UNIVERSIDAD AUTONÓMA DE NUEVO LEÓN
FCFM

UNIDAD DE APRENDIZAJE: LABORATORIO DE DISEÑO ORIENTADO A OBJETOS.

PROFESOR. MIGUEL ÁNGEL SALAZAR S.

ALUMNA. VALERIA MARTÍNEZ DE LA ROSA

MATRICULA. 1678575

Grupo: 007

San Nicolás de los garza, Nuevo león a 14 de Marzo de 2017

PATRONES DE DISEÑO

¿Qué es? Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño.

En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

¿Para qué sirve? Son modelos muestra que sirven como guía para que los programadores trabajen sobre ellos.

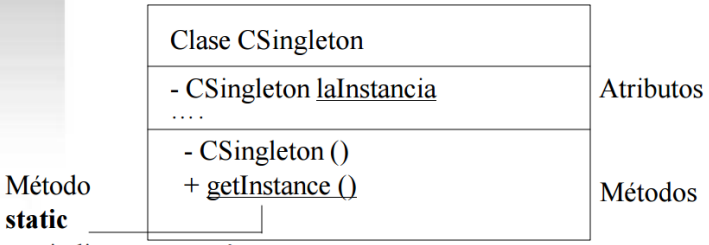
Menciona 5 ejemplos (nombre, y para qué sirve)

- Patrones de creación. Estos patrones se utilizan cuando debemos crear objetos pero debemos tomar decisiones dinámicamente en el proceso de creación. Para ésto lo que hacemos es abstraer el proceso de creación de los objetos para realizar la decisión de qué objetos crear o cómo crearlos para el momento en que se tenga que hacer. Patrones de creación son: *Abstract Factory*, *Builder*, *Factory Method*, *Object Pool*, *Prototype* y *Singleton*.
- Patrones estructurales. Nos describen como utilizar estructuras de datos complejas a partir de elementos más simples. Sirven para crear las interconexiones entre los distintos objetos y que estas relaciones no se vean afectadas por cambios en los requisitos del programa. Algunos ejemplos de patrones estructurales son: *Adapter*, *Bridge*, *Decorator*, *Facade*, *Flyweight* y *Proxy*.
- Patrones de comportamiento. Fundamentalmente especifican el comportamiento entre objetos de nuestro programa. Hay varios: *Chain of Responsability*, *Command*, *Interpreter*, *Iterator*, *Mediator*, *Memento* (o *Snap shot*), *Observer*, *State*, *Strategy*, *Template Method* y *Visitor*.

1. Singleton: Asegurar que una clase tiene una sola instancia y proporcionar un punto de acceso global a ella.

– El constructor de la clase DEBE SER PRIVADO

– Se proporciona un método ESTÁTICO en la clase que devuelve LA ÚNICA INSTANCIA DE LA CLASE: getInstance()

