11ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 2

Principios y Patrones de Diseño

Mtro. ISC Miguel Ángel Sánchez Hernández

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

50 AÑOS Educación Continua 1971 - 2021

Objetivo

Aprender patrones de diseño, para poderlos aplicar al proyecto final

Lo que veremos

- Que es un Patrón de diseño
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

¿Qué es un Patrón?

Se puede decir que un patrón son soluciones probadas para un problema en especifico, estos nos ayudaran a comprender a la perfección los principios fundamentales de diseño.

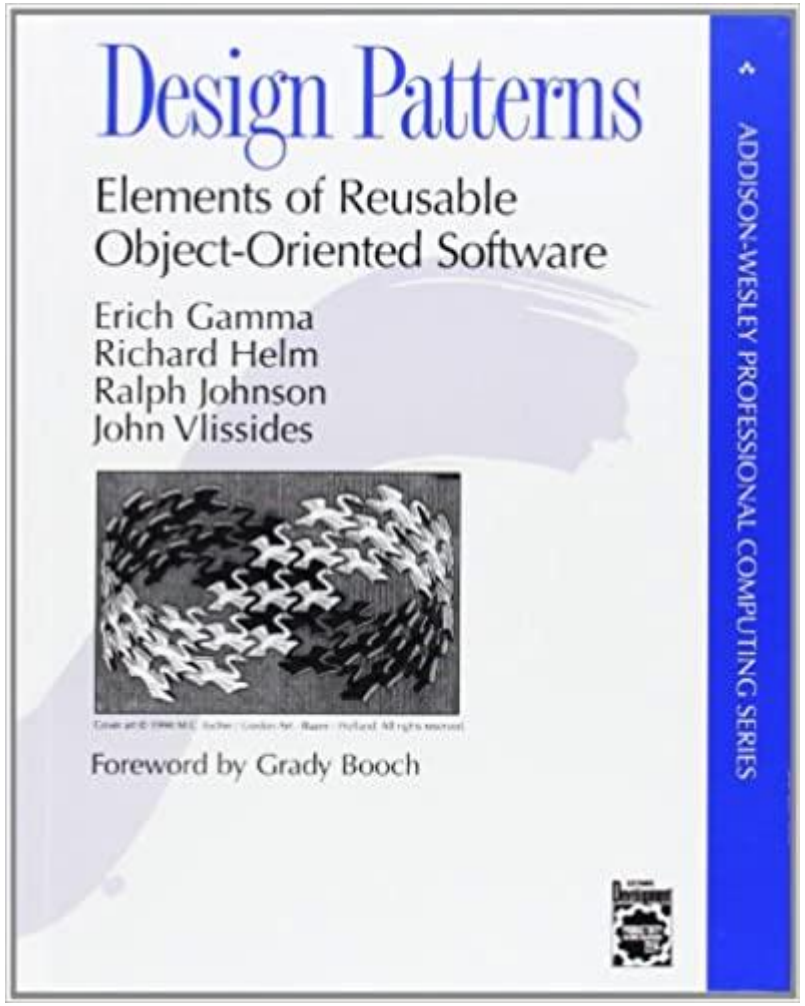
Las decisiones del diseño nos brindan la posibilidad de identificar los módulos de un sistema de software y como funcionaran estos módulos juntos para alcanzar los objetivos.

GoF

1994 Design patterns:
Elements of Reusable Object-Oriented Software

- **Erich Gamma**
- **Richard Helm**
- **Ralph Johnson**
- **John Vissides**

Gang of Four→ GoF



Criterios de Diseño

- **Variaciones protegidas:** Si se piensa que un componente podría cambiar, utilice interfaces, estas permiten cambiar la clase de implementación sin afectar las dependencias existentes.

- **Bajo acoplamiento:** Esto debe aplicarse para que los cambios hechos en un sección no afecten a otras secciones relacionadas o no relacionadas, el clásico efecto es si hacemos un cambio en la interfaz gráfica y tengamos que cambiar algo en la base de datos, es frágil la aplicación.

- **Alta cohesión:** Si una clase tiene una única responsabilidad.

