Linguaggio C++

Programmazione ad oggetti

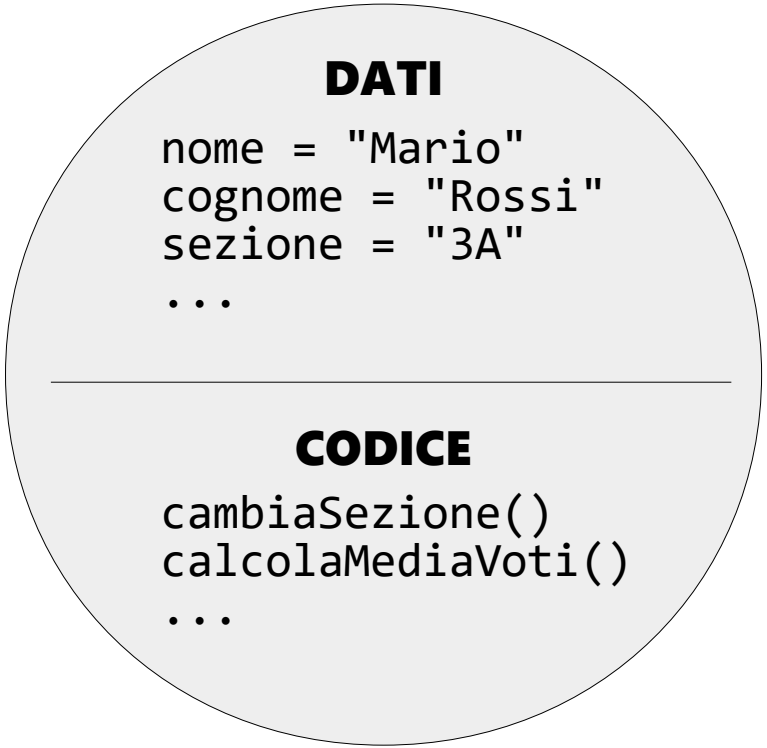
Prof. M. Camurri

La programmazione ad oggetti (OOP)

- La programmazione ad oggetti (**Object Oriented Programming**) è un paradigma di programmazione sviluppato a partire dagli anni '60, e basato sul concetto di **oggetto** software.
- Intuitivamente, possiamo definire un oggetto come un "contenitore" in grado di contenere sia **dati** che **codice**.
- In particolare, un oggetto è costituito da:
  - un insieme di dati
  - un insieme di "funzioni" che operano sui quei dati (dette *metodi*)

Oggetto = dati + codice

- Esempio: ggetto software che rappresenta uno studente



Un oggetto è l'unione di dati e funzioni che operano su quei dati

Dalla programmazione procedurale alla OOP

- La programmazione ad oggetti è stata introdotta per superare i limiti della programmazione procedurale (vedi slide successive).
- La programmazione procedurale non è altro che la programmazione basata su strutture dati e funzioni, a cui siamo già abituati. Esempio:

```
struct rettangolo {
  float larghezza;
  float altezza;
};
float calcolaArea(rettangolo r) {
    return (r.larghezza * r.altezza);
}
void stampa(rettangolo r) {
    cout << r.larghezza << "x" << r.altezza;
}
```

Esempio di programma procedurale

```
struct rettangolo {
  float larghezza;
  float altezza;
};
float calcolaArea(rettangolo r) {
    return (r.larghezza * r.altezza);
}
void stampa(rettangolo r) {
    cout << r.larghezza << "x" << r.altezza;
}
int main() {
    rettangolo r;
    r.larghezza = 5;
    r.altezza = 6;
    stampa(r);
    cout << calcolaArea(r);
}
```

Lo stesso programma ... a oggetti!

```
class rettangolo {
  public:
    float larghezza;
    float altezza;
    float calcolaArea() {
        return (larghezza * altezza);
    }
    void stampa() {
        cout << larghezza << "x" << altezza;
    }
};
int main() {
    rettangolo r;
    r.larghezza = 5;
    r.altezza = 6;
    r.stampa();
    cout << r.calcolaArea();
}
```

Limiti della programmazione procedurale

- Nella program. procedurale dati e funzioni non sono legati tra loro in modo "stretto". E' possibile agire direttamente sui dati anche senza utilizzare le funzioni.
- Ad esempio, potrei scrivere nel main:

```
rettangolo r;
r.altezza = -5;  // valore privo di significato!
r.larghezza = -6; // valore privo di significato!
```

- Avendo inserito dati non validi, anche le funzioni che operano su questi dati potrebbero restituire risultati privi di significato, o generare errori inattesi!
- Il problema potrebbe sembrare poco grave, ma bisogna ricordare che, nei progetti reali, molto spesso il programmatore si trova a lavorare su strutture dati complesse, definite da altri. Potrebbe non essere immediato capire quale modifiche sono "dannose" e quali sono lecite. Con la programmazione procedurale manca un meccanismo semplice per fare in modo che sia IMPOSSIBILE impostare valori errati.

Limiti della programmazione procedurale

- Vediamo come è possibile superare il limite della prog. procedurale appena descritto, utilizzando gli strumenti della programmazione ad oggetti
- Risolveremo il problema in due passi:
  - Inizialmente renderemo gli attributi "privati", in modo che non più possibile accedere direttamente a questi attributi da codice esterno alla classe.
  - Poi introdurremo due speciali metodi, chiamati setAltezza e setLarghezza, che ci consentiranno di modificare gli attributi in modo "controllato"

Passo 1: attributi privati

- Rendendo gli attributi privati, questi non sono più accessibili (nè in lettura nè in scrittura) dall'esterno della classe.

```
class rettangolo {
  private:
    float larghezza;
    float altezza;
  public:
     float calcolaArea() {
         return larghezza*altezza;  // OK
     }
};
int main() {
    rettangolo r;
    r.larghezza = 5;  // ERRORE COMPILAZIONE
    cout << r.larghezza;  // ERRORE COMPILAZIONE
}
```

Passo 2: prima definizione dei metodi "setter"

- Introduciamo due metodi che ci consentano di impostare il valore degli attributi.

```
class rettangolo {
  private:
    float larghezza;
    float altezza;
  public:
     float calcolaArea() {
         return larghezza*altezza;
     }
     void setLarghezza(float l) {
         larghezza = l;
     }
     void setAltezza(float a) {
         altezza = a;
     }
};
```

Utilizzo dei metodi "setter"

- Ora nel main possiamo modificare gli attributi richiamando i metodi "setter".

```
int main() {
    rettangolo r;
    r.setAltezza(5);
    r.setLarghezza(6);
    cout << r.calcolaArea();
}
```

- Non abbiamo però ancora risolto il nostro problema, poichè posso ancora scrivere:

```
r.setAltezza(-5);
```

- Come fare? Soluzione nella pagina seguente....

Definizione migliore dei metodi "setter"

- L'introduzione di attributi privati e metodi setter ci ha permesso di raggiungere un importantissimo risultato: ora l'unico modo per modificare il valore di altezza e larghezza è "passare" attraverso i metodi setAltezza e setLarghezza. Poichè i metodi sono porzioni di codice, è possibile eseguire controlli di validità prima di modificare i dati.
- Ad esempio, il metodo setAltezza si potrebbe riscrivere in questo modo:

```
void setAltezza(float a) {
    if (a>0) {
        altezza = a;
    }
    else {
        cout << "Errore";
    }
}
```