

Diseño de Sistemas

Unidad 3: Diseño con Objetos - parte 1

Martín Agüero

Pablo Sabatino

2019



UTN.BA

DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CÁTEDRA DISEÑO DE SISTEMAS

Unidad 3:

*Patrones de Diseño: Concepto de Patrón. Ventajas y Desventajas de su uso.
Patrones Creacionales. Patrones estructurales. Patrones de comportamiento.*



Patrones de Diseño

Ingeniería Mecánica

Problema: La energía de impacto vertical suele dañar la columna de los tripulantes de helicópteros.

Objetivo: Reducir la energía del impacto

Solución: Agregar bolsas de aire para absorber parte de la energía.

Patrones de Diseño

Ingeniería Mecánica



Desventaja: Las bolsas desplegadas no son aerodinámicas.

Patrones de Diseño

Ingeniería Mecánica

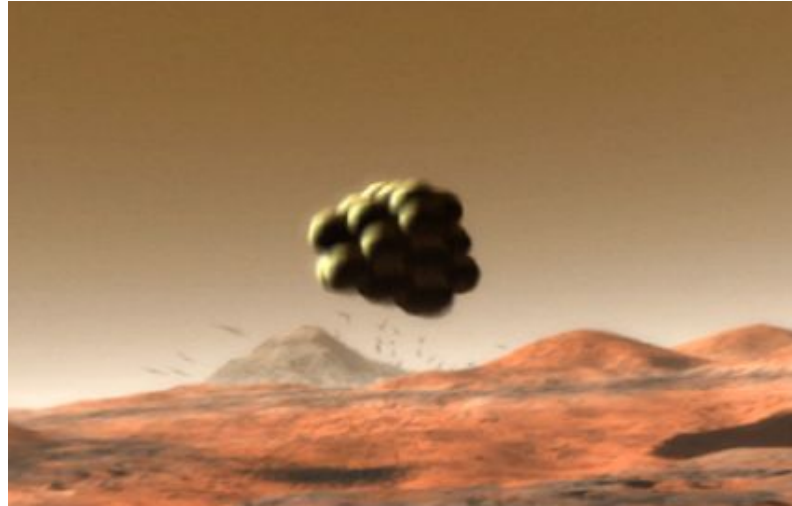
Problema: La energía de impacto dañaría el robot de exploración.

Objetivo: Reducir la energía del impacto. Retro propulsores son muy costosos.

Solución: Construir una coraza de bolsas infladas con un gas que amortigüe parte de la energía.

Patrones de Diseño

Ingeniería Mecánica



Desventaja: Las bolsas desplegadas no soportan el lanzamiento por cohete desde la tierra.

Patrones de Diseño

Ingeniería Mecánica

Problema: La energía generada por un choque frontal suele producir que la cabeza del conductor impacte en la columna de dirección.

Objetivo: Reducir la energía del impacto en la cabeza.

Solución: Air-bag

Patrones de Diseño

Ingeniería Mecánica



Desventaja: El air-bag desplegado no permite manejar.

Patrones de Diseño

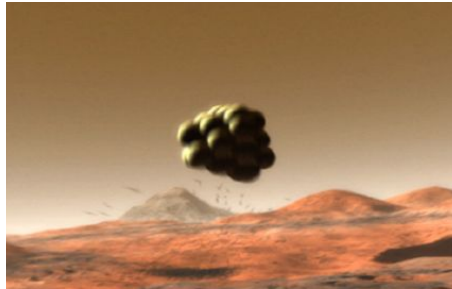
Ingeniería Mecánica

Patron: Air-Bag

Problema: Los ocupantes de un vehículo sufren severos daños ante un impacto.

Objetivo: Absorber la energía del impacto.

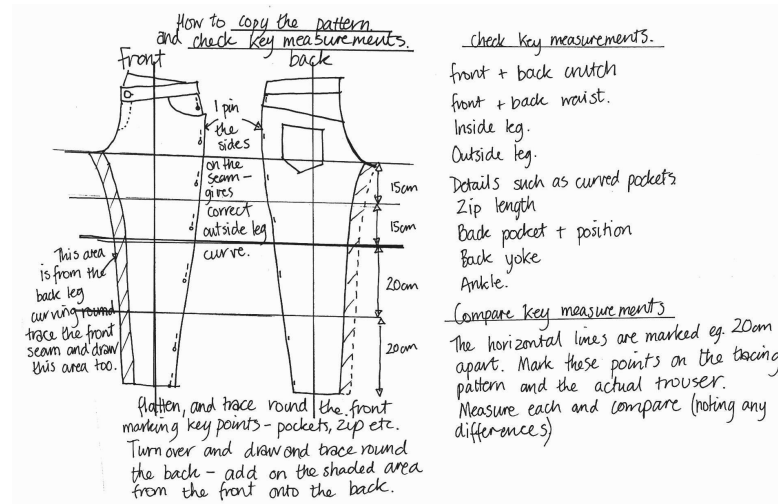
Solución: Utilizar contenedores con una sustancia comprimible (gas) para absorber la energía a niveles aceptables.



