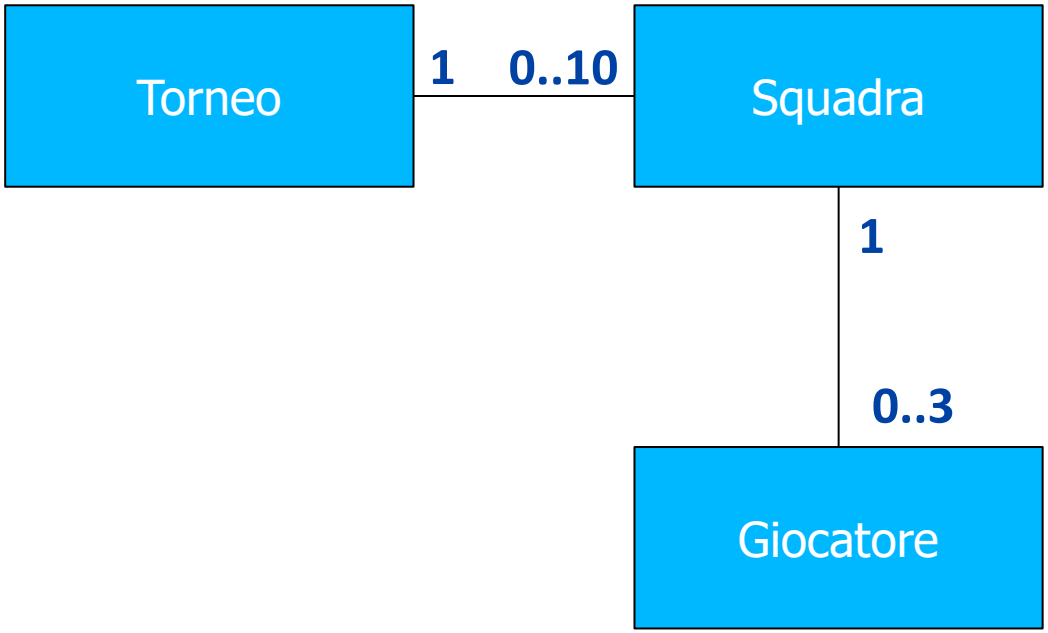
CLASSI ED OGGETTI

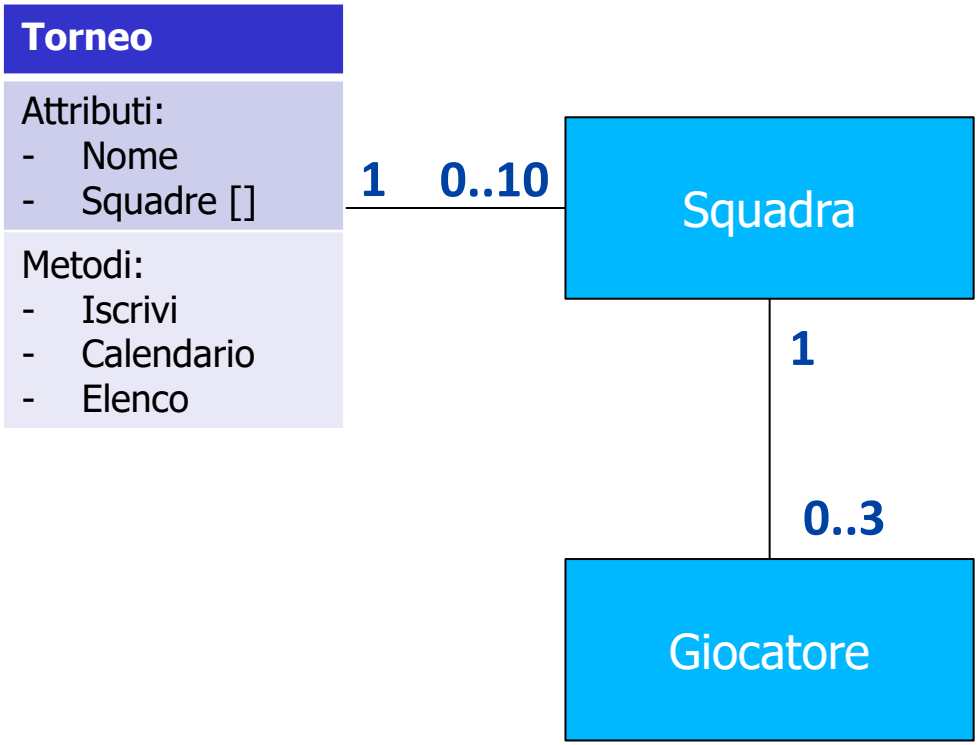
Object Oriented Programming

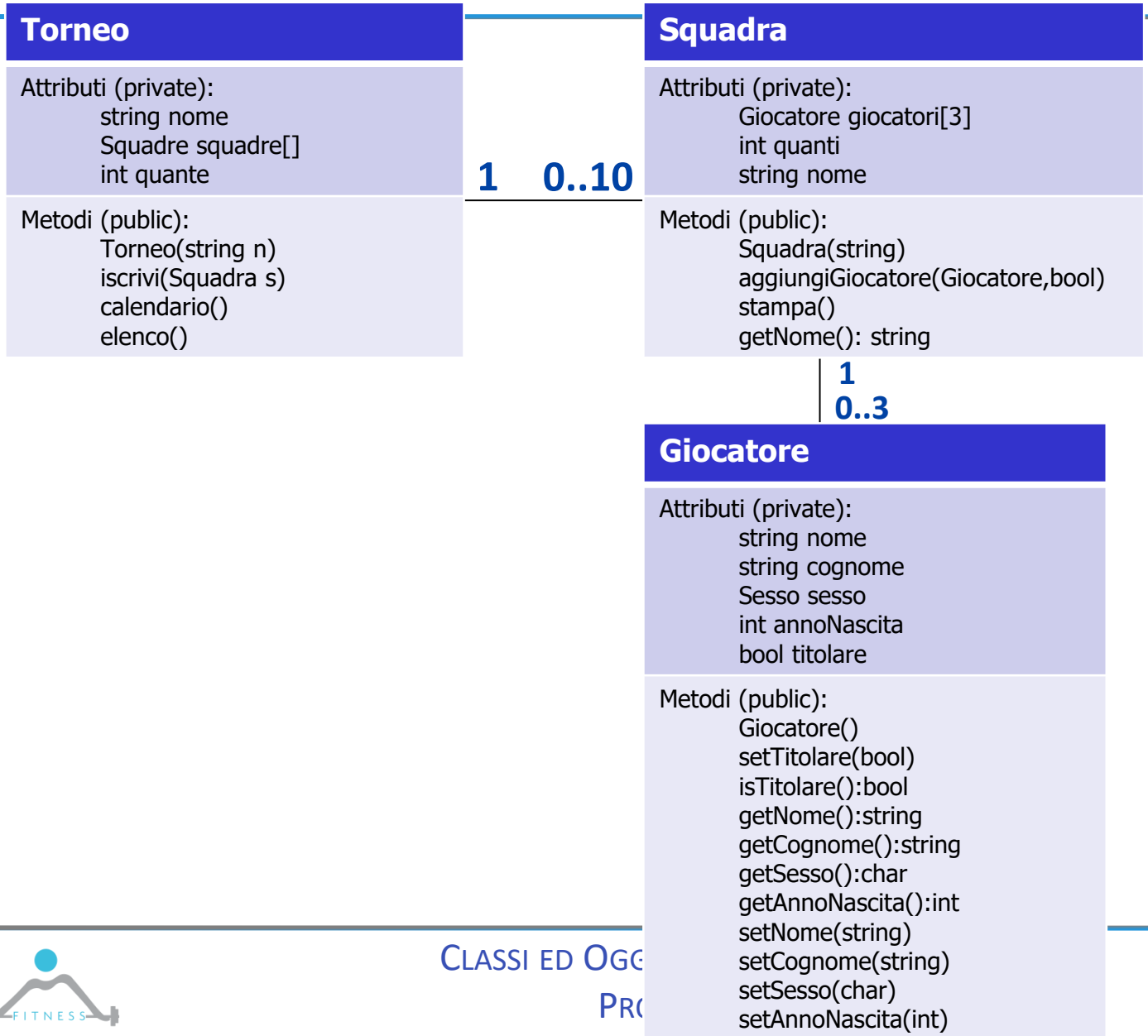
Roberto Nardone, Luigi Romano

ESERCIZI

- Creare un programma che consenta di gestire un torneo di beach-volley. Il torneo può coinvolgere un massimo di 10 squadre. E' possibile iscrivere una squadra al torneo, stampare l'elenco delle squadre e generare il calendario delle partite.
- Una squadra di beach-volley ha un nome ed è composta da 2 giocatori e 1 riserva. Per ogni giocatore bisogna ricordare nome, cognome, sesso (m/f), anno di nascita. Una squadra può essere inizialmente creata dandole un nome che non può essere poi modificato. Successivamente si possono aggiungere i giocatori specificando se un giocatore è titolare o panchina.
- Per una squadra è possibile stampare l'elenco dei giocatori ed in particolare nome, cognome, anno, ruolo del giocatore (titolare/panchina).