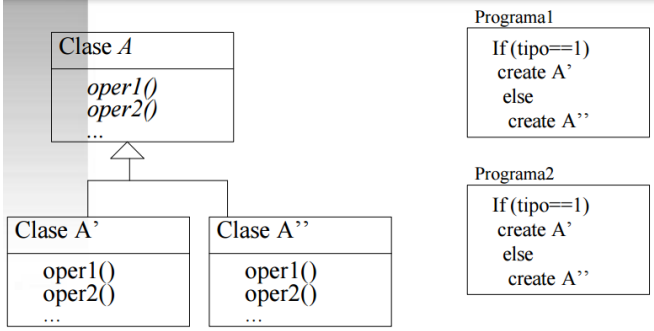


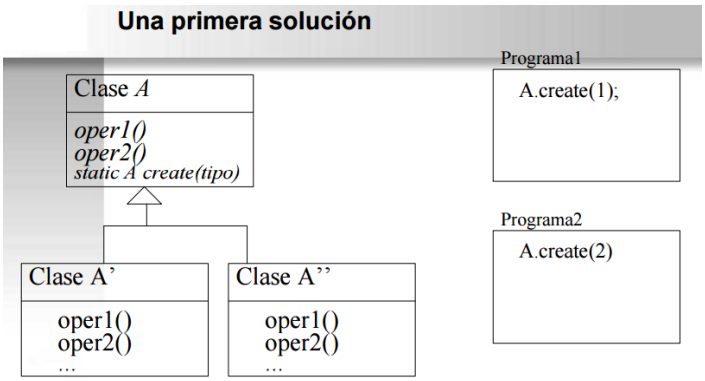
EJEMPLO DE PATRÓN SINGLETON

```
public final class Facultad implements Serializable {
      private Vector listaProfesores;
      private Vector listaEstudiantes;
      private Vector listaAsigs;
      private Vector listaMatrs;
      private static Facultad laFacultad=new Facultad();
      private Facultad()      // EL CONSTRUCTOR ES PRIVADO
      {  listaProfesores = new Vector();  // Sólo hay UNA instancia
         listaEstudiantes= new Vector();  // y se guarda en laFacultad
         listaAsigs = new Vector();
         listaMatrs = new Vector(); }
   public static Facultad getInstance() {return laFacultad;}
   public Vector obtListaProfesores()
         {return listaProfesores;}
   public void anadirProfesor(Profesor p)
         {listaProfesores.addElement(p); }
```

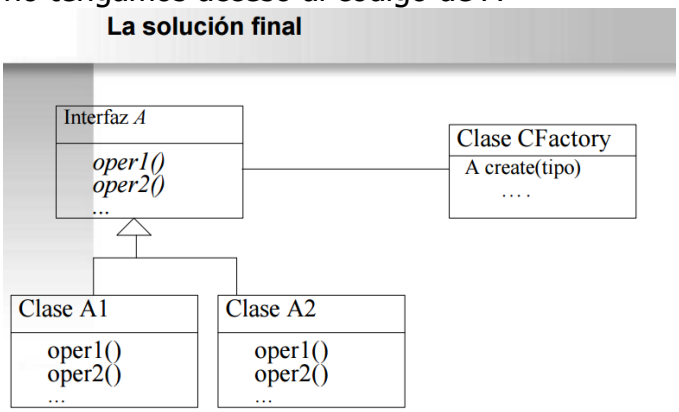
2. FACTORY METHOD: Separar la clase que crea los objetos, de la jerarquía de objetos a instanciar
   ¿Qué pasa si queremos añadir A'''?

El problema



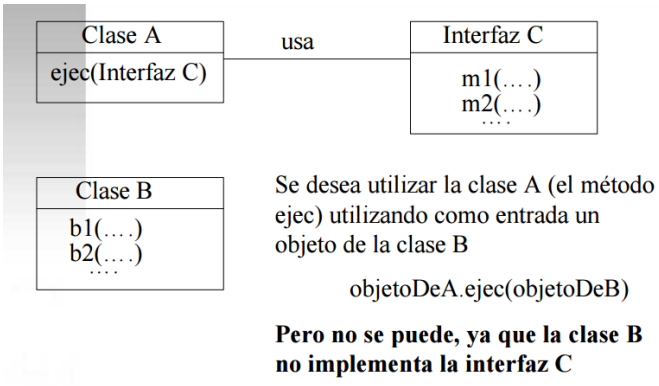


Solución: Hay que recompilar todas las clases que heredan de A Puede que no tengamos acceso al código de A

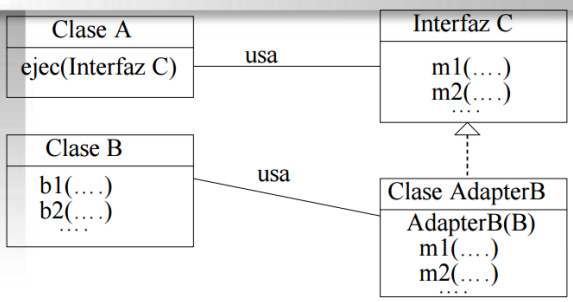


Solución: Separar el creador de las instancias de la propia clase las instancias se crean en una clase CFactory
class Aplicación { ... A newA(...) { A miA; CFactory fact= new CFactory(); ... miA = fact.create("TIPO 2"); ... return miA; } • No se crean las instancias directamente en Aplicación • Si se quisiera añadir un nuevo A A'", NO NECESARIAMENTE habría que modificar la clase Aplicación. Los tipos que CFactory puede crear los puede devolver un método y Aplicación trabajar con él.

