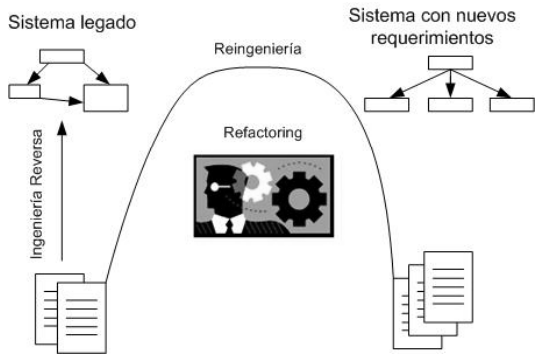
Aspect Refactoring

Desarrollo de Software Orientado a Aspectos

Santiago Vidal **ISISTAN-CONICET**

Introducción



Agenda

1. **Introducción**
2. **Refactoring**
3. **Aspect-Refactoring**
4. **Cátalogo**
5. **Herramientas**

Refactoring (1)

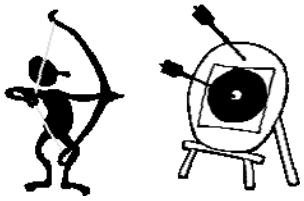
- "Es el proceso mediante el cual se hacen cambios en un sistema de software de forma tal que se mejore la estructura interna del mismo sin alterar el comportamiento externo"

  Martin Fowler

Refactoring (2)

Objetivos

- ❑Legibilidad
- ❑Diseño
- ❑Mantenimiento



Refactoring (3)

Actividades

1. Identificar los lugares del sistema que podrían ser refactorizados
2. Determinar que refactorings serán aplicados a cada uno de los lugares identificados
3. Verificar que la aplicación de estos refactorings preservaran el comportamiento
4. Aplicar los refactorings

Refactoring (4)

Identificación (Bad Smells)

- ❑Métodos demasiado largos
- ❑Clases con muchas responsabilidades
- ❑Cantidad de parámetros excesivos
- ❑Código duplicado

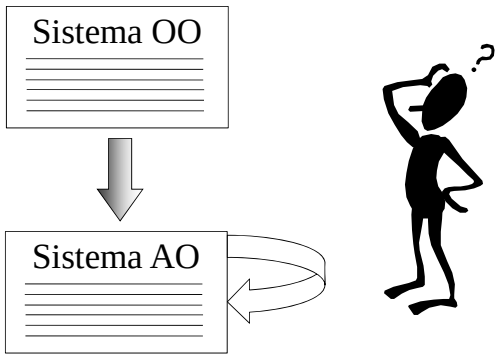
Refactoring (5)

Se tiene un fragmento de código que puede ser agrupado. Cambiar el fragmento por un método con un nombre representativo

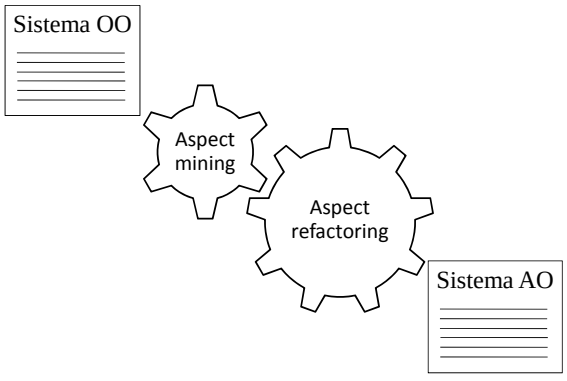
```
void printOwing() {
printBanner();
//print details
System.out.println ("name:" + _name);
System.out.println ("amount" + getOutstanding());
}
```

```
void printOwing() {
printBanner();
printDetails(getOutstanding());
}
void printDetails (double outstanding) {
System.out.println ("name:" + _name);
System.out.println ("amount" +
outstanding);
}
```

Aspect-Refactoring (1)



Aspect-Refactoring (2)



Aspect-Refactoring (3)

