Appunti Completi di Informatica

Un'Analisi Approfondita per lo Studio

Elaborazione e Ampliamento

14 giugno 2025

Indice

1 Basi di Dati e SQL (Espansione Approfondita)

1.1 Normalizzazione: L'Arte di Organizzare i Dati

La normalizzazione non è un mero esercizio accademico, ma la spina dorsale di un database sano. Il suo scopo è la creazione di uno schema che sia **resistente alle anomalie**, ovvero a comportamenti inattesi e indesiderati che si verificano durante la manipolazione dei dati.

Le Anomalie di Gestione dei Dati

Prima di analizzare le forme normali, è cruciale capire i problemi che risolvono. Esistono tre tipi principali di anomalie in tabelle non normalizzate:

Anomalia di Inserimento (Insertion Anomaly) Impossibilità di inserire un dato perché ne manca un altro. **Esempio:** In una tabella 'PROGETTI$_IMPIEGATI(IDProgetto, Nome$

Anomalia di Cancellazione (Deletion Anomaly) La cancellazione di un record causa la perdita involontaria di altre informazioni. **Esempio:** Nella stessa tabella 'PROGETTI$_IMPIEGATI$', s

Anomalia di Aggiornamento (Update Anomaly) Per modificare un'informazione, è necessario aggiornare più record, con il rischio di creare inconsistenze. **Esempio:** Se il 'NomeProgetto' "Alpha" deve essere cambiato in "Beta", e a tale progetto lavorano 10 impiegati, dobbiamo aggiornare 10 record diversi. Se l'operazione si interrompe a metà, avremo alcuni record con "Alpha" e altri con "Beta", corrompendo la coerenza dei dati.

Le forme normali sono una serie di regole progressive per eliminare queste anomalie.

1.1.1 Prima Forma Normale (1NF)

Definizione: Una tabella è in 1NF se tutti i suoi attributi sono **atomici**. **Obiettivo Profondo:** Garantire che ogni "cella" della tabella contenga un singolo valore, non una lista o un oggetto complesso. Questo rende i dati interrogabili in modo strutturato tramite SQL, senza dover ricorrere a parsing di stringhe.

Analogia della 1NF: Il Foglio di Calcolo

Immagina un foglio di calcolo dove in una cella "Contatti" scrivi "Mario (333111), Luigi (333222)". Questo è un disastro per l'analisi dei dati. Non puoi ordinare per numero di telefono, né contare quanti contatti ha una persona. La 1NF impone di creare una riga separata per ogni contatto: una riga per Mario e una per Luigi. I dati diventano strutturati e utilizzabili.

1.1.2 Seconda Forma Normale (2NF)

Prerequisito: La tabella deve essere in 1NF. **Definizione:** Ogni attributo non-chiave deve dipendere funzionalmente dall'**INTERA chiave primaria**. Questa regola si applica solo a tabelle con **chiave primaria composta**. **Obiettivo Profondo:** Eliminare le dipendenze parziali, che sono la causa principale di ridondanza e anomalie di aggiornamento in tabelle con chiavi composite.

Esempio Approfondito 2NF

Tabella NON in 2NF: Consideriamo una tabella per tracciare le ore lavorate dagli impiegati sui progetti. La chiave primaria è '(IDImpiegato, IDProgetto)'.
`REGISTRO_ORE(PK(IDImpiegato, IDProgetto), OreLavorate, NomeImpiegato, SedeProgetto)`

Problema (Dipendenze Parziali):

- 'NomeImpiegato' dipende solo da 'IDImpiegato'.
- 'SedeProgetto' dipende solo da 'IDProgetto'.

Questa struttura genera ridondanza massiccia: il nome di un impiegato e la sede di un progetto vengono ripetuti per ogni singola registrazione di ore.

Soluzione (Decomposizione in 2NF): Si isolano le dipendenze parziali in tabelle separate.

```
REGISTRO_ORE(PK(IDImpiegato, IDProgetto), OreLavorate)
IMPIEGATI(PK(IDImpiegato), NomeImpiegato)
PROGETTI(PK(IDProgetto), SedeProgetto)
```

Ora ogni informazione è memorizzata una sola volta, nel posto giusto.

