Introduzione alla programmazione *Object Oriented*

Luca Lista

Concetti base del software “OO”

- Classi e oggetti
- Incapsulamento
- Relazione di ereditarietà
- Polimorfismo

Cos'è un *Oggetto*?

- Definizione da vocabolario:

> *"Un oggetto è un'entità che si possa immaginare dotata di determinate caratteristiche e funzionalità".*

- Il confinamento di *informazioni* e *funzionalità* in oggetti permette livelli maggiori di astrazione e semplifica la gestione di sistemi complessi.

Gli oggetti come modello naturale

- Gli oggetti nel software possono rappresentare entità del mondo reale
  - Il primo uso di linguaggi a oggetti è stato fatto per effettuare simulazioni

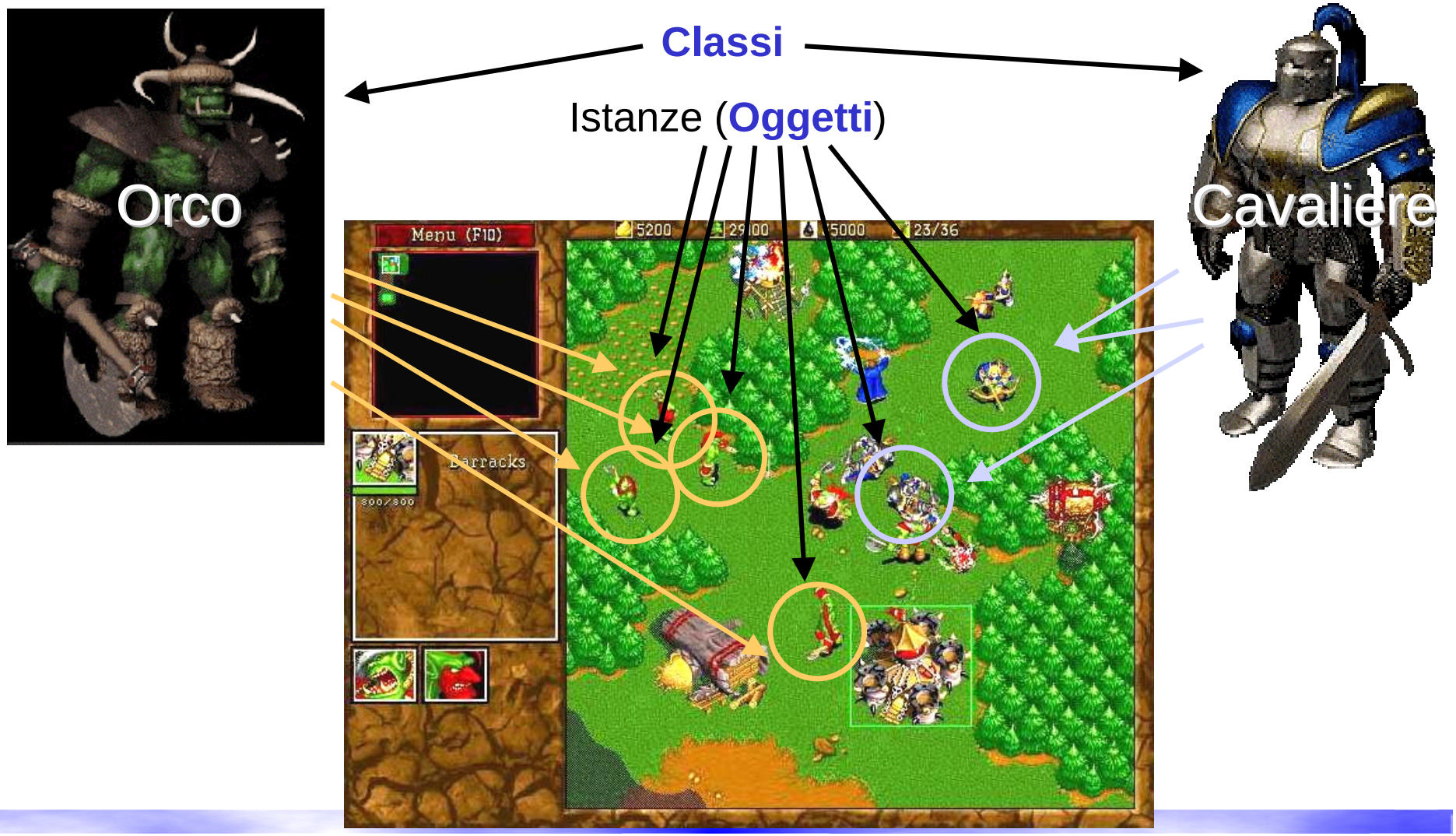
