Gli oggetti e le classi

Programmazione
Corso di laurea in Comunicazione digitale

Introduzione agli oggetti

- Interagiamo con oggetti di uso quotidiano, conoscendone le *funzioni*, ma non il *funzionamento interno*
  - Gli oggetti sono ***scatole nere*** dotate di interfaccia che limita l'accesso ai meccanismi interni
  - Gli oggetti hanno uno ***stato***
    - L'insieme delle proprietà che lo caratterizzano in un dato istante
  - e un ***comportamento***
    - L'insieme delle azioni che un oggetto può compiere
- Un oggetto sw è un'***astrazione*** o un ***modello*** della realtà che limita il numero dei *dettagli* rappresentati *all'essenziale* per il contesto considerato

Astrazione

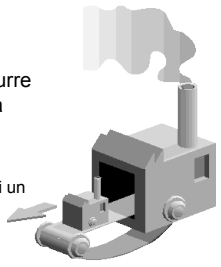
- L'astrazione nasconde o ignora dettagli inessenziali
- Effettuiamo astrazioni continuamente
  - Possiamo trattare solo poche informazioni contemporaneamente
  - Ma se raggruppiamo le informazioni (come gli oggetti) allora possiamo trattare informazioni più complicate
- Un *oggetto sw* è un'astrazione
  - Non ci preoccupiamo dei suoi dettagli interni per usarlo
  - Non conosciamo come funziona il metodo `println` quando l'invochiamo
- Quindi, possiamo anche scrivere software complesso organizzandolo attentamente in classi e oggetti

Gli oggetti software

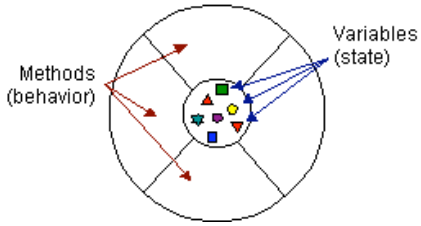
- Lo ***stato*** di un oggetto sw è descritto e rappresentato dai suoi ***campi***
  - I campi sono sostanzialmente variabili, ma si chiamano diversamente per distinguerli dalle variabili usate nei metodi
  - I campi, come le variabili, sono individuati da un ***identificatore*** e custodiscono un ***dato***
- il ***comportamento*** è definito dai ***metodi***
- Un oggetto sw è costituito dall'insieme dei suoi ***membri***: campi e metodi

Oggetti e classi

- Una ***classe*** genera ***oggetti***
  - Dette anche ***istanze*** della classe
- La ***classe*** è uno schema per produrre una categoria di oggetti di struttura identica
  - La classe costituisce il *prototipo*
  - La classe descrive le caratteristiche di un oggetto
  - Una classe è una fabbrica di istanze: possiede lo schema e la tecnica di produzione

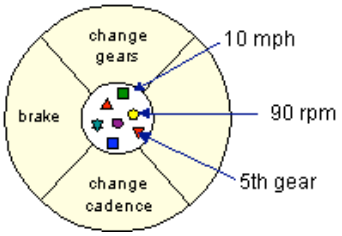


Gli oggetti come astrazione



- Un oggetto che modella una bicicletta
  - La marca, il colore, la velocità (20 Km/h), il giro dei pedali (15 g/m) e la marcia (5°) sono ***campi d'istanza***
  - proprietà rappresentate in ciascuna bicicletta modellata

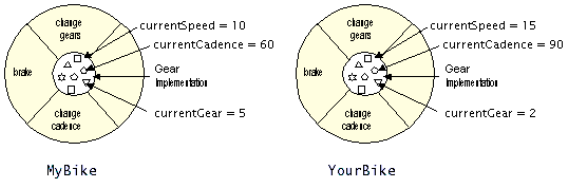
Gli oggetti come astrazione – 2



- Inoltre nel modello rappresentiamo funzioni come ***frenare*** o ***cambiare marcia***, che modificano i campi d'istanza
- Si chiamano ***metodi d'istanza*** perché hanno accesso ai campi d'istanza e li modificano

Le istanze

- Definita una classe, si possono creare un numero arbitrario di oggetti appartenenti alla classe

