Uso de Patrones de diseño

jsr@dit.upm.es

Relaciones

- La importancia de la relaciones en los diseños de orientación a objetos.
- Identificación de situaciones comunes.
- Necesidad de reciclar “ideas” no “código”
- Identificado por la banda de los cuatro
  - Gamma, Helm, Johnso y Vlissides.
- Son como “fábulas” historietas cortas con moraleja.

Motivación

- El diseño Orientado a Objetos (DOO) es un arte más que una ciencia.
  - Habilidad, talento
  - Experiencia
  - Creatividad, imaginación.
- Muy poca experiencia en estos diseños (10-20 años).

Problema

- Diseño con objetos: muy complicado.
- Diseño arquitectónico: Uso de múltiples objetos para resolver un problema.
- Literatura:
  - Primero “casar las letras”· (principios)
  - Ser capaz de escribir. (reglas)
  - Estudiar literatura. (diseños)

Solución

- Solución del campo de la arquitectura (de la de casas):

Identificar problemas típicos y sus soluciones adecuadas.

- No intentar ser siempre novedoso: utilizar y comunicar soluciones que funcionan.

Arquitectura

- Chirstopher Alexander
  - “the timeless way of building” (1979).
  - Nuevo enfoque: el mismo conjunto de leyes determina la estructura de una ciudad, un edificio o de una habitación.
  - Aplicación a la arquitectura y a los diseños de alfombras iraníes.

Christopher Alexander

- Un arquitecto que intenta observar la repetición de ciertos aspectos de diseño:
  - La calidad sin nombre.
  - La puerta a la solución.
  - El camino para alcanzarlo.
- Observación por ing. Software de mismas relaciones

Início

- Patrón: “Una solución para un problema en un contexto”.
- Lenguaje de Patrones: ??
- Propósito:
  - Reutilización eficiente.
  - Diseminación de soluciones.

Partes de un patrón

1 Nombre del patrón

(una de las partes más difíciles).

2 Problema: Descripción de cuando utilizarlo.

3 Solución:

4 Consecuencias (buenas y malas).

Reusabilidad

- Dificultad y complejidad de clase para ser utilizada en cualquier caso crece exponencialmente.
- Construir con objetos es como colocar un ladrillo sobre otro.
- Necesidad de ladrillos útiles.
  - (problemas de lenguajes tipo C++).

Antipatrones

- No solo identificar soluciones.
- También identificar trampas en el diseño

  “.. Animal que tropieza 2 veces .. “

- Esa solución que se le ocurre a todo el mundo y siempre caemos en ella.

- Tan importantes como los patrones.

Java

- Diseñadores basados en patrones.
- Casi todos los patrones están en las clases base de Java.

- Si se conocen los patrones: fácil comprender las decisiones de diseño.
- Necesidad de conocer limitaciones para extender las clases base.

Entornos

- Biblioteca (library)
  - Ladrillos para construir una casa.
- Cajas de herramienta (toolkits)
  - Soluciones prefabricadas para pegar.
- Entornos (frameworks)
  - Conjunto de clases Cooperantes que son reutilizables en un diseño de un cierto campo.
  - Principio de Hollywood: (no nos llame, nosotros le llamaremos).

Tipos de Patrones

- Patrones arquitecturales (conceptuales)
- Patrones de Diseño (diseño)
- Idioms (trucos de programación).

- Existencia de distintos patrones en diferentes niveles de abstracción.

Necesidad de catálogos

- Existen diversos catálogos con los patrones identificados
  - El grupo de los cuatro:

    Design patterns: Elements of Reusable Object-Oriented Software (Erich Gamma, Richard Helm, ralph Johnson, and John Vlissides)

Descripción de gran número de patrones en C++ y Smalltalk.

Clasificación

- Patrones sobre creación:
  (factoría abstracta, builder, prototipo, singleton).
- Patrones estructurales
  - Como agrupar y organizar objetos

  (bridge, adapter, proxy, facade, flyweight).
- Patrones de comportamiento
  - Como se relacionan en ejecución grupos de obj.

  (command, interpreter, iterator, mediator, observer)

