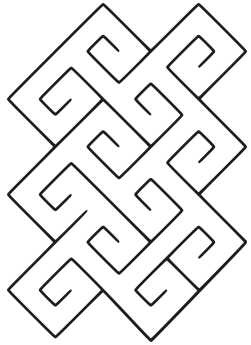
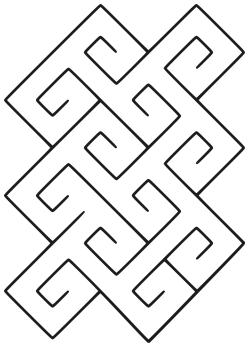


OBJETOS

Patrones de diseño



Agenda

- Patrones estructurales
- Patrones de comportamiento

Agenda

- Patrones estructurales
- Patrones de comportamiento

Patrones estructurales

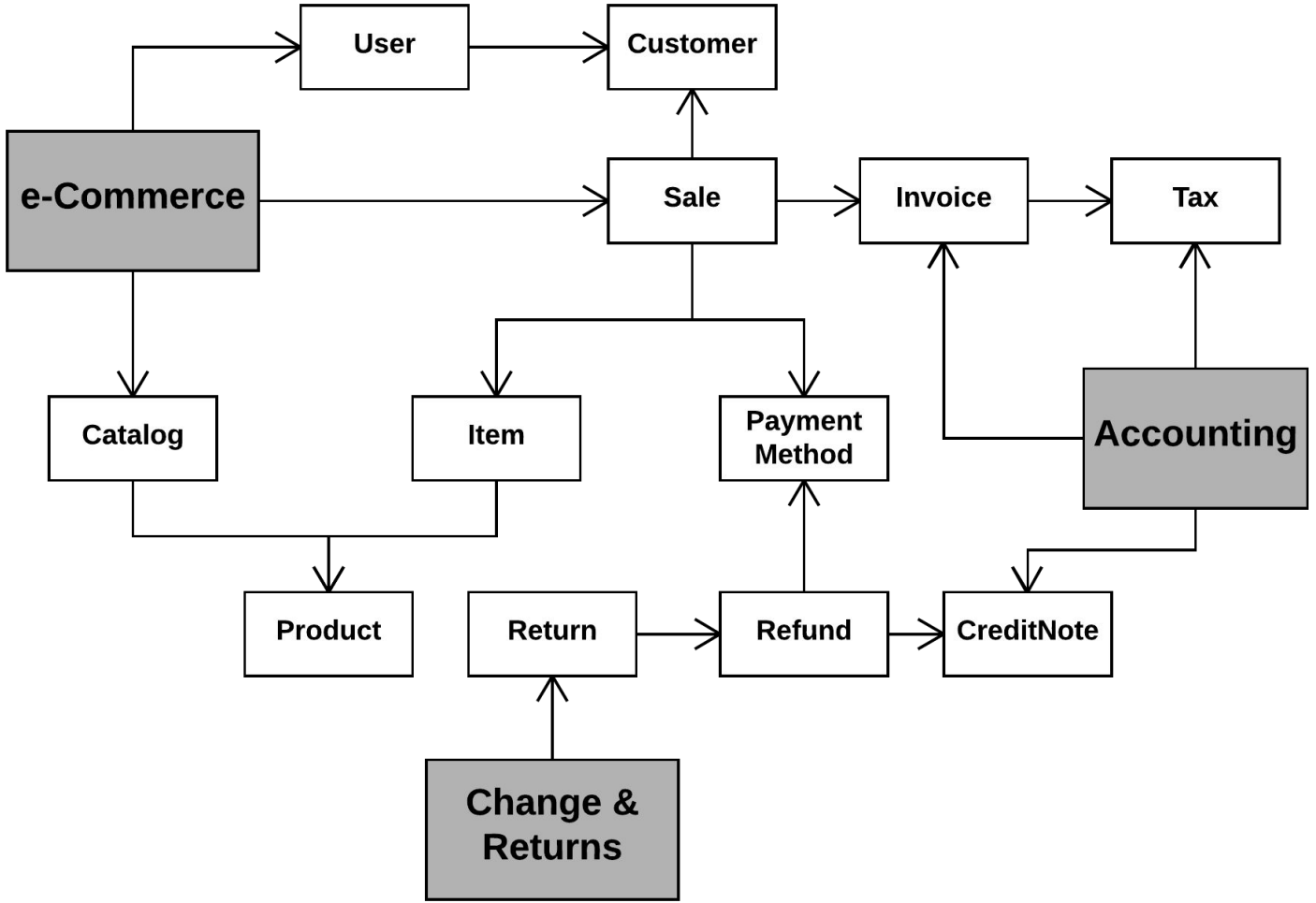
- Facade
- Proxy
- Composite

Facade

Se tiene una interfaz altamente compleja, de la que se usa una porción muy reducida en la gran mayoría de los casos.

