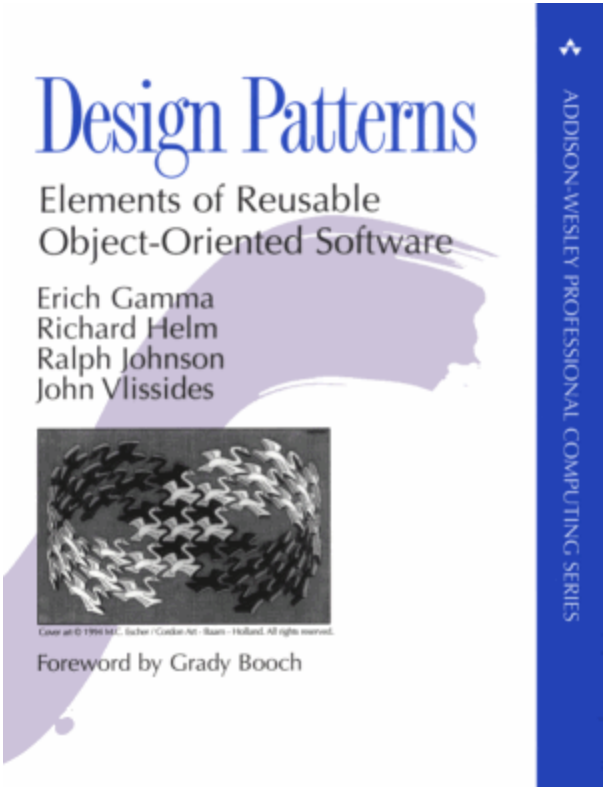
Design Pattern Implementation in Java and AspectJ

Jan Hannemann
Gregor Kiczales

In Proceedings of 2002 ACM SIGPLAN conference on OOPSLA. NY, USA.

