

Charla de Tesis 1

Magíster en TI

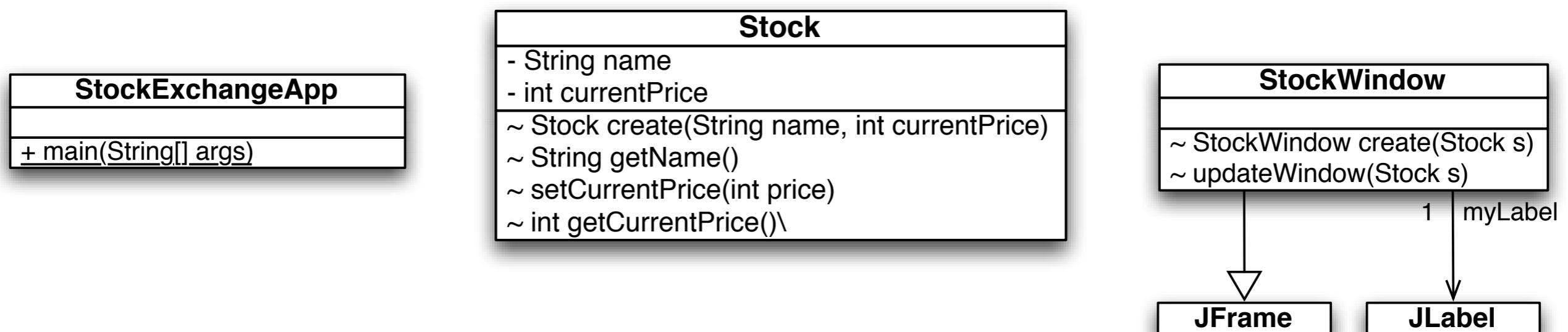
Marel Josué Oliva - moliva@dcc.uchile.cl
(material seleccionado de Joerg Kienzle)



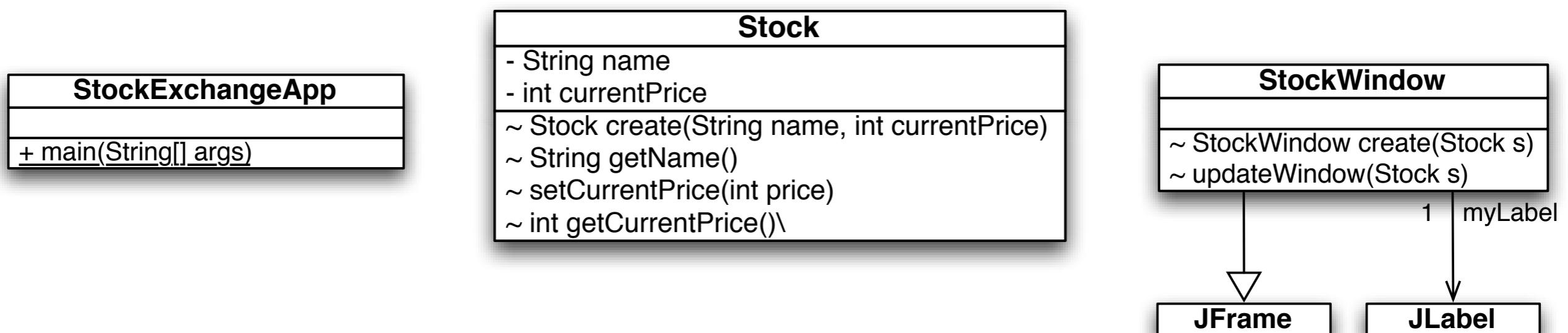
Stock Exchange Application (1)

- Un servidor con la información de las acciones (nombre y precio)
- Varios GUI
- Acceso remoto

Stock Exchange Application (2)

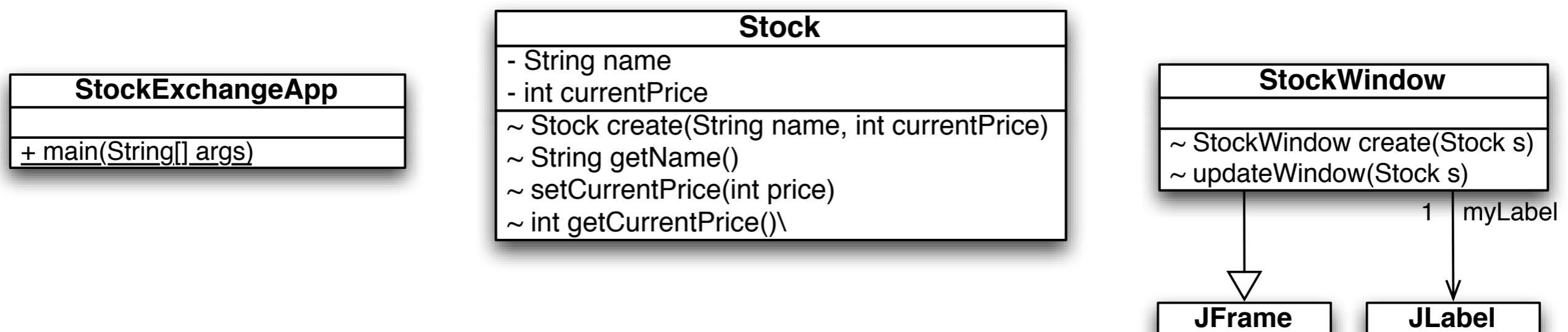


Stock Exchange Application (2)



¿Cómo desacoplar Stock de StockApplication?

Stock Exchange Application (2)



¿Cómo desacoplar Stock de StockApplication?

Patrón de diseño
Observer

Consecuencias del patrón Observer



Consecuencias del patrón Observer

- Observer es crosscutting

Consecuencias del patrón Observer

- Observer es crosscutting
- Implementaciones de aspectos:
 - AspectJ, PHANTom, etc.

Consecuencias del patrón Observer

- Observer es crosscutting
- Implementaciones de aspectos:
 - AspectJ, PHANTom, etc.
- ¿Cómo analizar/diseñar preocupaciones transversales?

Consecuencias del patrón Observer

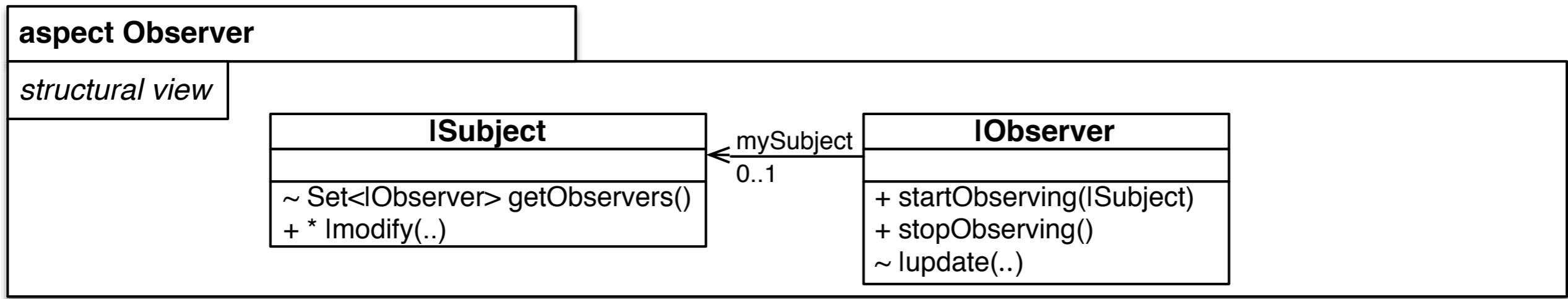
- Observer es crosscutting
- Implementaciones de aspectos:
 - AspectJ, PHANTom, etc.
- ¿Cómo analizar/diseñar preocupaciones transversales?

Modelamiento orientado a aspectos (AOM)

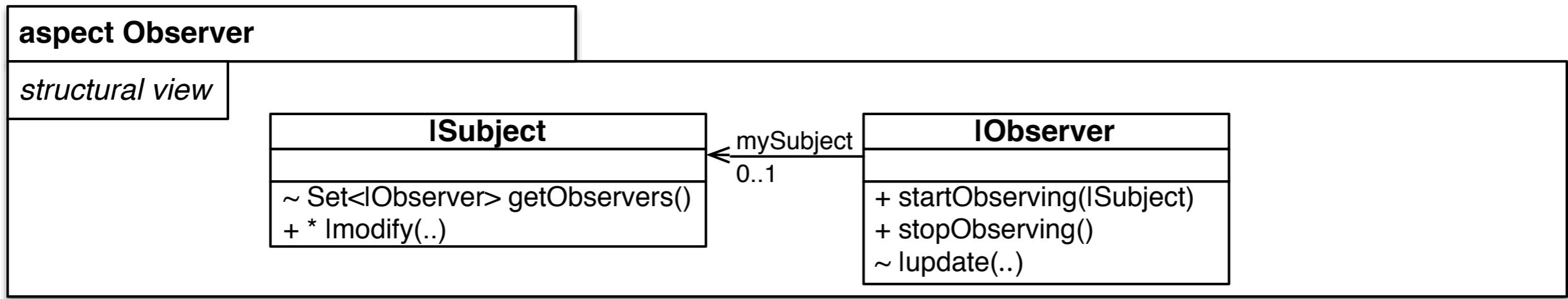
Reusable Aspect Models (RAM)

- Diseño de modelos de software reutilizables, gracias a AOM
- Preocupación = aspecto
- "Aspect Model" => paquete UML => estructura y comportamiento de una preocupación
- Resuelve problemas de escalabilidad y consistencia de AOM

Vista estructural (Observer)

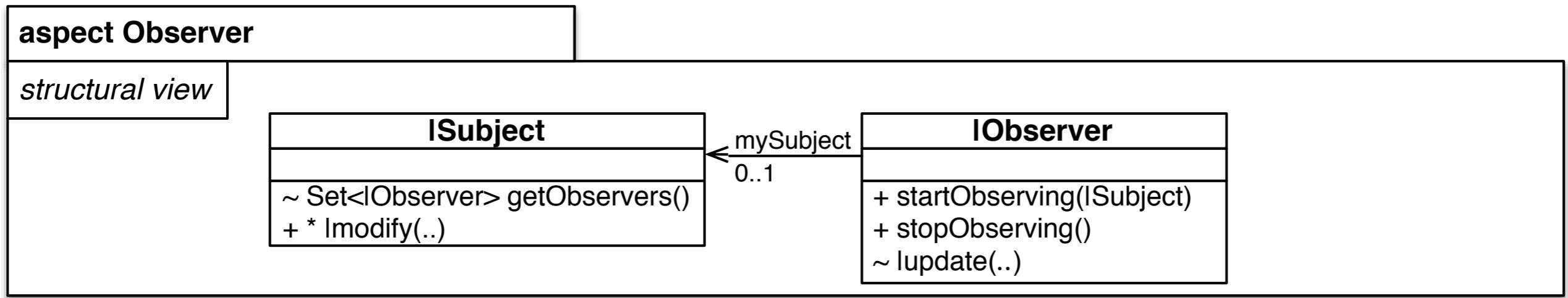


Vista estructural (Observer)



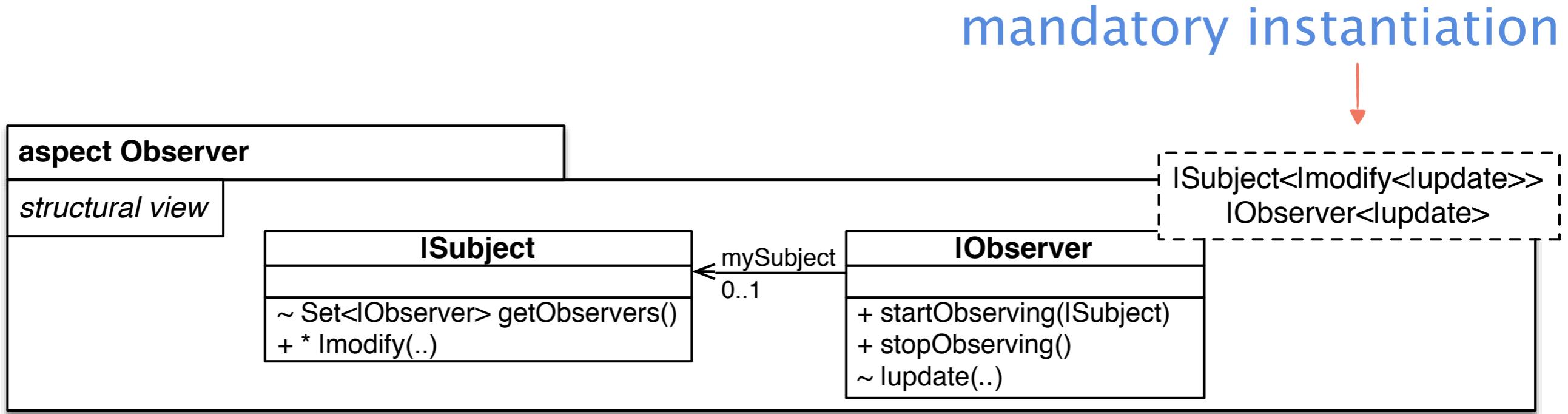
1. Conceptos relevantes al usuario (clases, operaciones)

