UNIVERSITA' DEGLI STUDI ROMA TRE

DIPARTIMENTO DI FISICA "E. AMALDI"

Laboratorio di Programmazione e Calcolo

6 crediti

a cura di

Severino Bussino

Anno Accademico 2021-22

0) Struttura del Corso

1) Trattamento dell'informazione
Elementi di Architettura di un Computer

Verra' trattata in una delle prossime lezioni

2) Sistemi operativi

3) Introduzione alla
Programmazione ad oggetti (OO)

4) Simulazione del Sistema Solare

5) Introduzione al linguaggio C/C++

6) Elementi di linguaggio C/C++

A 1 - istruzioni e operatori booleani

2 - iterazioni (`for`, `while`, `do … while`)

B - istruzioni di selezione (`if`, `switch`, `else`)

C - funzioni predefinite. La classe `math`.

7) Puntatori

8) Vettori (Array)

9) Vettori e Puntatori

10) Classe SistemaSolare (prima parte)

11) Gestione dinamica della memoria

12) Classe SistemaSolare

13) Programma di Simulazione (main)

14) Ereditarieta'

- Concetto
- Funzione
- Realizzazione in C++
- Esempi

15) Classe Sonda

16) Output su file

17) Polimorfismo

- Concetto
- Funzione
- Realizzazione in C++
- Esempi

18) "per referenza" e "per valore"

19) Richiami sulle funzioni

20) Cenni alle classi Template

21) La libreria STL
(Standard Template Library)

22) L'algebra di una classe (cenni)

23) L'algebra di una classe (complementi)

24) Un esempio: la gestione dei dati nello studio delle oscillazioni del pendolo semplice

25) Un esempio: la descrizione dell'apparato ARGO-YBJ

Le lezioni in aula si concludono oggi 3 dicembre

Le Esercitazioni di Laboratorio proseguono senza variazioni con l'orario gia' comunicato (e pubblicato sul sito)

14 - 15 - 16 dicembre Simulazione Prova Individuale

11 - 12 - 13 gennaio Prova Individuale

Date degli Esami

I appello - Prova Pratica: 26 gennaio - Ore 14:30-17:30
Prova Orale: 31 gennaio - (Ore 14:00) stanza 80

II appello - Prova Pratica: 10 febbraio - Ore 14:30-17:30
Prova Orale: 17 febbraio - (Ore 14:00) aula 80

Chi non supera gli "esoneri" puo' sostenere un esame **pratico** e **orale**

Sostenendo l'esame si rinuncia pero' al voto ottenuto "in itinere"

Verbalizzazione
(per chi ha superato gli esoneri)

E' INDISPENSABILE iscriversi su GOMP all'esame del Corso per la prova Pratica **e** per la prova **Orale**

"Laboratorio di Programmazione e Calcolo"

(e **non** Laboratorio di Calcolo , ecc)
nella data in cui ci si presentera' per la verbalizzazione

La **verbalizzazione** avverra' nelle stesse date in cui e' prevista la prova Orale (**online su Team [meglio]** oppure in aula 80)

31 gennaio Ore 14:00-(18:00)

17 febbraio Ore 14:00-(18:00)

Se non potete verbalizzare in queste date e in questi Orari, per favore avvisate per E-mail

(sbussino@uniroma3.it **alessandro.dicicco@roma3.infn.it**)

22) Applicazioni

Applicazioni (Barone et al.)

- esempi di applicazioni sviluppate nel testo di Barone et al.:
  - Ricerca degli 0 di una funzione mediante il metodo di bisezione e mediante il metodo di Newton (4.3.2)
  - Ricerca dei numeri primi (4.3.3)
  - Problemi di arrotondamento (4.5)
  - Riordinamento degli elementi di un vettore: Bubblesort (5.2.1)
  - Ricerca binaria (5.2.2)
  - Soluzione di sistemi di equazioni lineari (5.4, 7.3.2)
  - Generazione di numeri casuali (5.5)
  - Manipolazione di testi (5.6.3, 6.5.1)
  - Istogrammi (7.4.1) – Visto
  - Calcolo del χ^2 di una distribuzione (7.4.2)
  - Calcolo di una derivata (7.6)
  - Interpolazione e integrazione numerica (cap. 8)

Applicazioni

- esempi di applicazioni sviluppate nel testo di Barone et al.:
  - Ricerca degli 0 di una funzione mediante il metodo di bisezione e mediante il metodo di Newton (4.3.2)
  - Ricerca dei numeri primi (4.3.3)
  - Problemi di arrotondamento (4.5)
  - Riordinamento degli elementi di un vettore: Bubblesort (5.2.1)
  - Ricerca binaria (5.2.2)
  - Soluzione di sistemi di equazioni lineari (5.4, 7.3.2)
  - Generazione di numeri casuali (5.5)
  - Manipolazione di testi (5.6.3, 6.5.1)
  - Istogrammi (7.4.1) – Visto
  - Calcolo del χ^2 di una distribuzione (7.4.2)
  - Calcolo di una derivata (7.6)
  - Interpolazione e integrazione numerica (cap. 8)

Sistemi "fisici" integrati durante il corso

Moto di un grave

Moto di un grave nell'aria

Moto di un grave in un liquido

Moto di un sistema a molti corpi (in due dimensioni) (forze gravitazionali)

Equazioni differenziali risolte numericamente durante il corso (1)

Moto di un grave

Moto di un grave nell'aria

Moto di un grave in un liquido

Moto di un sistema a molti corpi
(in due dimensioni) (forze gravitazionali)

$$\frac{d^2x}{dt^2} = -mg$$

$$\frac{d^2x}{dt^2} + k_1\frac{dx}{dt} = -mg$$

$$\frac{d^2x}{dt^2} + \left(k_1 + k_2\left|\frac{dx}{dt}\right|\right)\frac{dx}{dt} = F_A - mg$$

Equazioni differenziali risolte numericamente durante il corso (2)

Moto di un grave

Moto di un grave nell'aria

Moto di un grave in un liquido

Moto di un sistema a molti corpi (in due dimensioni) (forze gravitazionali)

$$\begin{cases} \dfrac{d^2 x_i}{dt^2} = -\displaystyle\sum_{i \neq j} G \frac{m_i m_j}{\left((x_i - x_j)^2 + (y_i - y_j)^2\right)^{\frac{3}{2}}} (x_i - x_j) \\ \dfrac{d^2 y_i}{dt^2} = -\displaystyle\sum_{i \neq j} G \frac{m_i m_j}{\left((x_i - x_j)^2 + (y_i - y_j)^2\right)^{\frac{3}{2}}} (y_i - y_j) \end{cases}$$

23) Un esempio: la gestione dei dati nello studio delle oscillazioni del pendolo semplice

ripasso!!!

Template, STL ...

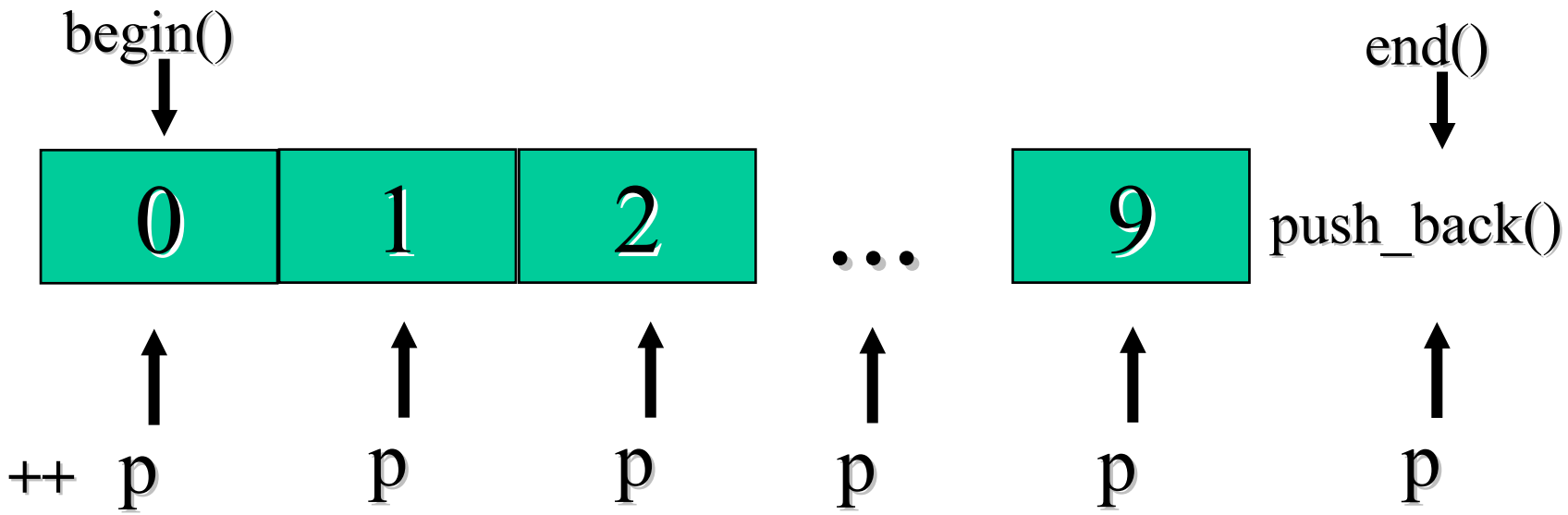
... i container STL

---

vector list map

vector: 1 - Come funziona

vector e' un array contiguo di oggetti



`push_back(value)` e' un metodo della classe vector per:

- aggiungere alla fine del vettore un nuovo elemento
- assegnare al nuovo elemento il valore `value`

vector: 2 - gli iteratori

Gli iteratori permettono di scorrere il container e si comportano come i puntatori

Se v e' un container di tipo vector

`v.begin()` - e' un iteratore
punta all'inizio del container

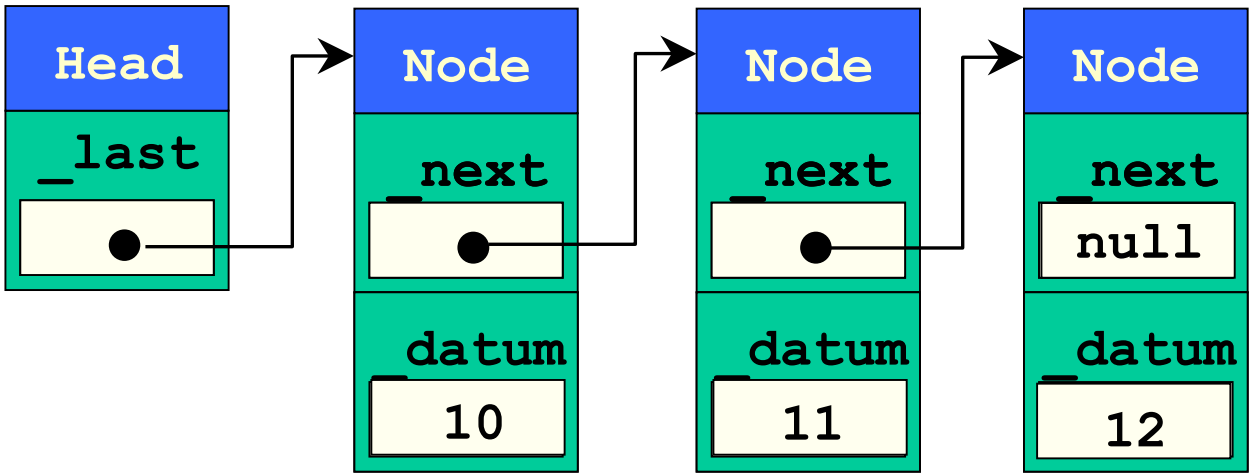
`v.end()` - e' un iteratore
punta alla prima locazione **dopo** la fine

```
vector<T>::const_iterator p ;
```

Si puo' usare il typedef

```
typedef vector<T>::const_iterator viter ;
```

List



Si utilizza come vector (sostituendo a `vector` la parola `list`)

Piu' efficiente nel gestire le inserzioni di elementi

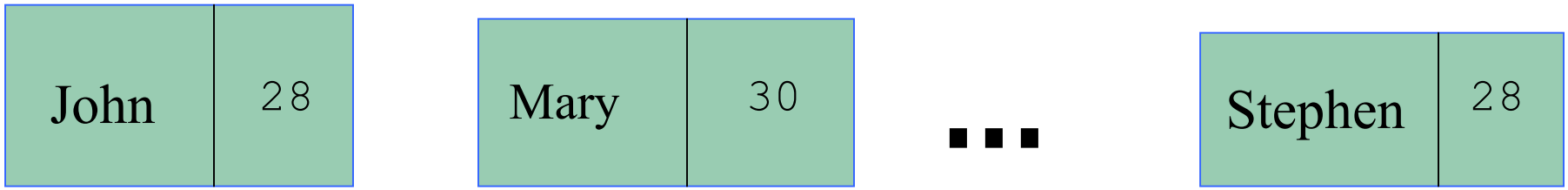
Map

Una "map" e' un container associativo costituito da coppie chiave-valore (key - T)

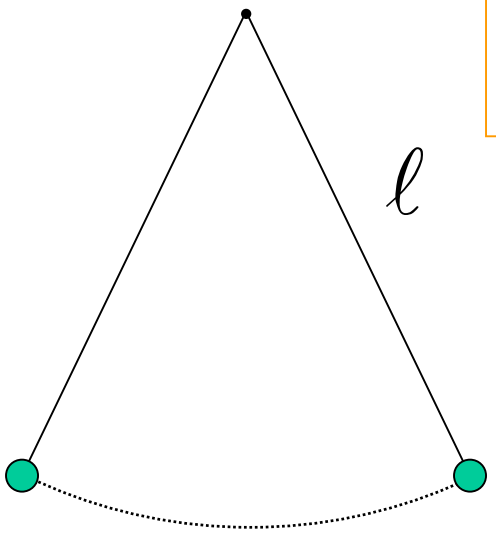
La map e' indicizzata rispetto alla chiave

Il valore della chiave deve essere univoco

Esempio: `map<string, int>`



Un esempio: il pendolo



- Seconda legge dinamica
- Forza peso
- Equazioni differenziali
- Sviluppo in serie di Taylor (Mac L
- ……

$$T = 2\pi\sqrt{\frac{\ell}{g}}$$

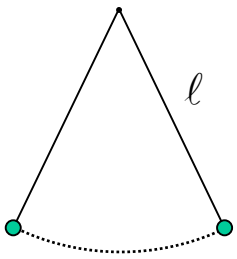
- Sara' vero?
- Posso misurare g

$$g = 4\pi^2 \frac{\ell}{T^2}$$

Come effettuo le misure ? (1)

- Con un metro **misuro** la lunghezza ℓ del pendolo
- Con un cronometro **misuro** il periodo T

- Per migliorare la qualita' delle misure del periodo T
  - lascio oscillare il pendolo per np periodi
  - misuro il tempo di oscillazione t per n periodi
  - il periodo T sara' dato da $T = \frac{t}{np}$



Per ciascun valore della lunghezza ℓ effettuo una serie n_ℓ di misure del periodo T

Come effettuo le misure ? (2)

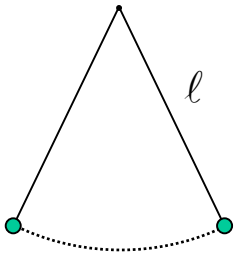
- Per verificare la relazione tra T ed ℓ

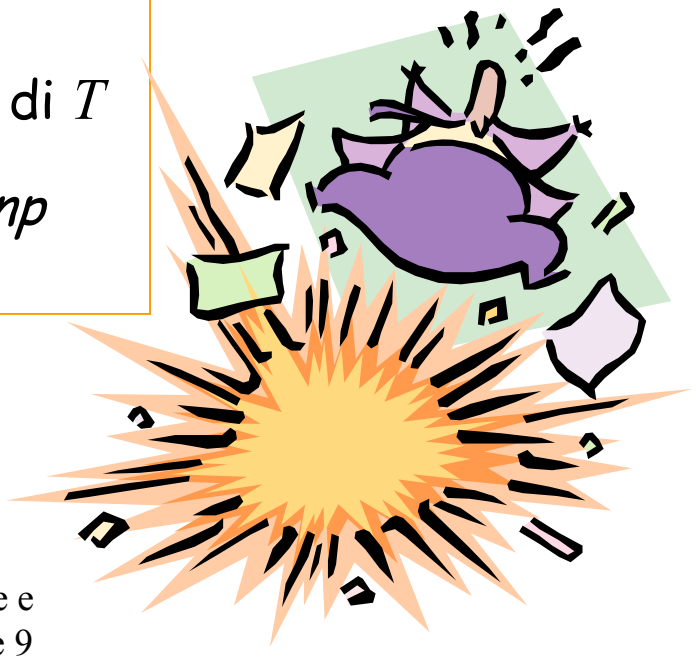
- Per migliorare la qualita' della misura di g

Effettuo una serie di misure per diversi valori di ℓ

Quindi ho:

- Una serie di misure a diversi valori di ℓ
- Per ogni valore di ℓ ho una serie di misure di T
- Ciascuna misura di T deriva dalla misura di *np* oscillazioni

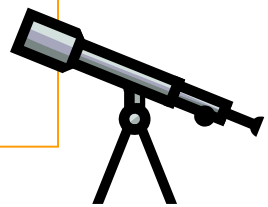




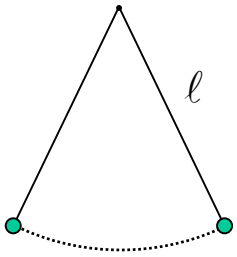
L'analisi delle misure

- Ho raccolto una grande quantita' di dato

- Voglio analizzarli!