Desventaja: El contenedor de gas sólo es necesario durante el impacto.

Patrones de Diseño

Un patrón describe un **problema recurrente** en un determinado contexto y describe el **núcleo de la solución** a ese problema, de forma tal que pueda ser **reutilizada** infinitas veces en situaciones similares.



Patrones de Diseño

¿Qué es un patrón de diseño?

“Son descripciones de clases y objetos relacionados que están adaptados para resolver un problema de diseño general en un contexto determinado”.

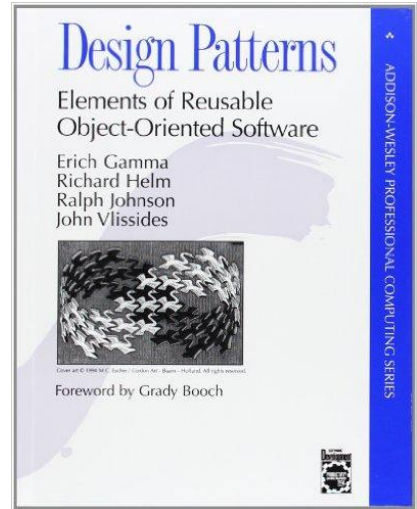
Erich Gamma, Richard Helm, John Vlissides y Ralph Johnson

Patrones de Diseño

En el año 1994 se publica “**Design Patterns: Elements of Reusable Object-Oriented Software**” (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides).



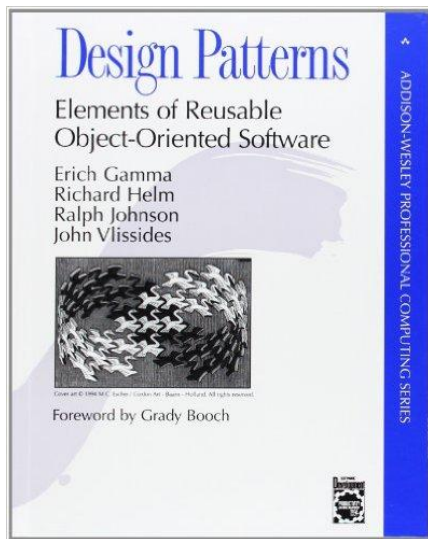
Fuente: <http://www.vincehuston.org/dp/>



El objetivo es reutilizar la experiencia de quienes ya se han encontrado con problemas similares y han encontrado una buena solución.

Patrones de Diseño

Catálogo de Patrones de Diseño



Describe patrones creacionales, estructurales y de comportamiento a través de las siguientes secciones:

- Propósito
- Motivación
- Participantes
- Colaboraciones
- Consecuencias
- Implementación
- Código de ejemplo
- Usos conocidos
- Patrones relacionados

Patrones de Diseño

Partes esenciales de la descripción de un patrón

Nombre: Comunica el objetivo del patrón en una o dos palabras. Aumenta el vocabulario sobre diseño.

Problema: Describe el problema que el patrón soluciona y su contexto. Indica cuándo se aplica el patrón.

Solución: Indica cómo resolver el problema en términos de elementos, relaciones, responsabilidades y colaboraciones. La solución debe ser lo suficientemente abstracta para poder ser aplicada en diferentes situaciones.

Consecuencias: Indica los efectos de aplicar la solución. Son críticas al momento de evaluar distintas alternativas de diseño.

Patrones de Diseño

Patrones de Diseño (Gamma 1994)

Creacionales: Abstract Factory Builder Factory Method Prototype Singleton	Comportamiento: Chain of Responsibility Command Interpreter Iterator Mediator Memento Observer State Strategy <i>Template Method</i> Visitor	Estructurales: Adapter Bridge Composite Decorator Facade Flyweight Proxy
---	--	--

Patrones de Diseño

¿Qué es un patrón?

- Es una descripción:
 - Describe el problema en forma sencilla.
 - Describe el contexto en el que ocurre.
 - Describe los pasos a seguir.
 - Describe los puntos fuertes y débiles de la solución.
 - Describe otros patrones asociados.
- Son una forma de comunicar diseños.
 - Se definen con un alto nivel de abstracción.
 - Son independientes de los lenguajes de programación y de los detalles de implementación.
- Documentan experiencias de diseño existentes y bien probadas.
- Son una forma de documentar la arquitectura del software.

Patrones de Diseño

¿Qué describe y/o resuelve un patrón?

- Un problema recurrente que se da en situaciones específicas y ofrecen una solución.
- Describen esquemas de soluciones probadas.
- La mejor solución a un problema dado.