Vista estructural (Observer)



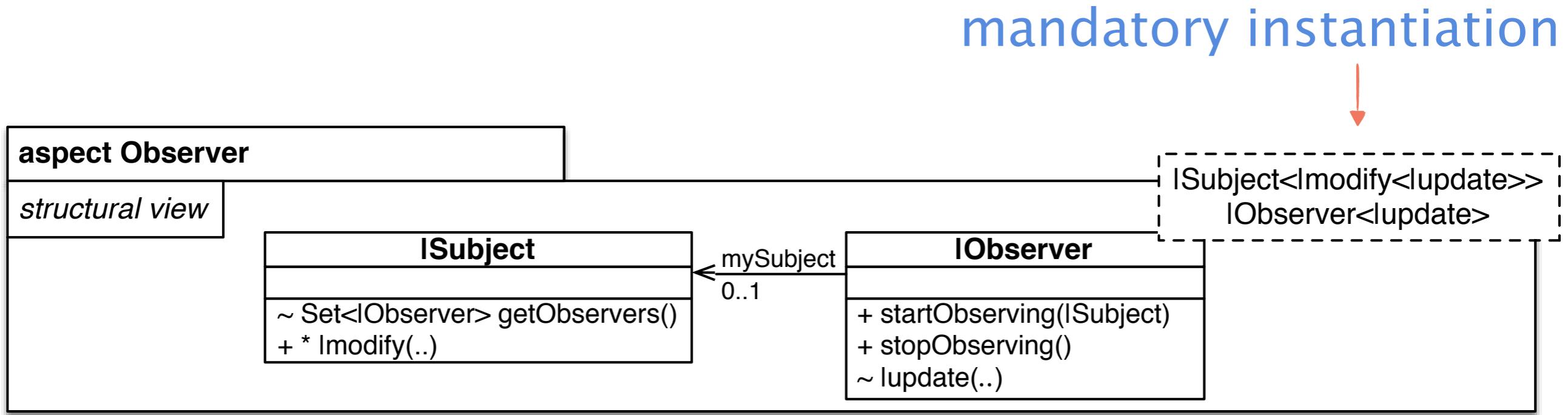
2. (+) público
(~) priv. aspecto
(-) privado

Vista estructural (Observer)



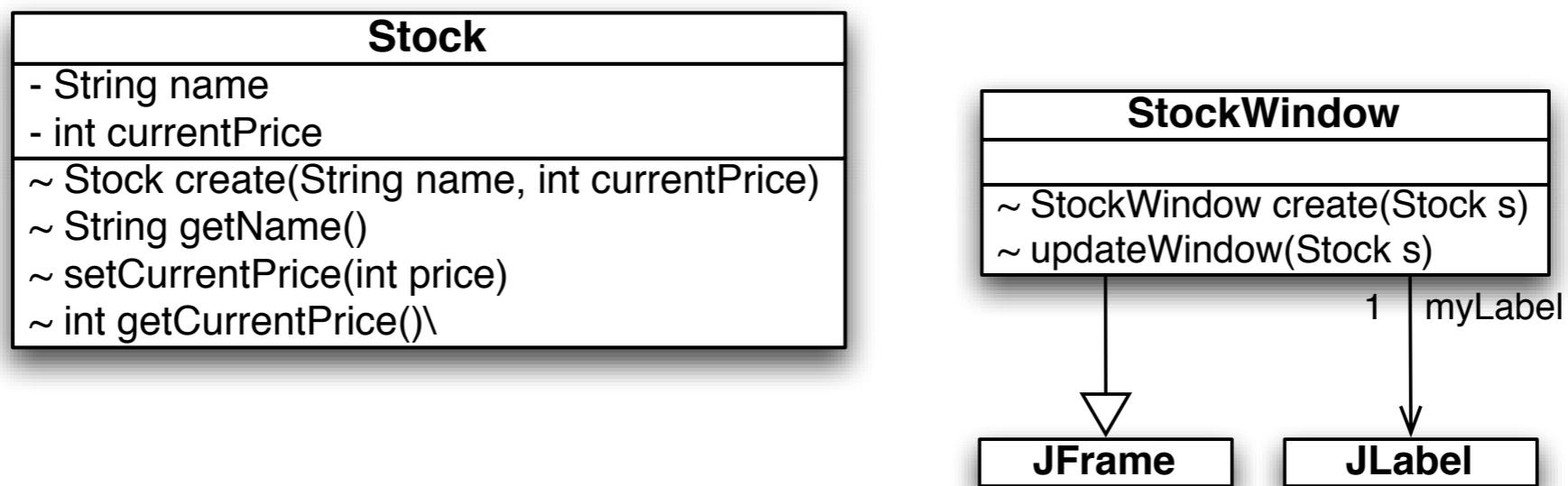
3. | -Clases y operaciones incompletas

Vista estructural (Observer)



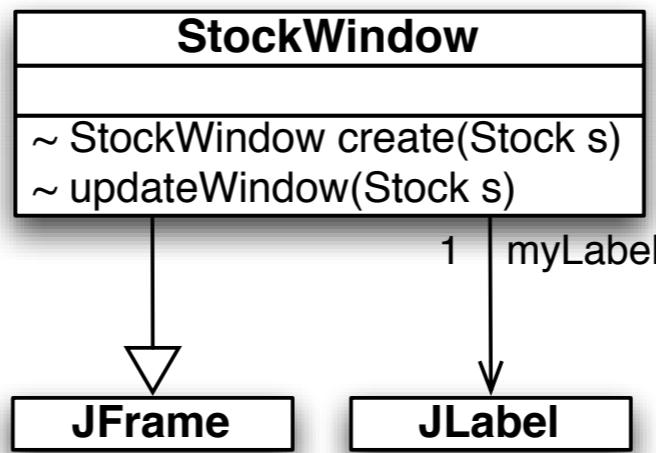
3. | -Clases y operaciones incompletas

Reutilización por instanciación

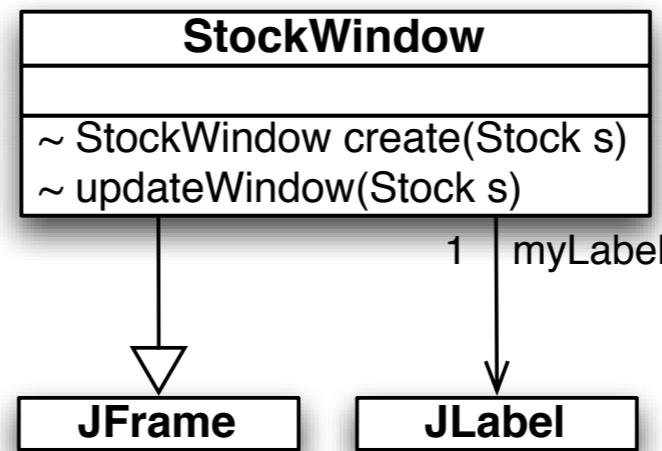
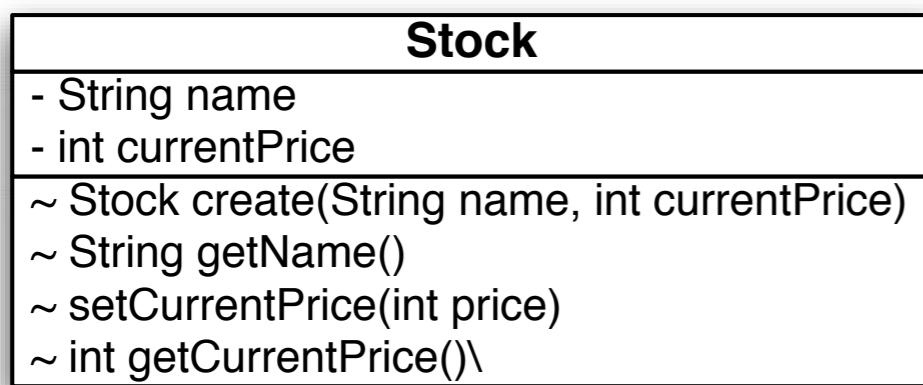
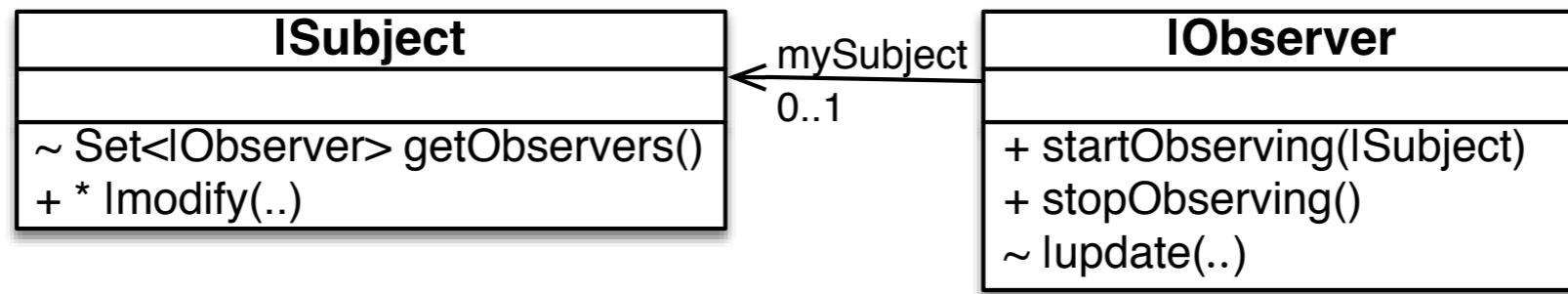


Reutilización por instanciación

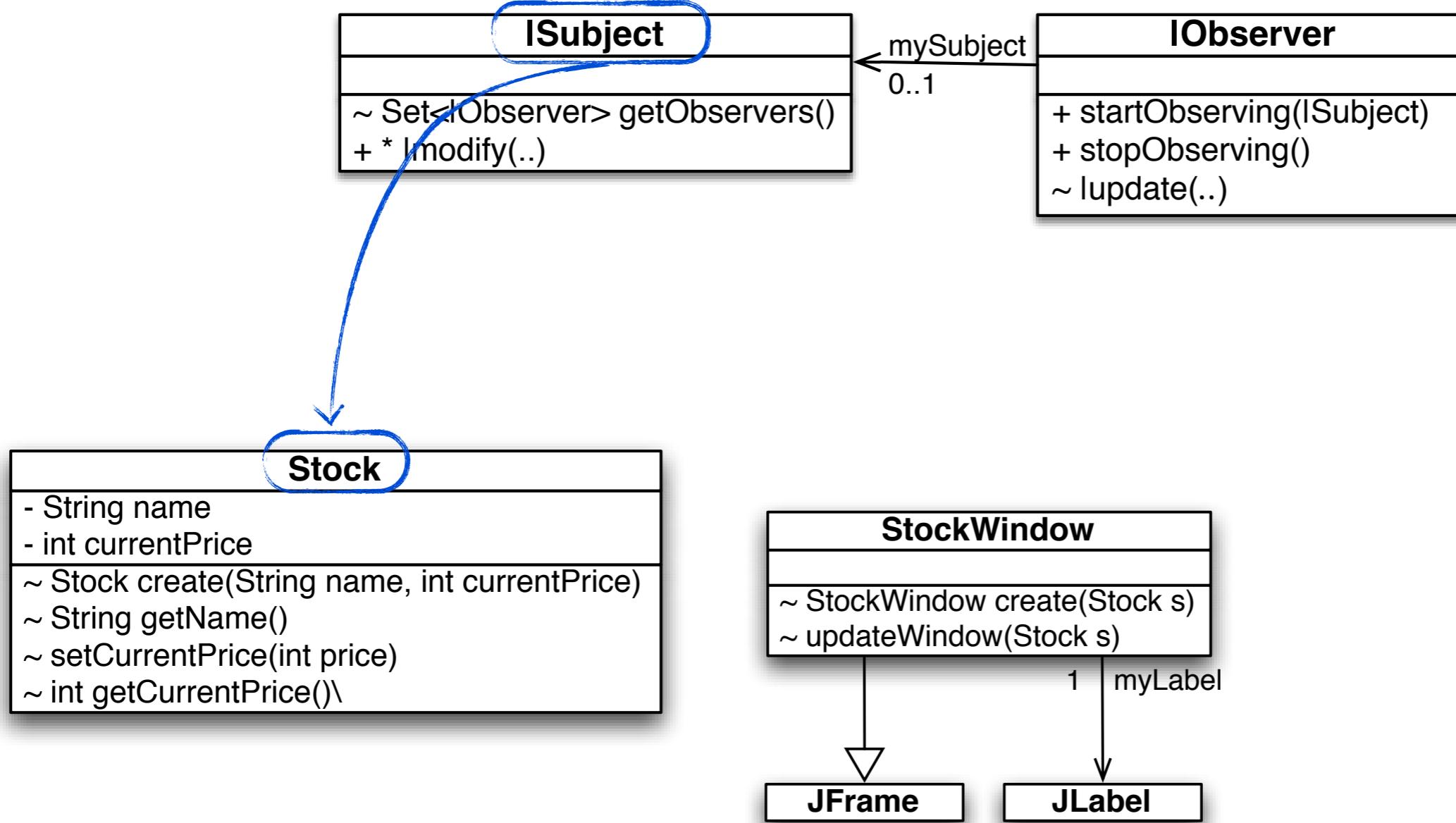
Stock
- String name
- int currentPrice
~ Stock create(String name, int currentPrice)
~ String getName()
~ setCurrentPrice(int price)
~ int getCurrentPrice()



