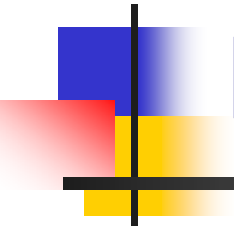


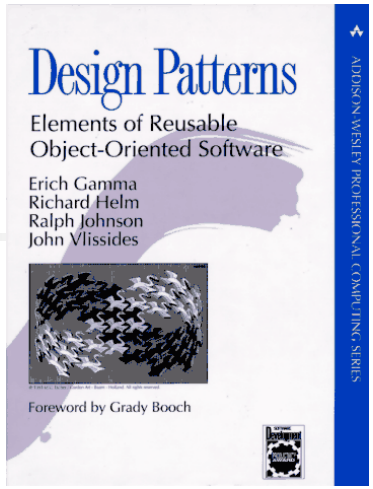
Due slides sui Design Patterns

Luciano Pandola

INFN-LNGS

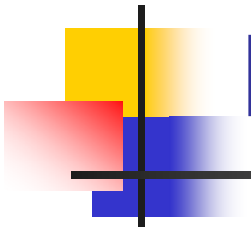
Corso INFN su C++, ROOT e Geant4

Design Patterns



Gamma, Helm, Johnson and Vlissides, Design Patterns, Addison-Wesley 1995, ISBN 0-201-63361-2 (Gang-of-Four)

- Ciascun "**design pattern**" (schema di progettazione) individua, spiega e risolve un problema di design ricorrente in sistemi object-oriented
- Un **design pattern** può essere definito "*una soluzione progettuale generale a un problema ricorrente*"
  - **non è una libreria**, quanto piuttosto una descrizione o un **modello** da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del software



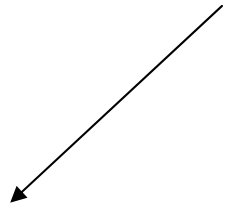
Design Patterns

- Nel libro sono descritti 23 tipi di design pattern, suddivisi in 3 categorie: strutturali, creazionali e comportamentali.
- Creazionali
  - nascondono i costruttori delle classi e mettono dei metodi al loro posto creando un'interfaccia
- Strutturali
  - Consentono di utilizzare degli oggetti esistenti fornendo agli utilizzatori un'interfaccia più adatta alle loro esigenze
- Comportamentali
  - forniscono soluzione alle più comuni tipologie di interazione tra gli oggetti

Lista dei design patterns

- Abstract Factory
- Adapter
- Bridge
- Builder
- Chain of Responsibility
- Command
- Composite
- Decorator
- Facade
- Factory Method
- Flyweight
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- Prototype
- Proxy
- **Singleton**
- State
- Strategy
- Template Method
- Visitor

Usato in applicazioni Geant4

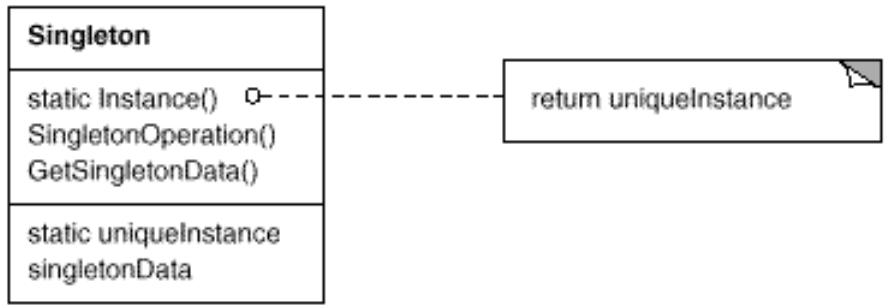


Singleton (“singoletto”)

- Design Pattern per assicurare che una classe abbia **una e una sola istanza** nel corso di un’esecuzione
  - Il costruttore è privato → non si può creare *direttamente* un’istanza della classe
- Utile quando c’è bisogno di avere di gestire risorse comuni, condivise da più parti diverse del programma (“classe globale”)
  - Logging, gestione database
  - Gestione analisi (Geant4)
    - Accumulo di energia, booking istogrammi, etc.

Singleton (“singoletto”)

- L’intefaccia fornisce anche un **punto di accesso** univoco alla classe
  - metodo **`Get()`** (pubblico e statico) che restituisce l’unica istanza ammissibile della classe
  - Il Getter è anche responsabile di creare l’istanza, la prima volta che è chiamato



Implementazione di un singleton - definizione

```
class Analysis {
public:
        static Analysis* GetInstance() ;
        virtual ~Analysis() {};
private:
        Analysis();
        static Analysis* singleton;
};
```

Interfaccia pubblica

Costruttore privato

Unica istanza

```
Analysis* Analysis::GetInstance(){
      if ( singleton == NULL )
            singleton = new Analysis();
       return singleton;
}
```

Restituisce l'unica istanza (creandola, se ancora non esiste)

Utilizzo di un singleton

- Possibile da **dovunque** nel programma e nelle altre classi

```
#include “Analysis.hh”
…
```

No: costruttore privato

```
Analysis* theAnalysis = new Analysis();
```

```
Analysis* theAnalysis = Analysis::GetInstance();
```

OK

```
theAnalysis->CallAMethod(…)
```

È sempre la **stessa istanza** (da dovunque chiamata)