Escuela de Sistemas y Tecnologías

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2017 – Materia: **Diseño de Aplicaciones Web**

TEMA: Patrones de Diseño

Plantel y Contactos

- **Bedelía:**
  - Mail: bedeliasistemas@bios.edu.uy

- **Encargado de Sucursal:**
  - Pablo Castaño
  - Mail: pablocasta@bios.edu.uy

Recursos

- **Recursos Imprescindibles:**
  - Sitio Web de material
    (comunicarse con Bedelía por usuario/contraseña).
  - Transparencias del Curso.
  - Contar con el software necesario

Agenda

- Introducción
- Fachada
- Fabrica (*Factory*)
- Singleton

Introducción

Definición (1)

- ¿Qué es un **patrón**?
  - *"Cada patrón describe un problema que ocurre una y otra vez, y luego describe una solución, de tal forma que Ud. puede utilizar esa solución un millón de veces, pero nunca haciéndolo de la misma forma."* [**Christopher Alexander,** http://es.wikipedia.org/wiki/Christopher_Alexander]
  - ¡Esta cita hace referencia a patrones en arquitectura urbana (edificios) no a patrones de software!

Definición (2)

- ¿Qué es un **patrón de diseño**?
  - Es aplicar esa misma idea para diseñar software, en términos de objetos, clases e interfaces.
  - Es una descripción de objetos y clases comunicándose entre sí, que se personalizan para resolver un problema general del diseño en un contexto particular.
  - La descripción de la solución al problema general se realiza en términos de **estructura** (clases, interfaces, relaciones) y **comportamiento** (interacciones entre objetos).
  - Típicamente contiene (al menos) 4 ítems: **nombre**, **problema**, **solución** y **consecuencias**

Nombre

- Se utiliza para aumentar el vocabulario del diseñador.
- Sólo con el nombre, se está haciendo referencia también al problema, la solución y las consecuencias.
- Esto permite diseñar a un mayor nivel de abstracción (simplemente diciendo, por ejemplo: *"¿Por qué no utilizar State?"*).

Problema

- El problema describe cuando aplicar el patrón.
- Explica el problema y su contexto.
- El problema puede ser de bajo nivel (ej: cómo representar algoritmos como objetos) o de más alto nivel (ej: cómo acceder a un subsistema).
- En ocasiones el problema presenta condiciones que deben ser cumplidas antes de aplicar el patrón.
- Entender el problema es tan importante como entender la solución, pues una buena solución a un problema incorrecto no sirve de nada.

Solución

- Describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones.
- Se divide en **estructura** y **comportamiento**.
- La solución no describe un diseño particular y concreto, ya que un patrón es un *template* que puede ser aplicado en diferentes situaciones (todas comprendidas dentro del problema).
- El diseñador debe, a partir de su problema particular y cuando éste encaje dentro del problema general planteado por el patrón, tomar la solución general del patrón y personalizarla a su solución particular…

Consecuencias

- Presenta el resultado y los *trade-off* de aplicar el patrón.
- Toda solución (incluso las buenas) tienen un "costo".
- Dentro de las consecuencias también pueden haber comentarios sobre lenguajes de programación o detalles de implementación, más allá del propio diseño.
- Se deben tener en cuenta las consecuencias de un patrón al momento de aplicarlo, particularmente si el "costo" de aplicarlo es demasiado alto frente a las ventajas de haberlo aplicado.

Clasificación(1)

- Los patrones de diseño se pueden clasificar según su propósito:
  - **Patrones de Creación**: les concierne el proceso de creación de objetos.
  - **Patrones de Estructura**: les concierne la composición de clases y objetos.
  - **Patrones de Comportamiento**: les concierne la distribución de responsabilidades y detallan de qué forma interactuarán clases y objetos.

