

# Informatica e Computazione

## Introduzione a UML

**Armando Tacchella**

Corso di Laurea in Ingegneria Informatica  
Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

# Sommario

- 1 Introduzione
- 2 Diagramma delle classi
- 3 Macchine a stati
- 4 Diagramma di sequenza

# Algoritmi vs. applicazioni

Gli strumenti di progettazione validi per **singoli algoritmi** non sono adeguati per il progetto di **applicazioni** ottenute combinando algoritmi, strutture dati, interfacce utente e comunicazione su larga scala.




Specifiche quali lo pseudo-codice o i diagrammi di flusso

- hanno una **complessità** pari (o superiore) alla complessità del **programma** che le realizza;
- hanno un livello di dettaglio **elevato e rigido**;
- hanno difficoltà nella specifica di **elementi comuni** diversi dagli algoritmi quali
  - ▶ strutture dati e relative librerie,
  - ▶ comunicazione via rete e via file,
  - ▶ interazione con l'utente.

# Specifiche strutturate per il software

- A partire dagli anni '70 del secolo scorso vi sono diverse proposte.
  - ▶ David Parnas nell'articolo "*On the criteria to be used in decomposing systems into modules*" del 1972 introduce il problema della **modularità**.
  - ▶ Grady Booch nell'articolo "*Object Oriented Design*" del 1982 introduce una delle prime **metodologie** di progettazione orientata agli oggetti.
  - ▶ James Rumbaugh e altri nel libro "*Object Oriented Modeling and Design*" del 1990 introducono OMT (Object Modeling Technique).
- Nel 1999, Rumbaugh e Booch insieme a Ivar Jacobson, propongono Unified Modeling Language (UML) come **sintesi** delle varie proposte.
- La proposta di **metodi strutturati** segue l'evoluzione dei linguaggi, da **procedurali** (C, Pascal), a **modulari** (Modula 2, Ada), fino ad arrivare a quelli **orientati agli oggetti** (C++, Java, C#).

# Object Management Group (OMG - [www.omg.org](http://www.omg.org))

- Organizzazione internazionale aperta no-profit fondata nel 1989.
- Raggruppa venditori, utenti finali, università e agenzie governative.
- Propone standard di progettazione e sviluppo in ambito informatico:
  - ▶ Unified Modeling Language (UML) 
  - ▶ Systems Modeling Language (SysML) 
  - ▶ Model Driven Architecture (MDA) 
  - ▶ ...

*“UML helps you **specify, visualize, and document** models of software systems, including their **structure and design...**”*

*“Using any one of the large number of UML-based tools on the market, you can analyze your future application’s requirements and design a solution that meets them, representing the results using UML 2.0’s **thirteen standard diagram types.**”*

*“You can model just about any type of **application**, running on any type and combination of hardware, operating system, programming language, and network, in UML”*

# I 13 diagrammi UML... ([www.uml.org](http://www.uml.org))

|                                 |                      |  |
|---------------------------------|----------------------|--|
| <b>Structure<br/>Diagrams</b>   | Class                | Architettura delle classi e loro relazioni |
|                                 | Object               | Oggetti dinamici e loro relazioni          |
|                                 | Component            | Architettura dei moduli                    |
|                                 | Composite Structure  |  |
|                                 | Package              | Architettura dei package                   |
|                                 | Deployment           |  |
| <b>Behavior<br/>Diagrams</b>    | Use Case             | Esemplificazioni di funzionamento          |
|                                 | Activity             | Modello per flussi di esecuzione           |
|                                 | State Machine        | Macchine a stati estese                    |
| <b>Interaction<br/>Diagrams</b> | Sequence             | Chiamate tra metodi di oggetti             |
|                                 | Communication        |  |
|                                 | Timing               |  |
|                                 | Interaction Overview |  |

# Diagramma delle classi

- **Classi**: nome, attributi (campi) e metodi con relativa visibilità.
- **Relazioni** tra le classi:
  - ▶ **ereditarietà (IS-A)** tra una classe e le sue derivate (dirette);
  - ▶ **aggregazione (HAS-A)** tra una classe e le sue componenti;
  - ▶ **utilizzo (USES)** tra una classe e le sue accessorie.
- Notazione per altre relazioni, ad esempio **implementazione** ed **istanziamento**.
- Notazione per classi **astratte**, per classi **generiche** (template) e per **interfacce**.



