



Università degli Studi della Campania
Luigi Vanvitelli
Dipartimento di Ingegneria

Programmazione ad Oggetti

a.a. 2020-2021

Introduzione alla OOP

Docente: Prof. Massimo Ficco
E-mail: massimo.ficco@unicampania.it

1

1

Ciclo di Vita del Software (CVS):

Un modello del ciclo di vita del software (CVS) è una caratterizzazione descrittiva o prescrittiva di come un sistema software viene o dovrebbe essere sviluppato

(W. Scacchi - Encyclopedia of Software Engineering Vol. II pag. 860)



2

Fasi di un CVS: una vista di alto livello V:

Definizione: si occupa del **cosa**.

Determinazione dei requisiti, informazioni da elaborare, funzioni e prestazioni attese, comportamento del sistema, interfacce, vincoli progettuali, criteri di validazione.

Sviluppo: si occupa del **come**

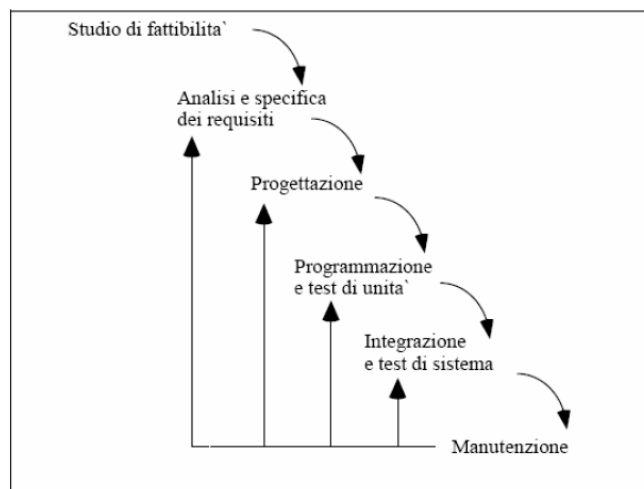
Definizione del progetto, dell'architettura software, della strutturazione dei dati e delle interfacce e dei dettagli procedurali; traduzione del progetto nel linguaggio di programmazione; collaudi.

Manutenzione: si occupa delle **modifiche**

correzioni, adattamenti, miglioramenti, prevenzione.



Modello a cascata V:



Modello a cascata

V:

Studio di fattibilità: Valutazione **Costi/Benefici**

Risorse finanziarie e umane

Soluzioni alternative

Tempi e modalità di sviluppo

Analisi e specifica dei requisiti: Valutazione **Requisiti Funzionali**

Produzione di un Documento di Specifica dei Requisiti (DSR)

Piano di Test di Sistema (PTS)

Progettazione: **Architettura generale** (hardware e software)

Definizione modulare del software e delle funzionalità associate

Produzione Documento di Progetto (DSP)



Modello a cascata

V:

Fase di Test

Test di Unità

Test di Integrazione

Test di Sistema

- *Alfa test*
- *Beta test*

Manutenzione

Correttiva

Adattativa

Perfettiva



V:

Fase di Sviluppo



Programmazione Orientata agli Oggetti (OOP)



L'Astrazione

V:

L'**astrazione** è il processo che porta ad estrarre le proprietà rilevanti di un'entità, ignorando i dettagli inessenziali

Le proprietà estratte definiscono una **vista** dell'entità
Una stessa entità può dar luogo a viste diverse

Esempio: un'automobile

- vista dal venditore:
prezzo, durata della garanzia, colore, ...
- vista dal meccanico:
tipo di motore, cilindrata, tipo di olio, ...



Meccanismi di astrazione (1/2)V:

Nella progettazione di un sistema software è opportuno adoperare delle tecniche di astrazione per dominare la complessità del sistema da realizzare.

I meccanismi di astrazione più diffusi sono:

- **ASTRAZIONE SUL CONTROLLO** (o funzionale)
- **ASTRAZIONE SUI DATI**



Meccanismi di astrazione (2/2)V:

ASTRAZIONE SUL CONTROLLO (o funzionale)

- Consiste nell'astrarre una data **funzionalità** dai dettagli della sua implementazione;
- E' ben supportata dai linguaggi di programmazione tradizionali tramite il concetto di **sottoprogramma**.

ASTRAZIONE SUI DATI

- Consiste nell'astrarre le **entità** (oggetti) costituenti il sistema, descritte in termini di una struttura dati e delle operazioni possibili su di essa;
- Può essere realizzata con un uso opportuno delle tecniche di **programmazione modulare** nei linguaggi tradizionali;
- E' supportata da appositi costrutti nei linguaggi di **programmazione ad oggetti**.