- Un grafico di T in funzione di ℓ
- Un grafico di T^2 in funzione di ℓ
- Un valore di g che derivi dall'analisi di tutti i dati
- Se cambio il materiale... (massa inerziale e massa gravitazionale!)
- Se le oscillazioni non sono piccole (Taylor - Mac Laurin ...)



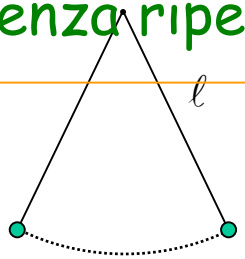
I/O (Input Output)

Input

- Dove scrivo i dati che ho raccolto?
- Dove li memorizzo per poterli analizzare?
- Se li voglio analizzare in un'altra seduta di laboratorio?
- Se voglio aggiungere dati raccolti in due esercitazioni diverse?

- Dove scrivo i risultati dell'analisi?
- Come mi interfaccio con un programma di grafica (ad esempio gnuplot) ?
- Posso combinare insieme i risultati di due analisi (senza ripetere l'analisi) ?

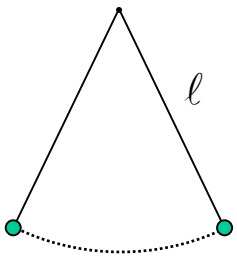
Output



Un approccio OO

- Quali sono gli oggetti?
- Come posso organizzare i dati?
- Di chi sono le responsabilita'?

- Chi sono gli attori?
- In che relazione stanno tra loro?
- Chi fa cosa?

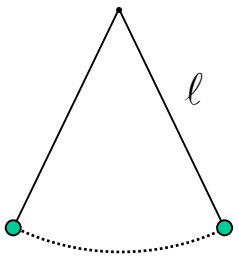


Quali sono gli oggetti?

I **dati** sperimentali!

Cioe' le misure che avete annotato sul quaderno!

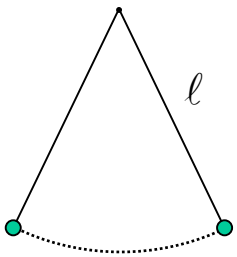
- `datiT` - un oggetto (e quindi una classe) con le misure relative al tempo impiegato a compiere np oscillazioni
- `SetdatiL` - un oggetto (e quindi una classe) con l'insieme delle misure relative a una o piu' lunghezze



La classe datiT (attributi)

datiT.h

```
#ifndef DATIT_H
#define DATIT_H

#include <iostream>
using namespace std ;

class datiT {

   protected:

    double misT   ;
    double d_misT ;
    int nmis      ;

                                  // continua
```



La classe datiT (metodi)

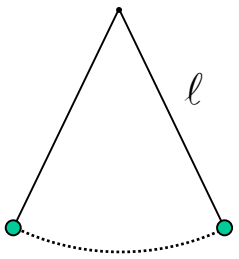
datiT.h

```
public:

    // costruttori
    datiT()  {  } ;
    datiT(double Tn, double dTn, int n) ;

    // distruttore
    ~datiT() {  } ;

    // metodi di tipo Get
    double Tmis()  const  { return misT   ; } ;
    double dTmis() const  { return d_misT ; } ;
    int Nmis()     const  { return nmis   ; } ;

    // altri metodi
    double  T() const ;
    double dT() const ;
                              // continua
```

La classe `datiT` (operatori)

datiT.h

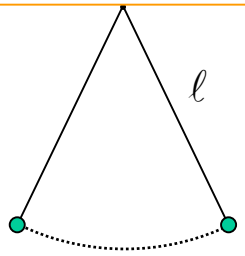
```
    // operatori booleani di confronto
    bool operator == (const datiT &d2) const ;
    bool operator != (const datiT &d2) const ;

    bool operator > (const datiT &d2)  const ;
    bool operator < (const datiT &d2)  const ;

    bool operator >= (const datiT &d2) const ;
    bool operator <= (const datiT &d2) const ;


} ;

#endif
```



L'implementazione della classe `datiT` (costruttore)

datiT.cc

```
#include "datiT.h"

    // costruttore
    datiT::datiT(double Tn, double dTn, int n)  {

      misT   =  Tn   ;
      d_misT =  dTn  ;
      nmis   =  n    ;

      if(nmis<=0) {
          cerr << "  datiT::datiT(...) Attenzione! nmis = "
               << nmis << endl ;
      }

    }
                                  // continua
```



L'implementazione della classe `datiT` (altri metodi)

datiT.cc

```
// altri metodi
double datiT::T() const  {

  if(nmis<=0) {
    cerr << " datiT::T() Attenzione! nmis = " << nmis << endl ;
  } else {
    return  misT/nmis ;
  }

}

double datiT::dT() const {

  if(nmis<=0) {
    cerr << " datiT::dT() Attenzione! nmis= " << nmis << endl ;
  } else {
    return  d_misT/nmis ;
  }

}
                    // continua
```

L'implementazione della classe datiT (operatori - 1)

datiT.cc

```
// operatori booleani di confronto
    bool datiT::operator == (const datiT &d2) const  {

      if( T() == d2.T() ) {
          return true  ;
      } else {
          return false ;
      }

    }

    bool datiT::operator != (const datiT &d2) const  {

      if( T() != d2.T() ) {
          return true  ;
      } else {
          return false ;
      }

    }

    bool datiT::operator > (const datiT &d2) const  {

      if( T() > d2.T() ) {
          return true  ;
      } else {
          return false ;
      }

    }                                          // continua
```

L'implementazione della classe `datiT` (operatori - 2)

```
bool datiT::operator < (const datiT &d2) const  {

  if( T() != d2.T() ) {
      return true  ;
  } else {
      return false ;
  }

}

bool datiT::operator >= (const datiT &d2) const  {

  if( T() >= d2.T() ) {
      return true  ;
  } else {
      return false ;
  }

}

bool datiT::operator <= (const datiT &d2) const {

  if( T() <= d2.T() ) {
      return true  ;
  } else {
      return false ;
  }

}                          // fine del file di implementazione datiT.cc
```

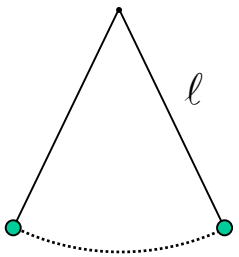
Come posso organizzare i dati?

Posso usare un container STL !

- per ogni lunghezza L ho una serie di misure di T

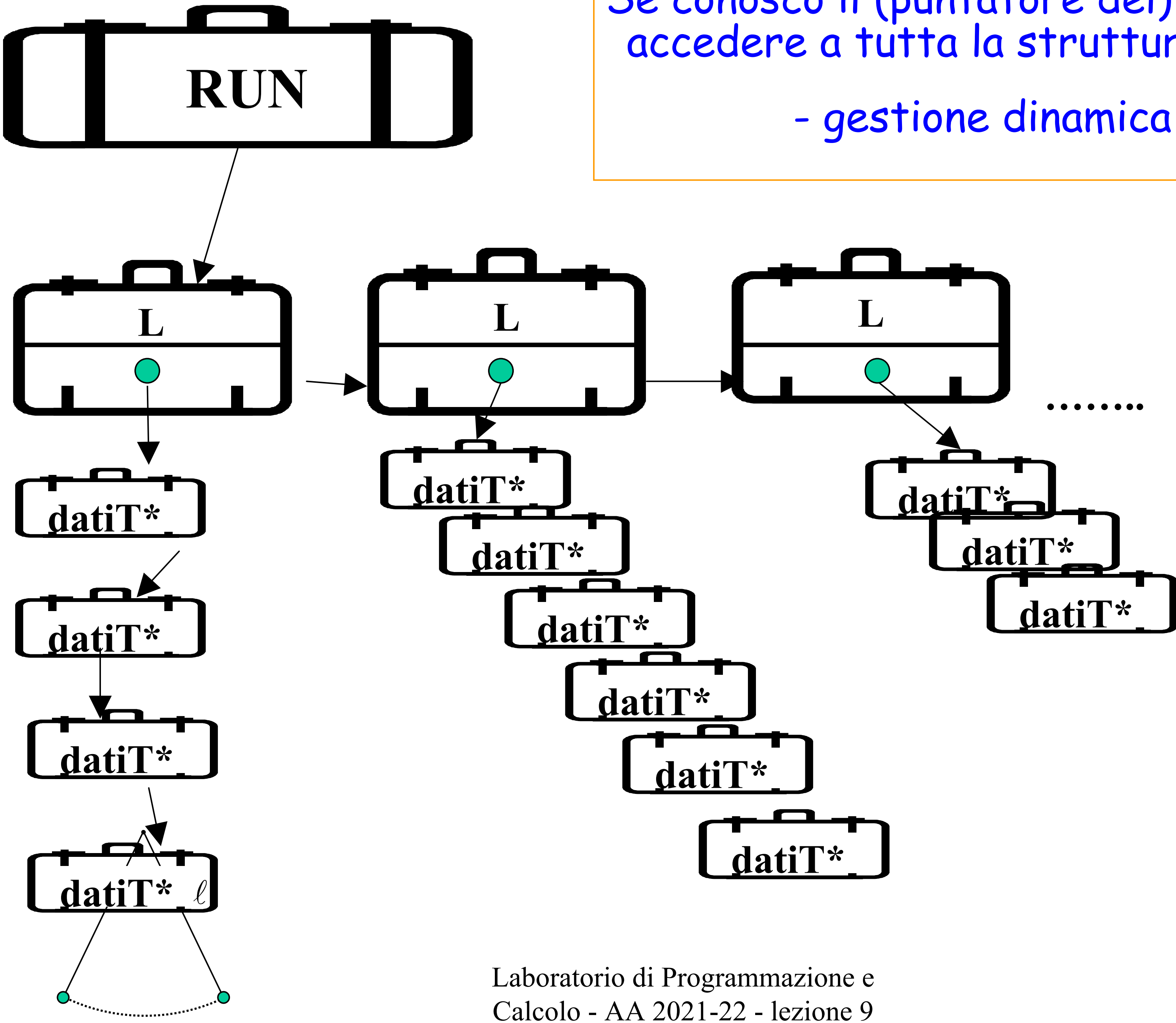
- quindi posso usare

  - un **vector** per l'insieme delle misure di T corrispondenti ad un L fissato
  - una **map** per associare ciascuna lunghezza alla serie di misure di T

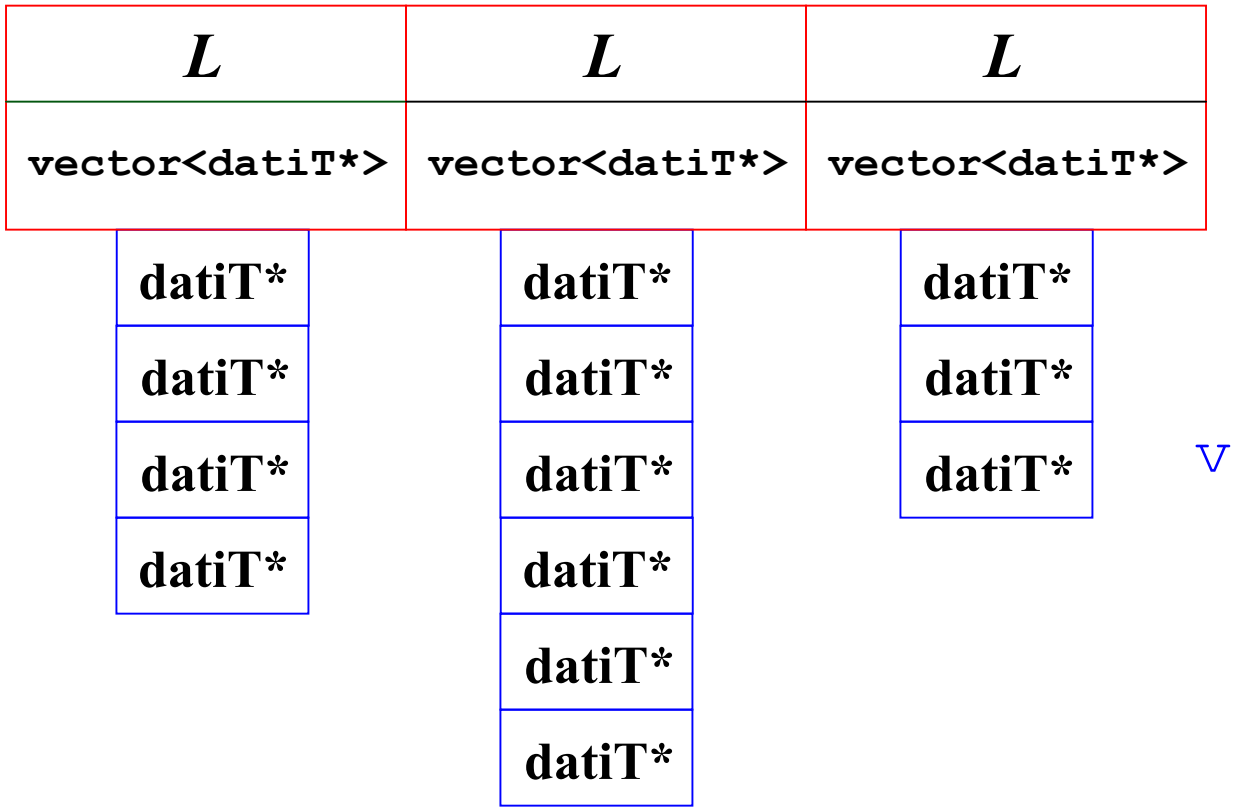


Se conosco il (puntatore del) Run, posso accedere a tutta la struttura dei dati

- gestione dinamica -



`map<double, vector<datiT*> >`



Se ho un problema semplice, posso usare questa struttura nel `main()` ...

```
#include "datiT.h"

#include <iostream>

#include <cstdlib>
#include <map>
#include <vector>

using namespace std ;

int main() {

  int nlmax = 10   ;
  int nTmax[nlmax] ;

  map< double, vector<datiT*> > setDati ;
  vector<datiT*> vlt ;

  typedef map< double, vector<datiT*> >::const_iterator mapiter ;
  typedef vector<datiT*>::const_iterator viter   ;

  typedef map< double, vector<datiT*> >::iterator nc_mapiter    ;
  typedef vector<datiT*>::iterator nc_viter      ;


                                        // continua
```

```
for (int i =0; i<nlmax/2; i++ ) {

   nTmax[i]          = i+1 ;
   nTmax[nlmax-i-1] = i+1 ;

}
cout << endl << endl ;

for (int il=0; il<nlmax; il++) {

   vlt.clear() ;
   cout<< vlt.size()<< endl ;

   for (int iT=0; iT<nTmax[il]; iT++) {

      vlt.push_back( new datiT(il+1,iT+1,iT+2) ) ;

   }

   setDati[il+1] = vlt ;

   cout << " mapsize=" << setDati.size() << " il+1=" << il+1 << endl;
   cout << endl ;

 }

 cout << endl ;
```

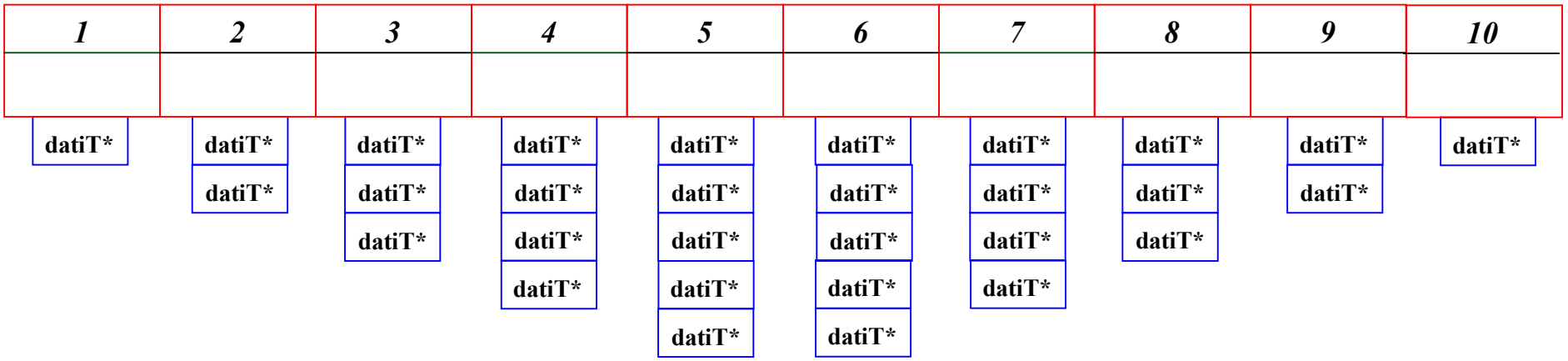
```
// output


for(mapiter pm=(setDati.begin()); pm!=setDati.end(); pm++) {

  cout << pm->first << " " << (pm->second).size() << endl ;

  for(viter pv=(pm->second).begin(); pv!=(pm->second).end(); pv++) {

   cout << "   " << (*pv)->Tmis() << "  " << (*pv)->Nmis()
                 << "  " << (*pv)->T() << endl ;

  }
  cout << endl ;

}
```

```
  // delete
  // ( oggetti datiT - vector<datiT*> - map<double, vector<dati*> )

  for(nc_mapiter pm=(setDati.begin()); pm!=setDati.end(); pm++) {

    for(nc_viter pv=(pm->second).begin();
                                    pv!=(pm->second).end(); pv++) {

      delete *pv ;

    }

    (pm->second).clear() ;

  }

  setDati.clear() ;


 cout << endl ;

 return 1;

}
```

```
nbseve(~/lez_10)>prvmap

0
 mapsize = 1   il+1 = 1

0
 mapsize = 2   il+1 = 2

0
 mapsize = 3   il+1 = 3

0
 mapsize = 4   il+1 = 4

0
 mapsize = 5   il+1 = 5

0
 mapsize = 6   il+1 = 6

0
 mapsize = 7   il+1 = 7

0
 mapsize = 8   il+1 = 8

0
 mapsize = 9   il+1 = 9

0
 mapsize = 10   il+1 = 10
```

```
1 1
    1   2   0.5

2 2
    2   2   1
    2   3   0.666667

3 3
    3   2   1.5
    3   3   1
    3   4   0.75

4 4
    4   2   2
    4   3   1.33333
    4   4   1
    4   5   0.8

5 5
    5   2   2.5
    5   3   1.66667
    5   4   1.25
    5   5   1
    5   6   0.833333
```

```
6 5
    6   2   3
    6   3   2
    6   4   1.5
    6   5   1.2
    6   6   1

7 4
    7   2   3.5
    7   3   2.33333
    7   4   1.75
    7   5   1.4

8 3
    8   2   4
    8   3   2.66667
    8   4   2

9 2
    9   2   4.5
    9   3   3

10 1
    10   2   5


nbseve(~/lez_10)>
```