3. ADAPTER: Convertir la interfaz de una clase en otra interfaz esperada por los clientes. Permite que clases con interfaces incompatibles se comuniquen
Problema:



La solución



Solución: construir una clase Adaptadora de B que implemente la interfaz C. Al implementarla, usa un objeto de B y sus métodos
Para utilizar la clase A:

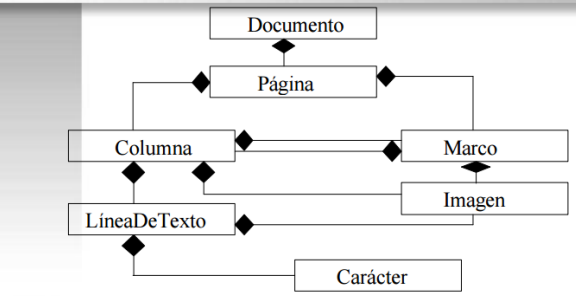
objetoDeAdapterB =NEW AdapterB(objetoDeB)
objetoDeA.ejec(objetoDeAdapterB)

38

Convierte la interfaz de una clase en otra interfaz que los clientes esperan. Permite que clases con interfaces incompatibles puedan ser utilizadas conjuntamente.

4. COMPOSITE: Componer objetos en jerarquías todo-parte y permitir a los clientes tratar objetos simples y compuestos de manera uniforme.

El problema: La escalabilidad



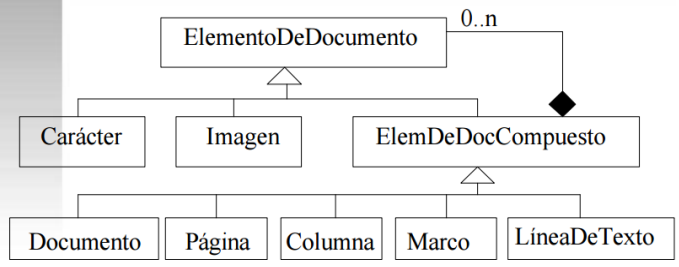
Un **documento** está formado por varias **páginas**, las cuales están formadas por **columnas** que contienen **líneas de texto**, formadas por **caracteres**.

Las **columnas** y **páginas** pueden contener **marcos**. Los **marcos** pueden contener **columnas**.

Las **columnas**, **marcos** y **líneas de texto** pueden contener **imágenes**.