1.1.3 Terza Forma Normale (3NF)

Prerequisito: La tabella deve essere in 2NF. **Definizione:** Nessun attributo non-chiave può dipendere da un altro attributo non-chiave. In altre parole, non devono esistere **dipendenze transitive**. **Obiettivo Profondo:** Rimuovere le dipendenze "indirette" dalla chiave primaria. Se un'informazione non descrive direttamente l'entità identificata dalla chiave, allora appartiene a un'altra tabella.

Esempio Approfondito 3NF

Tabella NON in 3NF: Consideriamo una tabella di impiegati e i dipartimenti in cui lavorano.
`IMPIEGATI(PK(IDImpiegato), Nome, IDDipartimento, NomeDipartimento, ManagerDipartime`

Problema (Dipendenza Transitiva): 'IDImpiegato' → 'IDDipartimento' (dipendenza diretta, corretta). Ma anche 'IDDipartimento' → 'NomeDipartimento' e 'IDDipartimento' → 'ManagerDipartimento'. Di conseguenza, 'NomeDipartimento' e 'ManagerDipartimento' dipendono transitivamente dalla chiave 'IDImpiegato'. Se il manager di un dipartimento cambia, bisogna aggiornare i record di TUTTI gli impiegati di quel dipartimento.

Soluzione (Decomposizione in 3NF):

```
IMPIEGATI(PK(IDImpiegato), Nome, FK(IDDipartimento))
DIPARTIMENTI(PK(IDDipartimento), NomeDipartimento, ManagerDipartimento)
```

Oltre la 3NF: La Forma Normale di Boyce-Codd (BCNF)

La BCNF è una versione leggermente più forte della 3NF. **Definizione:** Una tabella è in BCNF se, per ogni dipendenza funzionale non banale $X \rightarrow Y$, X è una **superchiave** (cioè, una chiave candidata). **In parole semplici:** Ogni "determinante" (la parte sinistra di una dipendenza) deve essere una chiave candidata. Risolve rare anomalie che la 3NF può ancora consentire, specialmente in tabelle con chiavi candidate multiple e sovrapposte.

1.2 Proprietà ACID: Il Patto di Affidabilità delle Transazioni

ACID è l'acronimo che definisce le garanzie fondamentali fornite da un sistema di gestione di database (DBMS) per assicurare che le transazioni siano processate in modo affidabile, anche in caso di errori, crash di sistema o accessi concorrenti.

Proprietà	**Definizione Semplice**	**Spiegazione**
(A)TOMicità	O tutto o niente.	Una transazione è un'unità logica d
(C)ONSISTENZA	Il database rimane valido.	La consistenza assicura che una transazione
(I)SOLAMENTO	Le transazioni non si disturbano.	L'isolamento garantisce che le transazioni concorrenti
(D)URABILITÀ	I risultati sono permanenti.	Una volta che una transazione ha ricevuto

2 Sicurezza Informatica (Espansione Approfondita)

La sicurezza informatica non è un optional, ma un requisito fondamentale nella progettazione di qualsiasi applicazione. Ignorarla equivale a costruire una casa senza porte né serrature. Analizziamo due delle vulnerabilità web più comuni e devastanti.

2.1 SQL Injection (SQLi): Corrompere le Interrogazioni

Definizione: Una vulnerabilità che permette a un attaccante di interferire con le query che un'applicazione fa al suo database. Si verifica quando l'input dell'utente viene inserito in una query SQL in modo insicuro, permettendo all'attaccante di modificarne la struttura e la logica.

Tipologie di SQL Injection

In-band (Classic) L'attacco più diretto. L'attaccante usa lo stesso canale di comunicazione per lanciare l'attacco e ottenere i risultati. L'esempio del login bypass (‘’ OR ’1’=’1’‘) rientra in questa categoria. Un'altra tecnica comune è l'attacco ‘UNION‘, che permette di "cucire" i risultati di una query malevola a quelli di una query legittima per estrarre dati da altre tabelle.

