Progettare una classe

Avevo anticipato che per raggiungere rapidamente un punto compiuto abbiamo saltato un passaggio.

Il punto è questo: una buona tecnica di sviluppo delle classi consiste nel definire una classe per ogni obiettivo o sotto-obiettivo in modo che ciascuna abbia un proprio ambito di lavoro il più possibile circoscritto, verificabile e componibile con gli altri.

Nel nostro caso separiamo il metodo main che ha la funzione di innesco dalla classe che sviluppa la nostra logica.

Stiamo parlando di azioni estremamente semplici, ma imparare a separare i sotto obiettivi da subito aiuta a farlo anche quando il gioco si fa più duro.

Il passaggio da compiere nel risolvere un problema è “quale oggetto mi aiuta a risolvere il problema?”.

Quindi mi serve una classe con un metodo in grado di compiere lo scopo, e da qui istanziare un oggetto con cui invocare il metodo.

Iniziamo a definire la nuova classe seguendo le regole note:

```
class Saluta {
  <descrizione della classe>
}
```

La classe Saluta deve prevedere un metodo che ha il compito di produrre la visualizzazione di saluto.

Per procedere occorre conoscere le regole di definizione di un metodo di una classe.

Definizione di un metodo

In generale la regola di definizione è:

```
  <modificatori> <identificatoreDiMetodo>(<eventuali parametri>) {
    <contenuto del metodo>
  }
```

L'identificatore del metodo è il nome da dare al metodo per distinguerlo dagli altri. Sulla scrittura dell'identificatore di metodo valgono regole simili a quelle di identificatore di classe salvo che la convenzione tipicamente in uso è che iniziano sempre con lettera minuscola. Può essere una parola composta da più termini ottenuta scrivendo di seguito, tutto attaccato, le parole componenti, ciascuna con l'iniziale maiuscola.

Nel nostro esempio avremo:

```
  <modificatori> visualizza(<eventuali parametri>) {
    <contenuto del metodo>
  }
```

Dopo l'identificatore, tra le parentesi, i “parametri” sono dedicati al eventuale scambio di informazioni nel momento in cui si fa uso del metodo. Per ore non ne facciamo uso e quindi lasciamo vuoto.

I “modificatori” che troviamo al primo posto servono per descrivere alcune caratteristiche del metodo. Per ora ne utilizziamo 2:

- **public** che indica il il metodo può essere liberamente utilizzato. In futuro vedremo altre opzioni alternative a questa.
- **void** che indica che il metodo ha un comportamento di tipo 'azione'. Affronteremo con più precisione nel seguito.

Ecco dunque come descrivere il metodo:

```
  public void visualizza() {
    <contenuto del metodo>
  }
```

Il contenuto del metodo è esattamente lo stesso del metodo main della classe Ciao dell'esempio precedente, per cui la classe Saluta è così completata:

```
class Saluta {
  public void visualizza() {
    System.out.println("Ciao mondo!");
  }
}
```

Ritornano allo sviluppo della nuova classe Ciao vediamo come poter istanziare un oggetto.

Istanziare oggetti

Per istanziare un oggetto questa è la sintassi standard:

```
new <IdentificatoreDiClasse>(<eventuali parametri>);
```

dove

- **new** è la parola chiave che identifica l'operatore che è in grado di istanziare l'oggetto di una classe
- come in precedenza i parametri sono opzionali e al momento non li trattiamo

Nel nostro caso diventa:

```
new Saluta();
```

l'effetto è che viene creato un oggetto della classe Saluta.

Questa operazione da sé non è particolarmente utile, perché non indica cosa fare con questo oggetto appena creato.

La cosa più semplice che possiamo fare è di ottenere la memorizzazione di questo oggetto, cioè far in modo che l'oggetto rimanga a disposizione per il tempo che ci occorre durante l'esecuzione del nostro programma.

Variabili e attributi

La memorizzazione di oggetti e dati è affidata a specifici componenti che possono essere definiti all'interno di un metodo, in tal caso si chiamano variabili, o a livello di classe e in tal caso si chiamano attributi.

Una prima caratteristica di variabili e attributi è di essere tipizzata, cioé ogni variabile si specializza per poter ositare oggetti di una determinata classe su cui sono definiti.