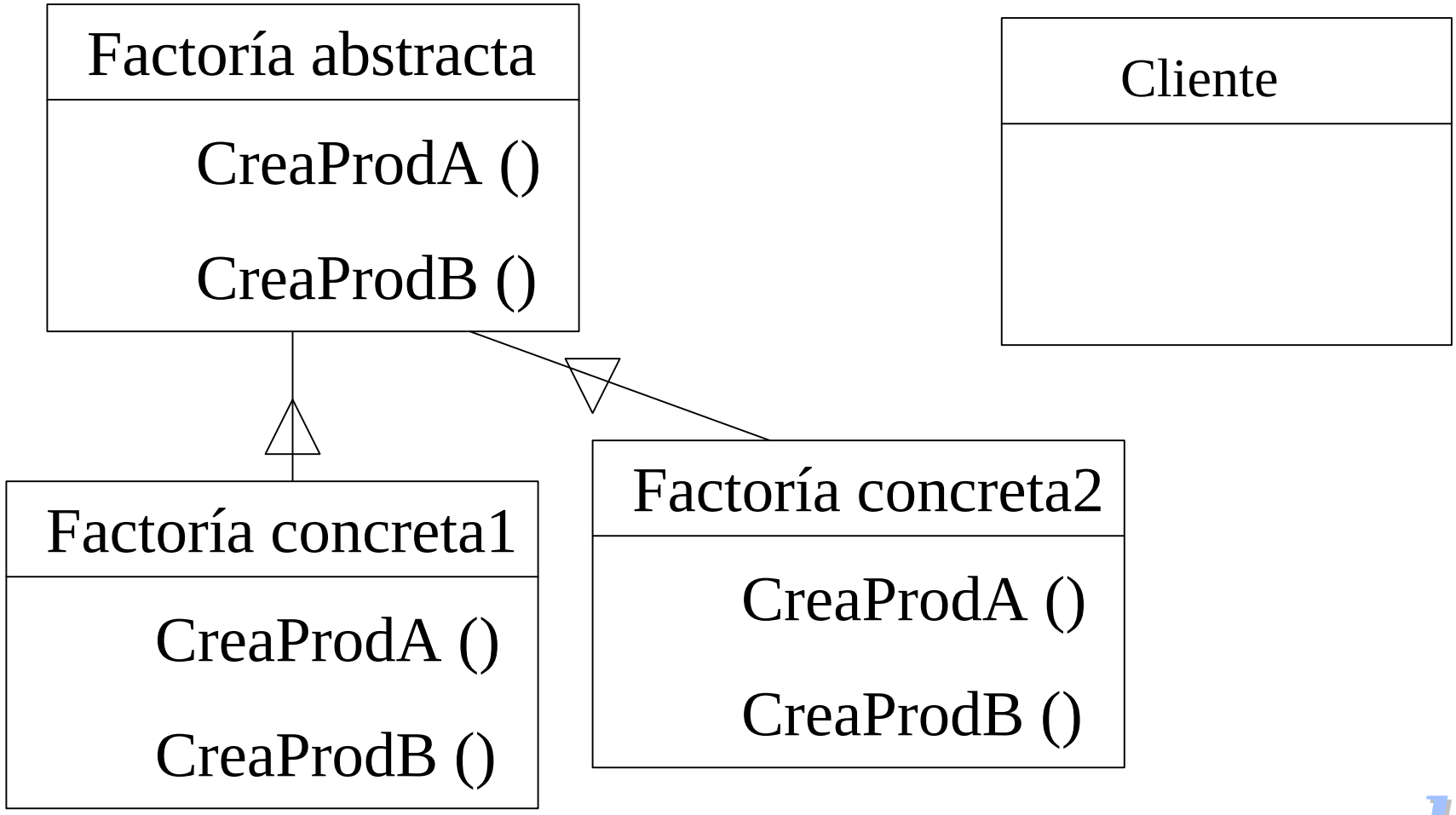
Patrones sobre creación

- Se enfrenta al problema de la instanciación.
- Muchas veces los constructores no son suficientes:
  - Delegar la creación a otros objetos
  - Generalizar y encapsular detalles para permiten su reuso.
  - Flexibiliza quien crea, que produce, como es creado y cuando.

Uso de factorías abstractas

- Queremos familias de multiples objetos que se manipulen igual:
  - Interfaz de usuario: creación de widgets.
  - Comunicaciones: creación de sockets.

