Introduzione a Design Pattern

Elementi di software orientato agli oggetti riutilizzabile

D. Roversi

Laboratorio di Informatica Free e Open 

April 20, 2009

Outline

Outline

Perché usare i Pattern

- Scrivere programmi ad oggetti più facilmente modificabili e riutilizzabili.
- Creare una nomenclatura comune.

Perché usare i Pattern

- Scrivere programmi ad oggetti più facilmente modificabili e riutilizzabili.
- Creare una nomenclatura comune.

Outline

Cos'è un Pattern

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the soluzion, to that problem, in such a way that you can use this solution a million times over, without ever doing in the same way twice"
- E' una soluzione a una classe di problemi ricorrenti
- E' uno "schema" di soluzione che riguarda una o più classi o oggetti e come sono interagiscono fra loro

Cos'è un Pattern

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the soluzion, to that problem, in such a way that you can use this solution a million times over, without ever doing in the same way twice"
- E' una soluzione a una classe di problemi ricorrenti
- E' uno "schema" di soluzione che riguarda una o più classi o oggetti e come sono interagiscono fra loro

Cos'è un Pattern

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the soluzion, to that problem, in such a way that you can use this solution a million times over, without ever doing in the same way twice"
- E' una soluzione a una classe di problemi ricorrenti
- E' uno "schema" di soluzione che riguarda una o più classi o oggetti e come sono interagiscono fra loro

Outline

Come è fatto un pattern

- Nome
- Motivazione: quale problema intende risolvere
- Soluzione: come risolve il problema
- Consequenze: vantaggi o limiti della soluzione

Come è fatto un pattern

- Nome
- Motivazione: quale problema intende risolvere
- Soluzione: come risolve il problema
- Consequenze: vantaggi o limiti della soluzione

Come è fatto un pattern

- Nome
- Motivazione: quale problema intende risolvere
- Soluzione: come risolve il problema
- Consequenze: vantaggi o limiti della soluzione

Come è fatto un pattern

- Nome
- Motivazione: quale problema intende risolvere
- Soluzione: come risolve il problema
- Consequenze: vantaggi o limiti della soluzione

Outline

Singleton

- Motivazione: in alcuni casi e' importante che esista una sola istanza di una certa classe, e che possa essere facilmente utilizzabile da un qualsiasi punto del programma.
- Soluzione: una classe con un costruttore privato, e un metodo getInstance() che restituisce l'istanza unica.

Singleton

- Motivazione: in alcuni casi e' importante che esista una sola istanza di una certa classe, e che possa essere facilmente utilizzabile da un qualsiasi punto del programma.
- Soluzione: una classe con un costruttore privato, e un metodo getInstance() che restituisce l'istanza unica.

Singleton

```
class Singleton {
  private Singleton() { };
  private static Singleton instance=null;
  public getInstance() {
    if (instance==null)
      instance=new Singleton();
    return instance;
  }
}
```

Singleton

- Conseguenze:
  - Non è possibile creare più di un istanza: il costruttore è privato.
  - è possibile tener traccia di chi usa l'istanza
  - è possibile in un secondo tempo cambiare il numero di istanze esistenti
  - è possibile creare l'oggetto solo se ce ne è veramente bisogno
  - di contro non è semplice passare parametri al costruttore

Singleton

- Conseguenze:
  - Non è possibile creare più di un istanza: il costruttore è privato.
  - è possibile tener traccia di chi usa l'istanza
  - è possibile in un secondo tempo cambiare il numero di istanze esistenti
  - è possibile creare l'oggetto solo se ce ne è veramente bisogno
  - di contro non è semplice passare parametri al costruttore

Singleton

- Conseguenze:
  - Non è possibile creare più di un istanza: il costruttore è privato.
  - è possibile tener traccia di chi usa l'istanza
  - è possibile in un secondo tempo cambiare il numero di istanze esistenti
  - è possibile creare l'oggetto solo se ce ne è veramente bisogno
  - di contro non è semplice passare parametri al costruttore