Clasificación(2)

- Los patrones de diseño también se pueden clasificar según su alcance:
  - **Patrones que aplican principalmente a Clases**: se centran en las relaciones entre clases y subclases, típicamente mediante herencia, por lo que son relaciones fijas (se fijan en tiempo de diseño).
  - **Patrones que aplican principalmente a Objetos:** se centran en las relaciones entre objetos, las cuales pueden cambiar dinámicamente (en tiempo de ejecución).

Tomar En Cuenta

- La solución que un patrón presenta contiene aspectos tanto estructurales como de comportamiento.
- Particularmente no se debe olvidar (o despreciar) el comportamiento sugerido por la solución, pues éste indica cómo se utiliza la estructura.
- Esto resulta especialmente importante pues muchos patrones presentan estructuras similares en sus soluciones, lo cual, sin un estudio de sus comportamientos, llevará a la confusión entre ellos.
- Además, muchos patrones utilizan:
  - Herencia (o implementación de interfaz)
  - Delegación
  - Redefinición de operaciones
- Por lo que aún más las estructuras podrán parecer similares, siendo imprescindible estudiar el comportamiento sugerido para dichas estructuras.

Fachada

Definición (1)

- **Problema**: *Unificar múltiples servicios en un único servicio que cumpla el rol de ser el único punto de acceso.*
- **Aplicabilidad**: utilice Fachada como forma de brindar un único punto de acceso a un subsistema que contenga múltiples formas de acceso a sus servicios.
- **Solución**: definir una única clase que unifique los diferentes servicios brindados por el subsistema y asignarle la responsabilidad de colaborar con éste.

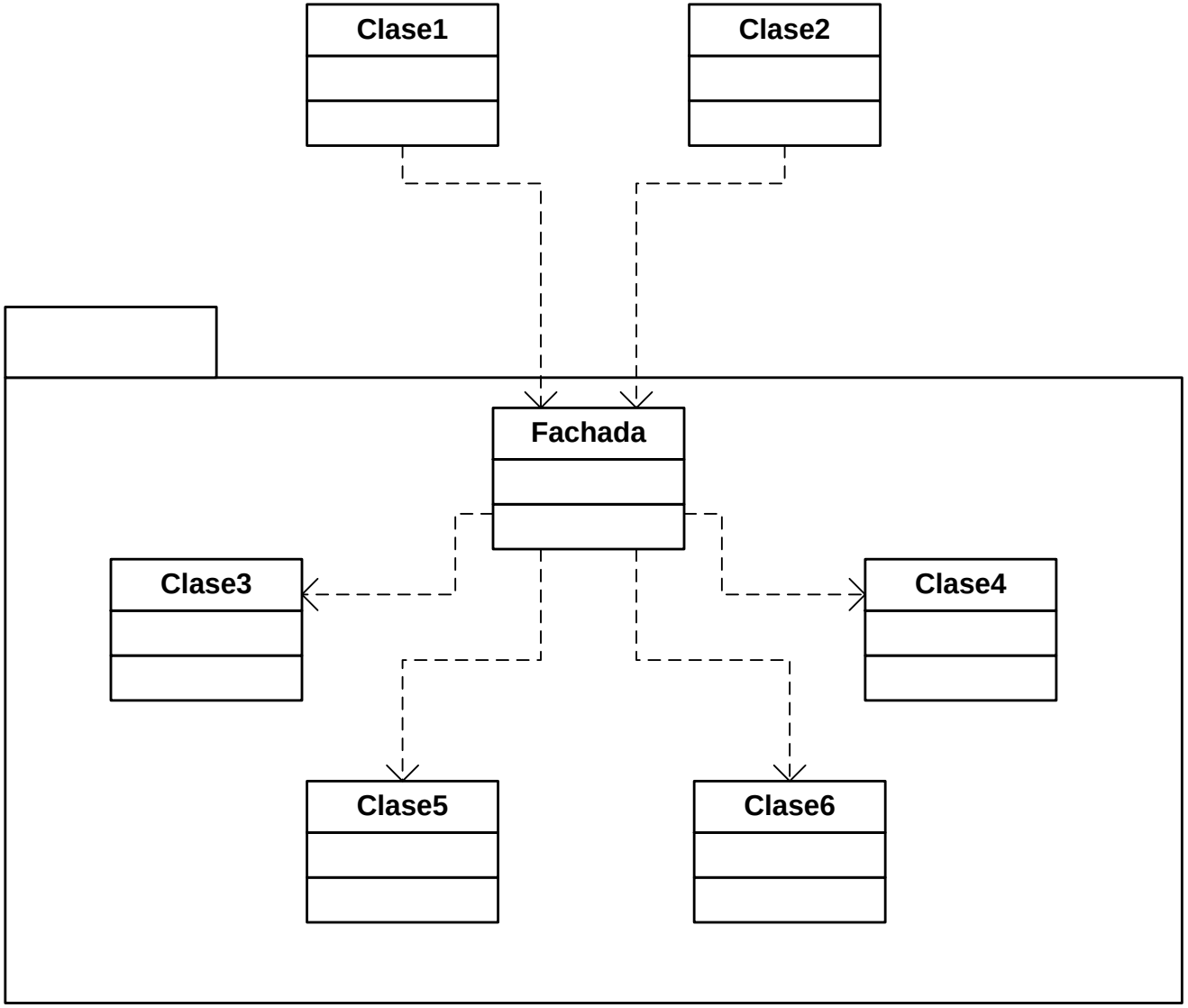
Definición (2)

