Il mondo ad oggetti

Nella nostra esperienza quotidiana facciamo uso continuativo di oggetti che utilizziamo per raggiungere i nostri scopi.

Ad esempio siamo abituati a prendere in mano il cellulare e ottenere la memorizzazione informazioni, comunicare ecc ecc
Quanto ci serve è sapere cosa è in grado gestire, operare e quali sono i comandi, le azioni da operare su di lui per raggiungere l'obiettivo.

D'altra parte sappiamo anche comporre oggetti diversi per ottenere funzioni che individualmente non sono in grado di svolgere, ad esempio con l'accoppiata lettore dvd-televisore si riesce a godere di un buon film su DVD, cosa non possibile se si usasse uno solo dei due oggetti.

Possiamo anche indurre un ragionamento a ritroso: quello che noi noi vediamo ed usiamo come un oggetto unico a sua volta "all'interno" è costituito di più oggetti, ognuno dei quali svolge compiutamente la sua parte.

Problemi ed oggetti

Nel nostro agire quotidiano ci è naturale affrontare e gestire le nostre esigenze individuando di volta in volta l'oggetto che ha le caratteristiche idonee a realizzare il nostro obiettivo.
Se devo scrivere su carta cerco una biro.
E anche desidero tanto il nuovo modello di lettore MP3 perché ha molti più GB di memoria per portarmi sempre dietro tutta la mia collezione di musica!

Oggetti, classi e istanze

Abbiamo dunque introdotto l'importanza che gli oggetti rivestono nel nostro quotidiano.
Completa il quadro l'osservare che ogni oggetto è la realizzazione concreta di una categoria di oggetti tutti dalle medesime caratteristiche.
Chiamiamo **classe** una categoria che definisce le caratteristiche di un certo insieme di oggetti.
Gli oggetti di un categoria possono essere assolutamente identici (ad esempio più biro identiche) oppure pur avendo le stesse caratteristiche differiscono per un particolare "valore" come ad esempio il colore dell'inchiostro: una biro rossa e una biro blu.
Il legame fra oggetto e classe viene chiamato **istanza**. Il termine non è molto diffuso nel linguaggio corrente, ma occorre farci abitudine...
Si dice che un oggetto è una istanza di una classe.

L'informatica, i programmi, gli oggetti

Nel mondo informatico abbiamo due grandi aree.
Una è costituita dalle parti materiali (hardware) i cui oggetti si vedono e toccano, come ad esempio il monitor, il mouse, il router, il cavo, eccetera.
L'altra non tangibile (software) i cui oggetti sono chiamati programmi, icone, messaggi.
Ci interessiamo di quest'ultima area che è oggetto di studio del corso.
Questi oggetti ci permettono di svolgere attività anche molto diverse: navigare per siti web, giocare, comunicare in voce e video, scrivere un testo, modificare un'immagine...

Ancora una volta sono oggetti che utilizziamo per raggiungere un nostro scopo.
Abbiamo a disposizione stimoli con i quali pilotiamo il comportamento desiderato, il click col mouse, i tasti o altri dispositivi come un joystick, un telecomando, eccetera.

Lo sviluppo del software

Come si producono questi oggetti software?
L'insieme di tecnologie e metodologie di lavoro che portano alla realizzazione di questi oggetti viene chiamata "sviluppo" del software, ed è sostanzialmente la struttura portante di questo corso.
L'enfasi data agli oggetti in questi paragrafi introduttivi serve a introdurre la metodologia che attualmente risulta essere più efficace per lo sviluppo. Secondo questa metodologia si giunge al prodotto finito realizzando uno o più oggetti che composti fra loro realizzano l'obiettivo.
O più precisamente si progettano una o più classi con cui istanziare oggetti che permettono di raggiungere l'obiettivo.

cc

Il mondo ad oggetti

Nella nostra esperienza quotidiana facciamo uso continuativo di oggetti che utilizziamo per raggiungere i nostri scopi.

Ad esempio siamo abituati a prendere in mano il cellulare e ottenere la memorizzazione informazioni, comunicare ecc ecc
Quanto ci serve è sapere cosa è in grado gestire, operare e quali sono i comandi, le azioni da operare su di lui per raggiungere l'obiettivo.

D'altra parte sappiamo anche comporre oggetti diversi per ottenere funzioni che individualmente non sono in grado di svolgere, ad esempio con l'accoppiata lettore dvd-televisore si riesce a godere di un buon film su DVD, cosa non possibile se si usasse uno solo dei due oggetti.