Abstract factory

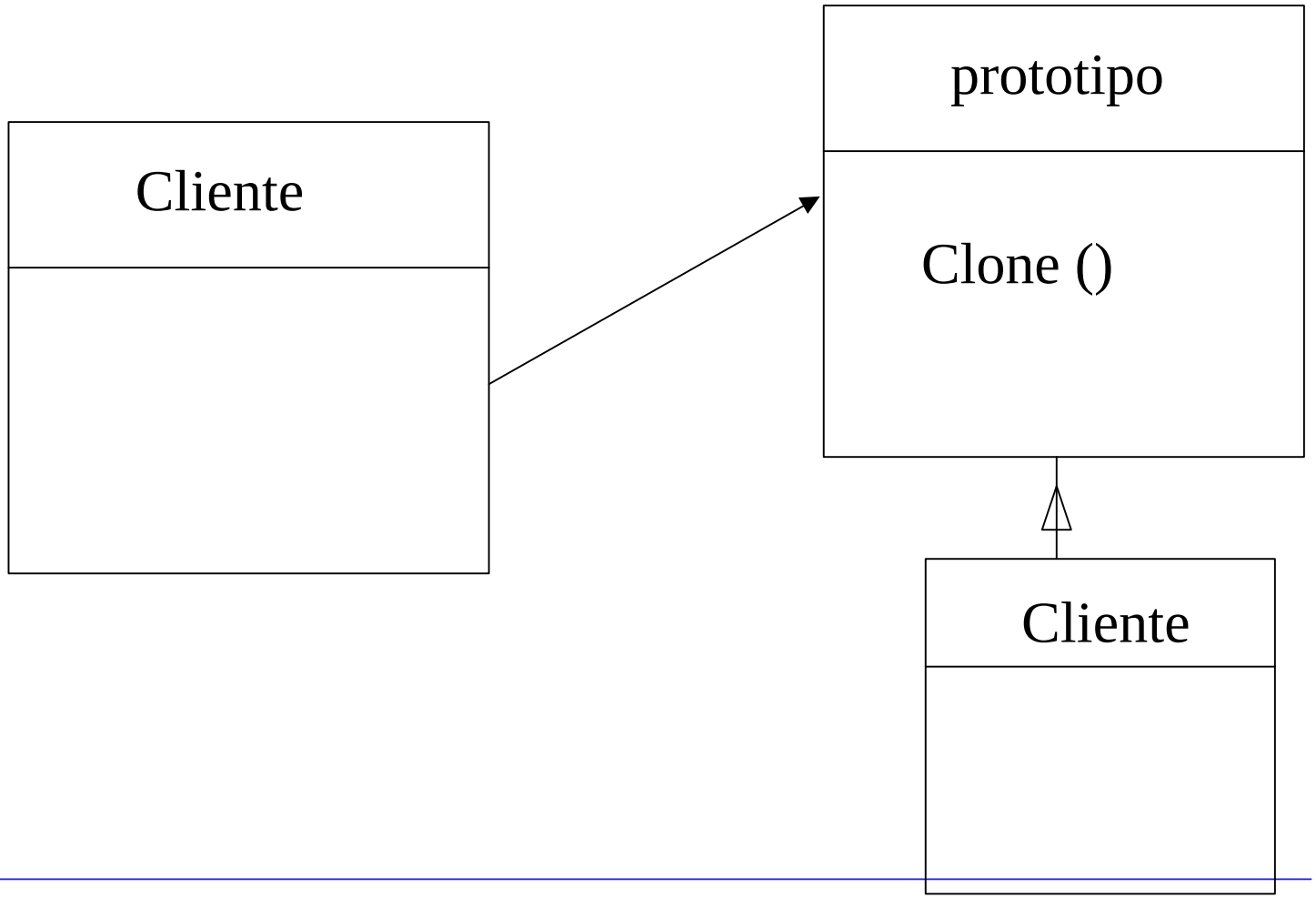


Singleton

- Solo puede haber una instancia.

```
Class Singleton {
  public:
      static Singleton * instance (void);
      void DoSomething (int);
  protected:
      singleton ();
  private:
      static Singleton * Sinstance;
};
```