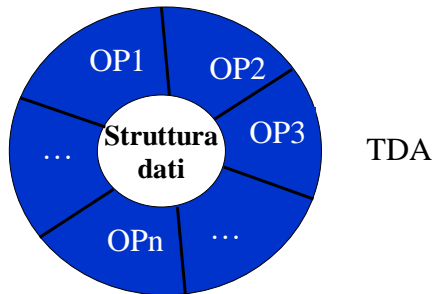


Tipi di dati astratti (1/2)

V:

Il concetto di **tipo** di dato in un linguaggio di programmazione tradizionale è quello di insieme dei valori che può assumere un dato (una variabile).

Il **tipo di dati astratto** (TDA) estende questa definizione, includendo anche l'insieme di **tutte e sole le operazioni** possibili su dati di quel tipo. La **struttura dati "concreta"** è **incapsulata** nelle operazioni su di essa definite.



TDA (2/2)

V:

Non è possibile accedere alla struttura dati incapsulata (né in lettura né in scrittura) se non attraverso le operazioni definite su di essa

Esempio:



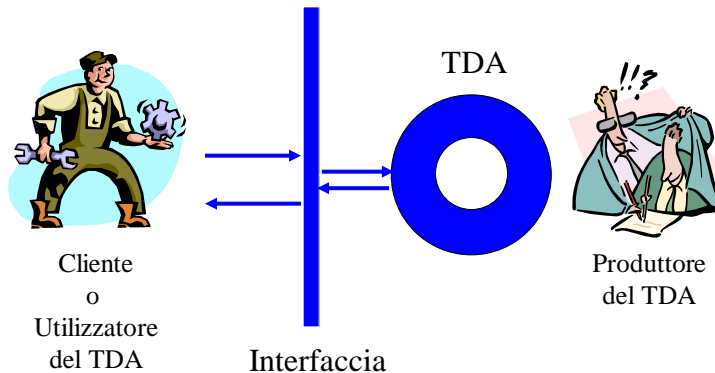
Un vantaggio: la struttura dati interna non può venire alterata da operazioni scorrette da parte dell'utente, in quanto ad essa si accede solo tramite le operazioni previste e realizzate dal produttore.



Interfaccia, uso e realizzazione V:

Interfaccia: specifica del TDA, descrive la parte direttamente accessibile dall'utilizzatore

Realizzazione: implementazione del TDA



Produttore e utilizzatore V:

Il **cliente o utilizzatore** fa uso del TDA per realizzare procedure di un'applicazione, o per costruire TDA più complessi

Il **produttore** realizza le astrazioni e le funzionalità previste per il dato

Un produttore di un TDA può essere utilizzatore di un altro TDA

Una modifica nella sola realizzazione del TDA non influenza i moduli che ne fanno uso (in quanto non cambia l'interfaccia)



Programmazione Modulare V:

La **modularità** è l'organizzazione in parti (per moduli) di un sistema, in modo che esso risulti più semplice da comprendere e manipolare

Gran parte dei sistemi complessi sono modulari

Esempio: Un'automobile è suddivisa in più sottosistemi:

- Motore
- Trasmissione
- ...



Il concetto di modulo V:

Un modulo di un sistema software è un componente che:

- ▶ Realizza una astrazione
- ▶ È dotato di una chiara separazione tra:
 - **Interfaccia**
 - **Corpo**

L'**interfaccia** specifica "**cosa**" fa il modulo (l'astrazione realizzata) e "**come**" si utilizza.

Il **corpo** descrive il "**come**" l'astrazione è realizzata

Modulo

Interfaccia

(Visibile dall'esterno)

Corpo

(Nascosto all'esterno e protetto)



Incapsulamento e *information hiding* V:

L'**incapsulamento** consiste nel nascondere e proteggere alcune informazioni di un'entità

► L'accesso in maniera controllata alle informazioni nascoste è possibile grazie ad un insieme di operazioni descritte dall'interfaccia

- Se l'interfaccia non cambia, le informazioni nascoste possono essere modificate senza che questo influisca sulle altre parti del sistema di cui l'entità fa parte

► **Esempio: un'autoradio**

- L'interfaccia consiste dei controlli e dei connettori tramite i quali è collegata all'automobile
- I dettagli di come funziona sono nascosti
- Per installarla e usarla non è necessario conoscere alcunché della sua struttura interna



Programmazione ad Oggetti V:

La programmazione ad oggetti rappresenta un ulteriore sviluppo rispetto alla programmazione modulare.