- **Solución – Participantes**: la clase Fachada brindando un servicio unificado y las clases del subsistema brindando los servicios originalmente presentes.
- **Solución – Comportamiento**: las demás clases (código cliente) se comunica exclusivamente con la Fachada para acceder a los servicios del subsistema.

Definición (3)

- **Consecuencias**:
  - Se rompe la dependencia entre las clases clientes y las clases del subsistema.
  - Se incorpora una nueva clase en la solución.
  - La clase Fachada es altamente acoplada con las clases del subsistema (aunque no necesariamente con todas).
  - Se agrega un nuevo nivel de indirección.

Estructura



clases cliente

subsistema

Implementación Practica

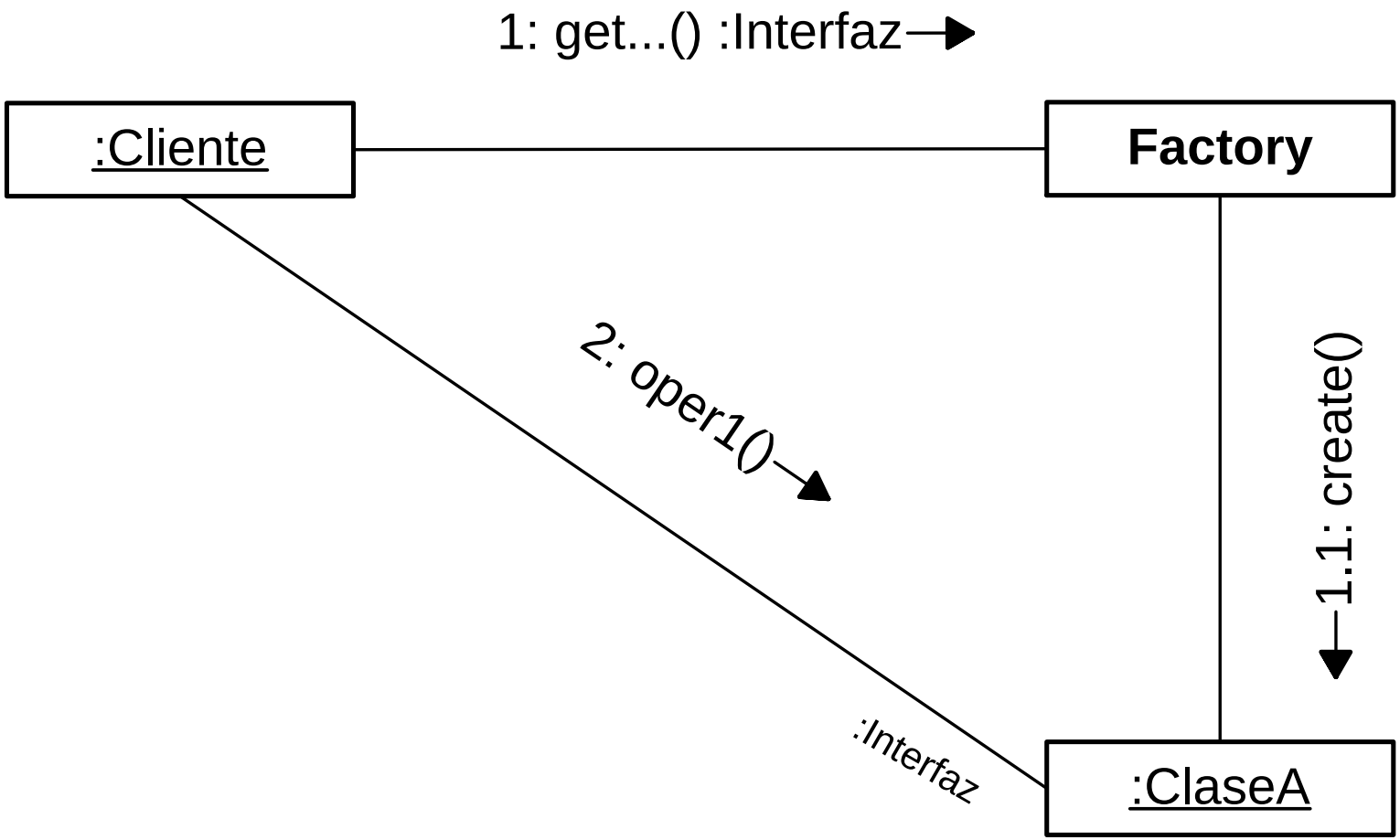
- Los formularios del componente de Interfaz grafica dialogan solamente con la clase fachada del componente de Lógica de negocio.
- La fachada del componente de Lógica puede implementar métodos de clase o de instancia.
- Las clases de trabajo de dicho componente, dialogan solamente con la clase fachada del componente de Acceso a Datos.
- Se utilizan objetos de negocio (Entidades Compartidas) para el intercambio de información.

Factory

Definición (1)

- **Problema**: *Proveer un servicio de creación de objetos sin especificar su clase concreta.*
- **Aplicabilidad**: utilice Factory para solicitar la creación de objetos sin especificar ni conocer el tipo real de éstos.
- **Solución – Participantes**: la clase Factory, las otras clases concretas cuyas instancias son creadas por la fábrica y generalmente una interfaz que permite romper la dependencia.

Definición (2)