Torneo
1
0..10
Squadra
1
0..3
Giocatore


Torneo
Attributi:
- Nome
- Squadre []
Metodi:
- Iscrivi
- Calendario
- Elenco
1
0..10
Squadra
1
0..3
Giocatore


Torneo
Attributi (private):
string nome
Squadre squadre[]
int quante
Metodi (public):
Torneo(string n)
iscrivi(Squadra s)
calendario()
elenco()
1
0..10
Squadra
Attributi (private):
Giocatore giocatori[3]
int quanti
string nome
Metodi (public):
Squadra(string)
aggiungiGiocatore(Giocatore,bool)
stampa()
getNome(): string
1
0..3
Giocatore
Attributi (private):
string nome
string cognome
Sesso sesso
int annoNascita
bool titolare
Metodi (public):
Giocatore()
setTitolare(bool)
isTitolare():bool
getNome():string
getCognome():string
getSesso():char
getAnnoNascita():int
setNome(string)
setCognome(string)
setSesso(char)
setAnnoNascita(int)

SOVRACCARICAMENTO DI METODI

- Anche i metodi di una classe possono essere sovraccaricati (overloading)
  - Più versioni del metodo con liste di parametri diversi in tipo e/o numero
- Es.

```
class Persona {
   public:
     void invecchia(){eta++;}
     void invecchia(int anni){eta += anni;}
};
```

SOVRACCARICAMENTO - ESECUZIONE


```
void  invecchia () {eta++;};
void  invecchia (int anni) {eta+=anni;};

p.invecchia();
cout << "Eta': " << p.getEta() << endl;
p.invecchia(2);
cout << "Eta': " << p.getEta() << endl;
```

```
ubuntu@ubuntu:~/cppprogs$
ubuntu@ubuntu:~/cppprogs$ g++ -c Persona.cpp PersonaDemo.cpp
ubuntu@ubuntu:~/cppprogs$ g++ -o PersonaDemo.exe PersonaDemo.o Persona.o
ubuntu@ubuntu:~/cppprogs$ ./PersonaDemo.exe
Nome: Gennaro
Cognome: Esposito
Eta': 1
Eta': 3
ubuntu@ubuntu:~/cppprogs$
```

METODI CONST E ATTRIBUTI MUTABLE

- Un metodo `const` non può modificare gli attributi della classe cui appartiene
- Es.
  - `void Persona::presentati() const {}`
- Gli attributi dichiarati `mutable` possono essere modificati anche da metodi `const`
- Es.

```
class X  {
  public:
   bool GetFlag() const {
      m_accessCount++;
      return m_flag; }
  private:
   bool m_flag;
   mutable int m_accessCount;
};
```

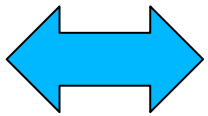
PUNTATORI A OGGETTI E OPERATORE ->

- E' possibile definire ed inizializzare un puntatore ad oggetto come un qualiasi altro puntatore.
- Es.

```
Persona* p = new Persona();
```

- Per accedere ai membri della classe attraverso un puntatore si può usare l'operatore ->
- Es.

```
(*p).invecchia();
(*p).eta=0;
```



```
p->invecchia();
P->eta=0;
```

PUNTATORE THIS

- Ogni oggetto ha un puntatore implicito (definito automaticamente dal compilatore) a se stesso denominato «this» che può essere usato internamente all'oggetto per far riferimento a se stesso
- Es.

```
class Persona {
    string nome;
    string cognome;
    int eta
  public:
      Persona();
      Persona(string nome, string cognome, int eta){
          this->nome=nome;
          this->cognome=cognome;
          this->eta=peta;
      }
};
```

ATTRIBUTI STATICI

- Sono attributi non associati a singoli oggetti ma all'intera class
  - Poiché non è allocata memoria al momento della dichiarazione è necessario definirla nell'ambito locale con operatore di visibilità ::
- Es.

```
class Pecora{
   public:
      static int n;
      Pecora(){n++;}
      ~Pecora(){n--;}
};
int Pecora::n = 0;
```

```
int main(){
Pecora p1;
  cout << "ci sono" <<
    Pecora::n << "pecore";// 1
  {
      Pecora p2;
      cout<<"Ora: "
        << Pecora::n;// 2
   }
   cout<<"Infine: "
        << Pecora::n; // 1
}
```