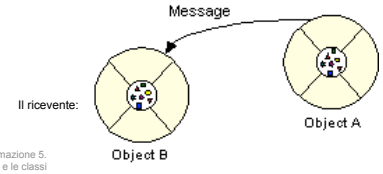


Incapsulamento dei dati

- Nascondere le informazioni fornendo *un'interfaccia*
  - I *campi* di un oggetto, che ne rappresentano lo stato, sono *nascosti* all'interno dell'oggetto, *accessibili* solo ai metodi
  - Idealmente i metodi proteggono i campi: solo i metodi possono accedere e modificarne il contenuto
- Per controllare l'accesso usiamo 4 modificatori diversi per campi e metodi:
  - `public`: accessibili a chiunque
  - `private`: accessibili solo alla classe
  - `protected`: accessibili a classe, sottoclassi e pacchetto
  - senza modificatori: accessibili solo alle classi del pacchetto
- Consente modularità e flessibilità e protegge i dati

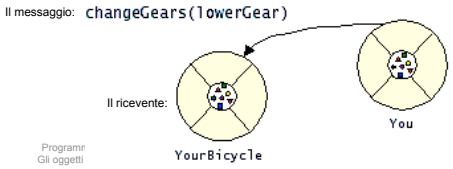
I messaggi

- Gli oggetti interagiscono tra loro per ottenere funzioni più complesse
  - La bicicletta appesa in garage è un oggetto e basta, ci vuole un ciclista che interagisca con lei perché diventi interessante
- Gli oggetti sw per interagire si mandano messaggi
  - la richiesta all'oggetto di eseguire un certo metodo



I messaggi – 2

- Spesso i metodi necessitano di informazioni per poter essere eseguiti: ***i parametri***
- Tre componenti:
  - L'oggetto a cui il messaggio è rivolto: il ricevente
  - Il metodo da eseguire per ottenere un certo effetto
  - I parametri, se necessari al metodo e separati da virgola



I messaggi – 3

- Un ***oggetto*** può essere visto come un insieme di servizi che possiamo chiedere di eseguire
- I servizi sono definiti dai ***metodi***
- Il comportamento degli oggetti è definito dai suoi metodi e il meccanismo di invio dei messaggi consente l'interazione tra gli oggetti
- Gli oggetti che si scambiano i ***messaggi*** possono anche essere '*distanti*' tra loro
  - Su macchine diverse
  - Non appartenenti allo stesso modello

Interfaccia

- L'***interfaccia*** è l'insieme dei messaggi che un oggetto è in grado di interpretare
- Un oggetto soddisfa la richiesta di un messaggio, se questo è contenuto nella sua interfaccia
- Eseguendo il ***metodo*** si soddisfa la richiesta da parte di un agente e si da la risposta ad un messaggio

Inviare messaggi

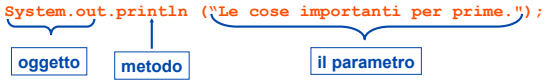
- Il ciclista `ciclista_A` che vuole cambiare marcia invia il messaggio all'oggetto `bicicletta_rosa`

Invocazione di un metodo

- Molte istruzioni sono invocazioni di metodi su oggetti
- La sintassi della chiamata del metodo:

```
oggetto.nomeMetodo (parametri)
```

- Chiediamo il *servizio di stampa*, invocando il metodo `println` dell'oggetto `System.out`



Metodi e oggetti

- I metodi possono essere invocati su oggetti della classe che hanno quel metodo nella loro interfaccia
  - Il metodo `println` si può applicare a oggetti della classe `PrintStream`

    ```
    System.out.println()
    ```
  - Il metodo `length` si può applicare a oggetti della classe `String`

    ```
    "salute a tutti".length()
    ```
  - Quindi causa errore chiamare:

    ```
    "salute a tutti".println()
    ```


I metodi println e print

- L'oggetto `System.out` della classe `PrintStream` fornisce altri servizi
  - Il metodo `print`
  - simile al metodo `println` - non fa avanzare il cursore alla riga successiva
  - Quindi quello che è stampato dopo l'istruzione `print` appare sulla stessa riga
- Esempio Conto_alla_rovescia.java

