UML (unified modeling language)

I sistemi, i modelli, il linguaggio UML

1. **I sistemi**
2. **I modelli**
3. **Perché nasce il linguaggio UML**
4. **Il linguaggio UML**
5. **Il diagramma delle classi e le relazioni tra classi**
6. **L'applicazione online "draw.io" per creare il diagramma delle classi**

1. I sistemi : definizioni ed esempi

In informatica e meccanica per sistema si intende, **un insieme di componenti interconnessi che lavorano insieme per raggiungere un determinato obiettivo.** E' una cosa reale che esiste e funziona.
Spesso si presenta come unico oggetto od entità

Esempi in informatica, meccanica ed edilizia:

- Un personal computer, un cellulare, una stampante, un tablet, un server, una rete di computer, un sistema di gestione database, un sistema di elaborazione di transizione online, ecc..
- Una automobile, un robot, uno scooter, una lavatrice, una nave,un trattore, un distributore di bevande o di carburante , ecc..
- Un edificio, una scuola, una biblioteca, sistema impiantistico (idraulico, elettrico) ecc..






Riassumendo:

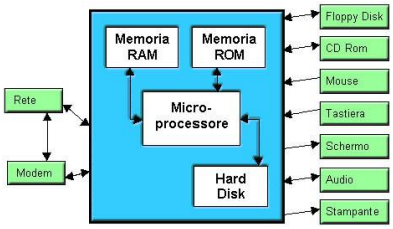
1. Un sistema è composto da diverse parti che hanno funzioni specifiche
2. Le parti sono interconnesse tra loro ed interagiscono per raggiungere l'obiettivo del sistema: ad esempio un'auto è composta da: motore, telaio, organi di trasmissione, carrozzeria, ruote, ecc..
3. Un sistema svolge nel suo complesso una funzione specifica: ad esempio l'auto trasporta le persone, il camion persone e merci, la stampante trasferisce le informazioni su carta

2. I modelli

Un modello è una rappresentazione semplificata di un sistema :
Esempi di modelli possono essere:

- Un diagramma, una tabella, un grafico, un grafo, una piantina di un edificio
- Un insieme di equazioni matematiche
- Un programma (software) con simulazione al computer
- Un disegno al cad, una simulazione al cad
- Ecc..

L'obiettivo di un modello è di aiutarci a capire come funziona un sistema, a progettarlo, o a prevederne il comportamento



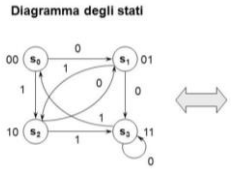
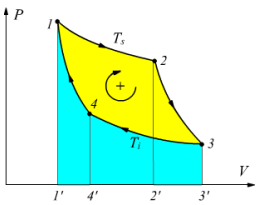
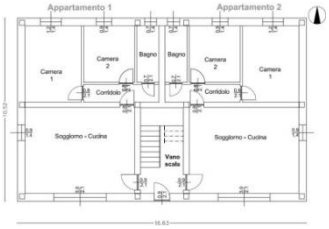


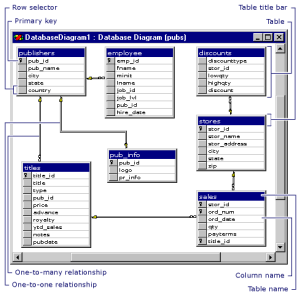
Tabella degli stati

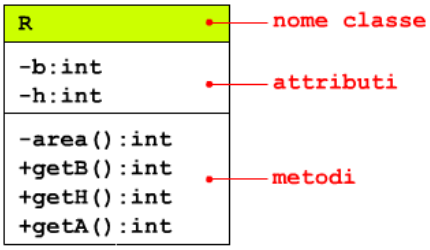
	0	1	U
S_0	S_1	S_2	00
S_1	S_3	S_2	01
S_2	S_1	S_3	10
S_3	S_3	S_0	11





$$\begin{cases} y = 2x + 1 \\ 3x + y = 0 \end{cases}$$





3. Perché nasce il linguaggio UML

Il linguaggio UML (Unified Modeling Language) è nato per affrontare diverse sfide nel mondo dello sviluppo software. Immagina che, prima della sua creazione negli anni '90, ogni programmatore e team, "architetto" di software utilizzasse i propri "schemi" e "disegni" per rappresentare un sistema. Questo rendeva la comunicazione tra team diversi, o anche all'interno dello stesso team in momenti diversi, molto difficile e soggetta a interpretazioni errate.
Ecco i motivi principali della sua nascita:

