UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Programmazione ad Oggetti

Carmine Dodaro

Anno Accademico 2019/2020

Riepilogo prima lezione

Cos'è la programmazione ad oggetti

È un paradigma di programmazione basato sul concetto di oggetto. Un oggetto può contenere dati, di solito chiamati attributi, e codice sotto forma di metodi.

I concetti fondamentali

- Incapsulamento
- Composizione
- Ereditarietà
- Polimorfismo

Riepilogo prima lezione

Cos'è la programmazione ad oggetti

È un paradigma di programmazione basato sul concetto di oggetto. Un oggetto può contenere dati, di solito chiamati attributi, e codice sotto forma di metodi.

I concetti fondamentali

- ~~Incapsulamento~~
- Composizione
- Ereditarietà
- Polimorfismo

Composizione

Cos'è

È naturale pensare ad oggetti che contengono altri oggetti. Ad esempio un'automobile è composta da un motore, delle ruote, ecc. Per composizione quindi si intende una classe che contiene uno o più istanze di oggetti di altre classi.

Relazione in caso di composizione

Una relazione di composizione tra due oggetti è in genere riferita utilizzando il termine ha. Ad esempio, un'automobile ha un motore, ha delle ruote, ecc.

Negli esempi che abbiamo visto in laboratorio, un supermercato ha una lista di prodotti.

Ereditarietà

Cos'è

È uno dei concetti principali della programmazione ad oggetti che permette di definire le relazioni tra classi e che facilita, non solo il riuso del codice, ma anche una progettazione migliore delle classi.

L'ereditarietà permette ad una classe di ereditare, cioè acquisire, gli attributi e i metodi di un'altra classe. Questo permette di creare nuove classi che condividono gli stessi attributi e comportamenti.

Tipicamente, quando si progettano delle classi è opportuno valutare se condividono attributi o comportamenti. Ad esempio, supponiamo di voler progettare le classi Studente e Professore.

Ereditarietà

Professore.h

```
class Professore {
  public:
    Professore();
    string getCodiceFiscale() const;
    string getNome() const;
    string getCognome() const;
    void setCodiceFiscale(string);
    void setNome(string);
    void setCognome(string);

    float getStipendio() const;
    void setStipendio(float);
  private:
    string codiceFiscale;
    string nome;
    string cognome;
    float stipendio;
};
```

Studente.h

```
class Studente {
  public:
    Studente();
    string getCodiceFiscale() const;
    string getNome() const;
    string getCognome() const;
    void setCodiceFiscale(string);
    void setNome(string);
    void setCognome(string);

    float getIsee() const;
    void setIsee(float);
  private:
    string codiceFiscale;
    string nome;
    string cognome;
    float isee;
};
```

Ereditarietà

Base comune

Le classi Professore e Studente condividono sia campi e sia metodi. I campi codiceFiscale, nome e cognome sono in comune perché entrambi professori e studenti condividono il fatto di essere persone.

Persona.h

```
class Persona {
  public:
    Persona();
    string getCodiceFiscale() const;
    string getNome() const;
    string getCognome() const;
    void setCodiceFiscale(string);
    void setNome(string);
    void setCognome(string);

  private:
    string codiceFiscale;
    string nome;
    string cognome;
};
```

Ereditarietà

Professore.h

```
class Professore : public Persona {
  public:
    Professore();

    float getStipendio() const;
    void setStipendio(float);
  private:
    float stipendio;
};
```

Studente.h

```
class Studente : public Persona {
  public:
    Studente();

    float getIsee() const;
    void setIsee(float);
  private:
    float isee;
};
```

Le classi Professore e Studente ereditano da Persona. Questo significa che sugli oggetti delle classi Professore e Studente possiamo utilizzare i metodi della classe Persona.

La classe Persona è detta superclasse, o classe genitore, o classe base.

Le classi Studente e Professore sono dette sottoclasse, o classi figlie, o classi derivate.

Tipi di ereditarietà

Differenze

In C++ abbiamo tre tipi di ereditarietà:

- public

```
class Derived1: public Base {
...
};
```

- protected

```
class Derived2: protected Base {
...
};
```

- private

```
class Derived3: private Base {
...
};
```

Tabella riassuntiva

Visibilità classe Base		Tipo di ereditarietà		
		Public	Protected	Private
	Public	Public nella classe derivata.	Protected nella classe derivata.	Private nella classe derivata.
	Protected	Protected nella classe derivata.	Protected nella classe derivata.	Private nella classe derivata.
	Private	Non visibile nella classe derivata.	Non visibile nella classe derivata.	Non visibile nella classe derivata.

- class B : public A; tutto ciò che è public in A è public in B, tutto ciò che è protected in A è protected in B
- class B : protected A; tutto ciò che è public in A diventa protected in B, tutto ciò che è protected in A rimane protected in B
- class B : private A; tutto ciò che è public e protected in A diventa private in B

Tutto ciò che è private in A non è visibile da B.

Ereditarietà multipla

Relazione in caso di ereditarietà public

Una relazione di ereditarietà tra due oggetti è riferita utilizzando il termine è. Quindi uno studente è una persona, un professore è una persona, ecc.

Ereditarietà multipla

In C++ è possibile definire anche classi che ereditano da più classi. Ad esempio la classe Professore può ereditare sia dalla classe Dipendente e sia dalla classe Scienziato.

```
class Professore : public Dipendente, public Scienziato {
  ...
};
```

Un professore è un dipendente ed è uno scienziato.

Ereditarietà e conversione di tipo

```
// 1) Si può fare nel caso in cui Studente erediti da Persona in
     modo public
Persona* p  = new Studente();
// 2) Si può fare nel caso in cui Studente erediti da Persona in
     modo public
list<Persona*> l;
Studente* s = new Studente();
l.push_back(s);
// 3) In generale genera errore, uno studente è una persona, ma
    una persona NON è per forza uno studente
Studente* s = new Persona();
```

Ereditarietà e conversione di tipo

```
// 1) Si può fare nel caso in cui Studente erediti da Persona in
     modo public
Persona* p  = new Studente();
// 2) Si può fare nel caso in cui Studente erediti da Persona in
     modo public
list<Persona*> l;
Studente* s = new Studente();
l.push_back(s);
// 3) In generale genera errore, uno studente è una persona, ma
    una persona NON è per forza uno studente
Studente* s = new Persona();
```

Guardiamo un esempio pratico di utilizzo!