Resuelven problemas concretos y crean diseños más mantenibles, flexibles y reutilizables.

Patrones de Diseño

¿Porque usar patrones de diseño?

- No reinventamos soluciones a problemas conocidos.
- Re-utilizamos el conocimiento experto relativo a un diseño.
- Proveen un vocabulario común y comprensible. Mejora la comunicación y documentación.
- Mejora la calidad de la solución.
- Eleva el expertise y nivel del equipo de diseño y desarrollo de sw.
- Dan una perspectiva de alto nivel sobre el análisis y diseño.

Permiten la reutilización de la experiencia en Diseño.

Patrones de Diseño

¿Qué es un problema recurrente?

- Un conflicto o problema que ocurre en diferentes dominios (contextos).
- Puede ser reducido a un conjunto elemental de fuerzas.

Patrones de Diseño

¿Cómo se identifica la solución?

- Se observa cual es la solución que mejor reduce el conflicto.
- La solución debería ser implementada por alguien diferente al observador.
- También puede, en algunos casos, existir comprobaciones formales.

Patrones de Diseño

Beneficios en el diseño utilizando patrones

- Diseñadores con menor expertise pueden diseñar como expertos.
- Se diseña con menos errores debido al uso de diseños ya probados ampliamente en la industria.
- Los diseños probados son más fácilmente mantenibles.

Un patrón de software es un esquema genérico probado para solucionar un problema particular, el cual es recurrente dentro de cierto contexto. Este esquema se especifica describiendo sus componentes, con sus responsabilidades y relaciones.

Patrones de Diseño

¿Qué hacemos con los patrones?

Aceptarlos

Reconocerlos

Internalizarlos

Aplicarlos

Patrones de Diseño

Caso 1

Patrones de Diseño

Caso 1

Un sitio de comercio electrónico vende libros.

- Los libros poseen: Título, Autor, Edición, Precio
- Los libros a la venta pueden ser digitales o impresos.

El precio final de un libro digital es el precio de costo más un costo fijo por comisión; el precio final de un libro impreso tiene una comisión del 2% del costo y se agregan \$20 de gastos de envío.

Además el sitio administra sus clientes:

- Cuentan con un crédito para comprar libros.
 - ◆ Cuando un cliente compra un libro, se debita el precio de su crédito.
- Pueden hacer recargas de su crédito.
- Un cliente puede querer acceder a la lista de sus compras.

Patrones de Diseño

Caso 1

Cálculo de precio para los libros:

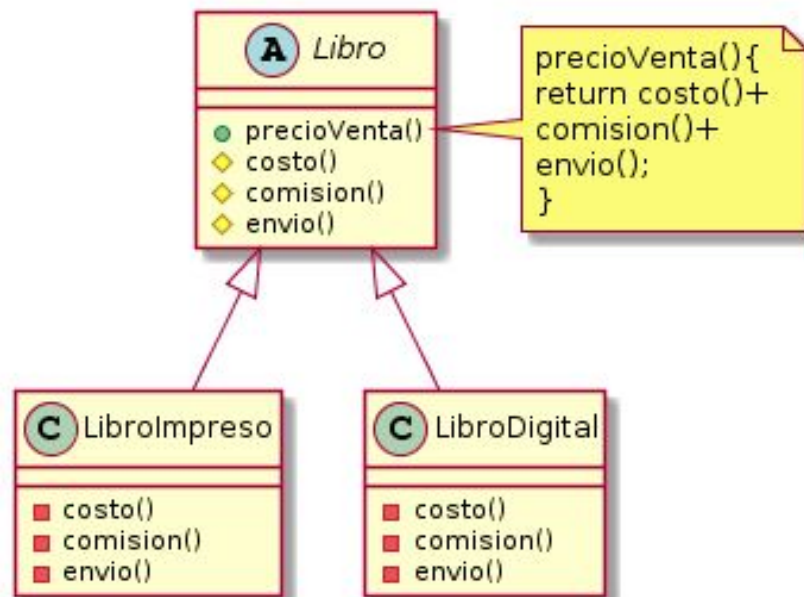
El precio final de un libro digital es el precio de costo más un costo fijo por comisión; el precio final de un libro impreso tiene una comisión del 2% del costo y se agregan \$20 de gastos de envío.

Preguntas:

- ¿Hay comportamiento en común?
- ¿Qué pasa se modifica algo en ese comportamiento común?
- ¿Hay alguna forma de factorizarlo al comportamiento común?
- ¿Qué pasa con el comportamiento distinto?