Possiamo anche indurre un ragionamento a ritroso: quello che noi noi vediamo ed usiamo come un oggetto unico a sua volta "all'interno" è costituito di più oggetti, ognuno dei quali svolge compiutamente la sua parte.

Problemi ed oggetti

Nel nostro agire quotidiano ci è naturale affrontare e gestire le nostre esigenze individuando di volta in volta l'oggetto che ha le caratteristiche idonee a realizzare il nostro obiettivo.
Se devo scrivere su carta cerco una biro.
E anche desidero tanto il nuovo modello di lettore MP3 perché ha molti più GB di memoria per portarmi sempre dietro tutta la mia collezione di musica!

Oggetti, classi e istanze

Abbiamo dunque introdotto l'importanza che gli oggetti rivestono nel nostro quotidiano.
Completa il quadro l'osservare che ogni oggetto è la realizzazione concreta di una categoria di oggetti tutti dalle medesime caratteristiche.
Chiamiamo **classe** una categoria che definisce le caratteristiche di un certo insieme di oggetti.
Gli oggetti di un categoria possono essere assolutamente identici (ad esempio più biro identiche) oppure pur avendo le stesse caratteristiche differiscono per un particolare "valore" come ad esempio il colore dell'inchiostro: una biro rossa e una biro blu.
Il legame fra oggetto e classe viene chiamato **istanza**. Il termine non è molto diffuso nel linguaggio corrente, ma occorre farci abitudine...
Si dice che un oggetto è una istanza di una classe.

L'informatica, i programmi, gli oggetti

Nel mondo informatico abbiamo due grandi aree.
Una è costituita dalle parti materiali (hardware) i cui oggetti si vedono e toccano, come ad esempio il monitor, il mouse, il router, il cavo, eccetera.
L'altra non tangibile (software) i cui oggetti sono chiamati programmi, icone, messaggi.
Ci interessiamo di quest'ultima area che è oggetto di studio del corso.
Questi oggetti ci permettono di svolgere attività anche molto diverse: navigare per siti web, giocare, comunicare in voce e video, scrivere un testo, modificare un'immagine...

Ancora una volta sono oggetti che utilizziamo per raggiungere un nostro scopo.
Abbiamo a disposizione stimoli con i quali pilotiamo il comportamento desiderato, il click col mouse, i tasti o altri dispositivi come un joystick, un telecomando, eccetera.

Lo sviluppo del software

Come si producono questi oggetti software?
L'insieme di tecnologie e metodologie di lavoro che portano alla realizzazione di questi oggetti viene chiamata "sviluppo" del software, ed è sostanzialmente la struttura portante di questo corso.
L'enfasi data agli oggetti in questi paragrafi introduttivi serve a introdurre la metodologia che attualmente risulta essere più efficace per lo sviluppo. Secondo questa metodologia si giunge al prodotto finito realizzando uno o più oggetti che composti fra loro realizzano l'obiettivo.
O più precisamente si progettano una o più classi con cui istanziare oggetti che permettono di raggiungere l'obiettivo.

cc

Linguaggi di programmazione

Esistono molti diversi linguaggi di programmazione. Sono stati pensati in base a diverse metodologie di lavoro e/ in riferimento a specifici campi di applicazione.

Nel nostro contesto ci occupiamo di linguaggi di uso generale (general purpose). Fra i linguaggi cito per esempio Java, C++, C#, Pascal, PHP, Fortran, Cobol, Basic, Ruby, Pyton... l'elenco può essere lunghissimo.

In questo corso faremo riferimento al linguaggo Java, che fa esplicito riferimento alla programmazione orientata ad oggetti.

Il codice sorgente

Il primo passo per realizzare un programma è di scrivere il "**sorgente**" (in inglese source). E' in sostanza il testo del programma, scritto seguendo le regole proprie del linguaggio utilizzando i normali caratteri alfabetici, numerici, di punteggiatura e quindi in qualche modo leggibili anche da noi.

