Progettazione e Programmazione Orientata agli Oggetti

In Collaborazione con Paola Turci (Università di Parma)

Paradigmi di Programmazione

- Un linguaggio supporta uno stile di programmazione se possiede funzioni che ne rendono conveniente (semplice, sicuro, efficiente) l'uso:
  - ***librerie standard***
  - ***ambienti di programmazione***
  - ***controlli in compilazione e/o in esecuzione***

Programmazione Procedurale

- *Definire le procedure*

I linguaggi supportano questo paradigma se forniscono la possibilità di:

- ***passare argomenti***
- ***restituire valori***

Programmazione Modulare

- *Definire i moduli*

Realizzazione delle procedure → Organizzazione dei dati

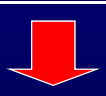
- Modulo: insieme di procedure poste in relazione ai dati che utilizzano (pila, lista, ...)
- Risulta sufficiente lo stile di programmazione procedurale se non esiste la necessità di definire un insieme di procedure con dati associati

Tipo di Dato Astratto

- *Definire tipi di dato e insieme completo di operazioni*
  - I tipi di dato astratto (o tipi definiti dall'utente) presentano caratteristiche simili ai tipi di dato predefiniti dal linguaggio
    - visibilità, controlli in fase di compilazione, ecc.
  - Se non esiste la necessità di più istanze di un tipo di dato astratto, risulta sufficiente lo stile di programmazione con *"dati nascosti"* (programmazione modulare)
  - Il tipo di dato astratto è, in generale, poco flessibile; non è possibile adattarlo ad un nuovo utilizzo, se non modificandone la definizione.

Paradigma di Programmazione ad Oggetti

- Occorre uno strumento che consenta di operare una distinzione tra le proprietà di tipi di dati più generali e le proprietà specifiche di singoli sottotipi
  - Nota: se non vi è alcuna similitudine tra i tipi del problema è sufficiente l'astrazione di dato.



Programmazione Orientata agli Oggetti

- Analisi del problema in termini di concetti (oggetti) che lo costituiscono (modello concettuale del problema)

Paradigma di Programmazione ad Oggetti

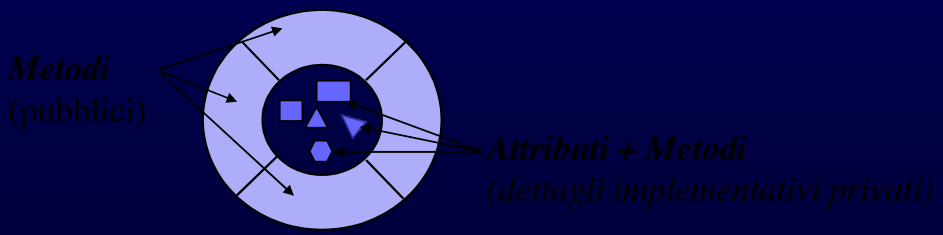
- Intende promuovere nei programmi caratteristiche ritenute fondamentali:
  - *Modularità*
  - *Information hiding (incapsulamento)*
  - *Riusabilità*
  - *Progettazione incrementale*
  - *Rapida prototipizzazione*
- I linguaggi di programmazione orientati agli oggetti sono accomunati da tre fattori:

Incapsulamento - Estendibilità - Polimorfismo

  - Incapsulamento, consente di **nascondere i dati** ed **esporre solo** (alcuni) **metodi**
  - Estendibilità, introduce il concetto di classificazione degli oggetti.
  - Polimorfismo, è caratterizzato dalla frase *"un'interfaccia più metodi"*.

