Corso sul linguaggio C++

Modulo 5

1 – Classi e oggetti

Prerequisiti

- Oggetti reali e oggetti software
- Proprietà e comportamento di un oggetto
- Programmazione elementare in C++

Introduzione

In questa Unità vediamo i primi concetti della **OOP** in C++.

Si esaminano le tecniche elementari per la realizzazione di una classe e alcuni concetti di base.

Argomenti

- Astrazione
- Implementazione
- Programmazione ad oggetti
- Il linguaggio C++
- Struttura di una classe
- Specificatori di accesso
- Un esempio di classe
- I costruttori
- Il distruttore
- Metodi accessori e modificatori
- Creare gli oggetti
- Funzionamento di una classe
- Dichiarazione di una classe
- La classe *Coppia*
- Dichiarazione degli attributi
- Definizione dei metodi
- Metodi *get* e *set*
- Il metodo costruttore
- Creazione di oggetti
- Utilizzo di oggetti

Astrazione

Quando si deve **progettare** qualcosa occorre:

- schematizzare la realtà considerando soltanto gli aspetti che ci interessano
- creare un modello semplificato che però svolga le funzioni che ci interessano
- considerare gli aspetti dell'entità che ci interessano ad un livello di dettaglio sufficiente per potere operare con l'entità stessa

Questa fase progettuale si dice **astrazione**.

Quando si progetta una classe, si fa un lavoro di astrazione: la classe consente di creare **oggetti software** che simulano **oggetti reali**

Implementazione

Una volta che si è creato un modello astratto, occorre **realizzare** il software che consente di:

- **rappresentarlo** (**attributi**)
- **operare** con esso (**metodi**)

Questa fase si dice **implementazione**.

Una volta implementata una classe, possiamo creare da essa quanti oggetti vogliamo ed operare con essi *solo conoscendone l'interfaccia.*

Programmazione ad oggetti

La **programmazione strutturata** (tipica dei linguaggi Pascal, C, Fortran) si è evoluta dagli anni ’60 fino agli anni ’80, consentendo di raggiungere buoni standard di produzione software.

Dagli anni ’80 in poi, la tecnica della programmazione ha subito un’ulteriore evoluzione mediante la **tecnica di programmazione ad oggetti** che consente un notevole passo in avanti per la realizzazione di software:

- **riusabile** (si evita di doverlo riscrivere)
- **modificabile** (si adatta con facilità a nuove esigenze)
- **sicuro** (che consenta *protezione dei dati*)

Il linguaggio C++
Applicazioni

Quando il linguaggio C++ è usato per creare oggetti, ogni programma è una **classe**.

Programmare ad oggetti in C++ significa scrivere un’**applicazione** *che definisca un insieme di classi*, **una per ciascuna tipo di oggetto** necessario per l’esecuzione del programma

Il linguaggio C++
Programmazione in C++

La programmazione **OOP** in C++ coinvolge i seguenti aspetti:

- *conoscenza del linguaggio C++* (sintassi e semantica di C++)
- *uso di oggetti e classi predefiniti* (ad esempio, definiti nelle **API** di C++ o in altri package a disposizione)
- *definizione di nuove classi*

Rivediamo brevemente i concetti già noti di C++, alla luce del paradigma ad oggetti.

Struttura di una classe

```
#include <iostream>
#include <cstdlib>
using namespace std;
class ident
{ private:   dichiarazione attributi
  public: elenco metodi
};
espansione delle definizioni dei metodi
int main()
{  istruziioni
}
```

Nome della classe

Le parole chiave **public** o **private**, prendono il nome di **specificatori di accesso** e indicano se l'attributo o il metodo è accessibile dall'esterno o meno

Il metodo **main()** consente di creare gli ogetti e di attivare i metodi

Struttura di una classe

```
#include <iostream>
#include <cstdlib>
using namespace std;
class ident
{  private:   dichiarazione attributi
   public: elenco metodi
};
espansione delle definizioni dei metodi
int main()
{   istruziioni
}
```

L'espansione dei metodi può essere fatta al momento della loro dichiarazione oppure dopo la dichiarazione della classe.

Specificatori di accesso

Alcune osservazioni sugli specificatori di accesso.

- Servono ad indicare l'accessibilità dei membti (attributi e metodi).
- La sezione indicata con **private** contiene i membri non accessibili all'esterno (in generale gli attributi vanno dichiarati privati).
- La sezione indicata con **public** contiene i membri che possono essere usati all'esterno.