Reutilización por instanciación

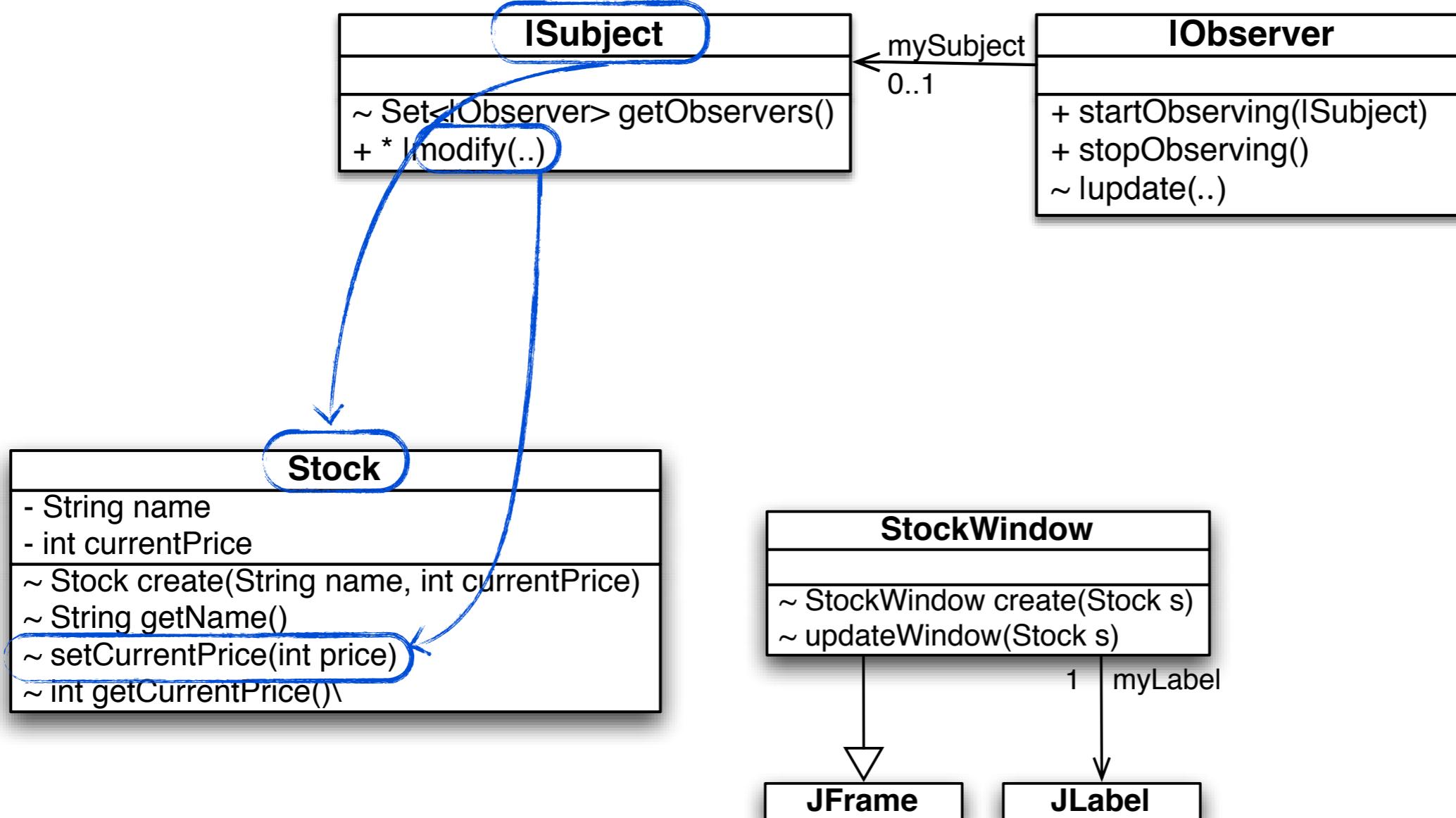


Reutilización por instanciación



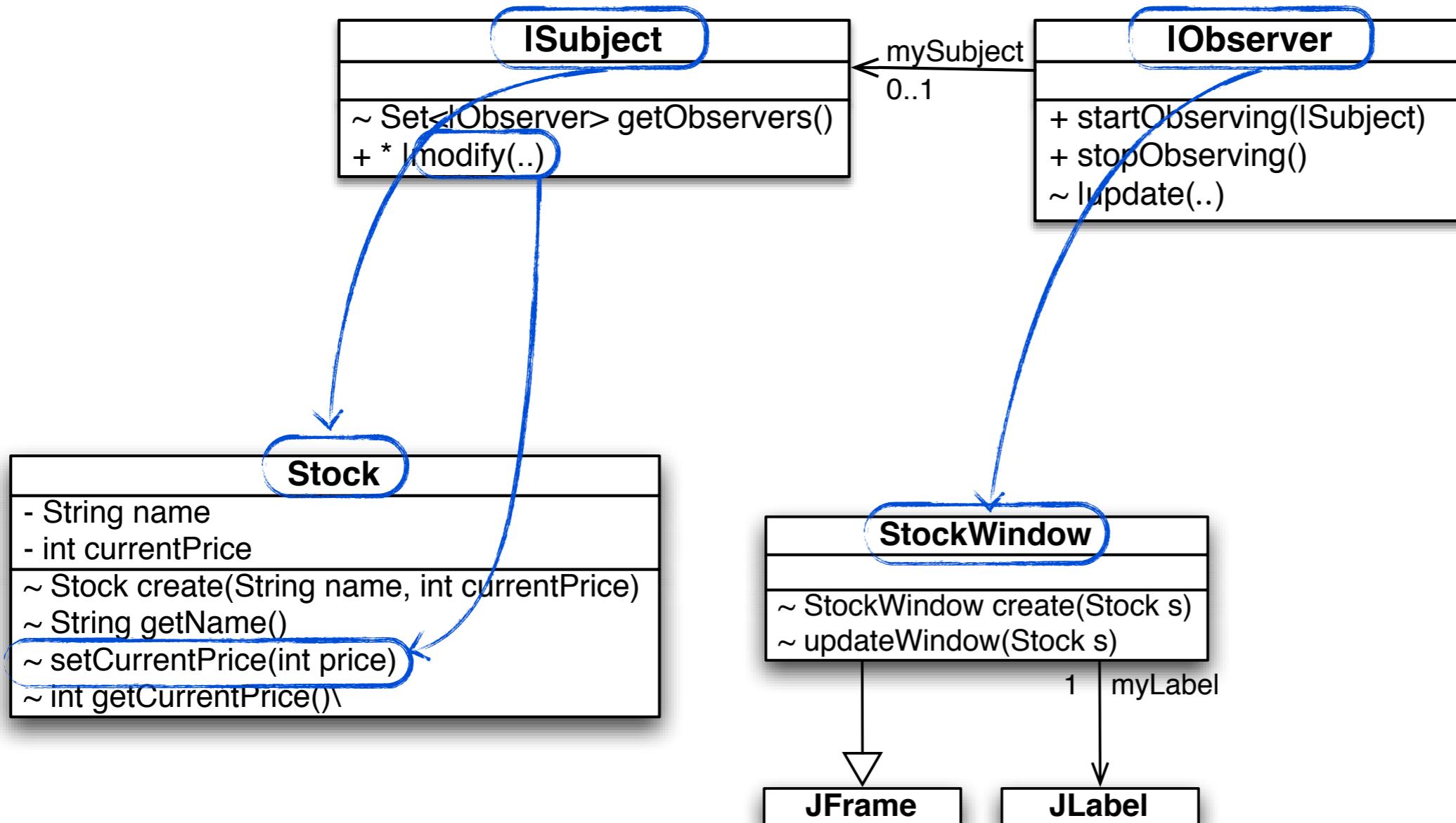
Stock -> |Subject

Reutilización por instanciación



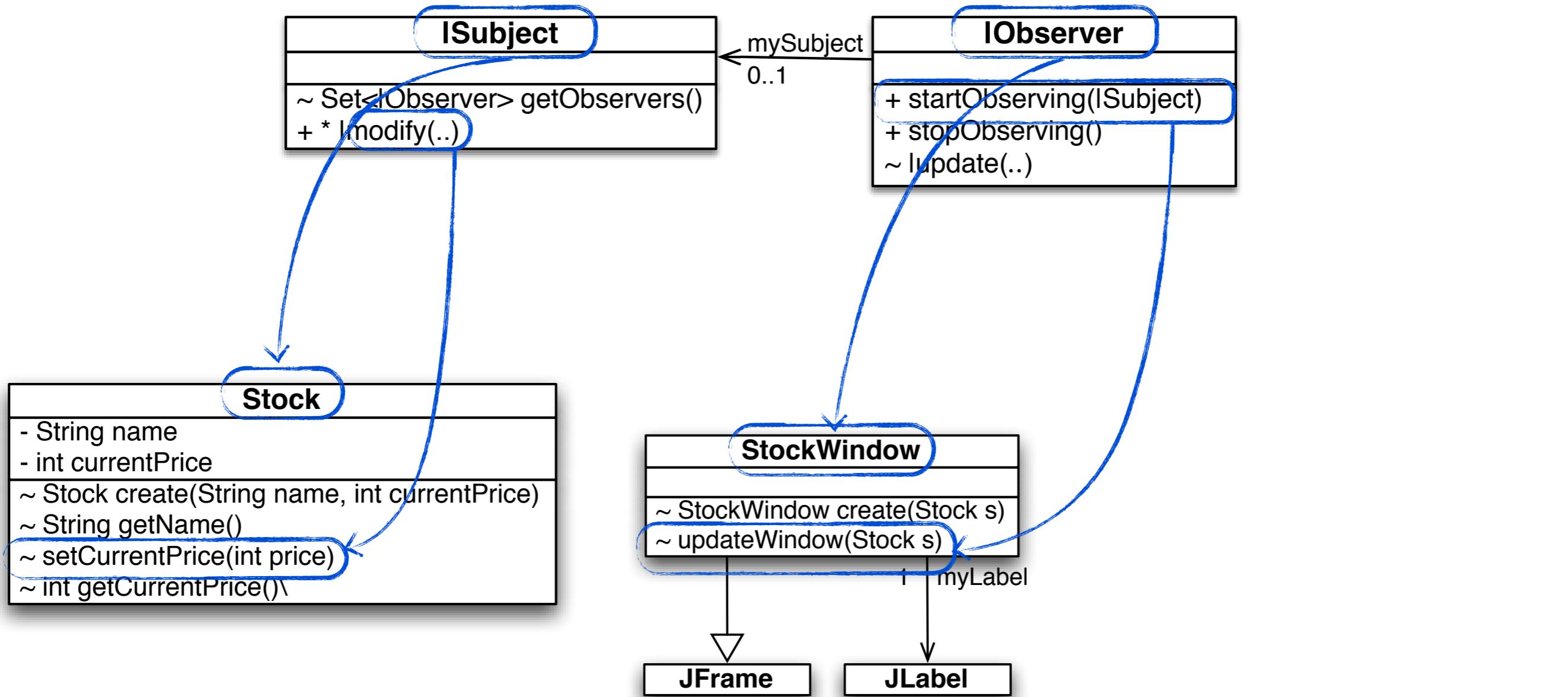
setCurrentPrice -> |modify

Reutilización por instanciación



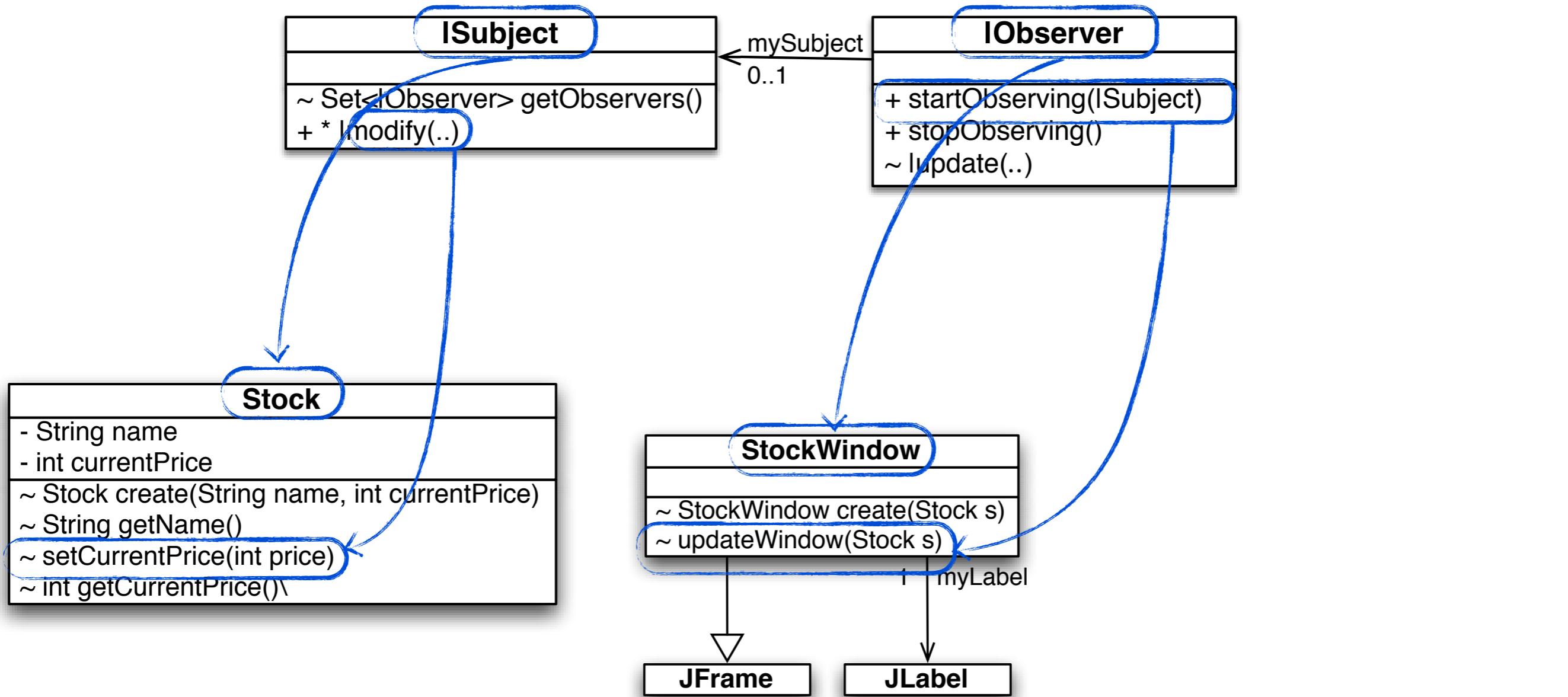
