PROGRAMMIAMO

C++ - Ereditarietà

Ereditarietà: classe base e classe derivata

Con **ereditarietà** (*inheritance*) si intende un meccanismo mediante il quale è possibile creare una nuova classe derivandola da una già esistente. La classe derivata è detta **classe derivata** (o child class, classe figlia) mentre la classe originale è la **classe base** (o parent class, classe genitore).

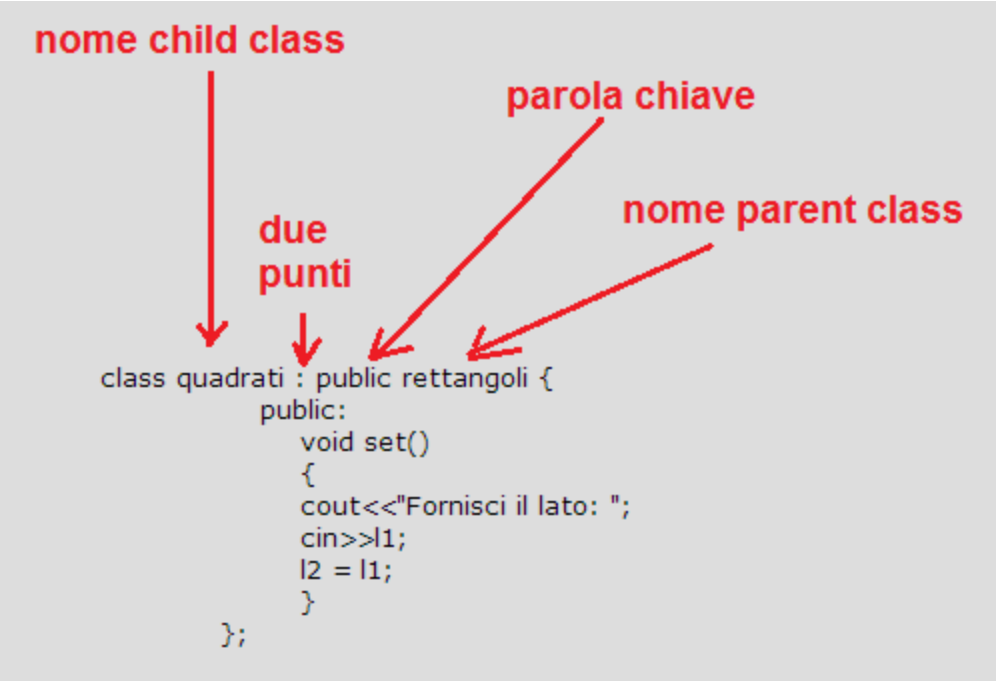
L'ereditarietà consente di creare nuove classi basandosi su classi già esistenti, senza dover riscrivere daccapo tutto il codice. Facciamo un esempio. Supponiamo di avere una classe rettangoli per il calcolo dell'area e del perimetro di un rettangolo:

```
class rettangoli {
        private: double l1, l2;
        public:
          void set();
          double area();
          double perimetro();
      };
```

Supponiamo ora di voler creare una classe per il calcolo di area e perimetro di un quadrato. Siccome il quadrato è un caso particolare di rettangolo, invece di riscrivere da zero una nuova classe, possiamo riutilizzare il codice della classe rettangoli definendo una classe derivata nel seguente modo:

```
class quadrati : public rettangoli {
        public:
          void set()
          {
          cout<<"Fornisci il lato: ";
          cin>>l1;
          l2 = l1;
          }
      };
```

Osserviamo anzitutto la sintassi usata:



Le cose importanti da sapere sono:

1. la classe derivata (*quadrati*) eredita tutti i dati e tutti i metodi della classe base (*rettangoli*). L'eventuale costruttore della child class chiama (invoca) automaticamente il costruttore della parent class prima di essere eseguito.

2. la classe derivata può accedere liberamente a tutti i campi *public* della classe base come se si trattasse di propri campi "interni" (cioè senza usare l'operatore punto '.'); questo a condizione di usare la parola chiave *public* nella dichiarazione della classe, come mostrato nell'esempio sopra.

3. inoltre la classe derivata può definire liberamente al proprio interno altri campi aggiuntivi (dati e/o metodi) rispetto a quelli ereditati dalla classe base.

4. in particolare la classe derivata può ridefinire al proprio interno i metodi della classe base usando gli stessi nomi (in questo modo i metodi della parent class con lo stesso nome vengono sostituiti dai corrispondenti metodi della child class col meccanismo già noto dell'overloading).

5. i campi *private* della classe base non sono direttamente accessibili alla classe derivata. In caso contrario sarebbe sempre possibile modificare dall'esterno i campi private di una classe dichiarando una classe child.

Torniamo ora al codice della classe quadrati precedentemente dichiarata:

```
class quadrati : public rettangoli {
        public:
          void set()
          {
          cout<<"Fornisci il lato: ";
          cin>>l1;
          l2 = l1;
          }
```

```
};
```

Poiché, come si è detto, il quadrato è semplicemente un caso particolare di rettangolo, la child class utilizza i metodi di calcolo dell'area e del perimetro della parent class rettangoli senza bisogno di ridefinirli. Essa si limita a riscrivere il metodo set che utilizza il semplice "trucco" di acquisire un unico valore di lato e di assegnarlo quindi uguale a entrambi i campi l1 e l2.

Tuttavia, in base a quanto detto prima, la nostra class quadrati non può funzionare dal momento che l1 e l2 sono stati dichiarati come private nella classe rettangoli e dunque non è possono essere utilizzati dall'esterno. Ovviamente si potrebbe aggirare il problema dichiarando l1 e l2 public nella class rettangoli, ma in questo modo chiunque potrebbe accedere direttamente a tali dati.

Una soluzione più elegante è la seguente:

```
class rettangoli {
        protected: double l1, l2;
        public:
            void set();
            double area();
            double perimetro();
};
```

Si noti la parola chiave **protected** che indica il fatto che i campi l1 e l2 sono private per tutti tranne che per le classi derivate dalla classe rettangoli.

Naturalmente l'esempio proposto è molto semplice e il lettore potrebbe giustamente dubitare dell'utilità del meccanismo dell'ereditarietà applicato alle classi rettangoli e quadrati. In realtà la potenza dell'ereditarietà si può vedere solo nel caso di classi parent complesse, per le quali risulta davvero vantaggiosa la possibilità di riutilizzare il codice già scritto senza doverlo riscrivere daccapo.

precedente - successiva