Interazioni tra diversi componenti

=

scambio di messaggi

Simulazione: un Videogioco

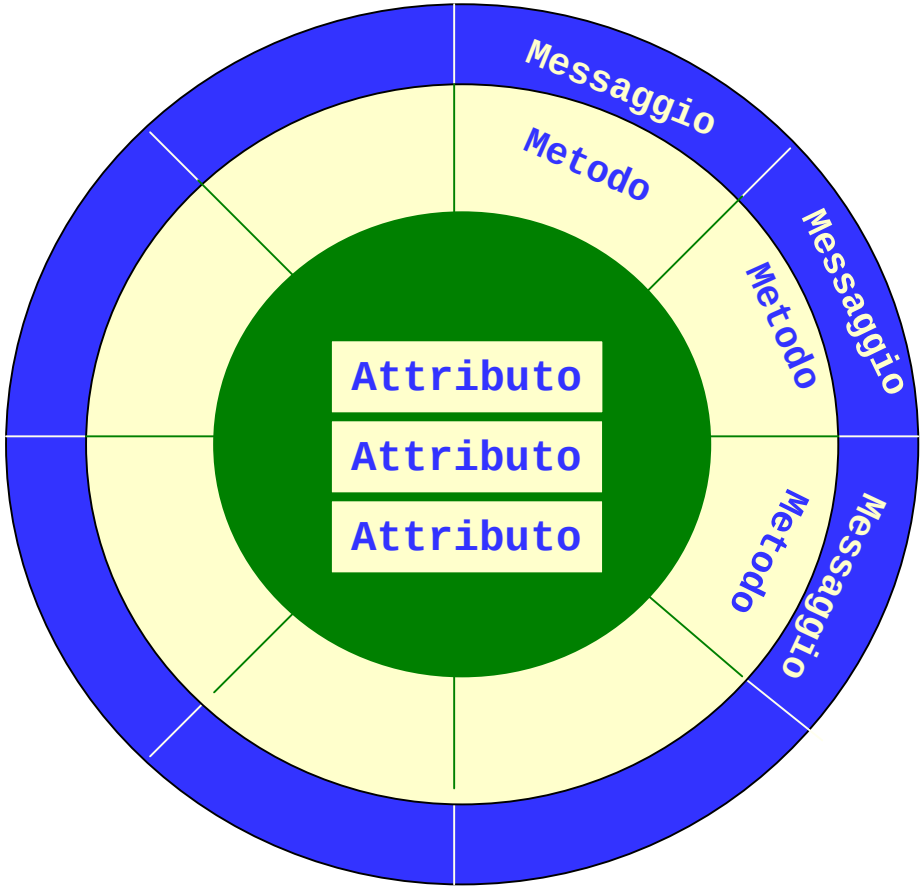
Oggetti e Classi di Oggetti

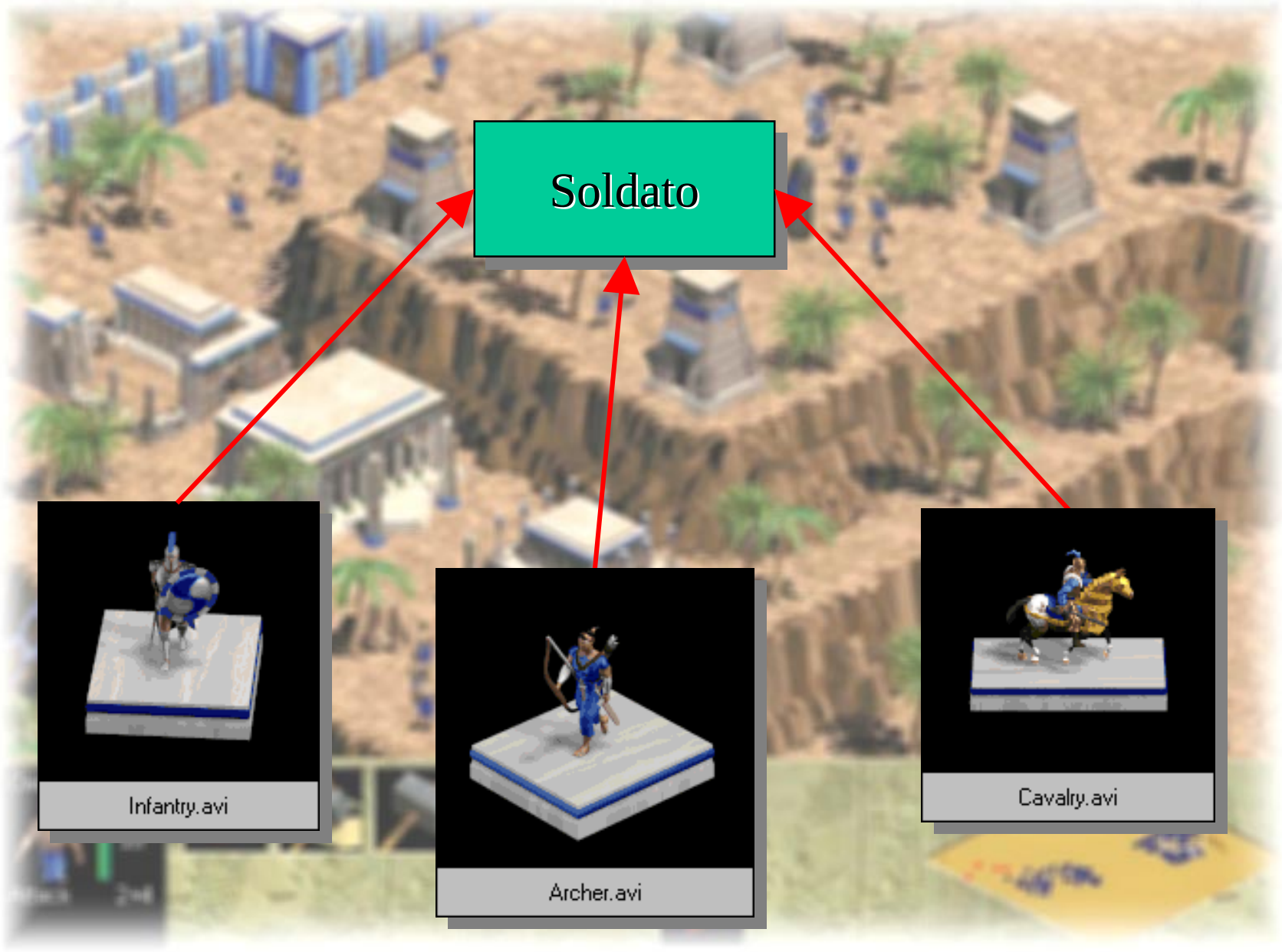


Incapsulamento

- Interfaccia
  - Definizione dei messaggi (*metodi*) possibili
  - Unico punto di accesso per l'utente all'oggetto
- Implementazione
  - Realizzazione dell'azione corrispondente a un messaggio
  - Nascosta all'utente
  - Dichiarazione di variabili private (*attributi*) che definiscono lo *stato* dell'oggetto

Rappresentazione di un oggetto

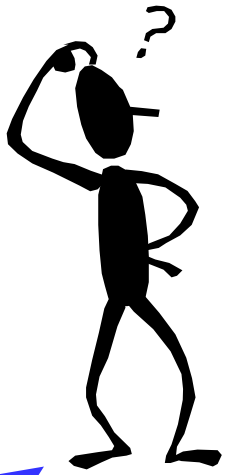



Soldato
Infantry.avi
Archer.avi
Cavalry.avi

Esempio: i soldati