- **Standardizzazione:** L'obiettivo primario era creare un **linguaggio visivo standard** per la modellazione di sistemi software. Proprio come un architetto edile usa dei blueprint universalmente comprensibili, UML mirava a fornire un insieme di notazioni grafiche che tutti gli sviluppatori, analisti e altri stakeholder potessero capire. Questo eliminava la confusione e facilitava la comunicazione.
- **Gestione della complessità:** I sistemi software stavano diventando sempre più complessi. UML offriva un modo per **visualizzare e astrarre** diversi aspetti di questi sistemi (struttura, comportamento, interazioni, ecc.) attraverso vari tipi di diagrammi. Questa visualizzazione aiutava a comprendere meglio la complessità e a gestirla in modo più efficace.
- **Comunicazione efficace:** Come accennato, UML migliorava la **comunicazione** tra i membri del team di sviluppo, con i clienti e con altri soggetti interessati. Un modello UML ben fatto poteva trasmettere informazioni complesse in modo più chiaro e conciso rispetto a lunghe descrizioni testuali o, peggio, alla sola lettura del codice.
- **Supporto al processo di sviluppo:** UML non è una metodologia di sviluppo software in sé, ma è stato progettato per essere **compatibile con i principali approcci object-oriented** dell'epoca (come OMT, Booch e OOSE, i cui "padri fondatori" sono stati i principali artefici di UML). Forniva uno strumento per formalizzare e documentare le diverse fasi del ciclo di vita del software, dall'analisi dei requisiti alla progettazione e all'implementazione.
- **Interoperabilità degli strumenti:** Si sperava che uno standard come UML avrebbe facilitato l'**interoperabilità tra diversi strumenti di modellazione software**. Se tutti gli strumenti utilizzassero lo stesso linguaggio, lo scambio di modelli e informazioni sarebbe diventato più semplice.

In sostanza, UML è nato dal bisogno di **ordine, chiarezza e comunicazione efficace** in un campo, lo sviluppo software, che stava diventando rapidamente più complesso e collaborativo. Ha fornito un "alfabeto" e una "grammatica" visiva per descrivere i sistemi software, un po' come uno spartito musicale permette a musicisti diversi di suonare la stessa melodia.

4. Il linguaggio UML

Figura 11 Il logo dello standard UML gestito dall'Object Management Group (OMG)

