2021_12_16

December 16, 2021

1 2021-12-16

1.0.1 Corso ITS

1.1 Magento & e-commerce software

1.2 ### Fondamenti di Programmazione (Andrea Ribuoli)

1.2.1 I macro-contenuti

- ciclo di vita del software
- elementi di base per la programmazione di un software complesso
- la programmazione con pseudo-linguaggi
- la programmazione orientata agli oggetti
- analisi del software sulla base delle esigenze del cliente
- redazione di software secondo gli standard
- il controllo di versione

1.2.2 Il diavolo è nei dettagli

Dubbi

1.3 Studio costruttori C++ in classi astratte

```
#include <iostream>
#include <string>

using namespace std;

class Azione {
  private:
    string nome;
  public:
    void setName(string n) { nome=n; }
    virtual string dettagli() = 0;
    void debug() {
        cout << "dettagli(" << nome << "): " << dettagli() << "\n";
    }
};

class Tweet : public Azione {
```

```
  private:
    string messaggio;
  public:
    void setMessage(string n, string m) {
        messaggio=m;
        setName(n);
    }
    string dettagli() {
        string risultato = messaggio;
        return risultato; }
};

class Email : public Azione {
  private:
    string messaggio;
    string destinatario;
  public:
    void setMessage(string n, string m, string d) {
        messaggio=m;
        destinatario=d;
        setName(n);
    }
    string dettagli() {
        string risultato = messaggio + "(" + destinatario + ")";
        return risultato; }
};

int main() {
   Tweet t;
   t.setMessage("tweet", "Mi dispiace se oggi è stata molto pesante la lezione!");
   Email e;
   e.setMessage("email", "vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.com");
   t.debug();
   e.debug();
}
```

1.3.1 introduciamo un costruttore nella classe padre (Azione)

Utilizzando la specifica `using Azione::Azione;` nelle classi derivate estendiamo la valenza del costruttore nelle classi derivate (Tweet, Email)

```
10d9
<     Azione(string n) { nome=n; }
22d20
<     using Azione::Azione;
37d34
<     using Azione::Azione;
49c46
```

```
<    Tweet t = Tweet(string("a tweet"));
---
>    Tweet t;
51c48
<    Email e = Email(string("an email"));
---
>    Email e;
```

```
#include <iostream>
#include <string>

using namespace std;

class Azione {
  private:
    string nome;
  public:
    Azione(string n) { nome=n; }
    void setName(string n) { nome=n; }
    virtual string dettagli() = 0;
    void debug() {
        cout << "dettagli(" << nome << "): " << dettagli() << "\n";
    }
};

class Tweet : public Azione {
  private:
    string messaggio;
  public:
    using Azione::Azione;
    void setMessage(string n, string m) {
        messaggio=m;
        setName(n);
    }
    string dettagli() {
        string risultato = messaggio;
        return risultato; }
};

class Email : public Azione {
  private:
    string messaggio;
    string destinatario;
  public:
    using Azione::Azione;
    void setMessage(string n, string m, string d) {
        messaggio=m;
        destinatario=d;
```

```
        setName(n);
    }
    string dettagli() {
        string risultato = messaggio + "(" + destinatario + ")";
        return risultato; }
};

int main() {
   Tweet t = Tweet(string("a tweet"));
   t.setMessage("tweet", "Mi dispiace se oggi è stata molto pesante la lezione!");
   Email e = Email(string("an email"));
   e.setMessage("email", "vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.com");
   t.debug();
   e.debug();
}
```