I membri delle classi

- Le classi contengono 2 tipi di ***membri***, definiti per l'intera classe o per le singole istanze
  - I ***campi***, che rappresentano lo stato della classe o degli oggetti
  - I ***metodi***, che rappresentano il comportamento: codice eseguibile sottoforma di istruzioni
- Il tipo di un oggetto è definito dalla classe di appartenenza

Esempio

```
Class Point {
      public int x, y;
}
```

- La classe `Point` della libreria `awt` ha due campi, `x` e `y`, che rappresentano le coordinate del punto
- I campi sono dichiarati `public`, cioè chiunque acceda alla classe `Point` può modificarli

I campi statici di classe

- Campi associati alle istanze della classe
  - Contengono informazioni specifiche per ogni oggetto
- Campi ***statici*** associati alla classe
  - Rappresentano dati condivisibili da tutti gli oggetti della classe, specifici della classe e non degli oggetti

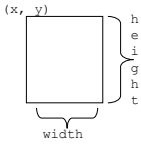
  Detti anche ***campi di classe***
  - Esempio: `origine` potrebbe essere dichiarata come campo di classe in una classe `SistemaCartesiano`, e riferimento a un oggetto della classe `Point`, cioè di tipo `Point`

```
public static Point origine = new Point();
```


La classe predefinita Rectangle

- La classe `Rectangle` della libreria `awt` di Java e i campi d'istanza

Rectangle	
5	x
10	y
15	width
20	height



- I campi `x` e `y` rappresentano la posizione dell'angolo alto sinistro e i campi `width` e `height` rispettivamente l'ampiezza e l'attezza
- Si noti che l'astrazione operata consiste nel considerare un rettangolo come una collezione di 4 valori numerici

Creare oggetti

- Gli oggetti vengono creati mediante uno speciale *metodo di istanziazione*, detto ***costruttore***
- L'operatore `new` seguito dal nome della classe istanzia un nuovo oggetto con valori di default dei campi
  - `new Rectangle()`
  - Costruisce un rettangolo con i 4 campi al valore 0
- o con i valori passati come parametri
  - `new Rectangle(5, 10, 15, 20)`
  - Costruisce l'oggetto raffigurato prima
- Esempio `TestRettangolo.java`

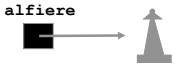
L'operatore `new`

- Si usa per istanziare nuovi oggetti di una classe
- È un operatore unario e viene prefisso al proprio argomento: un costruttore della classe
- `new costruttore_classe()` costituisce una espressione di Java (e non un'istruzione)
- Riporta un valore: un ***riferimento*** all'oggetto della classe specificata dal costruttore
- Il riferimento viene generalmente salvato in una variabile mediante assegnamento

Riferimenti

- Il *riferimento* a un oggetto contiene l'indirizzo di memoria dell'oggetto
- Descriviamo l'indirizzo come un *puntatore* all'oggetto

```
PezzoScacchi alfiere = new PezzoScacchi();
```



Oggetti e i loro riferimenti

- Gli oggetti creati sono collocati in un'area di memoria detta ***heap*** e sono accessibili mediante ***riferimenti***

```
Point altoSinistra = new Point();
Point bassoDestra = new Point();
```

- Le variabili che rappresentano oggetti contengono il loro riferimento tramite cui possiamo accedere ai campi degli oggetti

```
bassoDestra.x = 112;
bassoDestra.y = 40;
```


In memoria

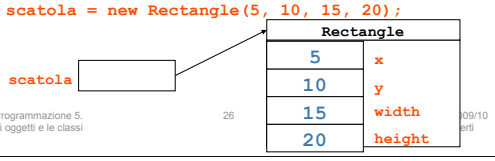
- La dichiarazione di una variabile oggetto riserva una posizione di memoria e associa l'indirizzo all'identificatore

```
Rectangle scatola;
```

scatola

non causa la sua inizializzazione, che va effettuata esplicitamente mediante l'operatore **new**:

```
scatola = new Rectangle(5, 10, 15, 20);
```