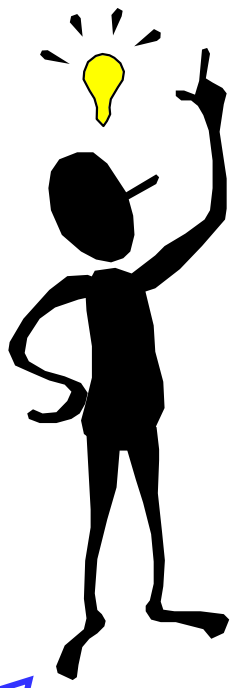
- Tutti i soldati devono capire il messaggio *attacca*. Il messaggio ha conseguenze diverse a seconda del tipo di soldato:
  - un arciere scaglia una freccia
  - un fante colpisce di spada
  - un cavaliere lancia una lancia
- Il giocatore/computer deve gestire una lista di soldati e poter chiedere ad ogni soldato di attaccare indipendentemente dal tipo ma basandosi solo sulla posizione.

```
// esempio di codice rigido!

void attacca(Soldato tipo)
{
  if      ( tipo == arciere )
    scagliaLaFreccia();
  else if ( tipo == fante )
    colpisciDiSpada();
  else if ( tipo == cavaliere )
    lanciaLaLancia();
}
```



Che succede se devo aggiungere nuovi tipo di soldato?

