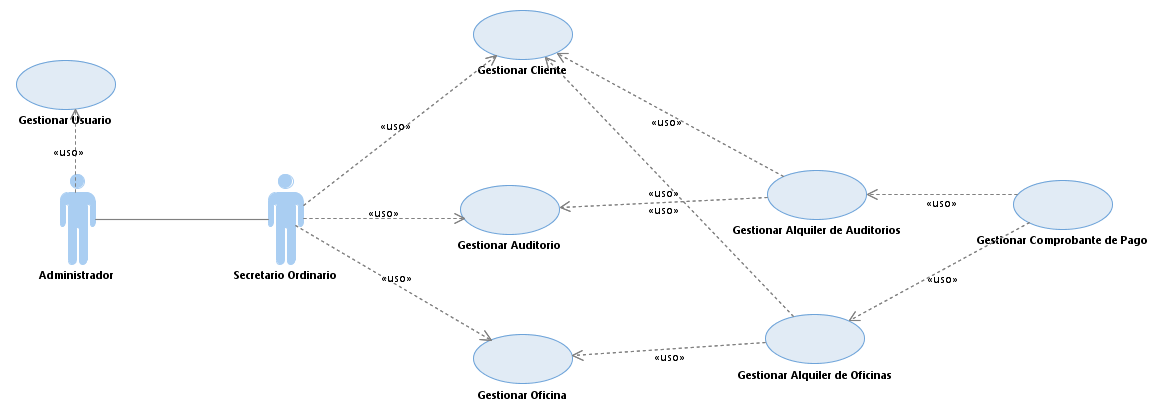
**Estructura de Diseño**

**Subsistemas**

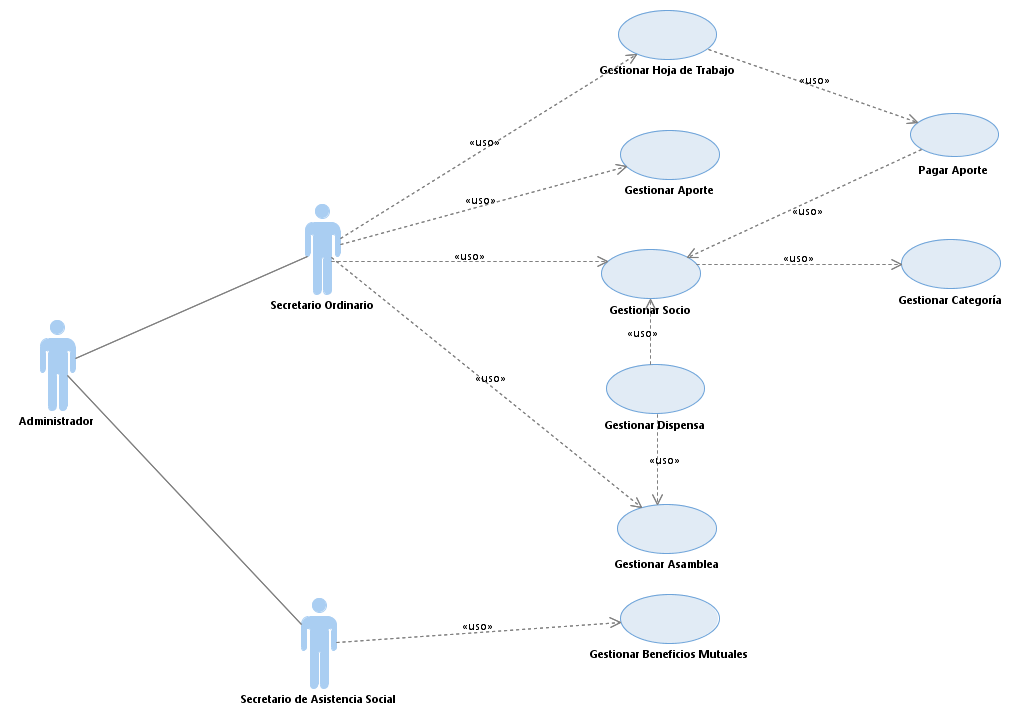
La aplicación está compuesta por dos subsistemas. Se hará una breve descripción de cada uno antes de especificar las funcionalidades de cada uno de ellos.