StockWindow -> |Observe

Reutilización por instanciación



updateWindow -> |startObserving

Reutilización por instanciación

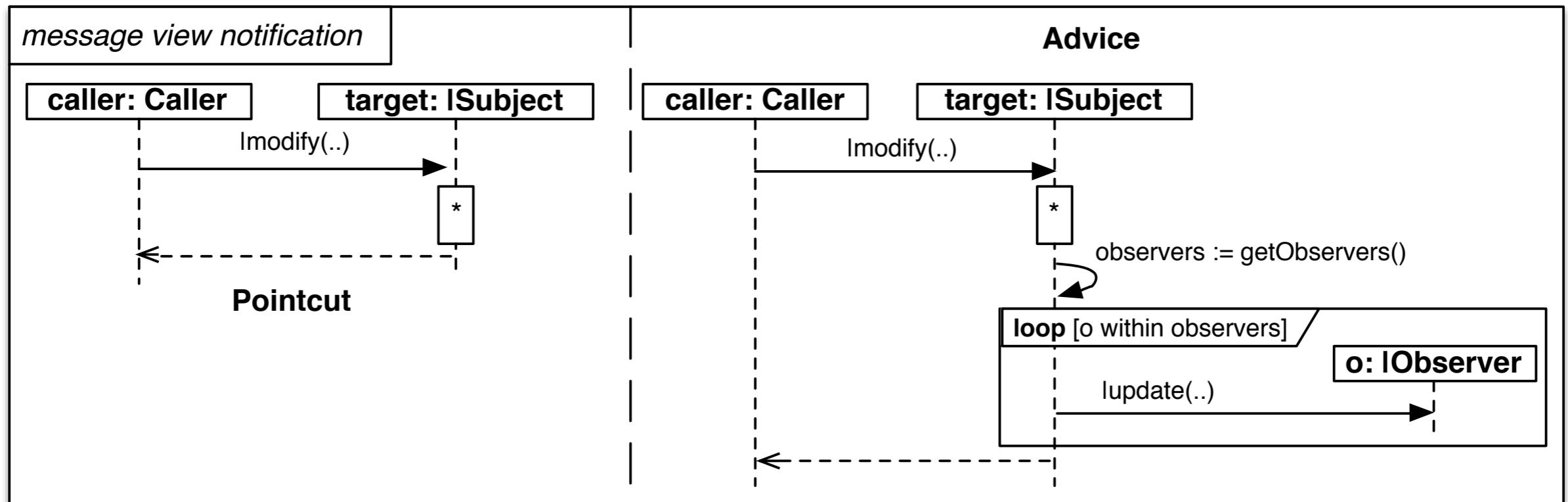


updateWindow -> |startObserving

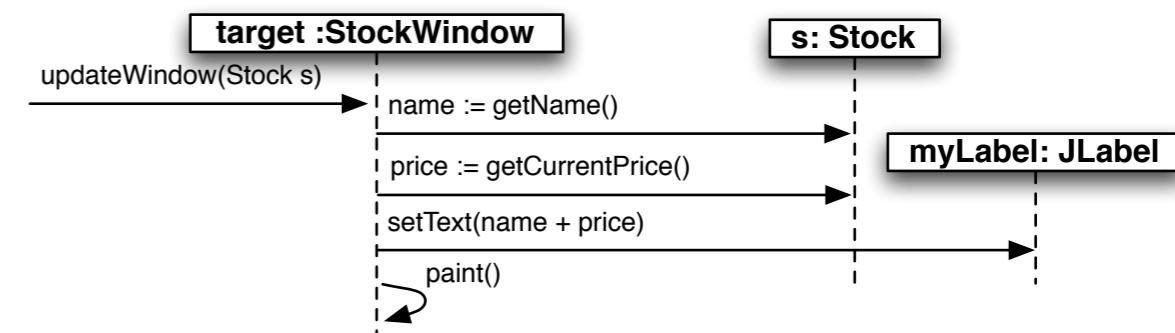
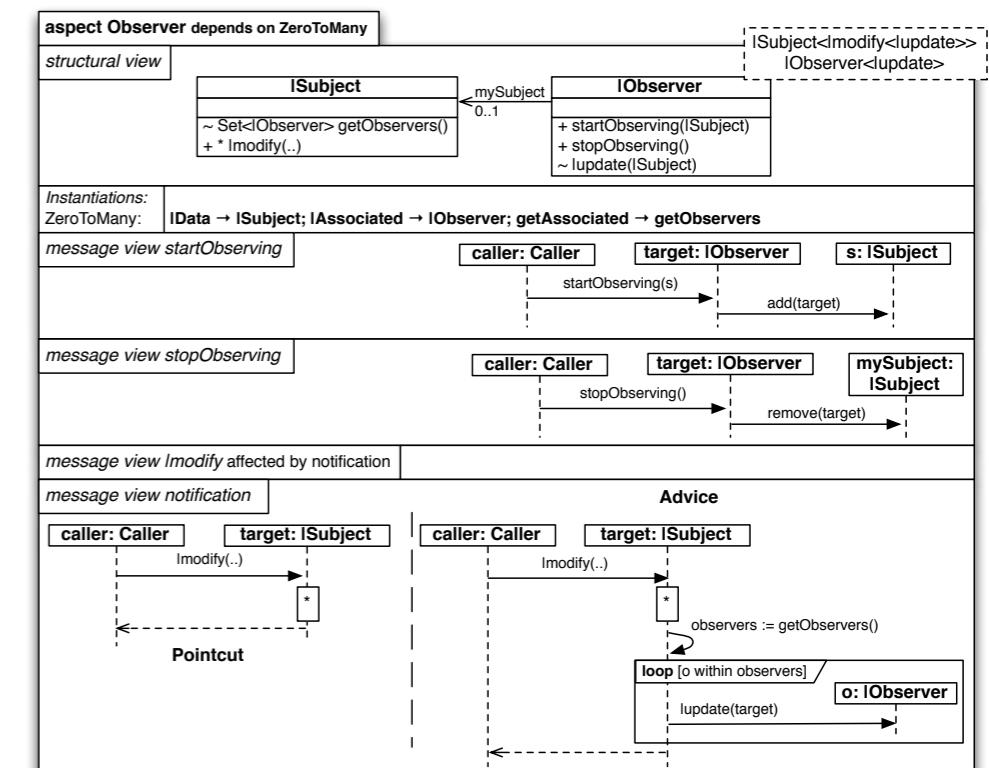
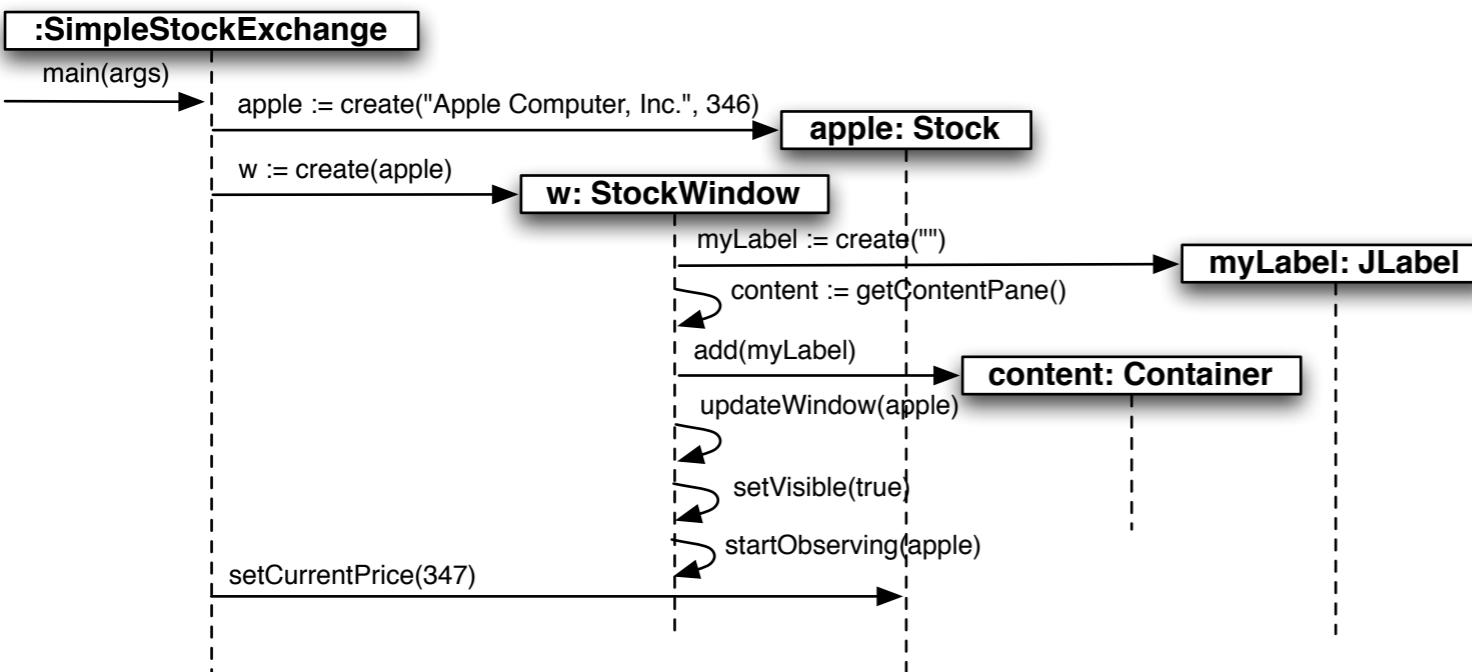
Weaving