Utilización de Refactoring en sistemas AO

```
aspect AspectSample {
    before(): call(* Sample.pm()) {
        System.out.println("pm ok");
    }
}

class Sample {
    public static void main(String args[]) {
        new Sample().pm();
    }
    void pm() {
        System.out.println("print method");
    }
}
```

```
aspect AspectSample {
    before(): call(* Sample.pm()) {
        System.out.println("pm ok");
    }
}

class Sample {
    public static void main(String args[]) {
        new Sample().print_method();
    }
    void print_method() {
        System.out.println("print method");
    }
}
```


Aspect-Refactoring (4)

El refactoring orientado a aspectos combina AOP y refactoring para refactorizar crosscutting concerns

- Migración de sistemas OO a AO

Permite mejoras en el código orientado a aspectos

- Evolución de sistemas AO

Aspect-Refactoring - Tipos

Técnica/Enfoque	Objetivo	Motivación	Ejemplos
Refactorings Aspect-Aware OO	Construcciones OO	Asegurar que los refactorings OO actualicen correctamente las referencias a las construcciones AOP.	Rename/Inline Method
Refactorings de construcciones AOP	Construcciones OO y AO	Proveer nuevos refactorings que incluyan construcciones AOP.	Replace Member con Intertype Declaration, Pull up Pointcut, etc.
Refactorings de CCC	Implementaciones CCC	Proveer refactorings para reemplazar las implementaciones no modularizadas de CCC con aspectos.	Reemplazar cualquier CCC que no se encuentre modularizado con un aspecto.

Catalogo (2)

Monteiro (*) define un conjunto de 28 refactorings que abordan:

- Extracción de crosscutting concerns
- Estructura interna de los aspectos
- Generalizaciones de aspectos
- Código legado

(*) Monteiro, M. P., Catalogue of Refactorings for AspectJ, Technical Report UM-DI-GECSD-200401, Departmento de Informática, Universidade do Minho, Aug/2004.

Catalogo (1)

- Nombre del refactoring
- Situaciones típicas
- Acciones recomendadas
- Motivación
- Mecanismos de reestructuración
- Ejemplos de código

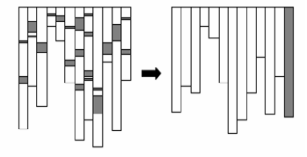
Catalogo (3)

Problema que aborda	Refactorings propuestos
Extracción de crosscutting concerns	Change Abstract Class to Interface, Encapsulate Implements with Declare Parents, Extract Feature into Aspect, Extract Fragment into Advice, Extract Inner Class to Standalone, Inline Class within Aspect, Inline Interface within Aspect, Move Field from Class to Inter-type, Move Method from Class to Inter-type, Split Abstract Class between Aspect and Interface.
Estructura interna de los aspectos	Extend Marker Interface with Signature, Generalize Target Type with Marker Interface, Introduce Aspect Protection, Replace Inter-type Field with Aspect Map, Replace Inter-type Method with Aspect Method, Tidy Up Internal Aspect Structure
Generalizaciones de aspectos	Extract Superaspect, Pull Up Advice, Pull Up Declare Parents, Pull Up Inter-type Declaration, Pull Up Marker Interface, Pull Up Pointcut, Push Down Advice, Push Down Declare Parents, Push Down Inter-type Declaration, Push Down Marker Interface, Push Down Pointcut
Código legado	Partition Constructor Signature

Catálogo - Extract Feature into Aspect (1)

Situación típica: el código presenta un CCC que esta diseminado en diferentes módulos (scattered).

Acción recomendada: modularizar el CCC extrayendo todo el código relacionado en un aspecto.