La regola generale della dichiarazione di varibile (o attributo) è :

```
<IdentificatoreDiClasse> <identificatoreDiVariabile>;
```

Per gli identificatori di variabili valgono le stesse regole di scrittura che abbiamo visto per gli identificatore di metodo: iniziano sempre con lettera minuscola. L'identificatore può essere una parola composta da più termini ottenuta scrivendo di seguito, tutto attaccato, le parole componenti, ciascuna con l'iniziale maiuscola.

Con una dichiarazione di questo tipo

Nel nostro caso desideriamo una variabile che sia destinata ad un oggetto di classe saluta, per cui la dichiarazione sarà:

```
Saluta ciao;
```

L'effetto di questa dichiarazione è che 'ciao' è una variabile destinata ad ospitare un oggetto di classe Saluta, ma di fatto non contiene nulla.

E' solo “collegando” l'istanziazione alla variabile che effettivamente si potrà disporre un oggetto con cui svolgere azioni.

Il “collegamento” ci viene fornito dall'operatore di assegnamento.

Assegnamento

L'assegnamento è l'operazione che consente di 'riempire' di contenuto una variabile.

La sua forma sintattica è:

```
<identificatoreDiVariabile> = <espressione>
```

dove '**=**' non sta ad indicare che quello che è scritto a sinistra di '=' è identico a ciò che è scritto a destra, ma che ciò che si ottiene dalla 'espressione' scritta a destra deve essere memorizzato nella variabile indicata a sinistra.

Nel nostro esempio sarà:

```
ciao = new Saluta();
```

E' consentito riassumere in unica riga i due passaggi (dichiarazione di variabile e assegnazione di istanza di oggetto in questo modo:

```
Saluta ciao = new Saluta();
```

Esecuzione di un metodo

Ora che abbiamo una variabile con oggetto di tipo classe (o più sinteticamente diciamo un oggetto di tipo classe) per fare eseguire il metodo abbiamo a disposizione l'operatore '.' (punto).

```
<oggetto>.<metodo>();
```

l'effetto è che viene eseguito il metodo richiesto della classe a cui appartiene l'oggetto.

Nel nostro caso sarà:

```
ciao.visualizza();
```

Seconda versione del primo programma

Ecco dunque sviluppato per intero il primo programma riscritto in questa seconda versione.

```
class Saluta {
  public void visualizza() {
    System.out.println("Ciao mondo!");
  }
}

public class Ciao {
  public static void main(String[] args) {
    Saluta ciao = new Saluta();
    ciao.visualizza();
  }
}
```

Indentazione

Osservando il codice appena scritto si può notare che le riche non sono tutte allineate a sinistra ma sono presenti degli 'sfasamenti' verso destra, che prendono il nome di **indentazione**.

Questa tecnica di presentazione del codice sorgente si basa sul presentare incolonnate allo stesso livello elementi “pari” tra di loro, di spostare verso destra un elemento contenuto in un altro.

Infatti troviamo all'estrema sinistra le definizioni di classe, più a destra le definizioni di metodo contenute nelle rispettive classi e più a destra ancora le istruzioni interne ad ogni metodo.

Riguardo alle parentesi graffe che in generale “racchiudono” qualcosa una convenzione 'tipica” (anche se non unica) è che la parentesi graffa aperta è scritta all'estremo destro della riga “contenitore” e analogamente la parentesi graffa chiusa viene incolonnata al livello pari della riga “contenitore”.

Esercizi ed approfondimenti

Ripetizione di consolidamento

Scrivere un programma simile al precedente (con le due classi) in cui la frase visualizzata è 'Ciao lettore!”

Chiamata successiva di metodo

Modificare il programma CiaoMondo scrivendo in tre righe di seguito l'invocazione del metodo ciao.visualizza(). Che accade?

Realizzazione di classe

Scrivere una classe 'Autore' con un metodo che permette di visualizzare “Questo programma è stato scritto da Mario Rossi” (dove sostituirai con il tuo nome e cognome) e poi una classe con metodo main per verificare che funzioni correttamente

Uso di due classi

Modificare il primo programma Ciao mondo con le due classi aggiungendo come terza classe Autore, il metodo main dovrà disporre di due oggetti uno di classe Salute a l'altro di classe Autore, con questi visualizzare prima il saluto e poi la nota di autore.

Approccio a un problema

Produrre il programma 'Biglietto da visita' Che visualizza il testo:

Mario Rossi

Classe 3Xi

ITIS Belluzzi Bologna

cc