Introduzione ai Linguaggi di Programmazione e al paradigma Object Oriented

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Sommario

- Introduzione ai linguaggi di programmazione
- Principi base del paradigma O.O.
  - Astrazione
  - Oggetto e Classe
  - Incapsulamento e Information Hiding
  - Modularità
  - Ereditarietà
  - Polimorfismo
- Java vs C++

Introduzione ai linguaggi di Programmazione

» Un linguaggio di programmazione è un insieme di parole, codici, e simboli che permettono al programmatore di dare istruzioni al calcolatore

» Diversi linguaggi di programmazione sono stati concepiti nel corso degli anni, ognuno con le proprie regole, o sintassi per scrivere tali istruzioni

Introduzione ai linguaggi di Programmazione

- Tutti i linguaggi di programmazione forniscono astrazioni
- La complessità dei problemi che si è in grado di risolvere è direttamente correlata al genere e alla qualità dell'astrazione considerata
  - Il linguaggio assembler è una piccola astrazione della macchina sottostante
  - I cosiddetti linguaggi "imperativi" (Fortran, BASIC, C) sono astrazioni del linguaggio assembler anche se comunque continuano ad imporre di pensare in termini di struttura del calcolatore invece che della struttura del problema che si sta risolvendo

Introduzione ai linguaggi di Programmazione

» Tocca al programmatore stabilire l'associazione fra il modello della macchina (nello "spazio delle soluzioni") e il modello del problema che viene risolto (nello "spazio dei problemi")

» Anziché modellare una macchina per risolvere il problema si può pensare di modellare il problema

Storia dei linguaggi di programmazione

» Fortran (Formula translator)

> 1954-1956 da John Backus presso IBM

> espressioni esprimibili per mezzo di notazioni matematiche ordinarie

> i+2*j

> primo linguaggio ad introdurre le variabili, cicli, procedure, etc.

> nessun supporto per ricorsione e gestione implicita della memoria

» Cobol

> 1959 da Grace Murray Hopper

> applicazioni finanziarie

> sintassi definita in modo da rendere la scrittura di istruzioni simile alla scrittura di frasi nel linguaggio naturale

> e.g., “add 1 to x giving y” (y = x+1)

Storia dei linguaggi di programmazione

» Lisp

> 1965 da John McCarthy

> funzioni high-order

> garbage collection

> Successori del LISP: Miranda, ML, Haskell anche se non particolarmente usati

» Algol

> 1958-1960 da un comitato internazionale di informatici

> un migliore sistema dei tipi

> prime strutture dati

» Entrambi hanno ricorsione, funzioni e procedure

Storia dei linguaggi di programmazione

» BASIC

> Il primo linguaggio di programmazione pensato per uso personale

> Semplice da imparare anche se limitato

> Le versioni correnti del linguaggio non sono così “basic” e semplici da imparare

» SIMULA 67

> Estensione di Algol 60 progettato per la simulazione di processi concorrenti

> Introduce i concetti base dell’object oriented: classe ed incapsulamento

> Predecessore di Smalltalk e C++

Storia dei linguaggi di programmazione

» Pascal

> Successore di Algol 60.

> Ottimo linguaggio per introdurre la programmazione strutturata

> Un buon primo linguaggio da imparare

» Modula-2

> Successore di Pascal

> Meccanismi per gestire la concorrenza (più processi in parallelo)

» Ada

> Progettato per supportare in maniera efficace la concorrenza

> Ci sono due standard: Ada 83 (l'originale), and Ada 95

Storia dei linguaggi di programmazione

» C

> Il linguaggio di implementazione di Unix

> Pericoloso se non usato opportunamente: non raccomandato per i programmatori inesperti

> Relativamente di basso livello

» Smalltalk

> Linguaggio di programmazione object-oriented più pulito di Java e molto più pulito di C++

> Viene fornito con un'interfaccia grafica ed un ambiente di programmazione integrato

Storia dei linguaggi di programmazione

» C++

> Estensione object-oriented del linguaggio imperativo C

> Sintassi complicata con difficile semantica

> Molto richiesto

» Java

> Rielaborazione del C++

> Full object orientation (anche se non ai livelli di Smalltalk)