Prototype

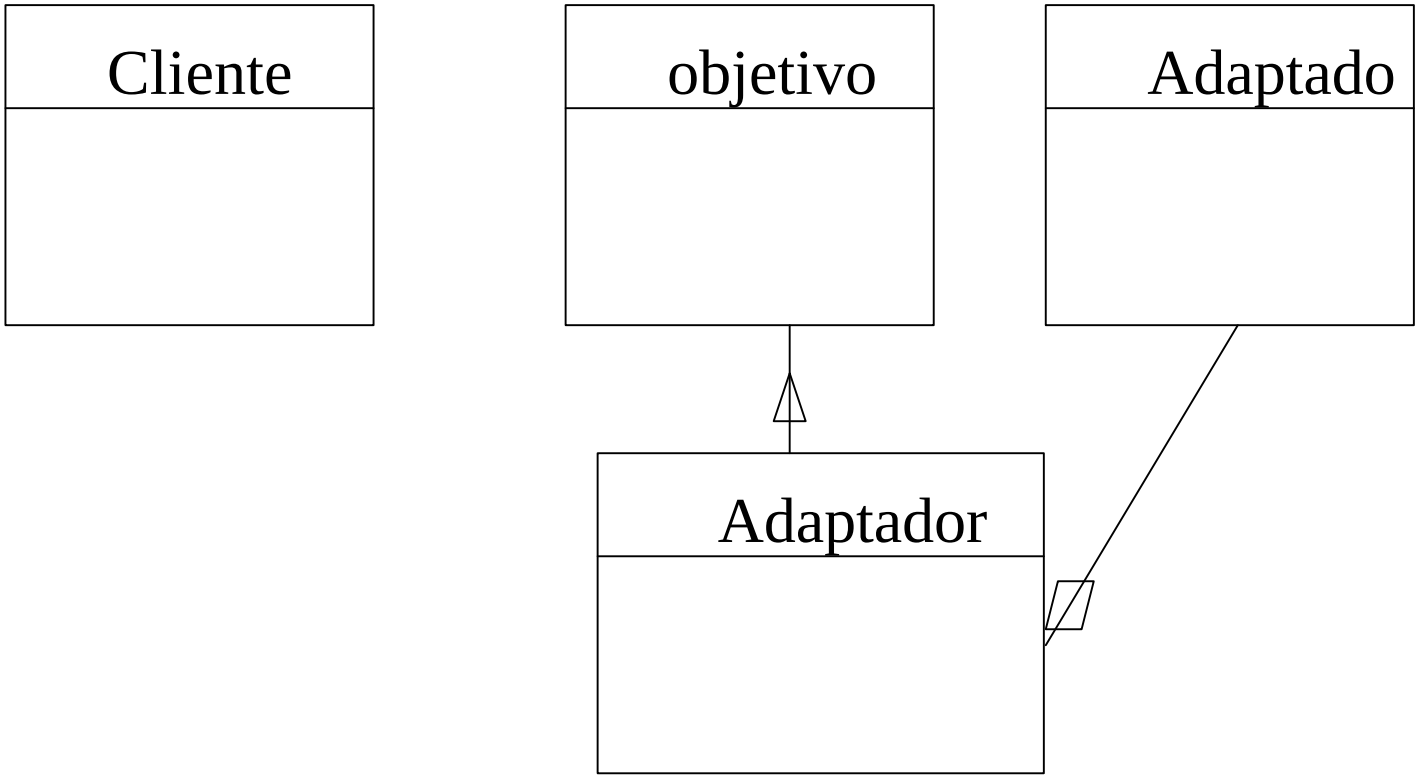


Patrones estructurales

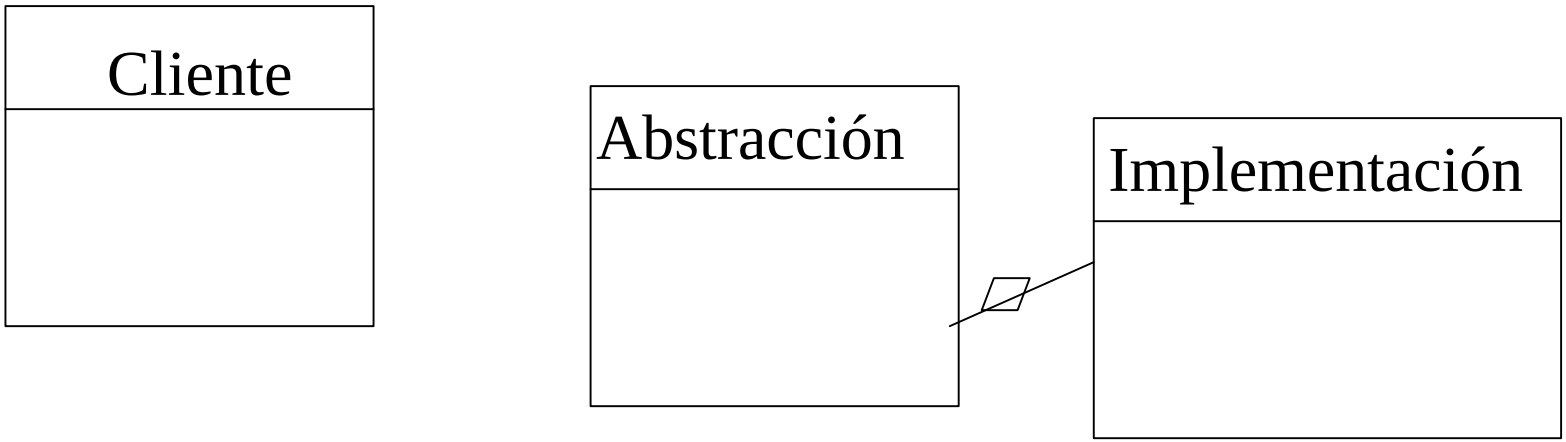
- Estudia como se relacionan los objetos en tiempo de ejecución (mapa del sistema).

- Sirven para diseñar las interconexiones entre objetos.