Un esempio di classe

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Numero
{  private:  int x;
   public:
        Numero();          // costruttore senza parametri
        Numero (int n);    // costruttore con parametri
        ~Numero();         // distruttore
        void setN();       // metodo di lettura
        int getN();        // metodo di stampa
};
```

In questo semplice esempio di classe è presente un solo attributo intero e alcuni metodi.

I costruttori

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Numero
{  private:  int x;
   public:
        Numero();          // costruttore senza parametri
        Numero (int n);    // costruttore con parametri
        ~Numero();         // distruttore
        void setN();       // metodo di lettura
        int getN();        // metodo di stampa
};
```

I metodi **costruttori** servono ad istanziare la classe, per ottenere un oggetto.

Il distruttore

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Numero
{  private:  int x;
   public:
        Numero();          // costruttore senza parametri
        Numero (int n);    // costruttore con parametri
        ~Numero();         // distruttore
        void setN();       // metodo di lettura
        int getN();        // metodo di stampa
};
```

Il metodi **distruttore** caratterizzato dal carattere tilda ("~") serve a eliminare l'oggetto dalla mamoria quando non serve più.
Normalmente la distruzione avviene automaticamente al termine del programma, ma il distruttore risulta utile quando si gestiscono oggetti mediante la memoria dinamica.

Metodi accessori e modificatori

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Numero
{  private:  int x;
   public:
        Numero();          // costruttore senza parametri
        Numero (int n);    // costruttore con parametri
        ~Numero();         // distruttore
        void setN();       // metodo di lettura
        int getN();        // metodo di stampa
};
```

Il metodi **accessori** e **modificatori** caratterizzati dai prefissi get e set consentono l'accesso ai membri per la lettura e per la stampa.

Creare gli oggetti

```
void Numero::setN()
{ cout<<"Immetti x: ";
  cin>>x;   }
int Numero::getN()
{ return x;   }
int main()
{  Numero n;
   n.setN();
   cout<<"numero:"<<n.getN();
   system("Pause");
   return EXIT_SUCCESS;
}
```

Definizione metodo

Nella definizione di un metodo poniamo le istruzioni che esso dovrà eseguire.

Il metodo **main**() *è l'unica operazione che può essere utilizzata direttamente dall'utente di una applicazione* in quanto viene eseguito automaticamente all'avvio dell'applicazione.

Funzionamento di una classe

Il funzionamento della classe *Numero* è il seguente (dopo aver ovviamente compilato il programma):

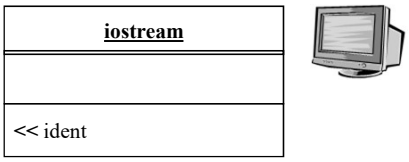
- si lancia l'applicazione
- viene eseguita la funzione **main**()
- viene creato automaticamente un oggetto di classe *Numero*
- **main**() esegue sull'oggetto *n* il metodo *setNumero*() che visualizzerà sullo schermo la frase

  Immetti x:
- successivamente, **main**() invoca il secondo metodo n.*getNumero()* che visualizza la frase

  numero:

Funzionamento di una classe

L'applicazione può visualizzare i valori sullo schermo perché utilizza l'oggetto **cout** di classe **iostream**



L'oggetto *cout*

- è un oggetto definito dalle **API** di Java
- modella lo *schermo del calcolatore*
- sa eseguire l'operazione di stampa **<<** che visualizza un valore *ident*

Dichiarazione di una classe

La **dichiarazione** di una classe ha le seguenti caratteristiche.

- Trattandosi di una dichiarazione si osserva che la *classe non occupa memoria*
- Quando viene istanziata, si crea **l'oggetto** (considerato come una variabile) che *occupa effettivamente memoria*.

Dichiarazione di una classe

Le clausole **public** o **private** prendono il nome di **specificatori di accesso** ed indicano se l'attributo o il metodo a cui si riferiscono può essere accessibile direttamente da*ll'utilizzatore della classe*.

In generale, per scopi di protezione, gli attributi vengono dichiarati **private** in modo che siano accessibili *solo tramite appositi metodi di lettura e scrittura* predisposti dal progettista e non direttamente dall'utente.

La classe *Coppia*

ATTIVITA': si vuole creare una classe *Coppia* con 2 attributi privati *x* ed *y*, un metodo costruttore e due metodi per comunicare all'esterno i valori degli attributi. Scrivere l'analisi del problema.

Analisi del problema

Per la classe richiesta sono necessari due attributi, *x* ed *y* che, per semplicità dichiariamo interi. La classe *Coppia* è l'unica classe necessaria e i suoi oggetti dovranno avere le seguenti funzionalità:

- un costruttore senza parametri *Coppia()*;
- un costruttore con parametri, per creare oggetti con uno stato definito dall'utente *Coppia (***int** a, **int** b)
- un metodo (*accessore*) *getx(),* per comunicare il valore *x*
- un metodo (*accessore*) *gety*(), per comunicare il valore *y*.

La classe *Coppia*

ATTIVITA': rappresentare la classe *Coppia* in UML.