1
2
3
4
5
6
7
8
9
10
datiT*

… oppure posso implementare una classe apposita …

SetdatiL.h

```
#ifndef SETDATIL_H
#define SETDATIL_H

#include "datiT.h"

#include <iostream>

#include <cstdlib>
#include <map>
#include <vector>

using namespace std ;

class SetdatiL {

   protected:

    double LMin   ;
    double LMax   ;

    map< double, vector<datiT*> > setDati ;


                                          // continua
```

```
public:
  // costruttori
  SetdatiL()  {  } ;
  SetdatiL(double minL, double maxL) ;

  // distruttore
  ~SetdatiL()  ;

  // metodi di tipo Get
  double Lmin()  const  { return LMin  ; } ;
  double Lmax()  const  { return LMax  ; } ;
  int Ndati()    const  { return setDati.size()  ; } ;
  const map< double, vector<datiT*> > * pSetDati() const
                                          { return &setDati ; }

  // altri metodi
  void  addT(double Lnow, datiT* & pTnow ) ;

  // eventualmente 3 metodi per scorrere il container (...)
    ......

} ;

#endif
```

L'implementazione della classe `SetdatiL`

SetdatiL.cc

```
#include "SetdatiL.h"

   // costruttore
   SetdatiL::SetdatiL(double minL, double maxL) {
         LMin = minL ;
         LMax = maxL ;
   }

   // distruttore
   SetdatiL::~SetdatiL() {

   // qui dovrei inserire i distruttori
   // lo stesso codice dell'esempio con solo main

   }

                                        //continua
```

```
   // altri metodi
   void  SetdatiL::addT(double Lnow, datiT* & pTnow ) {

        if(Lnow < LMin) { LMin = Lnow ; }
        if(Lnow > LMax) { LMax = Lnow ; }

        if(!setDati.count(Lnow)) {
          // il valore Lnow non esiste ancora
          // creo il container di tipo vector

          vector<datiT*>  vlc  ;
          vlc.push_back(pTnow) ;

          setDati[Lnow] = vlc ;

      } else {
          // il valore Lnow gia' esiste

          map< double, vector<datiT*> >::iterator mapiter ;

          mapiter = setDati.find(Lnow);
          (mapiter->second).push_back(pTnow) ;

      }

   }
// eventualmente + 3 metodi per scorrere il container (restituisce un
// iteratore a setDati, restituisce begin() e restituisce  end()  )
```

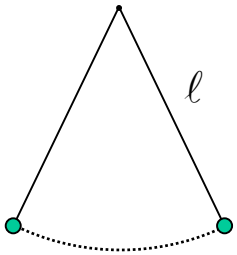
Di chi sono le responsabilita'?

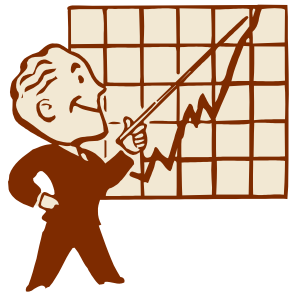
Input (da file)

Dati

Analisi Dati

Output (su file)





Input da file

Il file di input deve essere ben strutturato. Ad esempio:

```
nbseve(~)>sim_pend
0.20 1.10 10
0.20 5
0.8975 0.50 1
0.8975 0.50 1
0.8975 0.50 1
0.8975 0.50 1
0.8975 0.50 1
0.30 6
1.0992 0.50 1
1.0992 0.50 1
1.0992 0.50 1
1.0992 0.50 1
1.0992 0.50 1
0.40 4
1.2692 0.50 1
1.2692 0.50 1
1.2692 0.50 1
1.2692 0.50 1
0.50 8
1.4190 0.50 1
1.4190 0.50 1
1.4190 0.50 1
1.4190 0.50 1
1.4190 0.50 1
1.4190 0.50 1
1.4190 0.50 1
0.60 7
1.5545 0.50 1
1.5545 0.50 1
1.5545 0.50 1
1.5545 0.50 1
1.5545 0.50 1
1.5545 0.50 1
1.5545 0.50 1
0.70 9
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
1.6790 0.50 1
0.80 6
1.7950 0.50 1
1.7950 0.50 1
1.7950 0.50 1
1.7950 0.50 1
1.7950 0.50 1
1.7950 0.50 1
0.90 10
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.9038 0.50 1
1.00 7
2.0068 0.50 1
2.0068 0.50 1
2.0068 0.50 1
2.0068 0.50 1
2.0068 0.50 1
2.0068 0.50 1
2.0068 0.50 1
1.10 4
2.1048 0.50 1
2.1048 0.50 1
2.1048 0.50 1
2.1048 0.50 1

nbseve(~/lez_10)>
```

Lettura del file e uso delle classi `SetdatiL` e `datiT`

```
#include "datiT.h"
#include "SetdatiL.h"

#include <iostream>
#include <fstream>
#include <string

using namespace std ;

int main() {

   double LMinf, LMaxf, LNow ;
   double T, dT ;
   int nL, nTL, nT ;
   datiT* datnow ;

   SetdatiL pendolo ;
   string filename = "sim_pend.dat" ;

   ifstream inFile(filename.c_str()) ;


                                                    //continua
```

Lettura del file e uso delle classi `SetdatiL` e `datiT`

```
    if(!inFile.eof()) {
      inFile >> LMinf >>  LMaxf >>  nL ;
    }

    for(int iL=0; iL<nL; iL++) {
      if(!inFile.eof()) {
        inFile >> LNow >> nTL ;
      }
      if (inFile.eof()) { cerr << " errore lettura file" << endl;}

      for(int it=0; it<nTL; it++) {
        inFile >> T >> dT >> nT ;

        datnow= new datiT(T, dT, nT) ;
        pendolo.addT(LNow, datnow) ;

      }

    }

inFile.close();
return 1;

}
```

Il Manager di Analisi (1)

Anche ora, posso scorrere l'oggetto di tipo SetDatiL nel `main()` ....

... e calcolarmi tutte le quantita' che voglio!

- valori medi di T per ciascun valore di L
- l'errore associato a ciascun valore di T

  (varianza e varianza della media)
- il valore del χ^2 assumendo la relazione $T = 2\pi\sqrt{\frac{\ell}{g}}$
- il valore di g che si ottiene dall'insieme dei dati

  e l'errore ad esso associato
- stampare delle tabelle con L Δl T ΔT T^2 Δt^2

  (ad es. per gnuplot) (magari anche con i valori previsti)

Oppure mi costruisco una classe Manager

... in analogia alla classe `Organizer` utilizzata in laboratorio

Chiamiamo questa classe Analysis

Proviamo a scriverne la struttura

Potrete implementare gran parte dei metodi utilizzando cio' che avete appreso nel corso di Esperimentazioni di Fisica I !

Il Manager di Analisi (2)

Analysis.h

```
#ifndef ANALYSIS_H
#define ANALYSIS_H

#include "SetdatiL.h"

#include <iostream>
#include <string>

using namespace std;


class Analysis {

protected:

    SetdatiL* pDati                              ;
    map< double, vector<double> > mapTmedi ;

    double gvalue ;
    double deltag ;

  // ... altri eventualmente utili
                                                 //continua
```

Analysis.h

```
public:

  // costruttori
  Analysis() {} ;
  Analysis( SetdatiL * dati) {  pDati = dati ; } ;

  // distruttore
  ~Analysis()  ;

  // metodi specifici
  void calcolaMedieVar()    ;
  void calcola_g()          ;
  double  g()         ;
  double  errg()      ;

  void stampaTmedi()      ;

  void tabellepergrafici()                    ;
  void tabellepergrafici(string filename)    ;

  double chi2()  ;

} ;

#endif
```

L'implementazione del Manager di Analisi

Classe `Analysis`

```
#include "Analysis.h"

  // distruttore
  Analysis::~Analysis() {
        // eventuali istruzioni per cancellare le mappe
  }

  // metodi specifici
  void Analysis::calcolaMedieVar() { }
  void Analysis::calcola_g()       { }

  double Analysis::g()      { }
  double Analysis::errg()   { }

  void Analysis::stampaTmedi()  { }

  void Analysis::tabellepergrafici()                  { }
  void Analysis::tabellepergrafici(string filename)   { }

  double Analysis::chi2() { }
```

Un esempio: il metodo `calcolaMedieVar()` della Classe `Analysis`

Analysis.cc

```
void Analysis::calcolaMedieVar() {

   mapTmedi.clear() ;


  // scorro il container come nell'esempio a pag. 31

  // riempio la nuova mappa con i valori di L e di <T>
  // e di <T^2>

  // scorro la nuova mappa e mi calcolo la varianza (?)

  // aggiorno la nuova mappa che avra' come elementi
  //   primo elemento    L
  //   secondo elemento  un vector a tre componenti
  //                     T medio   Var T   errore sulla media


}
```

25) Un esempio: la descrizione dell'apparato ARGO-YBJ

Elementi di Linguaggio UML
e di
Strutture Riutilizzabili

software engineering

"Indice" (1)

- La "Grammatica" UML
- Perche' le strutture riutilizzabili ("*patterns*")
- Classificazioni dei *patterns*
- Studio di alcuni *patterns*

"Indice" (2)

0. Richiami sulla programmazione OO
   - Le classi
   - Le relazioni tra le classi
   - Le interazioni tra le classi
   - Gli *use case*

1. Grammatica del linguaggio UML
   - Le classi
   - Le relazioni tra le classi
   - Le interazioni tra le classi

2. Strutture di codice riutilizzabili (*reusable patterns*)
   - La necessita' del disegno: un esempio
   - Che cos'e' un *patterns*?
   - Ri-utilizzo del codice e *reusable patterns*
   - Classificazione dei *patterns*

3. Esempi di *patterns*
   - *Factory*
   - *Singleton*
   - *Composite*
   - *Strategy*
   - *Observer*
   - *Visitor*

Richiami sulla programmazione OO

1. Le classi
   - Quali classi intervengono nel problema
   - Come sono fatte le classi
   - Cosa fanno le classi
2. Le relazioni tra le classi
   - Come sono legate tra loro le classi
   - Classi in piu' e classi in meno
   - Relazioni di Ereditarieta'
3. Le interazioni tra le classi
   - Chi ha la responsabilita' di che cosa
   - Delegare le responsabilita'
4. Gli use case
   - Forse il punto 0 ?
   - Come verra' usato il programma?
   - Chi lo usera'?
   - A che cosa serve?

Grammatica del linguaggio UML (1)

Universal Modeling Language

Esistono delle convenzioni universali per rappresentare:

- Le classi
- Le relazioni tra le classi
- Le interazioni tra le classi
- Gli *use case*

http://www-01.ibm.com/software/rational/uml

Grammatica del linguaggio UML (2)

Esistono dei pacchetti applicativi per

- Costruire la struttura del programma C++ (o altro linguaggio OO, ad esempio Java) a partire dalla rappresentazione UML

- Ottenere la rappresentazione UML a partire dal codice del programma C++ (o altro linguaggio OO, ad esempio Java) (*Reverse Engeneering*)

La pagina UML (http://www.rational.com/uml)

Unified Modeling Language
- UML™ documentation
- Whitepapers
- Practice the UML
- Recommended reading
- UML thought leaders

Software Engineering Initiatives

Rational UML tools
- Rational XDE
- Rational Rose
- Rational Rose RealTime
- Rational Unified Process
- Rational Suite

In The news
- UML newsroom
- Rational Thought Leaders speak

Communities
- IBM developerWorks™
- UML Café

Other resources
- Services
- User groups
- The Rational Edge

UML Resource Center

UML (

UNIFIED MODELING LANGUAGE™

and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction.

For database professionals UML can be used for database design. Using the UML for database design allows the business and application teams who are already using the UML for their designs to share a common language and to communicate with the database team.

View UML Documentation

What is the purpose of modeling?

Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for large building. Good models are essential for communication among project teams and to assure architectural soundness. As the complexity of systems increase, so does the importance of good modeling techniques. There are many additional factors of a project's success, but having a rigorous modeling language standard is one essential factor.

Since the Unified Modeling Language (UML) became the standard notation for software architecture, it has become the topic of many books, discussions, and seminars. If you have questions or comments about the UML or if you'd like to begin your own topic of discussion, log on to the UML Café, an on-line news group where you can post and reply messages.

Highlights
- → Read what Grady Booch says in .NET Magazine
- → Read the latest work from Terry Quatrani: Introduction to the Unified Modeling Language (PDF 920 KB)

The Rational Edge
- → Adopting use cases Part I: Understanding types of use case and artifacts
- → The role of the service-oriented architect

Customer success
- → Rational and IBM Work Together to Support Development and Use of Web ServicesAIS finds the Rational Unified Process a perfect fit for projects of all sizes
More customer success

Special promotions
- → Order a free solutions for Java platform CD!
- → Order a free solutions for .NET CD!
- → Order a free embedded systems solutions CD!

Le Classi

Rational
the software development company

Rational.com | Rational Developer Network

Welcome! | Create Profile | Login

shop | contacts | searc

products | events | about us | support | services

Home > UML Resource Center

Unified Modeling Language
> Getting Started
> UML Resources
> UML[tm]
Documentation Frameworks
> Quick Reference
> Whitepapers
> Practice the UML

Rational Unified Process

Unified Lifecycle (Rational Suite)

Unified Change Management

Rational Development Accelerators

Quality By Design

Rational Suite Extensibility

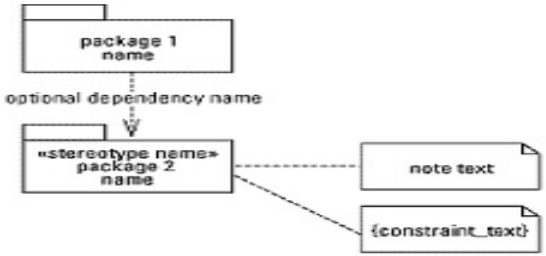
Unified Software Project Management

Click here for printer-friendly version

GENERAL-PURPOSE CONCEPTS

Can be used on various diagram types

Package, dependency, note



USE-CASE DIAGRAM

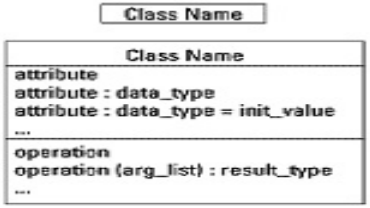
Shows the system's use cases and which actors interact with them

Actor, use case, and association

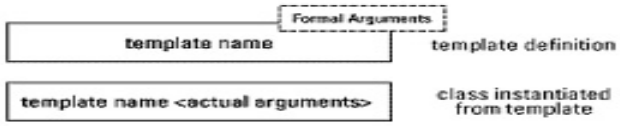
CLASS DIAGRAM

Shows the existence of classes and their relationships in the logical view of a system

Class

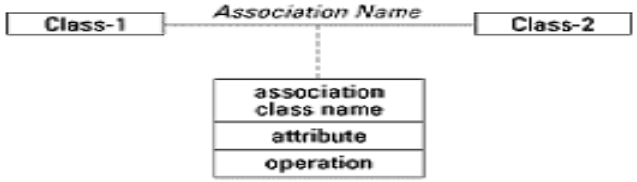


Parameterized class

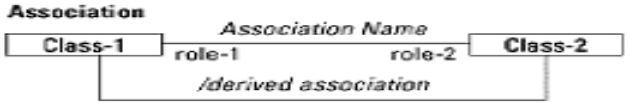


Le relazioni tra le classi

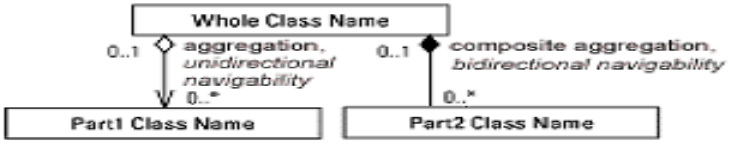
Association classes



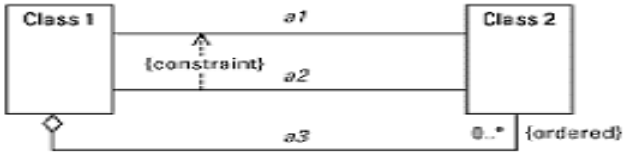
Role names and derived associations



Aggregation, navigability, and multiplicity



Constraints

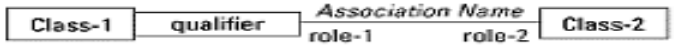


Visibility and properties

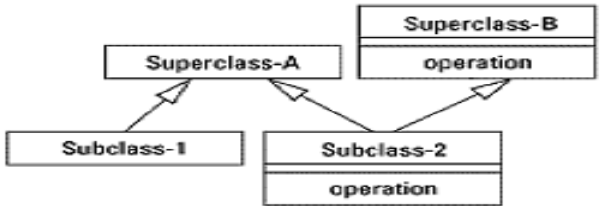
Class
- private attribute # protected attribute /- private derived attribute +$class public attribute
+ public operation # protected operation - private operation +$class public operation