> Progettato per la programmazione Internet programming, anche se general-purpose

> Solitamente è considerato essere lento

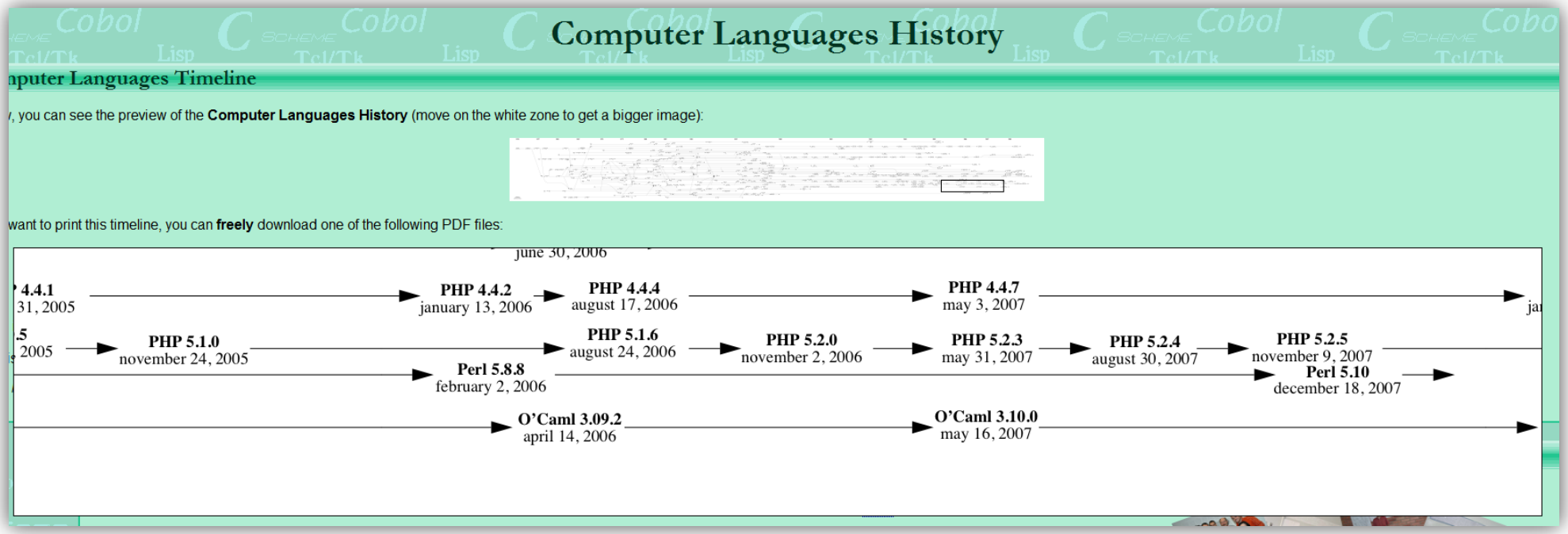
> Forse il prossimo standard de-facto

Storia dei linguaggi di programmazione

- Linguaggi di Scripting
  - Text processing:
    - Perl
    - Python
  - Web programming
    - JavaScript
    - PHP

Computer Languages History

» Per maggiori dettagli
http://www.levenez.com/lang/



Principi base del paradigma O.O.

- Astrazione
- Incapsulamento & Information Hiding
- Modularità
- Gerarchia
- Polimorfismo

Premesse al paradigma O.O.

- Ciascuno dei linguaggi di programmazione visti precedentemente può essere un'ottima soluzione per risolvere la particolare classe di problemi per la quale è stato concepito
- Non appena si esce da questo ambito il linguaggio di programmazione in questione risulta essere inadeguato

Astrazione

"Qualsiasi modello che include gli aspetti più importanti o essenziali di qualcosa mentre ignora i dettagli meno importanti, immateriali. Il risultato è di rimuovere le differenze ed enfatizzare gli aspetti comuni"

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Astrazione

- Permette di gestire la complessità concentrandosi sulle caratteristiche essenziali di un'entità che la distingue dalle altre
- E' dipendente dal dominio e dalla prospettiva, cioè quello che è importante in un contesto potrebbe non esserlo in un altro
  - def. macchina per un venditore diversa da quella di un progettista
- OO modella il sistema utilizzando l'astrazione (es. classi)

