OOP

OBJECT ORIENTED PROGRAMMING

- Astrazione

- 1980 necessità di riutilizzare il codice
- Crisi dimensionale dei sistemi informatici
- Crisi gestionale (costi per manutenzione)
- Crisi qualitativa (strumenti non suff)

- OGGETTO = astrazione di un oggetto reale

- ATTRIBUTI = caratteristiche di un oggetto o proprietà

- METODI = comportamenti o azioni che l’oggetto può svolgere

ATTRIBUTI
Velocità, colore, numero di porte

METODI
Accelera, frena, gira, cambia marcia

ISTANZA : Bmw520, Mercedes, Yaris

- ISTANZA = entità particolare con le sue specifiche caratteristiche

- CLASSE = rappresenta un modello per descrivere un insieme di oggetti. Essa specifica gli attributi, senza indicarne il valore, e i metodi

```
#include <iostream>

using namespace std;

class Rettangolo
{public:
    float base, altezza;
    float Area(){
    return base*altezza;}

};

int main()
{Rettangolo figura1;
figura1.base=10;
figura1.altezza=8;
cout << "L'area del rettangolo e' "<< figura1.Area() <<endl;
return 0;
}
```

Definizione di una classe

class *NomeClasse // Identificatore valido*
{
dichiarazioni dei dati // attributi
definizione delle funzioni // metodi
};

ha due parti:

dichiarazione: descrive i dati e l'interfaccia (cioè le "funzioni membro", anche dette "metodi")

definizioni dei metodi: descrive l'implementazione delle funzioni membro
gli attributi sono variabili semplici (interi, strutture, array, float, ecc.) o complessi (oggetti istanze di altre classi)
i metodi sono funzioni semplici che operano sugli attributi (*dati)*

Visibilità di un membro

- Public: gli elementi (attributi e metodi) sono visibili anche al di fuori della classe di appartenenza

- Private: tutti gli elementi che seguono non possono essere visti al di fuori della classe

- Protected: il membro è visibile solo dalle classi che derivano dalla classe principale

Information hiding / incapsulamento

questa caratteristica della classe si chiama *occultamento di dati (information hiding) ed è una proprietà dell'OOP*
si tratta di una tecnica, spesso denominata *incapsulamento,* che limita molto gli errori rispetto alla programmazione Strutturata

```
class Semaforo
{
public:
void cambiareColore();
//...
private:
enum Colore {VERDE, ROSSO, GIALLO};
Colore c;
};
```

Oggetti

definita una classe, possono essere generate *istanze della classe,*
cioè *oggetti*

nome_classe identificatore ;

Rettangolo figura1

un oggetto sta alla sua classe come una variabile al suo tipo

```
Rettangolo figura1;
figura1.base=10;
figura1.altezza=8;
cout << "L'area del rettangolo e' "<< figura1.Area() <<endl;
```

Attributi

possono essere di qualunque tipo valido, con eccezione del tipo della classe che si sta definendo
non è permesso inizializzare un membro dato di una classe all'atto della sua definizione; la seguente definizione di classe genera quindi errori:

```
class C {
private:
int T = 0; // Errore
const int CInt = 25; // Errore
int& Dint = T // Errore
// ...
};
```

non avrebbe senso inizializzare un membro dato dentro la definizione della classe, perché essa indica semplicemente il *tipo di ogni membro dato e non* riserva realmente memoria; si deve invece inizializzare i membri dato ogni volta che si crea un'*istanza specifica della classe mediante il costruttore della classe*

I metodi

i metodi possono essere sia dichiarati che definiti all'interno
delle classi; la definizione di un metodo consiste di quattro parti:

- il tipo restituito dalla funzione
- il nome della funzione
- la lista dei parametri formali (eventualmente vuota) separati da virgole
- il corpo della funzione racchiuso tra parentesi graffe

le tre prime parti formano il prototipo della funzione che *deve essere definito* dentro la classe, mentre il corpo della funzione può essere definito altrove

```
class Rettangolo
{public:
    float base, altezza;
    float Area(){
    return base*altezza;}

};
```

- Esercizio: costruire una piccola calcolatrice, che dati due numeri, sia in grado di elaborare le quattro operazioni elementari