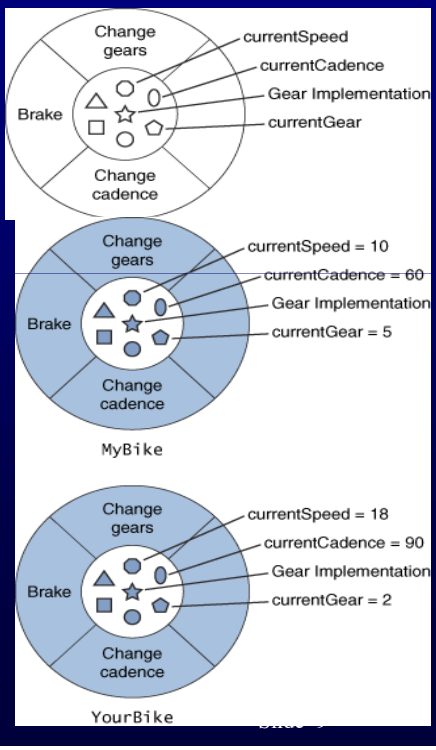
Gli Oggetti

- Un oggetto è un'area di memoria inizializzata
- Un oggetto è un'aggregazione software di attributi (*stato*) e relativi metodi/servizi (*comportamento*)
- Un oggetto è un'***astrazione*** che descrive oggetti reali del problema e concetti astratti che fanno parte della sua soluzione



Concetto di Classe

- "Entitá"/tipo che definisce le proprietà (attributi, metodi) che accomunano una famiglia di oggetti:
  - Uno stesso insieme di operazioni
  - Una stessa struttura esterna
  - Uno stesso comportamento
- Consente la creazione di istanze (oggetti)
- Riunisce proprietà dei tipi di dato e dei moduli



Classi e Istanze (Oggetti)

- Un oggetto è un'*istanza* di una classe (che è il suo *tipo* ed il *modulo* che ne regola l'accesso).
  - Istanziare un oggetto di una determinata classe significa:
    - Dichiarare una variabile di quel tipo
    - Allocare dinamicamente spazio per un oggetto di quel tipo
  - Oggetti della stessa classe hanno operazioni uguali (*metodi*), ma specifici valori per gli attributi (dati diversi).
  - Un metodo può accedere ai dati dell'oggetto (parte privata del modulo).
- Come si indica al metodo su quale oggetto agire ?
  - ***Istanza corrente*** di un oggetto (`this` o `self`).

Chiamata a Metodo

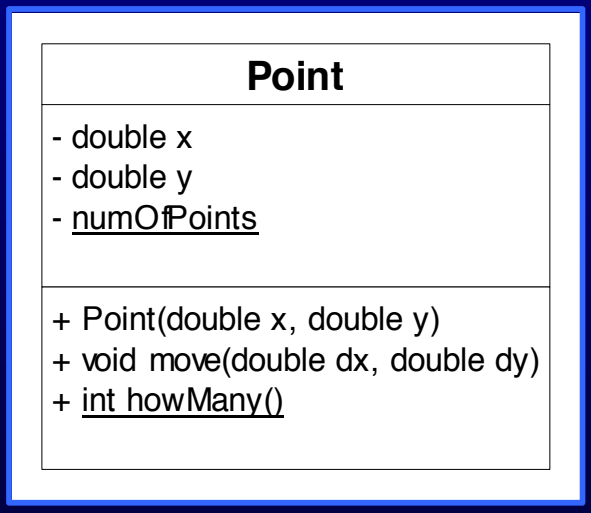
Meccanismo fondamentale di computazione della OOP

res = obj.meth(par)