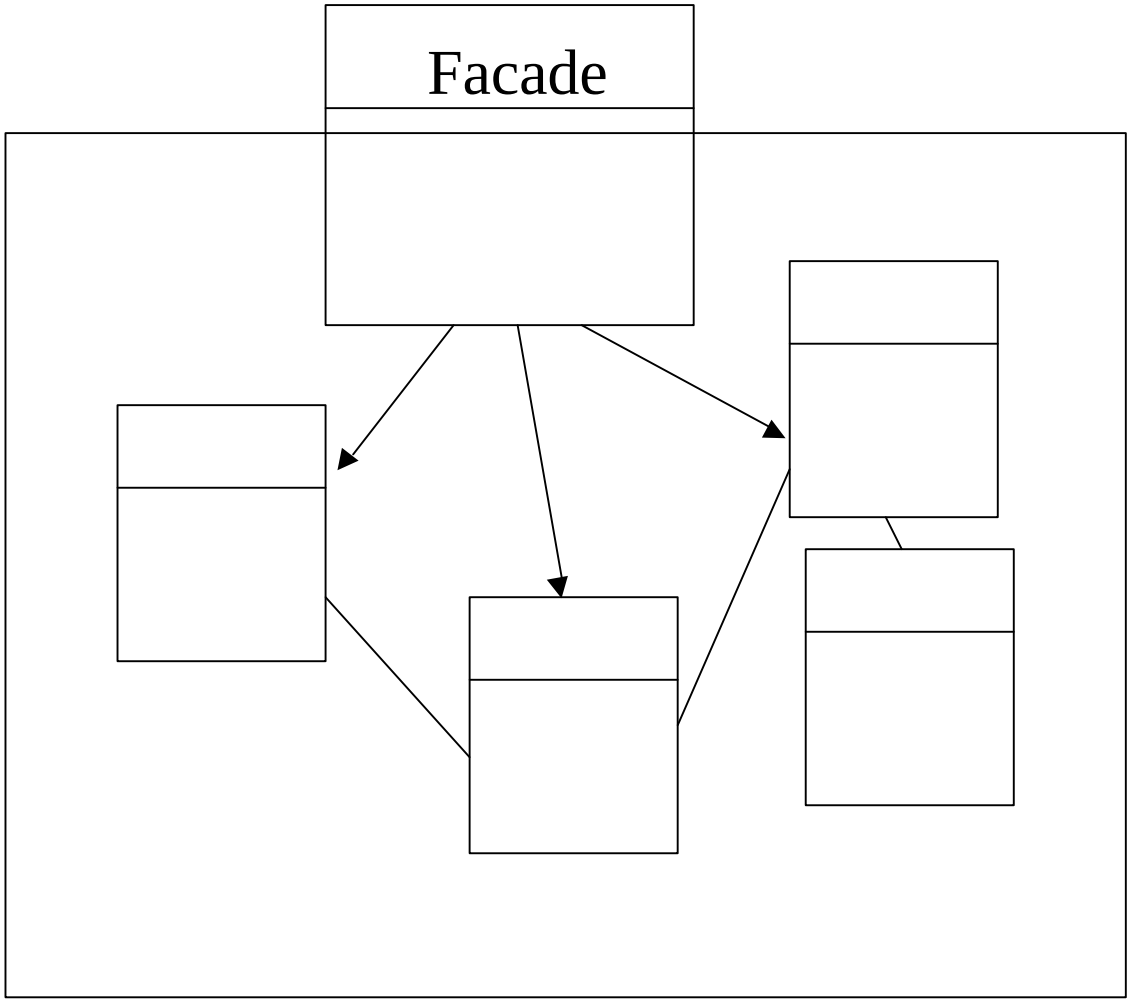
Adaptador



Bridge



Facade



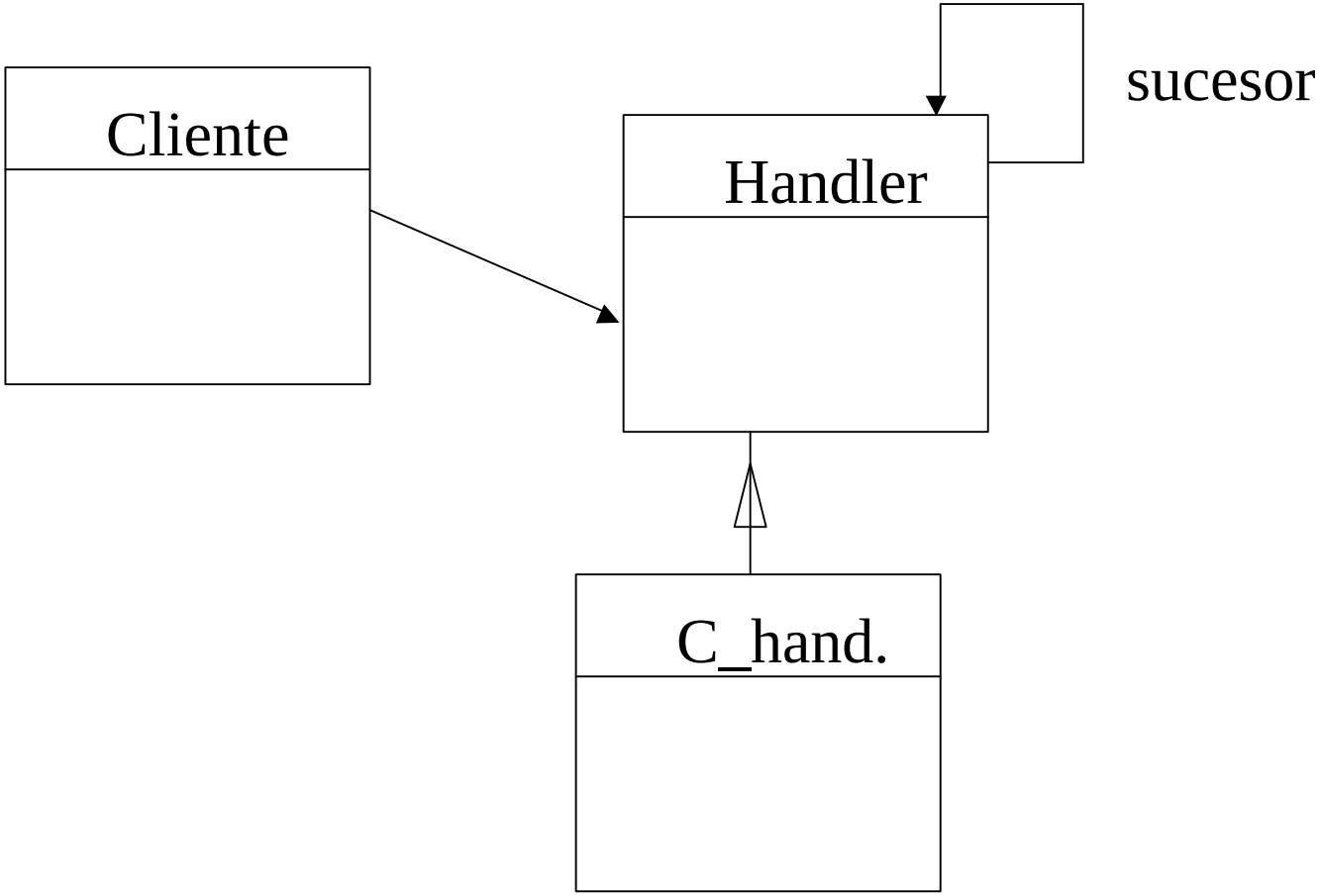
Proxy

- Separar la creación y utilización de un objeto (o conjunto de objetos) mediante un representante.
  - *Virtual proxy*: Creación costosa bajo demanda.
  - *Cache Proxy:* Almacenar objetos temporales
  - *Remote proxy*: Representante para Obj. Remoto.
  - *Protection proxy*: Controlar el acceso a objetos compartidos.