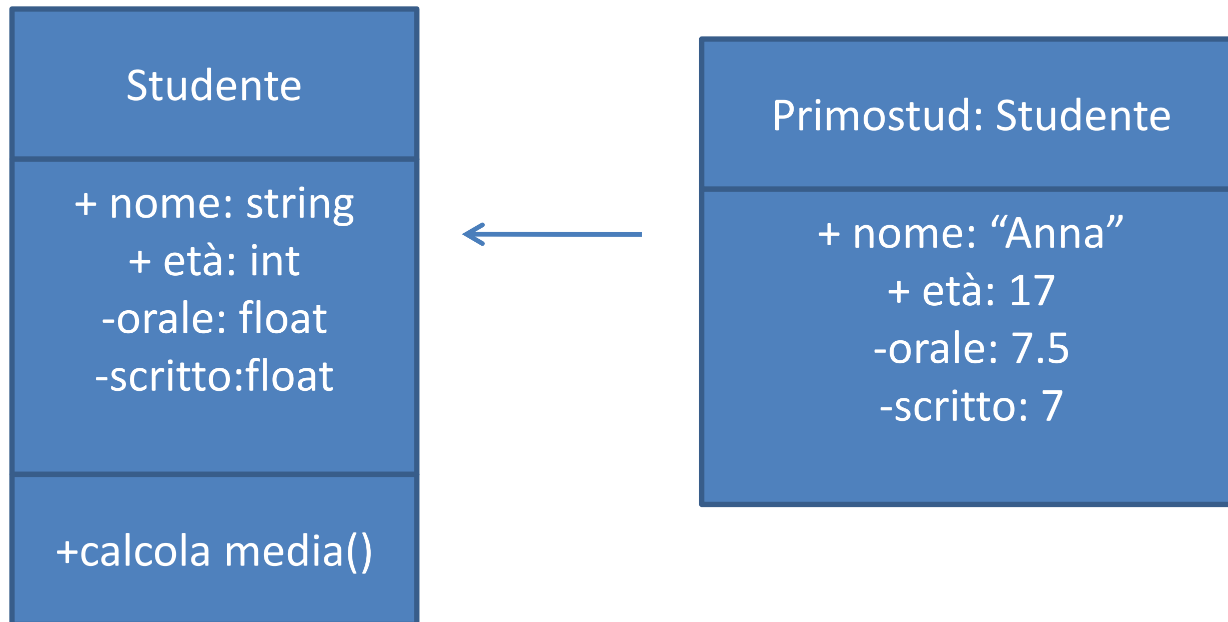
Header file ed intestazioni di classi

il codice sorgente di una classe si colloca normalmente in un file indipendente con lo stesso nome della classe ed estensione .cpp

le dichiarazioni si collocano normalmente in header file indipendenti da quelli che contengono le implementazioni dei metodi

Es.:
la dichiarazione di una classe mia_classe sarà inserita in in un header file mia_classe.h,
mentre l'implementazione sarà definita in mia_classe.cpp
Un file che crea oggetti di una classe dichiarata in mia_classe.h deve utilizzare la direttiva #include "miaclasse.h"

UML Unified Modeling Language



Inserimento dei dati da tastiera

```
#include <iostream>
using namespace std;
class Rettangolo
{public:
   float base, altezza;
   float Area(float base, float altezza){
   return base*altezza;}
};
int main()
{Rettangolo figura1;
float b,h;
cin>>b;
cin>>h;
cout << "L'area del rettangolo e' "<<
     figura1.Area(b,h) <<endl;
return 0;
}
```

```
#include <iostream>
using namespace std;
class Rettangolo
{public:
    float base, altezza;

    void Assegna(float b, float h){
        base=b;
        altezza=h;}

    float Area(){
    return base*altezza;}
};
int main()
{Rettangolo figura1;
float b,h;
cin>>b;
cin>>h;
figura1.Assegna(b,h);
cout << "L'area del rettangolo e' "<< figura1.Area()
<<endl;
return 0;
}
```

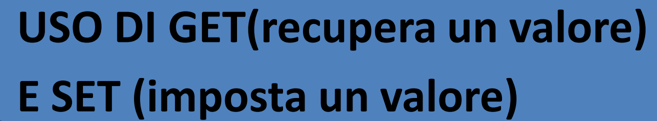
```
class Rettangolo
{public:
    float base, altezza;
    void  setbase(float b){
      base=b;}

    void  setaltezza(float  h){
      altezza=h;}

    float getbase(float b);
    float getaltezza(float h);
};

float Rettangolo::getbase(float b){
     return base;}

float Rettangolo::getaltezza(float h){
     return altezza;}
```



Esercizi

- Creare una classe Atleta per rappresentare le informazioni di un giocatore nome, cognome, squadra, ruolo, sport. Creare un metodo per l'inserimento dei dati e un metodo per la visualizzazione. Creare un'istanza nel main.
- Creare una classe CD per le informazioni su un disco musicale: titolo, autore, durata, prezzo. Creare un metodo per l'inserimento dei dati, uno per la visualizzazione ed uno per incrementare il prezzo di una percentuale inserita da tastiera.