Introdução

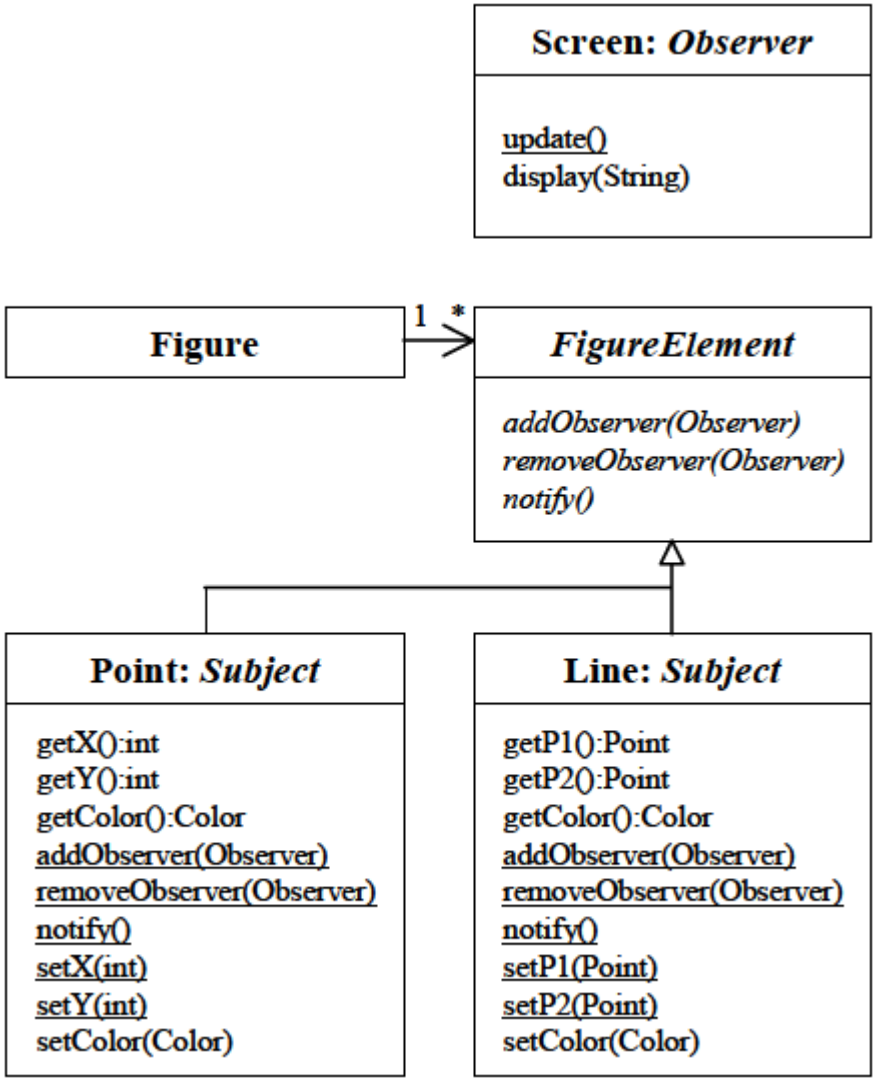

POWELLS TECHNICAL BOOKSTORE


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

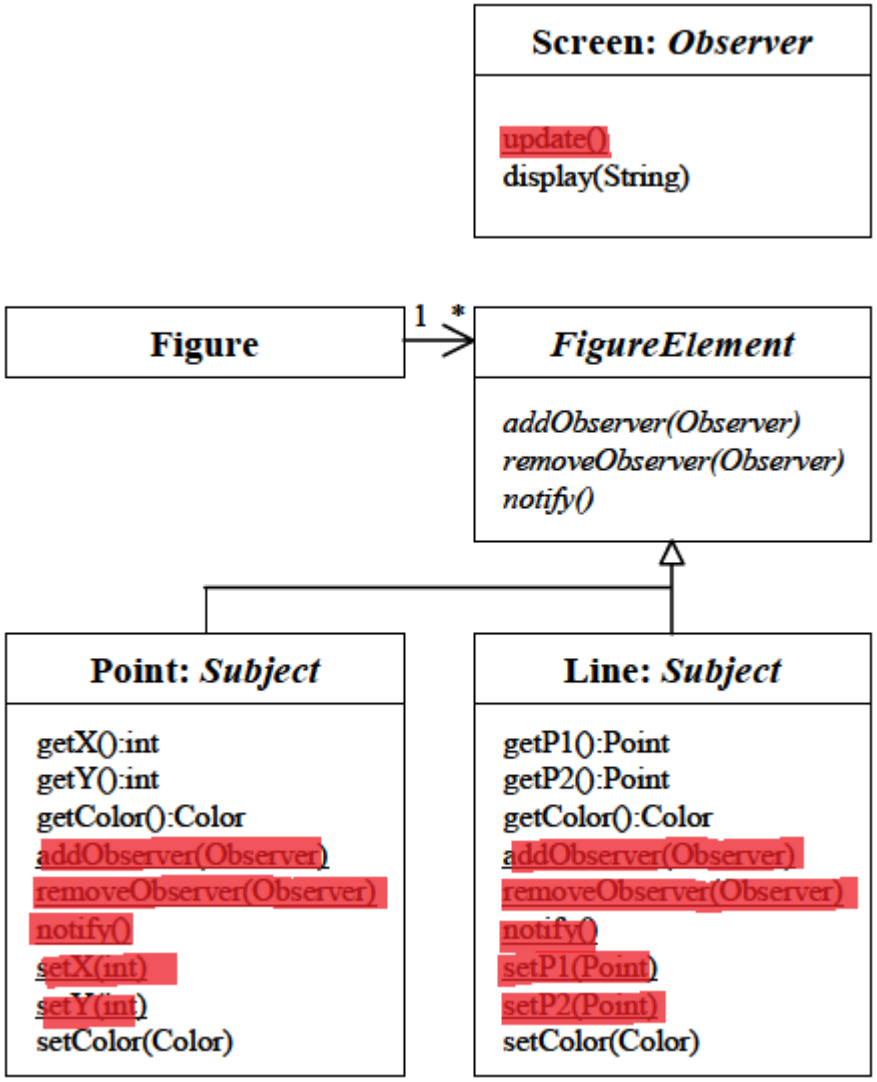
Introdução


Métodos e Atributos
Herança
Classes e Objetos
Classes Abstratas
Polimorfismo

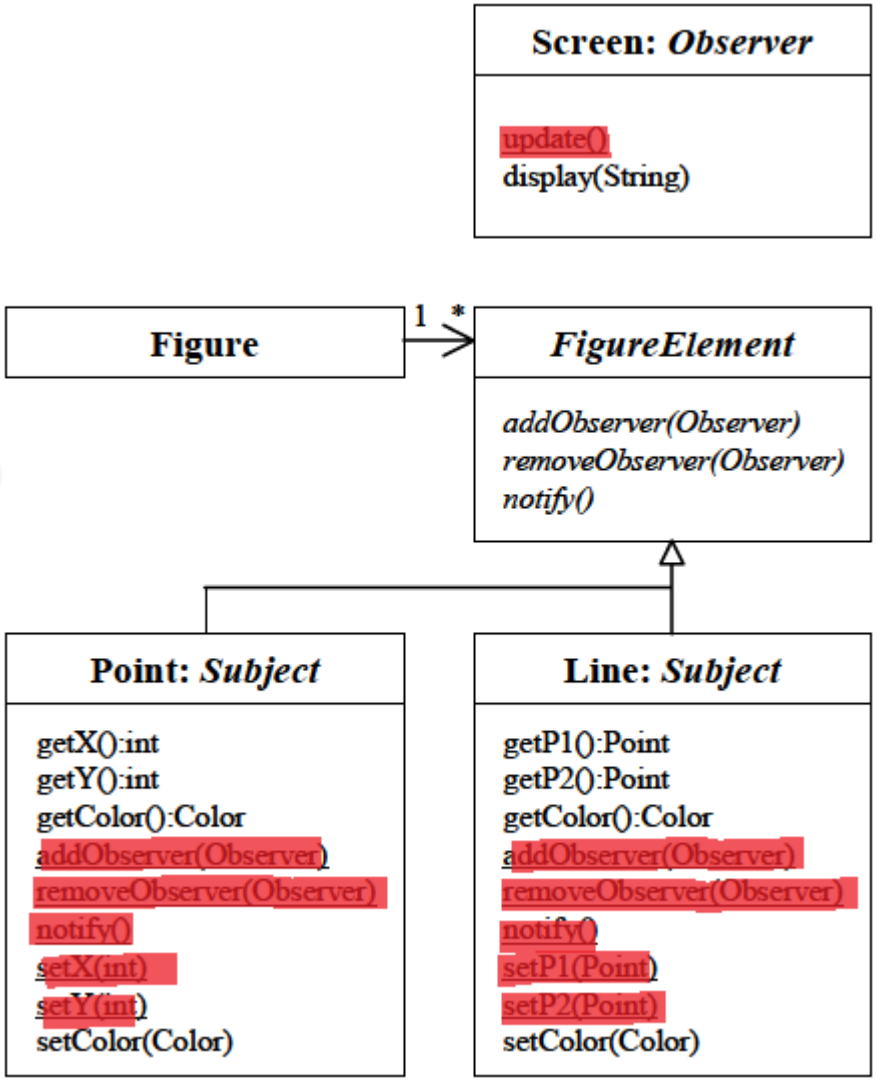
Introdução


Screen: Observer
update()
display(String)
Figure
1
*
FigureElement
addObserver(Observer)
removeObserver(Observer)
notify()
Point: Subject
getX():int
getY():int
getColor():Color
addObserver(Observer)
removeObserver(Observer)
notify()
setX(int)
setY(int)
setColor(Color)
Line: Subject
getP1():Point
getP2():Point
getColor():Color
addObserver(Observer)
removeObserver(Observer)
notify()
setP1(Point)
setP2(Point)
setColor(Color)

Introdução


Screen: Observer
update()
display(String)
Figure
1
*
FigureElement
addObserver(Observer)
removeObserver(Observer)
notify()
Point: Subject
getX():int
getY():int
getColor():Color
addObserver(Observer)
removeObserver(Observer)
notify()
setX(int)
setY(int)
setColor(Color)
Line: Subject
getP1():Point
getP2():Point
getColor():Color
addObserver(Observer)
removeObserver(Observer)
notify()
setP1(Point)
setP2(Point)
setColor(Color)