Optional visibility icons

Qualified association



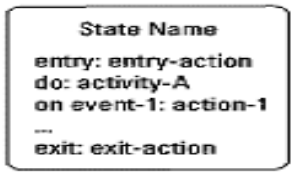
Generalization/specialization



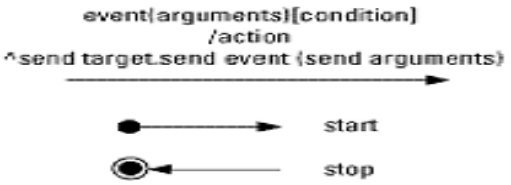
Le interazioni tra le classi

STATE-TRANSITION DIAGRAM Shows the state space of a given context, the events that cause a transition from one state to another, and the actions that result

State icon

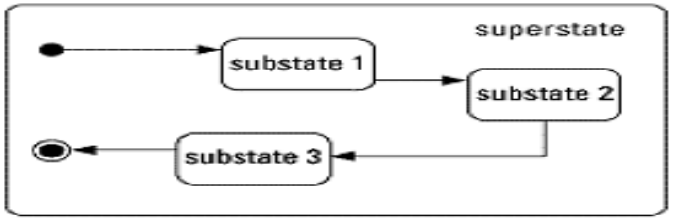


State transitions



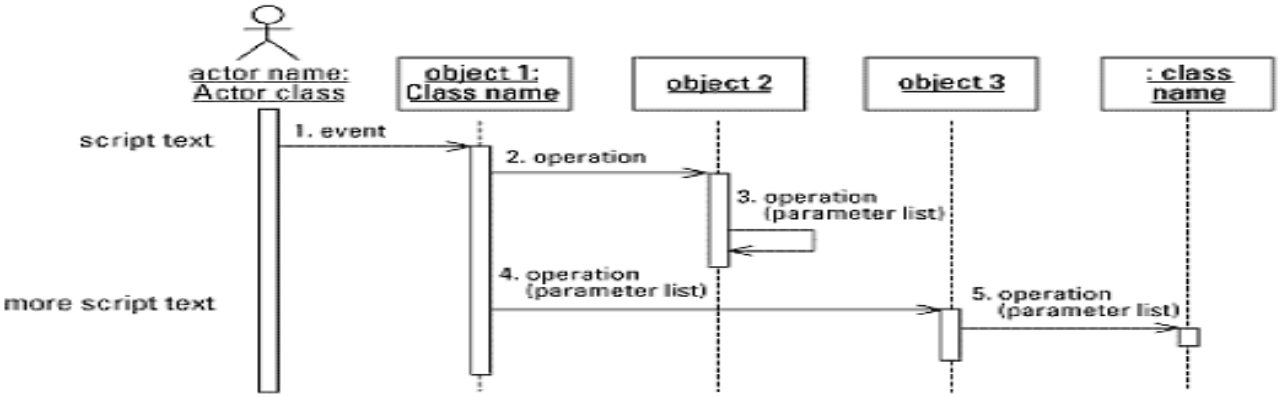
History Ⓗ

Nesting



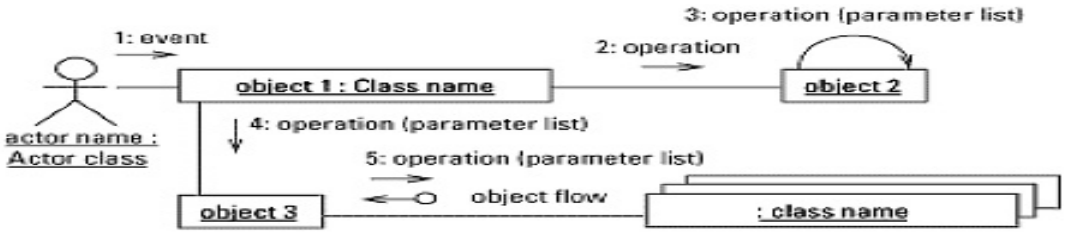
INTERACTION DIAGRAMS Show objects in the system and how they interact

Sequence diagram



Gli *use case*

Collaboration diagram



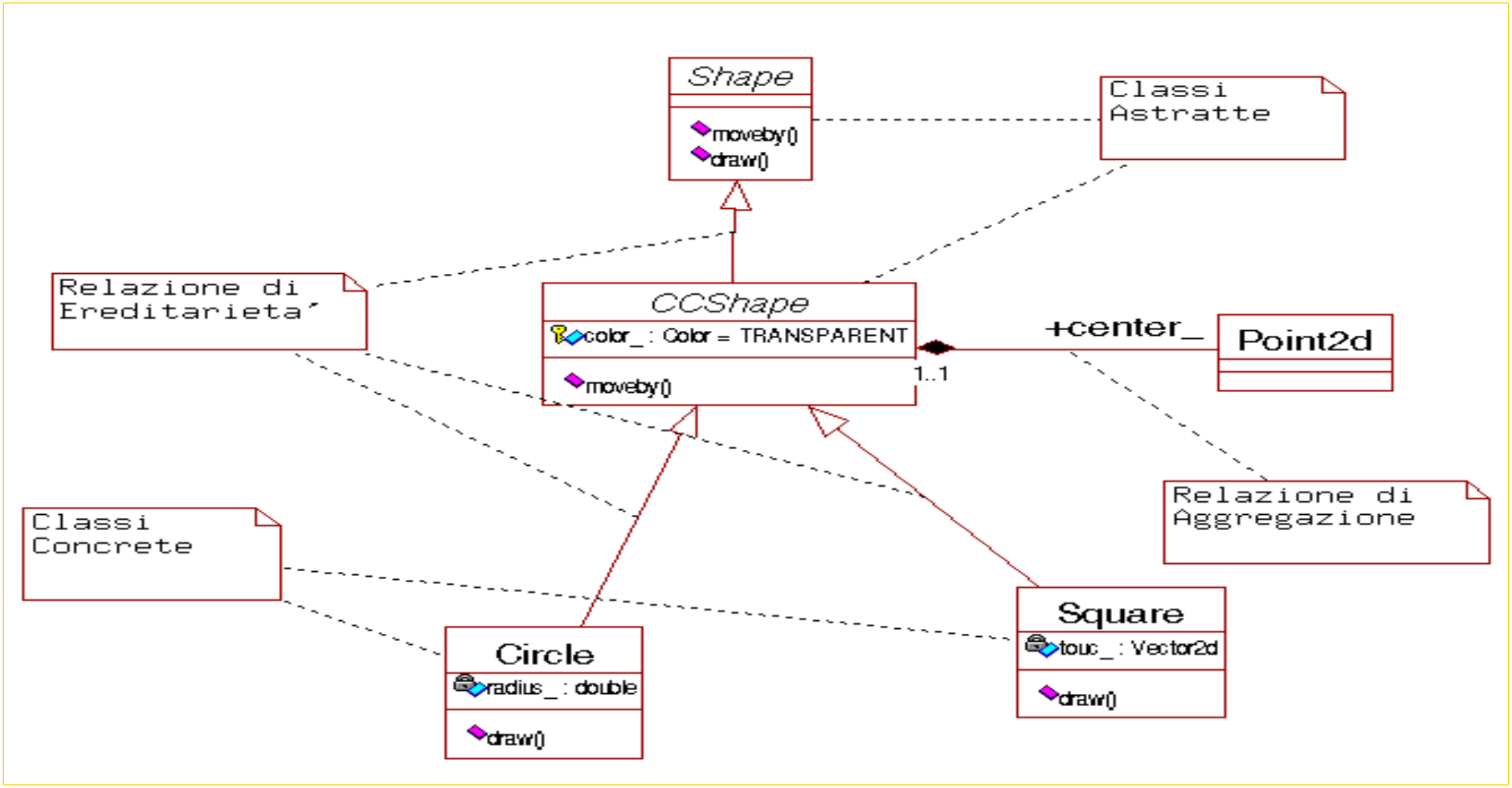
COMPONENT DIAGRAM

Shows the dependencies between software components

DEPLOYMENT DIAGRAM

Shows the configuration of runtime processing elements

Un esempio: rappresentazione UML delle Shape



Strutture di codice riutilizzabili

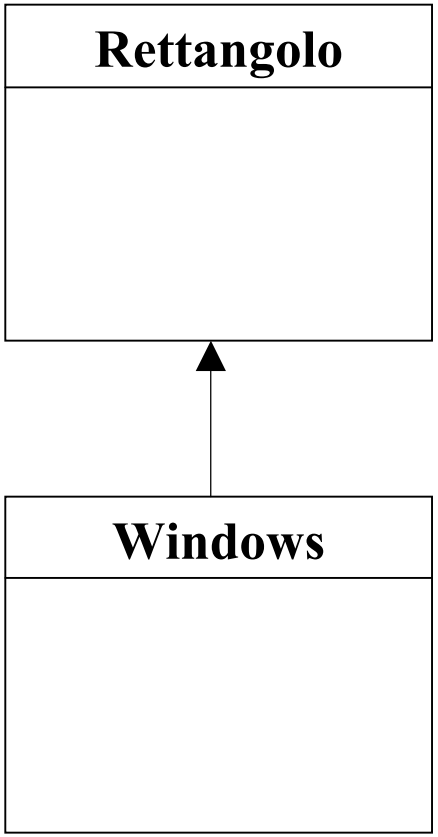
E. Gamma et al.

Design Patterns

Elements of Reusable Object-Oriented Software

1. La necessita' del disegno: un esempio
2. Che cos'e' un *patterns*?
3. Ri-utilizzo del codice e *reusable patterns*
4. Classificazione dei *patterns*

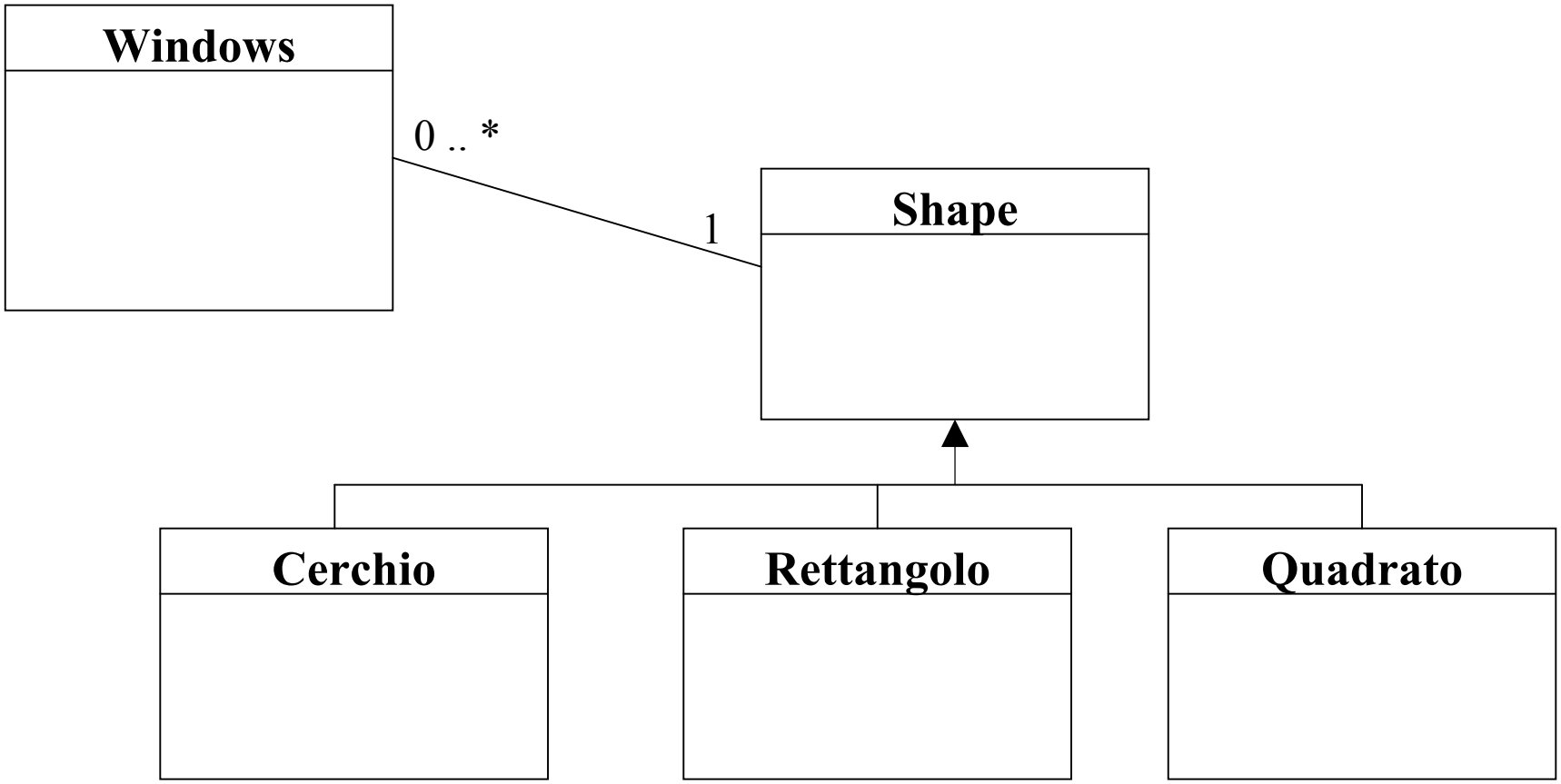
Un esempio: una *Windows (1)*



La windows **e'** un rettangolo

E se la Windows ha un'altra forma?

Un esempio: una *Windows (2)*



La windows **ha** un rettangolo, o meglio una Shape *

La scelta migliore dipende dall'*use case* e dall'uso di questa parte nell'intero programma

Patterns, Ri-utilizzo del codice e Re-usable patterns

Che cos'e' un *patterns*?

Ri-utilizzo del codice e *reusable patterns*

Le tecniche di Ereditarieta' e il meccanismo dei Template permettono di riutilizzare codice dal punto di vista del funzionamento

Qui si tratta di valutare l'uso di una strategia in un contesto diverso (indipendente) da quello in cui e' stata sviluppata.

Naturalmente i patterns usano template ed ereditarieta'

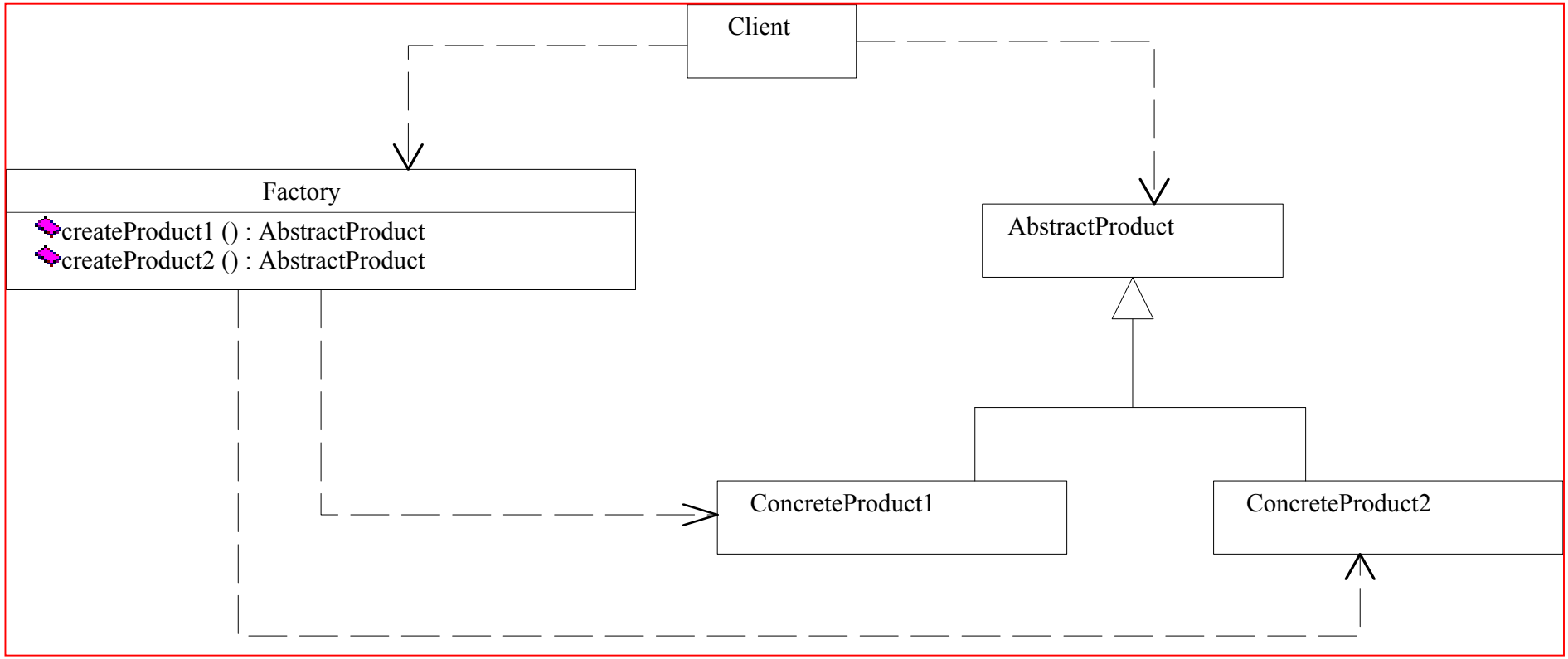
Classificazione dei *patterns*

1. Classificazione dei *patterns*
   - *Pattern* Creazionali
   - *Pattern* Strutturali
   - *Pattern* di Funzionamento
2. Caratteristiche dei *patterns*
   - Il nome
   - Il problema (che vuole affrontare)
   - La soluzione (che propone)
   - Le consequenze (della sua applicazione)
   - Collegamenti con altri *pattern*
   - (confronto con altre soluzioni possibili)