Programma = Rete di Oggetti

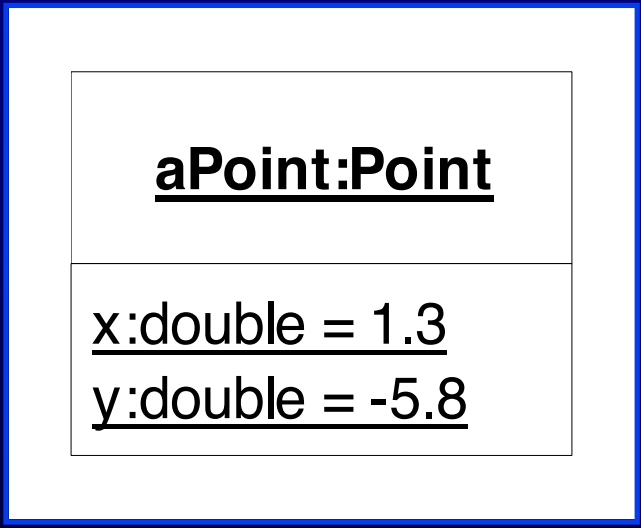
- ***Tutto*** è un oggetto
  - Ogni componente concettuale del problema viene rappresentata da un oggetto.
  - Un programma è costituito da un insieme di oggetti che si scambiano messaggi (ordini e richieste).
  - Ogni oggetto è fatto di altri oggetti (o di tipi base).
- I programmi saranno costituiti da una rete di **oggetti** connessi da relazioni *client-server*.
- La rete che rappresenta l'architettura del programma avrà spesso una struttura gerarchica.

Classi



- Una classe *Point*, espressa nella notazione standard *UML* (*Unified Modeling Language*).
  - Dati privati.
  - Funzioni pubbliche.
  - Membri di istanza.
  - Membri di classe.

Oggetti



- Un oggetto in notazione *UML*.
  - Un oggetto è un'*istanza* di una classe (che è il suo *tipo* ed il *modulo* che ne regola l'accesso).
  - Un oggetto ha struttura ed operazioni della sua classe, ma specifici valori per gli attributi.

Ereditarietà

- L'ereditarietà è un meccanismo base della OOP, ma ha più aspetti diversi.
- La prima distinzione da fare è tra ***ereditarietà di interfaccia*** ed ***ereditarietà di implementazione***.

Subtyping

- Se le classi sono tipi, l'ereditarietà è una relazione tra tipi, che chiamiamo *subtyping*.
- Se il tipo *C* è un sotto-tipo del tipo *B*, cosa significa *C is-a B* ?
  - **Tutto quello che posso fare ad un B lo posso fare anche ad un C**.

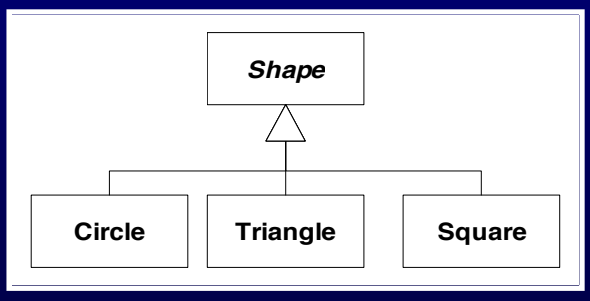
⇒ ***Principio di Sostituibilità di Liskov.***

Inheritance

- L'ereditarietà è una relazione tra moduli, che chiamiamo *inheritance*.
- L'*inheritance* garantisce l'*Open/Closed Principle*.
  - Un sotto-modulo avrà accesso privilegiato al modulo base (accesso *protected* in C++ e Java).
  - Un sotto-modulo potrà cambiare o estendere l'implementazione del modulo base.

Sintassi per l'ereditarietà in Java

Estensione singola

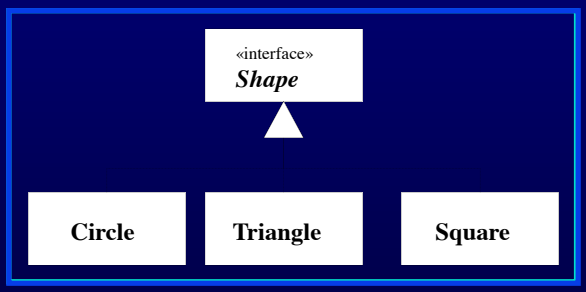



```
public abstract class Shape {
  // …
}

class Square extends Shape {
  // …
}
```