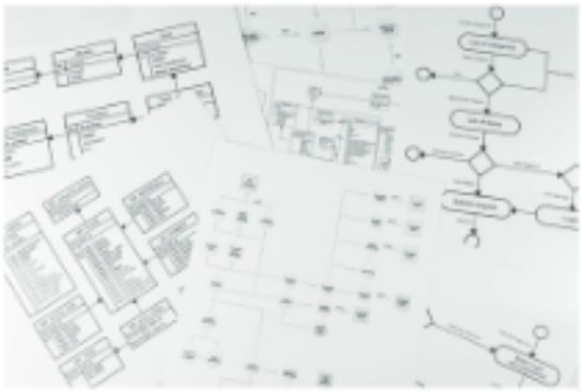


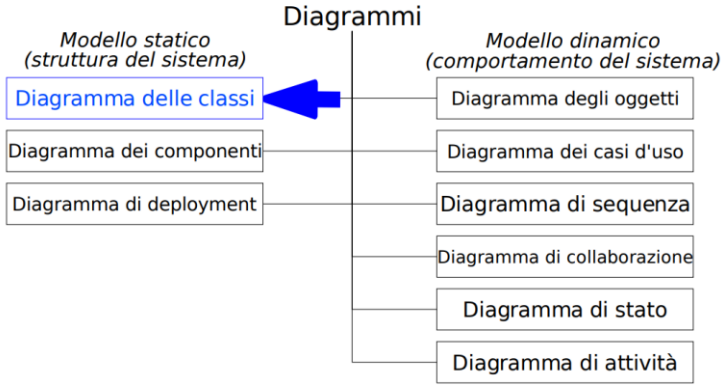
Figura 12 L'UML comprende una decina di tipologie di diagrammi per la descrizione e la progettazione di sistemi

Il linguaggio UML, definito nel 1996 da Booch, Rumbaugh e Jacobson, nasce nel contesto del paradigma di programmazione orientato agli oggetti e definisce una decina di tipologie di diagrammi (Figura 12), suddivisi in **diagrammi di struttura** e **diagrammi di comportamento**. L'obiettivo di ciascun diagramma è quello di catturare ed esprimere in forma grafica un aspetto del sistema, per favorire le operazioni di analisi, progettazione e sviluppo. I diagrammi si basano su elementi grafici e testuali formalizzati in termini sia di significato sia di aspetto (anche se sono diffuse numerose varianti). Inoltre, è prevista la possibilità di completare il diagramma con elementi testuali liberi, per commentare parti e aspetti che non possono essere modellati dagli elementi formali disponibili.

Fra i diagrammi UML ne esistono due, correlati e simili fra loro, che sono particolarmente utili nell'OOP e nello sviluppo di database (descritti nel prossimo modulo):

1. **il diagramma delle classi** (*class diagram*), che rappresenta le entità, cioè le classi del sistema, e le relazioni che intercorrono tra esse;
2. **il diagramma degli oggetti** (*object diagram*), che rappresenta gli oggetti, cioè le istanze del sistema, e le relazioni che intercorrono tra essi.

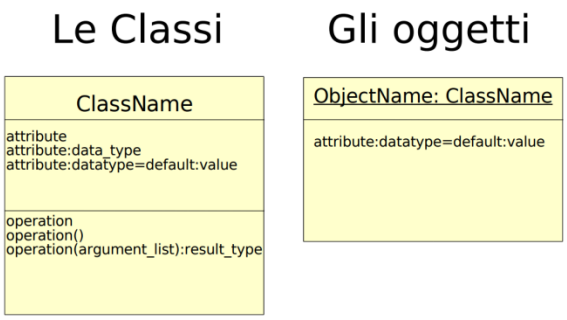
3. il diagramma dei casi d'uso.

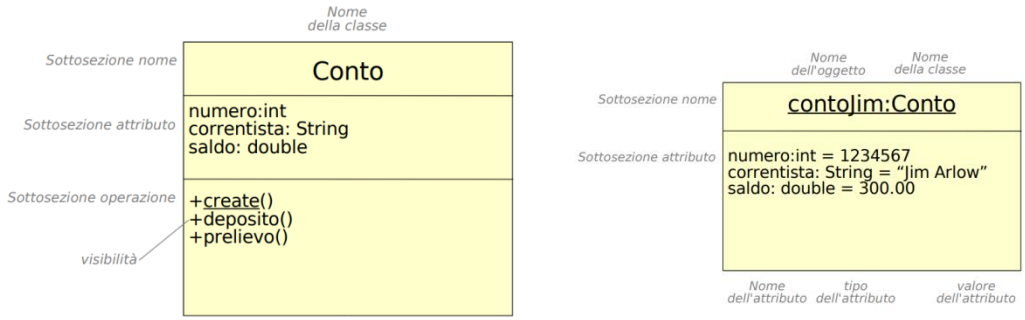


UML

5. Il diagramma delle classi ed oggetti, le relazioni tra classi

Nel linguaggio UML, una classe in un diagramma delle classi viene rappresentata graficamente come un **rettangolo** diviso orizzontalmente in **tre sezioni**. Queste sezioni sono utilizzate per visualizzare il nome della classe, i suoi attributi e i suoi metodi (o operazioni).