Introdução

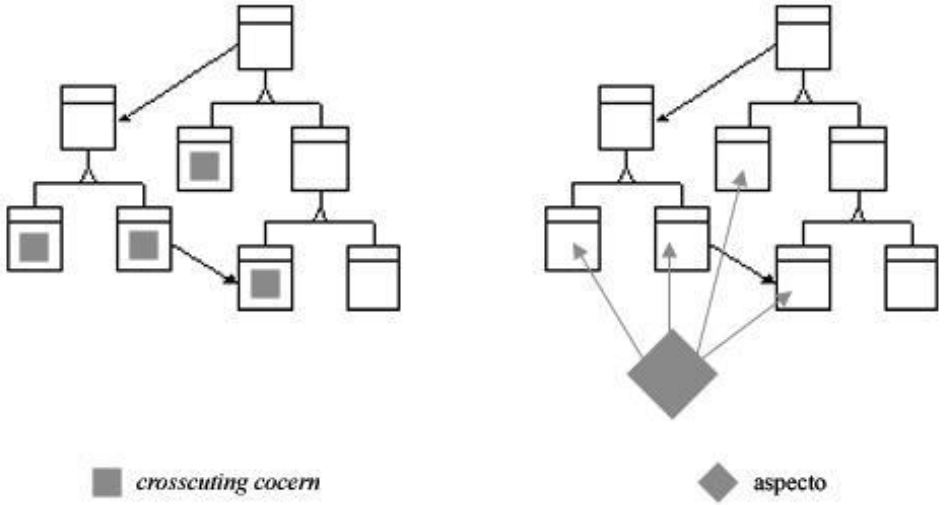


Reuso

?

Introdução

AspectJ



Proposta

State
Builder
Visitor
Proxy
Adapter
Abstract Factory
Strategy
Flyweight
Prototype
Command
Template Method
23
Interpreter
Singleton
Composite
Factory Method
Decorator
Observer
Mediator
Memento
Iterator
Bridge
Façade
Chain of Responsibility

Proposta


State
Builder
Visitor
Proxy
Adapter
Abstract Factory
Strategy
Flyweight
Prototype
Command
Template Method
23
Interpreter
Singleton
Composite
Factory Method
Java™
Decorator
Observer
Mediator
Memento
Iterator
Bridge
Façade
GoF
Chain of
Responsibility
C++

Proposta


Java™
State
Builder
Visitor
Proxy
Adapter
Abstract Factory
Strategy
Flyweight
Prototype
Command
Template Method
23
Interpreter
Singleton
Composite
Factory Method
Decorator
AspectJ
Observer
Mediator
Memento
Iterator
Bridge
Façade
Chain of
Responsibility

Proposta


State
Builder
Visitor
Proxy
Adapter
Abstract Factory
Strategy
Flyweight
Prototype
Command
Template Method
Interpreter
23
Singleton
Composite
+
Factory Method
Decorator
Observer
AspectJ
Mediator
Memento
Iterator
Bridge
Façade
Chain of
Responsibility
Java
Java

Padrão *Observer*

- Partes comuns na instanciação:
  - *Subject* e *Observer*
  - Mapeamento de *Subjects* para *Observers*
  - Mudança em *Subject* atualiza *Observer*

Padrão *Observer*

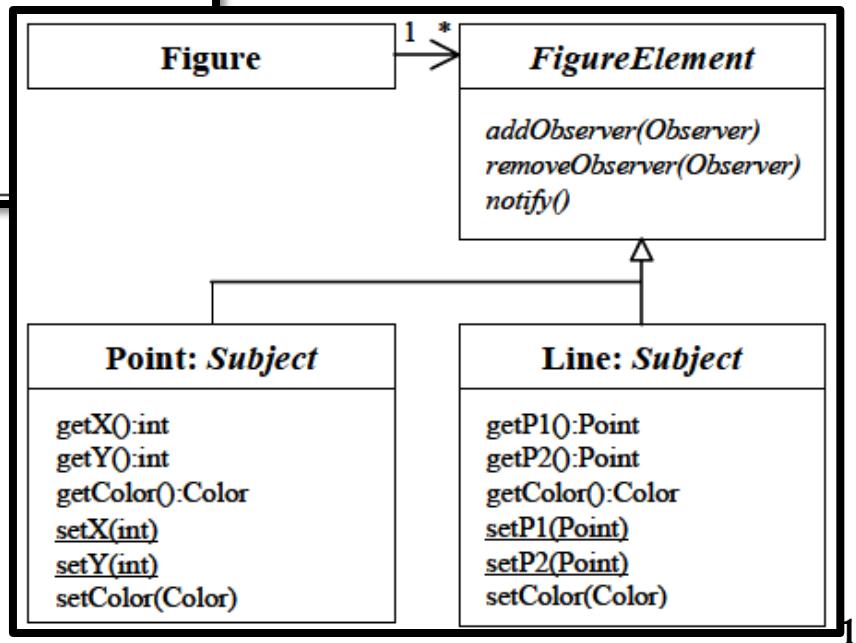
```
01 public abstract aspect ObserverProtocol {
02
03   protected interface Subject  { }
04   protected interface Observer { }
05
06   private WeakHashMap perSubjectObservers;07
08   protected List getObservers(Subject s) {
09     if (perSubjectObservers == null) {
10       perSubjectObservers = new WeakHashMap();
11     }
12     List observers =
13       (List)perSubjectObservers.get(s);
14     if ( observers == null ) {
15       observers = new LinkedList();
16       perSubjectObservers.put(s, observers);
17     }
18     return observers;
19   }
20
21   public void addObserver(Subject s,Observer o){
22     getObservers(s).add(o);
23   }
24   public void removeObserver(Subject s,Observer o){
25     getObservers(s).remove(o);
26   }
27
28   abstract protected pointcut
29     subjectChange(Subject s);
30
31   abstract protected void
32     updateObserver(Subject s, Observer o);
33
34   after(Subject s): subjectChange(s) {
35     Iterator iter = getObservers(s).iterator();
36     while ( iter.hasNext() ) {
37       updateObserver(s, ((Observer)iter.next()));
38     }
39   }
40 }
```

