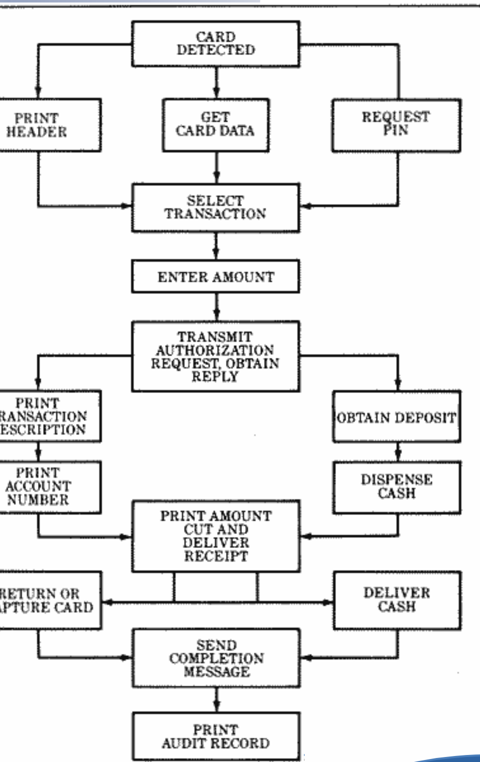
Programmazione, astrazioni e Linguaggi

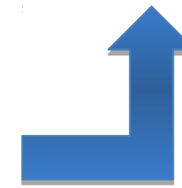
Cosa vuole dire programmare

- Comprendere i requistiti del problema da risolvere
- Progettare l’architettura del software
- Progettare le strutture dati e gli algoritmi
- Programmare (in senso stretto)
- Testing e debugging
- Profilare e ottimizzare il software
- Adeguarsi agli standard
- ...

Programmare e’ costruire un modello computazionale



al transaction sequence.



Paradigmi di programmazione

- **Paradigma**: insieme di strumenti concettuali forniti da un linguaggio per la scrittura di un programma, definendo il modo in cui il programmatore concepisce e percepisce il programma stesso
- Esempio di problema:
  - determinare l’area di una figura geometrica, ad esempio un rettangolo
- La costruzione del programma, può essere impostata secondo differenti paradigmi di programmazione

Paradigma procedurale

Esempio in C

```
#include<stdio.h>
#include<math.h>

main()
{
    double base, height, area;
    printf("Enter the sides of rectangle\n");
    scanf("%lf%lf",&base,&height);
    area = base * height;
    printf("Area of rectangle= %lf\n", area);
    return 0;
}
```

Paradigma funzionale

Esempio in OCaml

```
let area b h =  b *. h;;
  val area : float -> float -> float = <fun>
```

Passo di Astrazione

- Generalizziamo il problema:
  - Determinare il valore dell'area di varie figure geometriche
    - Square
    - Rectangle
    - Circle

Astrazioni sui dati

```
double size() {
  double total = 0;
  for (Shape shape : shapes) {
      switch (shape.kind()) {
      case SQUARE:
        Square square = (Square) shape;
        total += square.width * square.width;
        break;
     case RECTANGLE:
        Rectangle rectangle = (Rectangle) shape;
        total += rectangle.width * rectangle.height;
        break;
     case CIRCLE:
       :
      }
  }
  return total;
}
```

Paradigma object-oriented

```
double size() {
  double total = 0;
  for (Shape shape :shapes) {
      total += shape.size();
  }
  return total;
}
```

size() itera sulla lista sommando le superfici

```
public class Square extends Shape {...
  public double size() {
   return width*width;
  }
 }
```

Distribuiamo la computazione tra le strutture (oggetti).
Ogni forma geometrica è responsabile del calcolo della propria area.