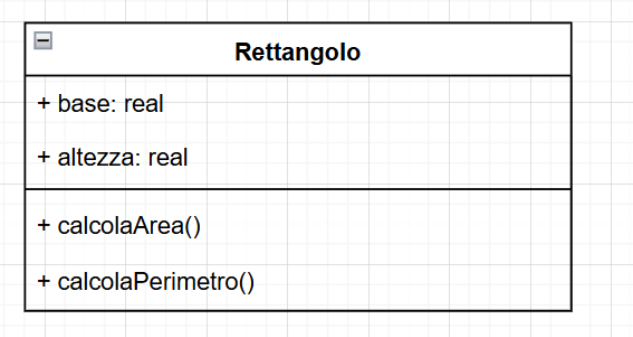



```python
#classi ed oggetti programmazione OOP
class Rettangolo:
    def __init__(self, dimensione1, dimensione2):
        self.base = dimensione1
        self.altezza = dimensione2

    def calcolaPerimetro(self):
        return self.base *2 + self.altezza * 2

    def calcolaArea(self):
        return self.base * self.altezza

#main
rettangolo1 = Rettangolo(3,7)
```



Ecco una descrizione dettagliata di ciascuna sezione:

1. Sezione Superiore: Nome della Classe

- Questa è la sezione più alta del rettangolo e contiene il **nome della classe**.
- Il nome della classe dovrebbe essere un sostantivo o una frase nominale che descrive l'entità o il concetto rappresentato dalla classe.
- Per convenzione, i nomi delle classi iniziano con una lettera maiuscola (PascalCase o UpperCamelCase).
- In questa sezione possono anche essere indicati degli **stereotipi** racchiusi tra doppie virgolette (<<Stereotipo>>). Gli stereotipi estendono il vocabolario UML per classificare gli elementi del modello in modo specifico per un determinato dominio o processo (ad esempio, <<Entity>>, <<Control>>, <<Boundary>>).
- Se la classe è **astratta**, il suo nome viene solitamente scritto in **corsivo**.

```
@startuml
class Persona
class "Utente Autenticato" <<Security>>
abstract class Animale
@enduml
```

2. Sezione Centrale: Attributi

- Questa sezione si trova al centro del rettangolo e contiene l'elenco degli **attributi** della classe. Gli attributi rappresentano le proprietà o le caratteristiche di un oggetto di quella classe.
- Ogni attributo viene solitamente visualizzato su una riga separata e segue un formato standard:
  `[visibilità] nomeAttributo : tipoDato [= valoreIniziale] [{proprietà}]`

    - **Visibilità:** Indica l'accessibilità dell'attributo dall'esterno della classe. I simboli comuni sono:
        - + (pubblico): L'attributo è accessibile da qualsiasi altra classe.
        - - (privato): L'attributo è accessibile solo all'interno della classe stessa.
        - # (protetto): L'attributo è accessibile all'interno della classe stessa e delle sue sottoclassi.
        - ~ (package/default): L'attributo è accessibile solo alle classi all'interno dello stesso package (se il linguaggio di modellazione supporta i package).
    - **nomeAttributo:** Il nome dell'attributo, solitamente scritto in minuscolo o camelCase (ad esempio, nome, dataDiNascita).
    - **: tipoDato:** Il tipo di dato dell'attributo (ad esempio, String, Integer, Boolean, Date).
    - [= valoreIniziale]: Un valore predefinito opzionale per l'attributo quando viene creato un oggetto della classe.
    - [{proprietà}]: Proprietà opzionali che specificano ulteriori caratteristiche dell'attributo (ad esempio, {readOnly}, {static}).

```
@startuml
class Persona {
    + nome : String
    - eta : Integer
    # dataNascita : Date
}
@enduml
```

