

2021_12_14

December 14, 2021

1 2021-12-14

1.0.1 Corso ITS

1.1 Magento & e-commerce software

1.2 ### Fondamenti di Programmazione (Andrea Ribuoli)

Nel 1979 **Bjarne Stroustrup** lavorava ai **Bell Laboratories** società che ha dato alla informatica innumerevoli contributi tra cui il sistema operativo Unix e svariati linguaggi di programmazione. Dal 2016 i Bell Labs sono parte della NOKIA.

Nel maggio del 1979 **Bjarne Stroustrup** iniziò a lavorare su un precompilatore per il linguaggio C cui diede il nome di **C with Classes** ispirandosi al linguaggio **Simula**.

Di fatto una **classe** nel *C with Classes* è più propriamente un **tipo**, un *tipo definito dall'utente* ma Stroustrup preferì conservare il nome adottato dal linguaggio Simula.

Con la notazione seguente nel 1980 Stroustrup introduceva con un report interno il concetto:

```
class stack {
    char s[SIZE];
    char *min;
    char *top;
    char *max;
    void new();
public:
    void push(char);
    char pop();
}
```

Nel gennaio 1982 viene pubblicato il primo articolo pubblico sul *C with Classes* e nel 1983 viene adottato il nome **C++** il cui primo rilascio esterno agli AT&T Bell Labs risale al 1985.

Nell'ottobre di quell'anno viene completato (ma pubblicato nel 1986) **The C++ Programming Language**. Oggi è arrivato alla sua quarta edizione.

Nel dicembre 1987 viene rilasciato il primo supporto GNU per il C++.

Dal 1998 il linguaggio è **standardizzato** ed ha subito svariate revisioni (2003, 2011, 2014, 2017, 2020) che dal 2012 hanno cadenza triennale.

La convenzione è di qualificare il nome C++ con le ultime due cifre dell'anno per riferirsi allo standard: **C++98**, **C++03**, **C++11**, **C++14**, **C++17** e **C++20**.

```
[1]: !/QOpenSys/riby/bin/gcc -v
```

```
Using built-in specs.
COLLECT_GCC=/QOpenSys/riby/bin/gcc
COLLECT_LTO_WRAPPER=/QOpenSys/riby/libexec/gcc/powerpc-ibm-aix7.1/11.2.0/lto-
wrapper
Target: powerpc-ibm-aix7.1
Configured with: ../../srcs/gcc-11.2.0/configure --build=powerpc-ibm-aix7.1
--prefix=/QOpenSys/riby --with-cpu=default64 --enable-languages=c,c++ --with-
mpc=/QOpenSys/riby --with-mpfr=/QOpenSys/riby --with-gmp=/QOpenSys/riby
Thread model: aix
Supported LTO compression algorithms: zlib
gcc version 11.2.0 (GCC)
```

```
[2]: !/QOpenSys/riby/bin/g++ -std=c++11 -dM -E -x c++ /dev/null \
    | grep -F __cplusplus ; \
/QOpenSys/riby/bin/g++ -std=c++14 -dM -E -x c++ /dev/null \
    | grep -F __cplusplus ; \
/QOpenSys/riby/bin/g++ -dM -E -x c++ /dev/null \
    | grep -F __cplusplus ; \
/QOpenSys/riby/bin/g++ -std=c++20 -dM -E -x c++ /dev/null \
    | grep -F __cplusplus
```

```
#define __cplusplus 201103L
#define __cplusplus 201402L
#define __cplusplus 201703L
#define __cplusplus 202002L
```

```
[5]: !g++ -std=c++11 -dM -E -x c++ /dev/null | grep -F __cplusplus ; \
    g++ -dM -E -x c++ /dev/null | grep -F __cplusplus
```

```
#define __cplusplus 201103L
#define __cplusplus 201402L

#include <iostream>
using namespace std;
double square(double x)
{
    return x*x;
}

void print_square(double x)
{
    cout << "the square of " << x << " is " << square(x) << "\n";
}

int main()
{
    print_square(1.234);
}
```

```
}
```

```
[8]: !/QOpenSys/riby/bin/g++ -o simple simple.cpp
```

```
[9]: !./simple
```