Inferential (Blind) Un tipo di attacco più sofisticato, usato quando l'applicazione non mostra direttamente i risultati di un errore SQL. L'attaccante "interroga" il database con una serie di domande vero/falso e deduce le informazioni osservando la risposta dell'applicazione (es. una pagina carica normalmente o dà un errore generico).

Esempio di Blind SQLi (Time-based)

L'attaccante inietta un comando che dice al database di "aspettare" se una condizione è vera.

```
' AND IF(SUBSTRING(user(),1,1) = 'r', SLEEP(5), 0) --

```

Se la pagina impiega 5 secondi a caricare, l'attaccante sa che il primo carattere del nome utente del DB è 'r'. Ripetendo il processo, può estrarre informazioni carattere per carattere.

Out-of-band La tecnica più avanzata, usata quando l'applicazione è molto restrittiva. L'attaccante costringe il database a inviare i dati a un server esterno sotto il suo controllo (es. tramite richieste DNS o HTTP).

Impatto di un Attacco SQLi

Le conseguenze possono essere catastrofiche:

- **Confidentiality Breach:** Estrazione di dati sensibili (credenziali, dati personali, segreti aziendali).
- **Integrity Breach:** Modifica o cancellazione di dati.

- **Authentication Bypass:** Accesso a sistemi protetti senza credenziali valide.
- **Denial of Service (DoS):** Esecuzione di query pesanti che bloccano il database.

Contromisure: La Difesa a Strati

Prepared Statements (Query Parametriche) La difesa d'elezione. Separa nettamente il codice SQL (la struttura della query) dai dati (i parametri). Il database tratta i parametri sempre e solo come dati, mai come codice eseguibile, rendendo l'iniezione strutturalmente impossibile.

Object-Relational Mappers (ORM) Librerie come Doctrine (PHP), Hibernate (Java) o SQLAlchemy (Python) astraono l'interazione con il database e, se usate correttamente, generano query sicure basate su prepared statements.

Principio del Minimo Privilegio L'account del database usato dall'applicazione web dovrebbe avere solo i permessi strettamente necessari. Ad esempio, non dovrebbe avere il permesso di cancellare tabelle ('DROP TABLE').

Input Validation Verificare sempre che l'input ricevuto corrisponda al formato atteso (un ID utente deve essere un numero, un'email deve avere un formato valido, etc.). È una buona pratica ma **non sostituisce** i prepared statements.

2.2 Cross-Site Scripting (XSS): Iniettare Script nel Browser

Definizione: Una vulnerabilità che permette a un attaccante di iniettare script malevoli (solitamente JavaScript) nelle pagine web visualizzate da altri utenti. A differenza dell'SQLi, l'obiettivo non è il server, ma il **browser della vittima**.

Analogia dell'XSS: Il Graffito sul Muro

Immagina un muro pubblico (una pagina web) dove tutti possono lasciare messaggi (commenti). Un utente normale scrive "Ciao a tutti". Un attaccante, invece di scrivere un testo, disegna un'istruzione pericolosa sul muro: "Chiunque legga questo, mi dia il suo portafoglio". L'XSS è l'equivalente digitale: lo script malevolo viene "disegnato" sulla pagina e il browser di ogni visitatore lo esegue, consegnando all'attaccante i "portafogli" digitali (i cookie di sessione).

Tipologie di XSS

Stored (Persistent) XSS Lo script malevolo viene memorizzato permanentemente sul server (es. in un post di un forum, in un profilo utente) e viene servito a ogni utente che visita quella pagina. È il tipo più pericoloso per l'ampio raggio d'azione.

Reflected (Non-Persistent) XSS Lo script malevolo è parte di una richiesta (tipicamente nell'URL) e viene "riflesso" dal server nella risposta. L'attaccante deve ingannare la vittima affinché clicchi su un link malevolo appositamente costruito (es. tramite email di phishing).

Esempio di Reflected XSS

Un sito con una ricerca vulnerabile: ‘https://sito.com/search?q=Ciao‘ L’attaccante crea un link: ‘https://sito.com/search?q=<script>alert(’XSS!’)</script>‘ Se la pagina mostra "Risultati per: [q]", il browser della vittima eseguirà lo script.