Costruttori

un *costruttore è appunto un metodo di una classe che viene a*utomaticamente eseguito all'atto della creazione di un oggetto di quella classe ha lo stesso nome della propria classe e può avere qualunque numero di parametri ma non restituisce alcun valore (neanche void)

a volte può essere conveniente che un oggetto si possa autoinizializzare all'atto della sua creazione, senza dover effettuare una successiva chiamata ad una sua qualche funzione membro

```
class Rettangolo
{private:
int base;
int altezza;
public:
Rettangolo(int b, int h): base(b), altezza(h){}; // Costruttore
Rettangolo () {
        base=0;
        altezza=0; }
Rettangolo(): base(0), altezza(0); {}
// definizioni di altre funzioni membro};
float Rettangolo::Area() { return base*altezza;}
```

Definizione di un oggetto con un costruttore

```
class Punto2D
{
public:
Punto2D();
Punto2D(int coord1, int coord2);
private:
int x;
int y;
};
Punto2D P; // chiama il costruttore di default
Punto2D Origine(0,0); // chiama il costruttore alternativo
```

Distruttore

si può definire anche una funzione membro speciale nota come *distruttore, che viene chiamata automaticamente quando si* distrugge un oggetto

il distruttore ha lo stesso nome della sua classe preceduto dal carattere ~ neanche il distruttore ha tipo di ritorno ma, al contrario del costruttore, non accetta parametri e *non ve ne può essere più d'uno*

```
class Demo
{
private:
int dati;
public:
Demo() {dati = 0;} // costruttore
~Demo() {}; // distruttore
};
```

~ Alt+126

serve normalmente per liberare la memoria assegnata dal costruttore

se non si dichiara esplicitamente un distruttore, C++ ne crea automaticamente uno vuoto

Esercizi

1. Creare una classe conto e un costruttore che assegni il valore iniziale al suo saldo. Prevedere poi dei metodi per: versare dei soldi nel conto, prelevare dei soldi se possibile, contare quante operazioni (prelievo e versamento) sono state fatte, applicare un costo in percentuale per ogni prelievo che viene svolto, visualizzare i dati, versare soldi in un altro conto corrente (creare un'altra istanza), azzerare il saldo per poi ripristinare il suo effettivo valore.
2. Creare la classe orologio con le ore, minuti, secondi. I metodi della classe consentono di azzerare l'orario, correggere l'ora, leggere l'orario completo con ore, minuti e secondi
3. Il computer di bordo di un'automobile è in grado di calcolare il consumo medio conoscendo i km percorsi e i litri di carburante consumati. Definire la classe automobile con il metodo per il calcolo oltre al costruttore per inizializzare gli attributi
4. Definire la classe libro con gli attributi: titolo, autore, anno, città, editore e soggetto. Implementare i metodi per inserire e visualizzare i dati dei libri, aggiungendo anche un costruttore e un distruttore

- Creare una classe batteria della quale conosciamo la capacità massima espressa con un numero intero e la carica attuale anch’essa numero intero (non deve superare la capacità massima). Creare una classe cellulare della quale conosciamo il numero di telefono (stringa), batteria, sms inviati, sms ricevuti, acceso o spento. Del cellulare vogliamo:

1) Visualizzare i dati
2) Spegnerlo
3) Accenderlo, sapendo che consuma 5 minuti di carica
4) Caricarlo per un certo periodo di tempo espresso in minuti, sapendo che ad ogni minuto si raddoppia la carica
5) Mandare un sms a un altro cellulare diminuendo di 10 minuti la carica del cellulare che invia il messaggio, mentre il cellulare che riceve il messaggio, aumenta la carica di 3 minuti
6) visualizzare quale cellulare ha la carica maggiore