Patrones de Diseño

Caso 1

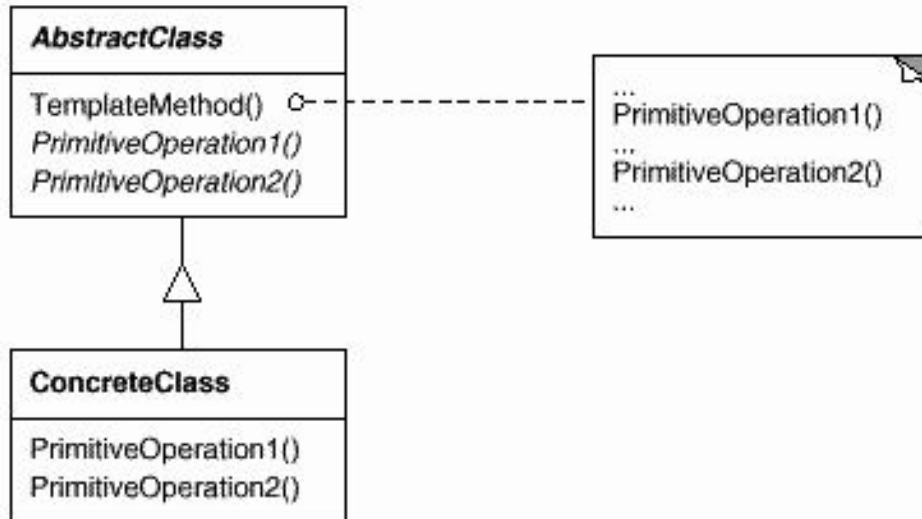


Patrones de Diseño

Template Method

Comportamiento

- Define el esqueleto del algoritmo de una operación o comportamiento común
- Delega en las subclases redefinir los pasos particulares del algoritmo



Patrones de Diseño

Template Method

Motivación:

- ❖ En el caso que sea necesario definir el invariante de un algoritmo y permitir que las subclasses modifiquen sus partes variables.
- ❖ En el caso que se encuentre comportamiento común en un conjunto de subclasses que pueda ser refactorizado en la superclase.
- ❖ Para brindar puntos de extensión a la clase en forma controlada (hooks).

Solución:

Definir una clase abstracta que implemente la estructura del algoritmo haciendo llamados a otros mensajes (template method).

Las subclasses concretas deben definir los métodos de los pasos del algoritmo. De esta forma el orden de los pasos queda fijo, pero cada clase puede realizar las variaciones de comportamiento necesarias en cada paso.

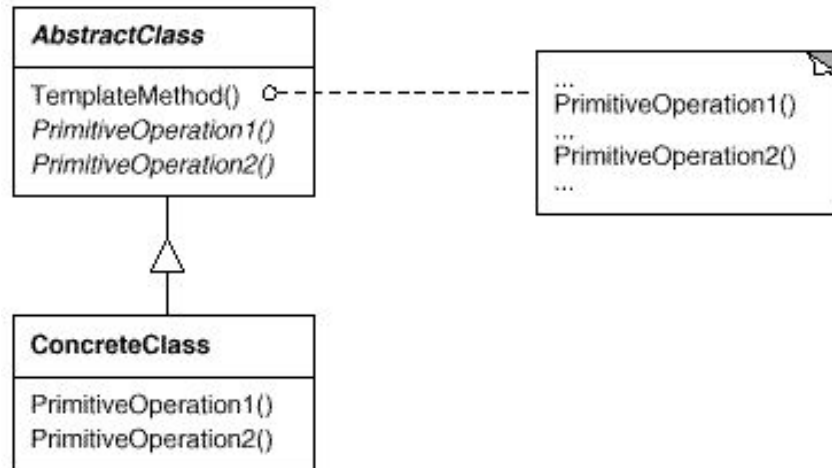
Patrones de Diseño

Template Method

Participantes:

AbstractClass: Define de forma abstracta las operaciones primitivas e implementa un template method con el esqueleto de una operación.

ConcreteClass: Implementa las operaciones primitivas.



Patrones de Diseño

Template Method

Consecuencias:

- Técnica fundamental de reuso: factorizar comportamiento común en la superclase, de manera que cambios en ese comportamiento se aplican automáticamente a todas las subclases.
- Inversión de control (la clase padre invoca a los mensajes de la subclase).

Existen varios tipos de mensajes que puede invocar un template method:

Mensajes concretos (en la clase abstracta o la subclase).

Mensajes abstractos que deben ser redefinidos por las subclases.

Métodos hook, con comportamiento por defecto que pueden ser redefinidos por las subclases.