Il programma così preparato non è ancora nella forma in cui il computer è in grado di eseguire, occorrono alcuni passaggi. Il tipo di passaggi può essere diverso tra un linguaggio e l'altro secondo tre grandi categorie:

- linguaggi compilati
- linguaggi interpretati
- linguaggi a codice intermedio

Linguaggi compilati

Caratteristica propria dei linguaggi di questo tipo è che il sorgente viene fatto elaborare da un particolare programma chiamato **compilatore** che svolge due funzioni: verifica la correttezza sintattica del programma scritto e se non incontra errori ne effettua la traduzione dal linguaggio del sorgente al linguaggio proprio del calcolatore (il linguaggio macchina). Ciò che si tiene viene chiamato **programma in versione oggetto**, ma in questo contesto il termine oggetto non assume lo stesso significato a cui abbiamo fatto riferimento parlando di oggetti e classi. Se il sorgente contiene uno o più errori sintattici non viene prodotto il file con la versione oggetto, ma viene fornito l'elenco degli errori riscontrati.

Il programma in versione oggetto è scritto in linguaggio macchina, ma non è ancora immediatamente eseguibile perché mancano alcune ulteriori informazioni e parti comuni che dovranno essere aggiunte. Per questo motivo la fase successiva è di far elaborare il programma in versione oggetto da un altro programma chiamato **linker** (termine inglese che significa 'collegatore') che appunto ha la funzione di collegare i riferimenti mancanti nel programma oggetto al fine di ottenere la versione definitiva chiamata **programma eseguibile**. A volte lo si trova indicata anche come versione binaria (binary).

In questa versione il programma può essere eseguito dal computer producendo gli effetti previsti dalle istruzioni che lo compongono.

Il programma eseguibile ottenuto è applicabile solamente ad un unico tipo di computer e tipo di sistema operativo. Quindi un programma compilato per un pc con sistema operativo windows potrà essere utilizzato dai compiter con queste caratteristiche e quindi non da

uno che abbia ad esempio il sistema operativo linux. Per questo ultimo occorre ri-effettuare i passaggi partendo dal sorgente e utilizzando compilatore e linker specifici per linux.

Appartengono a questa categoria ad esempio i linguaggi C++, Fortran, Cobol e molti altri

Linguaggi interpretati

Per linguaggi di questo tipo non esiste una fase di traduzione del sorgente. Si fa uso di un programma chiamato **interprete** a cui viene affidata l'elaborazione del sorgente. L'interprete esamina la prima istruzione, ne verifica la correttezza sintattica e in caso positivo esegue direttamente l'istruzione stessa, poi passa alla successiva e così via.

Se una istruzione contiene un errore l'interprete interrompe l'esecuzione e segnala a che punto si ferma e il motivo. Quindi un programma contenente uno o più errori potrà essere parzialmente eseguito.

Su computer e/o sistemi opertivi diversi lo stesso sorgente poà essere interpretato a patto di disporre del programma interprete adeguato all'ambiente specifico.

Fanno parte di questa categoria ad esempio i linguaggi Basic, PHP, Perl, Pyton e molti altri

Linguaggi a codice intermedio

Per questa categoria di linguaggi il programma sorgente deve subire una prima fase di verifica sintattica e traduzione producendo una versione detta **codice intermedio**. La differenza rispetto alla compilazione è che ciò che si è ottenuto non è ancora espresso secondo il linguaggio macchina, per essendo una forma non più comprensibile a noi. Il codice intermedio ottenuto deve essere affidato all'elaborazione di una '**macchina virtuale**' che è in grado di interpretare ed eseguire il suo contenuto. La macchina virtuale è in sostanza un programma specifico per tipo di computer e sistema operativo.

In sostanza ci troviamo in una situazione 'mista' delle due precedenti. La prima fase di traduzione è simile a quella del compilatore, ma produce un codice intermedio che è indipendente dal particolare tipo di computer/sistema operativo. Quindi un programma in versione codice intermedio è in sostanza eseguibile su piattaforme diverse. D'altra parte la macchina virtuale è in qualche modo simile ad un interprete, con la differenza che ha maggiore efficienza perché non deve occuparsi di verifica sintattica (già fatta nella traduzione verso il codice intermedio) e la traduzione finale stessa è più semplice visto che si parte da un codice molto più vicino alla macchina del linguaggio di programmazione.

Fanno parte di questa categoria ad esempio i linguaggi Java, C#, Pascal e altri

cc

Il mio primo programma java

Java

Java è il linguaggio di programmazione che utilizzeremo durante questo corso. Come già detto è un linguaggio di tipo a codice intermedio ed è modellato per la programmazione orientata agli oggetti (OOP Object Oriented Programming).