3. Sezione Inferiore: Metodi (Operazioni)

- Questa è la sezione inferiore del rettangolo e contiene l'elenco dei **metodi** (o operazioni) della classe. I metodi rappresentano i comportamenti o le azioni che un oggetto di quella classe può eseguire.
- Ogni metodo viene solitamente visualizzato su una riga separata e segue un formato standard:
  `[visibilità] nomeMetodo ( [parametro1 : tipo1, parametro2 : tipo2, ...] ) [ : tipoRestituzione] [{proprietà}]`

    - **Visibilità:** Indica l'accessibilità del metodo dall'esterno della classe (gli stessi simboli usati per gli attributi: +, -, #, ~).
    - **nomeMetodo:** Il nome del metodo, solitamente un verbo o una frase verbale che descrive l'azione (ad esempio, calcolaArea(), setNome(), getInfo()). Per convenzione, i nomi dei metodi iniziano con una lettera minuscola e seguono la convenzione camelCase.
    - ([parametro1 : tipo1, parametro2 : tipo2, ...]): La lista dei parametri del metodo, inclusi i loro nomi e tipi di dato. Se il metodo non ha parametri, le parentesi sono vuote ().
    - [: tipoRestituzione]: Il tipo di dato restituito dal metodo (se presente). Se il metodo non restituisce alcun valore, si può usare void.
    - [{proprietà}]: Proprietà opzionali che specificano ulteriori caratteristiche del metodo (ad esempio, {abstract}, {static}).

```
@startuml
class Persona {
    + getNome() : String
    + setEta(nuovaEta : Integer) : void
    - calcolaCodiceFiscale(cognome : String, nome : String, data : Date) :
String
}
abstract class Animale {
    + verso() : String {abstract}
}
@enduml
```

Le relazioni:

Nel linguaggio UML (Unified Modeling Language), i diagrammi delle classi rappresentano la struttura statica di un sistema, mostrando le classi, i loro attributi, i loro metodi e le relazioni tra di esse. Ecco i principali tipi di relazione che si possono trovare in un diagramma delle classi, con un esempio per ciascuno:

1. Associazione: Rappresenta una connessione generale tra due classi. Indica che oggetti di una classe possono essere collegati a oggetti di un'altra classe.

- **Simbolo:** Una linea continua che collega le due classi. Può avere delle frecce per indicare la navigabilità (in quale direzione è possibile "navigare" tra gli oggetti). Può anche avere delle etichette per descrivere il ruolo della relazione e delle molteplicità per indicare quanti oggetti di una classe sono associati a quanti oggetti dell'altra.
- **Esempio:** Un'associazione tra la classe Persona e la classe Libro. Una persona può leggere uno o più libri, e un libro può essere letto da zero o più persone.

```
@startuml
class Persona
class Libro

Persona -- Libro : legge
@enduml
```

2. Aggregazione: È un tipo speciale di associazione che rappresenta una relazione "ha-un" (has-a) tra classi, dove una classe contiene o è composta da un'altra. Tuttavia, l'oggetto contenuto può esistere indipendentemente dall'oggetto contenitore.

- **Simbolo:** Una linea continua con un rombo vuoto all'estremità della classe contenitore.
- **Esempio:** Un'aggregazione tra la classe Dipartimento e la classe Studente. Un dipartimento "ha" degli studenti, ma gli studenti possono esistere anche se il dipartimento viene sciolto (ad esempio, possono essere trasferiti ad altri dipartimenti).

```
@startuml
class Dipartimento
class Studente

Dipartimento o-- Studente
@enduml
```

3. Composizione: È un tipo più forte di aggregazione, anch'essa una relazione "ha-un", ma in questo caso l'oggetto contenuto dipende esistenzialmente dall'oggetto contenitore. Se l'oggetto contenitore viene distrutto, anche gli oggetti contenuti vengono distrutti.