Astrazione

- L’approccio orientato agli oggetti mette a disposizione strumenti con i quali il programmatore può rappresentare elementi nello spazio dei problemi
- La rappresentazione è sufficientemente generalizzata da non vincolare il programmatore a occuparsi soltanto di un determinato tipo di problemi
- Gli oggetti sono elementi che si trovano nello spazio dei problemi e le loro rappresentazioni nello spazio delle soluzioni
- L’OOP permette quindi di descrivere il problema nei suoi termini propri, invece che nei termini del computer sul quale verrà computata la soluzione

Classe

- Descrizione di un gruppo di oggetti con proprietà (attributi), comportamento (operazioni), relazioni e semantica comuni
- Astrazione che
  - Enfatizza caratteristiche rilevanti
  - Sopprime le altre caratteristiche
- *Principio O.O. Astrazione*

Esempio di classe

» Nome
  – Corso
» Proprietà
  – Nome, Luogo, Durata, Crediti, Inizio, Fine
» Comportamento
  – Aggiunta studente
  – Cancellazione studente
  – Verifica se è pieno

Oggetto (1)

» Informalmente rappresenta
  – entità fisica: trattore
  – entità concettuale: processo chimico
  – oppure entità software: lista, coda...

» Formalmente
  – Manifestazione concreta di un'astrazione
  – Entità con un confine e un'*identità* ben definite che incapsula *stato* e *comportamento*
  – **Istanza** di una classe

» Es.: Ferrari di Schumacher, Ferrari di Barrichello, Mio computer

Oggetto (2)

» Stato
- Possibile condizione nel quale l'oggetto potrebbe esistere e generalmente cambia nel tempo
- Implementato mediante proprietà (attributi) con valori, e collegamenti ad altri oggetti

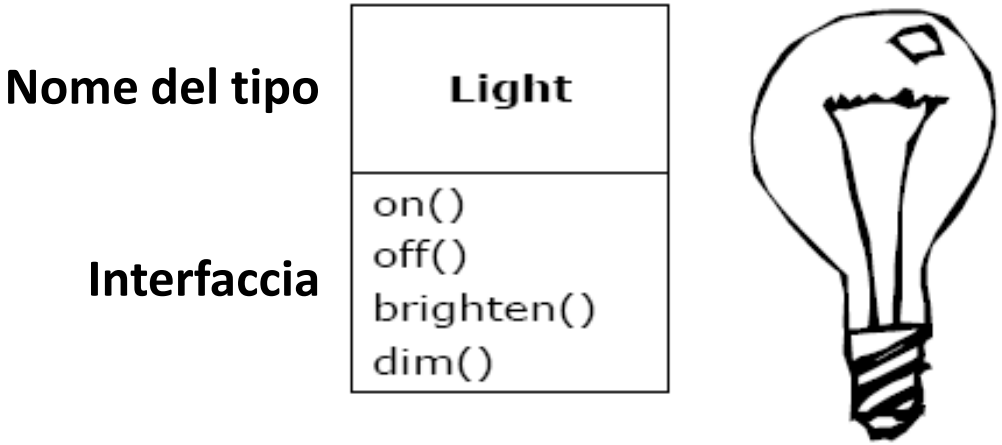
» Comportamento
- Determina come un oggetto agisce e reagisce alle richieste di un altro oggetto
- Rappresentato dall'insieme di messaggi a cui può rispondere (operazioni)

» Identità
- Rende possibile la distinzione tra due oggetti anche se hanno lo stesso stato e lo stesso valore nei suoi attributi

Relazione tra classe ed oggetto

- Classe è una definizione astratta di un oggetto
  - Definisce la struttura e il comportamento di ogni oggetto nella classe
  - Serve come *template* per creare oggetti
- Oggetti sono *raggruppati* in classi

Oggetto ed Interfaccia




```
Light lt = new Light();
lt.on();
```

- Le richieste che si possono fare ad un oggetto sono definite dalla sua *interfaccia* ed è il tipo a determinare l'interfaccia
- Il codice deputato a soddisfare tali richieste, insieme con i dati nascosti, costituisce *l'implementazione* dell'oggetto