**Subsistema Administrar Ambientes**

****

**Subsistema Administrar Socios**



**Patrones**

1. **Patrón Repository:**

**RESUMEN**

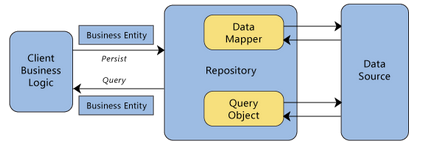
El propósito de utilizar este patrón será de separar la lógica que se encarga de recuperar los datos y le asigna el modelo de entidad de la lógica de negocio que actúa sobre el modelo, además de mediar .entre la capa de origen de datos y las capas de negocio de la aplicación.

El patrón Repository trabajo de la siguiente manera: el repositorio consulta el origen de datos para los datos, asigna los datos del origen de datos a una entidad comercial y persiste cambios en la entidad a una entidad comercial, y persiste cambios en entidad a Lafuente de datos.

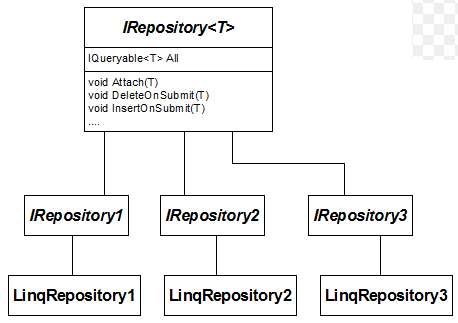
El patrón repositorio cuenta con tres características importantes, la cuales se detallan a continuación:

* Centraliza la lógica de datos o lógica de acceso a un servicio web.
* Proporciona un punto de sustitución para las pruebas unitarias.
* Se proporciona una arquitectura flexible que se puede adaptar como el diseño general de una aplicación escalable.

El patrón repositorio se encargará de encapsular el conjunto de objetos que persisten en un almacén de datos y todas las operaciones que se realizan sobre ellos, proporcionando una visión más orientada a objetos de la capa de persistencia.

**ESTRUCTURA**

**EJEMPLO**

****

1. **PATRON INYECCION DE DEPENDENCIA:**

**RESUMEN**

Su creador Fowler definió el concepto de forma informal denominándolo como el Principio de Hollywood, en el que, tras una audición, se le decía al actor la famosa frase de No nos llames, nosotros te llamaremos.

Este patrón se utilizará para inyectar comportamientos a componentes; dicho de otra manera con este patrón se extraerá responsabilidades a un componente para delegarlas en otro, estableciendo un mecanismo a través del cual el nuevo componente pueda ser cambiado en tiempo de ejecución.

1. **ABSTRACT FACTORY**

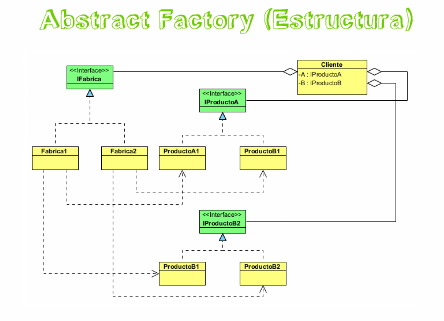
**REDUMEN:**

Proporciona una interfaz para la creación de familias de objetos relacionados sin especificar sus clases concretas.