Patrones de Diseño

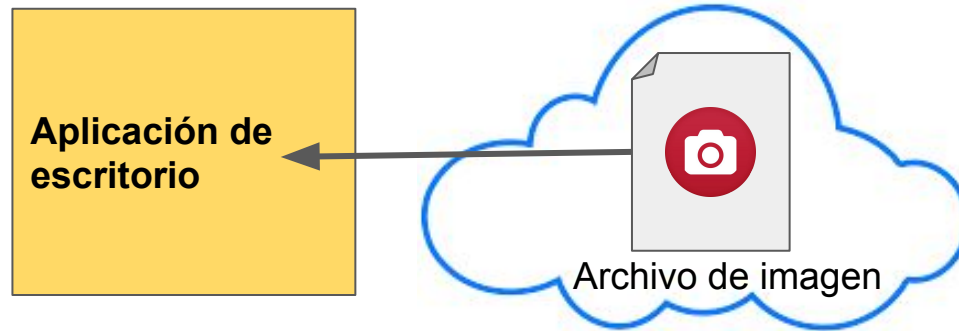
Caso 2

Patrones de Diseño

Caso 2

Estructural

- Se desea crear documentos que usen recursos externos.
- Esos recursos son archivos de gran tamaño.
- Se desea poder guardar el documento y el recurso de forma independiente.



Patrones de Diseño

Caso 2

Intención

- Manejar objetos remotos transparentemente.
- Agregar un nivel de indirección para controlar acceso distribuido o controlado.

Problema

- Manejar el acceso a la funcionalidad de un recurso remoto o un recursos de acceso costoso.

Solución

- Crear un objetos que represente al recurso en forma local.
- Proveer acceso local mientras oculta el acceso remoto.

Patrones de Diseño

Patrón Proxy

Propósito:

Proporciona un representante o sustituto de otro objeto para controlar el acceso a éste.

También conocido como:

Surrogate (Sustituto)

Motivación:

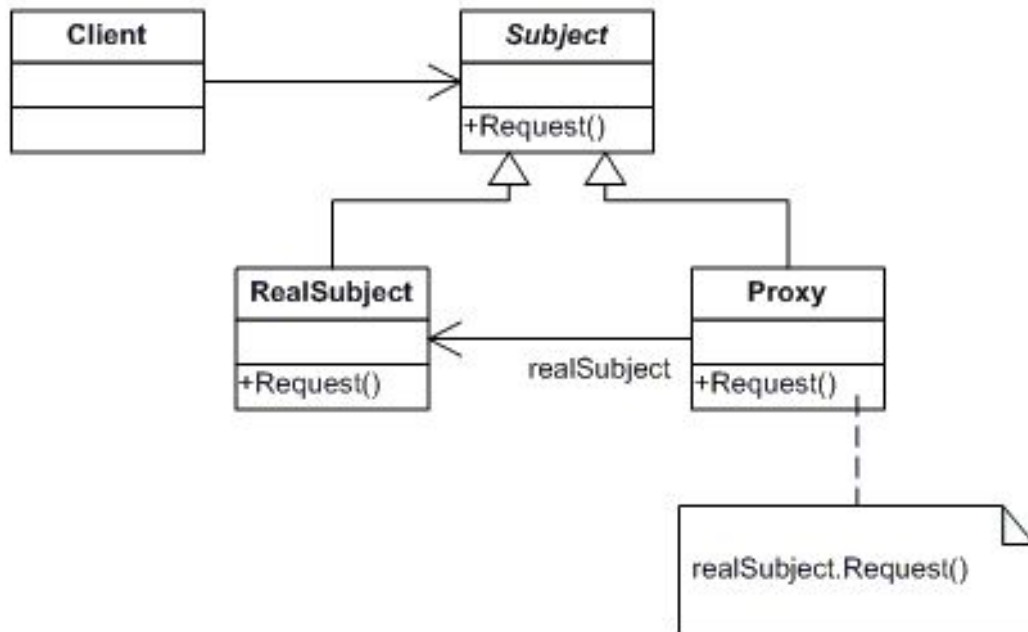
Una razón para controlar el acceso a un objeto es retrasar todo el costo de su creación e inicialización hasta que sea realmente necesario usarlo.

Una solución es utilizar otro objeto, un proxy del recurso, que actúe como sustituto del recurso real. El proxy se comporta igual que el recurso y se encarga de crearlo cuando sea necesario.