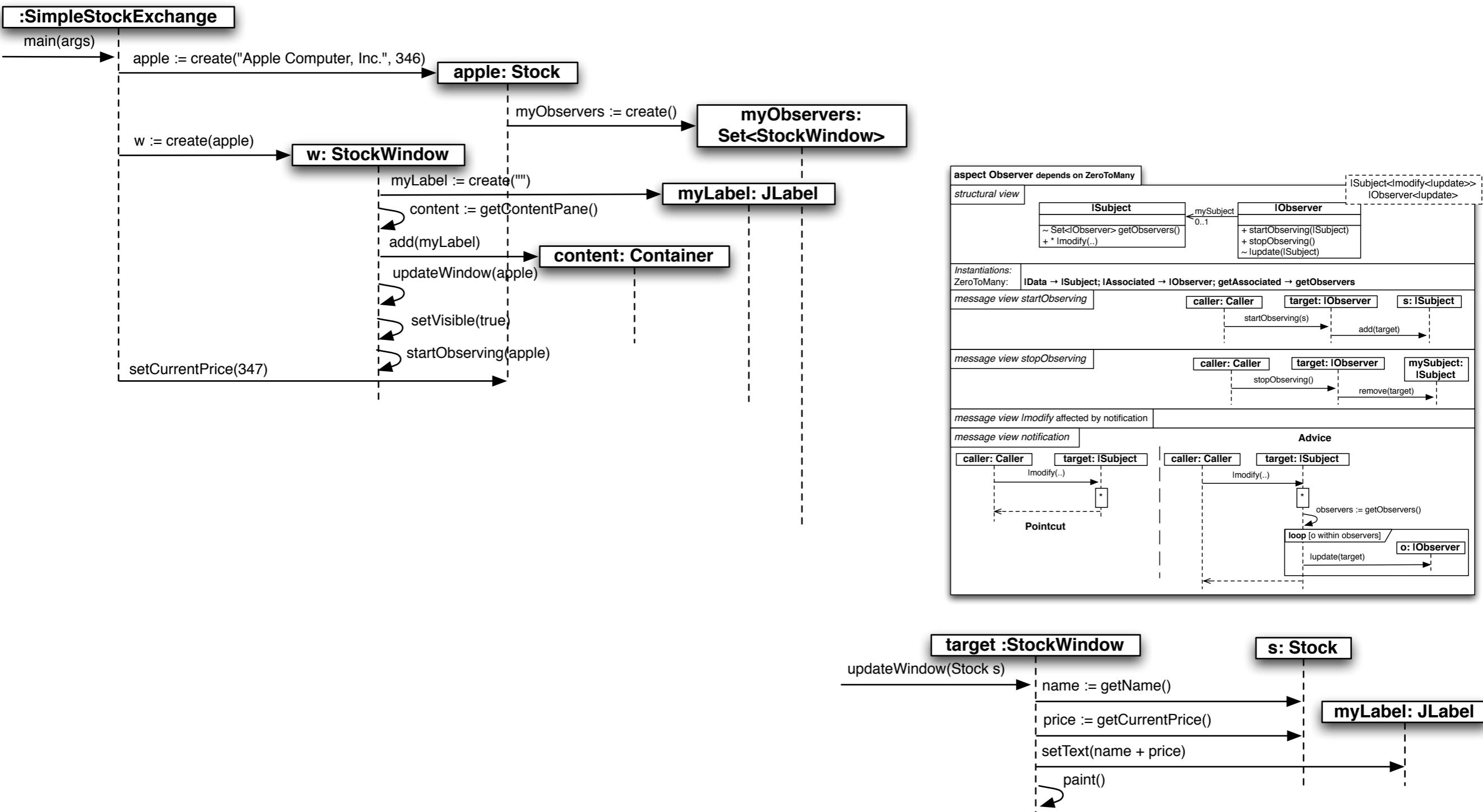
Vista Mensajes (Observer)