Padrão *Observer*

- Partes específicas da instanciação:
  - Classes *Subject* e *Observer*
  - Mudanças no *Subject*
  - Atualização do *Observer*

Padrão *Observer*

```
01 public aspect ColorObserver extends ObserverProtocol {
02
03   declare parents: Point  implements Subject;
04   declare parents: Line   implements Subject;
05   declare parents: Screen implements Observer;
06
07   protected pointcut subjectChange(Subject s):
08     (call(void Point.setColor(Color)) ||
09      call(void  Line.setColor(Color)) ) && target(s);
10
11   protected void updateObserver(Subject s,
12                                 Observer o) {
13     ((Screen)o).display("Color change.");
14   }
15 }
```



Padrão *Observer*

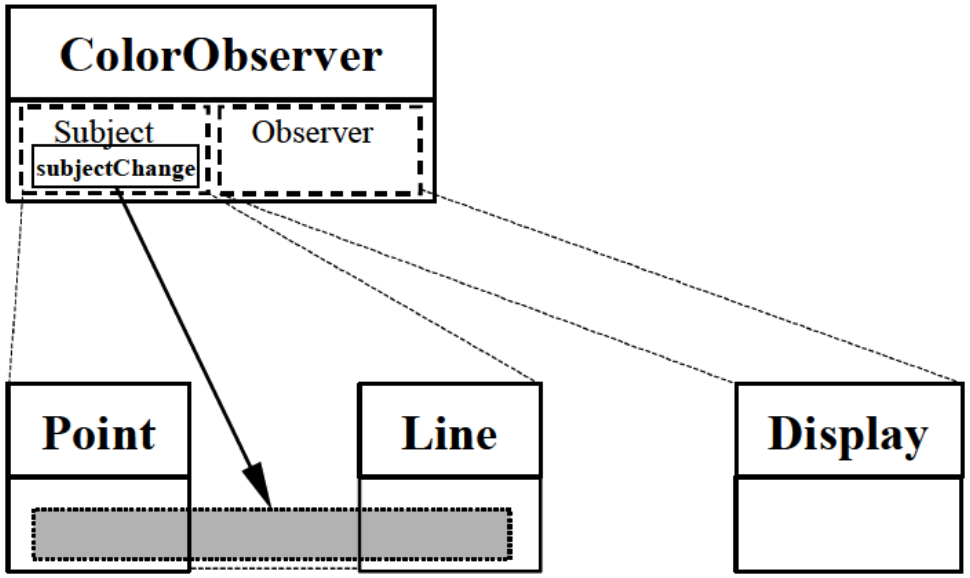
```
16 public aspect CoordinateObserver extends
17   ObserverProtocol {
18
19   declare parents: Point  implements Subject;
20   declare parents: Line   implements Subject;
21   declare parents: Screen implements Observer;
22
23   protected pointcut subjectChange(Subject s):
24      (call(void Point.setX(int))
25       || call(void Point.setY(int))
26       || call(void Line.setP1(Point))
27       || call(void Line.setP2(Point)) ) && target(s);
28
29   protected void updateObserver(Subject s,
30                                 Observer o) {
31     ((Screen)o).display("Coordinate change.");
32   }
33 }
```