# Rappresentazione delle classi

| ClassName   |
|---|
| +publicAttribute: <type><br>-privateAttribute: <type>                       |
| +publicMethod(param:<type>): <type><br>-privateMethod(param:<type>): <type> |

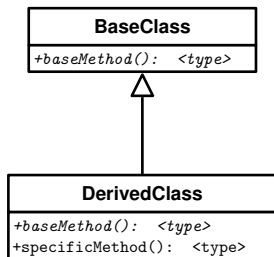
```
class ClassName {  
  
    public:  
        <type> publicAttribute;  
        <type> publicMethod( <type> param );  
  
    private:  
        <type> privateAttribute;  
        <type> privateMethod( <type> param );  
  
};
```

| ClassName                           |
|-------------------------------------|
| +publicAttribute: <type>            |
| +publicMethod(param:<type>): <type> |

| ClassName |
|-----------|
|-----------|

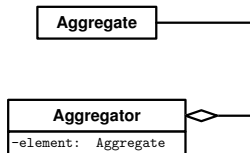
- Il contenuto della classe può essere modificato a seconda del **livello di dettaglio** desiderato.
- La visualizzazione più comune prevede l'**interfaccia pubblica** della classe (attributi e metodi pubblici).

# Rappresentazione dell'ereditarietà (IS-A)



```
class BaseClass {  
  
    public:  
        virtual <type> baseMethod( );  
  
};  
  
class DerivedClass : public BaseClass {  
  
    public:  
        virtual <type> baseMethod( );  
        <type> specificMethod( );  
  
};
```

# Rappresentazione dell'aggregazione (HAS-A)



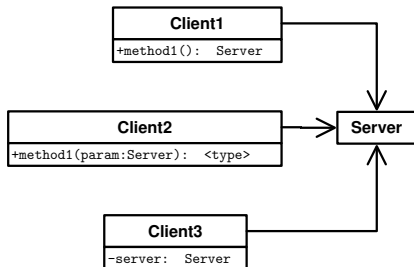
```
class Aggregate {
    ...
};

class Aggregator {
    public:
        ...

    private:
        Aggregate element;
        ...
};
```

- L'idea alla base dell'aggregazione è che gli oggetti **Aggregate** hanno un **tempo di vita** uguale a quello degli oggetti **Aggregator**.
- Se **Aggregator** allocasse dinamicamente oggetti **Aggregate** e li distruggesse all'atto della sua distruzione, si avrebbe **aggregazione**.

# Rappresentazione dell'utilizzo (USES)



```
class Server {
    ...
};

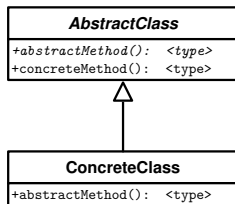
class Client1 {
    public:
        Server method1( );
    ...
};

class Client2 {
    public:
        <type> method1( Server param );
    ...
};

class Client3 {
    ...
    private:
        Server* serverPtr;
};
```

- Il principio di USES è che gli oggetti **Client** utilizzano oggetti **Server**, ma questi non sono “proprietà” degli oggetti **Client**.
- Se un oggetto **Client** ha un puntatore a un oggetto **Server**, non può essere **responsabile** della deallocazione del **Server** (altrimenti sarebbe aggregazione).

# Rappresentazione delle classi astratte

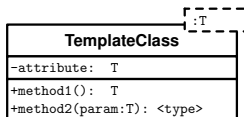


```
class AbstractClass {
    public:
        virtual <type> abstractMethod( ) = 0;
        virtual <type> concreteMethod( );
};

class ConcreteClass : public AbstractClass {
    public:
        virtual <type> abstractMethod( );
        ...
};
```

- Una classe è astratta quando **almeno uno** dei suoi metodi è astratto: in C++ sono i metodi “*pure virtual*”
- Se **ConcreteClass** eredita da **AbstractClass** deve ridefinire i metodi astratti e può ridefinire i metodi concreti.