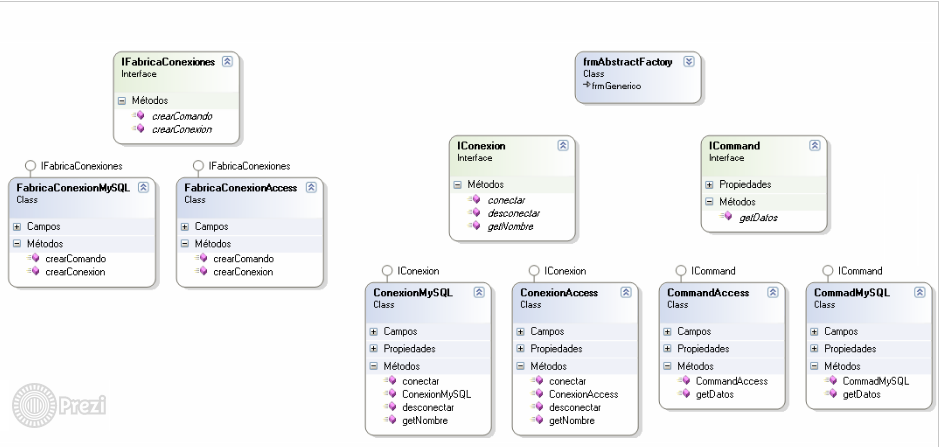
Aplicabilidad: Un sistema debe ser independiente de la forma en que sus productos son creados, compuestos y representados.

Un sistema debe ser configurado con una de muchas familias de productos disponibles.

**ESTRUCTURA:**

****

**EJEMPLO**

****

1. **ADAPTER**

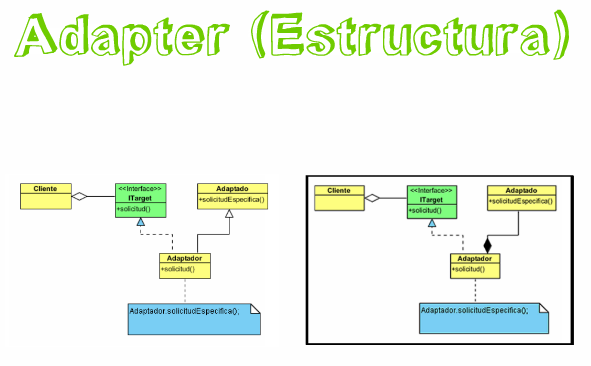
**RESUMEN**

El adaptador permite la colaboración entre clases con interfaces incompatibles.

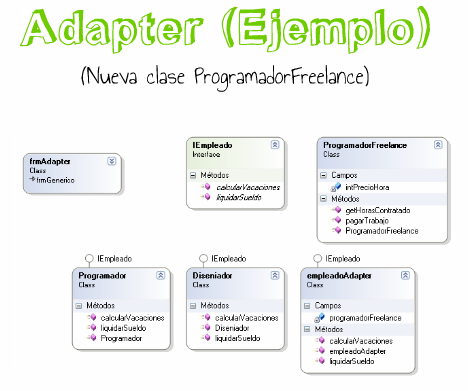
**Aplicabilidad:** Se emplea una clase existente que no es compatible con la interfaz que se le requiere.

Se desea crear una clase reutilizable que coopere con clases no relacionadas.

**ESTRUCTURA**

****

**EJEMPLO**

****

1. **PATRON OBSERVADOR**

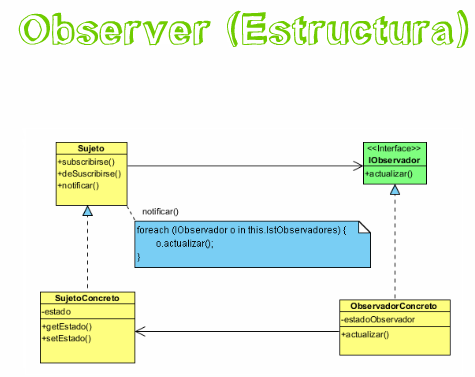
**RESUMEN**

También se le conoce como el patrón de Publicación - Inscripción.

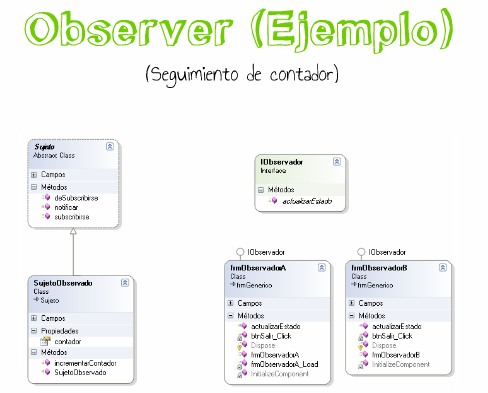
Este patrón se utiliza cuando un objeto quiere notificar a otros objetos de un evento. En principio, lo que sucede es que un Objeto (llamémoslo Observador) se inscribe a otro Objeto (llamémoslo Sujeto) y este le avisa cuando un evento es disparado (o cuando el estado del Sujeto ha cambiado).

