UNIVERSITÀ
DI PARMA

Informatica e Laboratorio di Programmazione

C++ Object Oriented Programming

Alberto Ferrari

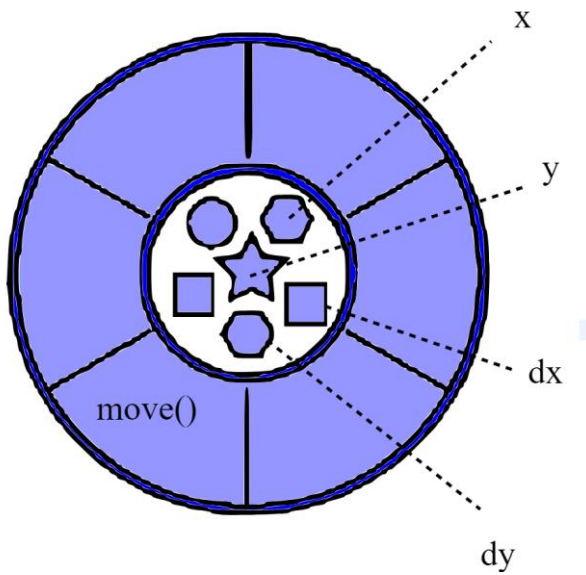
- in C++ la ***definizione*** della ***classe*** è ***separata*** dalla ***implementazione*** dei ***metodi***
  - definizione fornita agli utenti
  - implementazione compilata in libreria
- sorgenti organizzati in ***3 file***:
  - Ball.**hpp** – ***definizione*** della ***classe*** (*in alternativa Ball.**h***)
  - Ball.**cpp** – ***implementazione*** dei ***metodi***
  - **main.cpp** – ***applicazione*** che usa la classe
- è possibile inserire in un unico file la definizione e l'implementazione
  - Ball.**cpp** – definizione della classe e implementazione dei metodi

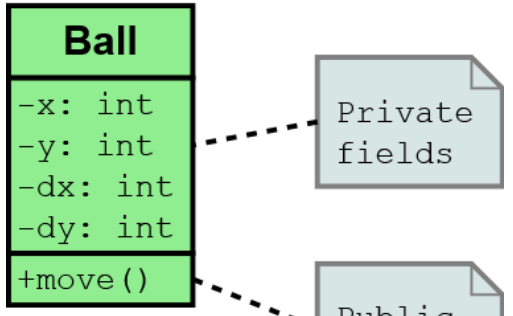
metodo costruttore

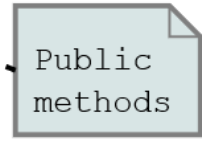
- il ***costruttore*** è un metodo particolare che viene ***invocato*** alla ***creazione*** dell'oggetto e che contiene tutte le ***istruzioni*** da eseguire per la sua ***inizializzazione***
- deve avere lo ***stesso nome*** della ***classe*** e non può ritornare un valore
- deve stare nella sezione pubblica della classe
- spesso si hanno più costruttori (overloading)
- un costruttore ***senza argomenti*** è detto costruttore di ***default***
  - se non definiamo nessun costruttore viene creato un costruttore di default
  - se definiamo almeno un costruttore il costruttore di default non viene creato
  - è bene includere sempre il costruttore di default

definizione Ball.hpp

```
#ifndef BALL_HPP
#define BALL_HPP

class Ball
{
    public:
        Ball(int x0, int y0);
        void move();
        int get_x();
        int get_y();

        static const int ARENA_W = 320;
        static const int ARENA_H = 240;
        static const int W = 20;
        static const int H = 20;

    private:
        int x;
        int y;
        int dx;
        int dy;
};
#endif // BALL_HPP
```