Catálogo - Extract Feature into Aspect (2)

Mecanismo:

- Crear un nuevo aspecto
- Mover los campos relacionados al concern al nuevo aspecto utilizando *Move Field from Class to Inter-type*
- Mover el código de inicialización del método constructor con *Extract Fragment into Advice.* (Se puede utilizar *Partition Constructor Signature en aquellos casos que el constructor es parte de una interfaz que no puede modificarse*)
- Mover los métodos relacionados al concern al aspecto con *Move Method from Class to Inter-type*
- Cambiar a privado los modos de acceso a los métodos del aspecto que se utilizan solo dentro de este

Catálogo - Extract Feature into Aspect (3)

- Estructura FIFO
- CCC de una ventana con el estado de la pila
- CCC de chequeo de precondición

```
import javax.swing.*;
public class TangledStack {
        private int _top = -1;
        private Object[] _elements;
        private final int S_SIZE = 10;
        private JLabel _label = new JLabel("Stack ");
        private JTextField _text = new JTextField(20);

        public TangledStack(JFrame frame) {
                _elements = new Object[S_SIZE];
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
```


Catálogo - Extract Feature into Aspect (4)

```
public String toString() {
      StringBuffer result = new StringBuffer("[");
      for(int i=0;i<=_top;i++) {
              result.append(_elements[i].toString());
              if(i!=_top)
                      result.append(", ");
       }
      result.append("]");
      return result.toString();
}
private void display() {
      _text.setText(toString());
}
public void push(Object element) {
      if(isFull())
      throw new PreConditionException("push when stack full.");
      _elements[++_top] = element;
      display();
}
```

Catálogo - Extract Feature into Aspect (5)

```
   public void pop() {
        if(isEmpty())
        throw new PreConditionException("pop when stack empty.");
        _top--;
        display();
   }
   public Object top() {
        if(isEmpty())
        throw new PreConditionException("top when stck empty.");
        return _elements[_top];
   }
   public boolean isFull() {
        return (_top == S_SIZE-1);
   }
   public boolean isEmpty() {
        return (_top<0);
   }
}
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Se mueven todos los miembros relacionados al concern.
Se utiliza el refactoring *Move Field from Class to Inter-type*

```
   public aspect WindowView {




   }
```

```
import javax.swing.*;
public class TangledStack {
                private int _top = -1;
                private Object[] _elements;
                private final int S_SIZE = 10;
                private JLabel _label = new JLabel("Stack ");
                private JTextField _text = new JTextField(20);
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Creamos un nuevo aspecto *WindowView* .

```
   public aspect WindowView {




   }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Se copia la declaración al aspecto.

```
   public aspect WindowView {
            private JLabel _label = new JLabel("Stack ");
            private JTextField _text = new JTextField(20);




   }
```

```
import javax.swing.*;
public class TangledStack {
                private int _top = -1;
                private Object[] _elements;
                private final int S_SIZE = 10;
                private JLabel _label = new JLabel("Stack ");
                private JTextField _text = new JTextField(20);
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Agregar 'TangledStack.' antes de las declaraciones.

```
public aspect WindowView {
        private Jlabel TangledStack._label = new JLabel("Stack ");
        private JTextField TangledStack._text = new JTextField(20);


}
```

```
import javax.swing.*;
public class TangledStack {
                private int _top = -1;
                private Object[] _elements;
                private final int S_SIZE = 10;
                private JLabel _label = new JLabel("Stack ");
                private JTextField _text = new JTextField(20);
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Copiar la declaración 'import javax.swing.*;' .

```
import javax.swing.*;
public aspect WindowView {
        private Jlabel TangledStack._label = new JLabel("Stack ");
        private JTextField TangledStack._text = new JTextField(20);


}
```

```
import javax.swing.*;
public class TangledStack {
                private int _top = -1;
                private Object[] _elements;
                private final int S_SIZE = 10;
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Se eliminan las declaraciones originales.

```
public aspect WindowView {
        private Jlabel TangledStack._label = new JLabel("Stack ");
        private JTextField TangledStack._text = new JTextField(20);


}
```

```
import javax.swing.*;
public class TangledStack {
                private int _top = -1;
                private Object[] _elements;
                private final int S_SIZE = 10;
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern "window view".

Cambiar los accesos a *public*. Compilar y testear.

```
import javax.swing.*;
public aspect WindowView {
        public Jlabel TangledStack._label = new JLabel("Stack ");
        public JTextField TangledStack._text = new JTextField(20);


}
```

```
import javax.swing.*;
public class TangledStack {
                private int _top = -1;
                private Object[] _elements;
                private final int S_SIZE = 10;
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Transferir el código de inicialización.
Utilizar *Partition Constructor Signature* .

```
import javax.swing.*;
public class TangledStack {
        public TangledStack(JFrame frame) {
                _elements = new Object[S_SIZE];
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Mover al nuevo constructor todas las sentencias no relacionadas con el CCC.