Categorías de los Patrones

Creational	Structural	Behavioral
• Abstract factory pattern • Builder pattern • Factory method pattern • Prototype pattern • Singleton pattern	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of responsability • Command • Interpreter • Iterator • Mediator • Observer • State • Strategy • Template method • Visitor

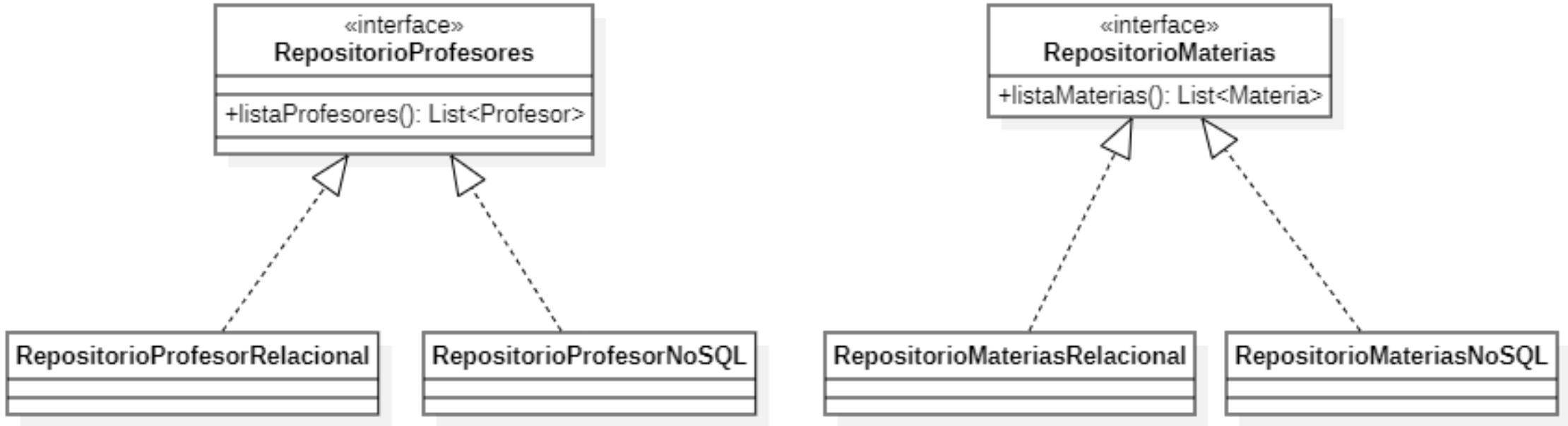
Abstract Factory pattern

Objetivo: Necesitamos crear colecciones de clases, pero no necesitamos indicar el tipo concreto de la clase a usar, dado que puede variar de una situación a otra.

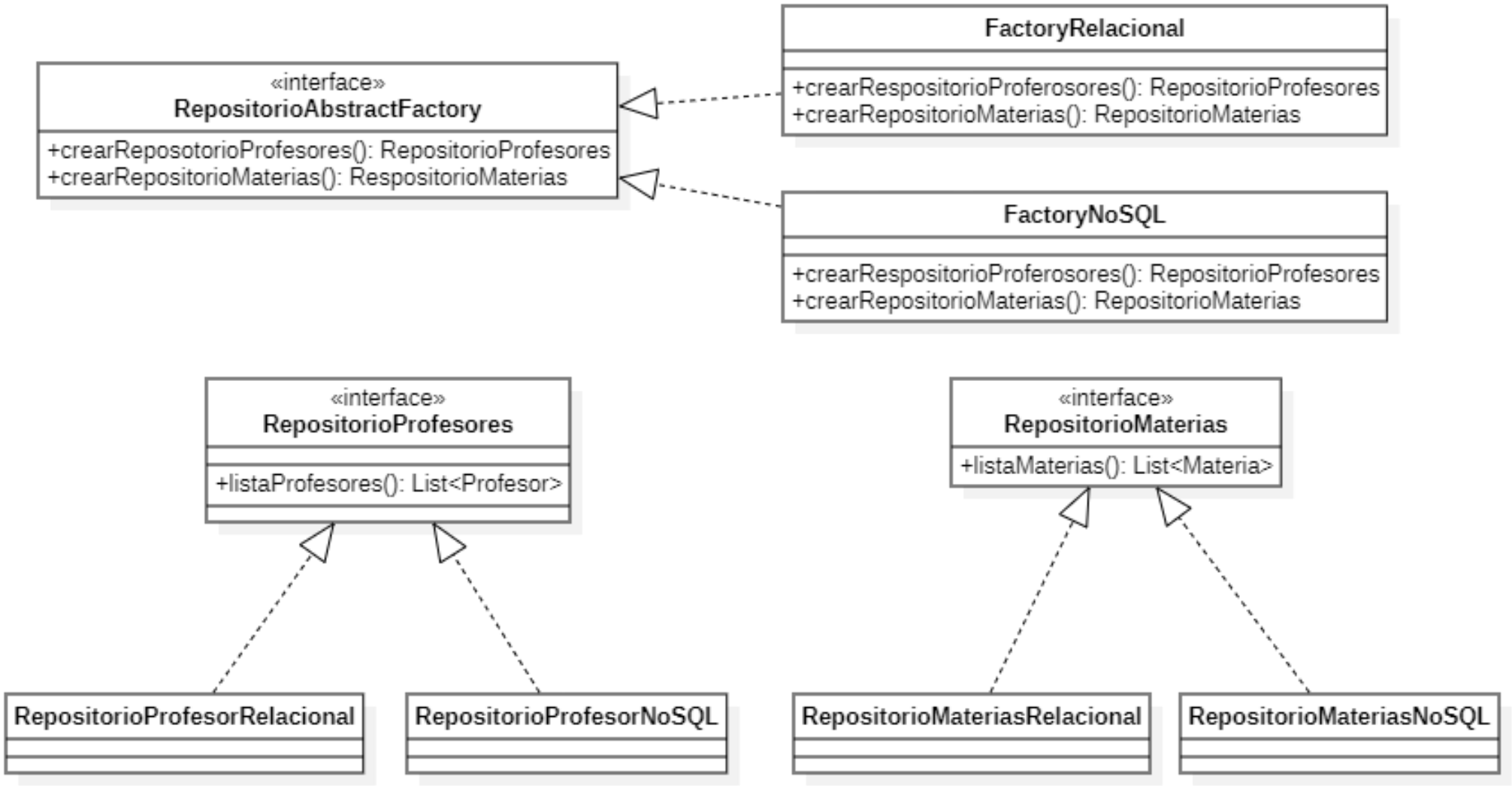
Técnica: Poder definir interfaces o clases abstractas para crear objetos, y dejar que las subclases decidan qué clase instanciar.