- Unifica el acceso a un conjunto de servicios o subsistemas
- Provee acceso de alto nivel para permitir un uso más simple
- Separa uso estándar y personalizado
- Reduce el acoplamiento con los subsistemas detrás de la fachada

Facade



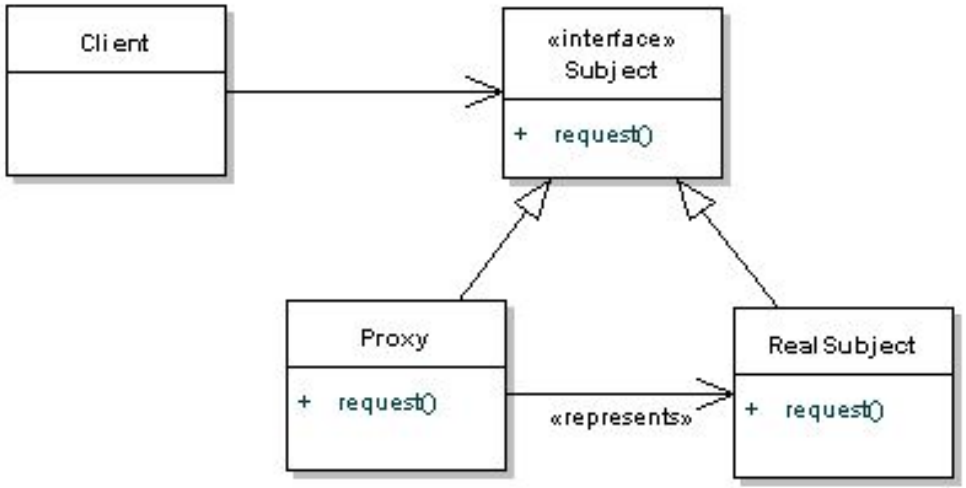
¿Dudas?

Proxy

Cuando se necesita postergar operaciones costosas, controlar el acceso a determinados tipos de recursos, acortar el tiempo de acceso a un objeto que lleva mucho tiempo conseguir, un proxy nos es de mucha ayuda.

- Permite postergar operaciones costosas y evitables
- Casos de uso
  - Remoto - para objetos separados de lugar - embajador
  - Virtual - para objetos costosos
  - Protección - para objetos que necesitan manejar permisos o niveles de acceso
  - Referencia Inteligente - para usarse como puntero inteligente - manejo de lockeos
- Mocks

Proxy


Client
«interface»
Subject
+ request()
Proxy
+ request()
«represents»
RealSubject
+ request()

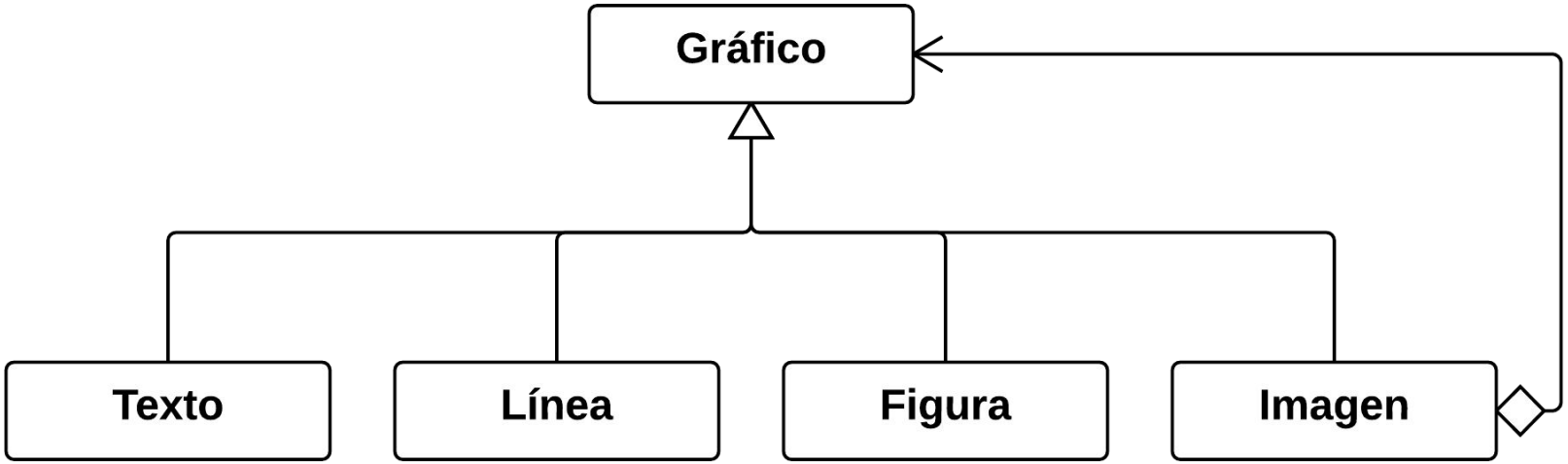
Composite

Tratar de forma unificada objetos simples y objetos compuestos, organizando estructuras jerarquizadas con interfaz uniforme.