```
import javax.swing.*;
public class TangledStack {
        public TangledStack(JFrame frame) {
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
        public TangledStack() {
                _elements = new Object[S_SIZE];
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Crear un nuevo constructor en la clase sin los argumentos relacionados al CCC.

```
import javax.swing.*;
public class TangledStack {
        public TangledStack(JFrame frame) {
                _elements = new Object[S_SIZE];
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
        public TangledStack() {

        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Agregar una llamada a *this()* como primer sentencia en el constructor original.

```
import javax.swing.*;
public class TangledStack {
        public TangledStack(JFrame frame) {
                this();
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
        public TangledStack() {
                _elements = new Object[S_SIZE];
        }
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Mover el constructor original modificado al aspecto.

```
public aspect WindowView {
        public TangledStack(JFrame frame) {
                this();
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
```

```
import javax.swing.*;
public class TangledStack {
        public TangledStack() {
                _elements = new Object[S_SIZE];
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Mover el método *display()* utilizando *Move Method from Class to Inter-type* .

```
import javax.swing.*;
public class TangledStack {
        private void display() {
                _text.setText(toString());
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Agregar *“.new”* después del nombre del constructor original en el aspecto. Compilar y testear.

```
public aspect WindowView {
        public TangledStack.new(JFrame frame) {
                this();
                frame.getContentPane().add(_label);
                _text.setText("[]");
                frame.getContentPane().add(_text);
        }
```

```
import javax.swing.*;
public class TangledStack {
        public TangledStack() {
                _elements = new Object[S_SIZE];
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Copiar la definición del método al aspecto. Agregar el nombre de la clase y “.” antes del nombre del método.

```
public aspect WindowView {
        private void TangledStack.display() {
                _text.setText(toString());
        }
```

```
import javax.swing.*;
public class TangledStack {
        private void display() {
                _text.setText(toString());
        }
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Cambiar temporalmente el acceso a *public*.

```
public aspect WindowView {
        public void TangledStack.display() {
                _text.setText(toString());
        }
```

```
import javax.swing.*;
public class TangledStack {
        private void display() {
                _text.setText(toString());
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Extraer las llamadas a *display()* utilizando *Extract Fragment into Advice.*

```
import javax.swing.*;
public class TangledStack {
        public void push(Object element) {
        if(isFull())
        throw new PreConditionException("push when stack full.");
        _elements[++_top] = element;
        display();
}
public void pop() {
        if(isEmpty())
        throw new PreConditionException("pop when stack empty.");
        _top--;
        display();
}
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Eliminar la definición del método en la clase.
Compilar y testear.

```
public aspect WindowView {
        public void TangledStack.display() {
                _text.setText(toString());
        }
```

```
import javax.swing.*;
public class TangledStack {
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Crear un pointcut que capture el conjunto de jointpoitns.

```
public aspect WindowView {
        pointcut stateChange():
        execution(public void TangledStack.push(Object));
```

