Programmazione 2 e Lab. di programmazione 2

Corso di Laurea in Informatica - Anno Accademico 2022-23

Docenti
Prof. Angelo Ciaramella
[angelo.ciaramella@uniparthenope.it]
Prof. Luigi Catuogno
[luigi.catuogno@uniparthenope.it]
Tutor
Dott. Antonio Vanzanella
[antonio.vanzanella@studenti.uniparthenope.it]

1

Descrizione del Corso

Libro di testo

[FdP]

H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

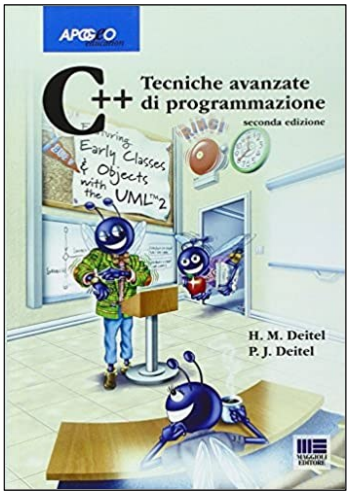
2

Descrizione del Corso

Libro di testo H. M. Deitel, P. J. Deitel

[TAP] C++ Tecniche avanzate di programmazione

II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



3

Orari e modalità di ricevimento studenti

Docenti:

Prof. Angelo Ciaramella	Martedì dalle 14:00 alle 16:00 - telematico (codice Teams r3p3w0z)
Prof. Luigi Catuogno	Giovedì dalle 11:00 alle 13:00 - telematico (codice Teams)

Tutor:

Dott. Antonio Vanzanella	Martedì dalle 11:00 alle 13:00 - telematico (codice Teams 92dbag0)

4

Il Linguaggio C++

(per programmatori C)

Parte prima

5

Le **`class`** in C++

6

Ereditarietà

7

Esercizio: *personaggi di un RPG*

I giocatori di un RPG sono di tipo diverso però condividono un insieme ristretto di attributi e capacità che definiscono un «personaggio base»

La classe **`RPG_PBase`** fornisce gli attributi:

```
int forza; int velocita; int salute;
```

E i metodi:

```
RPG_PBase(int f, int v, int s);
int getForza(); int getVelocita(); int getSalute();
void setForza(int f); void setVelocita(int v);
void setSalute(int s);
void show();
```

8

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.hpp**

```
 1  #ifndef _PERSONAGGIO_HPP_
 2  #define _PERSONAGGIO_HPP_
 3
 4  class RPG_PBase {
 5  public:
 6          int forza;
 7          int velocita;
 8          int salute;
 9          RPGPersonaggio(int f, int v, int s);
10          int getForza();
11          int getVelocita();
12          int getSalute();
13          void setForza(int f);
14          void setVelocita(int v);
15          void setSalute(int s);
16          void show();
17  };
18
19  #endif
```

9

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.cpp**

```
 1  #include<iostream>
 2  using std::cout;
 3
 4  #include"RPG_PBase.hpp"
 5
 6  RPG_PBase::RPGPersonaggio(int f, int v, int s) {
 7                  forza=f; velocita=v; salute=s;
 8  };
 9
10  int RPG_PBase::getForza(){
11                  return forza;
12  }
13  int RPG_PBase::getVelocita(){
14                  return velocita;
15  }
16  int RPG_PBase::getSalute(){
17                  return salute;
18  }
```

10

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.cpp**

```
19 void RPG_PBase::setForza(int f){
20         forza=f;
21 }
22 void RPG_PBase::setVelocita(int v){
23         velocita=v;
24 }
25 void RPG_PBase::setSalute(int s){
26         salute=s;
27 }
28 void RPG_PBase::show() {
29         cout << "[F:"<<forza<<", V:"<<velocita<<", S:"<< salute<<"]";
30 }
```

11

Esercizio: *personaggi di un RPG*

Dal «personaggio base» derivano tutti i personaggi del gioco, ciascuno caratterizzato da attributi e capacità proprie. Ad esempio, i «maghi»…

La classe **RPG_PMago** aggiunge gli attributi:

```
int magia; int incantesimi;
```

i metodi:

```
RPG_PMago(int f, int v, int s, int m, int i);
int getMagia(); int getIncantesimi();
void setMagia(int m); void setIncantesimi(int i);
```