Caratteristiche della OOP

Alan Kay ha riepilogato 5 caratteristiche essenziali di Smalltalk

1. Ogni cosa è un oggetto
2. Un programma è un insieme di oggetti che si dicono l'un l'altro che cosa fare **inviandosi messaggi**
3. Ogni oggetto ha la sua memoria formata da altri oggetti
4. Ogni oggetto ha un tipo
5. Tutti gli oggetti di un determinato tipo possono ricevere gli stessi messaggi

Incapsulamento & Information Hiding

"La localizzazione fisica di caratteristiche (es. proprietà, comportamento) in una singola scatola nera che nasconde l'implementazione (e le relative decisioni di design) dietro una interfaccia pubblica"

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Incapsulamento & Information Hiding

- Incapsulamento
  - L'incapsulamento è la proprietà per cui un oggetto contiene ("incapsula") al suo interno gli attributi (dati) e i metodi (procedure) che agiscono su di essi
- Information Hiding
  - Nascondere ai possibili client (utilizzatori) le scelte interne di design e gli effetti che eventuali cambiamenti di tali decisioni comportano
- Incapsulamento è una **facility** del linguaggio mentre l'Information Hiding è un **principio di design**

Information Hiding

- In generale i programmatori si distinguono in *creatori di classe* e *programmatori client*
- L'obiettivo del programmatore client è mettere insieme una serie di classi per sviluppare rapidamente le applicazioni
- L'obiettivo del creatore di classi è costruire una classe che mostra al programmatore client soltanto quello che è necessario, mantenendo nascosto tutto il resto
  - Se tutti i membri di una classe fossero disponibili a tutti, un programmatore client potrebbe fare qualsiasi cosa con quella classe e non vi sarebbe così modo di imporre delle regole

Incapsulamento & Information Hiding

» Esempio

> Sistema che identifica un punto sulla superficie terrestre e offre funzionalità per determinare la **distanza** (distance) e la **rotta** (heading) tra due punti

Esempio: prima soluzione


```
public class Position {

   public double latitude;
   public double longitude;

}

public class PositionUtility {

   public static double distance( Position pos1, Position pos2 ) {

     //Calculate and return the distance between the specified positions

   }


   public static double heading( Position pos1, Position pos2 ) {

     //Calculate and return the heading from pos1 to pos2.

   }

}
```

Esempio: prima soluzione


```
Position myHouse = new Position();

myHouse.latitude = 36.538611;

myHouse.longitude = -121.797500;


Position coffeeShop = new Position();

coffeeShop.latitude = 36.539722;

coffeeShop.longitude = -121.907222;


double distance = PositionUtility.distance( myHouse, coffeeShop );

double heading  = PositionUtility.heading( myHouse, coffeeShop );


System.out.println("From my house at (" + myHouse.latitude + ", " + myHouse.longitude +
  ") to the coffee shop at (" + coffeeShop.latitude + ", " + coffeeShop.longitude + ")
  is a distance of " + distance + " at a heading of " + heading + " degrees.");
```

Esempio: prima soluzione

» OUTPUT

```
From my house at (36.538611, -121.7975) to the coffee shop
at (36.539722, -121.907222) is distance of
6.0873776351893385 at a heading of 270.7547022304523
degrees.
```

Considerazioni

» Dati (espressi mediante `Position`) ed operazioni (espresse mediante `PositionUtility`) sono separati!!!

Esempio: seconda soluzione


```
public class Position {
   public double distance( Position position ) {
     //Calculate and return the distance from this object to   the specific position.
   }

   public double heading( Position position ) {
     //Calculate and return the heading from this object to    the specified position.
   }

   public double latitude;
   public double longitude;
}
```

Esempio: seconda soluzione


```
Position myHouse = new Position();
myHouse.latitude  = 36.538611;
myHouse.longitude = -121.797500;

Position coffeeShop = new Position();
coffeeShop.latitude  = 36.539722;
coffeeShop.longitude = -121.907222;

double distance = myHouse.distance( coffeeShop );
double heading = myHouse.heading( coffeeShop );

System.out.println("From my house at (" + myHouse.latitude + ", " +
  myHouse.longitude + ") to the coffee shop at (" + coffeeShop.latitude +
  ", " + coffeeShop.longitude + ") is a distance of " + distance + " at a
  heading of " + heading + " degrees.");
```