Padrão *Observer*

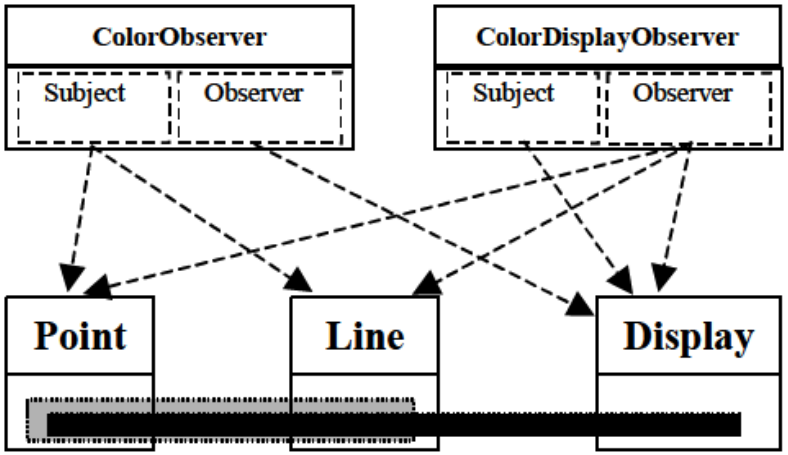
```
01 public aspect ScreenObserver
02                extends ObserverProtocol {
03
04  declare parents: Screen implements Subject;
05  declare parents: Screen implements Observer;
06
07  protected pointcut subjectChange(Subject s):
08    call(void Screen.display(String)) && target(s);
09
10  protected void updateObserver(
11    Subject s, Observer o) {
12      ((Screen)o).display("Screen updated.");
13  }
14 }
```