Esistono più di un modo operativo per scrivere programmi sorgenti Java, tradurli ed eseguirli. Inizialmente utilizziamo un ambiente che predispone in modo unitario tutto quanto server per le diverse fasi e viene pertanto chiamato ambiente di sviluppo (in inglese IDE: Integrated Development Environment).

Questo ambiente si chiama NetBeans ed è realizzato dalla società informatica SUN che è anche la società che ha inventato Java. Il programma è scaricabile gratutamente all'indirizzo web http://www.netbeans.org/ .

Scriviamo il primo programma

Ogni scritto dal più formale al più divulgativo che presenti come realizzare il primo programma in una qualunque ambiente propone di realizzare un saluto "Ciao mondo!" e così sarà anche nel nostro caso...

Esplicito in un modo un po' più formale l'obiettivo: intendiamo progettare e realizzare un programma che sia in grado di visualizzare la frase "Ciao mondo!". I passaggi sono :

- scrivere secondo le regole di java il programma, correggendo eventuali errori fino ad ottenere un programma che può essere eseguito
- richiedere l'esecuzione del programma ottenendo l'effetto desiderato

Per arrivarci dobbiamo conoscere alcune caratteristiche di Java.

La dichiarazione di classe

Un primo punto è la regola su come descrivere una classe:

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

in cui:

- class è un **parola chiave** (**keyword**) che va sempre scritta esattamente così, serve per indicare che ciò che segue è la dichiarazione di una classe, cioè la descrizione della classe che si sta realizzando
- <nomeclasse> rappresenta genericamente una parola che scegliamo noi arbitrariamente per indicare (identificare) la classe che stiamo dichiarando. Viene chiamato identificatore della classe

Quindi potrà essere:

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

Va precisato che il linguaggio java prevede la netta distinzione tra caratteri minuscoli e maiuscoli.

Quindi se si scrive Class, o CLASS o altre combinazioni minuscolo/maiuscolo la parola non viene riconosciuta come parola chiave.

Per quanto riguarda invece l'identificatore di classe seguiremo la convenzione che viene tipicamente adottata: l'identificatore di classe ha iniziale maiuscola e resto minuscolo se costituito da una unica parola. Può essere una parola composta da più termini ottenuta scrivendo di seguito, tutto attaccato, le parole componenti, ciascuna con l'iniziale maiuscola. Esempio potremmo chiamare la nostra classe CiaoMondo.

Dopo la parola chiave 'class' e l'identificatore 'Ciao' segue una parentesi graffa aperta '{' a cui corrisponde più avanti la graffa chiusa '}'. Anche questi sono elementi obbligatori che fanno parte della corretta sintassi di scrittura di una classe.

Tra le due graffe la *<descrizione della classe>* rappresenta genericamente quanto effettivamente descrive ciò che la classe contiene (attributi) e/o deve poter fare (metodi).

Metodi di una classe

I metodi di una classe descrivono ciò che desideriamo che la classe sia in grado di fare, rappresentano dunque le capacità di azione della classe.

In generale siamo liberi di progettare quanti metodi riteniamo siano utili ai nostri fini ma nel nostro caso stiamo progettando una classe che ha un compito specifico: rispondere al comando di avvio programma.

Quindi posticipo la descrizione di come comportarsi in modo generico sui metodi e partiamo dal particolare metodo di questa classe.

Il metodo main

Riformulo la proprietà che ci serve.

Ci serve una classe che è in grado di rispondere allo stimolo di avvio del programma (ad esempio il doppio click di un'icona, o un comando scritto in una finestra a comandi).

Per fare in modo che una classe abbia questa caratteristica deve avere un metodo che sia così scritto:

```
public static void main (String[] args) {
  <contenuto del metodo>
}
```

La parte *<contenuto del metodo>* rappresenta genericamente ciò che lo specifico nostro programma dovrà realizzare, le altre parti invece sono 'fisse' cioè vanno scritte esattamente così. Ci chiariremo strada facendo l'esatto significato di ognuno dei termini utilizzati. Per ora vorrei soffermarmi sul perché di questo vincolo su questo metodo.

Quando facciamo eseguire un programma (ad esempio con doppio click su un'icona) c'è una sorta di "passaggio di consegne" in cui il computer prende in carico il programma e incomincia ad eseguirlo. Ma da che punto? Ci vuole un punto "inizia da qui" che sia immediatamente riconoscibile. Bene, nei programmi java "l'inizia da qui" è costituito dal metodo con le regole descritte in precedenza. Per essere completi va anche aggiunto che tale metodo deve essere all'interno di una classe che ha come identificatore lo stesso

nome del programma.