Poliformismo: Overloading

anche le funzioni membro possono essere sovraccaricate, ma soltanto nella loro propria classe, con le stesse regole utilizzate per sovraccaricare funzioni ordinarie

due funzioni membro sovraccaricate non possono avere lo stesso numero e tipo di parametri

l'*overloading permette di utilizzare uno stesso nome per più metodi che si* distingueranno solo per i parametri passati all'atto della chiamata

```
class Prodotto
{
public:
int prodotto (int m, int n); // metodo 1
int prodotto (int m, int p, int q); // metodo 2
int prodotto (float m, float n); // metodo 3
int prodotto (float m, float n, float p); // metodo 4
```

Classi derivate: Ereditarietà

- Una *classe derivata eredita attributi e metodi dalla classe base già* esistente
- Un oggetto della classe derivata è un oggetto della classe base (Es.: bicicletta IS-A veicolo
- la dichiarazione di una classe derivata deve includere il nome della classe base da cui deriva ed, eventualmente, uno specificatore d'accesso indicante il tipo di ereditarietà (public, private o protected)
- secondo la seguente sintassi:
- **class *ClasseDerivata : specific_accesso_opz ClasseBase {*** *membri;* **};**
- **Class TriangoloRettangolo::public Rettangolo{….};**

- *Ereditarietà pubblica: specificatore di accesso public; significa che i*
- membri pubblici della classe base sono tali anche per quella derivata
- *Ereditarietà privata: specificatore di accesso private*
- *Ereditarietà protetta: specificatore di accesso protected*
- Se si omette lo specificatore di accesso si assumerà per default private

Esempio

```
Class Rettangolonew: public Rettangolo {
        Int codicecolore;
        Public:
        Void colora(int c) {
        Codicecolore=c; }
       };

        Int main()
 {             Rettangolonew  figura1;
               float b,h; int col;
               cin>>b;
               cin>>h;
               figura1.Assegna(b,h);
               cin>>col;
               Figura1.colora(col);
               cout << "L'area del rettangolo e' "<< figura1.Area() <<endl;
               return 0; }
```

Tipi di ereditarietà

- in una classe, gli elementi pubblici sono accessibili a tutte le funzioni, quelli privati sono accessibili soltanto ai membri della stessa classe e quelli protetti possono essere acceduti anche da classi derivate (*proprietà dell'ereditarietà)*
- vi sono tre tipi di ereditarietà: *pubblica, privata e protetta, la più* utilizzata delle quali è la prima
- una classe derivata non può accedere a variabili e funzioni private della sua classe base
- per occultare dettagli una classe base utilizza normalmente elementi protetti invece che elementi privati
- supponendo ereditarietà pubblica, gli elementi protetti sono accessibili alle funzioni membro di tutte le classi derivate
- per default, l'ereditarietà è privata; se accidentalmente si dimentica la parola riservata public, gli elementi della classe base saranno inaccesibili

- Esempio

```
class Base{
public:
int i1;
protected:
int i2;
private:
int i3;
};
```

