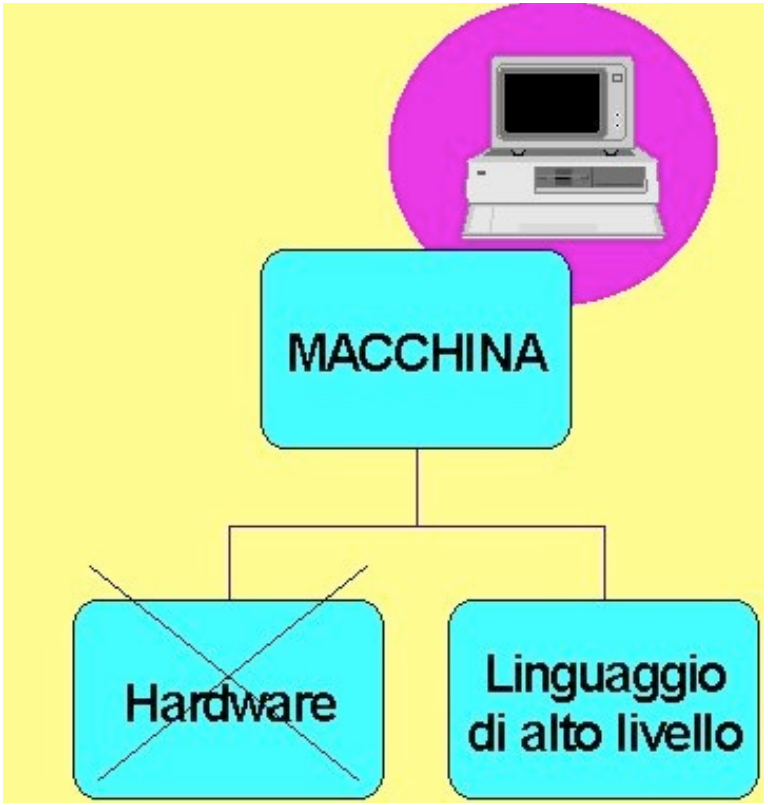
Introduzione all'OOP

- Introdurre l'evoluzione dei linguaggi di programmazione e la loro classificazione
- Introdurre l'Object-Oriented Programming (OOP) e i suoi principi (astrazione, incapsulamento, ereditarietà)
- Dal C a Java … un primo esempio (...anzi due)

LINGUAGGI DI ALTO LIVELLO

Si basano su una *macchina virtuale* le cui "mosse" non sono quelle della macchina hardware

Macchina di Von Neumann



LINGUAGGI DI PROGRAMMAZIONE

00111101000100100001000

confronto del contenuto dell'accumulatore AX con la costante 812

linguaggio macchina

insieme delle istruzioni per un certo processore (ad es. 8086)

3D1208

in esadecimale

Quali astrazioni?

- **Sui dati**: da indirizzi fisici a ***nomi simbolici*** per le operazioni e per le variabili (Assembler)

- Es. carica il numero 8 nel primo registro libero della CPU: 0011 1000

- ***Linguaggio Assembler***: LOAD 8

- C'è bisogno di un convertitore (assemblatore) da codice assembler a codice macchina

Quali astrazioni?

- **Sui dati**: da indirizzi fisici a ***nomi simbolici*** per le variabili (Assembler)
- **Sul controllo** (***programmazione strutturata***, Pascal, C):
  - sequenza (;),
  - blocco,
  - **`if else`**, **`while do`**, etc.

```
{ int p=1;
  for (i=1 ; i <= n ; ++i)   p = p*x;
  . . .
 }
```

Astrazioni funzionali

- **Sulle operazioni (*funzioni*):**

```
long potenza(int x, int n) /* par. formali */
{ int i;
  long p = 1;                /* var. locali */
  for (i=1 ; i <= n ; ++i)  p *= x;
  return p;
}
```