# Rappresentazione delle classi generiche



```
template < class T >
class TemplateClass {

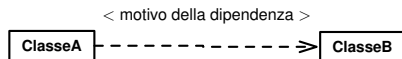
    public:
        T      method1( );
        <type> method2(T param);

    private:
        T attribute;

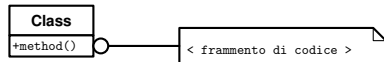
};
```

- All'interno di **TemplateClass** il parametro "T" può comparire come attributo, tipo di ritorno, e tipo di qualche parametro.
- Oltre a "T" possono essere presenti anche riferimenti "T&", puntatori "T\*" e istanze di altri template realizzate con "T"

# Altra notazione



Relazione di dipendenza



Annotazioni

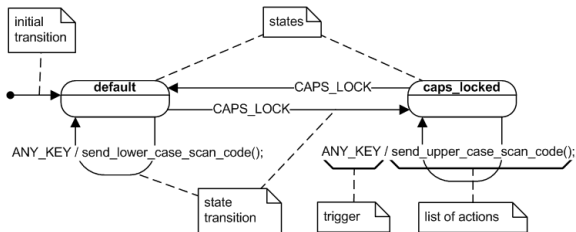
- Vi possono essere delle **dipendenze** che non sono facilmente riconducibili ai casi di USES o HAS.
- La dipendenza è un concetto **più generale** che può essere utilizzato in prima approssimazione al posto di USES o HAS.
- Le annotazioni consentono di aggiungere informazioni **non strutturali** ai diagrammi.
- Solitamente vengono impiegate per mostrare **frammenti di codice** significativi.

# Macchine a stati

- Una realizzazione fortemente estesa degli **automi a stati finiti**
- Variante degli *statechart* di D. Harel
- Consentono di modellare sistemi che
  - ▶ hanno un numero di **stati** (o modi di controllo)
  - ▶ hanno **transizioni** ben definite tra gli stati
  - ▶ reagiscono ad **eventi esterni**
  - ▶ hanno evoluzioni di durata (potenzialmente) **non finita**



# Esempio e notazione

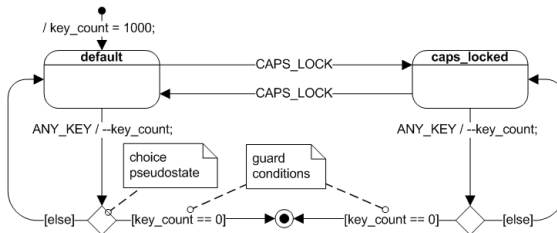


- **Stati**: rettangoli stondati
- **Transizioni**: frecce tra stati
- **Eventi/Azioni**: annotazioni sulle frecce
- **Stato iniziale**: cerchio nero
- **Commenti**: “foglietto” con linea tratteggiata

# Macchine a stati estese

- In UML è possibile utilizzare
  - ▶ **Variabili**: valori che vengono modificati durante le transizioni
  - ▶ **Guardie**: condizioni che determinano l'esecuzione di transizioni alternative
- Con queste estensioni, il formalismo diventa **Turing-completo**
- In questo senso, le macchine a stati UML sono **estensioni** degli automi a stati finiti

# Esempio e notazione

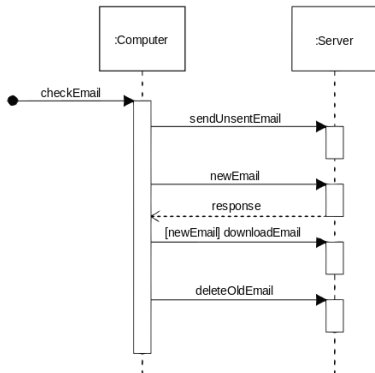


- **Variabili:** come nei linguaggi di programmazione
- **Pseudo-stati:** rombi
- **Guardie:** condizioni sulle transizioni
- Le guardie sono **sempre** associate a pseudo-stati di scelta
- Le variabili vengono modificate nella parte delle azioni di una transizione

# Diagramma di Sequenza

- Mostra **come** gli oggetti comunicano tra di loro e in quale **ordine**
- Sono relativi a particolari scenari di utilizzo
- Hanno due dimensioni:
  - ▶ in **orizzontale** sono elencati gli oggetti che compongono lo scenario
  - ▶ in **verticale** è rappresentato il tempo che scorre (dall'alto verso il basso)

# Esempio e notazione



- **Oggetti:** Rettangoli etichettati
- **Tempo di vita degli oggetti:** barre verticali
- **Frecce:** messaggi (chiamate a metodi)
  - ▶ **frecce solide:** chiamate sincrone
  - ▶ **frecce aperte:** chiamate asincrone
- **Frecce tratteggiate:** risposte (chiamate a metodi)