```
import javax.swing.*;
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
                display();
        }
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Chequear que se capture el contexto requerido por el fragmento de código (ej: this, super, etc.).

```
public aspect WindowView {
         pointcut stateChange(TangledStack stack):
         execution(public void TangledStack.push(Object))&&
         this(stack);
```

```
import javax.swing.*;
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
                display();
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Mover el código a extraer de la clase al advice.

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        execution(public void TangledStack.push(Object))&&
        this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                display();
        }
```

```
import javax.swing.*;
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;

        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Crear un advice para el pointcut.

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        execution(public void TangledStack.push(Object))&&
        this(stack);
        after(TangledStack _this) returning :stateChange(_this) {

        }
```

```
import javax.swing.*;
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
                display();
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Agregar las variables correspondientes al contexto.

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        execution(public void TangledStack.push(Object))&&
        this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                _this.display();
        }
```

```
import javax.swing.*;
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;

        }
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Crear un pointcut que capture el conjunto de jointpoitns. (Si ya existe modificarlo)

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        (execution(public void stack.TangledStack.push(Object))||
        execution(public void stack.TangledStack.pop()))&& this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                _this.display();
        }
```

```
import javax.swing.*;
public class TangledStack {
         public void pop() {
                if(isEmpty())
                throw new PreConditionException("pop when stack empty.");
                _top--;
                display();
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Crear un advice para el pointcut.

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        (execution(public void stack.TangledStack.push(Object))||
        execution(public void stack.TangledStack.pop()))&& this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                _this.display();
        }
```

```
import javax.swing.*;
public class TangledStack {
         public void pop() {
                if(isEmpty())
                throw new PreConditionException("pop when stack empty.");
                _top--;
                display();
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Chequear que se capture el contexto requerido por el fragmento de código (ej: this, super, etc.).

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        (execution(public void stack.TangledStack.push(Object))||
        execution(public void stack.TangledStack.pop()))&& this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                _this.display();
        }
```

```
import javax.swing.*;
public class TangledStack {
         public void pop() {
                if(isEmpty())
                throw new PreConditionException("pop when stack empty.");
                _top--;
                display();
        }
```


Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Mover el código a extraer de la clase al advice.

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        (execution(public void stack.TangledStack.push(Object))||
        execution(public void stack.TangledStack.pop()))&& this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                _this.display();
        }
```

```
import javax.swing.*;
public class TangledStack {
         public void pop() {
                if(isEmpty())
                throw new PreConditionException("pop when stack empty.");
                _top--;
        }
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern “window view”.

Agregar las variables correspondientes al contexto.

```
public aspect WindowView {
        pointcut stateChange(TangledStack stack):
        (execution(public void stack.TangledStack.push(Object))||
        execution(public void stack.TangledStack.pop()))&& this(stack);
        after(TangledStack _this) returning :stateChange(_this) {
                _this.display();
        }
```

```
import javax.swing.*;
public class TangledStack {
        public void pop() {
                if(isEmpty())
                throw new PreConditionException("pop when stack empty.");
                _top--;
        }
```


Catálogo - Extract Feature into Aspect (6)

Eliminar la declaración ‘import javax.swing.*;’ .

```
public class TangledStack {
    private int _top = -1;
    private Object[] _elements;
    private final int S_SIZE = 10;
    public TangledStack() {
        _elements = new Object[S_SIZE];
    }
    public void push(Object element) {
        if(isFull())
        throw new PreConditionException
         ("push when stack full.");
        _elements[++_top] = element;
    }
    public void pop() {
        if(isEmpty())
        throw new PreConditionException
        ("pop when stack empty.");
        _top--;
    }
    …
}
```

```
import javax.swing.*;
public aspect WindowView {
    public Jlabel TangledStack._label=new JLabel("Stack ");
    public JTextField TangledStack._text
     = new JTextField(20);
    public TangledStack.new(JFrame frame) {
        this();
        frame.getContentPane().add(_label);
        _text.setText("[]");
       frame.getContentPane().add(_text);
    }
    public void TangledStack.display() {
        _text.setText(toString());
    }
    pointcut stateChange(TangledStack stack):
    (execution(public void stack.TangledStack.push(Object))||
    execution(public void stack.TangledStack.pop()))&&
    this(stack);
    after(TangledStack _this) returning :stateChange(_this) {
        _this.display();
    }
}
```


Catálogo - Extract Feature into Aspect (6)

```
import javax.swing.*;
public class TangledStack {
    private int _top = -1;
    private Object[] _elements;
    private final int S_SIZE = 10;
    public TangledStack() {
        _elements = new Object[S_SIZE];
    }
    public void push(Object element) {
        if(isFull())
        throw new PreConditionException
         ("push when stack full.");
        _elements[++_top] = element;
    }
    public void pop() {
        if(isEmpty())
        throw new PreConditionException
        ("pop when stack empty.");
        _top--;
    }
    …
}
```

```
import javax.swing.*;
public aspect WindowView {
    public Jlabel TangledStack._label=new JLabel("Stack ");
    public JTextField TangledStack._text
     = new JTextField(20);
    public TangledStack.new(JFrame frame) {
        this();
        frame.getContentPane().add(_label);
        _text.setText("[]");
       frame.getContentPane().add(_text);
    }
    public void TangledStack.display() {
        _text.setText(toString());
    }
    pointcut stateChange(TangledStack stack):
    (execution(public void stack.TangledStack.push(Object))||
    execution(public void stack.TangledStack.pop()))&&
    this(stack);
    after(TangledStack _this) returning :stateChange(_this) {
        _this.display();
    }
}
```