Singleton

- Conseguenze:
  - Non è possibile creare più di un istanza: il costruttore è privato.
  - è possibile tener traccia di chi usa l'istanza
  - è possibile in un secondo tempo cambiare il numero di istanze esistenti
  - è possibile creare l'oggetto solo se ce ne è veramente bisogno
  - di contro non è semplice passare parametri al costruttore

Singleton

- Conseguenze:
  - Non è possibile creare più di un istanza: il costruttore è privato.
  - è possibile tener traccia di chi usa l'istanza
  - è possibile in un secondo tempo cambiare il numero di istanze esistenti
  - è possibile creare l'oggetto solo se ce ne è veramente bisogno
  - di contro non è semplice passare parametri al costruttore

Outline

Iterator

- Motivazione: accedere facilmente agli elementi di un aggregato di oggetti senza doversi preoccupare di come sono organizzati.
- Soluzione: l'aggregato restituisce un oggetto, che a sua volta restituisce uno ad uno gli elementi dell'aggregato

Iterator

- Motivazione: accedere facilmente agli elementi di un aggregato di oggetti senza doversi preoccupare di come sono organizzati.
- Soluzione: l'aggregato restituisce un oggetto, che a sua volta restituisce uno ad uno gli elementi dell'aggregato

Iterator

Iterator

```
interface Iterator {
  public Object next();
  public boolean hasMore();
}
Aggregate class Aggregate{
  ...
  public Iterator getIterator() {
    return new AggregateIterator();
  }
  class AggregateIterator() implements Iterator {
    ...
  }
}
```

Iterator

- Conseguenze:
  - Possibilità di vedere gli elementi di un aggregato in maniera uniforme, senza dover sapere i dettagli dell'aggregato.
  - Possibilità di avere pezzi di codice che scorrono autonomamente lo stesso aggregato
  - Avere un interfaccia comune per scorrere gli oggetti di un aggregato.

Iterator

- Conseguenze:
  - Possibilità di vedere gli elementi di un aggregato in maniera uniforme, senza dover sapere i dettagli dell'aggregato.
  - Possibilità di avere pezzi di codice che scorrono autonomamente lo stesso aggregato
  - Avere un interfaccia comune per scorrere gli oggetti di un aggregato.

Iterator

- Conseguenze:
  - Possibilità di vedere gli elementi di un aggregato in maniera uniforme, senza dover sapere i dettagli dell'aggregato.
  - Possibilità di avere pezzi di codice che scorrono autonomamente lo stesso aggregato
  - Avere un interfaccia comune per scorrere gli oggetti di un aggregato.

Outline

Visitor

- Motivazione: ripetere una stessa azione su tutti gli elementi di un aggregato
- Soluzione: creo un oggetto Visitor che viene passato all'aggregato. L'aggregato passa poi l'istanza Visitor a tutti gli elementi, e ne invocano un metodo, passando se stessi come argomento.

Visitor

- Motivazione: ripetere una stessa azione su tutti gli elementi di un aggregato
- Soluzione: creo un oggetto Visitor che viene passato all'aggregato. L'aggregato passa poi l'istanza Visitor a tutti gli elementi, e ne invocano un metodo, passando se stessi come argomento.

Visitor

- Conseguenze:
  - è più semplice aggiungere nuove operazioni
  - non serve modificare l'aggregato
  - a differenza dell'Iterator, non sono limitato ad un unico tipo di oggetto
  - ma rispetto all'iterator è più macchinoso da usare

Visitor

- Conseguenze:
  - è più semplice aggiungere nuove operazioni
  - non serve modificare l'aggregato
  - a differenza dell'Iterator, non sono limitato ad un unico tipo di oggetto
  - ma rispetto all'iterator è più macchinoso da usare

Visitor

- Conseguenze:
  - è più semplice aggiungere nuove operazioni
  - non serve modificare l'aggregato
  - a differenza dell'Iterator, non sono limitato ad un unico tipo di oggetto
  - ma rispetto all'iterator è più macchinoso da usare

Visitor

- Conseguenze:
  - è più semplice aggiungere nuove operazioni
  - non serve modificare l'aggregato
  - a differenza dell'Iterator, non sono limitato ad un unico tipo di oggetto
  - ma rispetto all'iterator è più macchinoso da usare