Ejemplo: Quieren los clientes ocupar conexiones de base de datos de dos paradigmas diferentes, modelo relacional y el modelo no relacional, de sus dos repositorios profesores y materias.

Práctica Abstract factory pattern versión 1



Ejercicio Abstract factory Pattern versión 2



Ejercicio Abstract factory pattern versión 2

```
package dgtic.inicio;
import dgtic.repositorio.implementacion.FactoryNoSQL;
import dgtic.repositorio.implementacion.FactoryRelacional;
import dgtic.repositorio.interfaces.RepositorioAbstractFactory;
import dgtic.repositorio.interfaces.RepositorioMaterias;
import dgtic.repositorio.interfaces.RepositorioProfesores;
public class Principal {
        public static void main(String[] args) {
                RepositorioAbstractFactory factory=new FactoryRelacional();
                RepositorioProfesores rep=factory.crearRespositorioProfesores();
                rep.listaProfesores();

                RepositorioAbstractFactory factoryDos=new FactoryNoSQL();
                RepositorioMaterias repDos=factoryDos.crearRespositorioMaterias();
                repDos.listaMaterias();
        }
}
```

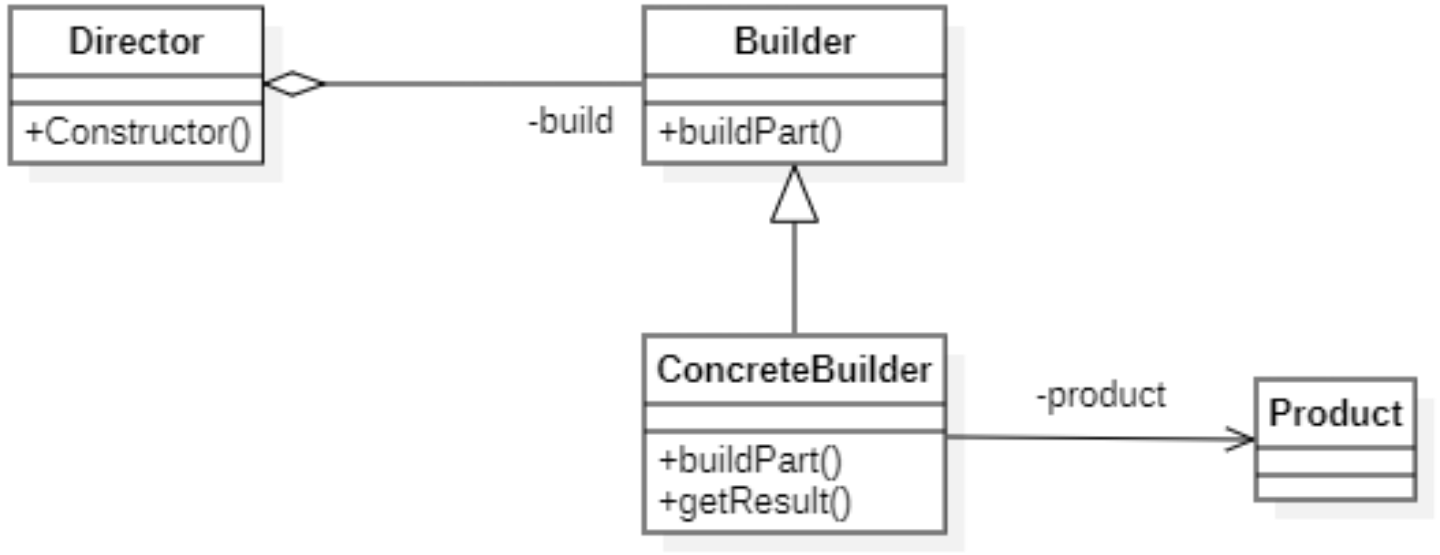
Builder Pattern

Objetivo: Crear un objeto con muchas opciones de posibles de configuración. Es decir construir objetos complejos.

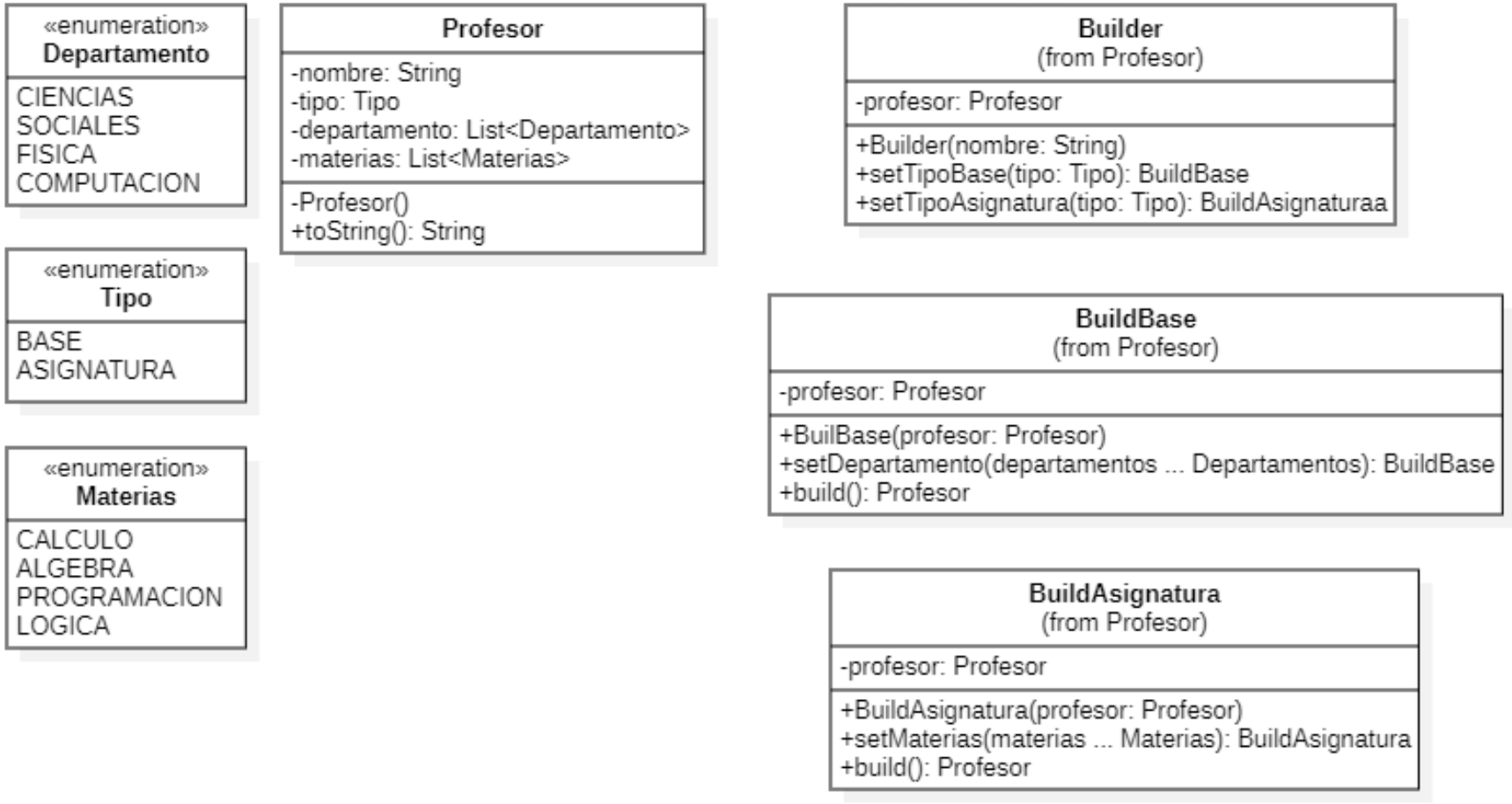
Técnica: Para la construcciones de objetos complejos separa el proceso de construcción de su representación.