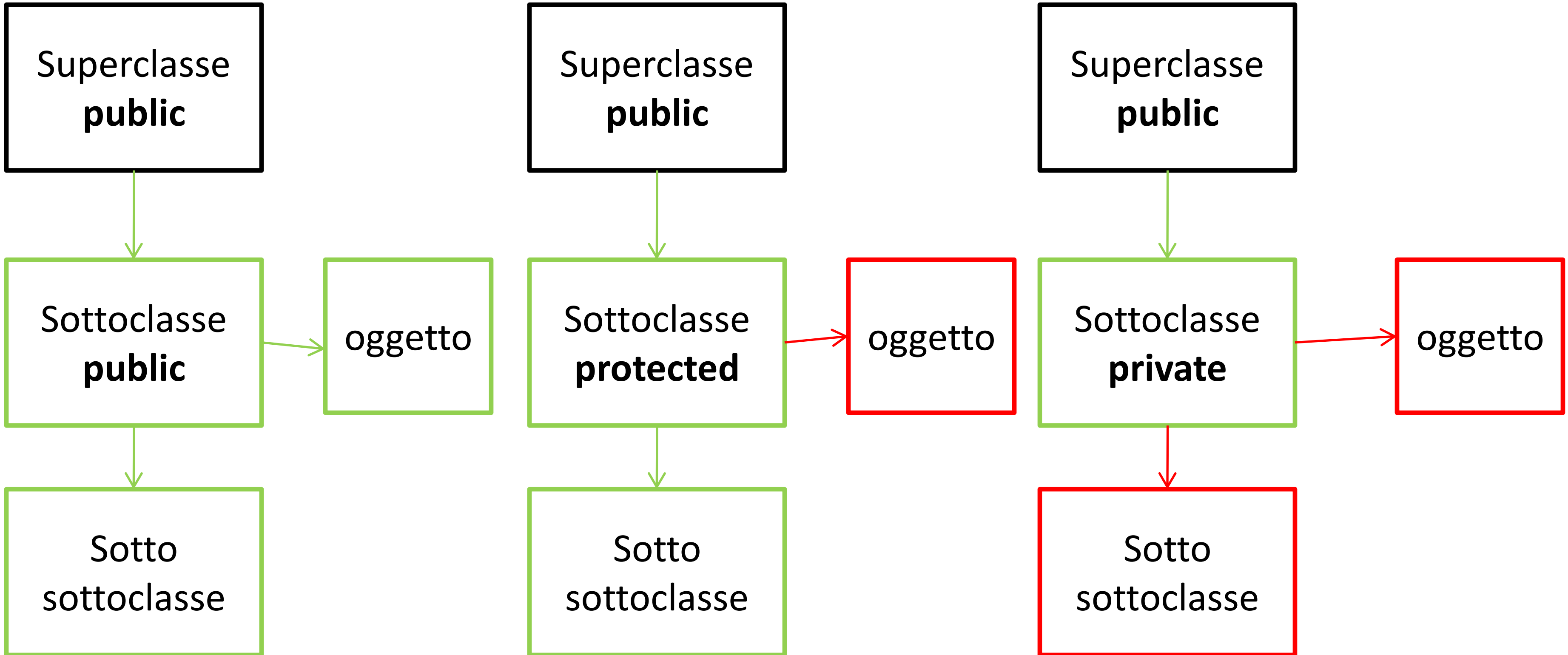
```
class D1: private Base {
void f();
};
…
void D1::f() {
i1=0; // corretto
i2=0 // corretto
i3=0 // ERRATO
};
```

```
class D2: protected Base {
void g();
};
class D3: public Base {
Void h();
};
```

Le tre classi accedono ai membri i1 e i2
i3 è inaccessibile a tutte

E dall'esterno?

```
int main()
{
Base b;
b.i1=0; // ok
b.i2=0; // ERRORE
b.i3=0; // ERRORE
D1 d1;
d1.i1=0; // ERRORE
d1.i2=0; // ERRORE
d1.i3=0; // ERRORE
D2 d2;
d2.i1=0; // ERRORE
d2.i2=0; // ERRORE
d2.i3=0; // ERRORE
D3 d3;
d3.i1=0; // ok
d3.i2=0; // ERRORE
d3.i3=0; // ERRORE
}
```


Superclasse
public
Superclasse
public
Superclasse
public
Sottoclasse
public
oggetto
Sottoclasse
protected
oggetto
Sottoclasse
private
oggetto
Sotto
sottoclasse
Sotto
sottoclasse
Sotto
sottoclasse

```
Class Rettangolonew: private Rettangolo
        Int codicecolore;
        Public:
        Void colora(int c)
             Codicecolore=c;
        ;

        Int main()
{               Rettangolonew  figura1;
                float b,h; int col;
                cin>>b;
                cin>>h;
                figura1.Assegna(b,h);
                cin>>col;
                figura1.base=3;       //non posso scriverlo perché ereditarietà privata
                figura1.colora(col);
                cout << "L'area del rettangolo e' "<< figura1.Area() <<endl;
                return 0;
```

- Costruttore di una classe derivata

la sintassi di un costruttore di una classe derivata è:

*ClasseDer**::ClasseDer(paramD):ClasseBase(paramB), Inizial {***

// corpo del costruttore della classe derivata**};**

Inizial è l'inizializzazione di membri dato della classe

```
class Data {
public:
Data(int, int, int);
private:
int giorno, mese, anno;
};
```

```
class Persona {
public:
Persona(string, string, int g, int m, int a);
private:
string nome;
string cognome;
Data data_nascita;
};
```

```
class Impiegato : public Persona {
public:
Impiegato(string, string, string,
int, int, int, int, int, int);
private:
string matricola;
const Data data_assunzione;
};
```