1.3.2 a questo punto possiamo eliminare la funzione *setName* non più utilizzata

Evitiamo anche di passare il nome della istanza nella *setMessage* (essendo già inizializzata tramite il costruttore.

```
10a11
>     void setName(string n) { nome=n; }
22c23
<     void setMessage(string m) {
---
>     void setMessage(string n, string m) {
23a25
>         setName(n);
36c38
<     void setMessage(string m, string d) {
---
>     void setMessage(string n, string m, string d) {
38a41
>         setName(n);
47c50
<    t.setMessage("Mi dispiace se oggi è stata molto pesante la lezione!");
---
>    t.setMessage("tweet", "Mi dispiace se oggi è stata molto pesante la lezione!");
49c52
<    e.setMessage("vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.com");
---
>    e.setMessage("email", "vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.com");
```

```
#include <iostream>
#include <string>

using namespace std;
```

```
class Azione {
  private:
    string nome;
  public:
    Azione(string n) { nome=n; }
    virtual string dettagli() = 0;
    void debug() {
        cout << "dettagli(" << nome << "): " << dettagli() << "\n";
    }
};

class Tweet : public Azione {
  private:
    string messaggio;
  public:
    using Azione::Azione;
    void setMessage(string m) {
        messaggio=m;
    }
    string dettagli() {
        string risultato = messaggio;
        return risultato; }
};

class Email : public Azione {
  private:
    string messaggio;
    string destinatario;
  public:
    using Azione::Azione;
    void setMessage(string m, string d) {
        messaggio=m;
        destinatario=d;
    }
    string dettagli() {
        string risultato = messaggio + "(" + destinatario + ")";
        return risultato; }
};

int main() {
   Tweet t = Tweet(string("a tweet"));
   t.setMessage("Mi dispiace se oggi è stata molto pesante la lezione!");
   Email e = Email(string("an email"));
   e.setMessage("vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.com");
   t.debug();
   e.debug();
}
```

1.3.3 ora possiamo creare dei costruttori specifici per le sottoclassi

```
22,23c22
<     Tweet(string n, string m) : Tweet(n) { messaggio=m; }
<     void setMessage(string n, string m) {
---
>     void setMessage(string m) {
37,38c36
<     Email(string n, string m, string d) : Email(n) { messaggio=m; destinatario=d; }
<     void setMessage(string n, string m, string d) {
---
>     void setMessage(string m, string d) {
48,49c46,49
<    Tweet t = Tweet(string("a tweet"), "Mi dispiace se oggi è stata molto pesante la lezione!"
<    Email e = Email(string("an email"), "vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo
---
>    Tweet t = Tweet(string("a tweet"));
>    t.setMessage("Mi dispiace se oggi è stata molto pesante la lezione!");
>    Email e = Email(string("an email"));
>    e.setMessage("vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.com");
```

```cpp
#include <iostream>
#include <string>

using namespace std;

class Azione {
  private:
    string nome;
  public:
    Azione(string n) { nome=n; }
    virtual string dettagli() = 0;
    void debug() {
        cout << "dettagli(" << nome << "): " << dettagli() << "\n";
    }
};

class Tweet : public Azione {
  private:
    string messaggio;
  public:
    using Azione::Azione;
    Tweet(string n, string m) : Tweet(n) { messaggio=m; }
    void setMessage(string n, string m) {
        messaggio=m;
    }
    string dettagli() {
        string risultato = messaggio;
```

```
        return risultato; }
};

class Email : public Azione {
  private:
    string messaggio;
    string destinatario;
  public:
    using Azione::Azione;
    Email(string n, string m, string d) : Email(n) { messaggio=m; destinatario=d; }
    void setMessage(string n, string m, string d) {
        messaggio=m;
        destinatario=d;
    }
    string dettagli() {
        string risultato = messaggio + "(" + destinatario + ")";
        return risultato; }
};

int main() {
   Tweet t = Tweet(string("a tweet"), "Mi dispiace se oggi è stata molto pesante la lezione!")
   Email e = Email(string("an email"), "vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.c
   t.debug();
   e.debug();
}
```