Patrones de Diseño

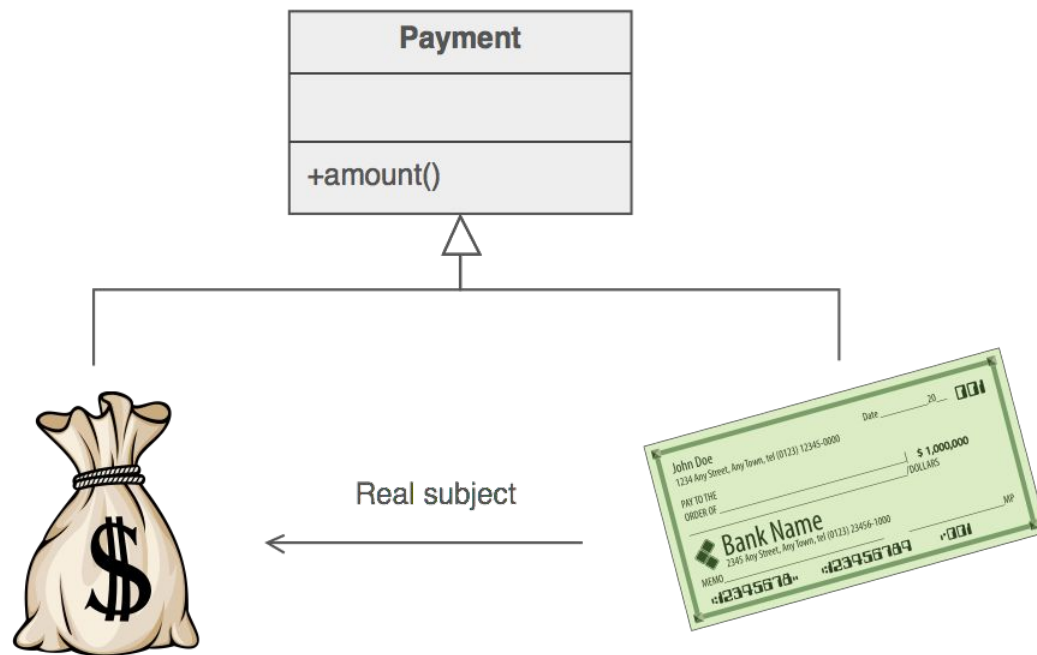
Patrón Proxy

Estructura:



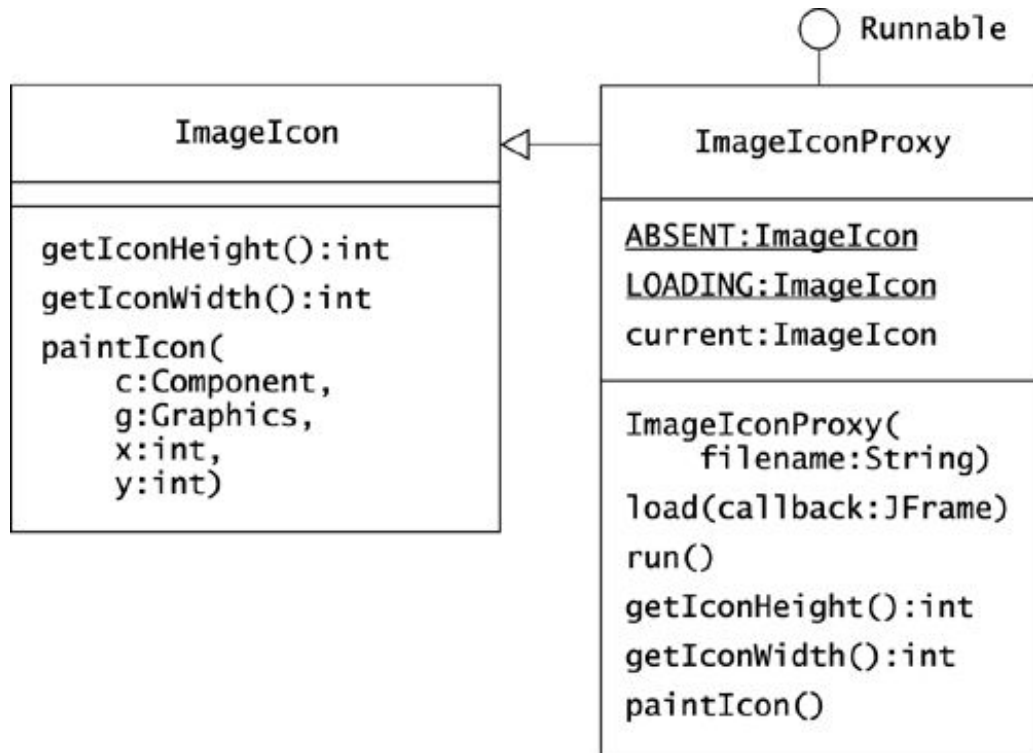
Patrones de Diseño

Patrón Proxy



Patrones de Diseño

Patrón Proxy Implementado para el Caso 2



Patrones de Diseño

Patrón Proxy

Implementación en Java:

Para crear un proxy en Java se debe crear una subclase de la clase que contendrá la imagen. El proxy define 2 variables estáticas que contienen los elementos sustitutos:

```
static final ImageIcon ABSENT = new ImageIcon("absent.jpg");  
static final ImageIcon LOADING = new ImageIcon("loading.jpg");
```

