Ereditarietà e polimorfismo

- Generalizzazione ed Ereditarietà
- Polimorfismo, tabella dei metodi virtuali (vtbl): virtual, override
- Classe astratta
- Classe e metodo final
- Distruttore Virtuale
- Ereditarietà multipla e virtuale
- Ereditarietà privata/protetta
- Codice di esempio
  - https://github.com/egalli64/corso-cpp folder b6

Generalizzazione ed Ereditarietà

- L'ereditarietà è un meccanismo chiave della programmazione Object Oriented
  - Determina una gerarchia di classi, basata su polimorfismo, implementato via funzioni virtuali
- Richiede un attento design dell'applicazione
  - Quali classi sono necessarie per descrivere il problema
  - Come interagiscono tra loro all'interno di una gerarchia o per mezzo di altre relazioni
- Tre tipi di derivazione: pubblica, protetta, privata
- La più comunemente usata è la derivazione pubblica
  - Definisce una relazione detta "**is-a**", la classe derivata è un sotto tipo della classe base
    - Tutto ciò che non è privato nella classe base è disponibile nella classe derivata
    - Tutto ciò che è pubblico nella classe base è disponibile a chi usa la classe derivata
- Ogni costruttore della classe derivata deve invocare un costruttore della classe base
  - Allo scopo di inizializzare la parte relativa dell'oggetto in creazione
  - Se non ne viene invocato uno esplicitamente, il compilatore assume l'uso del default ctor

Metodo virtuale – vtbl

- Un metodo virtuale è definito in una classe base e ridefinito in una classe figlia
  - Il metodo base deve essere indicato esplicitamente come **virtual**
  - Nella classe figlia possiamo definire un suo **override**, con esattamente la stessa signature
    - Concetto diverso dall'*overload*, stesso nome, diversa lista di parametri
- Solo metodi di istanza (non statici) possono essere virtuali
  - I ctor non possono essere virtuali
    - Se c'è bisogno di un virtual ctor, occorre implementarlo con un metodo apposito
    - Spesso chiamato clone(), ritorna un puntatore all'oggetto creato - vedi anche il design pattern Factory Method
  - I dtor possono essere virtuali, vedi più avanti
- Puntatori ai metodi virtuali sono inseriti in una tabella, detta **vtbl**
  - Permette di decidere dinamicamente, a run-time, quale metodo invocare
  - Meccanismo che determina un (piccolo) overhead in esecuzione
    - Va abilitato solo quando necessario

Polimorfismo

- Un metodo è virtuale se è identificato da "**virtual**" o se ridefinisce un metodo virtuale
- Si può segnalare che un metodo è un **override** alla fine della signature
- Da un override è possibile invocare la versione della classe base
  - Uso dell'operatore di risoluzione di scope ::
  - Non specificando la classe di appartenenza, si sottintende una chiamata ricorsiva
- Un **puntatore** a una classe base funziona in modo **polimorfico**
  - Può essere associato ad un oggetto di tutta gerarchia
  - L'invocazione di un metodo, a run-time, tiene conto del tipo effettivo dell'oggetto
- Un oggetto di classe derivata assegnato a un oggetto di classe base
  - Subisce il fenomeno dello "**slicing**", è trattato come un oggetto di classe base
  - È un motivo, oltre all’efficienza, per preferire i parametri by const reference

Astratto - finale

- Un metodo virtuale può essere puro
  - Non ha un body
  - Ha solo la dichiarazione, che termina con "= 0;"
  - Detto anche metodo astratto
- Una classe che ha almeno un metodo astratto è a sua volta astratta
  - Non si possono istanziare oggetti di quel tipo
- Ha senso solo in quanto classe base di una gerarchia
- Una classe che deriva da una classe astratta
  - È a sua volta astratta, se non implementa tutti i metodi astratti
  - Altrimenti è concreta, e può essere istanziata
- Si può impedire che una classe venga estesa, dichiarandola **final**
- Si può anche impedire l'override di un singolo metodo, sempre via final

Distruttore virtuale

- Se una classe è pensata per avere classi derivate
  - È *opportuno* che il suo distruttore sia dichiarato virtuale
- Se una classe ha metodi virtuali
  - È **necessario** che anche il distruttore sia virtuale
- Una classe che non è pensata per essere derivata andrebbe indicata **final**
  - E il suo distruttore **non deve** essere virtuale
- Se il dtor in una classe polimorfica non è virtuale
  - La distruzione di un oggetto di classe derivata ha un **comportamento indefinito**
  - In particolare, se l'oggetto è acceduto via puntatore alla classe base
    - Se il dtor non è virtuale normalmente i dtor delle classi derivate non sono invocati

Ereditarietà multipla e virtuale

- Multiple Inheritance (MI): una classe può avere più super-class
  - Es: un mezzo anfibio può avere due classi base, Autoveicolo e Barca
- Problema del diamante (Deadly Diamond of Death)
  - Classe base, due classi derivate
    - Una quarta classe deriva da entrambe le classi di livello intermedio
      - C'è un'istanza della classe madre in entrambe le istanze delle classi intermedie!
    - Se una classe figlia ha accesso a un membro di classi madri duplicato, quale va scelto?
- Una soluzione: accesso esplicito via operatore di risoluzione ::
- L'ereditarietà virtuale elimina la doppia istanza della classe di base
  - Le classi intermedie devono dichiarare di estendere virtualmente la classe base
  - La classe figlia deve chiamare i costruttori delle altre classi del diamante

Ereditarietà privata / protetta

- L'ereditarietà più comunemente usata è quella pubblica
  - Implica una relazione **IS-A** tra classe figlia e classe madre
- L'ereditarietà **privata**
  - È un modo usato in C++ per implementare la relazione **HAS-A**
  - La classe figlia HA al suo interno la classe madre, *non É* dello stesso tipo
- L'ereditarietà **protetta** è poco usata
  - Non è chiara dal punto di vista concettuale
  - In pratica, l'accesso alla classe madre è come per l'ereditarietà privata
    - In più, l'accesso è concesso anche alle classi che derivano dalla figlia