Ejemplo: Los clientes desean construir un objeto profesor, con la diferencia que en tiempo de compilación si tiene base, que indique en que departamento esta o si tiene asignatura indicar las materias que imparte.

Builder Pattern



Ejercicio Builder Pattern



Ejercicio Builder Pattern

```
package fes.aragon.inicio;
import fes.aragon.modelo.Departamento;
import fes.aragon.modelo.Materias;
import fes.aragon.modelo.Profesor;
import fes.aragon.modelo.Tipo;
public class Principal {
		public static void main(String[] args) {
			Profesor profesor=new Profesor.Builder("Fernando")
					.setTipoBase(Tipo.BASE)
					.setDepartamento(Departamento.COMPUTACION)
					.build();
			Profesor profesorDos=new Profesor.Builder("Maria")
					.setTipoAsignatura(Tipo.ASIGNATURA)
					.setMaterias(Materias.ALGEBRA,Materias.CALCULO)
					.build();
			System.out.println(profesor.toString());
			System.out.println(profesorDos.toString());
		}
}
```

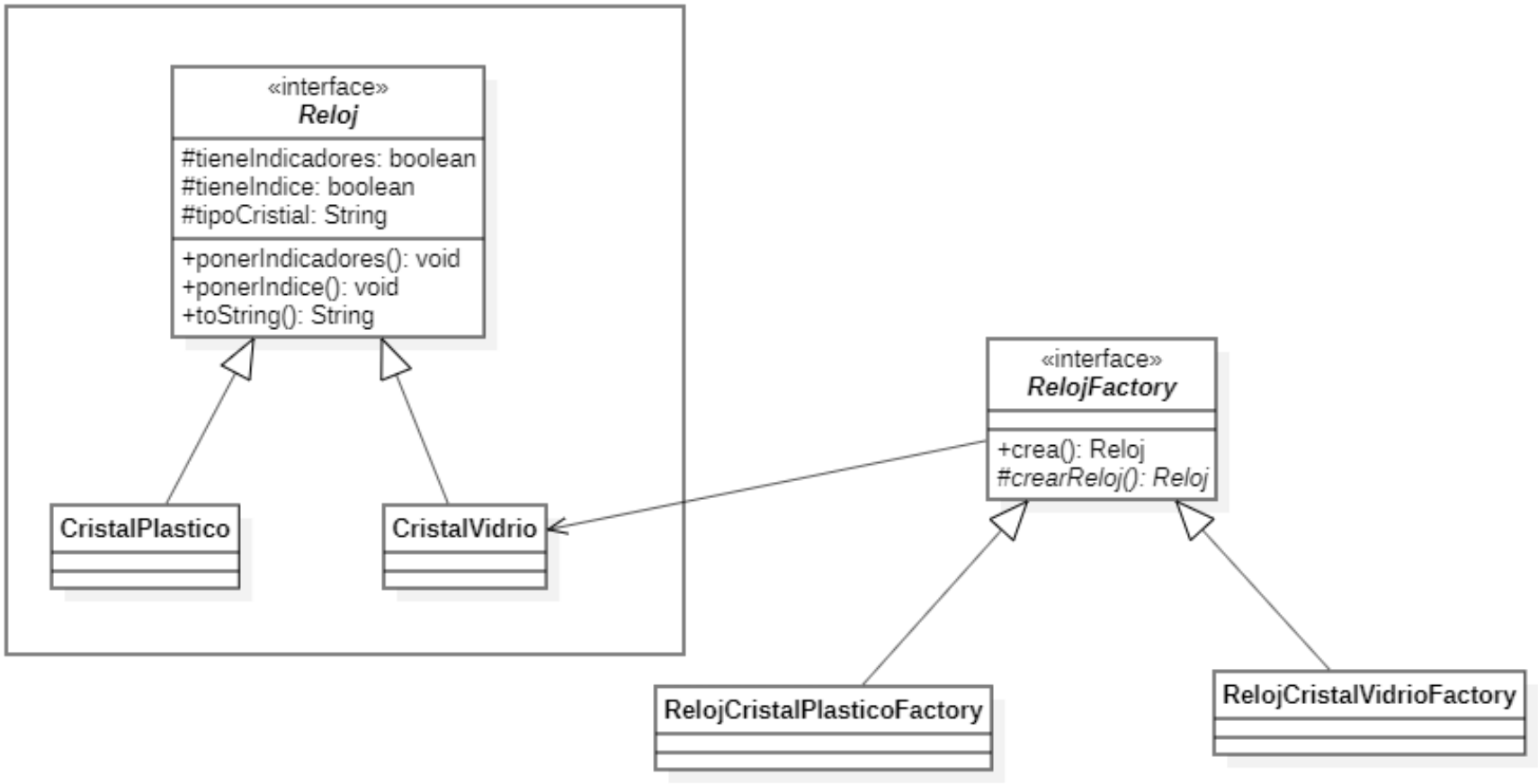
Factory Method Pattern

Objetivo: Definir una interfaces para crear un objeto, pero se dejar a las subclases que clase concreta instanciar.

Técnica: Utilizar una interfaz (clase abstracta o interfaz), y las subclases indicaran el tipo de clase a instanciar.

Ejemplo: Se quiere construir un objeto de tipo reloj, pero las subclases deben definir el tipo de cristal, ya sea de vidrio o platico.