- **Simbolo:** Una linea continua con un rombo pieno all'estremità della classe contenitore.
- **Esempio:** Una composizione tra la classe Automobile e la classe Motore. Un'automobile "ha" un motore, e se l'automobile viene demolita, anche il suo motore cessa di esistere come parte di quella specifica automobile.

```
@startuml
class Automobile
class Motore

Automobile *-- Motore
@enduml
```

4. Generalizzazione (Ereditarietà): Rappresenta una relazione "è-un-tipo-di" (is-a-type-of) tra una classe più generale (superclasse o classe padre) e una o più classi più specifiche (sottoclassi o classi figlie). La sottoclasse eredita gli attributi e i metodi della superclasse e può aggiungerne di nuovi o modificarne alcuni.

- **Simbolo:** Una linea continua con una freccia a triangolo vuoto che punta verso la superclasse.
- **Esempio:** Una generalizzazione tra la classe Animale (superclasse) e le classi Cane e Gatto (sottoclassi). Un cane "è un tipo di" animale, e un gatto "è un tipo di" animale. Entrambi ereditano caratteristiche generali degli animali (come avere un nome, un'età, ecc.) ma possono avere caratteristiche specifiche (come abbaiare per il cane e miagolare per il gatto).

```
@startuml
class Animale
class Cane
```

```
class Gatto

Animale <|-- Cane
Animale <|-- Gatto
@enduml
```

5. Realizzazione (Implementazione): Rappresenta una relazione tra una classe e un'interfaccia. La classe realizza (implementa) l'interfaccia, il che significa che fornisce un'implementazione per tutti i metodi definiti nell'interfaccia.

- **Simbolo:** Una linea tratteggiata con una freccia a triangolo vuoto che punta verso l'interfaccia.
- **Esempio:** Un'interfaccia Ordinabile che definisce un metodo ordina(), e una classe Prodotto che realizza questa interfaccia, fornendo una specifica implementazione del metodo ordina() per i prodotti.

```
@startuml
interface Ordinabile {
    ordina()
}
class Prodotto implements Ordinabile {
    + ordina()
}
@enduml
```

6. Dipendenza: Rappresenta una relazione di "uso" tra due classi, in cui un cambiamento in una classe (il fornitore) può influenzare l'altra classe (il cliente), ma non c'è una relazione strutturale permanente tra le istanze delle due classi. La dipendenza è spesso temporanea e si verifica, ad esempio, quando un metodo di una classe utilizza un oggetto di un'altra classe come parametro o variabile locale.

- **Simbolo:** Una linea tratteggiata con una freccia aperta che punta verso la classe da cui si dipende.
- **Esempio:** La classe ServizioStampa ha un metodo stampaDocumento(Documento doc). In questo caso, ServizioStampa dipende da Documento perché utilizza oggetti di tipo Documento come parametro, ma non "ha" permanentemente un Documento.

```
@startuml
class ServizioStampa {
    + stampaDocumento(doc: Documento)
}
class Documento

ServizioStampa ..> Documento : usa
@enduml
```

Questi sono i tipi di relazione fondamentali che troverai nei diagrammi delle classi UML. Comprendere queste relazioni è cruciale per modellare accuratamente la struttura di un sistema software.

6. L'applicazione online "draw.io" per creare il diagramma delle classi

Draw.io è un'applicazione online gratuita per creare diagrammi di ogni tipo: organigrammi, diagrammi di flusso, diagrammi UML, diagrammi E/R, logici e di rete, planimetrie e molto altro. La sua caratteristica principale è che i tuoi dati rimangono nel tuo spazio di archiviazione (Google Drive, OneDrive, Dropbox, ecc.) o sul tuo dispositivo, offrendoti il pieno controllo sulla privacy. È facile da usare grazie all'interfaccia drag-and-drop e offre molte opzioni di personalizzazione e integrazioni con altre piattaforme come Confluence, Jira e Microsoft Teams. In poche parole, è uno strumento versatile e gratuito per visualizzare idee e processi.