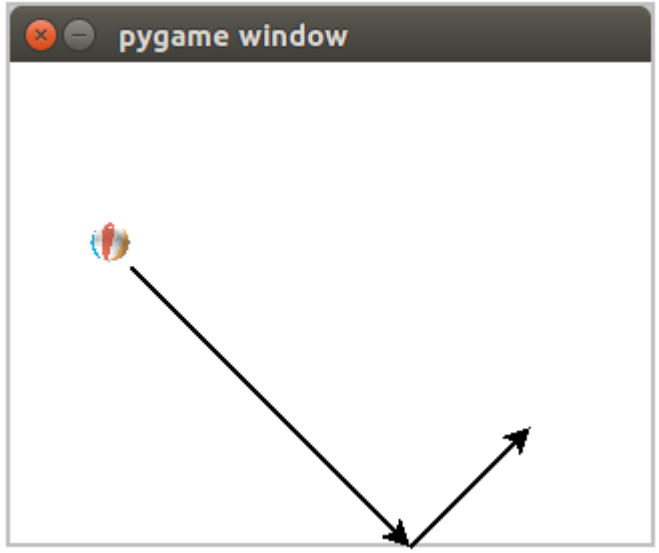






implementazione: Ball.cpp

```
#include "Ball.hpp"

Ball::Ball(int x0, int y0) {
    x = x0; y = y0; dx = 5; dy = 5;
}
void Ball::move() {
    if (!(0 <= x + dx && x + dx <= ARENA_W - W)) dx = -dx;
    if (!(0 <= y + dy && y + dy <= ARENA_H - H)) dy = -dy;
    x += dx; y += dy;
}
int Ball::get_x() {
    return x;
}
int Ball::get_y() {
    return y;
}
```



applicazione: main.cpp

```
#include <iostream>
#include "Ball.hpp"

using namespace std;

int main() {
    Ball b1{40, 80};
    Ball b2{80, 40};

    for (auto i = 0; i < 25; ++i) {
        b1.move();
        b2.move();
        cout << b1.get_x() << ", " << b1.get_y() << endl;
        cout << b2.get_x() << ", " << b2.get_y() << endl << endl;
    }
}
```

```
Ball b1{40, 80};
Ball* b2 = new Ball{80, 40};
// Ball* alias1 = &b1; // no new ball is created
// Ball* alias2 = b2;  // no new ball is created

for (auto i = 0; i < 25; ++i) {
    b1.move();
    b2->move();
    cout << b1.get_x() << ", " << b1.get_y() << endl;
    cout << b2->get_x() << ", " << b2->get_y() << endl << endl;
}
delete b2;
```

no garbage collection: a new deve corrispondere delete

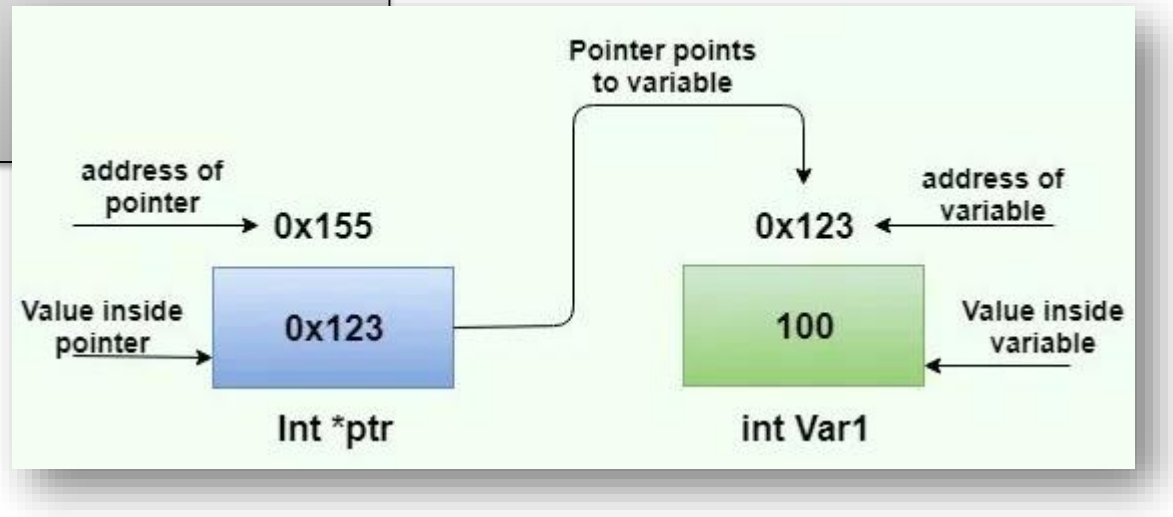
garbage collector

- per **garbage collection** *(letteralmente raccolta dei rifiuti)* si intende una modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, **liberano le porzioni di memoria** che **non** dovranno più essere successivamente **utilizzate** dalle applicazioni