Ejercicio Factory Method Pattern



Ejercicio Factory Method Pattern

```
package dgtic.inicio;
import dgtic.modelo.Reloj;
import dgtic.modelo.interfaz.RelojCristalPlasticoFactory;
import dgtic.modelo.interfaz.RelojCristalVidrioFactory;
import dgtic.modelo.interfaz.RelojFactory;
public class Inicio {
		public static void main(String[] args) {
				RelojFactory factory=new RelojCristalPlasticoFactory();
				Reloj reloj=factory.crea();
				System.out.println(reloj.toString());

				RelojFactory factoryDos=new RelojCristalVidrioFactory();
				Reloj relojDos=factoryDos.crea();
				System.out.println(relojDos.toString());
		}
}
```

Singleton Pattern

Objetivo: Crear un objeto para una única instancia, por ejemplo para configuraciones globales, para almacenar registros o pools de conexiones.

Técnica: Una clase tiene una sola instancia y debe proporcionar acceso global.

Ejemplo: Se necesita crear un único objeto que permita guardar datos en archivos.

Ejercicio Singleton Pattern


```
package dgtic.inicio;
import dgtic.modelo.Archivo;
public class Inicio {
		public static void main(String[] args) {
				Archivo archivo=Archivo.getInstancia();
				Archivo archivoDos=Archivo.getInstancia();
				System.out.println(archivo.equals(archivoDos));
				System.out.println(archivoDos.equals(archivo));
		}
}
```