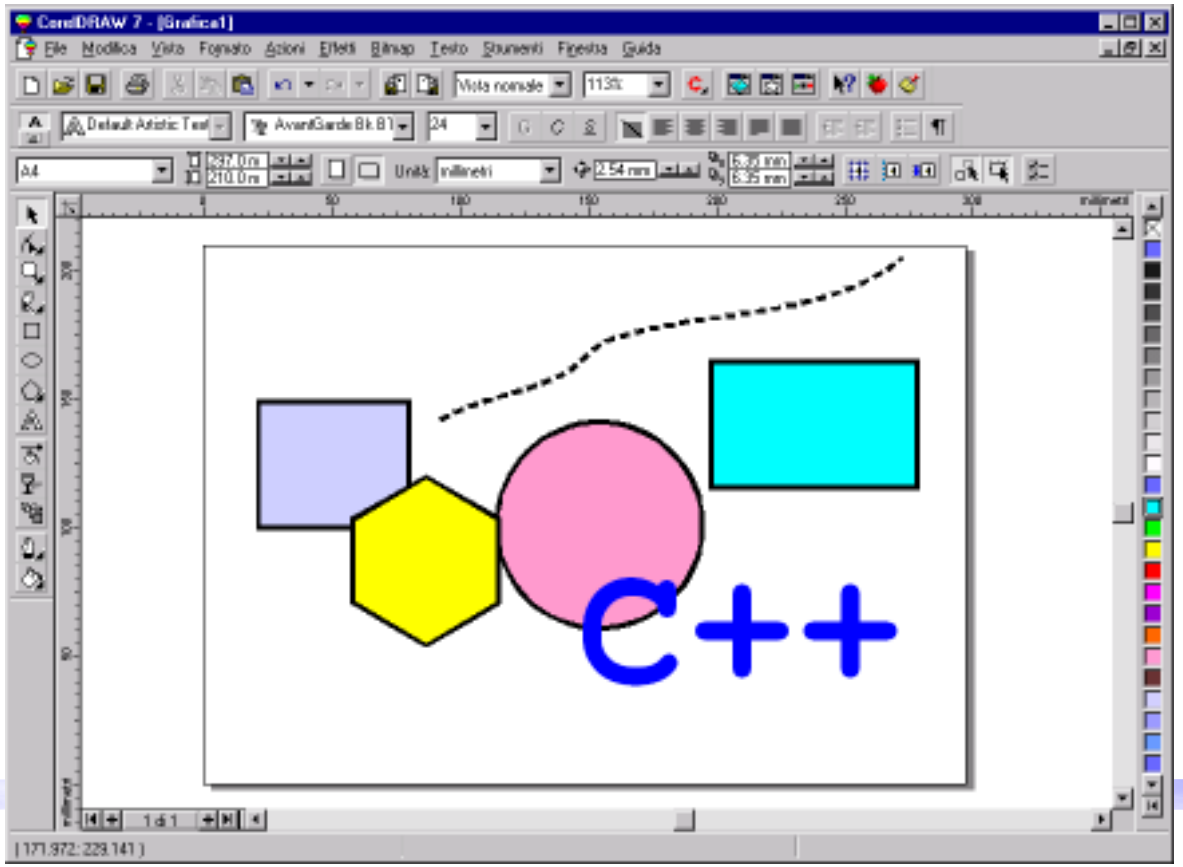
```
// esempio di codice flessibile


Soldato s;
// . . .
s.attacca();
// . . .
```



Se devo aggiungere nuovi tipo di soldato il codice non cambia!