Variabili

- Una ***variabile*** rappresenta un ***dato*** identificato da un ***identificatore***, valore simbolico dell'indirizzo della cella di memoria in cui il dato è archiviato
- Una variabile deve essere ***dichiarata***, specificandone l'identificatore e il tipo di informazione che deve contenere

tipo del dato **identificatore**

```
int totale;

int contatore, temp, risultato;
```

Più variabili possono essere specificate in un'unica dichiarazione

Assegnamento

- Una ***istruzione di assegnamento*** modifica il valore di una variabile

```
totale = 55;
```

- **Semantica operazionale**:
- *L'espressione* alla destra del simbolo = è valutata e il suo valore viene assegnato alla variabile a sinistra
- L'eventuale valore precedente di **totale** è sovrascritto
- Si possono assegnare *solo* valori compatibili con il tipo dichiarato
- Esempio Geometry.java

Variabili e astrazione

- Il concetto di *variabile* è un'astrazione del concetto di locazione di memoria.
- L'assegnamento di un valore a una variabile è un'astrazione dell'operazione **STORE**

Variabili riferimento a un oggetto

- Una variabile può contenere un valore di un tipo primitivo o il ***riferimento*** a un oggetto
- Il nome di una classe può essere usato per dichiarare una variabile di *riferimento a un oggetto*

```
String titolo;
```

- Nessun oggetto viene creato in questa dichiarazione
- Si dichiara che la variabile di riferimento conterrà l'indirizzo di un oggetto
- L'oggetto stesso deve essere creato separatamente

```
titolo = new String ("Il manuale di Java");
```

Variabili e riferimenti

- L'istruzione di assegnamento per memorizzare un riferimento in una variabile

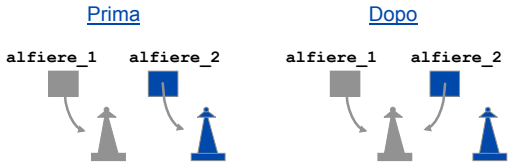
  `<sinistra> = <destra>;`
- `<sinistra>` è una variabile di un dato tipo e nome
- `<destra>` è un'espressione Java
  - Es: La chiamata al costruttore di una classe che genera un riferimento ad un nuovo oggetto
  - La classe deve essere coerente con quella dichiarata come tipo della variabile
- L'operatore di assegnamento = assegna il riferimento all'oggetto creato alla variabile

```
Rectangle rett = new Rectangle();
```


Assegnamento

- Per variabili riferimenti a oggetti, l'istruzione di assegnamento copia l'indirizzo di memoria:

```
alfiere_2 = alfiere_1;
```



Alias

- Due o più riferimenti allo stesso oggetto si chiamano ***alias***
- Un oggetto e i suoi dati possono essere visti mediante variabili diverse, che lo referenziano
- Gli alias sono utili, ma devono essere usati con cautela
- Cambiare lo stato di un oggetto (le sue variabili) tramite un riferimento, causa il cambiamento anche per tutti gli altri ***alias***

Garbage Collection

- Quando un oggetto non ha più un riferimento non può più essere referenziato da alcun programma
- L'oggetto diventa inaccessibile e quindi inutile
  - si chiama *garbage* (spazzatura)
- Java effettua una raccolta automatica e periodica degli oggetti inutili (*garbage collection*)
  - per rendere nuovamente utilizzabile per usi futuri lo spazio di memoria che l'oggetto occupava
  - Per ridurre lo spazio occupato dallo *heap*
- In altri linguaggi è responsabilità del programmatore effettuare il rilascio della memoria occupata da oggetti non referenziati e quindi inaccessibili

Riferimenti a oggetti

- Il riferimento descrive la posizione dell'oggetto sullo ***heap***
- Più variabili possono fare riferimento allo stesso oggetto

```
Rectangle scatola;
scatola = new Rectangle (5,10,15,20);
Rectangle contenitore = scatola;
```

- Ora `scatola` e `contenitore` si riferiscono allo stesso oggetto
- Esempio `TestRettangolo_2.java`

Espressioni

- Le espressioni sono sequenze di *operatori* e di *operandi* secondo le regole sintattiche del linguaggio.
- Hanno un *tipo complessivo* determinato dagli operatori e dal tipo degli operandi.
- Danno luogo, in fase di esecuzione, a un *valore*.
- In particolare le *espressioni di creazione di oggetti* hanno come tipo la classe dell'oggetto costruito e in fase di esecuzione producono come valore un *riferimento a un oggetto* della classe specificata dal costruttore.