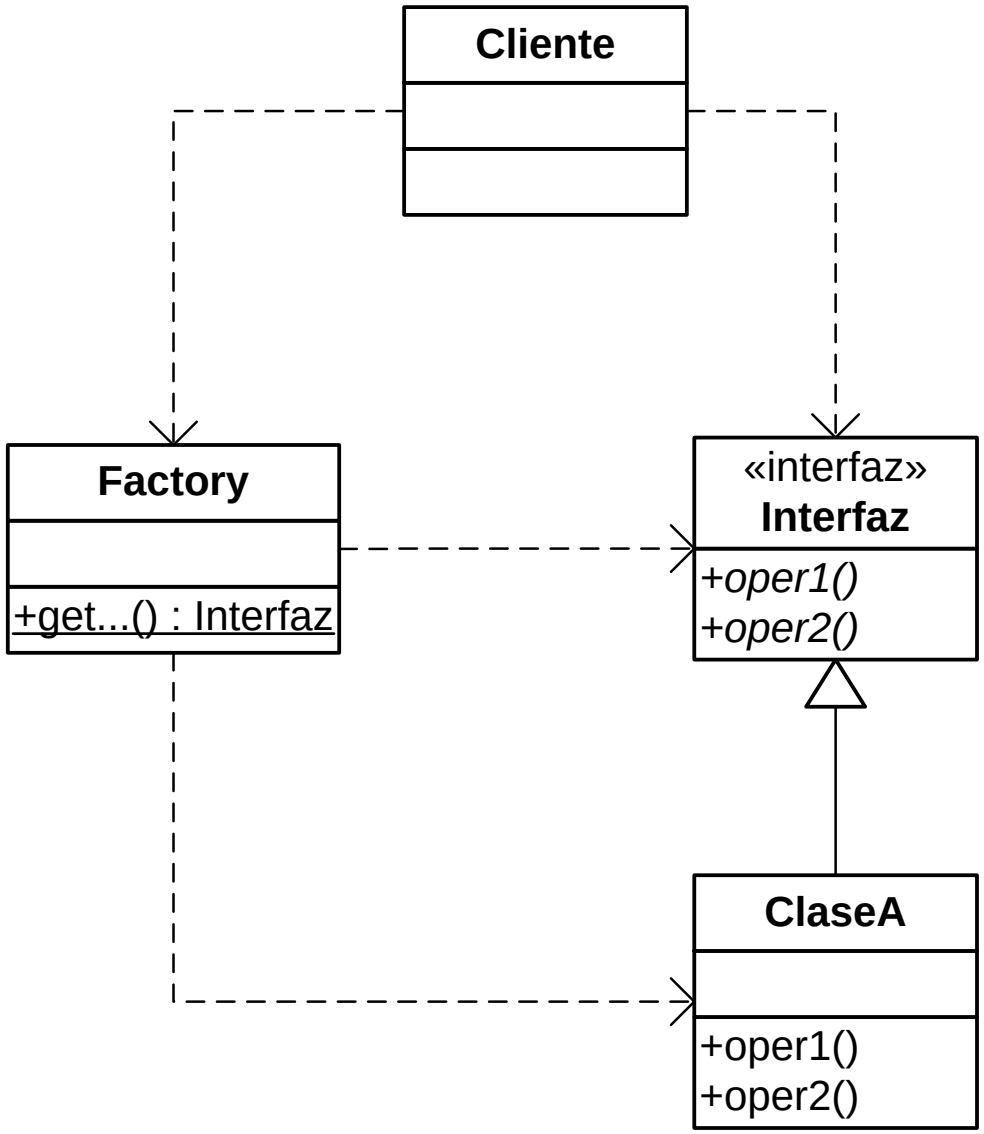
Definición (3)

- **Solución – Comportamiento (cont.)**:
  1. El código cliente solicita a la Factory la creación de un objeto de tipo Interfaz (es decir de cualquier clase que implemente la interfaz).
  2. La Factory obtiene una instancia del tipo solicitado y la retorna.
  3. El código cliente utiliza al objeto de tipo ClaseX sin conocer su tipo (pues lo "ve" como de tipo Intefaz) por lo que no queda acoplado a éste.

Definición (4)

- **Consecuencias**:
  - Rompe la dependencia entre las clases clientes y las clases concretas, permitiendo a las primeras crear y manipular instancias de las segundas pero sin conocer su tipo concreto.
  - Se incorpora una nueva clase en la solución.
  - Se necesitan interfaces de forma de no conocer las clases concretas que se instancian.