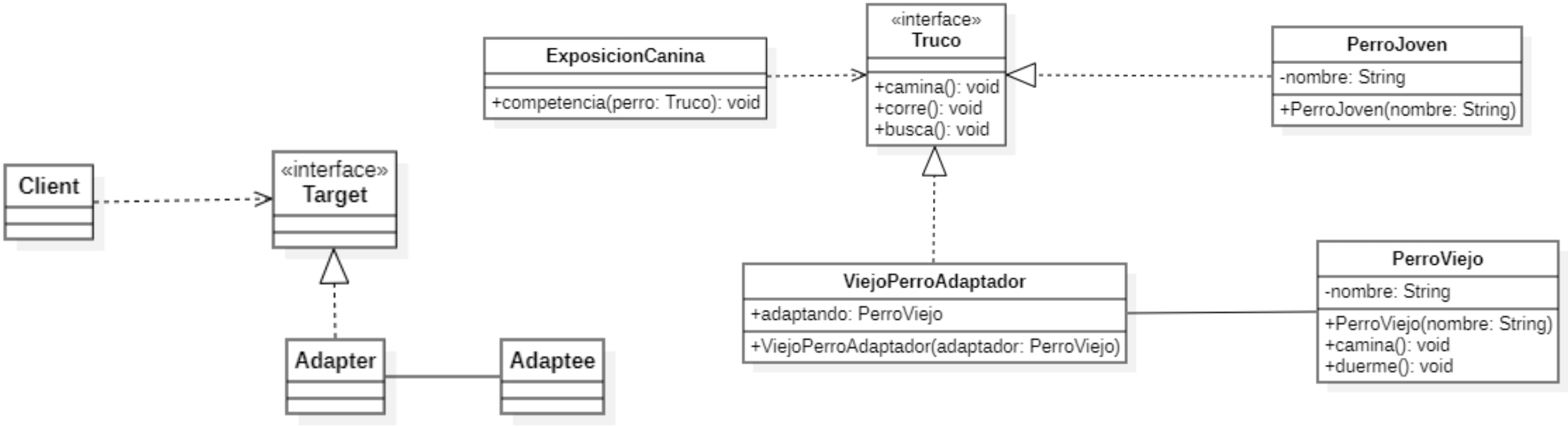
Adapter Pattern

Objetivo: Se necesita comunicar dos interfaces que son incompatibles.

Técnica: Ocupar los principios de delegación, herencia y abstracción.

Ejemplo: Se tienen perros que corren, caminan y buscan cosas, pero también tenemos perros viejos que solo caminan y duermen, se quiere unir las dos clases de perros para una exposición canina.

Ejercicio Adapter Pattern



Ejercicio Adapter Pattern

```
package dgtic.inicio;
import dgtic.adapter.ViejoPerroAdaptador;
import dgtic.modelo.PerroJoven;
import dgtic.modelo.PerroViejo;
import dgtic.servicio.ExposicionCanina;
public class Inicio {
		public static void main(String[] args) {
				PerroViejo viejo=new PerroViejo("Rojo");
				PerroJoven nuevo=new PerroJoven("Negro");
				ViejoPerroAdaptador adapta=new ViejoPerroAdaptador(viejo);
				ExposicionCanina expo=new ExposicionCanina();
				expo.competencia(adapta);
				System.out.println("-----");
				expo.competencia(nuevo);
		}
}
```

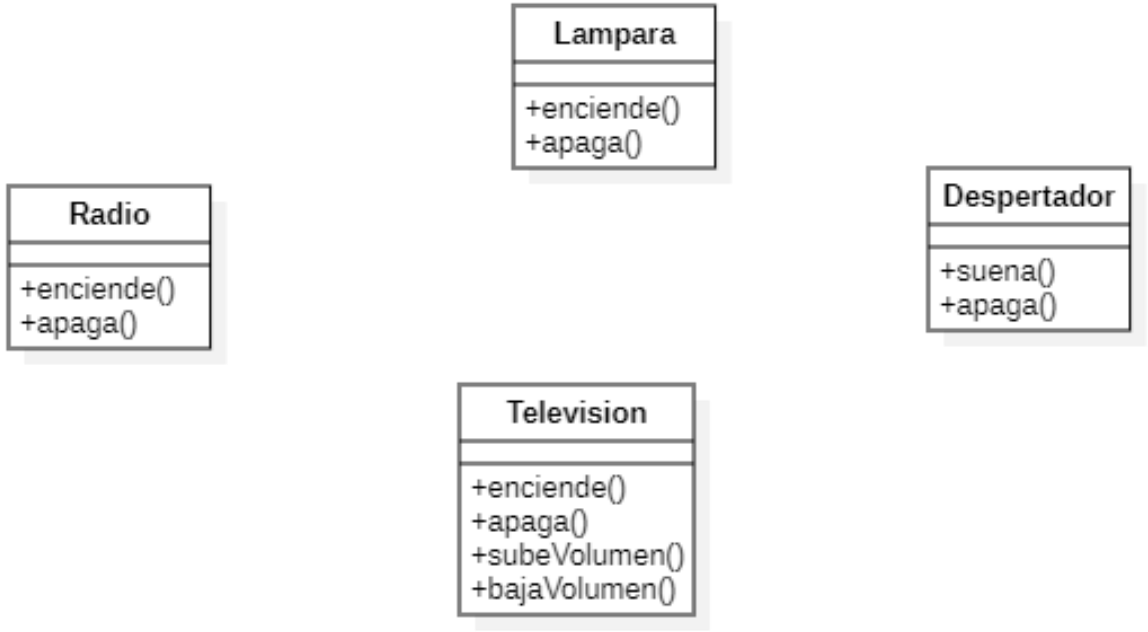
Mediator Pattern

Objetivo: Poder encapsular como se interactúan con un conjunto de objetos

Técnica: Encapsular en un solo objeto la lógica de negocio que se necesita para interactuar con los otros objetos.

