OOP in C++

Alberto Ferrari

Classi e Oggetti

- Una **classe** è un tipo che ha variabili membro (dati membro) e funzioni membro
- *In Java i dati membro sono chiamati attributi e le funzioni membro metodi*
- Una variabile di un tipo classe è detta **oggetto**
- Un programma è un insieme di oggetti che interagiscono

Incapsulamento

- Un tipo di dato consiste di
  - Un insieme di valori
  - Un insieme di operazioni
- È un **tipo di dato astratto (ADT)** se l'implementazione dei valori e delle operazioni è nascosta al programmatore
- I tipi di dati predefiniti sono ADT
- **Le classi devono essere ADT**

Alberto Ferrari

Membri Privati e Pubblici

- I membri **privati** possono essere referenziati solo all'interno delle funzioni membro
- I membri **pubblici** possono essere referenziati ovunque
- È buona norma **rendere private tutte le variabili membro** e pubbliche le sole funzioni membro necessarie (quelle che espongono le funzionalità della classe)

Alberto Ferrari

Sezioni Pubblica e Privata

- Generalmente si definisce una sola sezione pubblica e una sola sezione privata
- Generalmente la sezione pubblica compare prima (per maggior chiarezza verso chi legge il codice)
- In assenza di specificazione C++ considera i membri privati

Alberto Ferrari

Un esempio: classe Tempo

```
class Tempo
{
public:
   void setOre (int o){ore=o;}
   int getOre() {return ore;}
   void setMinuti (int m){minuti=m;}
   int getMinuti() {return minuti;}
   void visualizza() {cout<<ore<<":"<<minuti<<endl;}
private:
   int ore;
   int minuti;
};
```

esTempo01.cpp

Alberto Ferrari

Classi: Utilizzo Pratico

- Una volta definita, la classe può essere utilizzata come oggetto, array, puntatore e riferimento

```
Tempo t1,t2;
t1.setOre(9);
t2=t1;
t2.visualizza();

Tempo* t3;
t3 = new Tempo();
t3->setOre(8);
(*t3).setMinuti(20);     //forma deprecata

Tempo vt[3];
vt[0].setOre(10);

Tempo &t4=t1;
```

Alberto Ferrari

Funzioni accessor e mutator

- Le funzioni **accessor** consentono di leggere i dati dell'oggetto
- Le funzioni **mutator** consentono di modificare i dati dell'oggetto
- Queste funzioni forniscono un accesso controllato ai dati
- Generalmente si definiscono setter/getter functions
- E' buona norma mantenere private le variabili membro e dotarle di funzioni accessor e mutator **solo se necessario**

Alberto Ferrari

Funzioni accessor e mutator (2)

- Una funzione che setta il valore di un dato dovrebbe anche controllare la validità di tale valore (validation) e lasciare comunque il dato in uno stato consistente (oppure lanciare un'eccezione...)

```
void setMinuti (int m)
{
  if (m>0 && m<60)
      minuti=m;
  else
      minuti=0;
}
```

Alberto Ferrari

Costruttore

- È una funzione membro che viene chiamata automaticamente quando viene dichiarato un oggetto della classe
- Usata per operazioni di inizializzazione
- Deve avere lo stesso nome della classe
- Non può ritornare un valore

esTempo02.cpp

Alberto Ferrari

Costruttore (2)

- Deve stare nella sezione pubblica della classe
- Non può essere invocato come le altre funzioni membro
- Definizione alternativa preferibile: **sezione di inizializzazione** (vedi)
- Spesso si ha overloading dei costruttori

Alberto Ferrari

Costruttore senza Argomenti

- Quando si dichiara una variabile di tipo classe e si vuole invocato il costruttore senza argomenti, non si usano le parentesi

  Esempio: **Tempo t1;**

Alberto Ferrari

Chiamata Esplicita del Costruttore

- Il costruttore può essere chiamato esplicitamente per modificare le variabili membro di un oggetto
- Crea un oggetto anonimo e lo inizializza con i valori degli argomenti
- L'oggetto anonimo può essere assegnato a una variabile del tipo classe

  Esempio:
  ```
  Tempo t2 = Tempo(9);
  Tempo t4(12);
  Tempo* t3;
  t3 = new Tempo(10,5);
  ```
- È più efficiente usare le mutator functions
- Se sono disponibili sia costruttore/i che mutator functions, queste è consigliabile siano chiamate anche dal costruttore (per non ripetere codice)

Alberto Ferrari

Costruttore di Default

- Un costruttore senza argomenti è detto **costruttore di default**
- Se non definiamo nessun costruttore viene creato un costruttore di default
- Se definiamo almeno un costruttore il costruttore di default non viene creato
- È bene includere sempre il costruttore di default

Alberto Ferrari

Sezione di Inizializzazione

- Le variabili membro vengono create e inizializzate dal costruttore **nell'ordine in cui sono definite nella classe**
- La sezione di inizializzazione è più efficiente per variabili membro di tipo classe
- Le variabili membro **possono** essere inizializzate all'interno di una sezione di inizializzazione, mentre costanti, riferimenti e membri di tipo classe **devono** utilizzare questo costrutto

```
Tempo (int o=0, int m=0) :ore(o),minuti(m)
{ }
```

esTempo03.cpp

Variabili Membro di Tipo Classe

- Una classe può avere una variabile membro di tipo classe
- I parametri del costruttore della classe più esterna possono essere usati nella chiamata del costruttore della variabile membro