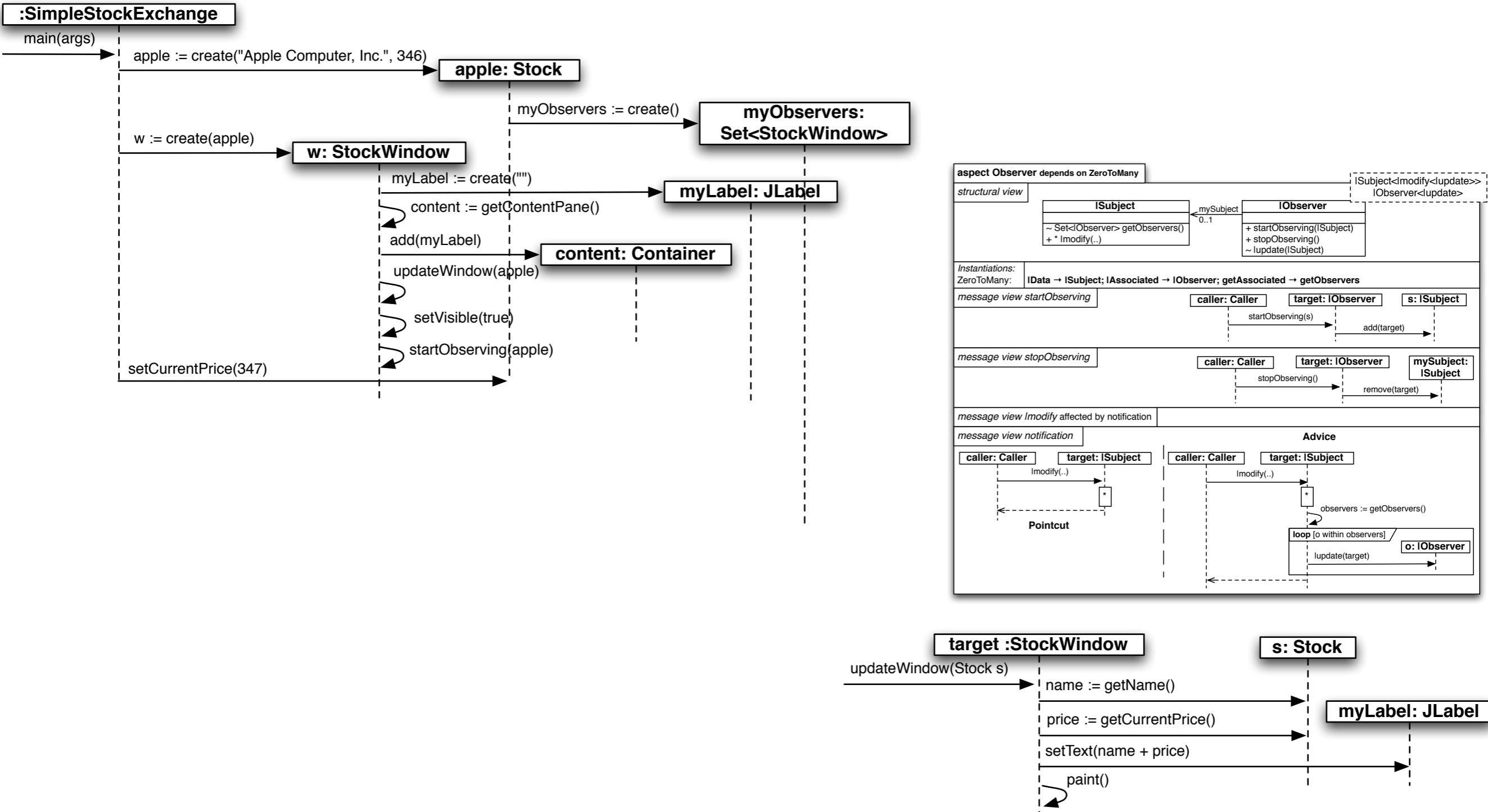
Weaving - diagramas de secuencia



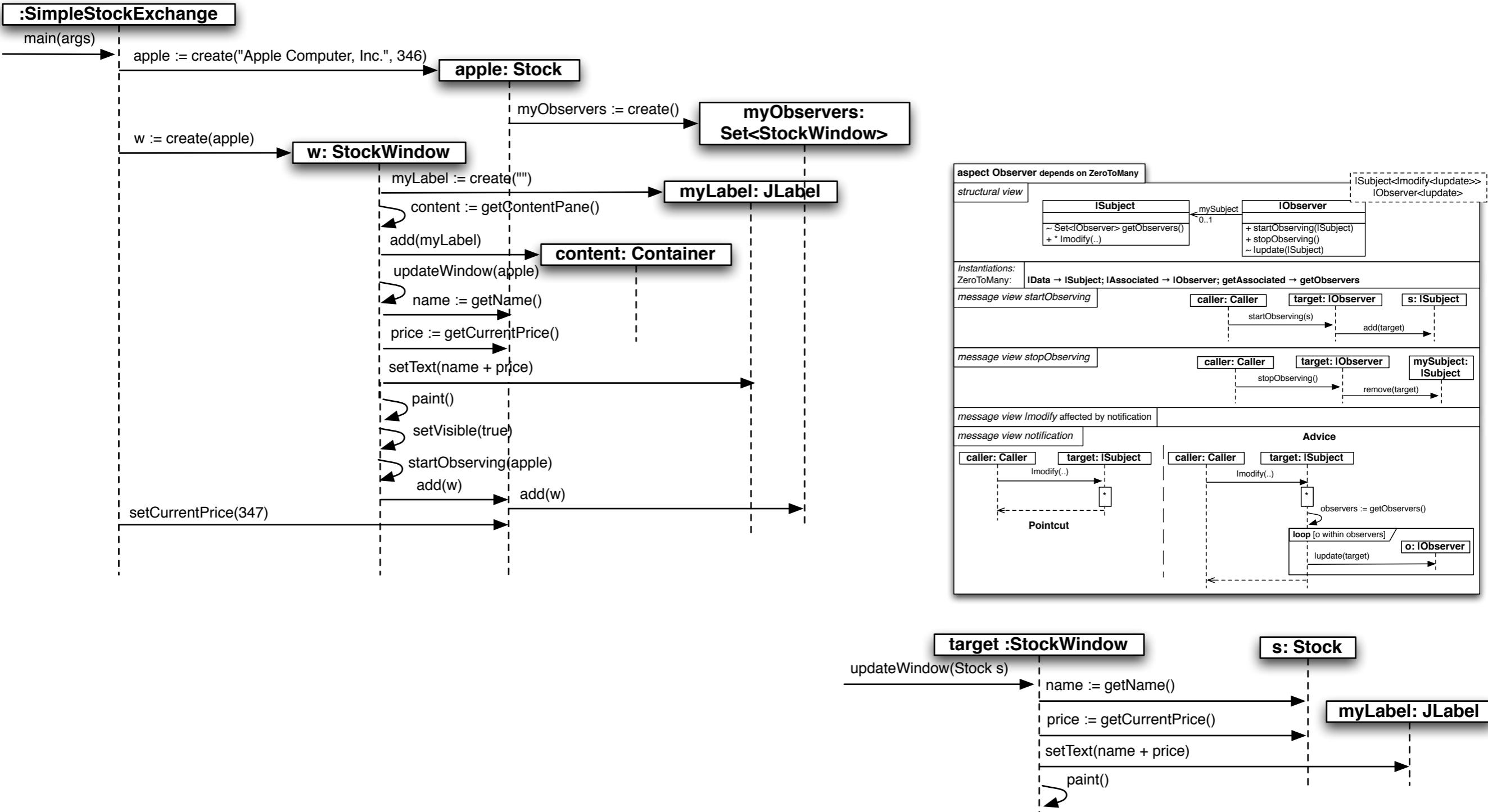
Weaving - diagramas de secuencia



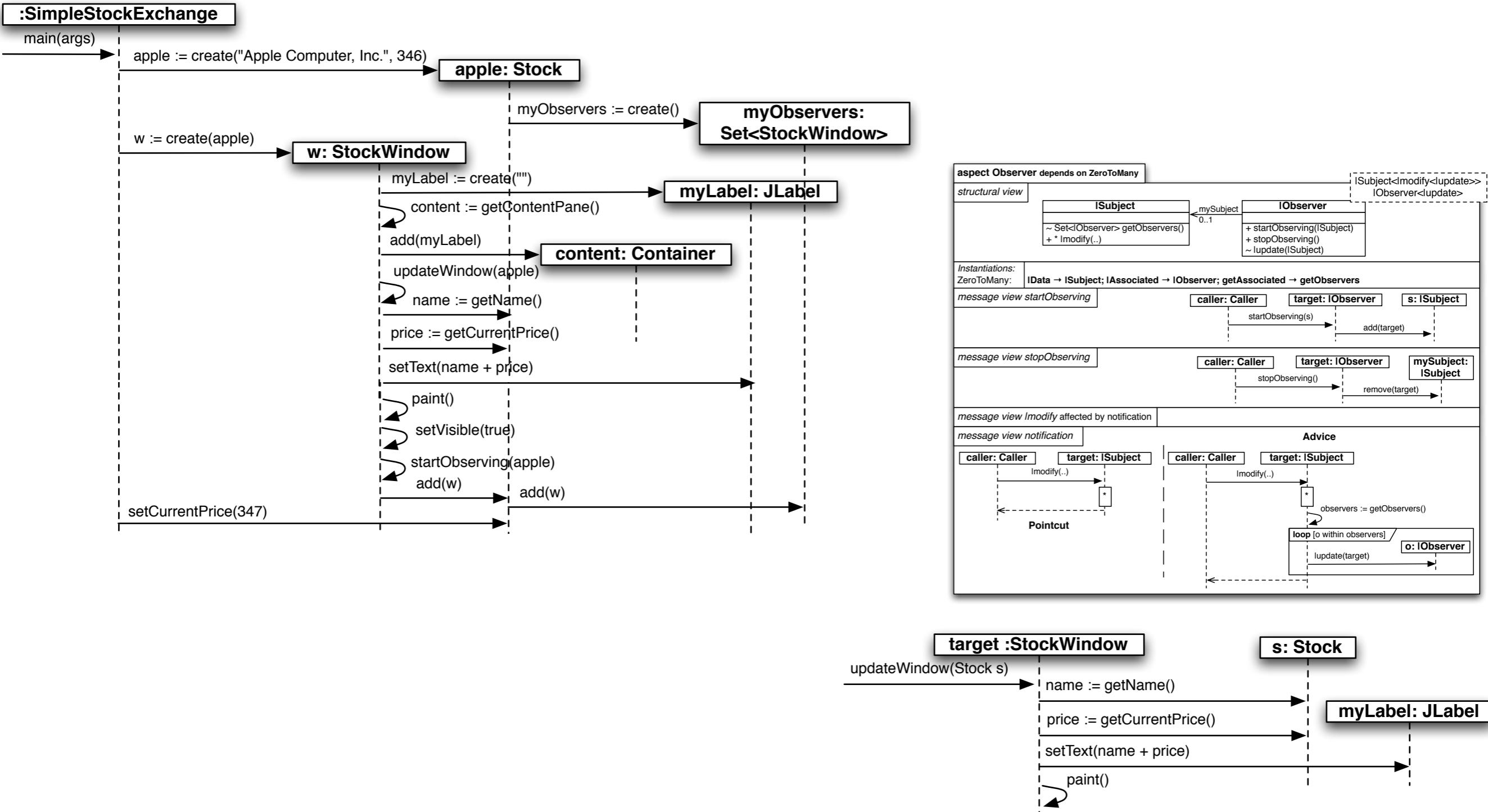
Weaving - diagramas de secuencia



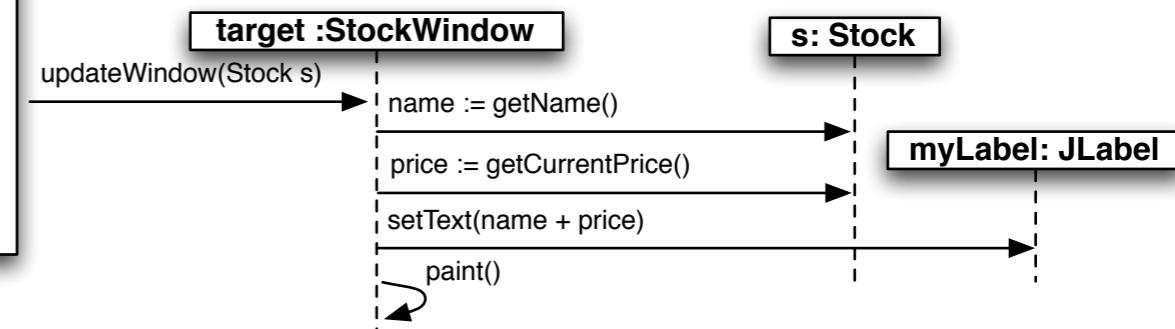
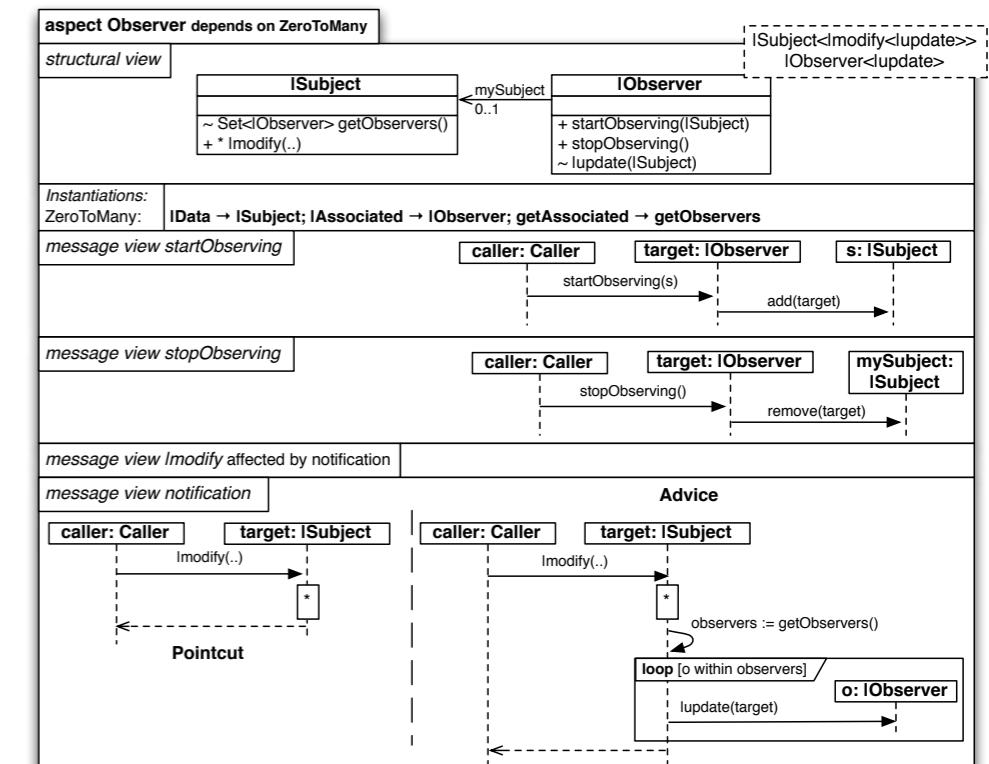
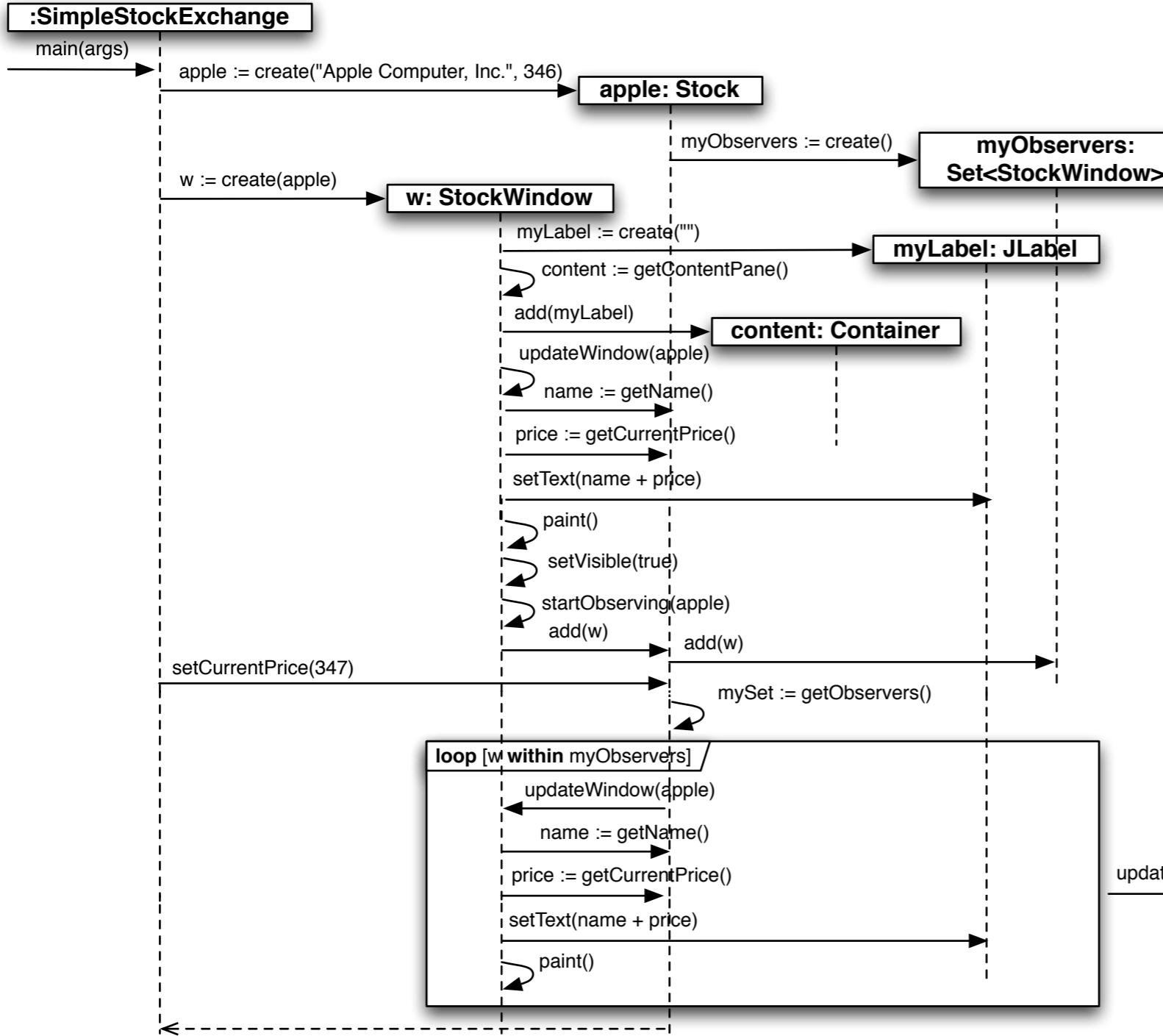
Weaving - diagramas de secuencia