E ridefinisce il metodo **show()**

12

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.hpp**

```
20  class RPG_PMago: public RPG_PBase {
21  public:
22          int magia;
23          int incantesimi;
34          RPG_PMago(int f, int v, int s, int m, int i);
25          int getMagia();
26          void setMagia(int m);
27          int getIncantesimi();
28          void setIncantesimi(int i);
29          void show();
30
31  };
32
```

13

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.cpp**

```
31  int RPG_PMago::getMagia() {
32          return magia;
33  }
34  void RPG_PMago::setMagia(int m) {
35          magia=m;
36  }
37  int RPG_PMago::getIncantesimi() {
38          return incantesimi;
39  }
40  void RPG_PMago::setIncantesimi(int i) {
41          incantesimi=i;
42  }
43
```

14

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.cpp**

```
44  void RPG_PMago::show(){
45          RPG_PBase::show();
46          cout << " (Mag:"<<magia<<", Inc:"<<incantesimi<<")";
47  }
```

Si invoca in maniera esplicita il metodo della classe **RPG_PBase**. Questo metodo ha accesso ed opera sui membri che **RPG_PMago** ha ereditato dalla genitrice.

15

Esercizio: *personaggi di un RPG*

Partendo dalle classi **RPG_PBase** e **RPG_PMago** scrivere le seguenti classi:

La classe **RPG_PRanger** aggiunge al personaggio base gli attributi:

```
int ultravista; int superudito;
```

La classe **RPG_PStregone** aggiunge al mago base l'attributo:

```
int magiaNera;
```

Entrambi ridefiniscono il metodo **show()**

16

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.hpp**

```
34  class RPG_PRanger: public RPG_PBase {
35  public:
36          int ultravista;
37          int superudito;
38          RPG_PRanger(int f, int v, int s, int uv, int su);
39          int getUV();
40          void setSU();
41          int getUV();
42          void setSU();
43          void show();
44  };
45
```

17

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.hpp**

```
34  class RPG_PStregone: public RPG_PMago {
35  public:
36          int magiaNera;
37          RPG_PStregone(int f, int v, int s, int m, int i, int mn);
38          int getMN();
39          void setMN();
40          void show();
41  };
42
```

18

Esercizio: *personaggi di un RPG*

File: **RPG_PBase.cpp**

```
44  void RPG_PStregone::RPG_PStregone(int f, int v, int s, int m, int i, int mn):
45          RPG_PMago(f,v,s,m,i), magiaNera(mn) {
46  }
47
48  Void RPG_PStregone::show() {
49          RPG_PMago::show();
50          cout << endl << " (MNera:" << magiaNera << ")";
51  }
```

19

Esercizio: *punto2D #1*

Scrivere la classe **punto2D** con gli attributi (pubblici):

```
double x; double y;
```

Con i metodi:

```
punto2D(); punto2D(double c1, double c2);
double getX(); double getY();
void translate(double d1, double d2);
double distanza_origine();
void show();
```

20

Esercizio: *punto2D #1*

File: **punto2Dv1.hpp**

```
1  #ifndef _PUNTO_2Dv1_HPP_
2  #define _PUNTO_2Dv1_HPP_
3
4  class punto2D {
5  public:
6          double x;
7          double y;
8          punto2D();
9          punto2D(double c1, double c2);
10         double getX() const;
11         double getY() const;
12         void show() const;
13         void translate(double d1, double d2);
14         double distanza_origine();
15 };
16 #endif
```

21

Esercizio: *punto2D #1*

File: **punto2Dv1.cpp**

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  #include "punto2Dv1.hpp"
5
6  punto2D::punto2D() { x=0; y=0; }
7  punto2D::punto2D(double c1, double c2){
8          x=c1;
9          y=c2;
10 }
11
12 double punto2D::getX() const {
13         return x;
14 }
15 double punto2D::getY() const {
16         return y;
17 }
```

22

Esercizio: *punto2D #1*

File: **punto2Dv1.cpp**

```
18 void punto2D::show() const {
19         cout <<"("<<x<<","<<y<<")";
20 }
21
22 void punto2D::translate(double d1, double d2) {
23         x+=d1;
24         y+=d2;
25 }
26
27 double punto2D::distanza_origine(){
28         return sqrt(pow(x,2)+pow(y,2));
29 }
```

23

Esercizio: *punto3D #1*

Scrivere la classe **punto3D** che derivi dalla **punto2D** e che aggiunga la terza coordinata:

```
double z;
```

il nuovo metodo:

```
double getZ();
```

…i costruttori e ridefinisca i metodi:

```
void translate(double d1, double d2, double d3);
double distanza_origine(); void show();
```

24

Esercizio: *punto3D #1*

File: **punto3Dv1.hpp**

```
1  #ifndef _PUNTO_3Dv1_HPP_
2  #define _PUNTO_3Dv1_HPP_
3  #include"punto2Dv1.hpp"
4
5  class punto3D: public punto2D {
6  public:
7          double z;
8          punto3D(double c1,double c2,double c3);
9          double getZ() const;
10         void show() const;
11         double distanza_origine() const;
12         void translate(double d1, double d2, double d3);
13 };
14 #endif
```

25

Esercizio: *punto3D #1*

File: **punto3Dv1.cpp**

```
1  #include <iostream>
2  #include <cmath>
3  #include "punto3Dv1.hpp"
4  using namespace std;
5
6  punto3D::punto3D(double c1,double c2,double c3){
7          x=c1; y=c2; z=c3;
8  }
9  double punto3D::getZ() const {
10         return z;
11 }
12 void punto3D::show() {
13         std::cout << "("<<x<<","<<y<<","<< z <<")";
14 }
```

26

Esercizio: *punto3D #1*

File: **punto3Dv1.cpp**

```
15 double punto3D::distanza_origine(){
16         return sqrt(pow(x,2)+pow(y,2)+pow(z,2));
17 }
18
19 void punto3D::translate(double d1, double d2, double d3) {
20         x+=d1;
21         y+=d2;
22         z+=d3;
23 }
```

27

Esercizio: *punto3D #2*

Apportiamo una semplice modifica a **punto2D**

Ora, **x** e **y** sono private!

```
class punto2D {
      double x;
      double y;
public:
      punto2D();
      punto2D(double c1, double c2);
      double getX() const;
…
```

Riscriviamo la **punto3D** *di conseguenza…*

28

Esercizio: *punto3D #2*

File: **punto3Dv2.hpp**

```
5   class punto3D: public punto2D {
6           double z;
7   public:
8           punto3D(double c1,double c2,double c3);
9           double getZ() const;
10          …
```

Modifichiamo l'accesso anche alla terza coordinata, per coerenza.

29

Esercizio: *punto3D #2*

File: **punto3Dv2.cpp**

```
15  void punto3D::show() {
16          cout << "("<<getX()<<","<<getY()<<","<< z <<")";
17  }
18
19  double punto3D::distanza_o    ine(){
20          double tmp;
21          tmp=pow(punto2D      tanza_origine(),2);
22          return sqrt(t        (z,2));
23  }
```

I metodi ereditati dalla classe genitrice mantengono l'accesso ai membri privati dichiarati in essa…

z è un attributo privato di questa classe, vi si può accedere normalmente

30

Esercizio: *punto3D #2*

File: **punto3Dv2.cpp**

```
15  void punto3D::show() {
16          cout << "("<<getX()<<","<<getY()<<","<< z <<")";
17  }
18
19  double punto3D::distanza_origine(){
20          double tmp;
21          tmp=pow(punto2D::distanza_origine(),2);
22          return sqrt(tmp+pow(z,2));
23  }
```

Qui si invoca in maniera esplicita il metodo della classe genitrice che accede e opera esclusivamente sui membri da essa ereditati

31

Esercizio: *punto3D #2*

File: **punto3Dv2.cpp**

```
…   …
 9  punto3D::punto3D(double c1,double c2,double c3)
10          : punto2D(c1,c2), z(c3)
11  { }
…   …
```

Il costruttore della nuova classe usa l'inizializzatore della classe genitrice per impostare il valore dei membri privati da essa ereditati.

32

Esercizio: *punto3D #2*

File: **main.cpp**

```
30  int main(int argc, char** argv) {
31          punto2D A(2,3);
32          punto3D B(10,10,10);
33          cout << "A=";
34          A.show();
35          cout << endl;
36          cout << "B=";
37          B.show();
38          cout << endl;
39          cout << "Bdist="<<B.distanza_origine()<<endl;
40          return 0;
41  }
```

```
A=(2,3)
B=(10,10,10)
Bdist=17.3205
```

33