Tipo di un dato

- Il tipo di una variabile specifica:
  - L'insieme dei valori che questa può assumere e
  - l'insieme delle operazioni che possono essere effettuate su di essa.
- Ad esempio una variabile `x` di tipo intero può assumere come valori solo numeri interi
- su di essa possono essere effettuate soltanto le operazioni consentite per i numeri interi.

Tipi e astrazioni

- La nozione di tipo fornisce un'astrazione rispetto alla rappresentazione effettiva dei dati.
- Tutte le variabili sono rappresentate in memoria come sequenze di bit, che possono essere interpretate diversamente in base ai tipi.
- Il programmatore può utilizzare variabili di tipi differenti, senza necessità di conoscerne l'effettiva rappresentazione.

Dichiarazione e inizializzazione

- Nella dichiarazione può essere già assegnato un valore iniziale alla variabile, mediante l'operatore d'assegnamento `=`

```
int somma = 0;
int base = 32, max = 149;
```

- Quando si richiama una variabile se ne usa il valore
- Esempio `PianoKeys.java`

La classe `ConsoleOutputManager`

- Una classe per la gestione dell'output
- Ogni oggetto della classe realizza un canale di comunicazione con il dispositivo di output standard, il video
- La classe mette a disposizione metodi per visualizzare a video vari tipi di dati
- Per poter inviare un messaggio a un oggetto della classe dobbiamo prima creare l'oggetto

```
new ConsoleOutputManager()
```


Esempio

```
import prog.io.ConsoleOutputManager;
Class PrimoProgramma {
  public static void main(String[] a) {
     ConsoleOutputManager video
           = new ConsoleOutputManager();
     video.println("primo esempio");
  }
}
```


Compilazione ed esecuzione

- `import …` costituisce una direttiva per il compilatore e la Java Virtual Machine
- L'argomento della direttiva di importazione `import` indica che la classe da cercare fa parte di un package o libreria
- I separatori `.` indicano le directory in cui cercare
  - `prog.io.ConsoleOutputManager`
  - In ambiente UNIX: `prog/io/ConsoleOutputManager.class`
  - In ambiente DOS/Windows: `prog\io\ConsoleOutputManager.class`

La classe `ConsoleInput Manager`

- Le sue istanze realizzano canali di comunicazione con il dispositivo di input standard, cioè la tastiera.
- Alcuni messaggi che possiamo inviare a un oggetto di tipo `ConsoleInputManager`:

- `readLine()`
  - permette di leggere una riga di testo
- `readInt()`
  - permette di leggere un numero intero

Esempio

```
import prog.io.ConsoleOutputManager;
import prog.io.ConsoleInputManager;
class Pappagallo {
  public static void main(String[] args) {
    // i canali di comunicazione
    ConsoleInputManager tastiera =
              new ConsoleInputManager();
    ConsoleOutputManager video =
              new ConsoleOutputManager();
    // lettura e comunicazione
    String messaggio = tastiera.readLine();
    video.println(messaggio);
  }
}
```


Commenti

- Il metodo `readLine` *restituisce* un valore; un riferimento a un oggetto di tipo `String`
- L'*espressione* `tastiera.readLine()` è di tipo `String`

… come in …

```
import prog.io.*;
class Saluto {
  public static void main( String[] args) {
    // i canali di comunicazione
    ConsoleInputManager tastiera =
                new ConsoleInputManager();
    ConsoleOutputManager video =
                new ConsoleOutputManager();
    // lettura nome da tastiera
    String nome =
           tastiera.readLine("Come ti chiami?");
    // comunicazione
    video.print("Buongiorno ");
    video.print(nome);
    video.println("!");
  }
}
```


Commenti

- Si noti che abbiamo usato il metodo `readLine` con un parametro di tipo `String`
- Si tratta di esempio di **overloading** o **sovraccaricamento** di metodi