SHAPE
AREA

RETTANGOLO
BASE
ALTEZZA
AREA = BASE * ALTEZZA

CERCHIO
RAGGIO
AREA = RAGGIO * RAGGIO * PIGRECO

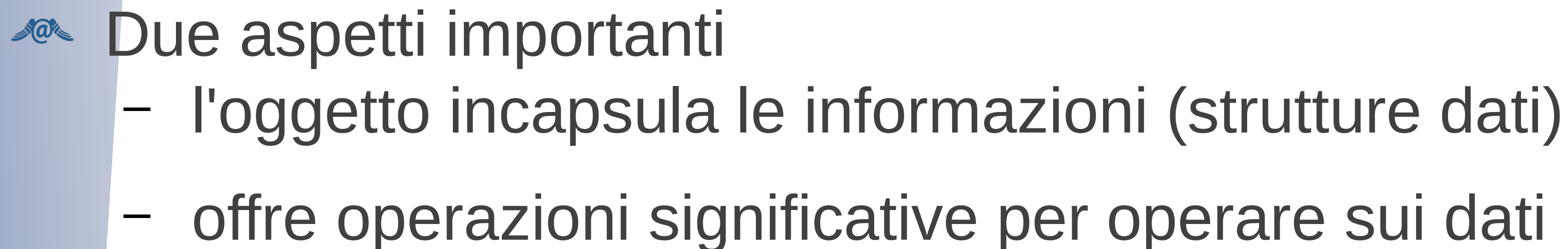
…

Shape: gestisce del contenuto informativo (dati)
e fornisce un servizio (il calcolo dell’area), demandandolo agli oggetti delle sottoclassi concrete

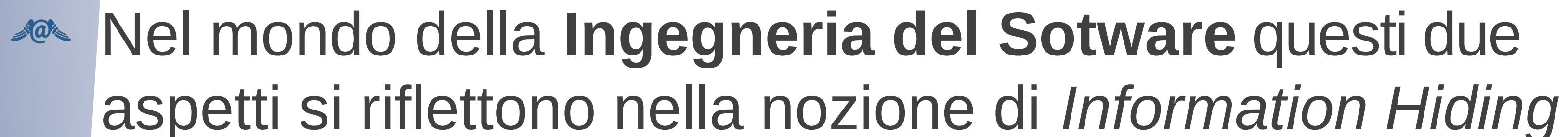
Object Oriented programming

- http://en.wikipedia.org/wiki/Object-oriented_programming

  “Object-oriented programming (OOP) is a programming paradigm that represents concepts as “objects” that have data fields (***attributes*** that describe the object) and associated procedures known as ***methods***.”



- Due aspetti importanti
  - l'oggetto incapsula le informazioni (strutture dati)
  - offre operazioni significative per operare sui dati



- Nel mondo della **Ingegneria del Sotware** questi due aspetti si riflettono nella nozione di *Information Hiding*

Information Hiding (Wikipedia)

- **Information hiding** is the principle of segregation of the design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed.
- The protection involves providing a stable interface which protects the remainder of the program from the implementation (the details that are most likely to change).