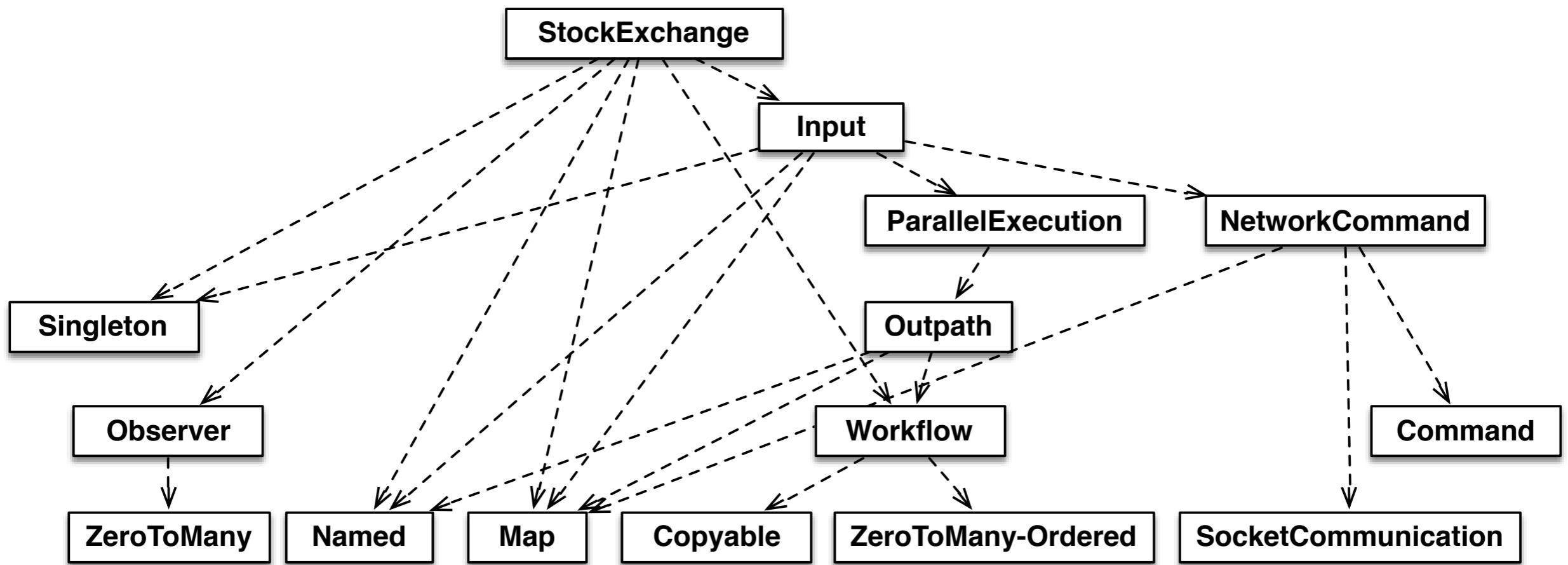
Weaving - diagramas de secuencia



Weaving - diagramas de secuencia

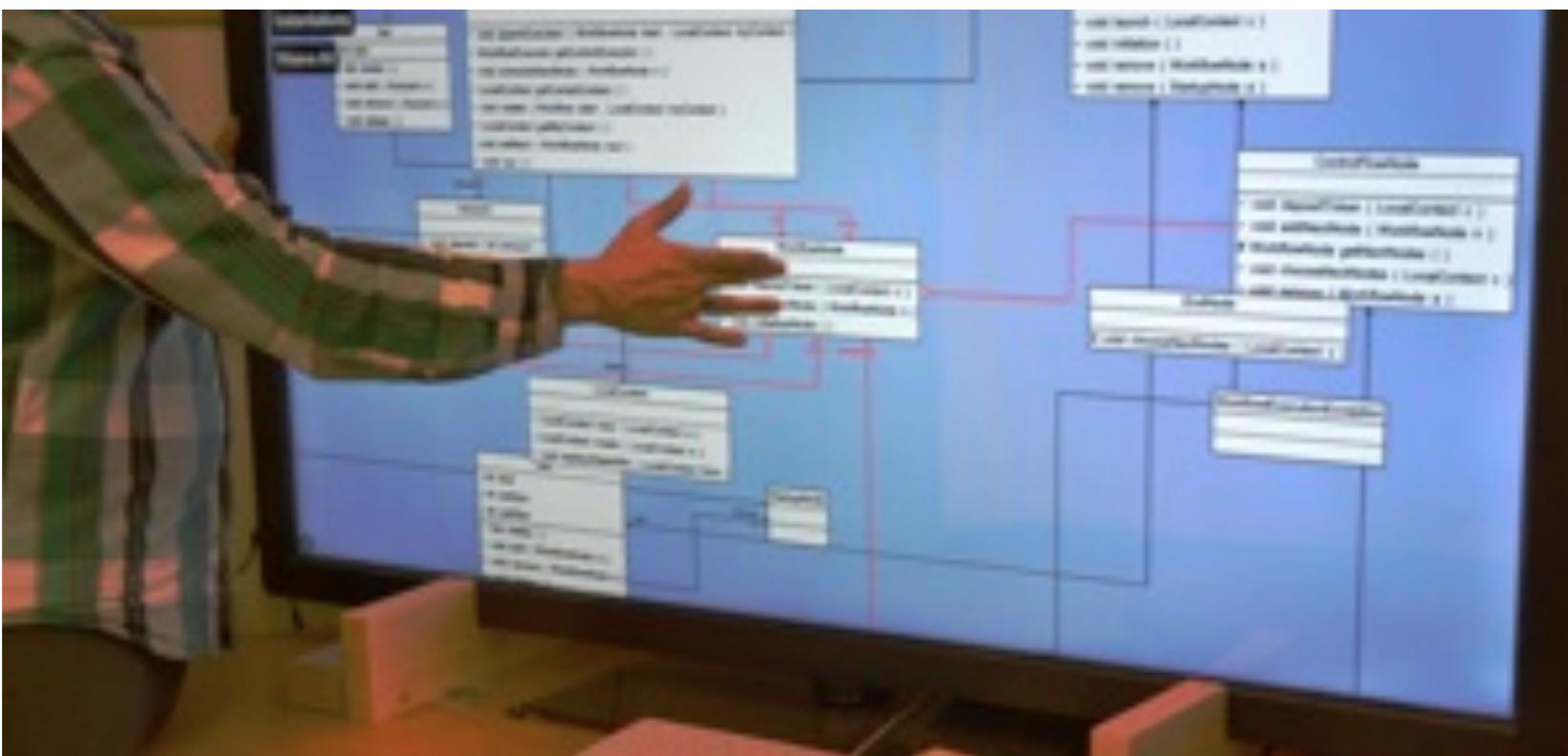


Jerarquía



TouchRAM

- Modela aspectos reutilizables por medio de una notación gráfica y una interfaz táctil.



Oportunidad

"El formato **textual** es por lo general más usado, **escala mejor** y construirlo requiere mucho menos esfuerzo" [Voelter]

TextRAM

Un lenguaje de especificación de dominio textual, para el desarrollo y manejo de modelos RAM.

Estrategia

- Model Driven Software Engineering (MDSE)
- Domain-Specific-Languages (DSL)
- Language Workbench (Xtext, Xtend & Eclipse)
- Herramienta de visualización (Graphviz)

MDSE

- Metodología para aplicar las ventajas del modelamiento a las actividades de ingeniería de software.
- Software = Modelos + Transformaciones
- Lenguaje de modelamiento (DSL)
- Meta-modelado
- Transformaciones (M2M, M2T)

