

MAC0413 - Tópicos de POO

Secret Partner Pattern / Facet

Bruno Pera
Dairton L. Bassi Filho

1

PLoP 2002

The Secret Partner Pattern

2

The Secret Partner Pattern

- ◆ "The Secret Partner Pattern" modela a relação entre duas entidades: um "Partner" (parceiro) e um "Secret Partner" (parceiro secreto).
- ◆ Um parceiro secreto é assim chamado pois sua participação na parceria é mantida secreta do público.

3

The Secret Partner Pattern

- ◆ A relação entre o parceiro e o parceiro secreto deve ser bastante próxima, de modo que essas duas entidades possam se comunicar de maneira eficaz.
- ◆ Apesar disso o parceiro secreto pode possuir informações que são realmente secretas, isto é, nem mesmo o parceiro tem conhecimento.

4

Exemplo

- ◆ Considere dois desenvolvedores de software, um júnior e um sênior, que participam de um mesmo projeto.
- ◆ A relação entre eles pode, em alguns casos, apresentar as seguintes características:

5

Exemplo (continuação)

- ◆ Somente o desenvolvedor sênior interage com os clientes, participa de reuniões, etc. Desse modo o desenvolvedor júnior fica "protegido" do cliente.
- ◆ Como apenas o desenvolvedor sênior interage com o cliente, a comunicação entre os dois parceiros deve ser bastante eficiente.

6

Exemplo (continuação)

- ◆ Ainda assim o desenvolvedor júnior pode precisar manter algumas informações desconhecidas pelo sênior (como o seu salário, por exemplo).

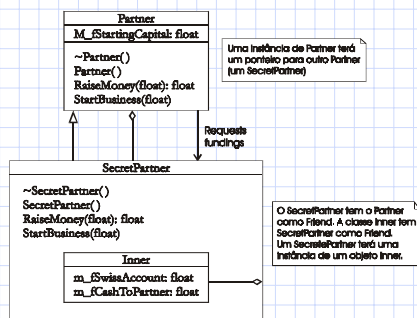
7

Problema

- ◆ Como podem duas entidades de "software" possuir um relacionamento bastante próximo e ainda assim permitir que uma delas mantenha informações desconhecidas pela outra?

8

Estrutura



9

Partner.h

```
#ifndef INCLUDE_PARTNER_H
#define INCLUDE_PARTNER_H

class Partner
{
public:
    Partner();
    virtual ~Partner();
    virtual void StartBusiness(float ventureCapital);
    virtual float RaiseMoney(float amountNeeded);

private:
    float m_fStartingCapital;
    Partner* m_pRef;
};

#endif //INCLUDE_PARTNER_H
```

10

SecretPartner.h

```
#ifndef INCLUDE_SECRETPARTNER_H
#define INCLUDE_SECRETPARTNER_H

#include "Partner.h"
class SecretPartner: Public Partner
{
public:
    friend class Partner;

protected:
    SecretPartner();
    virtual ~SecretPartner();
    virtual void StartBusiness(float ventureCapital);
    virtual float RaiseMoney(float amountRequested);
};
```

11

SecretPartner.h (continuação)

```
private:
    SecretPartner(const SecretPartner&);

class Inner
{
    friend class SecretPartner;

private:
    float m_fCashToPartner;
    float m_fSwissAccount;
};

    Inner myInner;
};

#endif //INCLUDE_SECRETPARTNER_H
```

12

Partner.cpp

```
#include "Partner.h"
#include "SecretPartner.h"

Partner::Partner()
{
    m_pRef = 0;
    m_fStartingCapital = 0;
}

void Partner::StartBusiness(float ventureCapital)
{
    m_fStartingCapital = ventureCapital;
    m_pRef = new SecretPartner;
    m_pRef->StartBusiness(ventureCapital);
}

float Partner::RaiseMoney(float amountNeeded)
{ return (m_pRef->RaiseMoney(amountNeeded)); }

Partner::~Partner() { delete m_pRef; }
```

13

SecretPartner.cpp

```
#include "SecretPartner.h"

SecretPartner::SecretPartner(): Partner() {}

void SecretPartner::StartBusiness(float cashToPartner)
{ myInner.m_fCashToPartner = cashToPartner;
  myInner.m_fSwissAccount = 100.0f*cashToPartner;
}

float SecretPartner::RaiseMoney(float amountRequested)
{ float amountGiven = 0.00f;
  if ((amountRequested <= myInner.m_fSwissAccount) &&
      (myInner.m_fSwissAccount >= 10000000f))
  {
      amountGiven = amountRequested * 0.9f;
      myInner.m_fSwissAccount -= amountGiven;
  }
  return (amountGiven);
}

SecretPartner::~SecretPartner() {}
```

14

Contra-indicação

- ◆ "The Secret Partner Pattern" não deve ser usado quando "friendship" não for necessária, nesses casos seria melhor usar os níveis de acesso "public", "protected" e "private".