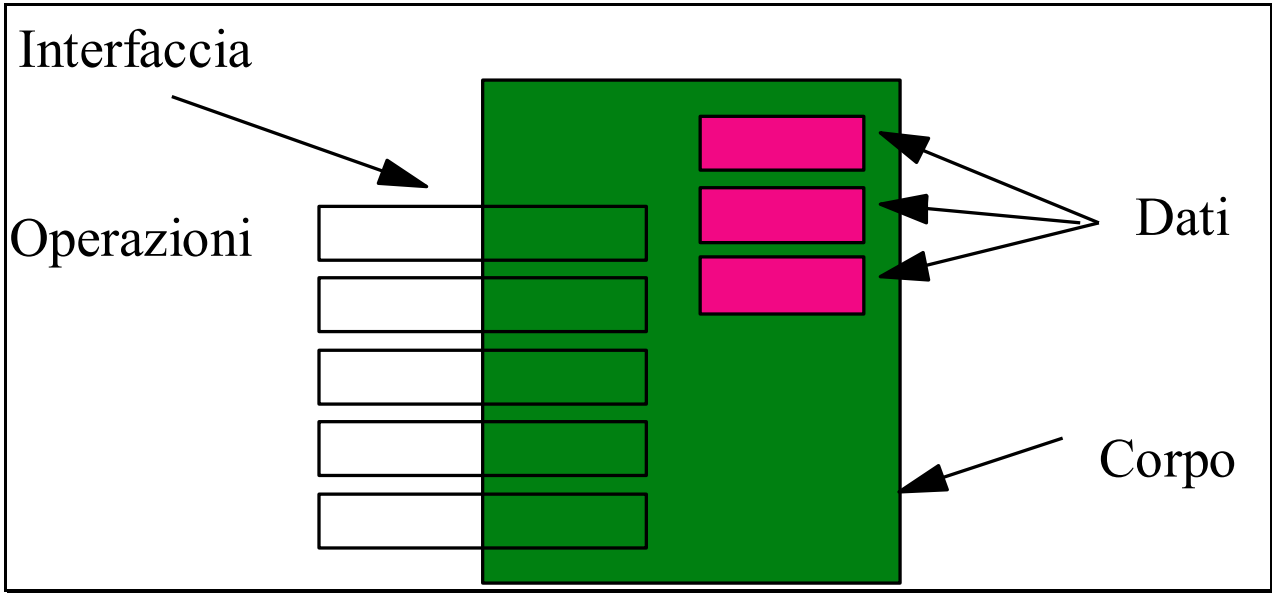
```
long Y;
Y=potenza(X,10);
```

Encapsulation (incapsulamento)

- Coniugare l'astrazione sui dati e sulle operazioni (astrazioni di dato, tipi di dato astratto, classi e oggetti)
  - Incapsulamento (dati nascosti, e operazioni visibili)
  - Protezione
  - Coesione
  - Riuso
  - … (Ereditarietà)

Astrazioni di dato

- Il componente ha dati locali (nascosti) e rende visibili all'esterno i prototipi delle operazioni invocabili (procedure e funzioni) su questi dati locali, ma non gli identificatori dei dati. Attraverso una di queste operazioni si può assegnare un valore iniziale ai dati locali nascosti.



Esempio in C: modulo `contatore`

1. *file header* `contatore.h`

```
void reset(void);
void inc(void);
void stampa(void);
```

Definisce cosa si può fare con il contatore

2. *file di implementazione* `contatore.c`

```
#include "contatore.h"
static int c;
void reset(void){ c=0; }
void inc(void){ c++; }
void stampa(void) { printf("%d", c); }
```

Incapsula il contatore **c** (IL dato) e specifica il codice delle funzioni con cui si agisce sul contatore

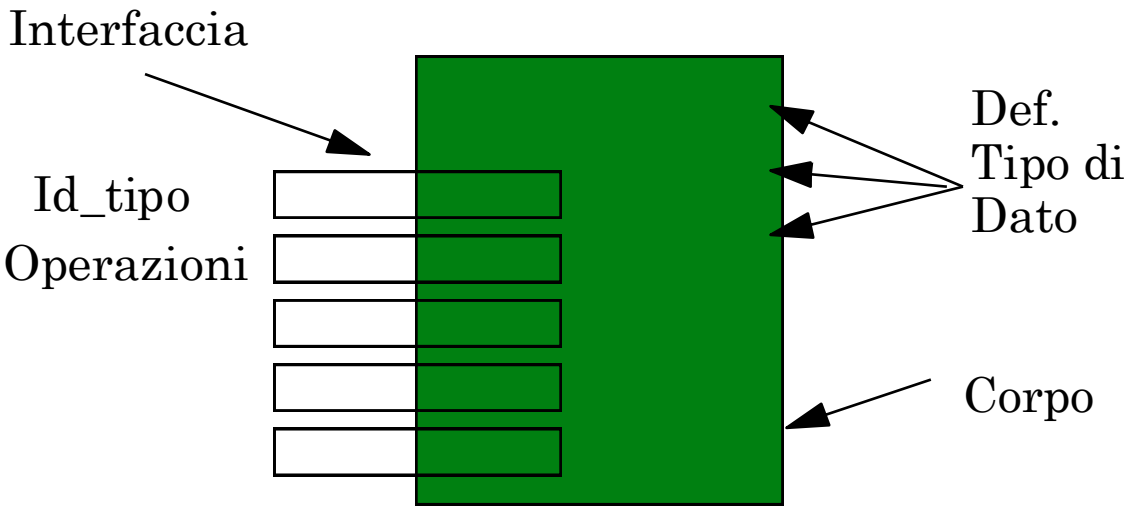
OOP e Java – astrazione di dato Contatore

```
public class Contatore{
  private static int x;
  public static void reset() { x = 0; }
  public static void inc()   { x++; }
  public static int getValue() { return x;}
}
```

```
int n;
Contatore.reset();
Contatore.inc();
n = Contatore.getValue();
```

Tipo di dato astratto (Abstract Data Type – ADT)

- Il componente esporta un ***identificatore di tipo T*** ed i prototipi delle operazioni eseguibili su dati dichiarati di questo tipo. I "clienti" del componente dichiarano e controllano quindi il tempo di vita delle variabili di tipo T.



