PARADIGMA OBJECT ORIENTED

PARADIGMI DI PROGRAMMAZIONE

- procedurale (Pascal, C)
- a oggetti (C++, Java)
- funzionale (LISP, Scala)
- logico (Prolog)

PARADIGMA PROCEDURALE

- E' stato trattato in maniera esclusiva nel primo corso di Informatica
- Sia la definizione del Problema che della Soluzione sono incentrate su di una singola elaborazione da risolvere
  - Il risultato è una *procedura* o *funzione* che implementa un *algoritmo*
- Non vengono invece curati molto gli aspetti di
  - Modellazione dei dati
  - Raggruppamento delle funzioni
  - …

ESEMPIO PROCEDURALE

```
int voto[20];
void sort(int [] v, int size) { // sort };
int search(int [] v, int size, int c) { // search };)
// …
int i;
void main(){
    for (i=0; i<20; i++) { voto[i]=0; };
    sort(voto, 20);
    search(voto, 20, 23);
}
```

- L'analisi è stata centrata sul *problema*, che è stato risolto scomponendolo in due *funzioni*
- Per ogni funzione sono stati individuati gli *input* e gli *output*
- Sono state progettate le chiamate (*call*) alle funzioni e il loro ordine
- (per l'inizializzazione del vettore non è stata utilizzata una funzione)

POTENZIALI PROBLEMI

- Concettualmente abbiamo scritto un algoritmo che opera su un vettore di voti ma in pratica non abbiamo definito un *tipo* vettore o un tipo *voto* riutilizzabile in altri problemi

  - Non c'è un legame tra le funzioni che operano sul vettore e la variabile *voto*
  - Non viene valutato il raggiungimento del limite della dimensione del vettore
  - L'inizializzazione del vettore è fatta dal main

POTENZIALI PROBLEMI

- Se andassimo ad espandere il nostro programma, potremmo inserire funzioni che vanno a modificare senza controllo il vettore

- Quante più funzioni vanno a toccare (leggere / scrivere) le stesse variabili, tanto più faticoso può essere il debugging del codice in caso di rilevazione di un fallimento

TIPO DI DATO ASTRATTO

- Il concetto di Tipo di Dato Astratto (**ADT**) viene introdotto già nella programmazione procedurale

- Caratteristiche fondamentali:
  - Astrazione
  - Modularità
  - Incapsulamento
  - Information Hiding

ASTRAZIONE

- Si vogliono raggruppare alcune entità che:
  - Corrispondano a concetti astratti oppure ad elementi del mondo reale
  - Abbiano delle proprietà loro intrinseche
  - Sui quali ha senso definire delle operazioni ben precise

- Il processo di *raggruppare* tutti questi elementi sotto un unico concetto si chiama **astrazione**

ESEMPI DI ASTRAZIONE

- Rispetto al codice precedente, può essere astratto il concetto di **elenco di voti**

- L'**elenco di voti** ha:
  - Un concetto astratto di riferimento
    - Corrispondente ad un elenco di voti
  - Delle proprietà specifiche
    - La dimensione (20 elementi), il tipo degli elementi, la memorizzazione sotto forma di array, i valori degli elementi
  - Un insieme di operazioni applicabili su di esso
    - L'inizializzazione
    - La ricerca di un valore
    - L'ordinamento

ESEMPI DI ASTRAZIONE

- Per i sistemi e servizi dell'università una astrazione utile può essere quella di **studente Universitario**

- Il concetto di riferimento è chiaramente la persona che si iscrive per studiare all'università
- Tra le proprietà specifiche ci possono essere:
  - I dati anagrafici, il corso di laurea, il piano di studi, gli esami sostenuti, le tasse da pagare, le borse di studio, i sondaggi effettuati, …
- Tra le operazioni effettuabili
  - L'iscrizione, la rinuncia, la registrazione di un esame, il pagamento delle tasse, la partecipazione a un concorso, …

MODULARITA'

- Negli algoritmi, il principio *divide-et-impera* consente di ricondurre la soluzione di un problema a quella di sottoproblemi più piccoli ad esso collegato

- Nell'Object Oriented è opportuno modellare un concetto complesso tramite concetti più semplici
  - Ma rispettando il principio di *astrazione*

ESEMPIO DI MODULARITA'