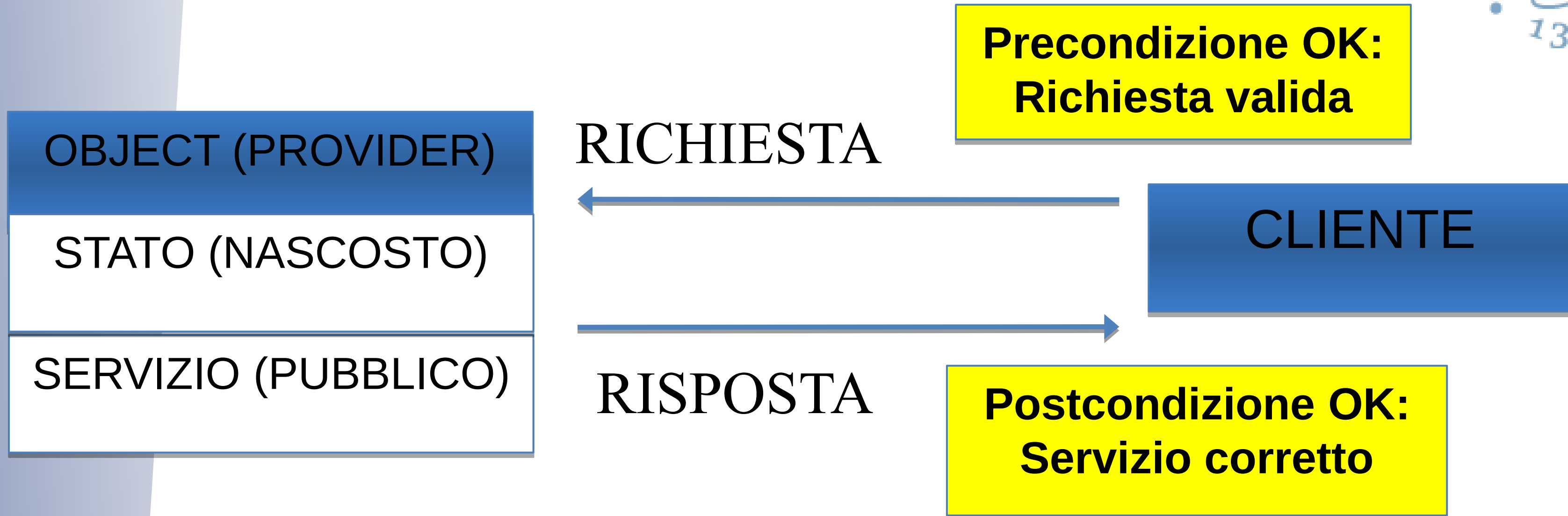
Perché l'Information Hiding è importante?

- Consente di esporre solamente le informazioni necessarie per operare
- Si dichiara “cosa si fa” non “come si fa” (*separation of concerns*)
- Facilita la decomposizione di un sistema in parti
- Garantisce la protezione dei clienti dalla modifiche di implementazione
- Consente la definizione di contratti di uso
- Facilita manutenzione e evoluzione del software

Programming by contract

- Metodologia ispirata alla metafora di un contratto legale tra fornitore e cliente.
- Il fornitore è l'oggetto, che offre procedure (metodi) per operare sui suoi dati. Il cliente è un programma che invoca i metodi.
- Un contratto stabilisce tipicamente:
  - **precondizione/postcondizione** per ogni metodo,
  - un **invariante** di rappresentazione per la classe.
- Obblighi reciproci:
  - Il cliente deve garantire che la sua invocazione/richiesta soddisfi la **precondizione**
  - Il metodo deve garantire che se un'invocazione soddisfa la **precondizione**, allora il risultato soddisfa la **postcondizione**
- Ogni metodo deve garantire che se **l'invariante** di rappresentazione era soddisfatto al momento della chiamata, lo sarà di nuovo quando viene restituito il risultato.

Programming by contract



E se la precondizione non è soddisfatta? Il fornitore (l'oggetto) non ha alcun obbligo. Situazione analoga alla **Logica di Hoare**.
Tipicamente viene lanciata un'eccezione
Questo facilita la verifica di correttezza dei metodi.

Java. Perché?

- Modello a oggetti semplice e maneggevole
- Ogni entità è un oggetto
- Ereditarietà singola
- Garbage collection
- Caricamento dinamico delle classi
- Librerie (viviamo in un mondo di API)
- Controllo di tipi statico e dinamico
- Meccanismi per la sicurezza
- Morale: poche novità, ma ragionevolmente pulito, semplice e fruibile