Esempio in C: ADT `counter`

1. *file header* `counter.h`

```
typedef int counter;
void reset(counter*);
void inc(counter*);
```

Definisce in astratto che cos'è un counter e che cosa si può fare con esso

2. *file di implementazione* `counter.c`

```
#include "counter.h"
void reset(counter *c){ *c=0; }
void inc(counter* c){ (*c)++; }
```

Specifica come funziona (quale è l'implementazione) di counter

ADT `counter`: un cliente

Per usare un **`counter`** :

```
#include "counter.h"
int main(){
   counter c1, c2;
   reset(&c1); reset(&c2);
   inc(&c1); inc(&c2); inc(&c2);
c1++;
}
```

Programmazione Orientata agli Oggetti (OOP) … ovvero cose che in C non possiamo ottenere

- Astrazione: si lavora considerando il “comportamento ai morsetti” e prescindendo da rappresentazioni interne (di stato e operazioni) - separazione fra interfaccia e implementazione
- Incapsulamento: insieme di meccanismi che consentono di proteggere lo stato di un oggetto e in generale di nascondere gli aspetti che non si vogliono rendere pubblici
  - **Linguaggi object-based**

OOP e Java – l'ADT Counter

```
public class Counter {
  private int x;
  public void reset() { x = 0; }
  public void inc()   { x++; }
  public int getValue() { return x;}
}
```

```
Counter Y;
Y = new Counter();
Y.reset();
Y.inc();
```

OOP e Java

```
public class Counter {
  private int x;
  public void reset() { x = 0; }
  public void inc()   { x++; }
  public int getValue() { return x;}
}
```

```
Counter Y;
Y = new Counter();
Y.reset();
Y.inc();
```

Programmazione Orientata agli Oggetti (OOP)

- Astrazione: si lavora considerando il "comportamento ai morsetti" e prescindendo da rappresentazioni interne (di stato e operazioni) - separazione fra interfaccia e implementazione
- Incapsulamento: insieme di meccanismi che consentono di proteggere lo stato di un oggetto e in generale di nascondere gli aspetti che non si vogliono rendere pubblici
  - **Linguaggi object-based**
- Ereditarietà: consente di creare un componente (classe) che riusa metodi e attributi di un componente (classe) già esistente
  - **Linguaggi object-oriented**

```
public class BiCounter extends Counter {
  public void dec()    { x--; }
}
```

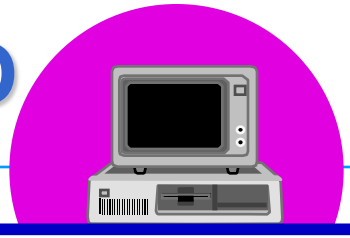
Ereditarietà

```
public class Counter {
  private int x;
  public void reset() { x = 0; }
  public void inc()   { x++; }
  public int getValue() { return x;}  }

public class BiCounter extends Counter {
  public void dec()   { x--; } }

BiCounter Y;  Y = new BiCounter();
Y.reset(); Y.inc(); Y.dec();
```

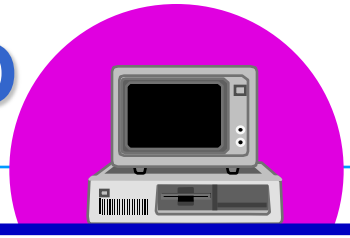
LINGUAGGI DI ALTO LIVELLO



Barriera di astrazione

Fortran
Cobol
Basic
Pascal
Python
C
Modula-2
Algol
Ada
Lisp
Smalltalk
Simula67
Scheme
Java
C++
Prolog
ML

LINGUAGGI DI ALTO LIVELLO



Barriera di astrazione

IMPERATIVI

Fortran
Cobol
Basic
Pascal
Python
C
Modula-2
Algol
Ada

Smalltalk
Simula67
Java
C++

A OGGETTI

Prolog

DICHIARATIVI

Lisp
Scheme
ML

FUNZIONALI

Introduzione all'OOP – prime conclusioni

- Componenti che incapsulano dati e operazioni
- Dal C a Java … due esempi:
  - Modulo (astrazione di dato)
  - Tipo di dato astratto
- Java è un linguaggio OOP con un costrutto (class) che facilita la programmazione per componenti