the square of 1.234 is 1.52276

1.3 Inizializzazione

- come in **C** con il simbolo `=`
- con la sola presenza di valori tra parentesi graffe

attenzione alle *narrowing conversion*

```
[ ]: !/QOpenSys/riby/bin/g++ -o error error.cpp
```

```
[11]: !/QOpenSys/riby/bin/g++ -o complex complex.cpp
```

```
[12]: !./complex
```

(3.5 + 7.5 i) * (2.5 + 4.5 i) = (-25 + 34.5 i)

```
[13]: !/QOpenSys/riby/bin/g++ -o vector vector.cpp
```

```
[14]: !./vector
```

-3 , 22 , -9 , 12

Al di là dei cosiddetti **tipi incorporati**, il C++ consente al programmatore la possibilità di costruire elementi di alto livello attraverso una serie di **meccanismi di astrazione**.

I tipi così personalizzati avranno rappresentazioni e operazioni idonee e potranno essere utilizzati in modo elegante e semplice.

1.3.1 LA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

Con la possibilità di evolvere dalla programmazione **C** verso il **C++** si introdusse un potente argomento per avvicinarsi alla **programmazione ad oggetti**.

- gli **oggetti** sono creati **istanziando** una **classe**
- è possibile definire nuove classi a partire da classi già esistenti attraverso l'**ereditarietà**

Quando una sotto-classe eredita da una classe padre, includerà tutte le definizioni di dati e operazioni che la classe padre definisce.

ParentClass

Operations()

Subclass
ExtraOps()

Una sotto-classe può effettuare l'**override** (la *ridefinizione*) di alcuni metodi forniti dalle classi “*antenate*” (la catena gerarchica può avere più livelli). Questa possibilità consente l'introduzione di vere e proprie **classi astratte**: solo le sottoclassi delle quali possono essere effettivamente istanziate.

Alcuni linguaggi ad oggetti consentono anche la ereditarietà **multipla** adottando approcci diversi per risolvere alcuni potenziali problemi

L'enfasi si è gradualmente spostata dalla **programmazione** al **progetto (design)** e il termine:

1.4 OBJECT ORIENTED DESIGN

è diventato *cool*. Alcuni autori hanno fatto la storia di questa branca dell'informatica introducendo i cosiddetti

1.5 DESIGN PATTERNS

Il testo (fondamentale) che porta questo nome *non insegna una notazione particolare, e neppure l'ultimo linguaggio di programmazione, ma spiega come risolvere in modo elegante, semplice e rapido alcuni problemi che i programmatori incontrano spesso*

Nascono con la tesi di **Erich Gamma** (1991). C'erano già metà degli attuali design pattern. **Richard Helm** si unisce al progetto nel 1991 e poco dopo si unisce anche **John Vlissides**. Nel 1992 si unisce anche **Ralph Johnson**.

1.6 Catalogo dei design pattern

1.6.1 Pattern creazionali

- **Abstract Factory**
- **Builder**
- **Factory method**
- **Prototype**
- **Singleton** (ex Solitarie)

1.6.2 Pattern strutturali

- **Adapter**
- **Bridge**
- **Composite**
- **Decorator** (ex Wrapper)
- **Facade** (ex Glue)
- **Flyweight**
- **Proxy**

1.6.3 Pattern comportamentali

- Chain of responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor (ex Walker)

*Gli esperti di progettazione hanno immediatamente riconosciuto il valore del catalogo fin dagli albori, ma gli unici in grado di **capire i pattern** erano le persone che li avevano già usati*

```
[64]: !/QOpenSys/riby/bin/g++ -o abstract abstract.cpp
```

```
[65]: !abstract
```

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

1.7 abstract.cpp

```
#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 = destinatario + "(" + 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();
}

```