1.3.4 infine possiamo eliminare la *setMessage* non più necessaria

```
22a23,25
>     void setMessage(string n, string m) {
>         messaggio=m;
>     }
34a38,41
>     void setMessage(string n, string m, string d) {
>         messaggio=m;
>         destinatario=d;
>     }
```

```
#include <iostream>
#include <string>

using namespace std;

class Azione {
  private:
    string nome;
  public:
    Azione(string n) { nome=n; }
```

```
    virtual string dettagli() = 0;
    void debug() {
        cout << "dettagli(" << nome << "): " << dettagli() << "\n";
    }
};

class Tweet : public Azione {
  private:
    string messaggio;
  public:
    using Azione::Azione;
    Tweet(string n, string m) : Tweet(n) { messaggio=m; }
    string dettagli() {
        string risultato = messaggio;
        return risultato; }
};

class Email : public Azione {
  private:
    string messaggio;
    string destinatario;
  public:
    using Azione::Azione;
    Email(string n, string m, string d) : Email(n) { messaggio=m; destinatario=d; }
    string dettagli() {
        string risultato = messaggio + "(" + destinatario + ")";
        return risultato; }
};

int main() {
   Tweet t = Tweet(string("a tweet"), "Mi dispiace se oggi è stata molto pesante la lezione!")
   Email e = Email(string("an email"), "vedi tweet che ti ho inviato", "andrea.ribuoli@yahoo.c
   t.debug();
   e.debug();
}
```

[7]:
```
!g++ -o abstract abstract.cpp
```

[8]:
```
!./abstract
```

```
dettagli(tweet): Mi dispiace se oggi è stata molto pesante la lezione!
dettagli(email): vedi tweet che ti ho inviato(andrea.ribuoli@yahoo.com)
```

[9]:
```
!g++ -o abstract abstract2.cpp
```

[10]:
```
!./abstract
```

```
dettagli(tweet): Mi dispiace se oggi è stata molto pesante la lezione!
dettagli(email): vedi tweet che ti ho inviato(andrea.ribuoli@yahoo.com)
```

[14]:
```
!g++ -o abstract abstract3.cpp
```

[15]:
```
!./abstract
```

```
dettagli(a tweet): Mi dispiace se oggi è stata molto pesante la lezione!
dettagli(an email): vedi tweet che ti ho inviato(andrea.ribuoli@yahoo.com)
```

[16]:
```
!g++ -o abstract abstract4.cpp
```

[17]:
```
!./abstract
```

```
dettagli(a tweet): Mi dispiace se oggi è stata molto pesante la lezione!
dettagli(an email): vedi tweet che ti ho inviato(andrea.ribuoli@yahoo.com)
```

[1]:
```
!g++ -o abstract abstract5.cpp
```

[2]:
```
!./abstract
```

```
dettagli(a tweet): Mi dispiace se oggi è stata molto pesante la lezione!
dettagli(an email): vedi tweet che ti ho inviato(andrea.ribuoli@yahoo.com)
```

1.4 Gestione liste in C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct _Elem {
  char nome[24];
  struct _Elem *pElem;
} Elem;

int main(int argc, char *argv[]) {
  Elem *root, *last, *p;
  FILE *file;
  char riga[24];

  if (argc != 2) {
     printf("Usage: liste <nomefile>\n");
     exit(-1);
  }
  file = fopen(argv[1],"r");
  if (file == NULL) {
      printf("il file %s non esiste nella posizione indicata\n",argv[1]);
      exit(-2);
  }
  root = NULL;
  last = NULL;
  while( fgets(riga, sizeof(riga), file) != NULL) {
```

```
    p = (Elem *) malloc(sizeof(Elem));
    strncpy(p->nome, riga, strlen(riga)-1);
    p->pElem = NULL;
    if (root == NULL) { root = p; }
    if (last != NULL) { last->pElem = p; }
    last = p;
  }
  fclose(file);

  p = root;
  while (p != NULL) {
    printf("%s\n", p->nome);
    p = p->pElem;
  }
}
```

[5]:
```
!gcc -o liste liste.c
```

[7]:
```
!liste lista.txt
```

```
abc
def
ghi
jkl
mno
pqr
stu
vwy
z
```