Coppia	
- int x **- int** y	Primo attributo Secondo attributo
+ Coppia() + Coppia (**int** a, int b) ~Coppia() + **void** getX() + **void** getY()	Costruttore senza parametri Costruttore con parametri Distruttore Comunica il valore di x Comunica il valore di y

ATTIVITA': implementare la classe *Coppia*

La classe *Coppia*

File *Coppia.cpp*

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Coppia
{ private:   int x,y;
  public:
    Coppia(): x(0), y(0){}
    Coppia (int a, int b) {x=a, y=b;}
    ~Coppia() {cout<<"Distrutto"<<endl;}
    void Coppia::setX (int n)   { x=n; }
    void setY (int n)     { y=n; }
    int getX()          { return x;  }
    int getY()          { return y;  }
};
```

I **costruttori** (senza parametri o con parametri) della classe *Coppia* servono ad inizializzare l'oggetto.

Il **distruttore** avvisa quando il progrmma termina

La classe *Coppia*

…Segue

```
int main()
{  Coppia c;
   int x, y;
   cout<<"X: ";  cin>>x;
   c.setX(x);
   cout<<"Y: ";   cin>>y;
   c.setY(y);
   cout<<c.getX()<<endl;
   cout<<c.getY()<<endl;
   system("Pause");
   return EXIT_SUCCESS;
}
```

ATTIVITA': creare la classe *Coppia* e la classe *testCoppia (contenente il* **main()**) in due file separati e verificarne il funzionamento.

Creazione oggetto *c* di classe *Coppia* mediante il costruttore senza parametri

Tra due oggetti della stessa classe è possibile effettuare l'istruzione di **assegnazione** mediante il simbolo "=":

```
Coppia c1;
…
c1 = c;
```


Dichiarazione degli attributi

Come si è visto, gli **attributi** si dichiarano con

tipo ident;

dove

- *tipo* è il tipo di dato dell'attributo
- *ident* è l'identificatore assegnato all'attributo dal programmatore

Definizione dei metodi

Sempre dall'esempio, si vede che i metodi si **definiscono** con

tipo ident (*lista_par*)
{ *istruzioni*;
}

dove

- *tipo* indica il tipo del valore ritornato
- *ident* indica il nome assegnato al metodo
- *lista_par* indica la lista dei parametri, ciascuno espresso come

tipo ident

tipo indica il tipo di dato a cui appartiene *ident*

Metodi *get* e *set*

Per consuetudine, i metodi accessori e modificatori hanno un prefisso standard e precisamente:

- get…() serve ad emettere il valore di un attributo
- set…() serve ad impostare il valore di un attributo

Nell'esempio della classe *Coppia*, il metodo

public int getX() esrestiruisce il valroe di x

public int getY() restituiscce il valore di y

Il metodo costruttore

Tra i metodi figurano i **costruttori**:

Coppia () // costruttore senza parametri

Coppia (**int** a, **int** b) // costruttore con parametri

I costruttori

- servono ad *inizializzare* i valori degli attributi al momento della creazione dell'oggetto;
- devono avere lo *stesso nome della classe*;
- non è previsto *alcun valore di ritorno*;
- possono essere *con parametri o senza*;
- possono essere *presenti in più forme nella stessa classe* (v. sopra)

Creazione di oggetti

La **creazione di un oggetto** di classe *Coppia* si effettua con l'istruzione

- Coppia c; // costruttore senza parametri
- Coppia c(a,b); // costruttore con parametri

che si dice **istanza** della classe e grazie al costruttore, mette a disposizione l'oggetto *c*.

Utilizzo di oggetti

L'oggetto *c:*

- *occupa memoria*, a differenza della classe, che è un modello astratto;
- consente l'accesso ad un membro con la notazione "." (*dot notation*) per cui
  - *c.getX()* attiva il metodo *getX*()
  - *c.getY()* attiva il metodo *getY*().

In generale l'attivazione di un metodo si esegue con la sintassi

oggetto.ident (*lista_param*)

Cosa ho imparato

- Cosa sono i metodi e gli attriibuti
- Cosa sono gli specificatori di accesso
- Come funziona una classe
- Cos'è un oggetto
- Cosa sono i metodi *get* e *set*

Cosa ho imparato a fare

- Come si dichiara una clasee
- Come si dichiarano gli attributi
- Come si dichiarano i metodi
- Come si definiscono i costruttori
- Come si crea un oggetto

Terminologia

- Metodi *get* e *set*
- Costruttore
- Specificatori di accesso
- Astrazione
- Implementazione

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale 1, ed. Minerva Italica
- M.Romagnoli, P.Ventura – Linguaggio C/C++, ed. Petrini
- M. Bigatti – Il linguaggio Java, ed HOEPLI