DOM-based XSS Una variante moderna dove la vulnerabilità è interamente sul lato client. Uno script legittimo sulla pagina prende dati dall’URL (es. ‘document.location.hash‘) e li usa per modificare il DOM della pagina in modo insicuro, eseguendo di fatto il codice malevolo.

Impatto di un Attacco XSS

Eseguendo codice nel browser della vittima, un attaccante può:

- **Session Hijacking:** Rubare i cookie di sessione della vittima per impersonarla.
- **Phishing e Credential Theft:** Modificare la pagina per mostrare un finto form di login e rubare le credenziali.
- **Keylogging:** Registrare tutto ciò che la vittima digita.
- **Defacement:** Modificare l’aspetto del sito web per l’utente.

Contromisure Fondamentali

Output Encoding La difesa primaria. Prima di inserire qualsiasi dato di origine utente nell’HTML, codificare i caratteri speciali in entità HTML (‘<‘ diventa ‘<‘, ‘>‘ diventa ‘>‘, etc.). Questo fa sì che il browser li visualizzi come testo innocuo invece di interpretarli come codice. Tutte le librerie di templating moderne (es. Twig, Blade, React) lo fanno automaticamente.

Content Security Policy (CSP) Un potente header HTTP che agisce come un firewall nel browser. Permette di definire una whitelist di sorgenti attendibili da cui è possibile caricare ed eseguire risorse (script, stili, immagini).

```
1 Content-Security-Policy: default-src 'self'; script-src 'self'
    https://apis.google.com;
2
```

Listing 1: Esempio di header CSP restrittivo

Questo header dice al browser: "Carica tutto solo dal mio dominio (’self’), tranne gli script, che puoi caricare anche da apis.google.com". Blocca tutti gli script inline e da altre fonti.

3 Sviluppo Web e Architetture (Espansione Approfondita)

3.1 Il Ciclo Richiesta-Risposta: Il Dialogo Fondamentale del Web

TUTTA l'interazione sul web si basa su un modello semplice ma potente: il **ciclo richiesta-risposta** (Request-Response Cycle) mediato dal protocollo HTTP. Capire questo flusso è essenziale per comprendere la distinzione tra client e server.

Analogia del Ciclo: Ordinare al Ristorante

1. **Il Cliente (Client/Browser):** Sei tu al tavolo. Decidi cosa vuoi dal menù (digiti un URL o clicchi un link).
2. **La Richiesta (HTTP Request):** Chiami il cameriere (la rete) e gli comunichi la tua ordinazione (la richiesta HTTP, che dice "GET /pagina.html").
3. **Il Cameriere e la Cucina (Server):** Il cameriere porta l'ordinazione in cucina (il server web). La cucina (il backend) prepara il piatto: recupera gli ingredienti (interroga il database), li cucina (applica la logica di business) e impiattà il risultato (costruisce la pagina HTML).
4. **La Risposta (HTTP Response):** Il cameriere ti porta il piatto pronto (la risposta HTTP, che contiene il codice HTML della pagina, le immagini, etc.).
5. **La Visualizzazione (Rendering):** Mangi il piatto (il browser interpreta l'HTML e visualizza la pagina).

I due attori di questo dialogo hanno ruoli e responsabilità nettamente distinti:

Caratteristica	Client (Frontend)	Server (Backend)
Dove viene eseguito?	Sul computer dell'utente, all'interno del **browser web** (es. Chrome, Firefox).	Su un computer remoto potente, chiamato **web server**.
Ruolo Principale	Gestire l'**interfaccia utente (UI)** e l'**esperienza utente (UX)**. Mostrare i dati e catturare l'input dell'utente. È responsabile della presentazione.	Gestire la **logica di business**, l'accesso e la persistenza dei **dati**, e la **sicurezza** (autenticazione, autorizzazione). È responsabile della sostanza.
Tecnologie Tipiche	**HTML** (struttura), **CSS** (stile), **JavaScript** (interattività, chiamate API). Framework come React, Angular, Vue.js.	Linguaggi come **PHP, Python, Java, C#, Node.js**. Web server come **Apache, Nginx**. Database come **MySQL, PostgreSQL**.
Esempio di compito	Validare che un campo email contenga una `@` (validazione immediata). Mostrare un messaggio di errore senza ricaricare la pagina. Animare un menu a tendina. Gestire lo stato dell'UI.	Controllare se l'utente e la password esistono nel database (validazione definitiva). Salvare un nuovo post del blog. Elaborare un pagamento. Generare la pagina HTML da inviare al client.