Sintassi per l'ereditarietà in Java

Implementazione singola

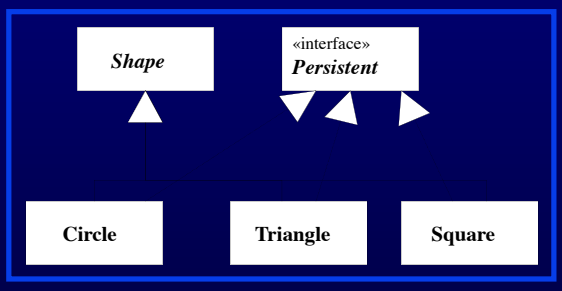



```
public interface Shape {
  // …
}

class Square implements Shape {
  // …
}
```


Sintassi per l'ereditarietà in Java

Derivazione multipla

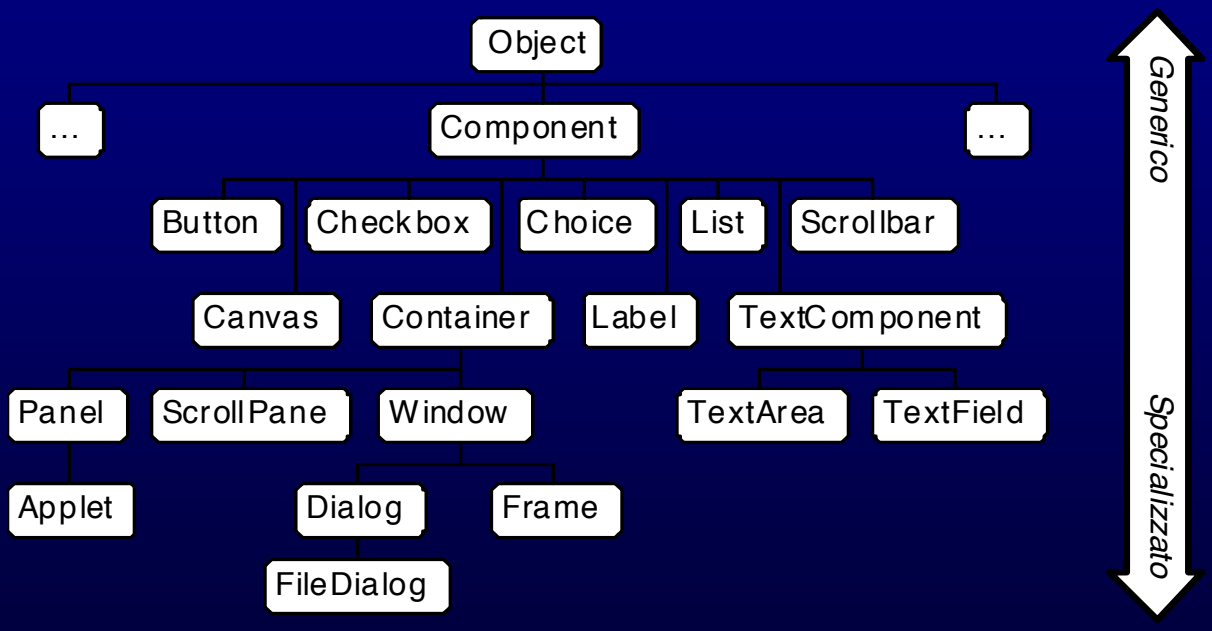



```
public abstract class Shape {
  // ...
}

public interface Persistent {
  // ...
}

class Square extends Shape
  implements Persistent {
  // ...
}
```

