Le Classi

alcuni termini

- Astrazione
- Informaiton Hiding
- Encapsulation
- Gerarchie

Come definire una classe

```
class NomeClasse{
        specificatore1: membro1;
        specificatore2: membro2;
        ...

};
```

Specificatori di accesso

1. `public`
   liberamente visibili da tutti
2. `protected`
   come private ma visibile alle classi derivate

3. `private`
   Default per le classi
   visibili solamente all'interno della classe

Esempio di definizione

```
class Rectangle{
        // questi membri sono privatì
        int width, height;

        // tutti i prossimi membri avranno visibilità public
public:
        void set_width(int);    //
        void set_height(int);   // Setters

        int get_width();        //
        int get_height();       // Getters

        // quando i metodi vengono specificati dentro la classe viene detta dichiarazione **inli
        // viene di solito fatto per classi *banali*
        int area() { return width * height; }

}       // fine classe


// **Dichiarazione esterna**
// Definizione al di fuori dalla classe
// i "::" è l'operatore di scope che va a specificare "dove" ci troviamo
void Rectangle::set_width(int x){
        // quindi width fa parte dello scope corrente
        width = x;
}

int Rectangle::get_width(){ return width; }

// ... tutte le altre definizioni
```

Usare gli oggetti

```
int main(){
        Rectangle rect, rectb;

        rect.set_width(3);
        rect.set_height(4);

        rectb.set_width(6);
        rectb.set_height(7);

        std::cout << rect.area() << std::endl; // 12
        std::cout << rectb.area() << std::endl; // 42

}
```

Costruttore

```
class Rectangle{
        // definizione di prima ...

        //il nome deve essere esattamente quello della classe
public:
        Rectangle(int,int)
        Rectangle(); // costruttore di Default
};

// il costruttore non è un metodo come gli altri,
// può essere chiamato solo quando andiamo ad inizializzare un oggetto
// e non durante la sua vita
Rectangle::Rectangle(int x, int y){
        width = x;
        height = y;
}

Rectangle::Rectangle(){
        width = 5;
        height = 5;
}


int main()
{

        Rectangle rect;
        cout << rect.area() << endl;
        // non darà errore, ma il tuo comportamento è indefinito

        // andiamo a chiamare il costruttore appena definito
        Rectangle recta(3, 6);
        Rectangle rectb(6, 15);
        cout << recta << " " << rectb << endl;


        // questo non sarà possibile una volta dichiarato il costruttore,
        // se non è stato dichiarato un costruttore di default
        // cioè un costruttore senza parametri
        Rectangle rectc;
}
```

Vari modi per inizializzare una classe

```
// se la classe ha solo 1 elemento da inizializzare
NomeClasse nomeOggetto = valore;

// inizializzazione uniforme
NomeClasse nomeOggetto {valore1, valore2, ...}

class Circle{
        double radius;
public:
        Circle(duble r){ radius = r; }
        double circum(){ return 2 * r * 3.14; }
};

int main(){
        // inizializzazione funzionale
        Circle foo(10);

        //altri modi per inizializzare gli oggetti
        Circle bar = 20;
        Circle baz {30};
        Circle qux = {40};

        Rectangle rectb; // si

        // NON E' UN'INIZIALIZZAZIONE,
        // VIENE INTERPRETATA COME DEFINIZIONE DI FUNZIONE
        Rectangle rectc();

        Rectangle rectd {};  // si
}



Rectangle::Rectangle(int x, int y){
        width = x;
        height = y;
}
```

OPPURE (pre-inizializzazione)

```
Rectangle::Rectangle(int x, int y) : width(x), height(y) {}
```

Se devo inizializzare degli oggetti che fanno parte della mia definizione di classe questo metodo è l'unico modo per chiamare il costruttore dell'oggetto che sta nella classe

Classi all'interno di classi

```
class Circle{
        double radius;
public:
        Circle(duble r){ radius = r; }
        double circum(){ return 2 * radius * 3.14; }
        double area(){ return radius * radius * 3.14; }

};

class Cylinder{
        Circle base;
        double height;
public:
        Cylinder(double r, double h): base(r), height(h) {}
        double volume(){ return base.area() * height; }

}
```