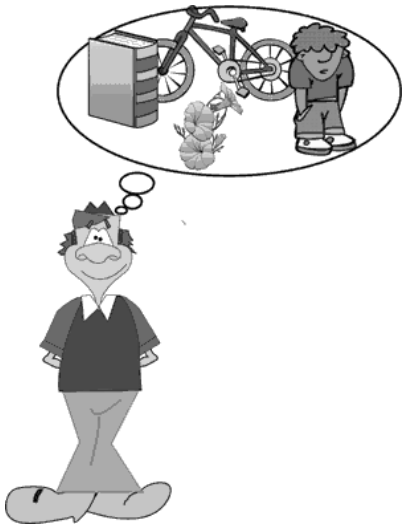
Oggetti e classi

Cos’è un oggetto

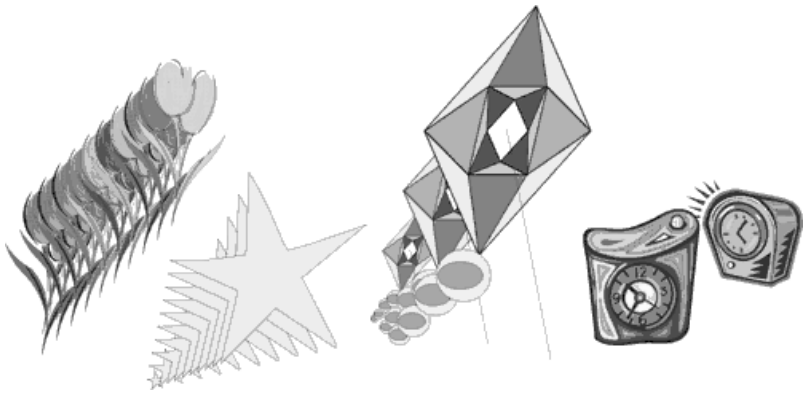
Basta guardarsi intorno per scoprire che il mondo reale è costituito da *oggetti*: libri, biciclette, giocattoli, ma anche ragazzi, bambini, fiori, gatti, cani, fiumi, montagne, laghi, pianeti, stelle, aquiloni, figure geometriche, orologi...



Tutti gli oggetti del mondo reale condividono due aspetti: uno ***stato*** ed un ***comportamento***. Lo stato è rappresentato dalle loro ***caratteristiche*** o ***proprietà***.(un ragazzo ha un nome, un cognome, un indirizzo, è contento, triste, stanco, simpatico...), il comportamento dalle ***azioni*** od ***operazioni*** che lo riguardano (lo stesso ragazzo può correre, cantare, studiare, dormire...). L’identificazione dello stato e del comportamento degli oggetti rappresenta il punto di partenza per la creazione di programmi in Java. S’inizia con l’osservazione attenta degli oggetti del mondo reale, delle loro caratteristiche e del loro comportamento. Certamente ognuno di essi può raggiungere gradi diversi di complessità. Gli oggetti in Java sono concettualmente simili agli oggetti del mondo reale: il loro stato è rappresentato da *caratteristiche* (dette anche *proprietà, attributi, variabili, campi* o *dati*) e il loro comportamento è rappresentato da *metodi* (o *funzioni*).

Componenti di una classe

Nel mondo reale ritroviamo generalmente diversi oggetti dello stesso *tipo*. Possiamo vedere cento tulipani, mille stelle, tre aquiloni, due orologi.



In Java ogni tulipano è **un'*istanza* della *classe* di oggetti costituiti da tulipani, un aquilone è un'*istanza* della *classe* di oggetti rappresentati da aquiloni, e così via**. **Una *classe* è il *modello* su cui si basa la creazione dei singoli oggetti.**

Un programma scritto in **Java** è costituito da un insieme di classi (ed eventualmente di ***interfacce***, come vedremo in seguito).
Una ***classe*** inizia sempre con la parola chiave *class* seguita da un nome, preferibilmente significativo, e con il primo carattere maiuscolo. Il contenuto di una classe è racchiuso fra due parentesi graffe. Ad esempio:

```
class Orologio {
  // contenuto della classe
}
```

Normalmente un programma contiene più classi in relazione l'una con l'altra e ripartite in più file, tutti con estensione ***.java***.

Ad ogni ***dato* (o *caratteristica*)** deve essere associato un ***tipo***, che può essere o meno numerico. Nel caso della classe Orologio i dati potrebbero essere rappresentati da ora, minuto, secondo (caratteristiche significative dell'orologio). Ora, minuto e secondo sono quindi i ***nomi*** dei dati e tutti e tre possono essere associati al tipo intero (in Java *int*):

```
class Orologio {
      private int ora;
      private int minuto,
      private int secondo;

   // segue restante contenuto della classe
   }
```

Le variabili di tipo intero possono contenere solo numeri interi. I dati devono preferibilmente essere *private* e comunque **MAI** *public*. Per rispettare il principio dell'***incapsulazione*** secondo cui un dato non deve essere direttamente accessibile dall'esterno della classe a cui appartiene (quindi da un'altra classe). I dati dall'esterno devono essere accessibili soltanto mediante i ***cosiddetti metodi di accesso get*** e ***set***, di cui parleremo fra breve.

I ***metodi*** rappresentano le operazioni sui dati e si riconoscono dalla presenza di parentesi tonde dopo il loro nome. Vediamo innanzitutto due tipi particolari di metodi:

1. i ***costruttori***
2. *i* ***metodi di accesso get e set***

1.I ***costruttori*** sono metodi che hanno lo stesso nome della classe e vengono automaticamente attivati ogni volta che si crea (istanzia) un oggetto (con l'istruzione *new*). Loro compito è quello di creare fisicamente un oggetto, allocando memoria, e di inizializzarlo. Ad esempio quando si crea un orologio in memoria come quello descritto in *class Orologio*, vengono allocati 4 byte per ogni dato di tipo int, cioè 12 byte in totale (sempre facendo riferimento ai processori a 32 bit).