- Nel caso degli **studenti universitari** conviene distinguere ad esempio tra:
  - Gli studenti
  - I corsi universitari
  - …

  - I dati anagrafici sono informazioni peculiari di uno studente
  - Il nome di un corso Universitario, il suo programma, il nome del docente sono invece proprietà del corso, indipendentemente dall'esistenza degli studenti

PARADIGMA OBJECT ORIENTED

- Il paradigma Object Oriented si basa sulla possibilità di risolvere un problema cominciando con la *modellazione* degli elementi del problema sotto forma di **classi** e **oggetti**

OGGETTO (COME ASTRAZIONE)

- Un **oggetto** (o **istanza**) nel paradigma object oriented rappresenta un elemento ben distinto e distinguibile nell'ambito della descrizione di un problema o della sua soluzione

- Un oggetto ha un insieme di informazioni associate (detti **attributi** o **proprietà)**
- Un oggetti ha un suo comportamenti (*behaviour*) che definisce come agisce o reagisce a seguito di sollecitazioni

CLASSE (COME ASTRAZIONE)

- Una **Classe** :
  - Rappresenta l'astrazione di un concetto del mondo reale oppure di un concetto che fa parte del problema da risolvere
  - Può essere *istanziata*, cioè può essere *costruito* un *oggetto* (*istanza)* della classe
  - Definendo la *classe* si definiscono anche:
    - Gli *attributi,* cioè le informazioni che dovranno essere valorizzate per ognuno degli oggetti
    - Le *operazioni* (o *metodi, o responsabilità*), cioè le operazioni che possono essere eseguite sugli oggetti della classe in modo che l'oggetto segue il suo *comportamento* (*behaviour*)
    - Diversi oggetti possono comunicare tra loro leggendo/scrivendo valori agli attributi oppure chiamando i metodi di altri oggetti
  - Una classe nasce per poter essere **riusabile** in altri problemi nei quali gli stessi oggetti e comportamenti sono richiesti

ESEMPI DI CLASSI

- Studente
  - Attributi: nome, cognome, numero di matricola, …
  - Operazioni : iscrizione, rinuncia, laurea, …
- Esame
  - Attributi: Nome della materia, nome del docente, lista degli appelli …
  - Operazioni: registra, aggiungi appello, ..
- Corso di Laurea
  - Attributi: Nome, universita', …
  - Operazioni: aggiungi nuovo corso, conta numero di iscritti, …
- Esame sostenuto
  - Attributi: voto, *Esame*, *Docente*
  - Operazioni: aggiungi esame, annulla esame, fissa voto
- …

ESEMPI DI OGGETTI

- Porfirio : Studente
  - Nome = ‘Porfirio’, cognome = ‘Tramontana’
- Object Orientation : Esame
  - Nome della materia = ‘Object Orientation’, nome del docente = ‘Di Martino’ , canale = ‘A-G’
- Informatica : Corso di Laurea
  - Nome = ‘Informatica’, universita’ = ‘Federico II’
- Esame#1 : Esame sostenuto
  - Voto = 30, *Esame* = *Object Orientation,* Studente = Porfirio
  - …

INCAPSULAMENTO E INFORMATION HIDING

- Un principio fondamentale è quello di limitare l'accesso / visibilità alle informazioni al minimo indispensabile

- Perchè rendere le informazioni invisibili di default?
  - All'aumentare del numero di informazioni visibili
    - Aumenta la complessità della progettazione
    - Aumenta il rischio di commettere errori di programmazione
    - E' più faticoso scoprire le cause dei fallimenti
    - E' più difficile correggere i difetti
    - E' più difficile manutenere i sistemi (ad esempio evolverli, fonderli, integrarli)

- L'insieme di informazioni (dati, operazioni) che un componente mostra all'esterno è detto **interfaccia**

ESEMPI DI INTERFACCIA

- Il Sistema segrepass mostra agli studenti i propri dati anagrafici ma
  - non mostra il proprio identificativo nel database
  - non mostra ad uno studente l'elenco degli altri studenti
  - fornisce ad uno studente la possibilità di leggere gli esami sostenuti ma sicuramente non quella di aggiungere un esame o modificare il voto di un esame sostenuto!
  - non mostra ad un docente la password di uno studente!