Se pueden inscribir varios Observadores, en ese caso (generalmente) la notificación del cambio de estado va de acuerdo al orden en cómo se suscribieron los Observadores

**ESTRUCTURA**



**EJEMPLO**

****

**Relación de Requisitos:**

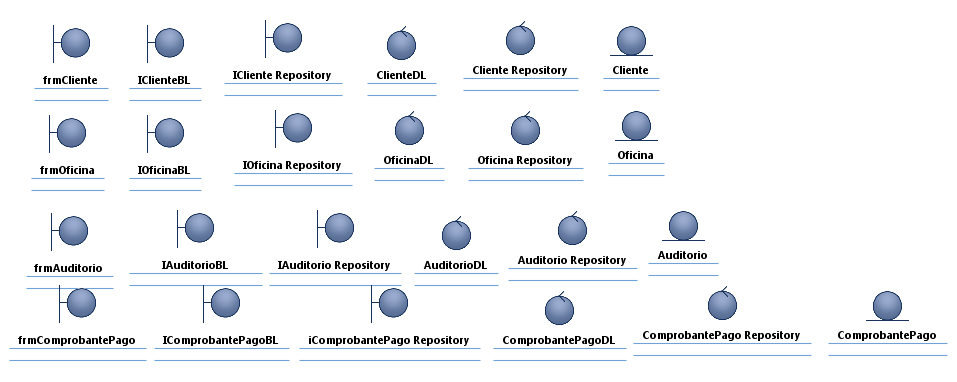
**[Relación 1]**

**Subsistema Administrar Socio**

**Vista de los Participantes**

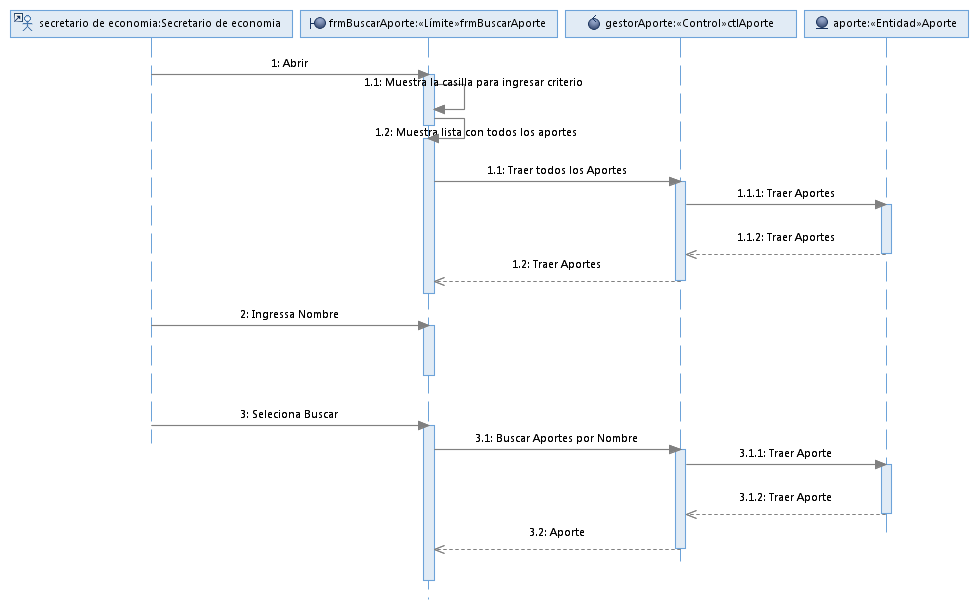
Límite es la vista****

**Subsistema Administrar Ambientes**

**Límite es la vista**

**Escenario Básico:**

**Subsistema Administrar Socios**

****