Modularità

"La decomposizione fisica e logica di cose (es. responsabilità e software) in gruppi piccoli e semplici (es. requisiti e classi rispettivamente) che incrementa il raggiungimento degli obiettivi dell'ingegneria del software"

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Modularità

- Per gestire la complessità si può suddividere qualcosa che è complesso in pezzi più piccoli che sono più maneggevoli
- O.O. modella la modularità con **package** e subsystem
- Un package è un *namespace* che organizza un insieme di classi ed interfacce corrispondenti
- Concettualmente un package è simile ad una directory nel file system. Possiamo quindi memorizzare file HTML in una directory, immagini in un'altra, e script in un'altra ancora
- Tipicamente, sistemi complessi consistono di migliaia di classi, ha quindi senso organizzare le classi e le interfacce in package

Ereditarietà

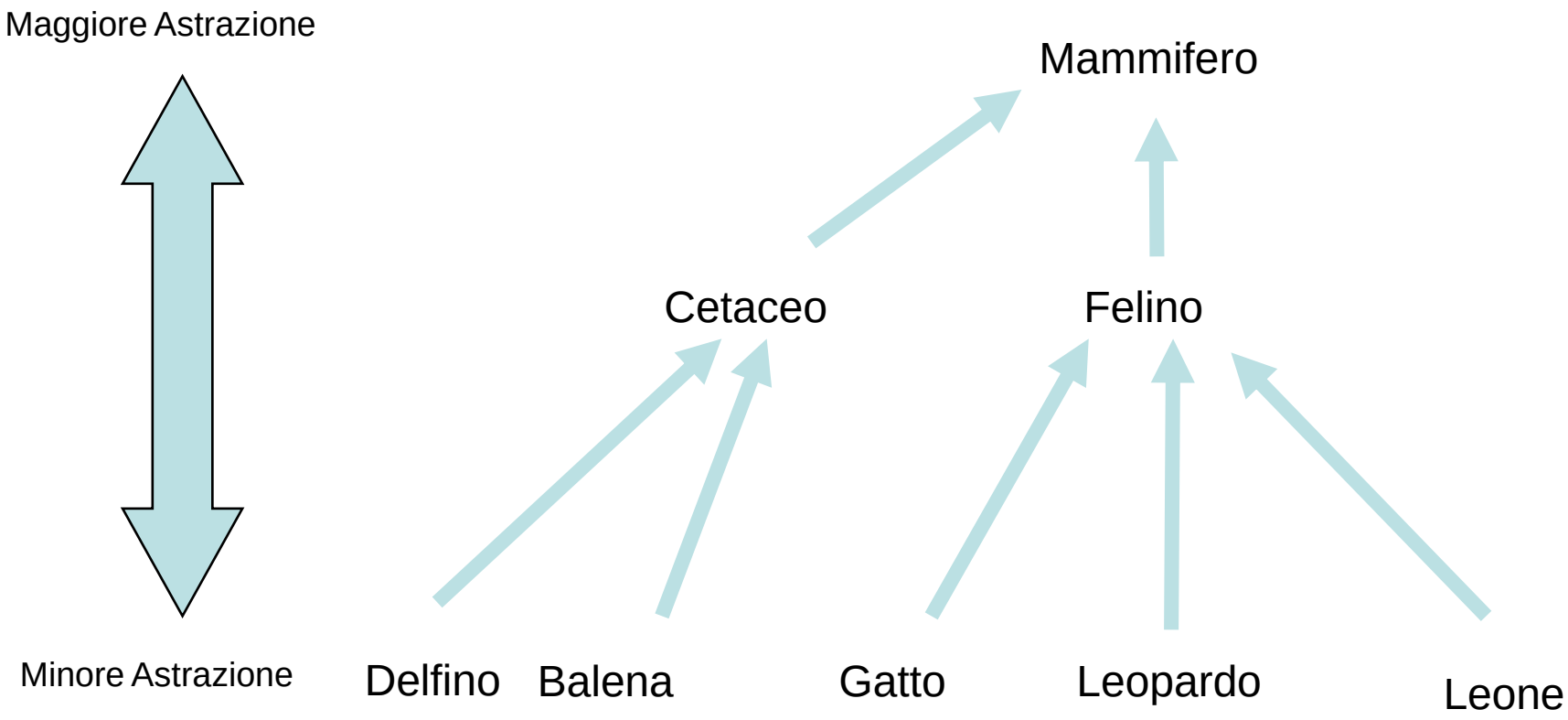
"Qualsiasi graduatoria (ranking) o ordine di astrazione in una struttura ad albero"