Patrones de comportamiento

- Tiene que ver con la dimensión temporal
- Estudia la relaciones entre llamadas entre los diferentes objetos.
- Incide en facilidades para tiempo de ejecución.

Cadena de responsabilidades

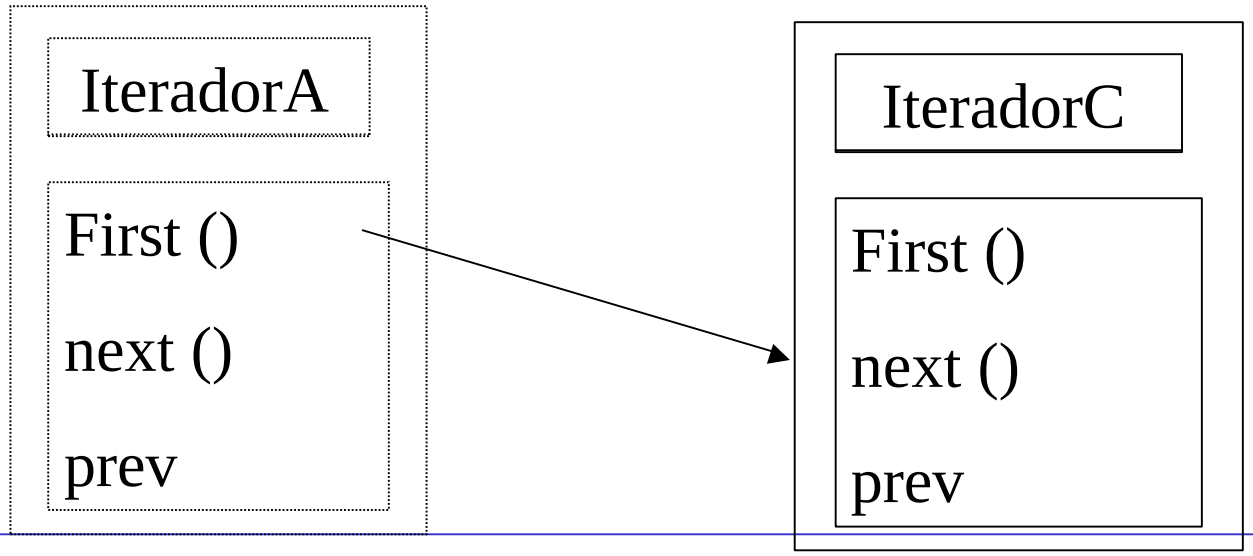


Lenguaje de comandos

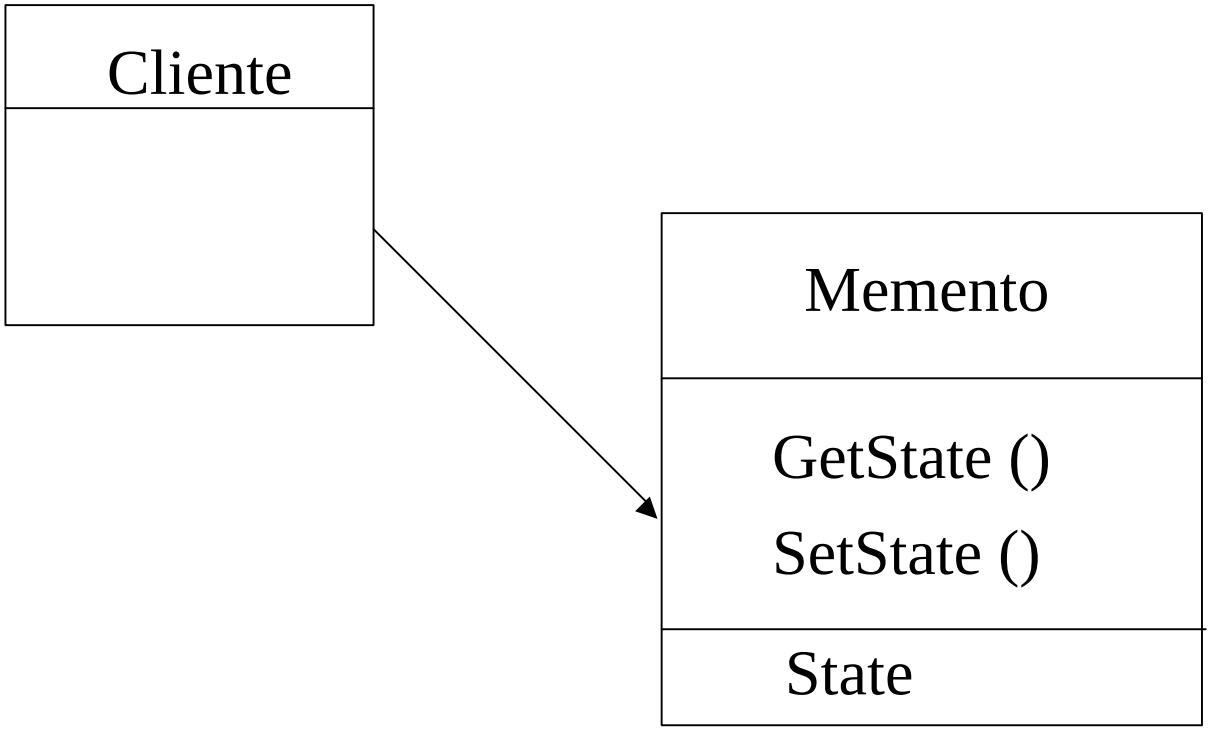
- Separar partes por un lenguaje de comandos
- Implementar un *parser* y un lenguaje simple
- Las comunicaciones entre las partes por *strings* que representan comandos del lenguajes.