Estela Maya -
Copán





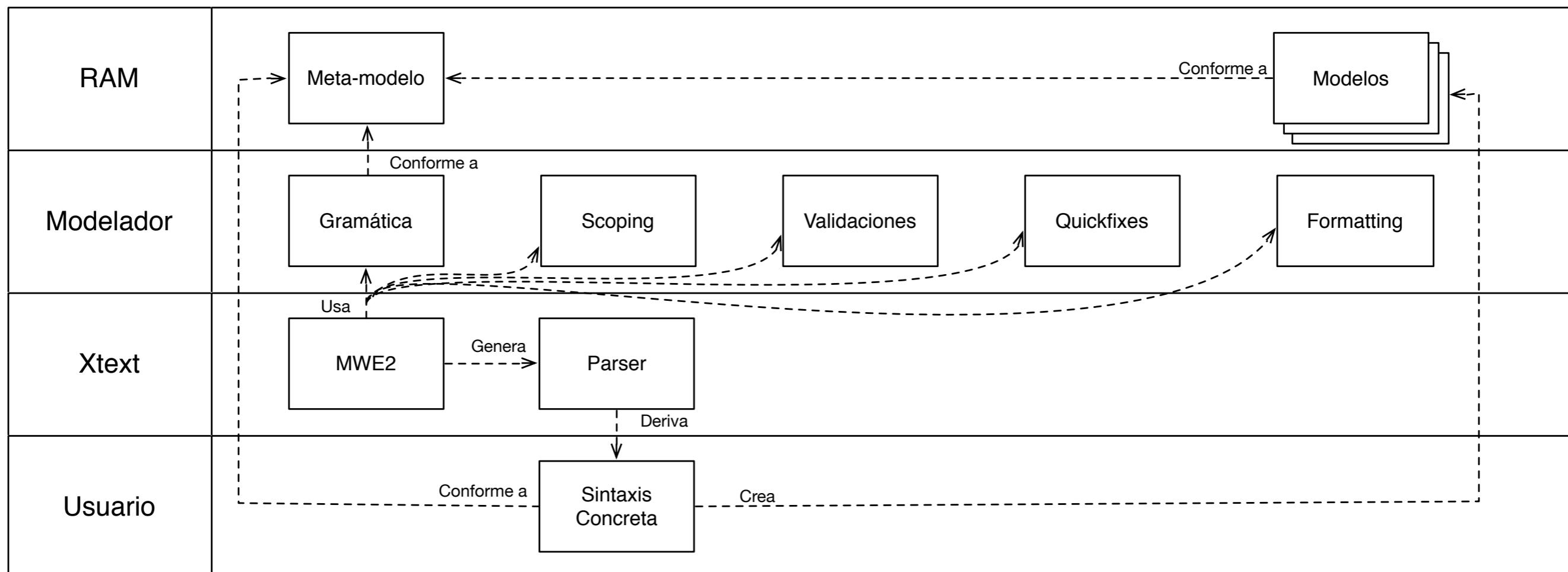
General vs
Expecífico



General vs
Expecífico

DSL

Metodología

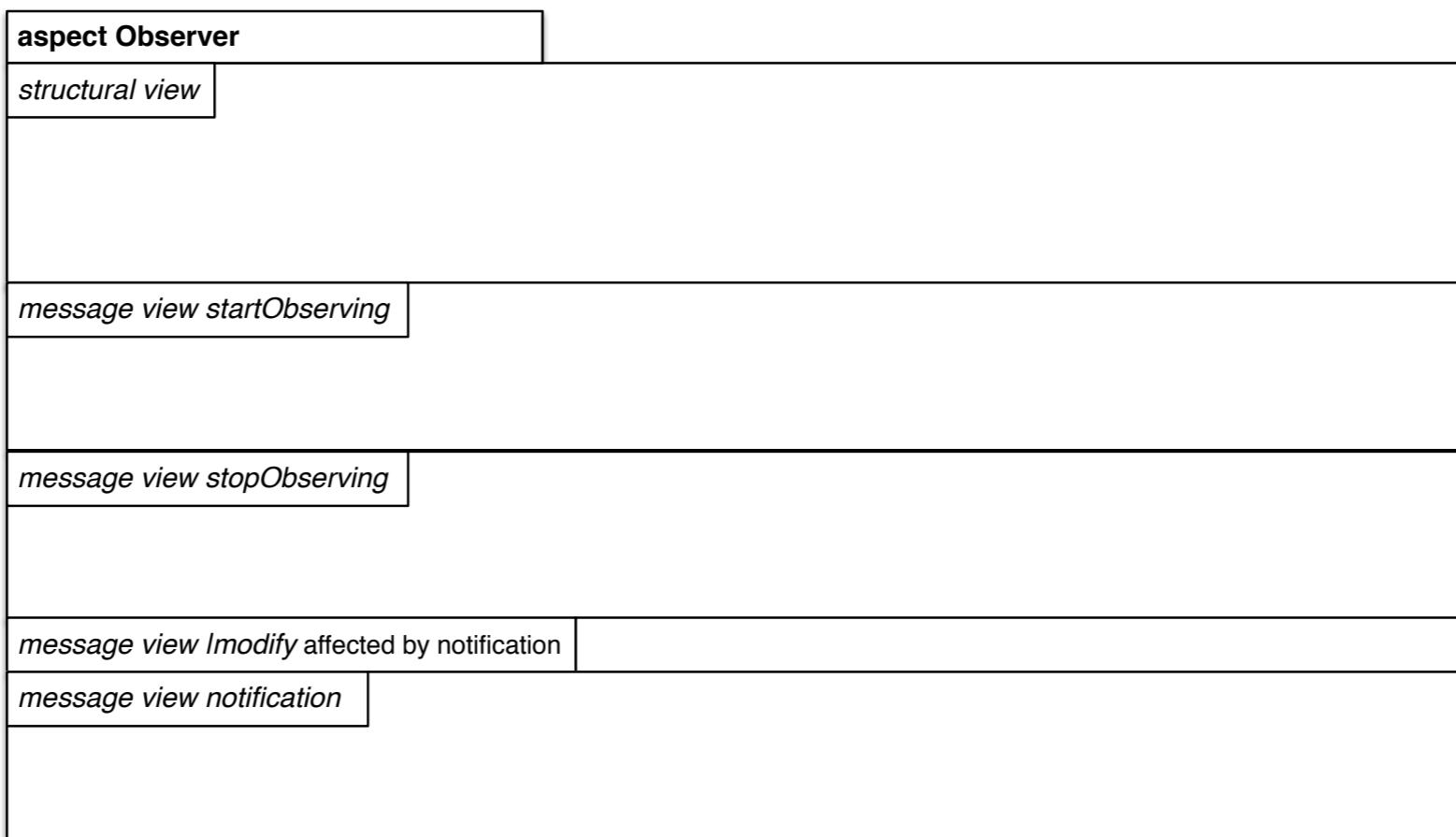


```

aspect Observer {
    structure { ... }

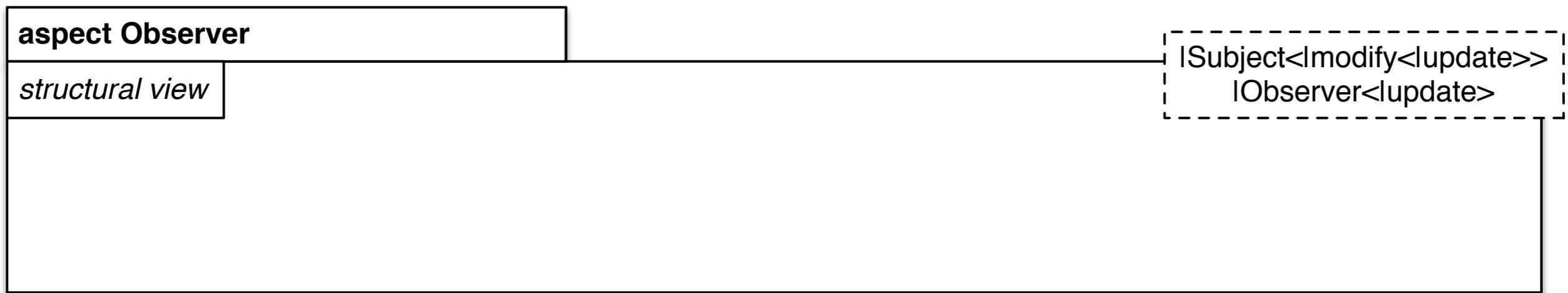
    message startObserving { ... }
    message stopObserving { ... }
    message |modify affectedBy notification
    message notification { ... }
}

```



Aspect
Model

```
aspect Observer {  
    structure {  
        mandatory {  
            | Subject<|modify<|update>>  
            | Observer<|update>  
        }...  
    }  
}
```

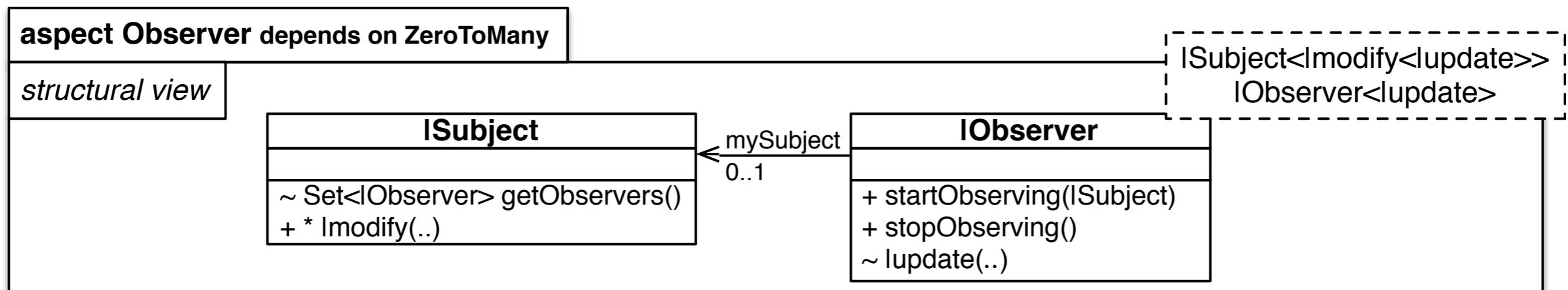


Mandatory
instantiations

```

aspect Observer dependsOn ZeroToMany {
    structure {
        mandatory { ... }
        class |Subject {
            ~ Set<|Observer> getObservers()
            + *modify(..)
        }
        class |Observer {
            + startObserving(|Subject)
            + stopObserving()
            ~ |update(..)
        }
    }
    ...
}

```



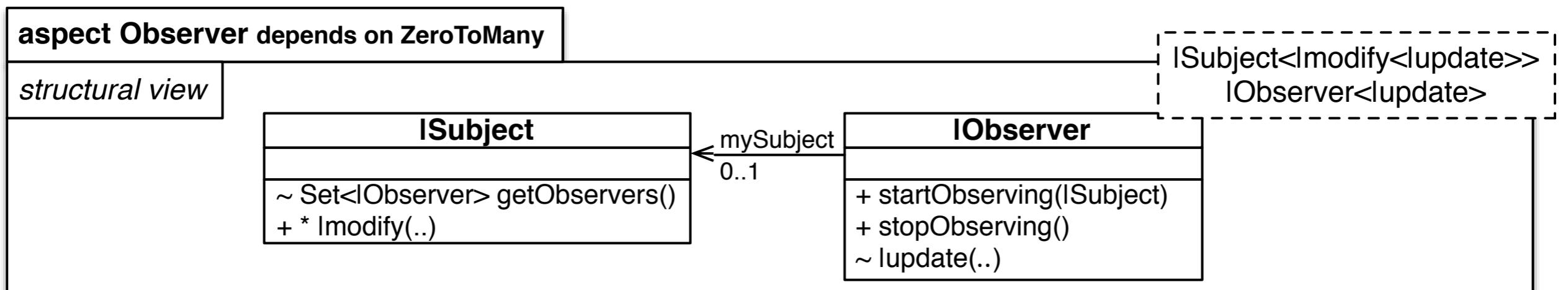
class definition

```

aspect Observer dependsOn ZeroToMany {
    structure {
        mandatory { ... }
        class |Subject { ... }
        class |Observer { ... }

        associations {
            |Observer -> 0..1 |Subject { mySubject }
        }
    }
    ...
}

```



class association

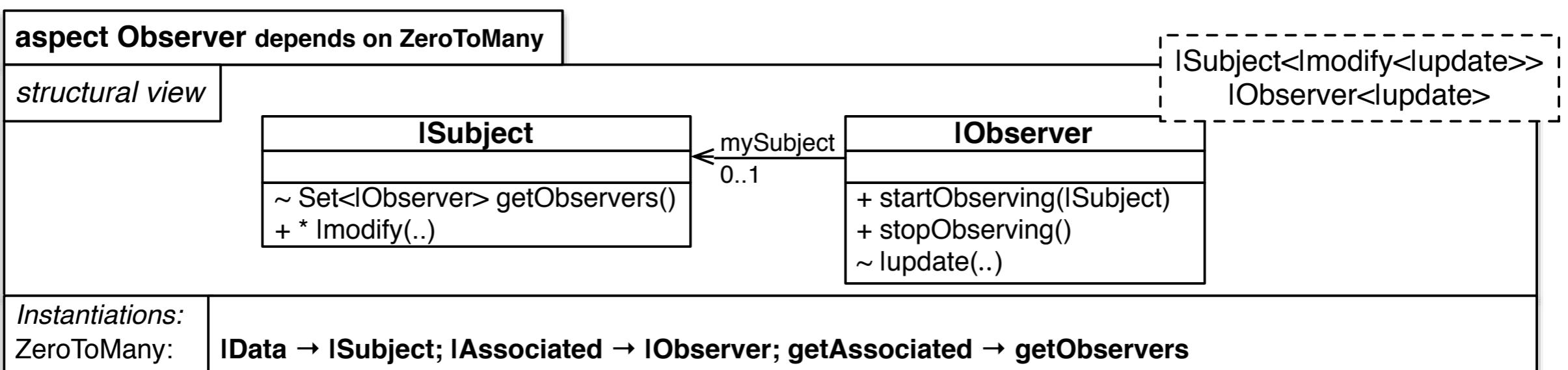
```

aspect Observer dependsOn ZeroToMany {
    structure {
        mandatory      { ... }
        class |Subject { ... }
        class |Observer { ... }
        associations   { ... }

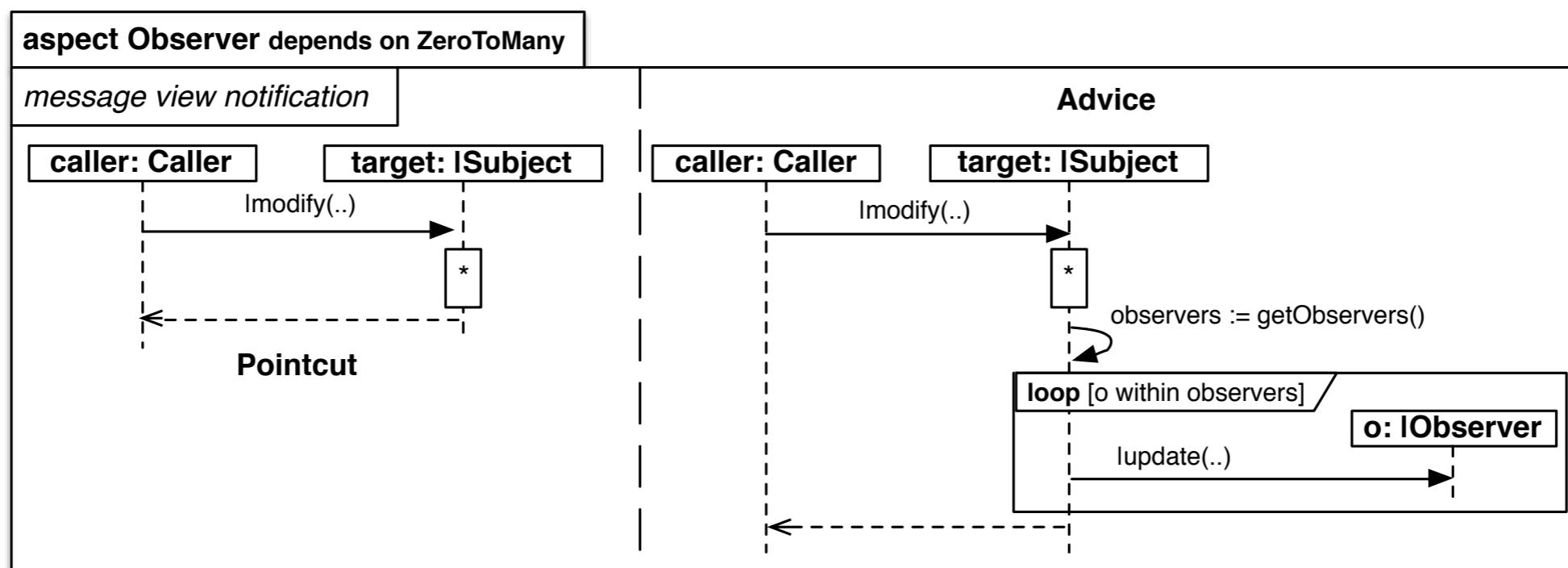
        instantiations {
            |Data          -> Subject
            |Associated    -> |Observer
            getAssociated -> getObservers
        }
    }

    ...
}

```



```
message notification {  
    pointcut {  
        caller:Caller -> target:|Subject { |modify(..) } *  
    }  
    advice {  
        caller:Caller -> target:|Subject { |modify(..) } *  
        target:|Subject -> target:|Subject { observers:= getObservers() }  
  
        loop [o within observers] {  
            target:|Subject -> o:|Observer { |update(..) }  
        }  
    }  
}
```



message view

Observer Aspect Model

