmazione
Corso di laurea in Informatica

Interfacce

- Un'***interfaccia*** Java è una collezione di metodi astratti (e di costanti)
- Un *metodo astratto* è un metodo non implementato
  - costituito dall'intestazione senza il corpo della definizione
- Un metodo astratto viene dichiarato mediante l'uso del modificatore `abstract`
  - ma poiché tutti i metodi di un interfaccia sono necessariamente astratti spesso viene omesso
- L'interfaccia viene usata per definire formalmente l'insieme dei metodi che una classe deve implementare

Interfacce

`interface` è una parola riservata

```
public interface Fattibile
{
   public void faiQuesto();
   public int faiQuello();
   public void faiQuesto(float value, char ch);
   public boolean faiQuest'altro (int num);
}
```

Per nessuno dei metodi in un'interfaccia viene definito il corpo

L'intestazione di ciascun metodo termina con il ;

Interfacce

- **Un'interfaccia non può essere instanziata**
- I metodi di un interfaccia hanno visibilità `public` per default
- **Le classi implementano un'interfaccia**
  1. Affermandolo nella intestazione della classe
  2. Fornendo l'implementazione per ciascun metodo astratto dell'interfaccia
- Una classe che implementa un'interfaccia, deve definire ***tutti*** i metodi dell'interfaccia altrimenti il compilatore ***segnala errore***

Implementare un interfaccia

```
public class Puo'Fare implements Fattibile
{
   public void faiQuesto()
   {
      // codice
   }

   public int faiQuello()
   {
      // codice
   }

   // etc.
}
```

`implements` è una parola riservata

A ogni metodo dell'interfaccia `Fattibile` viene data la definizione appropriata

Implementare un interfaccia

- La classe deve implementare **tutti** i metodi dichiarati nell'interfaccia
- Una classe che implementa un'interfaccia può **anche definire altri metodi**
- Una classe può **implementare diverse interfacce**
  - Le diverse interfacce sono separate da virgole nella clausola di implementazione
- `Parlante.java` rappresenta l'interfaccia
- `Filosofo.java` realizza una interfaccia `Parlante`
- `Cane.java` realizza una interfaccia `Parlante`

Polimorfismo con le interfacce

- Si ha ***polimorfismo*** quando un identificatore può riferirsi a oggetti di tipo differente in momenti diversi
  - Con le interfacce si creano riferimenti polimorfici
  - Il nome di un'interfaccia (es. `Fattibile`) può essere usato come tipo di una variabile di riferimento a un oggetto

```
Fattibile obj;
```

  - Il riferimento `obj` può puntare un oggetto di una qualunque classe che implementi l'interfaccia `Fattibile`
  - Il riferimento è ***polimorfo***, cioé può assumere diverse forme

Polimorfismo con le interfacce

- Il riferimento ***polimorfico*** viene risolto al tempo dell'esecuzione, cioé a ***run time***
- Si attua un ***legame dinamico***
  - Il metodo che viene invocato dipende dal tipo di oggetto a cui `obj` fa riferimento:

```
obj.faiQuesto();
```

  - La stessa linea di codice può eseguire diversi metodi in momenti diversi se l'oggetto cui punta `obj` cambia
- L'uso di riferimenti polimorfi può portare a un disegno elegante e robusto del software
- `Parlare.java`

Alcune interfacce standard

- L'interfaccia `Comparable` contiene un metodo astratto chiamato `compareTo`, usato per confrontare oggetti
  - La classe `String` implementa l'interfaccia `Comparable` che consente di confrontare stringhe in ordine alfabetico mediante il metodo `compareTo` specificato ad hoc
  - `int compareTo(Object obj)`
- L'interfaccia `Iterator` indica i metodi da implementare per gestire una collezione di oggetti
  - Caso per caso si deve decidere l'ordine con cui gli oggetti della collezione devo essere restituiti dai metodi
  - `boolean hasNext()`
  - `object next()`
  - `void remove()`

Eventi

- Un ***evento*** è un oggetto che rappresenta il verificarsi di una condizione a cui si vuole eventualmente reagire
- *Esempio*. Vogliamo che un programma esegua qualche azione quando:
  - Si muove il mouse
  - Viene premuto il bottone del mouse
  - Il mouse viene mosso
  - Si usa un bottone grafico
  - Si preme un tasto della tastiera
  - Passa un intervallo di tempo
- Spesso, ma non sempre, gli eventi corrispondono ad azioni dell'utente