Gerarchie di ereditarietà

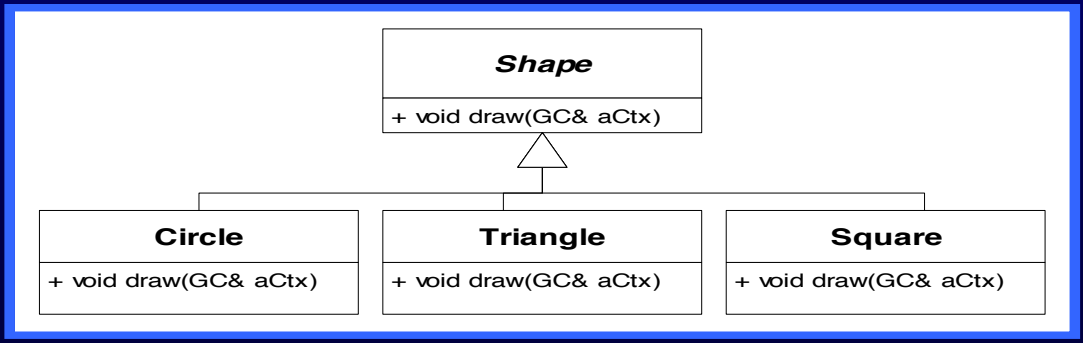


Polimorfismo

- Il polimorfismo è una caratteristica essenziale dei linguaggi orientati agli oggetti.
- Permette di trattare in modo uniforme tipi *"simili"*, ma rispettandone le differenze.
- Spesso il polimorfismo si usa per garantire il *Single Choice Principle*.

Polimorfismo

- Si individuano delle operazioni comuni per interfaccia, ma diverse per comportamento.
- Una classe dichiara la funzione e delle sottoclassi ne forniscono diverse implementazioni.



Polimorfismo

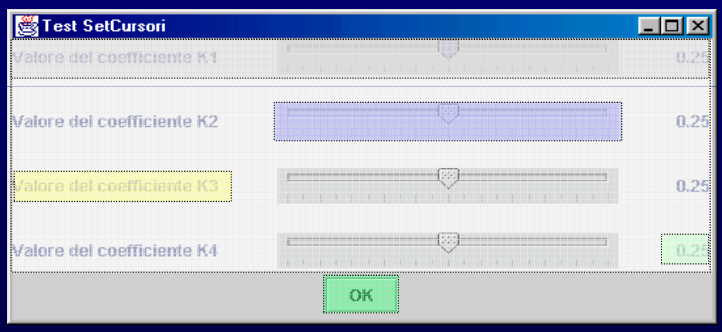
- Per il Principio di Sostituibilità:
  - Un riferimento di tipo ***`Shape`*** può puntare a oggetti di tipo ***`Shape`*** o *di una sottoclasse*.
- Esiste un ***tipo statico*** ed un ***tipo dinamico***:

Tipo statico → **tipo del riferimento, dichiarato a *compile time***

Tipo dinamico → **tipo dell'oggetto puntato a *run time***

Relazioni *part–of*

- Tra gli oggetti spesso sussiste una relazione ***part–of*** (detta anche **di composizione**).



Delega

- Attraverso la relazione *part–of* è possibile realizzare un importante comportamento: la **delega** (*delegation*).
  - Una classe demanda ad un'altra classe alcune parti del servizio che la prima deve rendere disponibile.
- Modularità nel codice: ogni classe si occupa di realizzare una specifica funzionalità, senza mescolare aspetti concettualmente diversi tra loro
- Nel disegnare l'architettura di un programma dobbiamo decidere quale tipo di relazione utilizzare per connettere le nostre classi:

  ***is–a* (ereditarietà)** **o** ***part–of* (delega)**

OOP

- Le tecnologie OO permeano tutto il ciclo di sviluppo del software
  - ***OOA*** - *(Object Oriented Analysis)*
  - ***OOD*** - *(Object Oriented Design)*
  - ***OOP*** - *(Object Oriented Programming)*
- Le soluzioni proposte dalla OOP nascono da considerazioni generali di Ingegneria del Software.
  - Le tecnologie orientate agli oggetti cercano di favorire la progettazione di ***software di qualità***.