Catálogo - Extract Feature into Aspect (6)

Cambiar a privado los accesos a los métodos y fields.

```
public class TangledStack {
    private int _top = -1;
    private Object[] _elements;
    private final int S_SIZE = 10;
    public TangledStack() {
        _elements = new Object[S_SIZE];
    }
    public void push(Object element) {
        if(isFull())
        throw new PreConditionException
         ("push when stack full.");
        _elements[++_top] = element;
    }
    public void pop() {
        if(isEmpty())
        throw new PreConditionException
        ("pop when stack empty.");
        _top--;
    }
    …
}
```

```
import javax.swing.*;
public aspect WindowView {
    private Jlabel TangledStack._label=new JLabel("Stack ");
    private JTextField TangledStack._text
     = new JTextField(20);
    public TangledStack.new(JFrame frame) {
        this();
        frame.getContentPane().add(_label);
        _text.setText("[]");
       frame.getContentPane().add(_text);
    }
    private void TangledStack.display() {
        _text.setText(toString());
    }
    pointcut stateChange(TangledStack stack):
    (execution(public void stack.TangledStack.push(Object))||
    execution(public void stack.TangledStack.pop()))&&
    this(stack);
    after(TangledStack _this) returning :stateChange(_this) {
        _this.display();
    }
}
```

Catálogo - Extract Feature into Aspect (6)

Extracción del concern "PreConditionChecking".

Extraer las declaraciones utilizando *Extract Fragment into Advice* .

```
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
        }
        public void pop() {
                if(isEmpty())
                throw new PreConditionException("pop when stack empty.")
                _top--;
        }
        public Object top() {
                if(isEmpty())
                throw new PreConditionException("top when stck empty.");
                return _elements[_top];
        }
}
```


Catálogo - Extract Feature into Aspect (7)

Extracción del concern "PreConditionChecking".

Crear un pointcut que capture el conjunto de jointpoitns.

```
public aspect PreConditionChecking{
        pointcut checkPush():
        execution(public void TangledStack.push(Object));




}
```

```
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
        }
}
```


Catálogo - Extract Feature into Aspect (7)

Extracción del concern "PreConditionChecking".

Creamos un nuevo aspecto *PreConditionChecking.*

```
public aspect PreConditionChecking{




}
```

```
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
        }
}
```


Catálogo - Extract Feature into Aspect (7)

Extracción del concern "PreConditionChecking".

Chequear que se capture el contexto requerido por el fragmento de código (ej: this, super, etc.).

```
public aspect PreConditionChecking{
        pointcut checkPush(TangledStack stack):
        execution(public void TangledStack.push(Object))&& this(stack);




}
```

```
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
        }
}
```

Catálogo - Extract Feature into Aspect (7)

Extracción del concern "PreConditionChecking".

Crear un advice para el pointcut.

```
public aspect PreConditionChecking{
        pointcut checkPush(TangledStack stack):
        execution(public void TangledStack.push(Object))&& this(stack);
        before(TangledStack _this): checkPush(_this) {


        }


}
```

```
public class TangledStack {
        public void push(Object element) {
                if(isFull())
                throw new PreConditionException("push when stack full.");
                _elements[++_top] = element;
        }
}
```


Catálogo - Extract Feature into Aspect (7)

Extracción del concern "PreConditionChecking".