3.2 Cookie vs. Sessioni: Dare una Memoria al Web Stateless

Il Problema: Il protocollo HTTP è **stateless** (senza stato). Ogni richiesta-risposta è un evento isolato. Il server non ha memoria della richiesta precedente. Come fa un sito a "ricordare" chi sei, che sei loggato o cosa hai messo nel carrello?

La "Stretta di Mano" della Sessione

Cookie e sessioni lavorano insieme per risolvere questo problema. La sessione è il "ricordo" sul server, e il cookie è il "gettone" che permette al client di farsi riconoscere.

Analogia della Sessione: Il Guardaroba

1. **Primo Accesso:** Entri in un teatro (il sito web) per la prima volta. Lasci il cappotto (i tuoi dati, es. chi sei, cosa metti nel carrello) al guardaroba (**il server crea una sessione**).
2. **Il Gettone:** L'addetto al guardaroba non si ricorderà la tua faccia. Ti dà un gettone numerato (**un cookie con l'ID di sessione**). Questo gettone non contiene il tuo cappotto, ma solo il numero per identificarlo.
3. **Visite Successive:** Ogni volta che interagisci con il teatro (fai una nuova richiesta), mostri il tuo gettone (il browser invia automaticamente il cookie). L'addetto usa il numero per trovare il tuo cappotto esatto nel guardaroba (il server usa l'ID per recuperare i dati della tua sessione).

Caratteristica	Cookie	Sessione
Luogo di memorizzazione	Sul **client** (browser dell'utente), come piccolo file di testo.	Sul **server** (in memoria, file o database). Sul client viene memorizzato solo un **ID di sessione** (solitamente in un cookie).
Contenuto	Piccole stringhe di testo (es. '$session_id = a1b2c3d4$', '$lingua = it$', '$remember_me = token$'). La dimensione è limitata (circa 4 KB).	Dati complessi e sensibili (es. l'oggetto utente completo, il contenuto del carrello come array). La dimensione è limitata solo dalle risorse del server.
Sicurezza	**Bassa.** I dati sono visibili e modificabili dall'utente. **MAI** memorizzare dati sensibili (password, permessi) direttamente in un cookie. Possono essere protetti con flag come 'HttpOnly' (per impedire l'accesso via JavaScript) e 'Secure' (per inviarli solo su HTTPS).	**Alta.** I dati veri e propri non lasciano mai il server. L'utente vede solo un ID casuale e privo di significato, che è inutile senza l'accesso al server.
Durata	Può essere **persistente** (impostando una data di scadenza) e sopravvivere alla chiusura del browser. Oppure può essere un "session cookie", che scade alla chiusura del browser.	**Temporanea.** Scade automaticamente dopo un periodo di inattività del server (es. 24 minuti) o alla chiusura del browser (se l'ID di sessione era in un session cookie).
Esempio d'uso	**Contenere l'ID di sessione**. Ricordare la scelta della lingua. Salvare un token "ricordami" per il login automatico. Tracciamento per l'analisi del traffico.	Gestire un carrello della spesa. Mantenere lo stato di login di un utente durante una singola visita. Memorizzare messaggi temporanei da mostrare all'utente (es. "Login effettuato con successo").

4 OOP e Sistemi Operativi (Espansione Approfondita)

4.1 Pilastri della Programmazione a Oggetti (OOP)

La programmazione a oggetti non è solo un insieme di funzionalità linguistiche, ma un **paradigma di programmazione** progettato per gestire la complessità dei sistemi software moderni. L'idea centrale è di modellare il software come una collezione di **oggetti** interagenti, proprio come il mondo reale è composto da oggetti. Questo approccio si contrappone a quello procedurale, che vede un programma come una sequenza di istruzioni.