Iterador

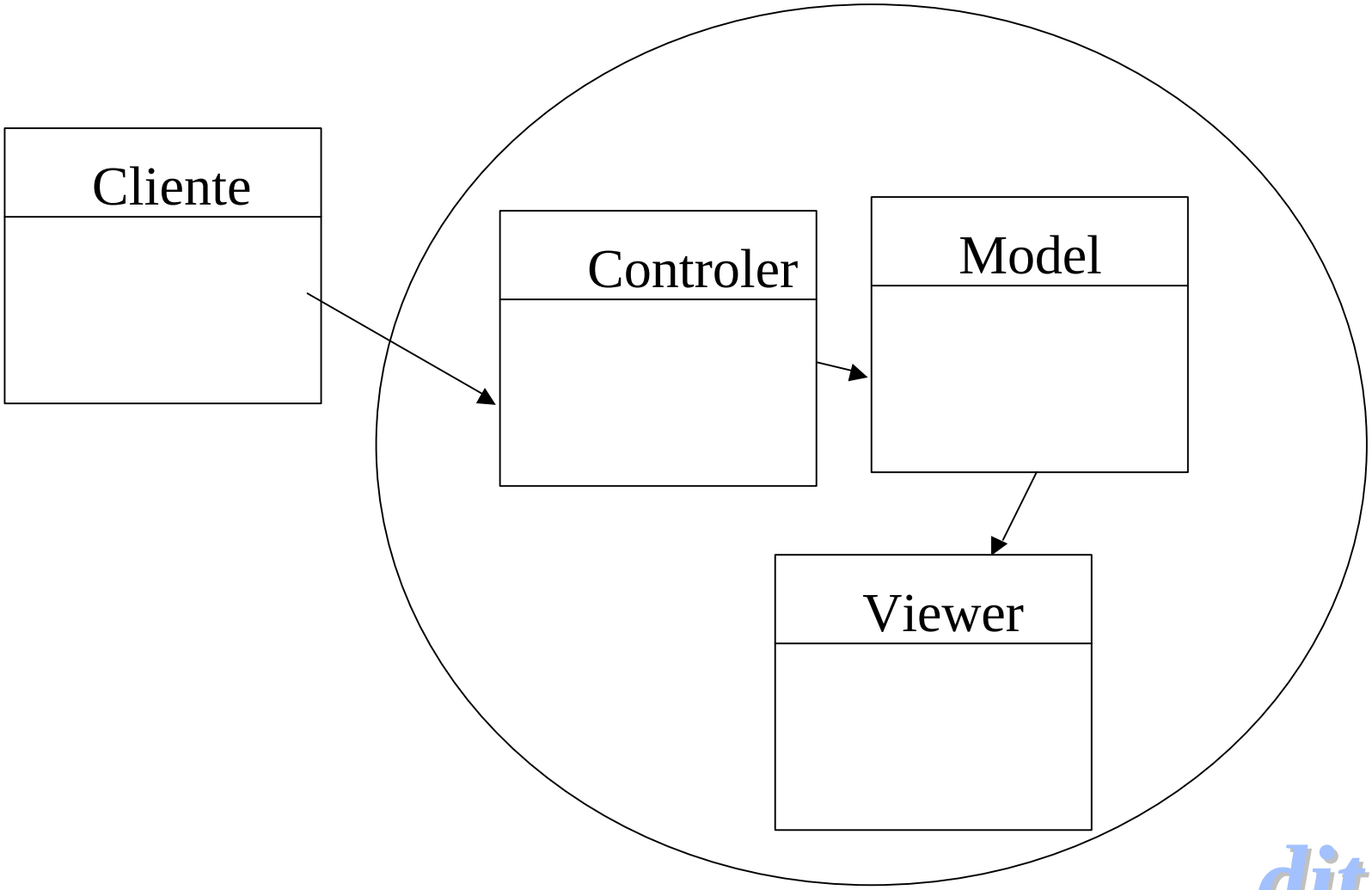
- Permitir acceso secuencial a los elementos de una estructura.
- Preservando la estructura interna.