Wikipedia

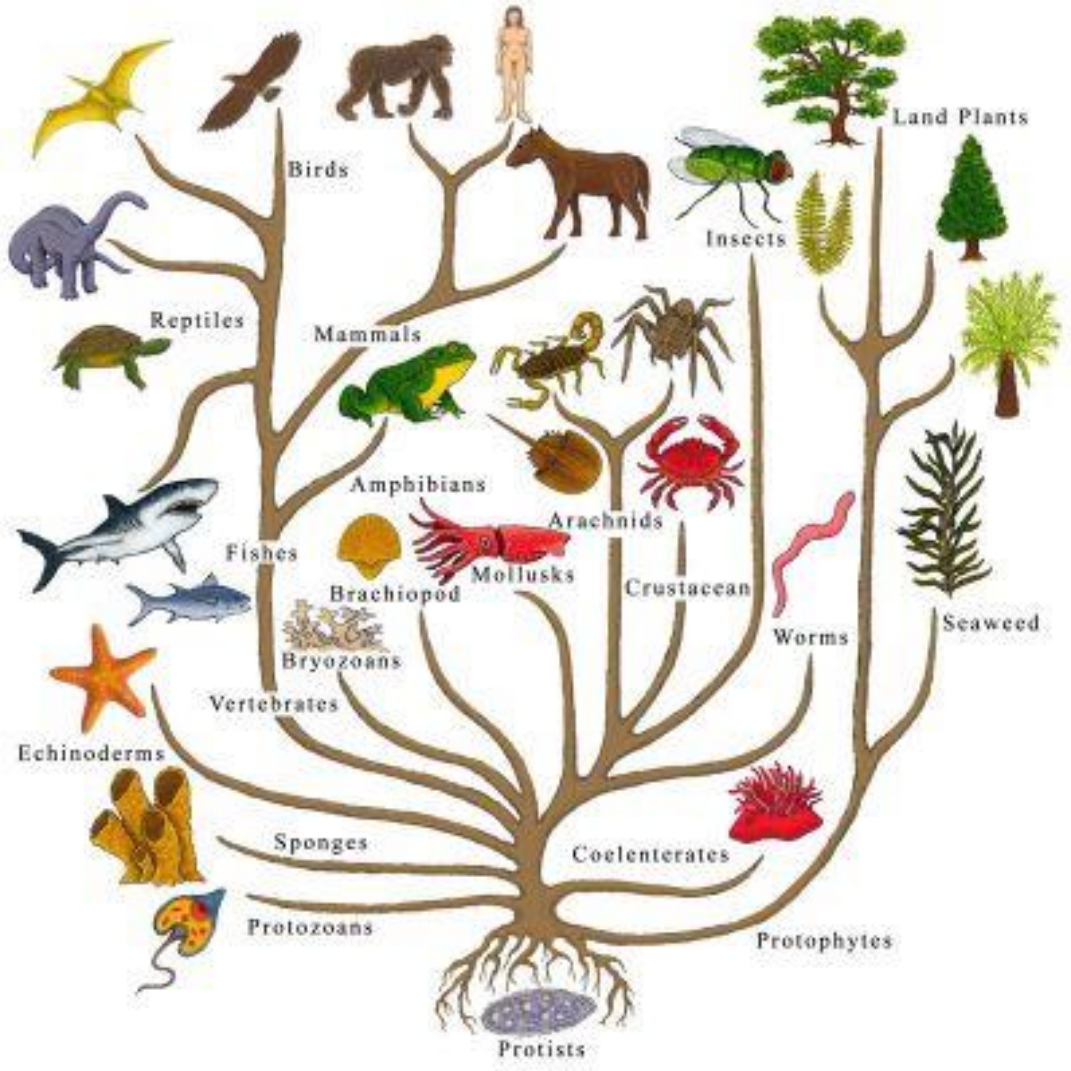
- ogni dato presente in memoria ha un ***indirizzo***
  - una variabile ***puntatore*** contiene un indirizzo
- operatore **&** per ***indirizzo*** di un dato
- operatore ***** per ***accesso*** a dato puntato (***dereferenziazione***)

```
char i = 56;   // a byte
char* p;       // a ptr to some byte (uninitialized)
p = &i;        // now p points to i
*p = *p + 1;   // ++i
p = nullptr;   // ptr to nothing
```