- Simplifica el código cliente
- Unifica el modo de uso de objetos compuestos de individuales

Composite

```
for (Grafico grafico : misGraficos) {
    grafico.dibujar()
}
```



Agenda

- Patrones estructurales
- Patrones de comportamiento

Patrones de comportamiento

- Observer
- Visitor
- ...más en la próxima

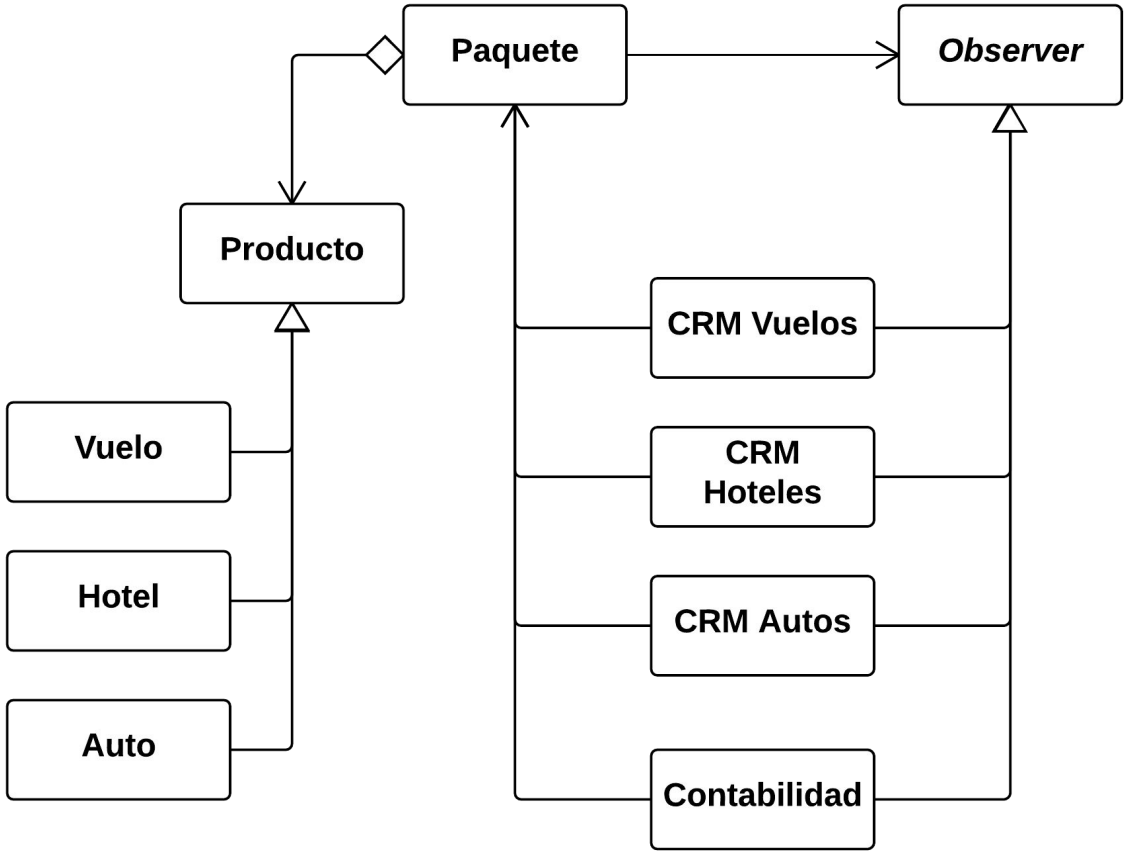
Observer

Cuando tenemos un objeto de interés para otros, creamos una interfaz (la generalidad) que implementan aquellos que están interesados en dicho objeto (la particularidad), de manera que el objeto de interés puede notificarle que hubo una modificación, manteniéndose completamente desacoplado de aquel que depende sí.