Patrones de Diseño

Patrón Proxy


La clase proxy se puede implementar:

```
import java.awt.*;  
import javax.swing.*;  
  
public class ImageIconProxy extends ImageIcon implements Runnable{  
    static final ImageIcon ABSENT = new ImageIcon("absent.jpg");  
    static final ImageIcon LOADING = new ImageIcon("loading.jpg");  
    ImageIcon current = ABSENT;  
    protected String filename;  
    protected JFrame callbackFrame;  
    ...  
}
```


Patrones de Diseño

Patrón Proxy

```
public ImageIconProxy(String filename) {  
    super(ABSENT.getImage());  
    this.filename = filename;  
}  
  
public void load(JFrame callbackFrame) {  
    this.callbackFrame = callbackFrame;  
    current = LOADING;  
    callbackFrame.repaint();  
    new Thread(this).start();  
}
```



Al instanciar el proxy, la imagen a mostrar será la referenciada por la variable estática **absent**.



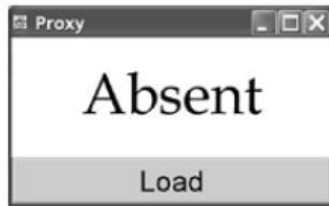
Cuando el cliente invoca al método **load()**, la imagen será la referenciada por la variable **loading** y se lanza un hilo que inicia la carga de la imagen real.

Patrones de Diseño

Patrón Proxy

```
public void run(){  
    current = new ImageIcon(filename);  
    callbackFrame.pack();  
}
```

Al ser invocado el método `run()`, la variable **current** hace referencia a la imagen real.



Patrones de Diseño

Caso 3

Patrones de Diseño

Caso 3

- La impresora departamental es un recurso compartido conectado a la red.
- El ambiente de ejecución es multithreading y por cada instancia de conexión, el sistema operativo ejecuta un nuevo hilo.
- Se desea definir que sólo pueda existir una instancia de esa impresora en el S.O. de manera simultánea. De esta manera se garantiza único punto global de acceso.



Patrones de Diseño

Caso 3

Intención

Asegurar que una clase tenga una única instancia y que ésta sea fácilmente accesible.

Problema

Una variable global hace accesible a un objeto, pero no previene de crear múltiples instancias de ese objeto.

Solución

Hacer que sea la propia clase la responsable de su única instancia. La clase puede garantizar que no se puede crear ninguna otra instancia.

Patrones de Diseño

Patrón Singleton

Propósito

Garantiza que una clase sólo tenga una instancia y proporciona un punto global de acceso a ella.

Aplicabilidad:

Se empleará el patrón Singleton cuando: deba haber exactamente una instancia de una clase y ésta deba ser accesible a los clientes desde un punto de acceso conocido.

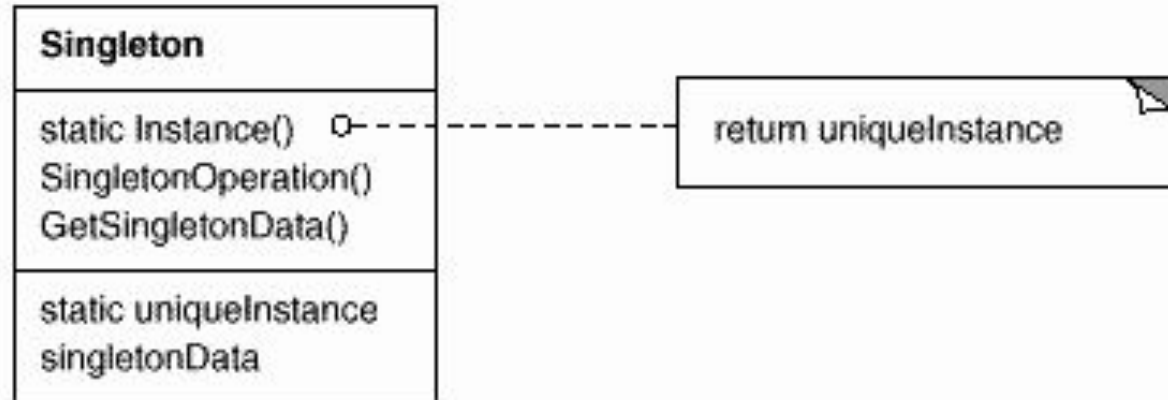
La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de usar una instancia extendida sin modificar el código.

Patrones de Diseño

Patrón Singleton

Creacional

Estructura:



Patrones de Diseño

Patrón Singleton

En Java:

```
public class Printer {  
    private static Printer INSTANCE = null;  
    private Printer() {  
    }  
    public static Printer getInstance() {  
        if (INSTANCE == null) {  
            INSTANCE = new Printer();  
        }  
        return INSTANCE;  
    }  
}
```

Se define la variable estática privada INSTANCE donde estará la referencia a la única instancia.