Esempi di *patterns*

1. *Factory*
   Creazionale
2. *Singleton*
   Creazionale
3. *Composite*
   Strutturale
4. *Strategy*
   Funzionamento
5. *Observer*
   Funzionamento
6. *Visitor*
   Funzionamento

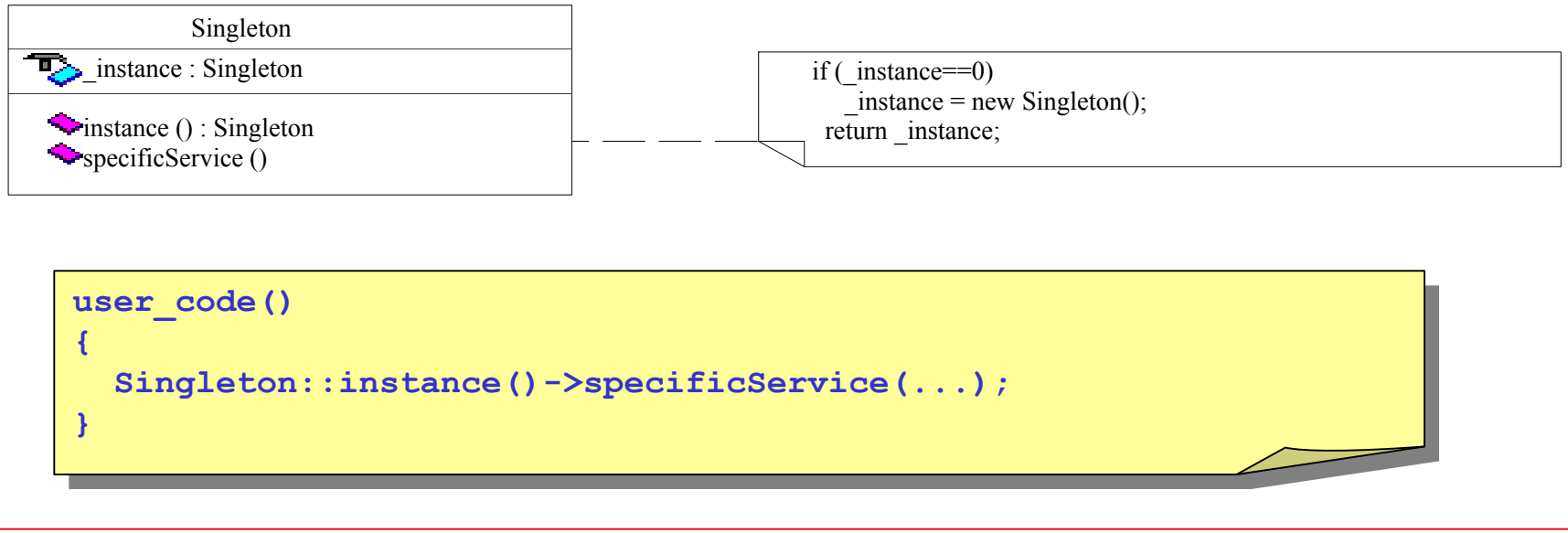
Factory



I **client** possono richiedere la creazione di un prodotto senza dipendervi.

La **`Factory`** dipende dai prodotti concreti, mentre i client dipendono solo **`AbstractProduct`**.

Singleton



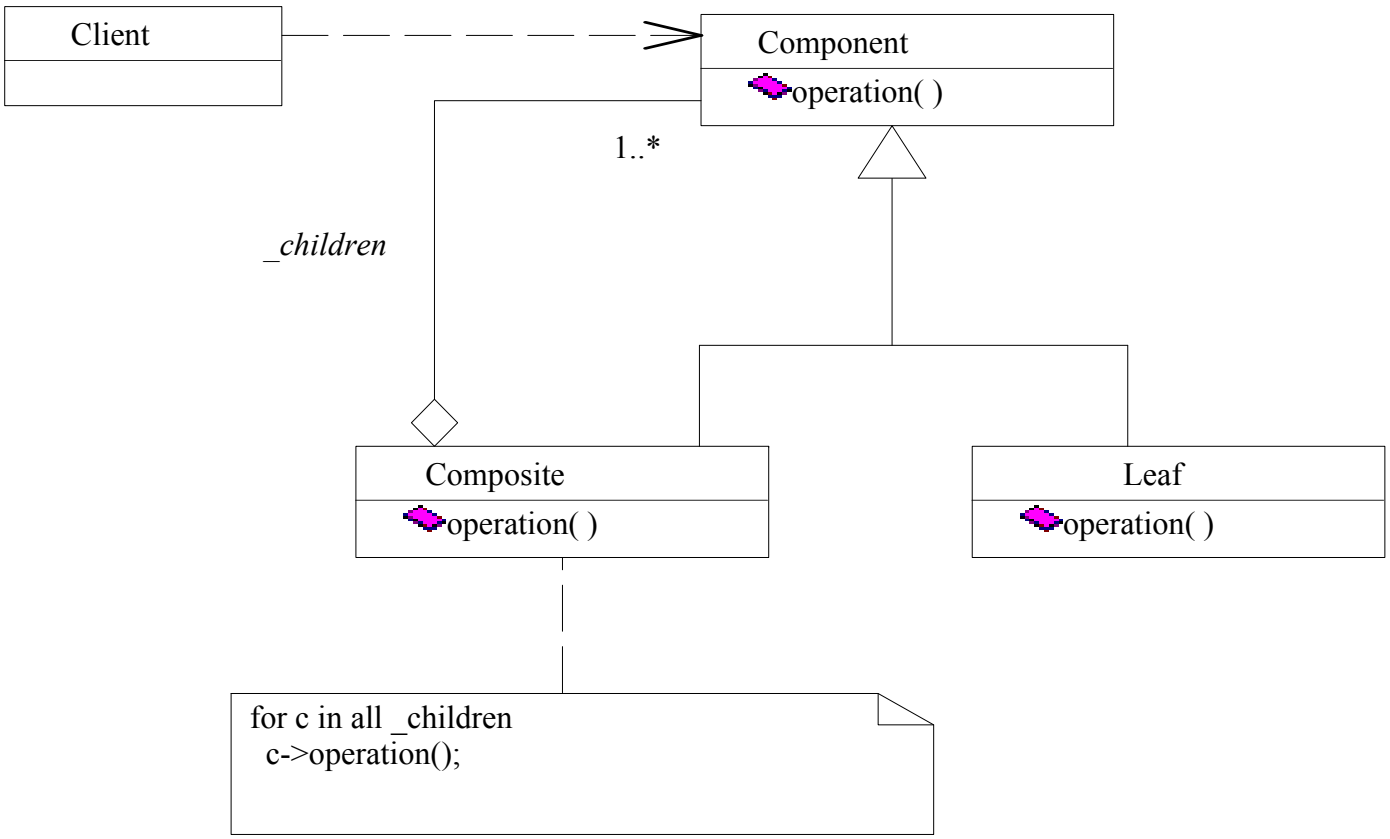
Il Singleton pattern può essere usato ogni volta che una classe deve essere instanziata una sola volta, e viene usata da diversi oggetti.

Per evitare istanziazione accidentale, il constructor deve essere privato.

[Più istanze, ma in numero ben determinato, possono esistere (*multiton*)]

Siccome vengono usate funzioni statiche, l'ereditarietà non può essere applicata.

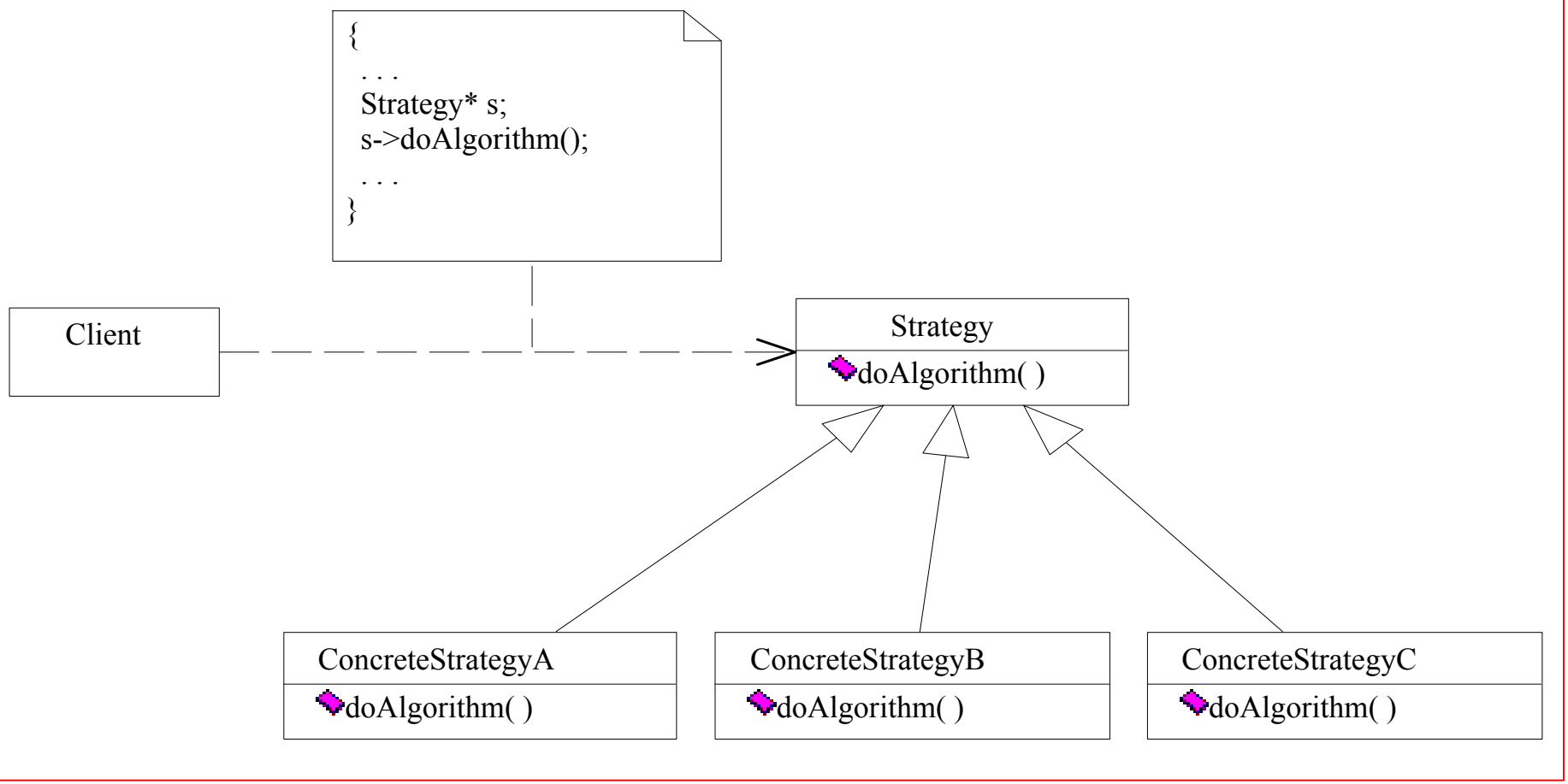
Composite (1)



Il **client** può trattare componenti e compositi usando la stessa interfaccia. La composizione può essere ricursiva.

Esempio: programmi di grafica vettoriale

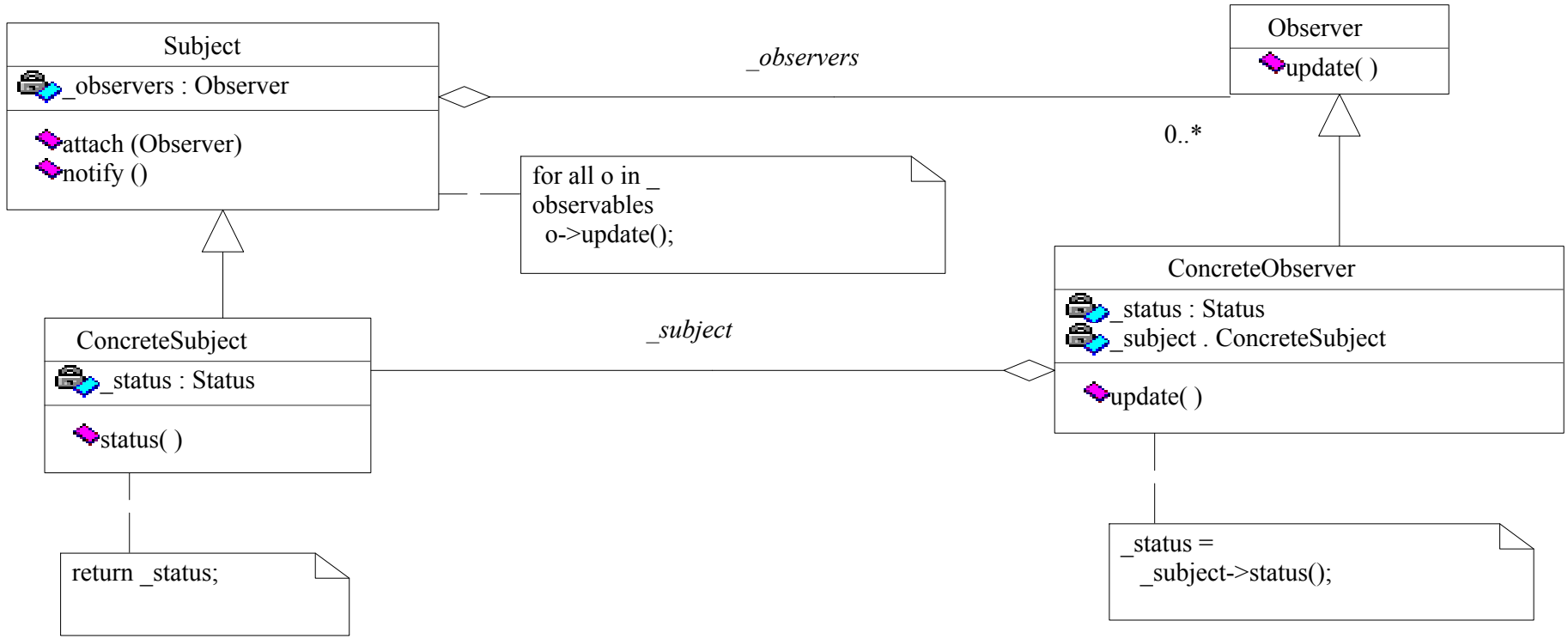
Strategy



Il pattern Strategy permette di scegliere l'algoritmo da eseguire a run-time.

Nuovi algoritmo possono essere introdotti senza modificare i client.

Observer

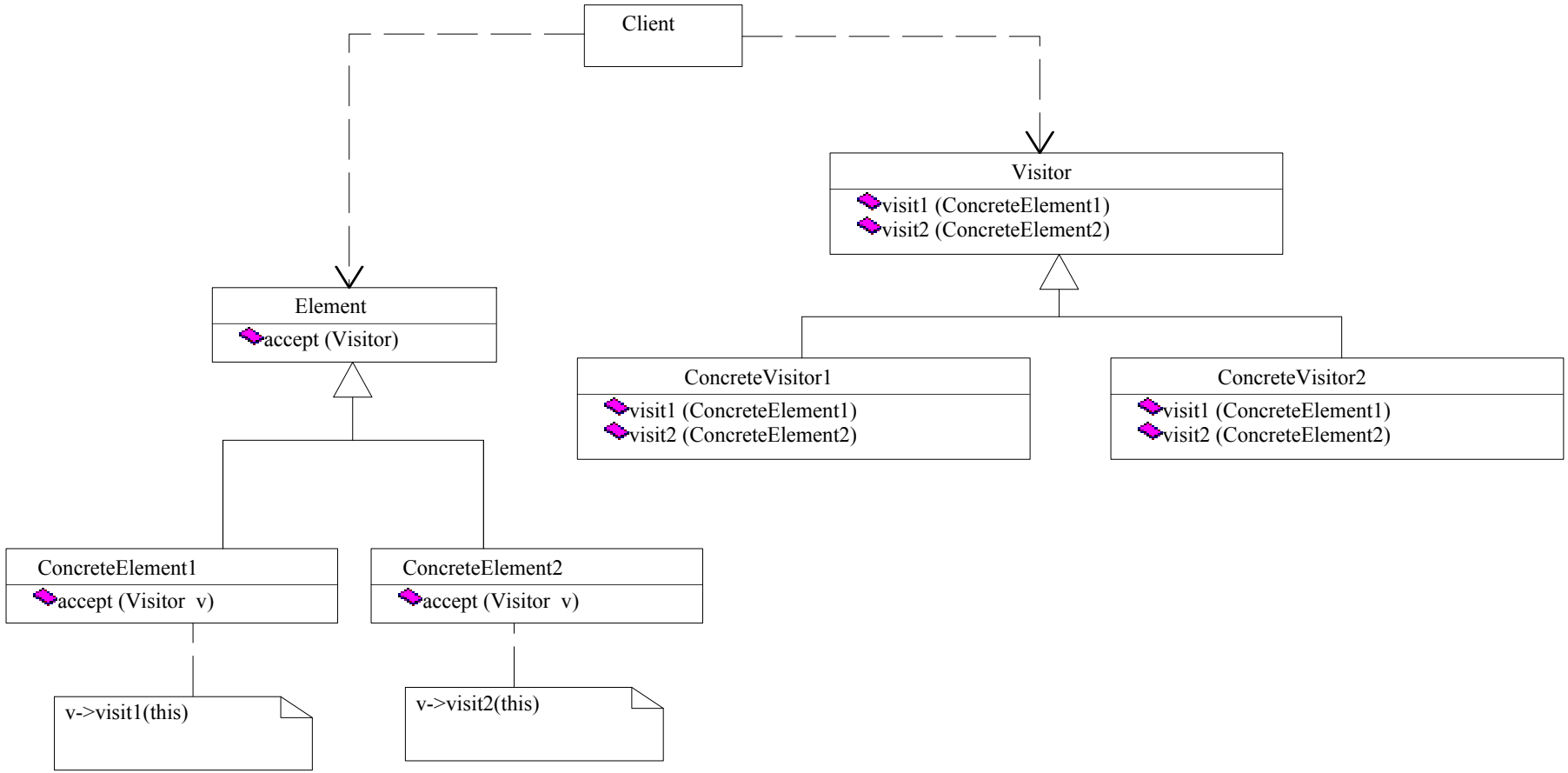


Lo stato dell'**Observer** dipende dallo stato del **Subject**.