- Desacopla partes interdependientes
- Abstrae un objeto de aquellos que lo consumen
- Organiza trabajo bajo demanda
- Optimiza recursos evitando consultas innecesarias

Observer

```
public void registrarCambio(Cambio c) {
    ...
    for (Observer observer: myObservers) {
        observer.notify(c.getTipo());
    }
    ...
}
```

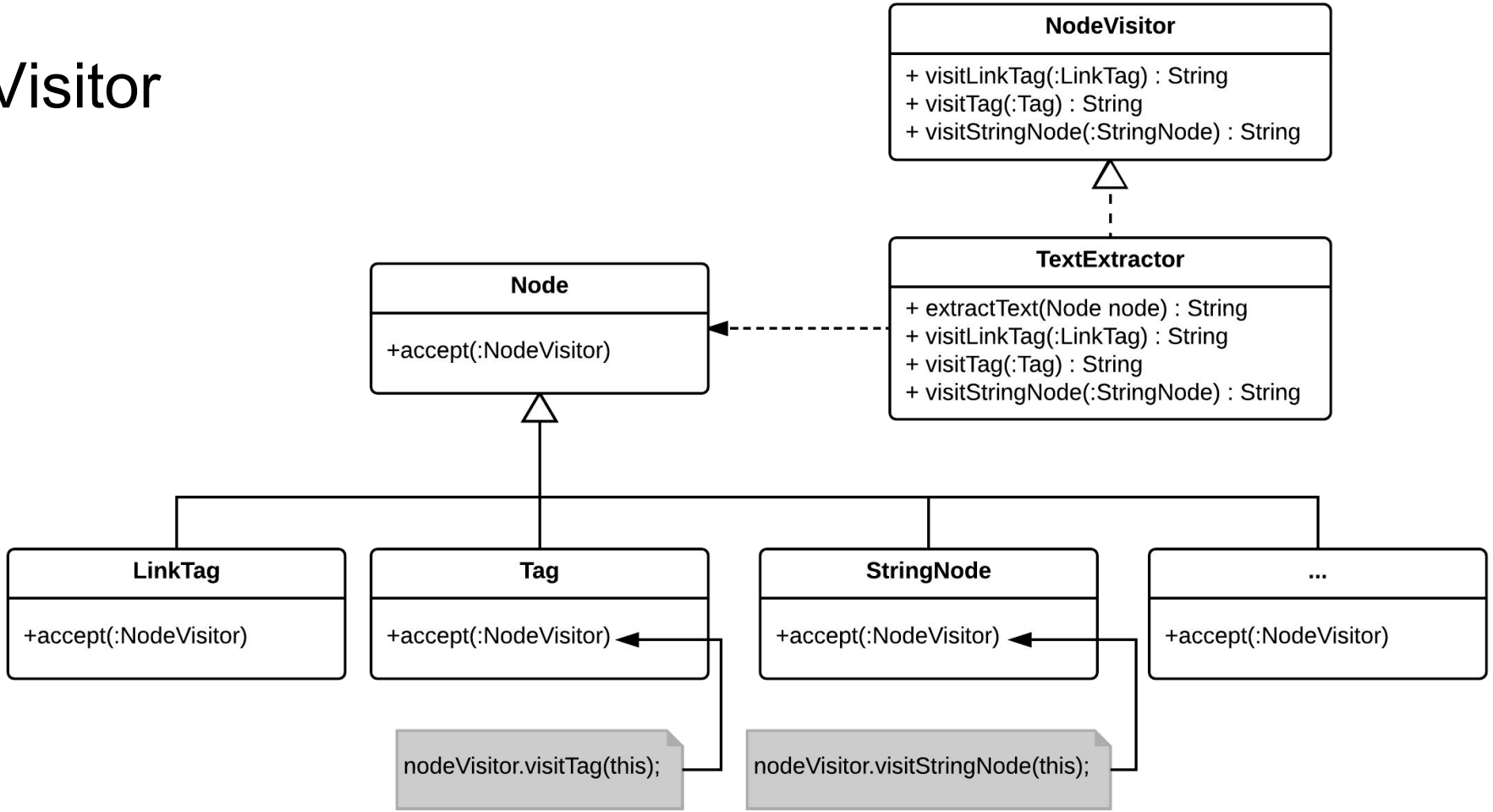


Visitor

Necesito variar mi accionar según 2 factores: el tipo de aquello sobre lo que estoy actuando y el tipo de resultado que pretendo lograr

- Habilita una arista alternativa de modificación o cambio
- Desacopla jerarquías paralelas

Visitor


NodeVisitor
+ visitLinkTag(:LinkTag) : String
+ visitTag(:Tag) : String
+ visitStringNode(:StringNode) : String
TextExtractor
+ extractText(Node node) : String
+ visitLinkTag(:LinkTag) : String
+ visitTag(:Tag) : String
+ visitStringNode(:StringNode) : String
Node
+accept(:NodeVisitor)
LinkTag
+accept(:NodeVisitor)
Tag
+accept(:NodeVisitor)
StringNode
+accept(:NodeVisitor)
...
+accept(:NodeVisitor)
nodeVisitor.visitTag(this);
nodeVisitor.visitStringNode(this);

Agenda

- Patrones estructurales
- Patrones de comportamiento

¿ ?

Bibliografía

- Design Patterns CD - Gamma, Helm, Johnson, Vlissides