Distruttori

- Funzione membro chiamata automaticamente quando un oggetto viene distrutto
- Non riceve parametri e non restituisce valori
- Una classe deve avere un **unico** distruttore
- Nome del distruttore: ~+ *nomeClasse*
- Il distruttore non distrugge materialmente l'oggetto, ma è delegato a compiere tutte quelle operazioni utili prima che l'oggetto venga distrutto (es.: rilascio della memoria allocata dinamicamente)

esTempo04.cpp

Alberto Ferrari

Interfaccia ed Implementazione

- L'**interfaccia** di una classe consiste di
  - Dichiarazioni delle funzioni membro pubbliche
  - Commenti
- L'**implementazione** di una classe consiste di
  - Variabili membro e dichiarazioni delle funzioni membro private
  - Definizioni delle funzioni membro
- Generalmente stanno in file diversi, ma è importante la separazione concettuale

Alberto Ferrari

Suddivisione del Codice in File

- Separazione tra la classe e i programmi che la usano
  - Riuso: parti separate facilmente riusabili (libreria)
  - Compilazione selettiva
- Separazione tra interfaccia e implementazione
  - Incapsulamento: occultamento dei dettagli
  - Diverse implementazioni di una stessa libreria

Alberto Ferrari

Compilazione Separata

- Il file che contiene il programma che usa la classe si chiama **file di applicazione**
- Sia l'implementazione che l'applicazione devono includere l'header file
- L'implementazione e l'applicazione vengono compilate separatamente
- Per ottenere l'eseguibile occorre linkare i due oggetti

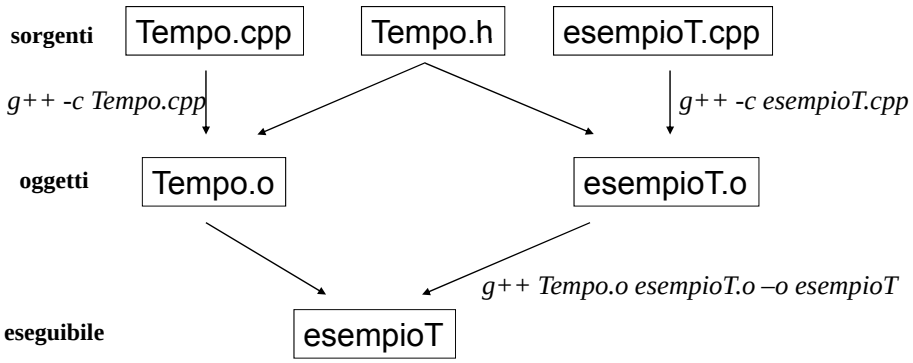
Alberto Ferrari

Programmazione Orientata agli Oggetti

- Un programmatore deve poter usare la classe conoscendo solo l'interfaccia
- Vantaggio: è possibile cambiare l'implementazione senza dover cambiare qualsiasi altro codice che usi la classe
- Filosofia: descrivere il problema in termini di oggetti che interagiscono, piuttosto che algoritmi che operano su dati (anche algoritmi e dati possono cambiare...)

esTempo05

Le fasi di compilazione



Classe di Esempio: Interfaccia

```
#ifndef TEMPO_H                  // direttiva al compilatore
#define TEMPO_H                  // per prevenire inclusioni multiple

class Tempo
{
public:

   Tempo (int, int);             // Costruttore
   ~Tempo();                     // Distruttore
   // Funzioni accessor e mutator
   void setOre (int);
   int getOre();
   void setMinuti (int);
   int getMinuti();
   // altre funzioni pubbliche
   void visualizza();
private:
   int ore;
   int minuti;
};
#endif
```

Alberto Ferrari

Classe di Esempio: Implementazione

```
#include "Tempo.h"
#include <iostream>
using namespace std;
…
// Costruttori
Tempo::Tempo (int o, int m)
{
          cout<<"costruttore con due parametri"<<endl;
          setOre(o);
          setMinuti(m);
}
Tempo::~Tempo()
{
     cout<<"chiamato distruttore"<<endl;
}
void Tempo::setMinuti (int m)
{
         minuti=m;
}
int Tempo::getMinuti()
{
  return minuti;
}
void Tempo::visualizza()
{
  cout<<ore<<":"<<minuti<<endl;
} …
```

esTempo05

Alberto Ferrari

Membri static: Variabili

- Una **variabile membro static** è condivisa da tutti gli oggetti di una classe
- Usata dagli oggetti della classe per comunicare e coordinarsi
- Solo gli oggetti della classe possono accedervi
- Va inizializzata al di fuori della definizione della classe, una sola volta

Alberto Ferrari

Membri static: Funzioni

- Una **funzione membro static** accede solo ai membri static
- Non può accedere ai dati dell'oggetto chiamante
- Viene invocata usando il nome della classe e lo scope resolution operator (::)
- La parola chiave **static** va messa solo nella dichiarazione

`esTempo06`

Alberto Ferrari