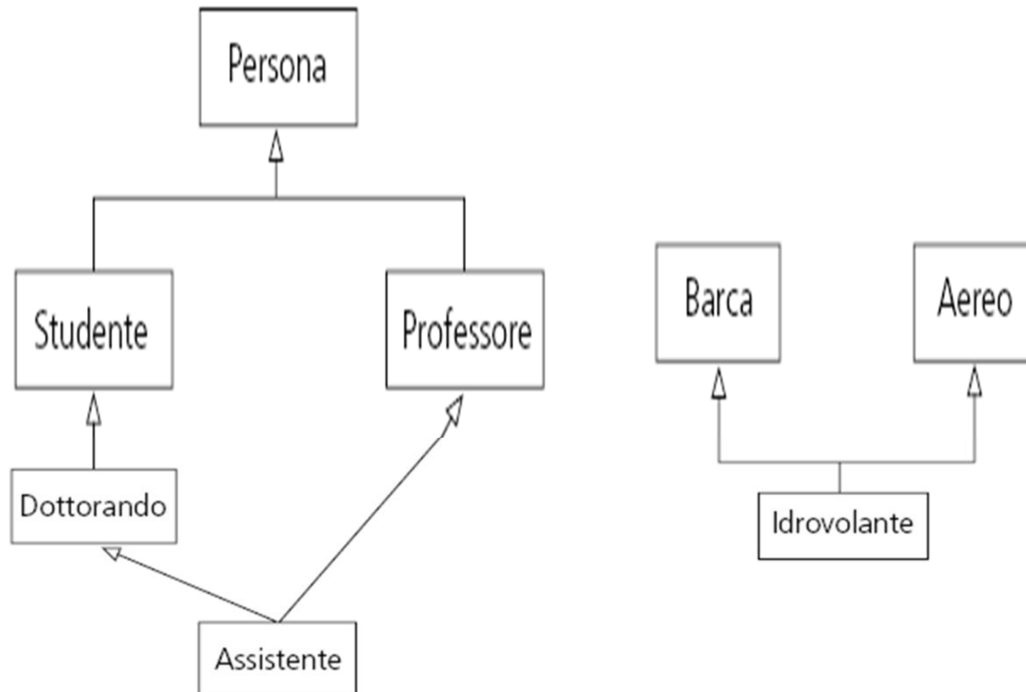
```
// costruttore per la classe Impiegato
Impiegato::Impiegato(string n, string c,
string mat, int gg, int mm, int
aa, int giornoa, int mesea, int
annoa):Persona(n,c,gg,mm,aa),
data_assunzione(giornoa, mesea, annoa)
```

Distruttori ed ereditarietà

- i distruttori non si ereditano, ma se necessario si genera un
- distruttore di default
- si utilizzano normalmente solo quando un corrispondente
- costruttore ha assegnato spazio in memoria che deve essere
- liberato

Ereditarietà multipla

- una classe può ereditare attributi e comportamento di più di una



Ereditarietà multipla

- la sintassi è:

class *Derivata : [virtual][tipo_accesso] Base1,*

[virtual][tipo_accesso] Base2,

[virtual][tipo_accesso] Basen {

public:

// sezione pubblica

private:

// sezione privata

...

};

- *derivata: nome della classe derivata*
- *tipo_accesso: public, private o protected*
- *Base1, Base2,...: classi base con nomi differenti*
- *virtual..: è opzionale e specifica una classe base compatibile.*
- Funzioni o dati membro che abbiano lo stesso nome in *Base1, Base2, Basen,*
- costituiranno motivo di ambiguità

Esempio

```
class Studente {
private:
long matricola;
…
public:
long getMatricola() const;
…
};
class Lavoratore {
private:
long stipendio;
public:
long getStipendio() const;
…
};
class studente_lavoratore: public studente, public lavoratore
```

Esercizi

- Ereditarietà: Creazione di una classe Studente derivata da Persona
- Arricchimento della classe Studente con nuove funzioni per la gestione degli esami
- Creare una classe libro con titolo, editore, prezzo e derivare la classe rivista aggiungendo l'attributo sulla periodicità
- Creare la classe veicolo e derivare la classe automobile e autobus: aggiungere gli attributi opportuni e il metodo che consente di visualizzare i valori degli attributi
- Derivare dalla classe conto (già creata in precedenza) una nuova classe conto corrente: aggiungendo i metodi Preleva, che decrementa il saldo della cifra fornita come parametro, Versa, che incrementa il saldo e Mostra, che visualizza il saldo