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Ereditarietà

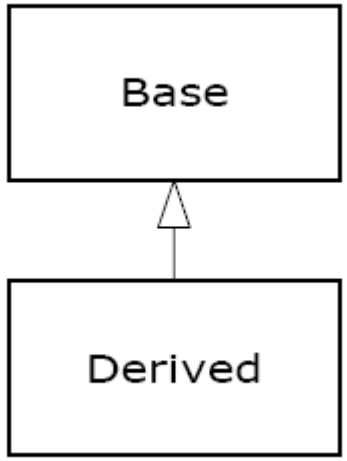
- Permette di organizzare un qualcosa in base ad un certo ordine
  - Esempi: complessità, responsabilità
- Descrive differenze o variazioni di un particolare concetto

Ereditarietà



***Fratelli* hanno stesso livello di astrazione**

Ereditarietà

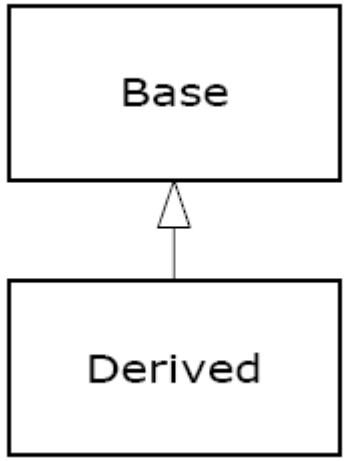


» Due tipi possono avere caratteristiche e comportamenti in comune, ma uno può contenere più caratteristiche dell'altro e può anche gestire più messaggi (o gestirli in modo diverso)

» L'ereditarietà esprime questa similarità fra tipi utilizzando i concetti di tipi base e tipi derivati

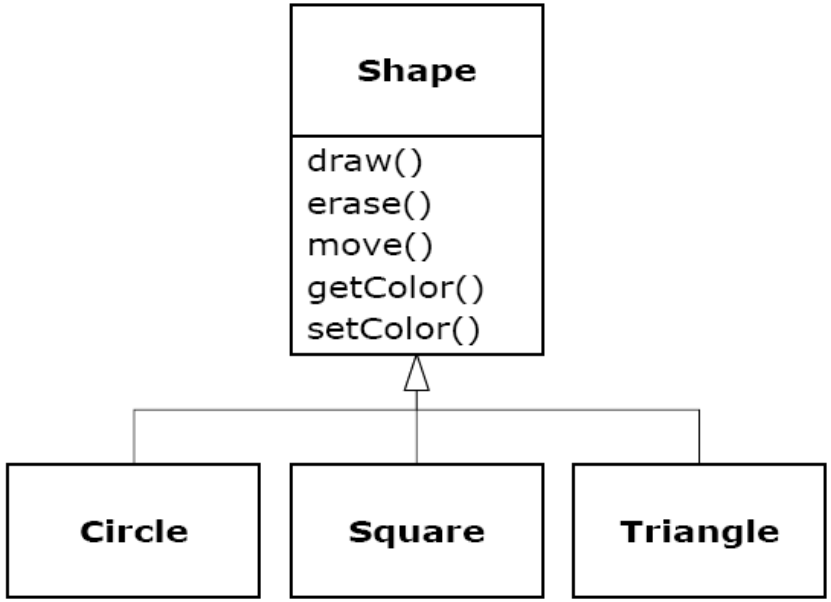
» Un tipo base contiene tutte le caratteristiche e i comportamenti che sono condivisi dai tipi che da esso derivano