Mediator Pattern

Ejemplo: Se tiene un conjunto de sistemas individuales que controlan la televisión, radio, lámparas y despertador.

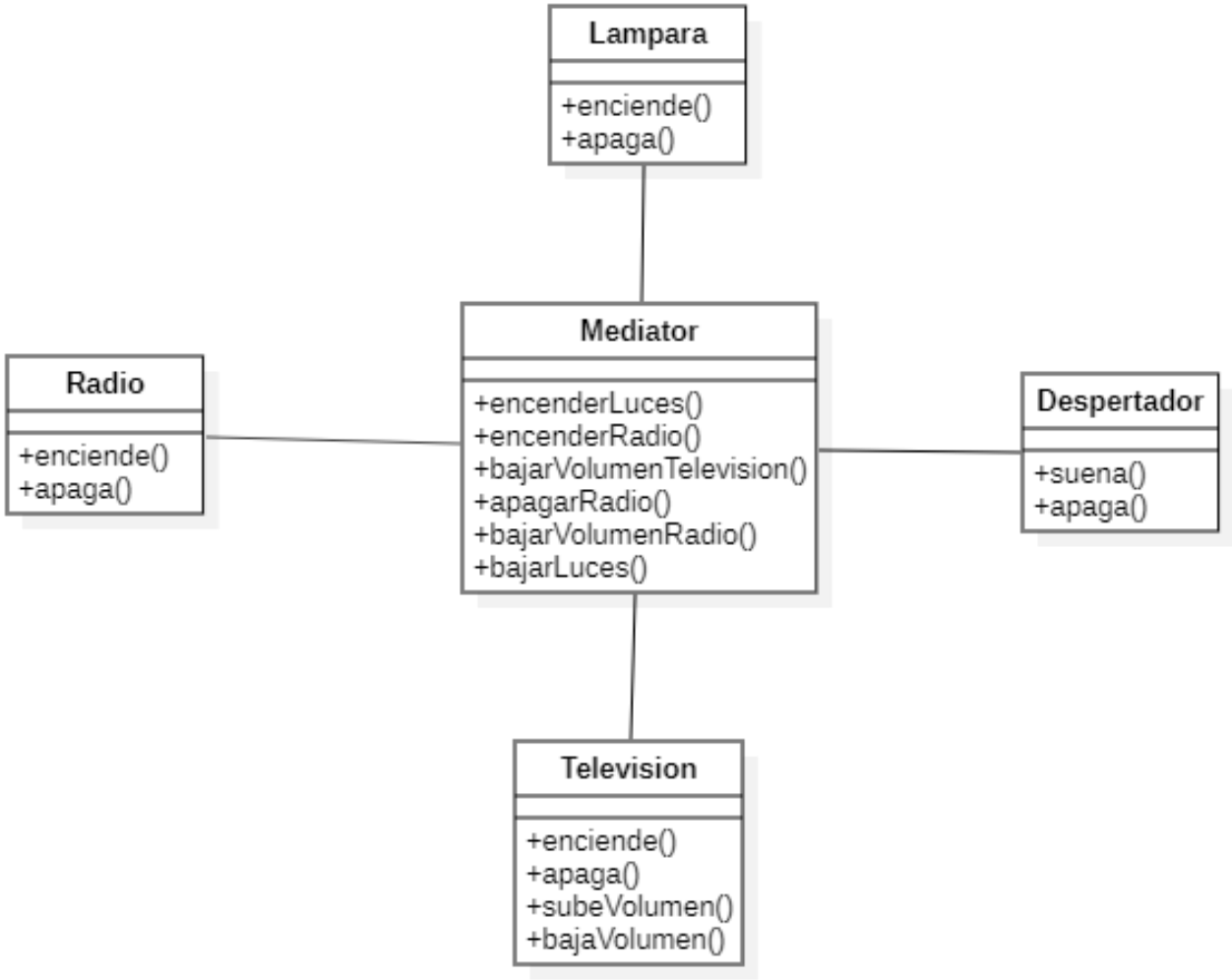


Mediator Pattern

Ahora se necesita estas reglas de negocio:

- Al sonar el despertador que se encienda la luces.
- Cuando se apague el despertador, encender el radio.
- Si encendemos el radio, bajar el volumen de la televisión.
- Al encender la televisión, apagar el radio.
- Al bajar el volumen del radio, bajar la intensidad de la luz

Mediator Pattern



Lo que aprendimos

- Comprender que es un Patrón de diseño
- Identificar como ocupar los patrones:
  - Creational Patterns
  - Structural Patterns
  - Behavioral Patterns