4.1.1 Incapsulamento: La Cassaforte Digitale

Definizione: Legare strettamente i dati (attributi) con le operazioni (metodi) che li manipolano, e nascondere i dettagli implementativi interni di un oggetto al mondo esterno. **Obiettivo Profondo:** Creare componenti software autonomi e protetti, simili a "scatole nere". Questo riduce la complessità e aumenta la robustezza del sistema.

Analogia dell'Incapsulamento: Il Televisore

Interagisci con il tuo televisore tramite un'interfaccia pubblica ben definita: il telecomando (con i metodi 'accendi()', 'cambiaCanale()', 'alzaVolume()'). Non sai, e non ti interessa, come i circuiti interni (dati e metodi privati) gestiscono i segnali per produrre l'immagine. Questo incapsulamento ti permette di usare il televisore senza rischiare di danneggiarlo e permette al produttore di cambiare completamente i circuiti interni in un nuovo modello senza che tu debba cambiare il modo in cui usi il telecomando.

Benefici Chiave:

- **Integrità dei Dati:** L'oggetto ha il pieno controllo del proprio stato. I dati privati possono essere modificati solo tramite metodi pubblici, che possono contenere logica di validazione (es. 'setEta()' può impedire l'inserimento di un'età negativa).
- **Manutenibilità e Flessibilità:** L'implementazione interna di un oggetto può essere modificata radicalmente senza impattare le altre parti del sistema che lo utilizzano, a patto che la sua interfaccia pubblica rimanga invariata.

4.1.2 Ereditarietà: Il DNA del Codice

Definizione: Meccanismo che permette a una nuova classe (sottoclasse o classe figlia) di acquisire (ereditare) attributi e metodi da una classe esistente (superclasse o classe madre). **Obiettivo Profondo:** Stabilire una relazione gerarchica di tipo "**is-a**" ("è un") per promuovere il riuso del codice e creare modelli concettuali chiari.

Pro e Contro:

- **Vantaggio (Riuso del Codice):** Il codice comune viene scritto una sola volta nella superclasse e riutilizzato da tutte le sottoclassi. Una gerarchia 'Veicolo' →

'VeicoloAMotore' → 'Automobile' permette ad 'Automobile' di ereditare tutte le proprietà e i comportamenti di un veicolo a motore e di un veicolo generico.

- **Svantaggio (Accoppiamento Forte):** La sottoclasse è strettamente legata alla sua superclasse. Una modifica alla superclasse può "rompere" inaspettatamente le funzionalità delle sottoclassi. Per questo, in molti design moderni, si preferisce la *composizione* all'ereditarietà ("has-a" vs "is-a").

Esempio di Gerarchia con Ereditarietà

```
// Superclasse base
abstract class Veicolo {
    protected int ruote;
    public void muovi() { System.out.println("Il veicolo si muove
    ...."); }
}

// Sottoclasse che aggiunge una caratteristica
class VeicoloAMotore extends Veicolo {
    private Motore motore;
    public void accendiMotore() { /* ... */ }
}

// Sottoclasse specializzata
class Automobile extends VeicoloAMotore {
    public Automobile() { this.ruote = 4; }
    public void apriPortiera() { /* ... */ }
}
```

4.1.3 Polimorfismo: Una Interfaccia, Molte Forme

Definizione: Dal greco "molte forme". È la capacità di un'interfaccia di essere implementata da più tipi di oggetti, permettendo di trattare oggetti di classi diverse in modo uniforme. **Obiettivo Profondo:** Scrivere codice generico, flessibile e disaccoppiato, in grado di operare su un'intera famiglia di oggetti senza conoscerne il tipo specifico.

Esistono due tipi principali di polimorfismo:

Polimorfismo Statico (Compile-time): Risolto a tempo di compilazione. L'esempio più comune è l' **overloading** di metodi: definire più metodi con lo stesso nome ma con una firma diversa (numero o tipo di parametri).

```
public class Calcolatrice {
    public int somma(int a, int b) { return a + b; }
    public double somma(double a, double b) { return a + b; }
}

```

Listing 2: Esempio di Overloading

Polimorfismo Dinamico (Run-time): Risolto a tempo di esecuzione. L'esempio cardine è l' **override** di metodi: una sottoclasse fornisce un'implementazione specifica per un metodo già definito nella sua superclasse.