OOP

astrazione



esempio: fattoria parlante

- ***classe base*** come ***interfaccia astratta***
- es. **`Animal`**
  - ***tutti*** fanno un verso (***interfaccia***)
  - ***ognuno*** lo fa in modo diverso (***polimorfismo***)

```
class Animal {
public:
    virtual void say() = 0;
};
```

classi concrete

```
class Dog : public Animal {
    string name_;
public:
    Dog(string name) { name_ = name; }
    void say() {
        cout << "I am " << name_ << " Dog. I say: WOOF!" << endl;
    }
};

class Cat : public Animal {
    string name_;
public:
    Cat(string name) { name_ = name; }
    void say() {
        cout << "I am " << name_ << " Cat. I say: MEOW!" << endl;
    }
};
```

lista di puntatori a oggetti

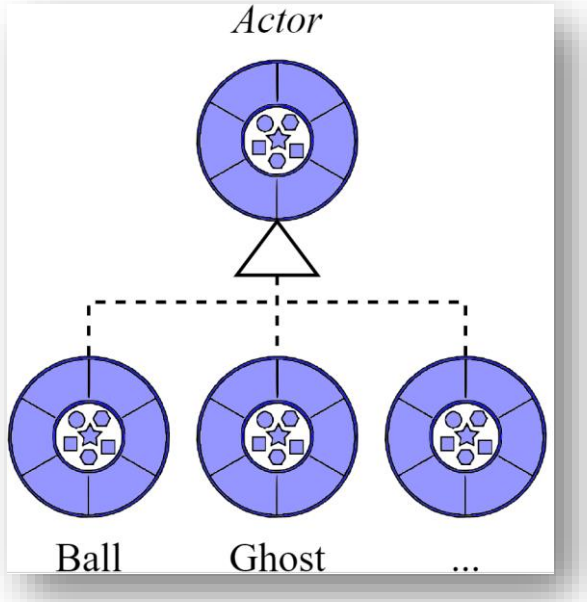
```
// a list of Animal objects
auto d = new Dog("Danny");
auto c = new Cat("Candy");
auto p1 = new Pig("Peppa");
auto p2 = new Pig("George");

vector<Animal*> animals = {d, c, p1, p2};

for (auto a : animals) {
    a->say();
}
```


```
I am Danny Dog. I say: WOOF!
I am Candy Cat. I say: MEOW!
I am Peppa Pig. I say: OINK!
I am George Pig. I say: OINK!
```

metodi virtuali puri

- esempio: metodo **`move`** di **`Actor`**
  - **`virtual`**: il metodo può essere ridefinito nelle sottoclassi (***polimorfo***)
  - = 0: il metodo ***non è implementato*** qui (la classe è ***astratta***)
- ***polimorfismo*** C++ funziona ***solo con puntatori a oggetti***

```
class Actor {
    virtual void move() = 0;
    // ...
};
```



Fifteen – gioco astratto

```
#ifndef BOARDGAME_HPP
#define BOARDGAME_HPP

#include <string>

class BoardGame
{
public:
    virtual void play_at(int x, int y) = 0;
    virtual void flag_at(int x, int y) = 0;
    virtual int cols() = 0;
    virtual int rows() = 0;
    virtual std::string get_val(int x, int y) = 0;
    virtual bool finished() = 0;
    virtual std::string message() = 0;

    virtual ~BoardGame() {}
};

#endif // BOARDGAME_HPP
```

gioco a console

```
void print_game(BoardGame* game) {
    for (auto y = 0; y < game->rows(); ++y) {
        for (auto x = 0; x < game->cols(); ++x) {
            cout << setw(3) << game->get_val(x, y);
        }
        cout << endl;
    }
}
void play_game(BoardGame* game) {
    print_game(game);
    while (! game->finished()) {
        int x, y; cin >> x >> y;
        game->play_at(x, y);
        print_game(game);
    }
    cout << game->message() << endl;
}
```