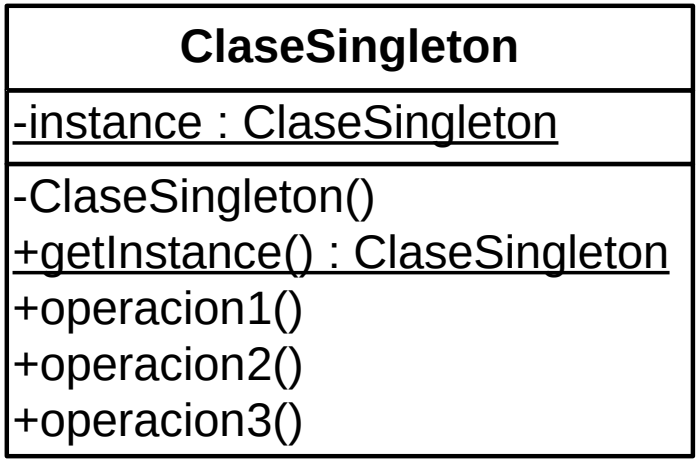
Estructura


Cliente
Factory
+get...() : Interfaz
«interfaz»
Interfaz
+oper1()
+oper2()
ClaseA
+oper1()
+oper2()

Singleton

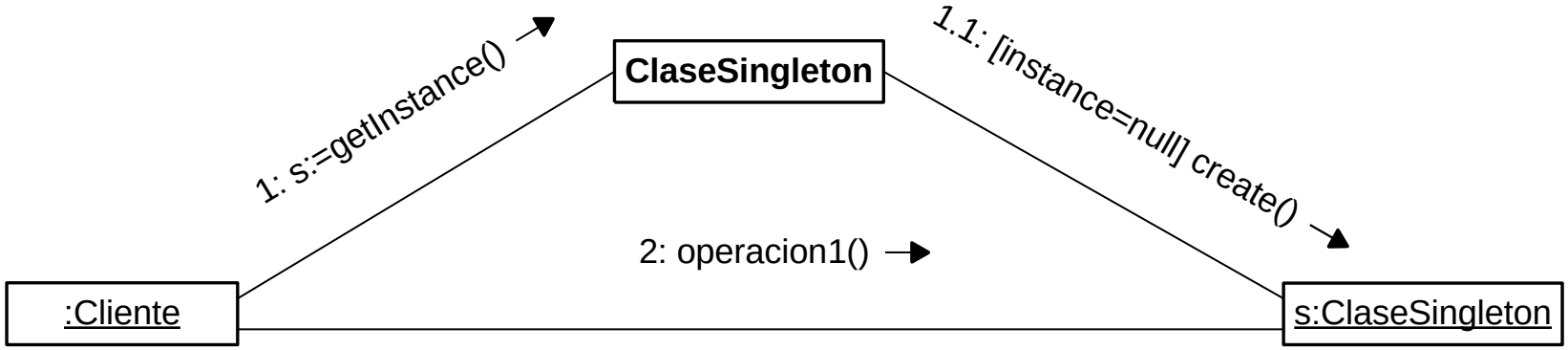
Definición (1)

- **Problema**: *¿Cómo asegurar que una clase tenga una sola instancia accesible globalmente?*
- **Aplicabilidad**: utilice Singleton cuando una clase deba tener una única instancia globalmente accesible para cualquier cliente.
- **Solución – Estructura**:



Definición (2)

- **Solución – Participantes**: el patrón Singleton posee una única clase participante: la propia clase Singleton que se encarga de crear la instancia únicamente cuando ésta no haya sido previamente creada.
- **Solución – Comportamiento**:



Definición (3)

- **Consecuencias**:
  - Permite el acceso controlado a una instancia.
  - Fácilmente modificable para permitir un conjunto de instancias y no sólo una.
  - Frente al uso de métodos de clase, el patrón Singleton sólo crea la instancia cuando ésta es utilizada por clientes, no desperdiciando recursos si no son necesarios, y permite fácil manejo de varias instancias (punto anterior).

Referencias

- "*Design Patterns*" de **E. Gamma, R. Helm, R. Johnson** & **J. Vlissides**
- "*UML y Patrones*" 1ª Edición de **Craig Larman** (Capítulo 35)
- "*UML y Patrones*" 2ª Edición de **Craig Larman** (Capítulo 23)