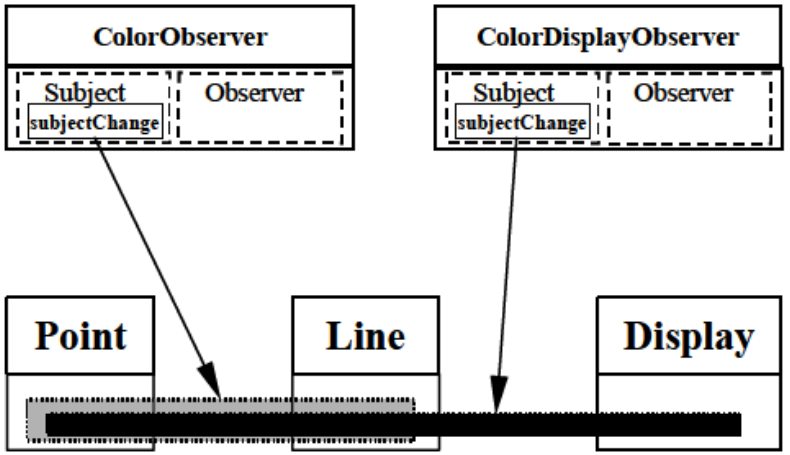
Padrão *Observer*

- Localidade
- Reusabilidade
- Composição
- Plugabilidade



Padrão *Observer*



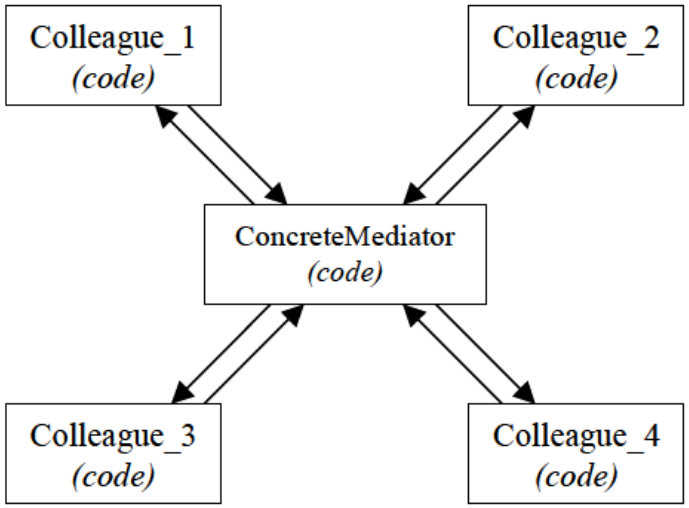


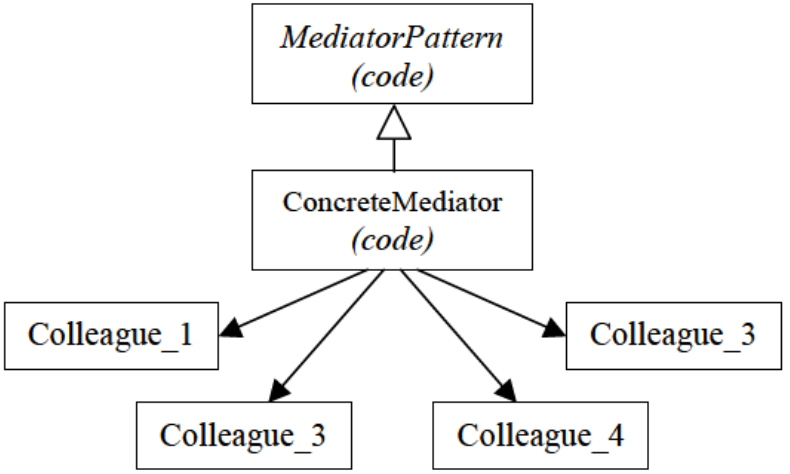
Padrões somente no aspecto

- Observer
- Composite
- Command
- Mediator
- Chain of Responsability

- Implementação no aspecto
- Classe abstrata geral
- Advice chama métodos apropriados

Mediator


Colleague_1
(code)
Colleague_2
(code)
ConcreteMediator
(code)
Colleague_3
(code)
Colleague_4
(code)


MediatorPattern
(code)
ConcreteMediator
(code)
Colleague_1
Colleague_3
Colleague_3
Colleague_4

Composition

```
public abstract aspect CompositionProtocol {

  protected interface Component {}
  protected interface Composite extends Component {}
  protected interface Leaf    extends Component {}

  private WeakHashMap perComponentChildren =
    new WeakHashMap();

  private Vector getChildren(Component s) {
    Vector children;
    children = (Vector)perComponentChildren.get(s);
    if ( children == null ) {
      children = new Vector();
      perComponentChildren.put(s, children);
    }
    return children;
  }

  public void addChild(Composite composite,
                       Component component) {
    getChildren(composite).add(component);
  }
  public void removeChild(Composite composite,
                          Component component) {
    getChildren(composite).remove(component);
  }

  public Enumeration getAllChildren(Component c) {
    return getChildren(c).elements();
  }

  protected interface FunctionVisitor {
    public Object doIt(Component c);
  }

  protected static Enumeration
   recurseFunction(Component c,
                   FunctionVisitor fv) {
    Vector results = new Vector();
    for (Enumeration enum = getAllChildren(c);
      enum.hasMoreElements(); ) {
        Component child;
        child = (Component)enum.nextElement();
        results.add(fv.doIt(child));
    }
    return results.elements();
  }
}
```

Composition

```
public aspect FileSystemComposite extends
              CompositeProtocol {

  declare parents: Directory implements Composite;
  declare parents: File      implements Leaf;

  public int sizeOnDisk(Component c) {
    return c.sizeOnDisk();
  }

  private abstract int Component.sizeOnDisk();

  private int Directory.sizeOnDisk() {
    int diskSize = 0;
    java.util.Enumeration enum;
    for (enum =
     SampleComposite.aspectOf().getAllChildren(this);
     enum.hasMoreElements(); ) {
       diskSize +=
        ((Component)enum.nextElement()).sizeOnDisk();
    }
    return diskSize;
  }

  private int File.sizeOnDisk()      {
    return size;
  }
}         }
```

Factories de objetos no aspecto

- Singleton
- Prototype
- Memento
- Iterator
- Flyweight

  - Classe abstrata
  - Métodos parametrizados
  - Métodos incluídos nos participantes