La Potenza del Polimorfismo Dinamico

Immaginiamo di dover disegnare diverse forme geometriche.

```
// Interfaccia comune
interface Forma {
    void disegna();
}

// Implementazioni concrete
class Cerchio implements Forma {
    @Override public void disegna() { System.out.println("Disegno un cerchio: O"); }
}
class Quadrato implements Forma {
    @Override public void disegna() { System.out.println("Disegno un quadrato: []"); }
}

// Codice client POLIMORFICO
public class Disegnatore {
    public void disegnaTutto(List<Forma> forme) {
        // Il codice qui non sa ne' gli importa se sta disegnando
        // un Cerchio, un Quadrato o un futuro Triangolo.
        // Chiama semplicemente il metodo disegna() sull'oggetto Forma.
        for (Forma f : forme) {
            f.disegna(); // Binding dinamico: Java sceglie il metodo giusto a runtime!
        }
    }
}
```

Questo codice è estensibile: possiamo aggiungere una nuova classe ‘Triangolo‘ senza modificare una sola riga della classe ‘Disegnatore‘.

4.2 Processi vs. Thread: L'Anima della Concorrenza

Nei sistemi operativi moderni, la capacità di eseguire più compiti "contemporaneamente" è fondamentale. Processi e thread sono i due modelli principali per raggiungere questo obiettivo.

Analogia della Concorrenza: La Cucina del Ristorante

- **Un Processo** è come un'intera **cucina di un ristorante**. Ha il suo spazio fisico (memoria), i suoi utensili (risorse), le sue ricette e i suoi cuochi. È un'unità autonoma e isolata. Avere due processi (es. Chrome e Word) è come avere due ristoranti separati, ognuno nella sua palazzina.
- **Un Thread** è come un **cuoco all'interno di una cucina**. Un ristorante (processo) può avere più cuochi (thread) che lavorano insieme. Condividono la stessa cucina, gli stessi fornelli e gli stessi ingredienti (memoria e risorse condivise). Possono collaborare per preparare un pasto complesso più velocemente.

Caratteristica	Processo	Thread (Lightweight Process)
Definizione	Un programma in esecuzione, con il suo stato e le sue risorse.	Un'unità di esecuzione *all'interno* di un processo.
Memoria	Ogni processo ha uno **spazio di memoria privato e isolato**. Il sistema operativo protegge un processo dalla memoria di un altro.	Tutti i thread di un processo **condividono lo stesso spazio di memoria** (heap, dati globali). Hanno solo uno stack e registri privati.
Creazione e Terminazione	**Costosa.** Richiede un intervento significativo del sistema operativo per allocare e deallocare le strutture di memoria.	**Economica.** È molto più veloce perché non richiede la duplicazione dello spazio di memoria del processo.
Comunicazione	**Complessa e lenta.** Richiede meccanismi espliciti di IPC (Inter-Process Communication) gestiti dal SO, come pipe, socket, o memoria condivisa mappata.	**Semplice e veloce.** Avviene naturalmente tramite variabili condivise in memoria. Questa facilità è anche una fonte di pericoli.
Isolamento e Robustezza	**Alto isolamento.** Un crash in un processo non influisce sugli altri (es. un'app che si blocca non fa crashare il sistema operativo).	**Basso isolamento.** Se un thread causa un errore fatale (es. accesso a memoria non valida), l'intero processo (e tutti i suoi thread) viene terminato.
Context Switching	**Lento.** Il SO deve salvare l'intero stato del processo (mappe di memoria, file aperti, etc.) e caricarne un altro.	**Veloce.** Il SO deve salvare e ripristinare solo i registri e lo stack del thread, poiché lo spazio di memoria rimane lo stesso.
La Sfida della Sincronizzazione	Non necessaria se non si usa IPC esplicita.	**Critica e complessa.** Poiché i thread condividono la memoria, è necessario usare meccanismi di sincronizzazione (es. *mutex, semafori, lock*)