Nella OOP esiste un nuovo tipo di dato, la **classe**, che rappresenta un implementazione di una astrazione sui dati.

Questo tipo di dato serve a modellare un insieme di oggetti dello stesso tipo.

Un **oggetto** è caratterizzato da un insieme di attributi e un insieme di funzionalità (metodi) che operano sugli attributi dell'oggetto stesso.

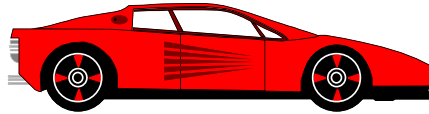


Un esempio di oggetto

V:

Funzioni Dati

- Avviati - Targa
- Fermati - Colore
- Accelera - Cilindrata motore
- ... - ...



Il conducente interagisce con una interfaccia per effettuare le operazioni consentite sull'automobile:

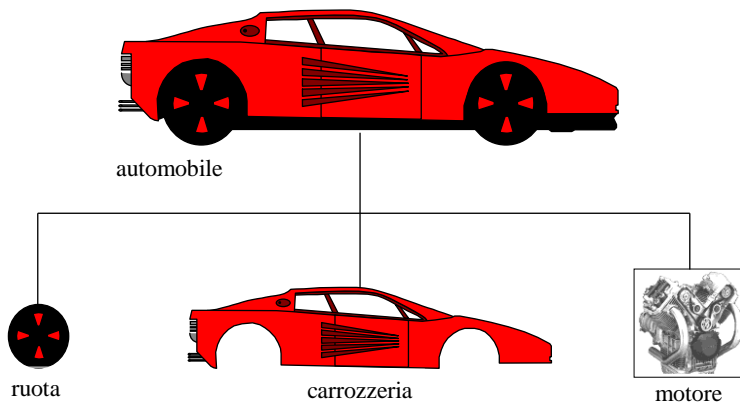
- Pedale del freno
- Pedale dell'acceleratore
- Leva del cambio
- Sistema di accensione
- ...



*Oggetti composti

V:

Un oggetto complesso può essere composto di oggetti più semplici detti componenti

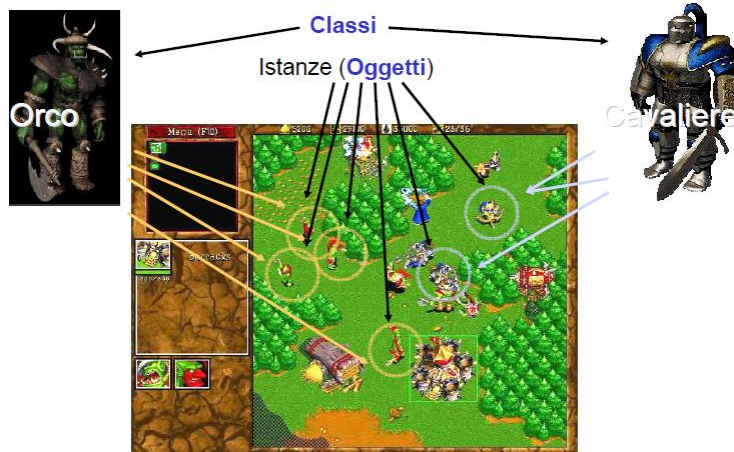


Programmazione orientata agli oggetti V:

La programmazione orientata agli oggetti (Object Oriented Programming – OOP) è un paradigma di programmazione, in cui un programma viene visto come un insieme di oggetti che interagiscono tra loro.



Classi e Oggetti V:



Vantaggi della programmazione OO:

Rispetto alla programmazione tradizionale, la programmazione orientata agli oggetti (OOP) offre vantaggi in termini di:

- **modularità**: le classi sono i moduli del sistema software;
- **information hiding**: sia le strutture dati che gli algoritmi possono essere nascosti alla visibilità dall'esterno di un oggetto;
- **coesione dei moduli**: una classe è un componente software ben coeso in quanto rappresentazione di una unica entità;
- **disaccoppiamento dei moduli**: gli oggetti hanno un alto grado di disaccoppiamento in quanto i metodi operano sulla struttura dati interna ad un oggetto;
- **riuso**: l'ereditarietà consente di riusare la definizione di una classe nel definire nuove (sotto)classi; inoltre è possibile costruire librerie di classi raggruppate per tipologia di applicazioni;
- **estensibilità**: il polimorfismo agevola l'aggiunta di nuove funzionalità, minimizzando le modifiche necessarie al sistema esistente quando si vuole estenderlo.