Il **Subject** notifica a tutti gli **Observer** registrati che il suo stato è cambiato.

Ogni **Observer** si aggiorna in maniera dipendente dall'implementazione concreta.

Visitor

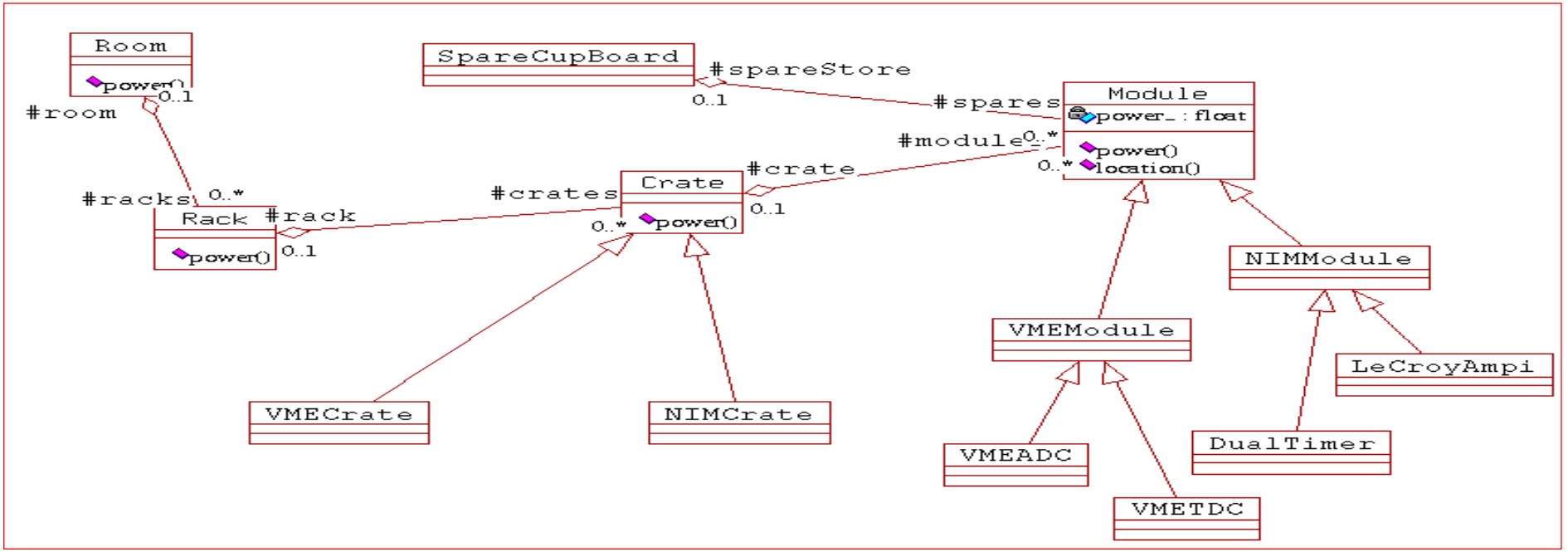


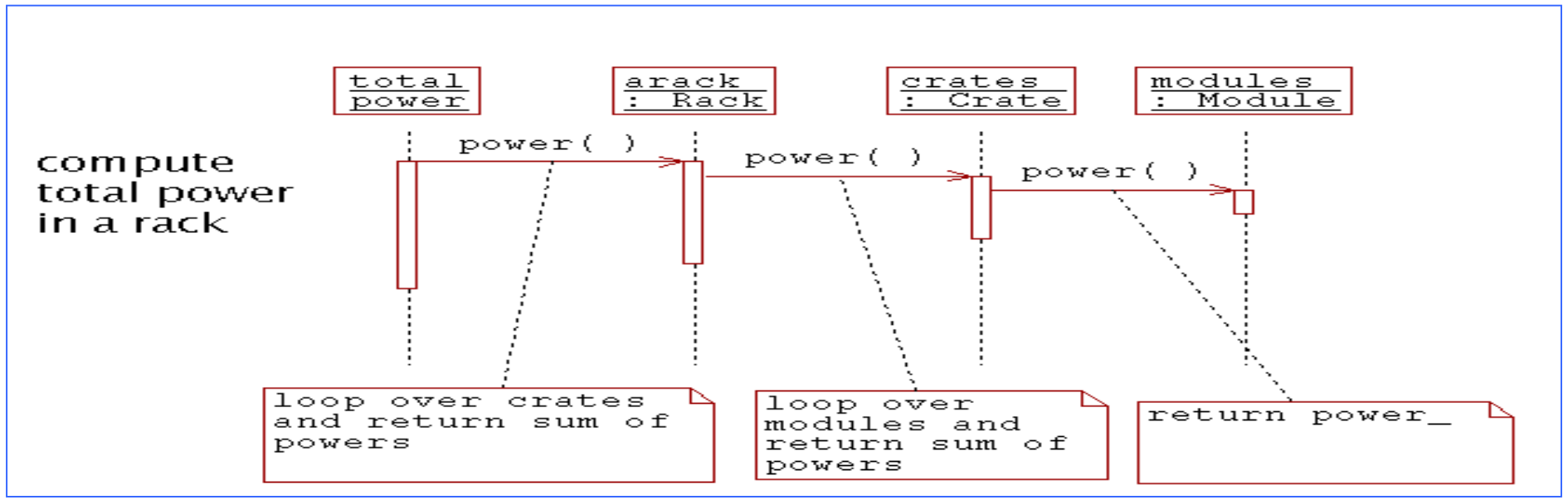
Permette di aggiungere nuove operazioni a **`Element`** senza modificarne l'interfaccia.

Per aggiungere nuovi **`ConcreteElement`**, bisogna modificare tutti i **`Visitors`**.

Composite (2)

Calcolo della Potenza in un Rack

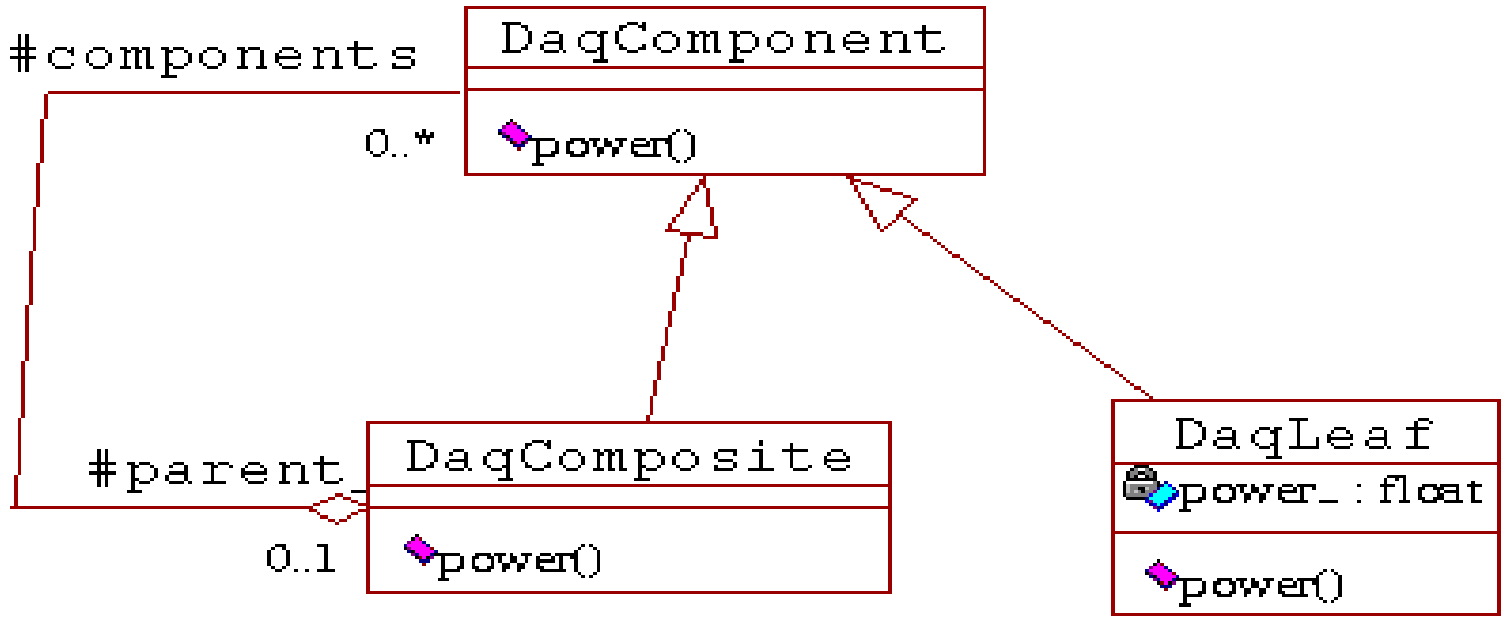




Composite (3)

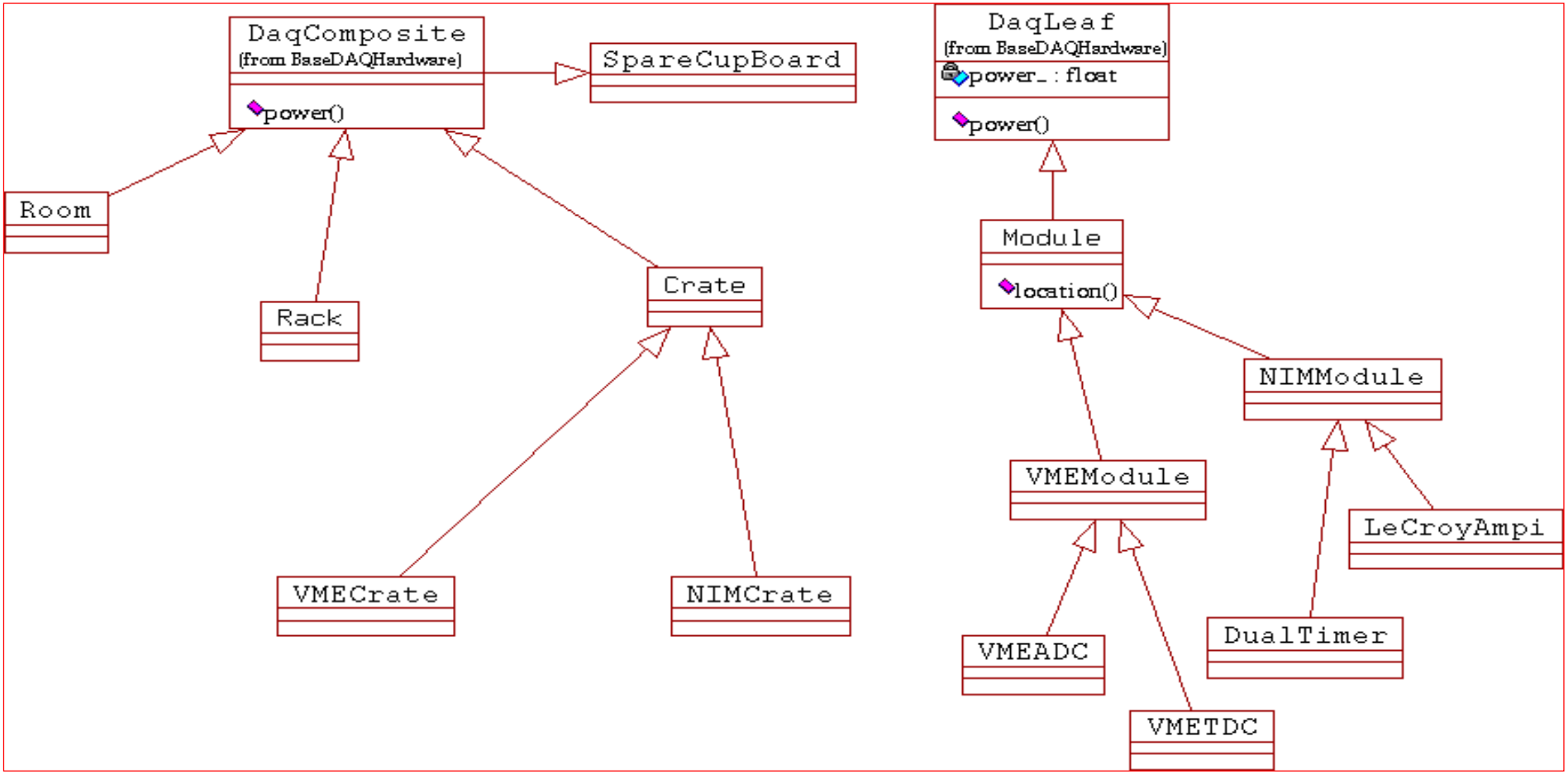
Calcolo della Potenza in un Rack:

Modello Composto



Composite (4)

Le Classi Concrete



Il Codice (1)

DaqComponent.h

```
class DaqComposite; // forward declaration
class DaqComponent {
public:
  DaqComponent() : parent_(0) {}
  virtual float power() const=0
protected:
  DaqComposite * parent_;
};
```

DaqLeaf.h

```
#include "DaqComponent.h"
class DaqLeaf: public DaqComponent {
public:
  explicit DaqLeaf(float ip=0):
   power_(ip) {}

  float power() const { return power_;}
  }
protected:
  float power_;
};
```

Il Codice (2)

DaqComposite.h

```
#include "DaqComponent.h"
class DaqComposite : public DaqComponent {
public:
  typedef vector<DaqComponent *> VC;
  typedef VC::const_iterator CIter;
  DaqComposite(){}
  float power() const {
    tp =0;
    CIter p=components.begin();
    CIter pe=components.end();
    while (p!=pe) {tp+=(*p)->power(); ++p;}
    return tp;
  }
protected:
  VC components;
};
```

Il punto del percorso…

La "Grammatica" UML

Perche' le strutture riutilizzabili ("*patterns*")

Classificazioni dei *patterns*

Studio di alcuni *patterns*

Esempi e applicazioni

Singleton

Composite

Factory

Esempi e applicazioni

1. *Singleton*
   - Un esempio: i Manager di Geant 4
2. Composite: l'esperimento ARGO-YBJ
   - Il problema
   - Soluzione Ingenua
   - Soluzione con patterns
   - Il codice
3. Esempio di *Factory*
   - Sonda o Corpoceleste ?

Singleton *(1-A)*

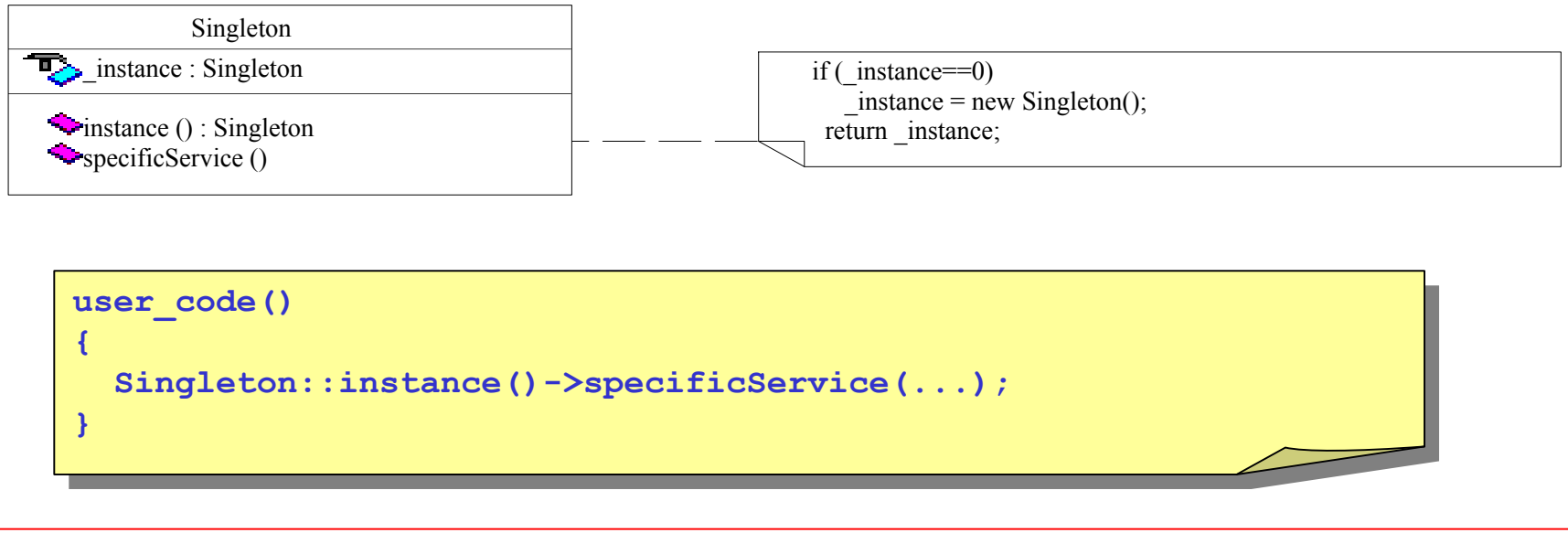
Un esempio: i Manager di Geant 4

Sezioni di Programma "Indipendenti" per effettuare varie operazioni

- Geometria di un Rivelatore
- Scelta dei materiali
- Generazione di particelle
- ecc.