15

Padrões Relacionados

- ◆ Semelhanças com "Façade" pelo fato de os clientes interagirem somente com o Partner.
- ◆ Também apresenta características do "Adapter", pois apesar de estar escondido, o "Secret Partner" é quem realmente faz o trabalho.

16

Exemplo de uso

- ◆ As classes de "streaming" de C++ tipicamente requerem o uso de "friendship" para implementar os vários operadores como << e >>. Mesmo assim as classes que concedem a amizade podem querer manter alguns atributos ou comportamentos em segredo.

17

Perguntas?

18

PLoP 2002

Facet

Um padrão para interfaces dinâmicas

19

Eric Crahen

crahen@cse.buffalo.edu

20

Contexto

Onde quer que seja desejável criar uma interface sensível a contexto, a fim de modificar ou controlar o comportamento aparente de um objeto.

21

Exemplo

Adicionar segurança à uma aplicação, isto implica inserir código que promove checagens em tempo de execução, dando ou não acesso às operações, dependendo do status e do nível de segurança.

22

Soluções Problemáticas I

- ◆ Criar subclasses
 - Ruim para projetos grandes
 - Repetição do mesmo código
 - Difícil de prestar manutenção
- ◆ Padrão Proxy
 - Porém, proxies são fixos, isto só trocaria o enfoque do problema.

23

Soluções Problemáticas II

- ◆ Padrão Decorator
 - Porém, como saber quais métodos estarão disponíveis a cada momento.

24

Solução

Usar um tipo especial de proxy chamado facet (faceta).
A faceta possui um sentinela (Sentry) que protege os objetos de operações indevidas, a proteção é baseada na consulta ao contexto atual.

25

Componentes I

Proxy

Implementa um conjunto de interfaces às quais o cliente tem acesso.

Target

Objeto cujo acesso deve ser restringido conforme o contexto vigente.

26

Componentes II

Context

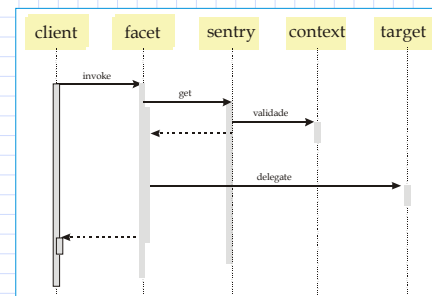
Toma as decisões de acesso aos targets em nome da faceta.

Sentry

Contém um conjunto de mapeamentos entre as interfaces e os targets.

27

Diagrama de Seqüência



28

Classe Facet

```
protected Sentry sentry;

protected Facet(Sentry sentry) { this.sentry = sentry; }
public abstract class Facet {
    public static Facet create(Sentry a, Class[] interfaces)
        throws FacetException
    public static Class[] query(Facet f, Class[] interfaces)
        throws Facet Exception;
    public static Facet narrow(Facet f, Class[] interfaces)
        throws Facet Exception;
}
```

29

Context.java

```
package polymorph.facet;

public interface Context {
    public Object validateComponent(Class interfaceClass,
        Object delegate);

    public boolean validateInterface(Class interfaceClass);
}
```

30

Target e interfaces

```
interface MutableAccount {  
    void credit(long amount);  
    void debit(long amount);  
}  
interface ImmutableAccount {  
    long balance();  
}  
class Account implements MutableAccount, ImmutableAccount {  
    void credit(long amount) { // ... }  
    void debit(long amount) { // ... }  
    long balance() { // ... }  
}
```

31

Críticas ao autor/padrão

- ◆ Alguns erros ortográficos
- ◆ Poucas divisões no texto
- ◆ Falta de clareza nas explicações
- ◆ Falta de diagramas

32

Referências

PLoP 2002

<http://jerry.cs.uiuc.edu/~plop/plop2002>

Exemplo completo

<http://www.cse.buffalo.edu/~crahen/projects/polymorph>

33

Perguntas ?

34