45

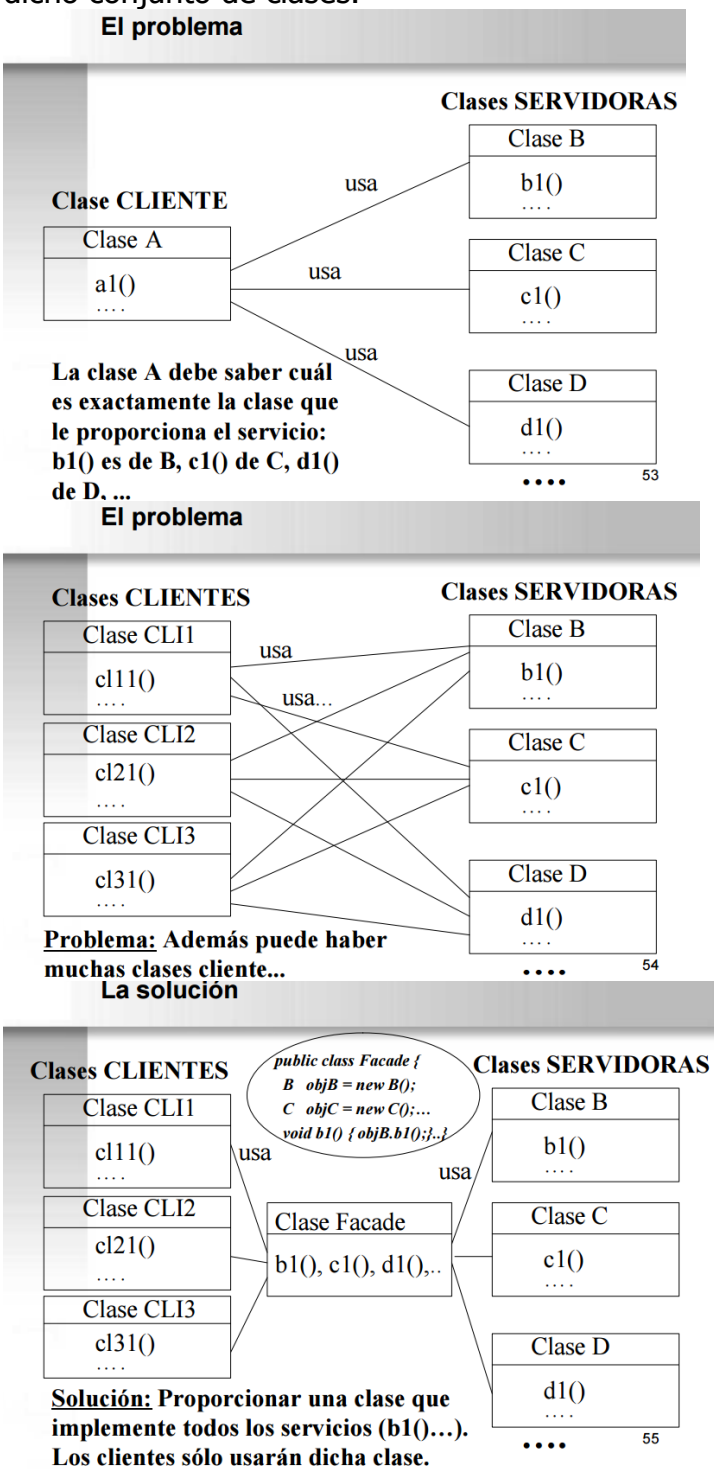
La solución



Un **documento** está formado por varias **páginas**, las cuales están formadas por **columnas** que contienen **líneas de texto**, formadas por **caracteres**.
Las **columnas** y **páginas** pueden contener **marcos**. Los **marcos** pueden contener **columnas**.
Las **columnas**, **marcos** y **líneas de texto** pueden contener **imágenes**.

46

5. FACADE: El patrón FACADE simplifica el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto de clases.



¿Cómo se documentan? Los patrones de diseño se documentan utilizando plantillas formales con unos campos estandarizados. Existen varias plantillas

ampliamente aceptadas, aunque todas ellas deben al menos documentar las siguientes partes de un patrón:

- Nombre: describe el problema de diseño.
- El problema: describe cuándo aplicar el patrón.
- La solución: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.

La plantilla más utilizada es la plantilla GOF que consta de los siguientes campos:

- Nombre del patrón: Ayuda a recordar la esencia del patrón.
- Clasificación: Los patrones originalmente definidos por GOF se clasifican en tres categorías "Creacional", "Estructural", "Comportamiento".
- Propósito / Problema: ¿Qué problema aborda el patrón?
- También conocido como: Otros nombres comunes con los que es conocido el patrón.
- Motivación: Escenario que ilustra el problema.
- Aplicabilidad: Situaciones en las que el patrón puede ser utilizado.
- Estructura: Diagramas UML que representan las clases y objetos en el patrón.
- Participantes: Clases y/o objetos que participan en el patrón y responsabilidades de cada uno de ellos
- Colaboraciones: ¿Cómo trabajan en equipo los participantes para llevar a cabo sus responsabilidades?
- Consecuencias: ¿Cuáles son los beneficios y resultados de la aplicación del patrón?
- Implementación: Detalles a considerar en las implementaciones. Cuestiones específicas de cada lenguaje OO.
- Código de ejemplo
- Usos conocidos: Ejemplos del mundo real.
- Patrones relacionados: Comparación y discusión de patrones relacionados. Escenarios donde puede utilizarse en conjunción con otros patrones.