2. ***I metodi di accesso get e set,*** generalmente specificati come *public* consentono l'accesso ai dati dall'esterno della classe: i metodi get solo ***in lettura*** (non consentono modifiche ai dati), i metodi set consentono di ***modificare i dati***.
Esempio di metodi get sono:

```
public int getOra()
        {
        return ora;
        }

public int getMinuto()
        {
        return minuto;
        }

public int getSecondo()
        {
        return secondo;
        }
```

Questi metodi rendono accessibili, "inviandoli" verso l'esterno mediante *return*, rispettivamente ora, minuto e secondo.
Ogni programma contiene un metodo particolare di nome *main():*

```
public static void main(String[] args)
```

Da questo metodo parte sempre l'esecuzione del programma stesso. Spesso al suo interno vengono istanziati degli oggetti, vale a dire creati degli esempi (istanze) di classi descritte altrove (o nello stesso file o in altri file).
Il *main()* normalmente è all'interno di una classe *public*.

```
public class ProvaOrologio {
public static void main(String[] args) {
```

```
    // Istanziazione di oggetti
  }
}
```

Di classi *public* all'interno di un singolo file ne può esistere una sola e deve necessariamente avere lo stesso nome del file che la contiene (maiuscole e minuscole comprese).

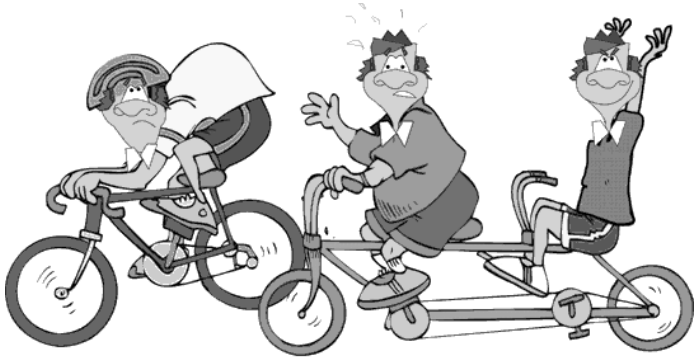
Raggruppare il codice all'interno di singole classi comporta una serie di vantaggi, fra cui:

1. la ***modularità***, grazie alla quale il codice di una classe può essere scritto e mantenuto indipendentemente da quello delle altre classi;
2. l'***incapsulazione*** dei dati, che mantiene nascosti all'esterno (alle altre classi) i dettagli interni di ogni singola classe; le classi comunicano fra di loro esclusivamente attraverso i metodi;
3. la ***riusabilità*** del codice: qualsiasi oggetto esistente, indipendentemente da chi ne sia l'autore, può essere riusato all'interno del proprio programma.

Relazioni fra classi

Ereditarietà

Spesso tipi diversi di oggetti hanno delle componenti comuni. Le mountain bike, i tandem e le biciclette da città condividono molte caratteristiche, ma ogni tipo di bicicletta ne aggiunge altre di particolari (un tandem, ad esempio, ha due manubri e due selle).



In Java una classe può ***ereditare*** da un'altra classe caratteristiche e metodi. Una ***classe di base*** o ***superclasse*** può avere un numero illimitato di ***classi derivate*** o ***sottoclassi***, ma una sottoclasse può ereditare da un'unica superclasse.
L'*ereditarietà* si usa **SE E SOLO SE** la classe derivata è del tipo della classe di base! **Esprime la relazione linguistica *è-un (is-a)*. Una classe di base deve contenere tutti gli elementi (dati e metodi) *comuni* alle sue classi derivate**. Una classe derivata eredita sia dati che metodi dalla sua classe di base. Non può però accedere direttamente ai dati **private** ereditati dalla classe di base. Nella classe derivata si usano a questo scopo metodi **get** e **set** *public* ereditati dalla classe di base. Dati e metodi derivati devono essere significativi: in caso contrario la derivazione non è corretta. Data la derivazione

```
class OrologioMeccanico extends Orologio {
        //.....
```

```
}
```

tutto ciò che appartiene ad orologio appartiene anche a OrologioMeccanico.
Quando ci si trova in una classe derivata è molto importante ricordare che i metodi ereditati(a meno che non siano *private*) vanno usati come se fossero stati definiti nella classe stessa: anche se "non si vedono" sono dentro la classe.
Se invece un metodo ereditato è stato ***ridefinito***, vale a dire riscritto, dentro la classe derivata, la versione originale del metodo viene "oscurata" e di fatto si usa la ridefinizione.

Composizione

Gli oggetti nel mondo reale possono essere in relazione reciproca, senza però per questo costituire delle gerarchie. Un pianista e il suo pianoforte sono in relazione, ma non per questo derivano l'uno dall'altro.

Per mettere in relazione due classi, ogni volta che non è consentita (per motivi logici) l'ereditarietà si usa la ***composizione,*** che **esprime la relazione linguistica *ha-un (has-a)***.
Per esprimere la relazione fra un pianista ed un pianoforte, posso scrivere:

```
class Pianista extends Strumentista {
   Pianoforte piano;
      //.....
   }
```

Ciò presuppone che esista una classe Pianoforte definita (e poi istanziata con *new*) all'interno del progetto. A questo punto, mediante il riferimento *piano*, sarà consentito richiamare, dall'interno di Pianista, tutti i metodi *public* (e come vedremo in seguito *friendly*) di Pianoforte.