METODI STATICI

- Anche alcuni metodi potrebbero essere indipendenti dal singolo oggetto
  - Ad esempio per inizializzare attributi statici

```
class Pecora(){
   static int n;
 public:
   Pecora(){n++;}
   ~Pecora(){n--;}
   static void init(int n);
};
```

```
void Pecora::init(int n){
        Pecora::n=n;
}

int main(){
        Pecora::init(0);
        Pecora p1;
        …
}
```

BUONE PRASSI

- Metodi Setter e Getter:
  - Tenere privati gli attributi TUTTI anche quelli che danno informazioni utili all'esterno (es. `class Persona`, attributo `altezza`)
  - Per gli attributi che dovrebbero essere pubblici preparare
    - Delle funzioni per leggerne il valore (es. `int Persona::getAltezza()`): metodi getter
    - Delle funzioni per settarne il valore (es. `int Persona::setAltezza(int)`): metodi setter
- Per le classi usare nomi con iniziale Maiuscola (`class` **`P`**`ersona`)
- Per i metodi, se il nome si compone di più parole evidenziarle usando le maiuscole (`get`**`V`**`alore`**`A`**`ltezza`)

ESEMPIO COMPLETO

- Si crei una classe Regione che possa avere un certo numero di Città.
- Per le Città: si può definire il numero di abitanti, incrementarlo o decrementarlo, recuperarne il nome e il numero di abitanti
- Per le Regioni si può stampare l'elenco delle città o l'elenco delle città il cui nome inizi per una certa lettera assegnata, ogni volta il numero di abitanti è stampato a seguire il nome separato da virgola
- Il main che usa le classi, tra le altre istruzioni avrà:

```
Regione r("Campania");
r.aggiungi(c1);
r.aggiungi(c2);
…
Regione r1 = r;
r1.stampa();
```

CLASSE CITTÀ


```
class Citta{
    private:
        string nome;
        int abitanti;
    public:
        Citta(string nome="", int abitanti = 0): nome(nome),
                abitanti(abitanti) {}
        int getAbitanti(){return abitanti;}
        string getNome(){return nome;}
        void setNome(string n){nome=n;}
        void setAbitanti(int n){abitanti=n;}
        int aumentaAbitanti(int n){return abitanti+=n;} // ritorna
                                                        // numero aggiornato
        int diminuisciAbitanti(int n){return abitanti-=n;} // ritorna
                                                        // numero aggiornato
};
```

CLASSE REGIONE


```
class Regione{
     private:
          string nome;
          Citta* listaCitta;
          int cittaMAX; // quante città può gestire
          int numeroCitta; // quante città sono attualmente inserite
     public:
          Regione(string nome="", int quante = 10): nome(nome), cittaMAX(quante){
               listaCitta = new Citta[cittaMAX];
               numeroCitta = 0;
          }
          void setNome(string nome){this->nome = nome;}
          string getNome(){return nome;};
          void aggiungi(Citta c);
          void stampa();
          void stampa(char c);
          Regione(const Regione&);
          ~Regione(){delete[] listaCitta;}
};
```

```
Regione::Regione(const Regione& r){
     nome=r.nome;
     cittaMAX=r.cittaMAX;
     numeroCitta = r.numeroCitta;
     listaCitta = new Citta[cittaMAX];
     for(int i=0; i < numeroCitta; i++){
          listaCitta[i] = r.listaCitta[i];
     }
}
void Regione::aggiungi(Citta c){
     if (numeroCitta >= cittaMAX){ //se array pieno aggiungere altre 10 slot
          cittaMAX +=10;
          Citta* temp = new Citta[cittaMAX];
          for(int i=0; i <numeroCitta; i++) temp[i]=listaCitta[i];
          delete [] listaCitta;
          listaCitta = temp;
     }
     listaCitta[numeroCitta]=c;
     numeroCitta++;
}
```

```
void Regione::stampa(){

    for(int i = 0; i < numeroCitta; i++){
        cout << endl << listaCitta[i].getNome() << ", " <<
                    listaCitta[i].getAbitanti();
    }
}

void Regione::stampa(char c){
    string temp;

    for(int i = 0; i < numeroCitta; i++){
        temp = listaCitta[i].getNome();
        if (temp[0]==c)
            cout << endl << temp << ", " << listaCitta[i].getAbitanti();
    }
}
```

MAIN


```
int main(){ // Ridotto
    Citta c1;
    c1.setNome("Napoli");
    c1.setAbitanti(900);
    Citta c2("Benevento", 300);
    Citta c3("Nola",100);
    Regione r("Campania");
    r.aggiungi(c1);
    r.aggiungi(c2);
    r.aggiungi(c3);

    Regione r1=r;
    r1.stampa();
    cout << endl << "############################";
    r1.stampa('N');
    cout << endl;
}
```