Programmazione a oggetti in C++

Approccio procedurale

```
struct Giocatore {
    int viteResidue, munizioni;
    int coordX, coordY;
}
void muovi( Giocatore* g, int direz ) {
  switch( direx ) {
    case 0: g->coordY -= 10; break; //SU
    case 1: g->coordY += 10;  break; //GIU
    case 2: g->coordX += 10; break; //DX
    case 3: g->coordX -= 10; break; //SX
  }
  if ( g->coordX <0 ) g->coordX=0;
  if ( g->coordY <0 ) g->coordY=0;
}
void spara( Giocatore* g) {
   if (g->munizioni > 0 )
       g->munizioni --;
}
```

Secondo il paradigma della programmazione procedurale un programma è composto da:

STRUTTURE DATI

FUNZIONI (procedure) che operano sui dati

Limiti della programmazione procedurale

- I programmi complessi sono sviluppati da team composti da più persone.
- Affinchè l'appicazione funzioni correttamente, spesso occore che le strutture dati (esempio "Giocatore") siano modificate solo attraverso le opportune funzioni.
- In C non esiste un modo semplice per impedire ai programamatori di modificare direttamente i campi delle strutture (anzichè modificarli indirettamente, tramite le funzioni)

Esempio

```
Giocatore g;
g.coordX = -1;
```

il Giocatore viene posizionato in una coordinata non valida. Ciò non sarebbe possibile se la variabile g potesse essere modificata solo attraverso la funzione *muovi*.

Il costrutto *class*

- Il problema precedete può essere risolto introducendo un nuovo costrutto (disponibile in C++ ma non in C): la classe.
- Una classe consente di "raggruppare" in un unico punto del programma:
  - la definizione della struttura dei dati
  - la definizione delle funzioni che operano sui dati

- Un programmatore C può pensare ad una classe come ad una struttura che, oltre a contenere i campi, contiene anche funzioni.
- Per le classi si usa un lessico specifico: i "campi" si chiamano **attributi** e le funzioni sono chiamate **metodi**.

Primo esempio: la classe Rettangolo

specificatori di accesso

```
class Rettangolo {
 private:                attributi (o campi)
   int altezza, larghezza;
 public:
    void setAltezza(int altezza) {
       if ( altezza >= 0 )
           this->altezza = altezza;
    }
    int getAltezza() {
       return altezza;
    }
    void setLarghezza(int l) { .... }
    int getLarghezza() { ... }
    int area() {
       return altezza*larghezza;
    }
};
```

metodi

r è un oggetto, ed è un'ISTANZA della classe Rettangolo.

```
int main() {
   Rettangolo r;
   r.altezza = 5;
   r.setAltezza(5);
   r.setLarghezza(3);
   cout << r.area() ;
}
```

Errore di compilazione: l'attributo *altezza* ha VISIBILITA' PRIVATA

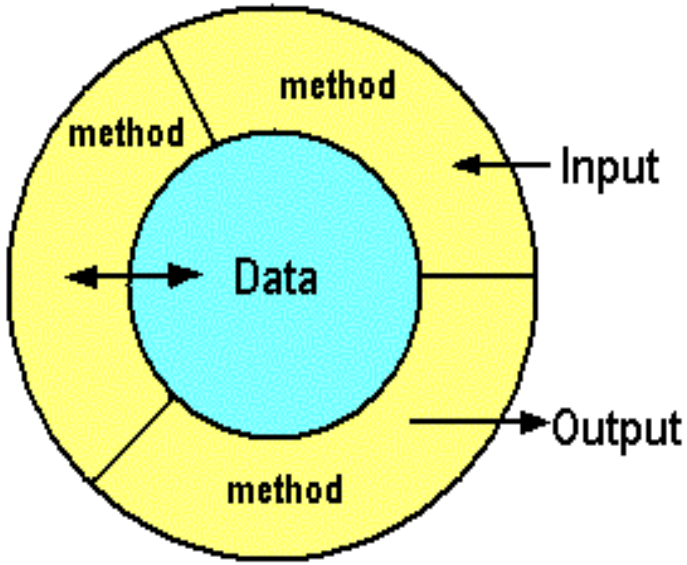
L'attributo altezza è modificabile solo attraverso al metodo setAltezza(...)
Così è impossibile impostare altezze negative

Incapsulamento / Information Hiding

E' buona regola impostare tutti gli attributi come privati.

I dati sono protetti da un "guscio" costituito dai metodi.

Questo concetto è noto come Incapsulamento / Information Hiding.



Metodi "setter" e "getter"

In genere, per ogni attributo privato modificabile dall'esterno della classe vengono definiti due metodi con questa forma:

- setNomeAttributo set = imposta
- getNomeAttributo get = ottieni

```
class Punto {
  private:
     int x, y;
  public:

     void setX(int x) {   this->x = x;  }

     int getX()  {  return x;  }
     ....
};
```

Un metodo può avere un argomento o una variabile locale con lo stesso nome di un attributo.
In questo caso si usa la parola chiave *this* per distinguere le due variabili.

x indica l'argomento del metodo
this->x indica l'attributo x della classe

Definizione esterna dei metodi

```
class Rettangolo {
  private:
     int altezza, larghezza;
  public:
     void setAltezza(int);
     int getAltezza();
     void setLarghezza(int);
     int getLarghezza();
     int area();
};

void Rettangolo::setAltezza(int altezza) {
    if ( altezza>0 )
      this->altezza = altezza;
}

int Rettangolo::getAltezza() {
     return altezza;
}
// ecc...
```

E' possibile scrivere al'interno della classe solo il prototipo dei metodi ...

... e specificare l'implementazione dei metodi all'esterno della classe:

NomeClasse::nomeMetodo

Costruttori e distruttori

```
class Rettangolo {
  private:
    int altezza, larghezza;
  public:
    Rettangolo() {
      altezza = rand()%100;
      larghezza = rand()%100;
    }
    Rettangolo(int a, int l) {
       altezza = a;
       larghezza = l;
    }
    ~Rettangolo() {
        // libero eventuali risorse
        // occupate dall'oggetto
    }
};
```

Il costruttore è una funzione con lo stesso nome della classe e senza tipo di ritorno.
Si possono definire più costruttori (diversi per tipo e/o il numero dei parametri)

Il distruttore è sempre uno solo e non può avere parametri.
Il nome DEVE essere: ~NomeClasse

Esempio: costruttori e distruttori

```
int main() {
  Rettangolo rettA;   // OK. Viene invocato il costruttore con 0 parametri

  Rettangolo rettB(5,3);  // OK. Viene invocato il costruttore con 2 parametri

  Rettangolo rettC(4);  // ERRORE DI COMPILAZIONE: non esiste un costruttore
                        // con un solo parametro.

  test();   // al termine della funzione viene invocato il distruttore sulla variabile
            // locale r.

} // al termine del main viene invocato il distruttore di rettA e rettB

void test() {
  Rettangolo r(40,50);
}
```

Esercizio: implementare la classe Date

```
class Date {
  private:
     int year, month, day;
  public:
     Date( int y, int m, int d) {
        year = y;  // equivale a:   this->year = y;
        month = m;
        day = d;
     }
     void addDays(int days) { .... }
     void print() {  ... }
     int DaysFrom(Date d) { ... }
     int compareTo(Date d) { ... }

};
```