ASTRAZIONE

Meccanismi di astrazione

- Sono alla base di meccanismi di decomposizione in moduli
  - Necessari per programmi di grandi dimensioni
- Consistono nel cambiare livello di dettaglio
  - Ignorando particolari considerati irrilevanti
- Meccanismi di astrazione legati alla programmazione:
  - I **linguaggi ad alto livello** forniscono significative astrazioni rispetto a linguaggi assembler/macchina
  - Enorme semplificazione per il programmatore
  - Si usano direttamente i costrutti del linguaggio ad alto livello invece che sequenze di istruzioni in linguaggio macchina “equivalenti” (si pensi a un assegnamento)

I linguaggi non bastano

```
// ricerca all'insù
   found = false;
   for (int i = 0; i < a.length; i++)
      if (a[i] == e) {
         z = i; found = true;}
```

```
// ricerca all'ingiù
   found = false;
   for (int i = a.length - 1; i >= 0; i--)
      if (a[i] == e) {
         z = i; found = true;}
```

- Sono diversi e possono dare risultati diversi
- Ma potrebbero essere stati scritti pei risolvere lo stesso problema:
  - verificare se l'elemento è presente nell'array e restituire una posizione in cui è contenuto

Aggiungere astrazioni nel linguaggio?

Il linguaggio potrebbe essere esteso con potenti operazioni su array, come **isIn** e **indexOf**:

```
// ricerca indipendente dall'ordine
   found = a.isIn(e);
   if (found) z = a.indexOf(e);
```

- Ma così l'astrazione è scelta dal progettista del linguag
  - Quali, quante, e quanto complicato diventa il linguaggio?
- Meglio progettare linguaggi **dotati di meccanismi che permettano di definire le astrazioni che servono**

Il più comune tipo di astrazion

- L'astrazione procedurale è presente in tutti i linguaggi di programmazione
  - Dichiarazione di procedure, chiamata di procedure
- La separazione tra "dichiarazione" e "chiamata" rende disponibili i due meccanismi fondamentali di astrazione:
  - L'astrazione attraverso parametrizzazione
    - si astrae dall'identità di alcuni dati, rimpiazzandoli con parametri
    - si generalizza una procedura per poterlo usare in situazioni diverse
    - Esempio: Shape definito in precedenza
  - L'astrazione attraverso specifica
    - si astrae dai dettagli dell'implementazione della procedura, per limitarsi a considerare il comportamento che interessa a chi usa la procedura (ciò che fa, non come lo fa)
    - si rende ogni procedura indipendente dalle implementazioni delle procedure che usa

Astrazione via parametrizzazione

- L'introduzione dei parametri permette di descrivere un insieme (anche infinito) di computazioni diverse con un singolo programma che le astrae tutte
- L'espressione seguente descrive una particolare computazione:

```
x * x + y * y
```

- La definizione

```
let fun x y = (x * x + y * y)
```

  descrive tutte le computazioni che si possono ottenere chiamando la funzione, cioè applicando la funzione ad una opportuna coppia di valori
- Per esempio,

```
fun 5 3
```

- Ha la stessa semantica di `(5 * 5 + 3 * 3)`

Astrazione via specifica

- La nozione di procedura si presta a meccanismi di astrazione più potenti della **parametrizzazione**
- Possiamo astrarre dalla specifica computazione descritta nel corpo della procedura, associando ad ogni procedura una **specifica**
  - che descrive la *semantica intesa* della procedura
- e derivare la semantica della chiamata dalla specifica invece che dal corpo della procedura
  - non è di solito supportata dal linguaggio di programmazione
  - se non in parte tramite le specifiche di tipo, come avviene nei linguaggi funzionali della famiglia di ML

- Si realizza con opportune annotazioni
  - Esempio: Aspect Oriented Programming (AOP)
  - Esempio: JML (Java Modeling Language)

```
float sqrt (float coef) {
   // REQUIRES: coef > 0
   // EFFECTS: ritorna una approssimazione
   // della radice quadrata di coef
   float ans = coef / 2.0; int i = 1;
   while (i < 7) {
   ans = ans-((ans*ans-coef)/(2.0*ans));
   i = i+1;    }
   return ans;  }
```

