OOP

Created	@January 21, 2025 5:49 PM
Class	Prog2

La programmazione orientata agli oggetti (OOP) si basa su quattro principi fondamentali: **Incapsulamento**, **Ereditarietà**, **Polimorfismo** e **Astrazione**. Questi principi consentono di modellare il software in modo modulare, flessibile e manutenibile. Di seguito vengono approfonditi ciascuno di essi con spiegazioni dettagliate e best practices applicabili al linguaggio C++.

---

1. Incapsulamento

Definizione

L'incapsulamento è il principio che prevede la segregazione tra dati e comportamenti di un oggetto. Permette di nascondere i dettagli interni e di esporre solo ciò che è necessario per l'utilizzo esterno. In C++, l'incapsulamento viene implementato utilizzando modificatori di accesso come `private`, `protected` e `public` per controllare la visibilità dei membri della classe.

Vantaggi

- **Protezione dei dati**: Previene modifiche non intenzionali o non autorizzate ai dati sensibili.
- **Manutenibilità**: I dettagli interni di una classe possono essere modificati senza influenzare il codice che utilizza quella classe.
- **Modularità**: Ogni classe diventa un'unità autonoma e ben definita.

Buone pratiche

- **Usare membri privati**: Proteggi i dati sensibili dichiarandoli come `private` e offri accesso controllato tramite metodi getter e setter.

- **Separazione dei ruoli**: Mantieni separate le responsabilità della logica di accesso ai dati da quella dell'elaborazione.
- **Utilizzo del modificatore **** `const`: Quando un metodo non deve modificare lo stato interno della classe, dichiaralo come `const` per aumentare la chiarezza e la sicurezza.

---

2. Ereditarietà

Definizione

L'ereditarietà permette di definire una nuova classe (classe derivata) basata su una classe esistente (classe base). La classe derivata eredita le proprietà e i metodi della classe base, permettendo il riutilizzo del codice e facilitando l'estensione delle funzionalità.

Tipi di ereditarietà in C++

- **Public**: I membri pubblici e protetti della classe base mantengono il loro livello di accesso nella classe derivata.
- **Protected**: I membri pubblici e protetti della classe base diventano protetti nella classe derivata.
- **Private**: Tutti i membri della classe base diventano privati nella classe derivata.

Vantaggi

- **Riutilizzo del codice**: Consente di centralizzare la logica comune nella classe base.
- **Modellazione logica**: Facilita la rappresentazione di relazioni "è un" (is-a) tra classi.

Buone pratiche

- **Utilizza ereditarietà solo per relazioni logiche**: Applicala quando la relazione tra classi è chiaramente di tipo "è un" (ad esempio, "un cane è un animale").
- **Evita alberi di ereditarietà troppo profondi**: Catene di ereditarietà lunghe possono complicare il debugging e la manutenibilità del codice.

- **Distruttori virtuali**: Quando lavori con ereditarietà polimorfica, dichiara un distruttore virtuale nella classe base per garantire una corretta gestione della memoria.

---

3. Polimorfismo

Definizione

Il polimorfismo permette a un oggetto di comportarsi in modi diversi in base al contesto. Si divide in due principali categorie:

- **Polimorfismo statico (compile-time)**: Implementato attraverso l'overloading di funzioni e operatori.
- **Polimorfismo dinamico (runtime)**: Implementato tramite funzioni virtuali e consente di scegliere il metodo appropriato in base al tipo effettivo dell'oggetto durante l'esecuzione.

Polimorfismo statico

Questo tipo di polimorfismo si ottiene definendo più funzioni con lo stesso nome ma con firme diverse. Viene risolto al momento della compilazione.

Polimorfismo dinamico

Si basa sull'uso di funzioni virtuali dichiarate nella classe base e sovrascritte nelle classi derivate. La selezione del metodo appropriato avviene al runtime in base al tipo effettivo dell'oggetto.

Vantaggi

- **Flessibilità**: Consente di scrivere codice che lavora con interfacce comuni, indipendentemente dall'implementazione specifica.
- **Estensibilità**: Le nuove classi derivate possono essere integrate facilmente nel sistema senza modificare il codice esistente.

Buone pratiche

- **Usa `override`**: Quando sovrascrivi un metodo della classe base, usa la parola chiave `override` per rendere il codice più chiaro ed evitare errori.

- **Polimorfismo responsabile**: Utilizza il polimorfismo dinamico solo quando strettamente necessario, poiché introduce un overhead di runtime.

---

4. Astrazione

Definizione

L'astrazione è il principio di rappresentare un oggetto o un concetto nascondendo i dettagli implementativi non essenziali. In C++, l'astrazione viene tipicamente realizzata tramite l'uso di classi astratte e interfacce, definite attraverso metodi puramente virtuali.

Vantaggi

- **Chiarezza**: Fornisce un modello concettuale chiaro di ciò che una classe rappresenta, separandolo dalla sua implementazione.
- **Flessibilità**: Consente di cambiare l'implementazione senza alterare il codice che usa l'interfaccia.

Buone pratiche

- **Definizione chiara dell'interfaccia**: Progetta le interfacce in modo che definiscano chiaramente il contratto che le classi concrete devono rispettare.
- **Evitare l'esposizione non necessaria**: Nascondi dettagli implementativi che non sono rilevanti per l'utente della classe.
- **Utilizzo mirato delle classi astratte**: Usa classi astratte solo quando più classi condividono un comportamento comune che può essere specificato in un contratto astratto.