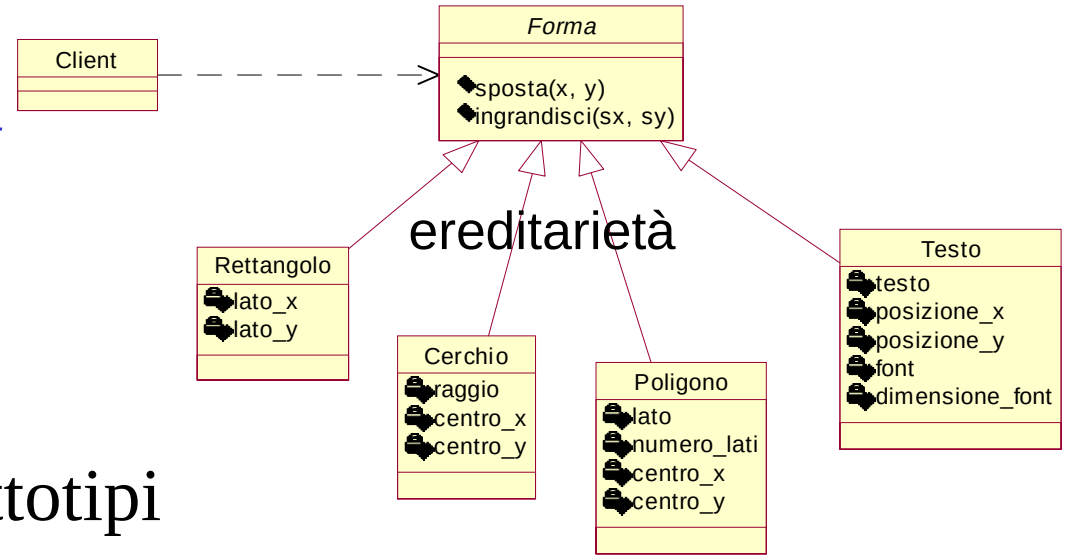
Classi astratte e polimorfismo

- Diversi tipi di oggetti hanno comportamenti comuni che possono essere considerati in astratto:
  - *disegna*, *sposta*, *ingrandisci*, etc.



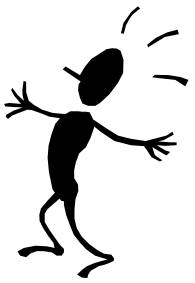
Vantaggi del polimorfismo

- I messaggi comuni (*sposta*, *ingrandisci*, …) sono definiti in maniera *astratta* in nella classe di base **`forma`**
- Le sottoclassi li *ereditano* e li implementano in modo specifico
- Il programma *client* ignora i dettagli di implementazione e gestisce in modo omogeneo tutti i sottotipi di forma



Perché programmare a *oggetti*?

- Programmare a *oggetti*…
  - non velocizza l'esecuzione dei programmi…
  - non ottimizza l'uso della memoria...



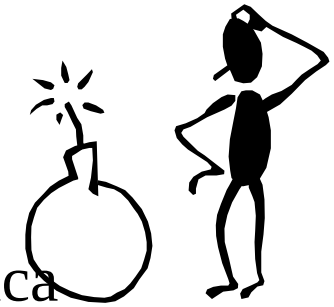
E allora perché programmare a *oggetti*?

- Programmare a *oggetti* facilita la progettazione, lo sviluppo e il mantenimento di sistemi software molto complessi!

Vantaggi dei programmi a oggetti

- La programmazione ad oggetti consente di:
  - Ridurre la dipendenza del codice di alto livello dalla rappresentazione dei dati
    - l'accesso ai dati è mediato da un'interfaccia
  - Riutilizzare del codice di alto livello
    - definendo opportune classi astratte
  - Sviluppare moduli indipendenti l'uno dall'altro
    - I clienti che dipendono da interfacce che non dipendono dalla loro implementazione
    - Le dipendenze da classi astratte sono più "leggere"

Difetti del software *non mantenibile*



- **Rigidità**
  - Non può essere cambiato con facilità
  - Non può essere stimato l'impatto di una modifica
- **Fragilità**
  - Una modifica singola causa una cascata di modifiche successive
  - I bachi sorgono in aree concettualmente separate dalle aree dove sono avvenute le modifiche
- **Non riusabilità**
  - Esistono molte interdipendenze, quindi non è possibile estrarre parti che potrebbero essere comuni

Open/Closed principle

- Un buon codice deve essere:
  - ***Aperto*** a estensioni
    - Le richieste di funzionalità del software sono in continua evoluzione
  - ***Chiuso*** a modifiche
    - Modificare un codice funzionante può introdurre bachi…
- La programmazione a oggetti, con il meccanismo delle classi astratte, permette di applicare questo principio

…**C++** e *Object Orientation*

- *Object Orientation* implementata in **C++** attraverso il concetto di classe:

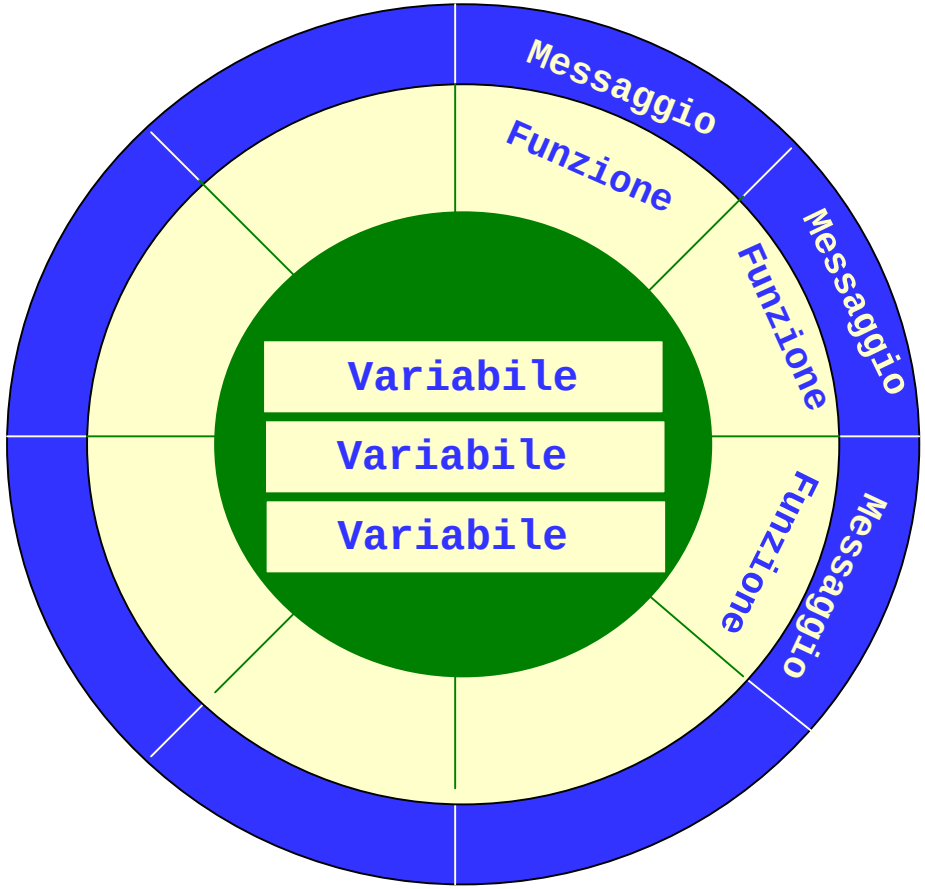
➢ Attributi

  – Le variabili di una classe definiscono lo stato dell'oggetto

➢ Metodi

  – Le funzioni definite in una classe implementano la risposta ai messaggi

Rappresentazione di un oggetto in C++



Controllo dei *tipi*

- Controllare i tipi significa verificare che ad un oggetto vengano inviati solo messaggi che è in grado di comprendere:
  - controllo del nome del metodo
  - controllo della lista degli argomenti
- In **`C++`** il controllo è fatto dal compilatore (*strong typing*)
- In altri linguaggi (ad esempio **`SmallTalk`**) è fatto a run-time (*weak typing*)

Assegnazione della funzione ad un messaggio

- I metodi di una classe costituiscono l'interfaccia della classe (cioè i messaggi che l'oggetto può interpretare)
- La funzione è assegnata al messaggio in fase di codifica (*early binding*)
- Può essere necessario assegnare la funzione al messaggio a run-time (*late binding*)

Polimorfismo

Typing & Binding

Typing	Definizione dei messaggi e degli argomenti	Strong	Consistenza dei tipi verificata dal compilatore
		Weak	Consistenza dei tipi verificata a run-time
Binding	Assegnazione di un metodo ad un messaggio	Early	In fase di programmazione INFLESSIBILE
		Late	A run-time POLIMORFISMO

Ereditarietà

- Polimorfismo con tipi controllati dal compilatore (Strong typing & late binding).

Come?

- In **C++** viene implementato tramite il concetto di ereditarietà (inheritance)
- Classe astratta: definisce i messaggi
- Classe concreta: assegna i metodi ai messaggi

La classe concreta eredita da quella astratta