Queste parti di programma, definite come **`Singleton`** sono uniche e sono accessibili da ogni parte del programma

Singleton (1-B)



Il Singleton pattern può essere usato ogni volta che una classe deve essere instanziata una sola volta, e viene usata da diversi oggetti.

Per evitare istanziazione accidentale, il constructor deve essere privato.

Più istanze, ma in numero ben determinato, possono esistere (*multiton*)

Siccome vengono usate funzioni statiche, l'ereditarietà non può essere applicata.

Singleton (2-A)

Il Codice

```
#ifndef AnalysisManager_h
#define AnalysisManager_h
#include <iostream.h>
#include <stdlib.h>
#include <string>
#include <list>
#include <map>
#include "AGManager/ArgoAnalysis.h"
#include "AGManager/ArgoAFile.h"

// Up to now typedef are here
// Maybe they have to be moved into ArgoTypes.h
  typedef list<ArgoAnalysis * >     AA_List  ;
  typedef AA_List::const_iterator   AAL_Iter  ;

  typedef map<string, ArgoAFile *, less<string> >  AAF_Map    ;
  typedef AAF_Map::const_iterator  AAFM_Iter;
```

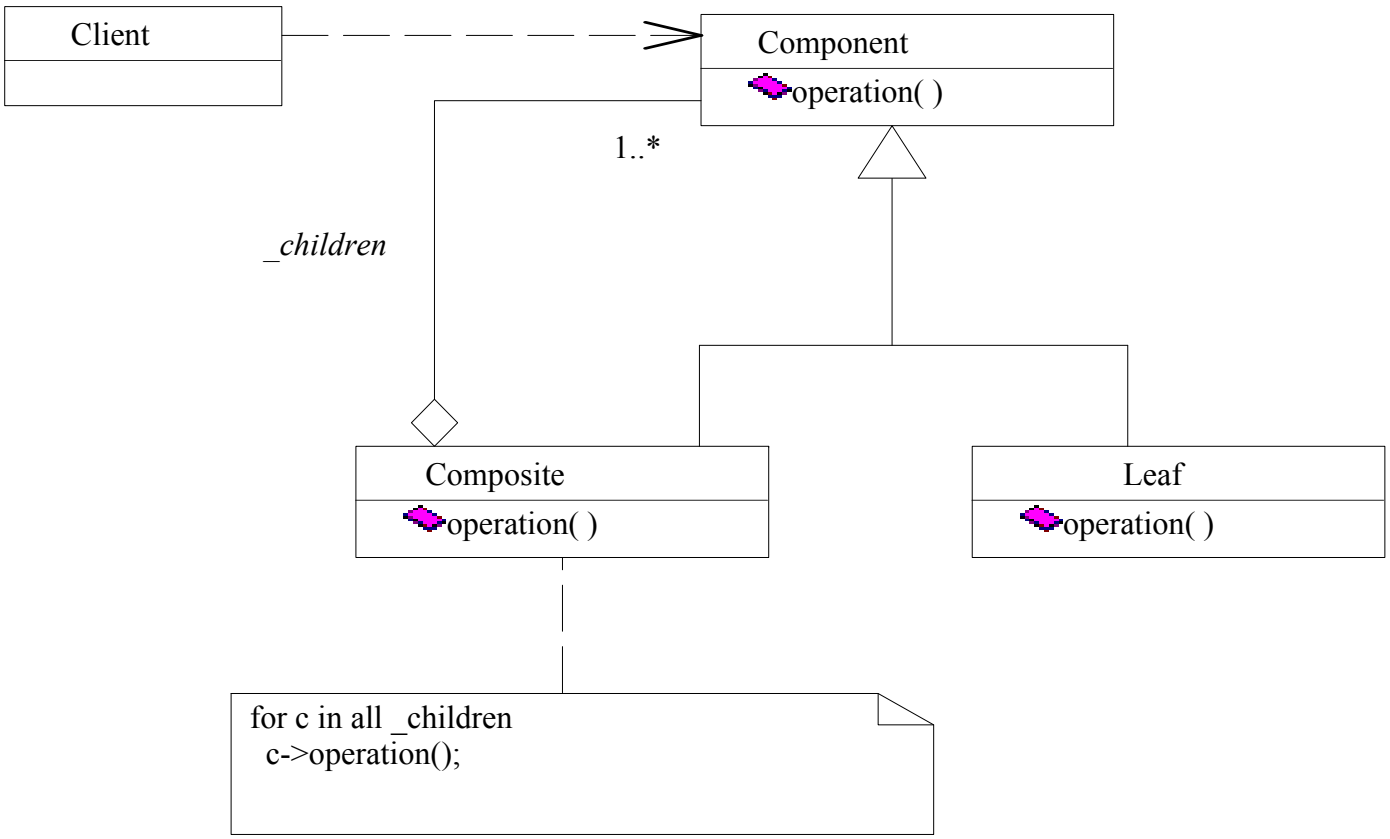
Singleton (2-B)

```
class AnalysisManager {

private:
    static AnalysisManager * _analysismanager;
    AA_List  _analysis_list  ;
    AAF_Map  _file_map       ;
    AnalysisManager()        ;


public:
// destructor
    virtual ~AnalysisManager() ;


// Static Pointer
    static AnalysisManager * GetPointer() ;


…….

    };


#endif
```

Singleton (2-C)

```
#include "AnalysisManager.h"

//static self-pointer
AnalysisManager* AnalysisManager::_analysismanager=0;

AnalysisManager::AnalysisManager()  {} ;

AnalysisManager::~AnalysisManager() {} ;

AnalysisManager * AnalysisManager::GetPointer() {
    if (! _analysismanager) {
       cout<< "Analysis Manager created" << endl;
       _analysismanager = new AnalysisManager;
    }
    return _analysismanager;
};
```

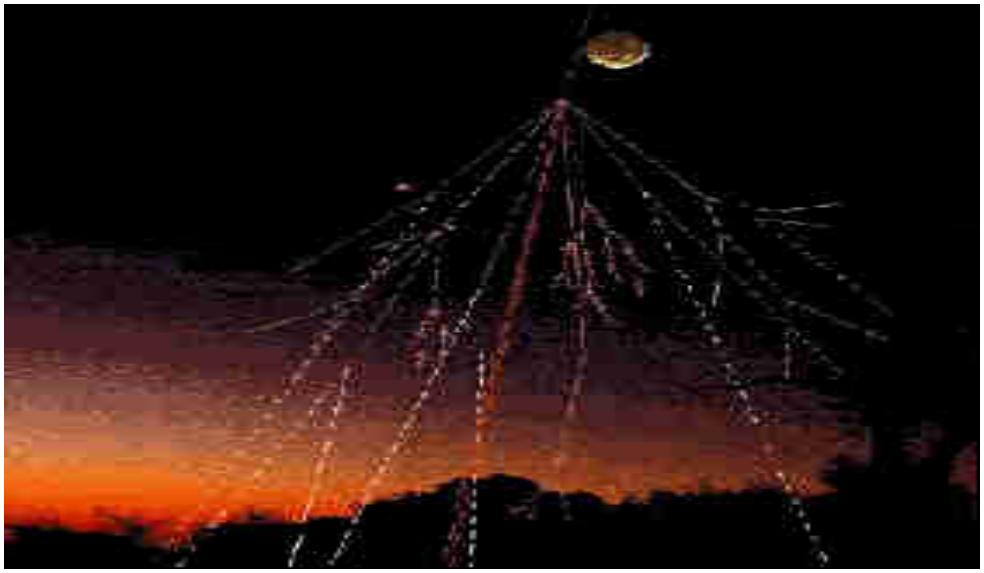
Composite (1)

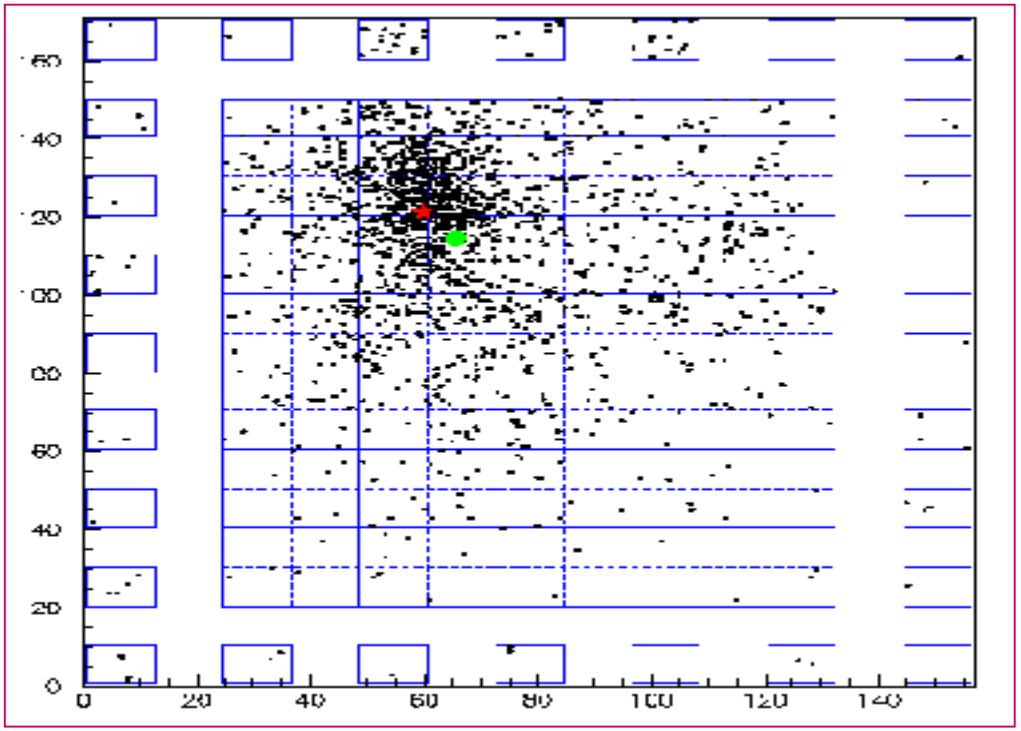


Il **client** può trattare componenti e compositi usando la stessa interfaccia. La composizione può essere ricursiva.

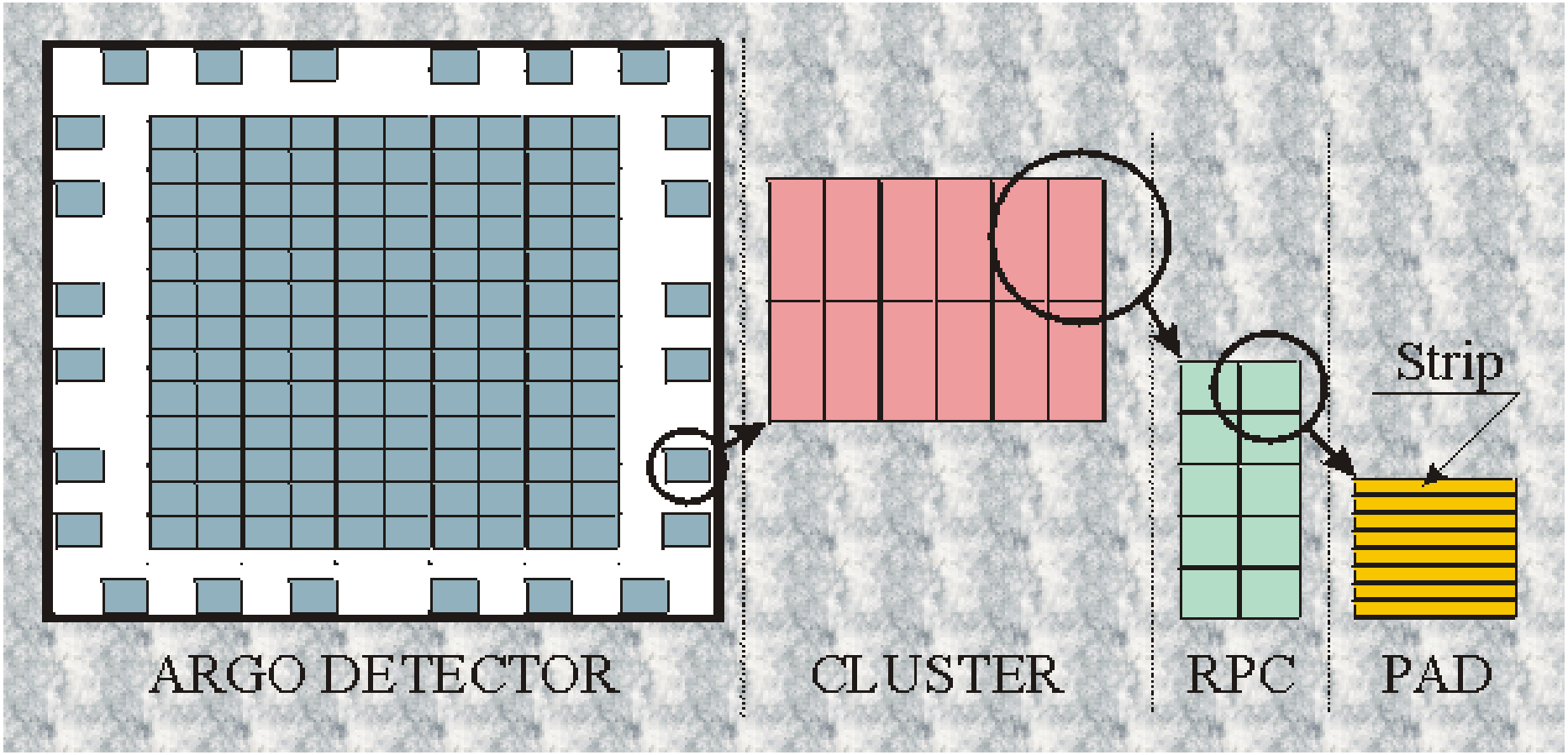
Esempio: programmi di grafica vettoriale

Composite: l'esperimento ARGO-YBJ (2)





Composite: l'esperimento ARGO-YBJ (3)

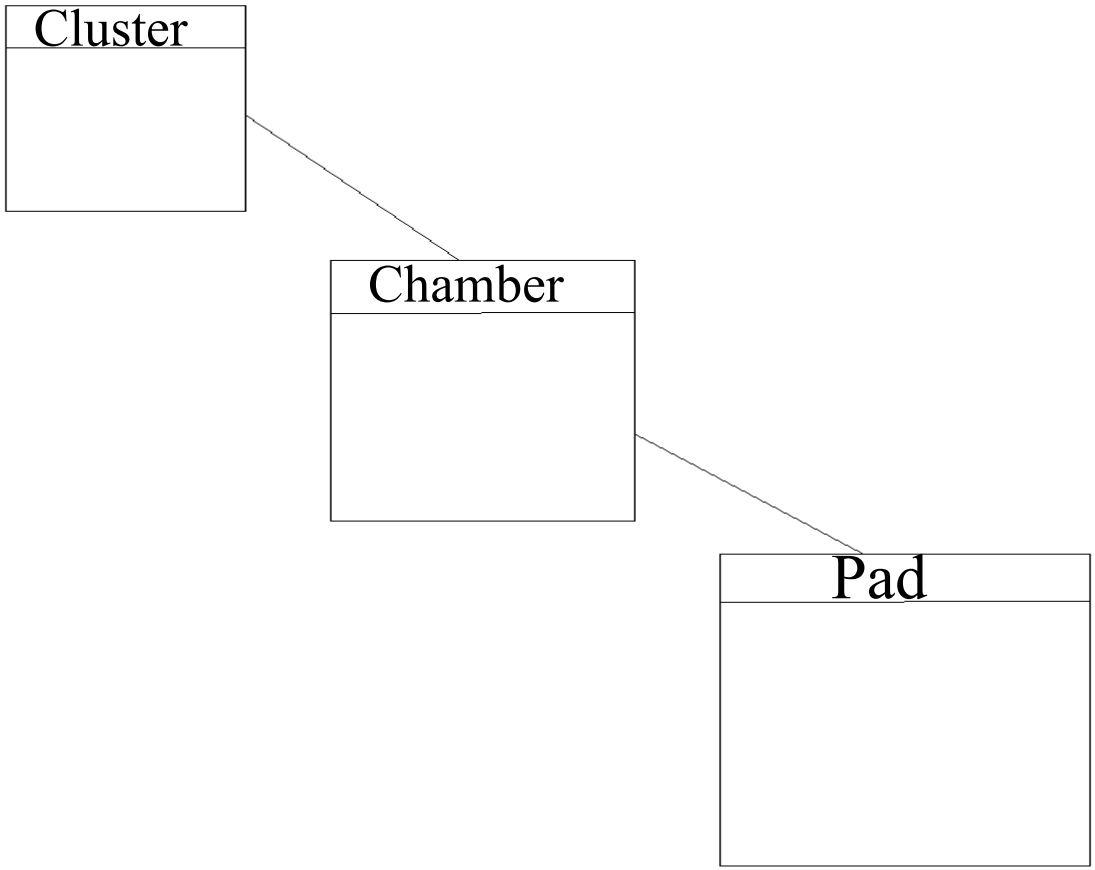


Struttura:

- Rivelatore
  - Cluster
    - Camere
      - Pad

Composite: l'esperimento ARGO-YBJ (4)

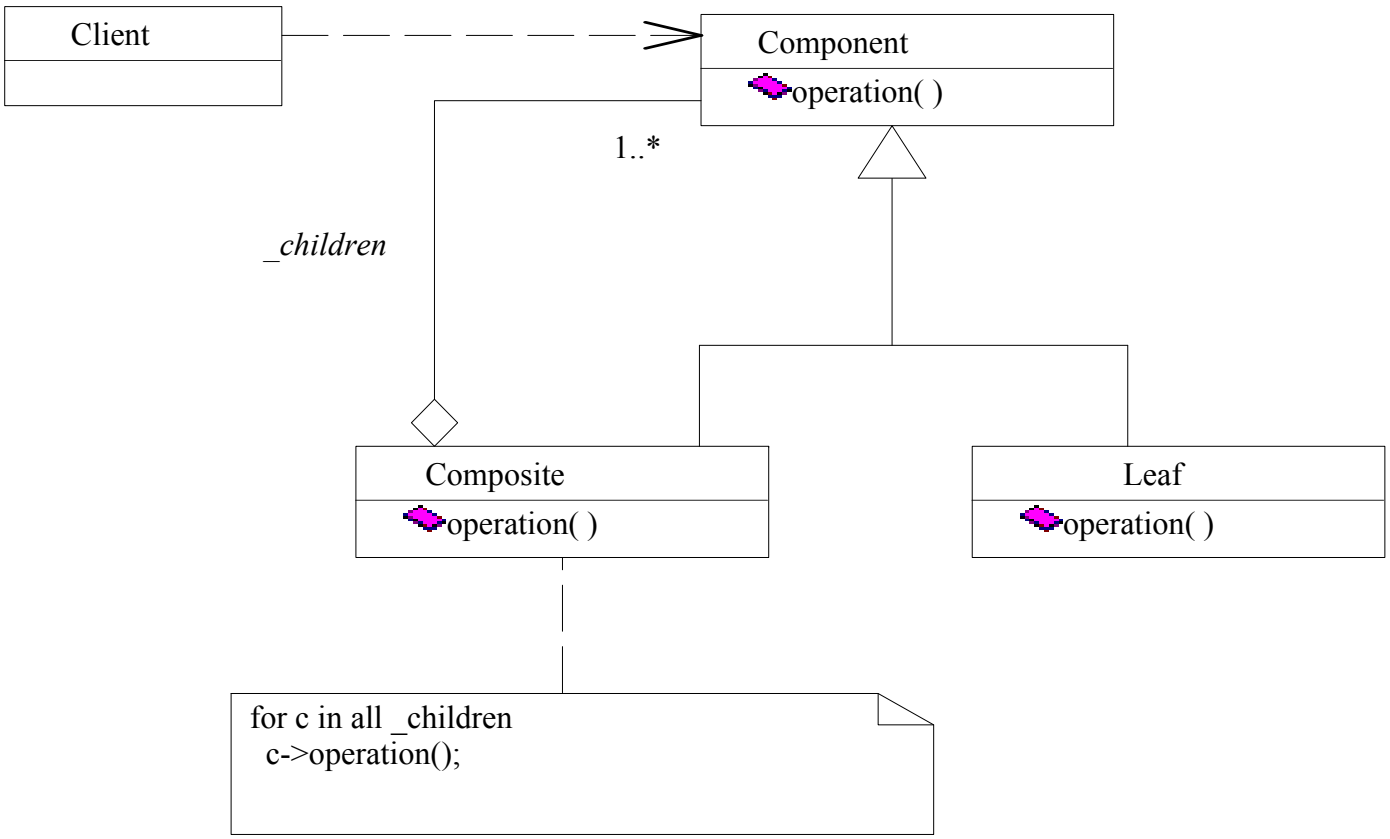
Una soluzione ingenua:



Codice duplicato

Inefficiente

Composite: l'esperimento ARGO-YBJ (5)



Il **client** può trattare componenti e compositi usando la stessa interfaccia. La composizione può essere ricursiva.

Esempio: programmi di grafica vettoriale


ArgoComp
_setted : bool
_active : bool
_ring : bool
ArgoComp()
ArgoComp()
ArgoComp()
setComponent()
operator =()
setParent()
setInactive()
setActive()
setNoInRing()
setInRing()
parent()
name()
idlocal()
idabsolute()
idraw()
xlocal()
ylocal()
zlocal()
xabsolute()
yabsolute()
zabsolute()
nxlocal()
nylocal()
nzlocal()
nxabsolute()
nyabsolute()
nzabsolute()
status()
ring()
calunitvector()
calib_update()
print()
~ArgoComp()
#_parent
0..1
1
T
ArgoCompT
ArgoCompT()
ArgoCompT()
add()
end()
setInactive()
begin()
setActive()
setNoInRing()
setInRing()
element()
~ArgoCompT()
ArgoLeaf
ArgoLeaf()
ArgoLeaf()
~ArgoLeaf()
(ArgoCompT< Module >)
(ArgoCompT< Chamber >)
(ArgoComp T< Pad >)
Pad
Pad()
Pad()
Pad()
parent()
parent()
print()
~Pad()
Cluster
Cluster()
Cluster()
Cluster()
~Cluster()
parent()
parent()
print()
Module
Module()
Module()
Module()
~Module()
parent()
parent()
print()
Chamber
Chamber()
Chamber()
Chamber()
~Chamber()
parent()
parent()
print()

Composite: l'esperimento ARGO-YBJ (6)

Le Classi

ArgoComp.h
ArgoComp.cc — Componente

ArgoCompT.h
ArgoCompT.cc — Composto (Template!)

ArgoLeaf.h — Leaf

Chamber.h
Cluster.h
Pad.h — Classi concrete

testgeom.cpp
testArgoGrid.cpp
testchamber.cpp
testpad.cpp
testcluster.cpp — Programmi di test!

(debug)

Composite: l'esperimento ARGO-YBJ (7-A)

Il Codice

```
#ifndef ArgoComp_H
#define ArgoComp_H

#include <string>

class ArgoComp {

protected:

  ArgoComp * _parent ;
  Aint _id;       // num. position of the component in its composition
  Adouble _x,_y,_z;    // Local coordinates (cm)
  string _name;        // Name of the concrete component
  ……..
public:

// Constructor
  ArgoComp() : _parent(0),_id(0), aid(0), x(0.), y(0.), z(0.),
                                                    …….) { } ;

// Other Constructors - Set and Get Methods
……
// Methods responsable of the component setting
  void setParent(ArgoComp * ip) {_parent = ip; } ;


  virtual ~ArgoComp() {}; // distructor
};

#endif // ArgoComp_H
```

Composite: l'esperimento ARGO-YBJ (7-B)

Il Codice

```
#ifndef ArgoCompT_H
#define ArgoCompT_H
#include <map>
#include "ArgoComp.h"

  typedef map<Aint, ArgoComp*>   VComp   ;
  typedef VComp::const_iterator CIComp   ;

template <class T> class ArgoCompT : public ArgoComp {

protected:
   VComp _components ;
   Aint _ncomponents, _icomponents;

public:
  // Constructors
  ArgoCompT() : _icomponents(0) {} ;
  ArgoCompT(const ArgoComp & AC) : ArgoComp(AC),_icomponents(0) {} ;

 ~ArgoCompT() {} ;

 void add(T * c) {
      if (….) {     _ components[c->idlocal()]=c  ;
                     c->setParent(this)           ;
                    _icomponents++               ;
         } else {              cout <<"Geometry error! << endl; }
  }
};

#endif // ArgoCompT_H
```

Composite: l'esperimento ARGO-YBJ (7-C)

Il Codice

```
#ifndef ArgoLeaf_H
#define ArgoLeaf_H

#include <vector>
#include "ArgoComp.h"

class ArgoLeaf : public ArgoComp {

protected:


public:

  ArgoLeaf() {}  ;

  ArgoLeaf (const ArgoComp &AC) :  ArgoComp(AC) {
          //  cout<<"ArgoComp copied\n";
  };

  ~ArgoLeaf() {} ;

};

#endif // ArgoLeaf_H
```

Composite: l'esperimento ARGO-YBJ (7-D)

Il Codice

Pad.h

Chamber.h

Cluster.h

```
#ifndef Pad_H
#define Pad_H

#include "ArgoLeaf.h"

class Pad : public ArgoLeaf {

public:
  Pad() { _name="Pad"; };
  Pad(const Aint & id,const Aint & aid, …….)
  {
     setComponent(id,aid,……);
     name="Pad";
  } ;

  Pad (const ArgoComp & AC) : ArgoLeaf(AC) { name="Pad";  };

// debug...
   virtual void print() const {
          cout << " Pad n. " << _id  << endl;
   } ;

  virtual ~Pad() {};

};

#endif // Pad_H
```

Composite: l'esperimento ARGO-YBJ (7-E)

Il Codice

Pad.h

Chamber.h

Cluster.h

```
#ifndef Chamber_H
#define Chamber_H
#include <stdlib.h>
#include <iostream>
#include "ArgoCompT.h"
#include "Pad.h"
class Chamber : public ArgoCompT<Pad> {
private:
public:
 Chamber() { _name="Chamber" ;
             _ncomponents=10 ;    } ;

 Chamber(const Aint & id,const Aint & aid, …….) {
        setComponent(id,aid,……) ;
       _name="Chamber" ;       _ncomponents=10 ;
 } ;

 Chamber(const ArgoComp & AC) : ArgoCompT<Pad>(AC) {
       _name="Chamber" ;       _ncomponents=10 ;
 };
  virtual ~Chamber() {
};
#endif // Chamber_H
```

Composite: l'esperimento ARGO-YBJ (7-F)

Il Codice

```
#ifndef Cluster_H
#define Cluster_H
#include <stdlib.h>
#include <iostream>
#include "ArgoCompT.h"
#include "Chamber.h"

class Cluster : public virtual ArgoCompT<Chamber> {
private:

public:
  Cluster() { _name="Cluster";  _ncomponents=12; };
  Cluster(const Aint & id,const Aint & aid, …. ) {
      setComponent(id,aid,……..) ;
     _name = "Cluster" ;    _ncomponents=12 ;
  };
  Cluster (const ArgoComp & AC) : ArgoCompT<Chamber>(AC) {
     _name="Cluster";       _ncomponents=12;
  };
 virtual ~Cluster() {};

};

#endif // Cluster_H
```

Sonda o CorpoCeleste? … un esempio di *Factory*

Esempio di *Factory*

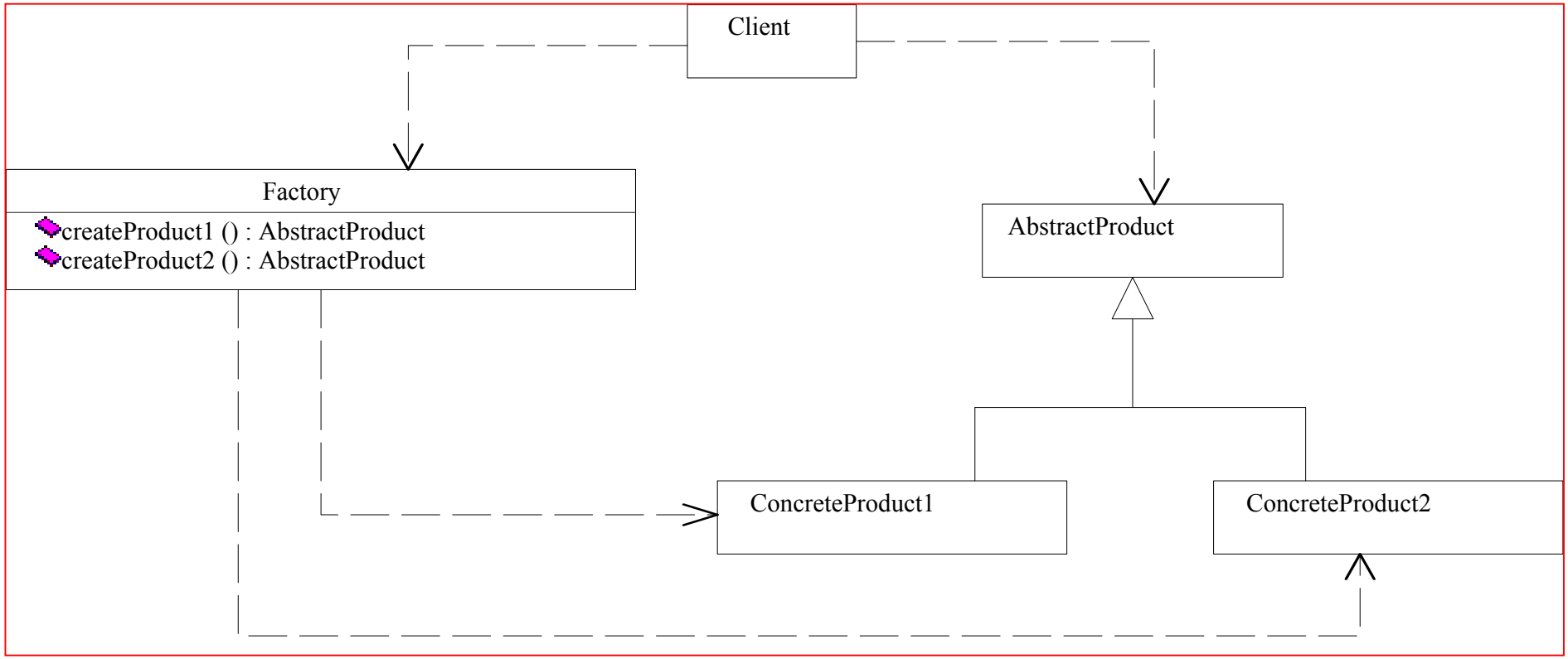
- Sonda o CorpoCeleste ?

Potremmo delegare ad una classe apposita (*Factory*) la responsabilita' di creare per noi oggetti di tipo `CorpoCeleste` o di tipo `Sonda`.

La *Factory* crea per noi l'oggetto e ci restituisce il puntatore ad un oggetto della Classe `CorpoCeleste`

Nel main includiamo solo la classe Corpoceleste, mentre la Factory conosce (= include) tutte le classi concrete (`Sonda`), oltre a `CorpoCeleste`.

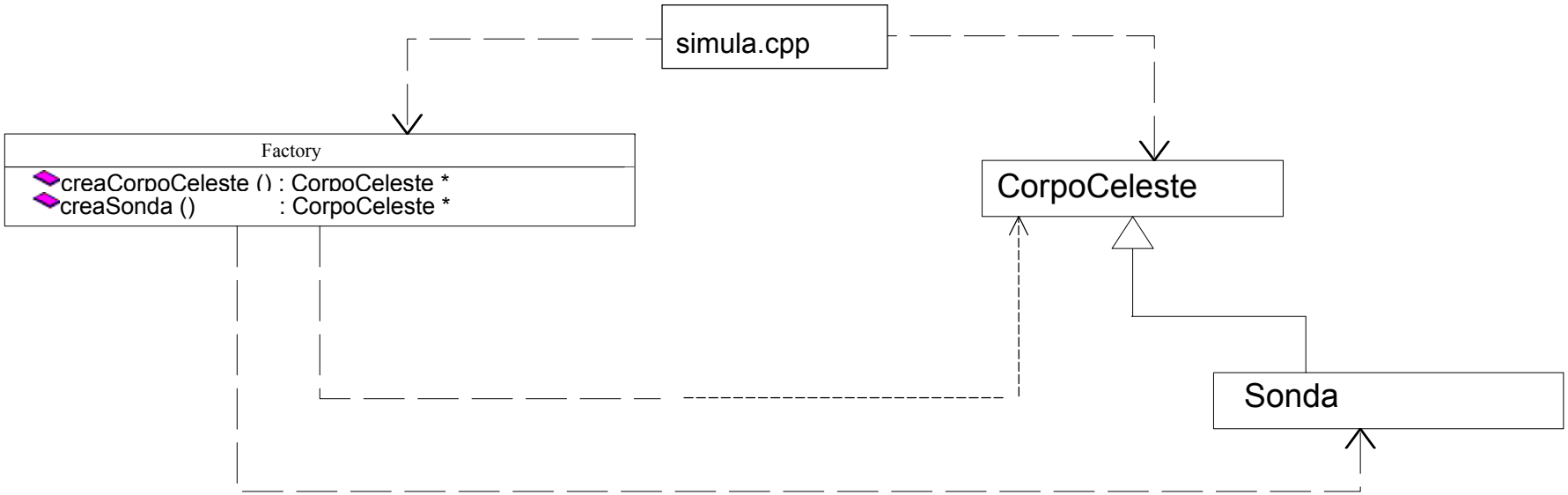
Factory



I **client** possono richiedere la creazione di un prodotto senza dipendervi.

La **`Factory`** dipende dai prodotti concreti, mentre i client dipendono solo **`AbstractProduct`**.

Factory



I **client** possono richiedere la creazione di un prodotto senza dipendervi.

La **`Factory`** dipende dai prodotti concreti, mentre i client dipendono solo **`AbstractProduct`**.

Domande ...

? !

Buon Natale!

Laboratorio di Programmazione e Calcolo - AA 2021-22 - lezione 9