Eventi

- La libreria standard di Java contiene diverse classi che rappresentano tipici eventi
  - *Esempio*. La classe `MouseEvent` o la classe `KeyEvent`
- Certi oggetti, come un applet o un bottone grafico, generano un evento
- Altri oggetti, chiamati *listeners*, reagiscono agli eventi
  - Esempio. La classe `MouseEventListener` o la classe `KeyListener`
- Gli oggetti *listener* contengono la definizione di quello che vogliamo che succeda in risposta a un evento

Eventi e listeners

L'oggetto può generare un evento

L'oggetto aspetta che l'evento sia generato per rispondere

Quando si verifica un evento, il generatore chiama il metodo appropriato dell'oggetto listener, passando l'oggetto che descrive l'evento

L'interfaccia di un ascoltatore

- Possiamo creare un oggetto `listener` scrivendo una classe che implementa una particolare *interfaccia* `Listener`
- La libreria standard di Java contiene diverse interfacce che corrispondono a particolari categorie di eventi
  - *Esempio.* L'interfaccia `MouseListener` contiene metodi correspondi a eventi del mouse rappresentati dalla classe `MouseEvent`
- Un oggetto listener, viene aggiunto alla componente che può generare l'evento per stabilire la *relazione* formale *tra il generatore dell'evento e l'oggetto in attesa*

La classe MouseEvent

- Genera eventi relativamente allo stato del mouse
  - *Mouse pressed* - il bottone del mouse è premuto
  - *Mouse released* - il bottone del mouse è rilasciato
  - *Mouse clicked* - il bottone del mouse è premuto e rilasciato
  - *Mouse entered*- il puntatore del mouse è mosso su una particolare componente
  - *Mouse exited* - il puntatore del mouse è allontanato da una particolare componente
- I metodi della classe
  - `Point getPoint ()`
  - `int getX ()`
  - `int getY ()`
  - `int getClickCount ()`

Interfaccia MouseListener

- L'interfaccia `MouseListener` reagisce a eventi della classe `MouseEvent`
  - `void mousePressed (MouseEvent event)`
  - `void mouseReleased (MouseEvent event)`
  - `void mouseClicked (MouseEvent event)`
  - `void mouseEntered (MouseEvent event)`
  - `void mouseExited (MouseEvent event)`
  - `Dots.java`
  - `DotsMouseListener.java`

Eventi del movimento del mouse

- Gli eventi del movimento del mouse rappresentati dalla classe `MouseEvent`:
  - *Mouse moved* - il mouse si è mosso
  - *Mouse dragged* - il mouse si è mosso mentre il bottone del mouse è tenuto premuto
- La corrispondente interfaccia listener `MouseMotionListener`
  - `void mouseMoved (MouseEvent event)`
  - `void mouseDragged (MouseEvent event)`
  - Una classe può essere sia generatore di eventi sia listener
  - Una classe può essere listener per diversi tipi di eventi
- `RubberLines.java`

Eventi dei tasti

- The following are called *key events*:
  - *key pressed* – tasto premuto
  - *key released* – tasto rilasciato
  - *key typed* – tasto premuto e rilasciato
- L'interfaccia `KeyListener` gestisce gli eventi legati ai tasti
- Le classi Listener sono spesso implementate come classi interne, annidate nelle componenti di cui sono in ascolto
- `Direction.java`

Animazioni

- Un'animazione cambia in continuazione immagini per creare l'illusione del movimento
- Possiamo creare animazioni in Java cambiando leggermente un'immagine
- La velocità di animazione è controllata dall'oggetto `Timer`
- L'oggetto `Timer` è definito nel pacchetto `javax.swing`

Animazioni

- Un oggetto `Timer` genera un `ActionEvent` ogni `n` millisecondi (dove `n` è definito dall'oggetto creatore)
- L'interfaccia `ActionListener` contiene il metodo `actionPerformed`
- Quando il tempo scade e viene generato un `ActionEvent`, l'animazione viene aggiornata
- `Rebound.java`