- Il Sistema Segrepass ha *molteplici* interfacce, per diverse tipologie di utenti
  - In ognuna di queste viene specificato l'elenco di dati e operazioni accessibili e le modalità di accesso consentite (lettura / scrittura / …)

EREDITARIETA'

- Quando tra una classe (superclasse o classe padre) e un'altra classe (subclasse o classe figlio) c'è una relazione di ereditarietà si intende che:
  - Dal punto di vista concettuale la classe padre rappresenta una generalizzazione della classe figlio
  - Dal punto di vista tecnico, che tutti gli attributi e i metodi della classe padre sono applicabili (ereditati) anche dalla classe figlio
    - Se la classe figlio può avere una propria versione specifica di un metodo rinunciando ad ereditarlo dal padre: in questo caso si parla di *overriding* del metodo

- Una fondamentale relazione tra le classi che consente il riuso parziale di classi complesse

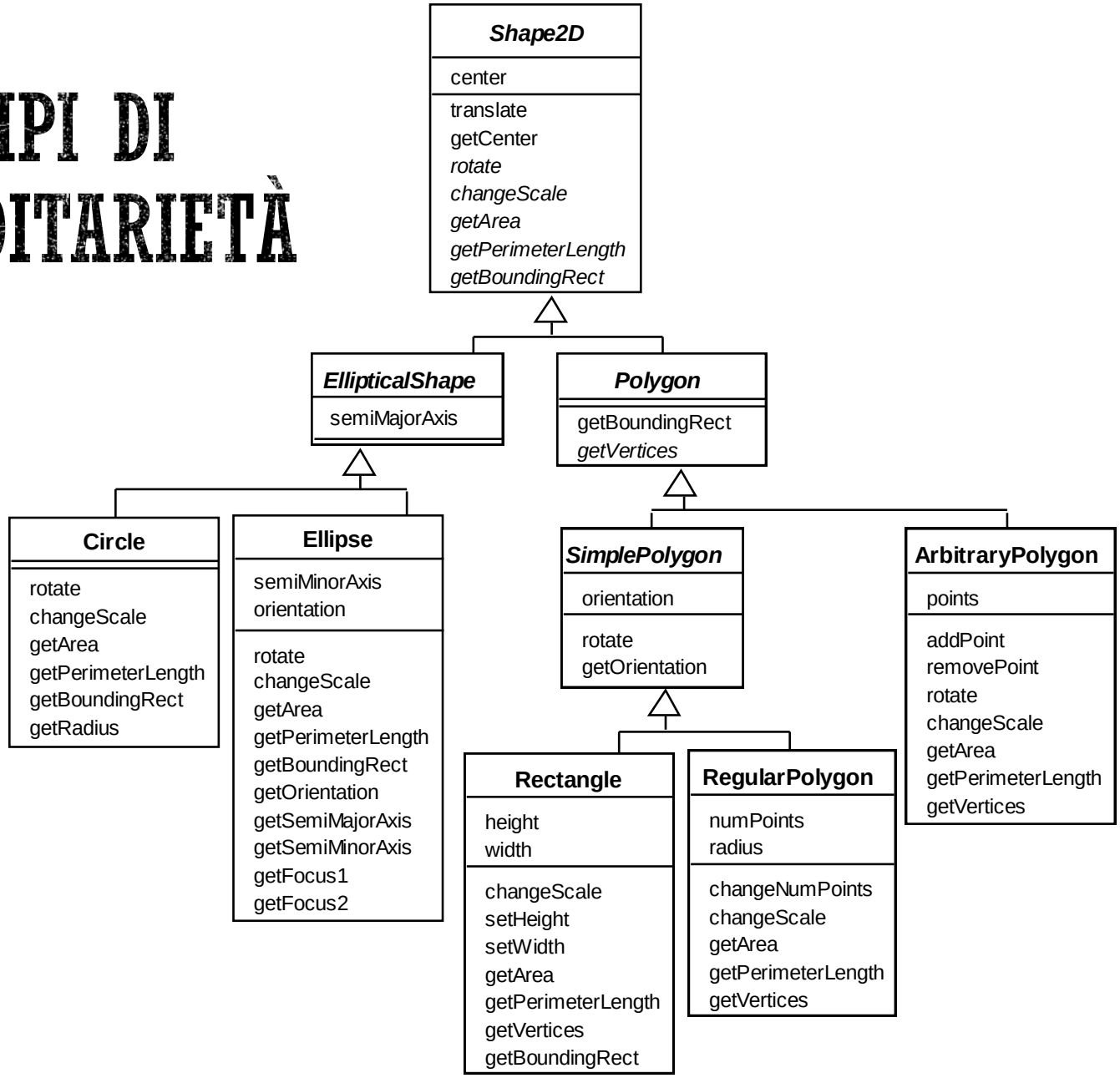
ESEMPI DI EREDITARIETA'