Ereditarietà



- Con l'ereditarietà la classe *base* viene "clonata" e modificata da quella *derivata*
- Se la classe base viene modificata anche la classe derivata riflette tali cambiamenti

Ereditarietà > Esempio



» I tipi derivati (Circle, Square, Triangle) possono avere ulteriori caratteristiche e comportamenti di quello base (Shape)

» Alcuni comportamenti possono essere diversi, come quando si vuole calcolare l'area di una forma geometrica

» La gerarchia dei tipi incorpora sia le somiglianze che le differenze fra le forme geometriche

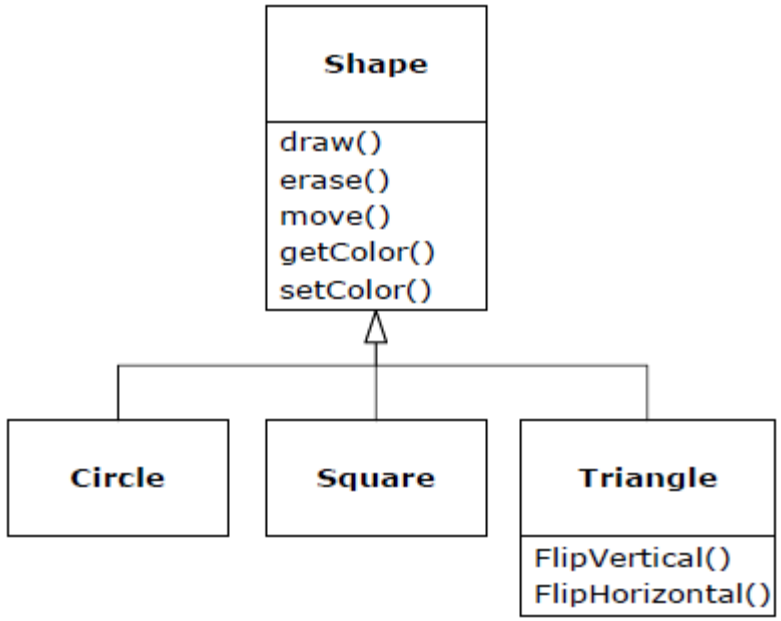
Ereditarietà

- Quando si eredita da un tipo esistente, il tipo nuovo duplica l'interfaccia della classe base
  - Tutti i messaggi che si possono inviare a oggetti della classe base si possono inviare anche a oggetti della classe derivata
- Siccome il tipo di una classe è identificato dai messaggi che si possono inviare, la classe derivata è *dello stesso tipo della classe base*
  - Considerando l'esempio precedente, *Un cerchio è una forma geometrica*

Ereditarietà

» Ci sono due modi per differenziare la nuova classe derivata dalla classe base originale

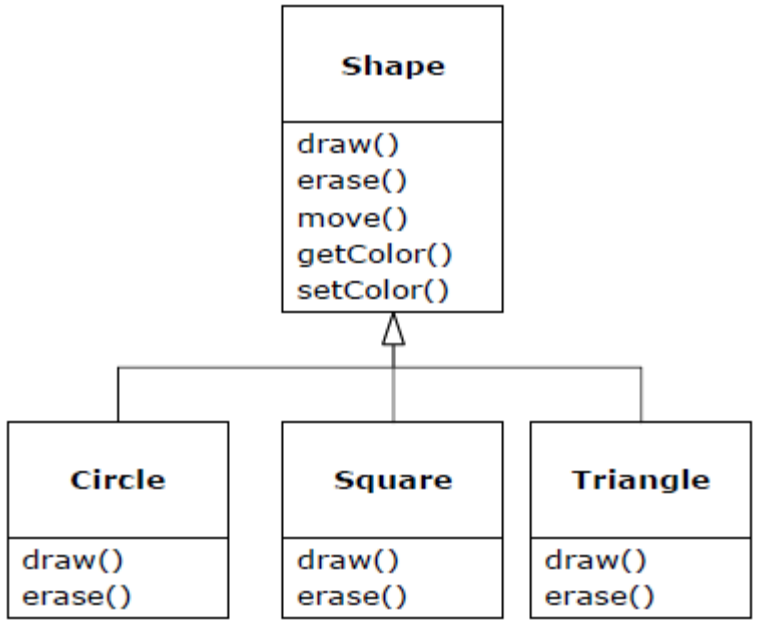
1. Si aggiungono nuovi metodi alla classe derivata