Riassumendo quanto visto fin qui, la classe è così definita:

```
class Ciao {
  public static void main (String[] args) {
    <contenuto del metodo>
  }
}
```

dobbiamo ancora definire cosa mettere nel contenuto del metodo ma intanto salviamo il file con il nome Ciao.java in modo che il programma che otterremo abbia lo stesso nome della classe.

Come descrivere il "contenuto del metodo". In sostanza vogliamo che appaia la scritta 'Ciao mondo!', quindi ci serve un 'qualcosa che 'sa' come scrivere una frase.

Per questo abbiamo a disposizione in java alcune classi ed oggetti "predefiniti" che sono cioè pronti all'uso. Si tratta di ricordarsi che esistono, cosa fanno e come vanno utilizzati... :-)

Nel nostro caso ci viene in aiuto "Sistem.out.println" che è un metodo che abbiamo a disposizione per poter far visualizzare una frase in una finestra di comandi. La frase va specificata come parametro passato al metodo in questo modo: Sistem.out.println("Ciao mondo!"), e dunque ora abbiamo un aspetto completo per la nostra classe:

```
class Ciao {
  public static void main (String[] args) {
    System.out.println("Ciao mondo!");
  }
}
```

La riga che contiene la 'chiamata' del metodo println termina con un ";". Questa regola è generale, ogni istruzione java termina con ";".

Preciso che siccome i passaggi sono già stati tanti ne ho qui saltato uno piuttosto importante che vedremo nel prossimo paragrafo e di cui faremo uso comunque da lì in poi.

cc

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

Classi e oggetti con più metodi

Eseguire un metodo più volte

Possiamo richiedere ad un oggetto di eseguire più volte un metodo, ottenendo quindi più volte l'effetto prodottoo da quel metodo.

Ecco come vedere tre volte il saluto:

```
package saluto;

public class Saluto {
  public void saluta(){
      System.out.println("Ciao, mondo!");
  }
}
```

```
package saluto;

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //preparo una variabile per oggetto di classe Saluto
        Saluto ciao;
        //istanzio un oggetto di classe Saluto
        ciao = new Saluto();
        //l'oggetto effettua il saluto per tre volte
        ciao.saluta();
        ciao.saluta();
        ciao.saluta();
    }

}
```

Si otterrrà questo output:

```
Ciao, mondo!
Ciao, mondo!
Ciao, mondo!
```

Classe con più metodi

Possiamo arricchire la classe con più metodi in modo da poter ottenere comportamenti diversi, ad esempio:

```
package saluto;
```

```
public class Saluto {
  public void saluta(){
      System.out.println("Ciao, mondo!");
  }
  public void salutaClasse(){
      System.out.println("Ciao, 3Ai!");
  }
  public void salutaProf(){
      System.out.println("Ciao, professore!");
  }
  public void salutaMe(){
      System.out.println("Ciao, Gianni!");
  }}
```

La classe Saluto offre ora anche altri tre metodi che realizzano messaggi di saluto di tipo diverso.

Il metodo main con un oggetto di questa classe può chiederne l'esecuzione.

```
package saluto;

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //preparo una variabile per oggetto di classe Saluto
        Saluto ciao;
        //istanzio un oggetto di classe Saluto
        ciao = new Saluto();
        //l'oggetto effettua vari saluti
        ciao.saluta();
        ciao.saluta();
        ciao.salutaClasse();
        ciao.salutaProf();
        ciao.salutaMe();
    }

}
```

Si otterrrà questo output:

```
Ciao, mondo!
Ciao, mondo!
Ciao, 3Ai!
Ciao, professore!
Ciao, Gianni!
```

Si noti che si è liberi di eseguire i metodi nell'ordine che si preferisce, indipendentemente

dall'ordine con cui sono definiti nella classe, ad esempio:

```
package saluto;

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //preparo una variabile per oggetto di classe Saluto
        Saluto ciao;
        //istanzio un oggetto di classe Saluto
        ciao = new Saluto();
        //l'oggetto effettua vari saluti
        ciao.salutaClasse();
        ciao.saluta();
        ciao.salutaMe();
        ciao.saluta();
        ciao.salutaProf();
    }

}
```

produce questo output:

```
Ciao, 3Ai!
Ciao, mondo!
Ciao, Gianni!
Ciao, mondo!
Ciao, professore!
```

cc