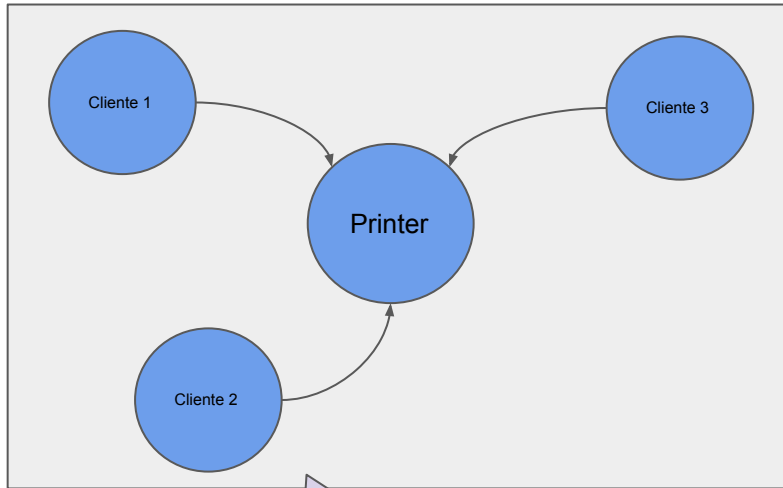
El acceso al constructor es privado, de modo tal que sólo la clase puede crear una instancia.

El método estático getInstance() verifica la presencia de instancia y retorna la referencia.

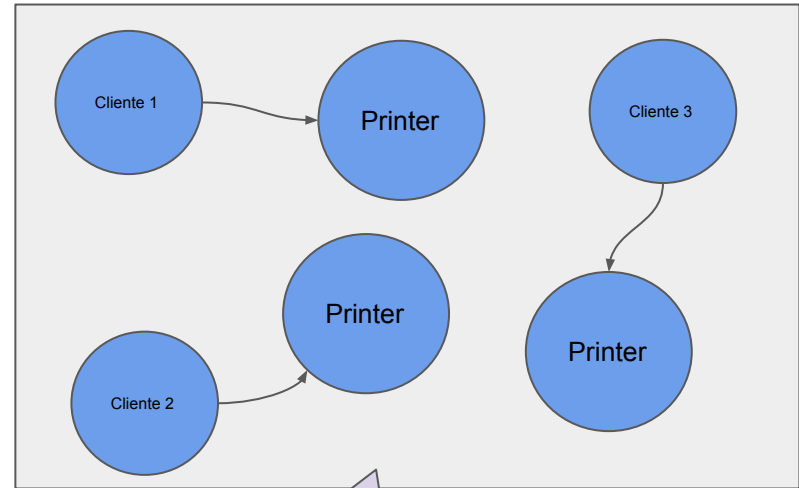
Patrones de Diseño

Patrón Singleton

Resultado:



Con el patrón Singleton, Printer garantiza que los clientes accedan siempre a una única instancia.



Sin el patrón Singleton, los clientes podrían crear nuevas instancias de la clase Printer (cuando el constructor es público).

Patrones de Diseño

Conclusiones

Patrones de Diseño

¿Cómo seleccionar un patrón?

- ✓ Considerar de qué forma los patrones resuelven problemas de diseño.
- ✓ Leer la sección que describe el propósito de cada patrón.
- ✓ Estudiar las interrelaciones entre los patrones.
- ✓ Analizar patrones con el mismo propósito.
- ✓ Examinar las causas de rediseñar.
- ✓ Considerar qué debería ser variable en el diseño.

Patrones de Diseño

¿Cómo emplear un patrón?

- ✓ Leer el patrón, todas sus secciones, para tener una visión global.
- ✓ Estudiar la Estructura, Participantes y Colaboraciones.
- ✓ Revisar el ejemplo de código.
- ✓ Asociar a cada participante del patrón un elemento del sistema a diseñar.
- ✓ Implementar las clases y métodos relacionados con el patrón.

Patrones de Diseño

¿Cuándo emplear un patrón?

- ▼ No es correcta la idea de tratar de aplicar patrones tanto como sea posible.
- ▼ No existe una receta mágica para determinar cuándo debe o cuando no debe aplicarse un patrón.

La experiencia es la mejor herramienta para saber cuándo es apropiado emplear un patrón y cuando no.

Patrones de Diseño

Cuestionario

Ingresar a: <https://b.socrative.com/login/student/>

Habitación/Room: **UTNDDS**



Referencias

Balaguer, F., Garrido, A., Introducción a Patrones de Diseño.

Tópicos de Ingeniería de Software II. Postgrado Universidad Nacional de La Plata. 2011.

Gamma, E. et al., Design Patterns: Elements of Reusable Object-Oriented Software.

Addison-Wesley. 1994.

Metsker, S., The Design Patterns Java Workbook.

Addison-Wesley. 2002.