Ereditarietà

» Ci sono due modi per differenziare la nuova classe derivata dalla classe base originale

2. Si cambia il comportamento di un metodo della classe base (ridefinizione di metodo)



Polimorfismo

- Linguaggi procedurali (esempio Pascal) basati su idea che procedure e funzioni, e i loro operandi, hanno un unico tipo
- Tali linguaggi sono detti *monomorphic,* cioè ogni valore e variabile può avere uno ed un solo tipo
- Linguaggi O.O. sono detti *polymorphic,* cioè i valori e le variabili possono avere più di un tipo
- Dal greco *polymorphos* “avere molte forme”

Polimorfismo


```
void doStuff(Shape s) {
  s.erase();
  // ...
  s.draw();

}
```

» Questo metodo parla con qualunque Shape, quindi è indipendente dal tipo di oggetto che sta disegnando e cancellando

Polimorfismo


```
Circle c = new Circle();
Triangle t = new Triangle();
Line l = new Line();
doStuff(c);
doStuff(t);
doStuff(l);
```

- La chiamata a `doStuff()` funziona automaticamente in modo corretto, indipendentemente dal tipo esatto dell'oggetto
- Questo è reso possibile dal *late binding*
  - Quando si invia un messaggio ad un oggetto, il codice che viene chiamato non è determinato fino al momento dell'esecuzione

Polimorfismo


```
…
Shape s = null;
if (x <= 3) {
   s = new Circle();
} else {
   s = new Triangle();
}
s.draw();
…
```

Polimorfismo


```
…
Shape s = null;
if (x <= 3) {
    s = new Circle();
} else {
    s = new Triangle();
}
s.draw();
…
```

- Posso invocare il metodo `draw()`?
  Compile time
-Quale metodo `draw()` deve essere invocato ?
  Run-time

Punti di forza dell'O.O.

- Singolo paradigma
- Facilita il riuso di codice e di architetture
- Modelli riflettono maggiormente la realtà
  - Descrizione più accurata dei dati e processi
  - Decomposizione basata su partizionamento naturale
  - Più facile da comprendere e mantenere
- Stabilità
  - Piccolo cambiamento nei requisiti non significa massicci cambiamenti nel sistema durante lo sviluppo

Java vs C++

- Sia C++ che Java sono linguaggi ibridi
- Un linguaggio ibrido permette molteplici stili di programmazione
- La ragione per cui C++ è ibrido è quella di garantire la compatibilità all'indietro con il linguaggio C
  - C++ comprende molte delle caratteristiche indesiderabili di C, il che può rendere alcuni aspetti del C++ eccessivamente complicati
- Java presuppone che si voglia fare solo programmazione orientata agli oggetti

Java vs C++

- **Java non supporta le strutture (*struct*), le unioni (*union*)**
  sono rese inutili dalla presenza delle classi
- **Java non supporta più i puntatori**
  sono stati eliminati poiché sono la maggior causa degli errori in C/C++
- **Java non supporta le funzioni**
  essendo un linguaggio orientato agli oggetti, Java forza i programmatori a considerare le funzioni come metodi della classe

Java vs C++

- **Java non supporta il preprocessore, il typedef e i files header**
  Java fornisce funzionalità simili ma con un maggior controllo
- **Java non supporta il coercion automatico**
  si riferisce alla conversione implicita di tipi di dato che alcune volte si verifica in C e C++
- **Java non permette l'overloading degli operatori**
- **Java non supporta l'ereditarietà multipla**
  è una caratteristica del C++ che permette di derivare una classe da più classi base

Java vs C++

- **Java gestisce gli argomenti della linea di comando in modo diverso dal C o C++**
- **Java ha una classe String come parte del package java.lang**
  questo differisce dall'array di caratteri terminante con null del C e C++
- **Java ha un sistema automatico per allocare e liberare la memoria (garbage collection)**
  non è necessario utilizzare funzioni di allocazione e deallocazione della memoria come in C e C++