Factories de objetos em aspecto

- Singleton
- Prototype
- Memento
- Iterator
- Flyweight

muda construtor retornando objeto com um advice *around*

- Classe abstrata
- Métodos parametrizados
- Métodos incluídos nos participantes

Language constructs

- Adapter
- Decorator
- Strategy
- Visitor
- Proxy

  - Simples
  - Modulares
  - Limitações de herança

Herança múltipla

- Abstract Factory
- Factory Method
- TemplateMethod
- Builder
- Bridge

  - Não melhoram o reuso
  - Classes abstratas por interface

Código espalhado modularizado

- Abstract Factory
- Factory Method
- TemplateMethod
- Builder
- Bridge

- Alto acoplamento
- Aspecto modulariza parte

Sem benefícios do aspecto

- Facade

Design Pattern Implementation in Java and AspectJ

	Modularity Properties				Kinds of Roles	
Pattern Name	Locality(**)	Reusability	Composition Transparency	(Un)pluggability	Defining(*)	Superimposed
Façade	Same implementation for Java and AspectJ				Façade	-
Abstract Factory	no	no	no	no	Factory, Product	-
Bridge	no	no	no	no	Abstraction, Implementor	-
Builder	no	no	no	no	Builder, (Director)	-
Factory Method	no	no	no	no	Product, Creator	-
Interpreter	no	no	n/a	no	Context, Expression	-
Template Method	(yes)	no	no	(yes)	(AbstractClass), (ConcreteClass)	(AbstractClass), (ConcreteClass)
Adapter	yes	no	yes	yes	Target, Adapter	Adaptee
State	(yes)	no	n/a	(yes)	State	Context
Decorator	yes	no	yes	yes	Component, Decorator	ConcreteComponent
Proxy	(yes)	no	(yes)	(yes)	(Proxy)	(Proxy)
Visitor	(yes)	yes	yes	(yes)	Visitor	Element
Command	(yes)	yes	yes	yes	Command	Commanding, Receiver
Composite	yes	yes	yes	(yes)	(Component)	(Composite, Leaf)
Iterator	yes	yes	yes	yes	(Iterator)	Aggregate
Flyweight	yes	yes	yes	yes	FlyweightFactory	Flyweight
Memento	yes	yes	yes	yes	Memento	Originator
Strategy	yes	yes	yes	yes	Strategy	Context
Mediator	yes	yes	yes	yes	-	(Mediator), Colleague
Chain of Responsibility	yes	yes	yes	yes	-	Handler
Prototype	yes	yes	(yes)	yes	-	Prototype
Singleton	yes	yes	n/a	yes	-	Singleton
Observer	yes	yes	yes	yes	-	Subject, Observer

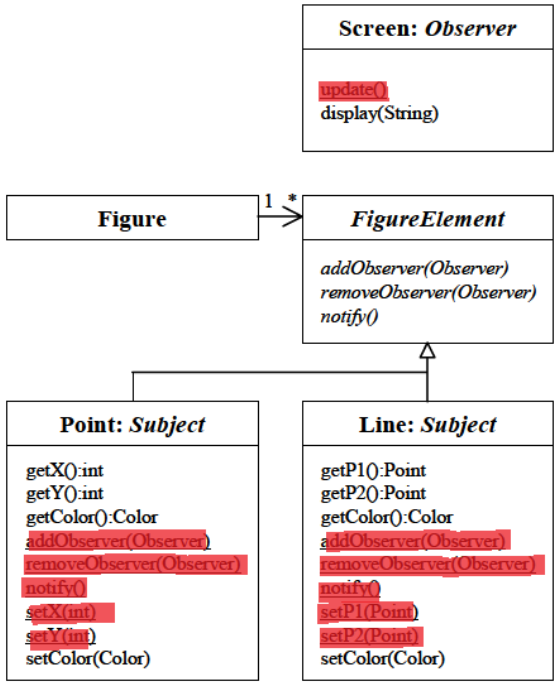
Trabalhos relacionados

- Padrões em outros paradigmas de linguagens
- Geração automática de padrões
- Classificar padrões
- Problema de composição
- Melhorar representação dos padrões

Benefícios do aspecto

- Localidade (17/23)
- Reusabilidade (12/17)
- Inversão de dependência
- Composição transparente
- Plugabilidade

Conclusões


Screen: Observer
update()
display(String)
Figure
1
*
FigureElement
addObserver(Observer)
removeObserver(Observer)
notify()
Point: Subject
getX():int
getY():int
getColor():Color
addObserver(Observer)
removeObserver(Observer)
notify()
setX(int)
setY(int)
setColor(Color)
Line: Subject
getP1():Point
getP2():Point
getColor():Color
addObserver(Observer)
removeObserver(Observer)
notify()
setP1(Point)
setP2(Point)
setColor(Color)

Conclusões

- *Defining* – sem melhorias
- *Defining* e *superimposed* – cada caso
- *Superimposed* – mais benefícios

Resultados

- 74% mais modulares
- 52% reusáveis
- Desacopla reduzindo dependências
- Padrões explícitos

Código dos padrões

- http://www.cs.ubc.ca/labs/spl/projects/aodps.html


VIDADEPROGRAMADOR
.COM.BR
/* SUGESTÃO ENVIADA POR DANIEL PEDROSO */
#165
CARA, O QUE VOCÊ ACHA SOBRE DESIGN PATTERNS?
BACANA, MAS O MELHOR PATTERN PARA RESOLVER NOSSO PROBLEMA É O "RCP"...
RCP? O QUE É RCP?
REUSE BY COPY AND PASTE
POOTZ!
WHY SO SERIOUS?