Overriding

- Sovrascrivere, nella classe derivata, un metodo ereditato, cambiandone il comportamento, ma mantenendo uguale il numero e il tipo di parametri

Esempio

```
Class Rettangolo2: public Rettangolo    {

    Public:
        float base, altezza;
        float Area(){
                If (base!=0) and(altezza!=0)
                        return base*altezza;
                else return 0;}
    };

    Int main()
{               Rettangolo2  figura1;
                float b,h;
                cin>>b;
                cin>>h;
                figura1.Assegna(b,h);
                cout << "L'area del rettangolo e' "<< figura1.Area() <<endl;
                return 0;}
```

Esercizi

- Costruire una classe base anagrafica con codice, cognome, nome, registrato (è un bool). I suoi metodi sono registra (bool diventa true) e mostra solo se è stato registrato. Prevedere anche il costruttore. Derivare da essa la classe Dipendente, aggiungendo gli attributi per la mansione e lo stipendio e modificando il metodo di registrazione della classe base con l'assegnazione della mansione che svolge in azienda
- Derivare dalla classe anagrafica, la classe cliente aggiungendo gli attributi per il telefono, partita IVA e modificando il metodo di registrazione della classe base con l'assegnazione della partita IVA
- Costruire la classe Dipendente e derivare le classi Operaio, Impiegato e Dirigente. Nella classe base l'attributo stipendio viene dichiarato protetto in modo che possa essere modificato anche con i metodi delle classi derivate
- Una classe solido consente di rappresentare una figura solida e di calcolarne il volume; derivare da essa la classe Bilancia che consente di modificare il metodo per il calcolo, in modo da ottenere anche il peso del solido conoscendone il peso specifico (peso= volume*peso specifico)

Oggetti dinamici

- Oggetti che vengono introdotti durante l'esecuzione del programma
- Usano una parte della memoria RAM chiamata heap e possono essere distrutti soltanto se ciò viene espressamente codificato
- Utili perché entrano in azione solo se espressamente richiesti (come il correttore ortografico di word)

```
Studente *miostudente=new Studente ()      //puntatore a
                                 oggetto e allocazione della memoria
miostudente->media();           //accesso a metodo
miostudente->nome;              //accesso a attributo
delete miostudente;             //distrugge l'oggetto dinamico
```

Array di oggetti

- Docente array.cpp

Il costruttore con parametri, se c'è un array di oggetti, ha bisogno di un costruttore vuoto e la sua istanza va inizializzata:

string targa=" ";

bool tipologia=true;

automobile macchina[10]={{targa, tipologia}}

macchina[i]=automobile(targa, tipologia)

- Creare una classe che permette l'inserimento di descrizione, prezzo, numero di reparto per 4 articoli di magazzino

Funzioni virtuali

- Virtual1.cpp
- La classe Cubo ha come attributo un numero intero, che rappresenta il lato, e possiede come metodi le funzioni per il calcolo del volume e per la visualizzazione delle caratteristiche della figura solida (lato, volume). Derivare dalla classe Cubo la classe Sfera che deve ridefinire il metodo per il volume attraverso una formula di calcolo diversa. Utilizzare le funzioni virtuali in modo che il metodo di visualizzazione ereditato dalla classe base produca in output i risultati corretti

Funzioni friend

- Una funzione è detta **friend** (letteralmente “amica”) di una classe, diversa di quella eventuale di appartenenza, se può accedere a tutti i suoi membri dichiarati private. La funzione può essere di qualsiasi tipo, cioè una normale funzione o una funzione membro di una classe.
  La dichiarazione di una funzione friend è molto semplice: basta inserire il prototipo della funzione nella definizione della classe, preceduto dalla parola chiave “friend” (non importa se nella sezione protetta o pubblica).

- Friend1.cpp

Esercizio

- Gli studenti delle scuole superiori di una città sono iscritti a diversi Istituti. Scrivi il programma per inserire gli studenti, assegnandoli alle scuole, oltre che per ottenere l’elenco completo degli studenti e l’elenco degli studenti iscritti a una determinata scuola. Le scuole sono identificate da un codice numerico che va da 1 al numero totale delle scuole. Utilizza una funzione friend per ritrovare gli studenti della scuola richiesta da tastiera.