Agregar las variables correspondientes al contexto.

```
public aspect PreConditionChecking{
        pointcut checkPush(TangledStack stack):
        execution(public void TangledStack.push(Object))&& this(stack);
        before(TangledStack _this): checkPush(_this) {
                if(_this.isFull())
                throw new PreConditionException("push when stack full");
        }


}
```

```
public class TangledStack {
        public void push(Object element) {
                _elements[++_top] = element;
        }
}
```


Catálogo - Extract Feature into Aspect (7)

Extracción del concern "PreConditionChecking".

Mover el código a extraer de la clase al advice.

```
public aspect PreConditionChecking{
        pointcut checkPush(TangledStack stack):
        execution(public void TangledStack.push(Object))&& this(stack);
        before(TangledStack _this): checkPush(_this) {
                if(isFull())
                throw new PreConditionException("push when stack full");
        }


}
```

```
public class TangledStack {
        public void push(Object element) {
                _elements[++_top] = element;
        }
}
```

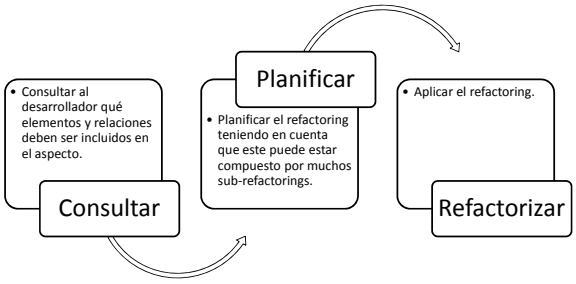

Catálogo - Extract Feature into Aspect (7)

```
public aspect PreConditionChecking {
        pointcut checkPush(TangledStack stack):
        execution(public void TangledStack.push(Object)) && this(stack);

        before(TangledStack _this): checkPush(_this) {
                if(_this.isFull())
                throw new PreConditionException("push when stack full");
        }

        pointcut checkPop(TangledStack stack):
        execution(public void TangledStack.pop()) && this(stack);

        before(TangledStack _this): checkPop(_this) {
                if(_this.isEmpty())
                throw new PreConditionException("pop when stack empty");
        }

        pointcut checkTop(TangledStack stack):
        execution(public Object TangledStack.top()) && this(stack);

        before(TangledStack _this): checkTop(_this) {
                if(_this.isEmpty())
                throw new PreConditionException("top when stack empty");
        }
}
```

Catálogo - Extract Feature into Aspect (7)

```
public class TangledStack {
        private int _top = -1;
        private Object[] _elements;
        private final int S_SIZE = 3;
        public TangledStack() {
                _elements = new Object[S_SIZE];
        }
        public String toString() {
                ...
        }
        public void push(Object element) {
                _elements[++_top] = element;
        }
        public void pop() {
                _top--;
        }
        public Object top() {
                return _elements[_top];
        }
        public boolean isFull() {
                return (_top == S_SIZE-1);
        }
        public boolean isEmpty() {
                return (_top<0);
        }
}
```


Herramientas (2)



Herramientas (1)

Herramientas interactivas (*)

- Semiautomáticas.
- Buscan capturar las descripciones de crosscutting concerns.
- Decisiones a cargo del desarrollador.

(*) J. Hannemann, T. Fritz, and G. C. Murphy. Refactoring to aspects: an interactive approach. In Proc. of 2003 OOPSLA Workshop on Eclipse technology eXchange, pages 74–78. ACM Press, 2003.

Herramientas (3)

Herramientas iterativas (*)

- Semiautomáticas.
- Suponen un pre-proceso de aspect-mining.
- Deben identificarse y señalarse los crosscutting concerns “aspectizables”.

(*) D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella. Automated refactoring of object oriented code into aspects. In 21st IEEE International Conference on Software Maintenance (ICSM), 2005.

Herramientas (4)

Basadas en Crosscutting concern sorts (*)

- Semiautomáticas
- Suponen un pre-proceso de aspect-mining basados en concern sorts
- Es conducido por los concerns sorts

(*) Marin, M.; Deursen, A.; Moonen, L. & Rijst, R. An integrated crosscutting concern migration strategy and its semi-automated application to JHotDraw Automated Software Engg., Kluwer Academic Publishers, 2009, 16, 323-356