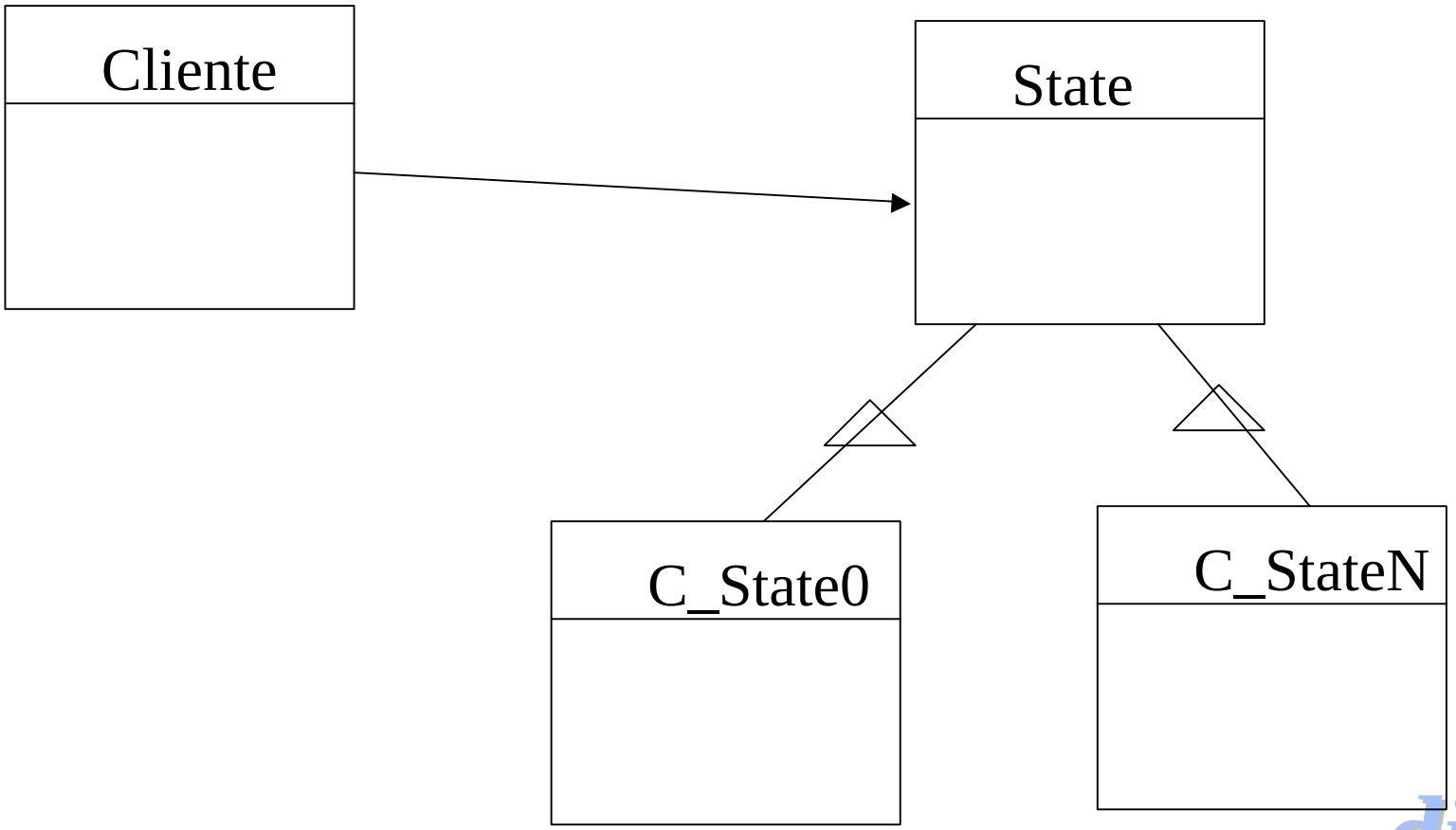
Memento



Observador



State



Problemas

- Las soluciones no son evidentes.
- Muchos patrones, a no se ser que estés inmerso en el problema, no son comprensibles.
- No son la “bala de plata” solo una ayuda más.

- Si conoces la solución ya no sirve :-)
- Efecto Aja (Zen).

URLs

- Patterns discussion FAQ:
  - http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html
- Patterns home page:
  - http://st-ww.cs.uiuc.edu/users/patterns/patterns.html
- Portland Pattern Repository
  - http://www.c2.com/ppr
- Communication systems patterns
  - http://www.agcs.com/patterns

Reusabilidad

- Dificultad y complejidad de clase para ser utilizada en cualquier caso crece exponencialmente.
- Construir con objetos es como colocar un ladrillo sobre otro.
- Necesidad de ladrillos útiles.
  - (problemas de lenguajes tipo C++).

Entornos

- Biblioteca (library)
  - Ladrillos para construir una casa.
- Cajas de herramienta (toolkits)
  - Soluciones prefabricadas para pegar.
- Entornos (frameworks)
  - Conjunto de clases Cooperantes que son reutilizables en un diseño de un cierto campo.
  - Principio Hollywood: (no nos llame, nosotros le llamaremos).

Antipatrones

- No solo identificar soluciones.
- También identificar trampas en el diseño

  “.. Animal que tropieza 2 veces .. “

- Esa solución que se le ocurre a todo el mundo y siempre caemos en ella.

- Tan importantes como los patrones.

Java

- Diseñadores basados en patrones.
- Casi todos los patrones están en las clases base de Java.

- Si se conocen los patrones: fácil comprender las decisiones de diseño.
- Necesidad de conocer limitaciones para extender las clases base.

URLs

- Patterns discussion FAQ:
  - http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html
- Patterns home page:
  - http://st-ww.cs.uiuc.edu/users/patterns/patterns.html
- Portland Pattern Repository
  - http://www.c2.com/ppr
- Communication systems patterns
  - http://www.agcs.com/patterns