- Consideriamo un sistema analogo al Sistema Teams
  - Semplificato a fini didattici
- Esso deve sicuramente tenere conto dell'esistenza di *Utenti*, tra i quali in particolare *Docenti* e *Studenti*
- Che relazioni ci sono tra queste classi?
  - Analizziamo alcune caratteristiche e operazioni
    - Tutti (utenti, student, docent) devono accedere tramite l'operazione di *sign in* ed hanno propri login e password
    - I docenti hanno il diritto di creare un *canale*
    - Gli studenti hanno il diritto di iscriversi ad un canale

ESEMPI DI EREDITARIETA'

- *Utente* rappresenta una generalizzazione (padre) di *Studente* e *Docente (figli)*
  - gli attribute login e password e l'operazione di *sign in* viene definita per la classe utente ed ereditata (è applicabile anche su) Studente e Docente
- Per Docente è inoltre definita l'operazione *crea canale*
- Per *Studente* è definita l'operazione di *iscrizione al canale*
- Docente e Studente non hanno un legame diretto (hanno la stessa classe padre: sono in qualche modo fratelli – *sibling*)
- Grazie all'ereditarietà possiamo limitare la visibilità/accesso alle informazioni

ESEMPI DI EREDITARIETÀ



CONTENIMENTO VS EREDITARIETA'

- Un errore che si commette spesso consiste nel confondere le gerarchie di contenimento o aggregazione (*tutto-parti)* con quelle di ereditarietà (*padre-figlio o is-a)* e con la relazione di istanziazione (*classe-oggetto*)

- Tutto-parti: una città è *parte* di una nazione
  - Conseguenza: La popolazione totale di una nazione è la somma delle popolazioni di tutte le sue città
- Padre-figlio : una repubblica è un (*is a*) tipo di nazione
  - Conseguenza: Il concetto di popolazione viene stabilito per la classe nazione ed è valido anche per una repubblica (o per una monarchia)
- Classe-oggetto : l'Italia è un oggetto della classe repubblica, Napoli è un oggetto della classe città
  - Conseguenza: Siccome repubblica eredita da nazione, l'attributo popolazione assume un valore anche per Italia. L'oggetto Napoli è una parte dell'oggetto Italia

RETROSPETTIVA: PROCEDURALE

- Quali di questi principi sono parzialmente seguiti già dalla programmazione procedurale?

- Astrazione
  - Le struct raggruppano dati, anche di tipo eterogeneo, in un'unica variabile
    - Ma non è possibile abbinare le operazioni alle struct
  - Le funzioni possono rappresentare *operazioni elementari* da eseguire (*astrazione sul controllo*)
- Modularità
  - Le funzioni rappresentano i *moduli* di un algoritmo complesso
  - I file possono contenere funzioni che sono concettualmente collegate (*librerie*)

RETROSPETTIVA

- Quali di questi principi sono parzialmente seguiti già dalla programmazione procedurale?

- Incapsulamento ed information hiding
  - In C++ si raccomanda di dare alle variabili *scope* più piccoli possibili
    - Una funzione non ha visibilità delle variabili dichiarate in un'altra funzione
    - Un blocco non ha visibilità delle variabili dichiarate in un altro blocco (a meno che non sia un blocco che lo contiene)

RETROSPETTIVA:
TIPI DI DATO ASTRATTO IN C++ (ADT)

NomeADT.h

```
// Interfaccia del modulo ADT

//Eventuali definizioni di tipo
//Prototipi delle operazioni
//previste sul tipo
```

NomeADT.cpp

```
// Implementazione del modulo ADT

#include "NomeADT.h"
//Implementazione delle operazioni
// previste sul tipo
```

Utilizzatore.cpp

```
// Modulo utilizzatore del modulo
// ADT
#include "NomeADT.h"
```