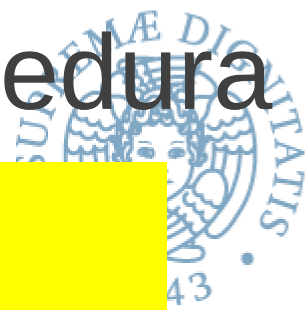
precondizione (asserzione **requires**)

- deve essere verificata quando si chiama la procedura

postcondizione (asserzione **effects**)

- tutto ciò che possiamo assumere valere quando la chiamata di procedura termina, se al momento della chiamata era verificata la precondizione

Il punto di vista di chi usa la procedura



```
float sqrt (float coef) {
   // REQUIRES: coef > 0
   // EFFECTS: ritorna una approssimazione
   // della radice quadrata di coef
   …   }
```

- Gli utenti della procedura non si devono preoccupare di capire cosa fa la procedura, astraendo le computazioni descritte dal corpo
  - cosa che può essere molto complessa
- Gli utenti della procedura non possono osservare le computazioni descritte dal corpo e dedurre da questo proprietà diverse da quelle specificate dalle asserzioni
  - astraendo dal corpo (implementazione), si "dimentica" informazione evidentemente considerata non rilevante

Tipi di astrazione

- parametrizzazione e specifica permettono di definire va tipi di astrazione
  - **astrazione procedurale**
    - ✓ si aggiungono nuove operazioni
  - **astrazione di dati**
    - ✓ si aggiungono nuovi tipi di dato
  - **iterazione astratta**
    - ✓ permette di iterare su elementi di una collezione, senza sapere come questi vengono ottenuti
  - **gerarchie di tipo**
    - ✓ permette di astrarre da specifici tipi di dato a famiglie di tipi correlati

Astrazione procedurale

- fornita da tutti i linguaggi ad alto livello
- aggiunge nuove operazioni a quelle fornite come primitive dal linguaggio di programmazione
  - Esempi che avete visto in C, e OCAML
- La specifica descrive le proprietà della nuova operazione

Astrazione sui dati

- Fornita da tutti i linguaggi ad alto livello moderni
- aggiunge nuovi tipi di dato e relative operazioni
  - l'utente non deve interessarsi dell'implementazione, ma fare solo riferimento alle proprietà presenti nella specifica
  - le operazioni sono astrazioni definite da asserzioni logiche (ricordate le specifiche di LPP?)
- la specifica descrive le relazioni fra le varie operazioni
  - per questo, è cosa diversa da un insieme di astrazioni procedurali

Iterazione astratta

- Permette di iterare su elementi di una collezione senza sapere come questi vengono ottenuti
- Evita di dire cose troppo dettagliate sul flusso di controllo all’interno di un ciclo
  - per esempio, potremmo iterare su tutti gli elementi di un MultiInsieme senza imporre nessun vincolo sull’ordine con cui vengono elaborati
- Astrae (nasconde) il flusso di controllo nei cicli

Gerarchie di tipo

- Fornite dai linguaggi ad alto livello moderni
  - per esempio, Ocaml, F#, Java, C#
- Permettono di astrarre gruppi di astrazioni di dati (tipi) a famiglie di tipi
- i tipi di una famiglia condividono alcune operazioni
  - definite nel supertype, di cui tutti i tipi della famiglia sono subtypes
- una famiglia di tipi astrae i dettagli che rendono diversi tra loro i vari tipi della famiglia

Astrazione e programmazione orientata ad oggetti

- il tipo di astrazione più importante per guidare la decomposizione è l'astrazione sui dati
  - gli iteratori astratti e le gerarchie di tipo sono comunque basati su tipi di dati astratti
- l'astrazione sui dati è il meccanismo fondamentale della programmazione orientata ad oggetti
  - anche se esistono altre tecniche per realizzare tipi di dato astratti
    - ✓ per esempio, all'interno del paradigma di programmazione funzionale