DA2 Teórico - Clase 2

23/08/2021

Siempre parados en la carpeta root (donde se creo el archivo .sln)

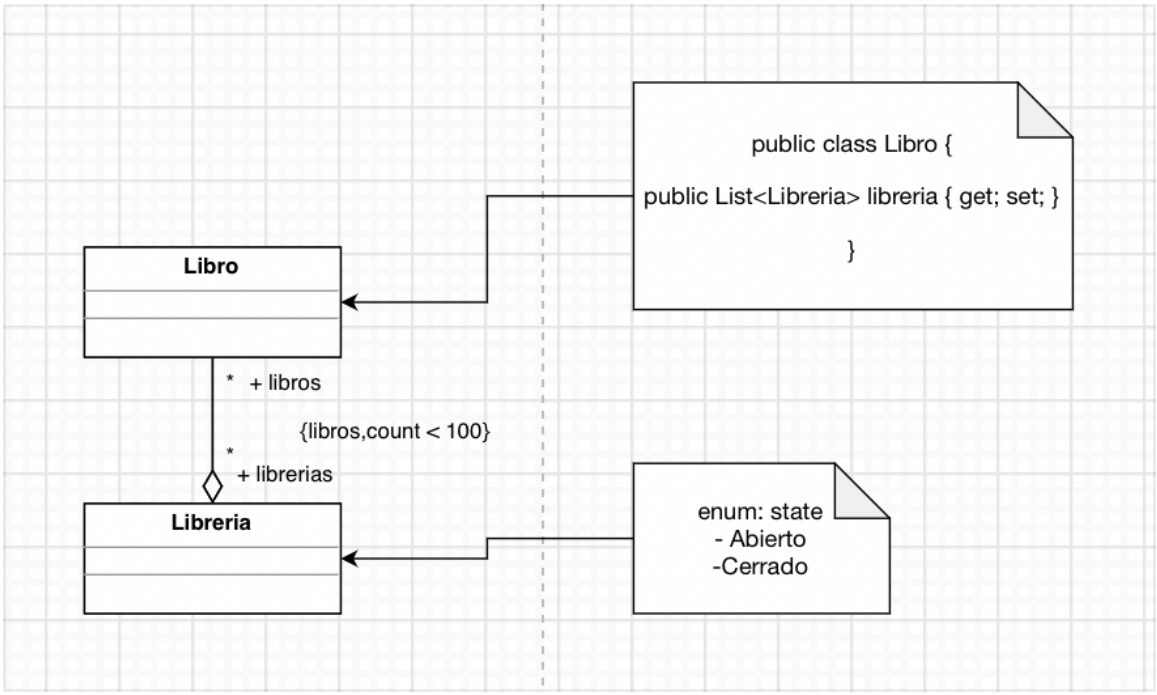
```
dotnet new sln -n Obligatorio2

dotnet new webapi -n Obligatorio2.WebAPI
dotnet new classlib -n Obligatorio2.BusinessLogic
dotnet new classlib -n Obligatorio2.DataAccess
dotnet new classlib -n Obligatorio2.Domain

dotnet sln add Obligatorio2.WebAPI
dotnet sln add Obligatorio2.BusinessLogic
dotnet sln add Obligatorio2.DataAccess
dotnet sln add Obligatorio2.Domain

dotnet build

cd Obligatorio2.WebAPI
dotnet run

cd ..
cd Obligatorio2.DataAccess
dotnet add package Microsoft.EntityFrameworkCore

cd ..
dotnet add Obligatorio2.WebAPI reference
Obligatorio2.BusinessLogic
dotnet add Obligatorio2.BusinessLogic reference
Obligatorio2.DataAccess

dotnet new console -n ProyectoConsola

dotnet run test
```

———————————————————————

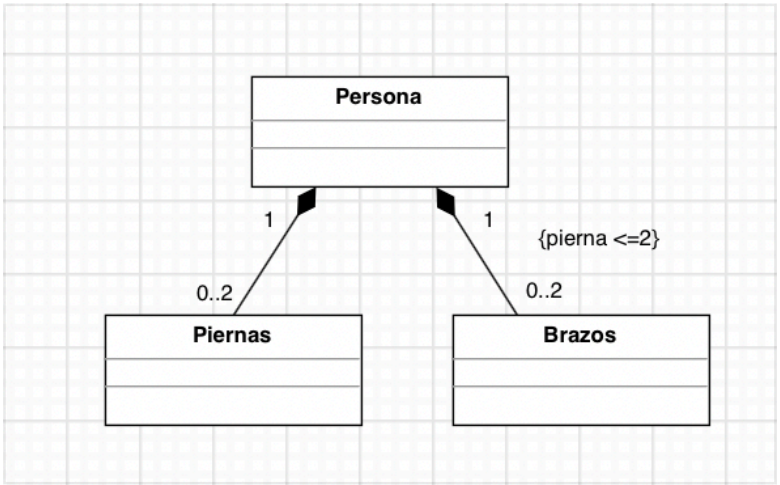
Repaso UML:

Adornos

- Notas
- Contraints



- Cardinalidad, multiplicidad
- Estereotipos
- Roles

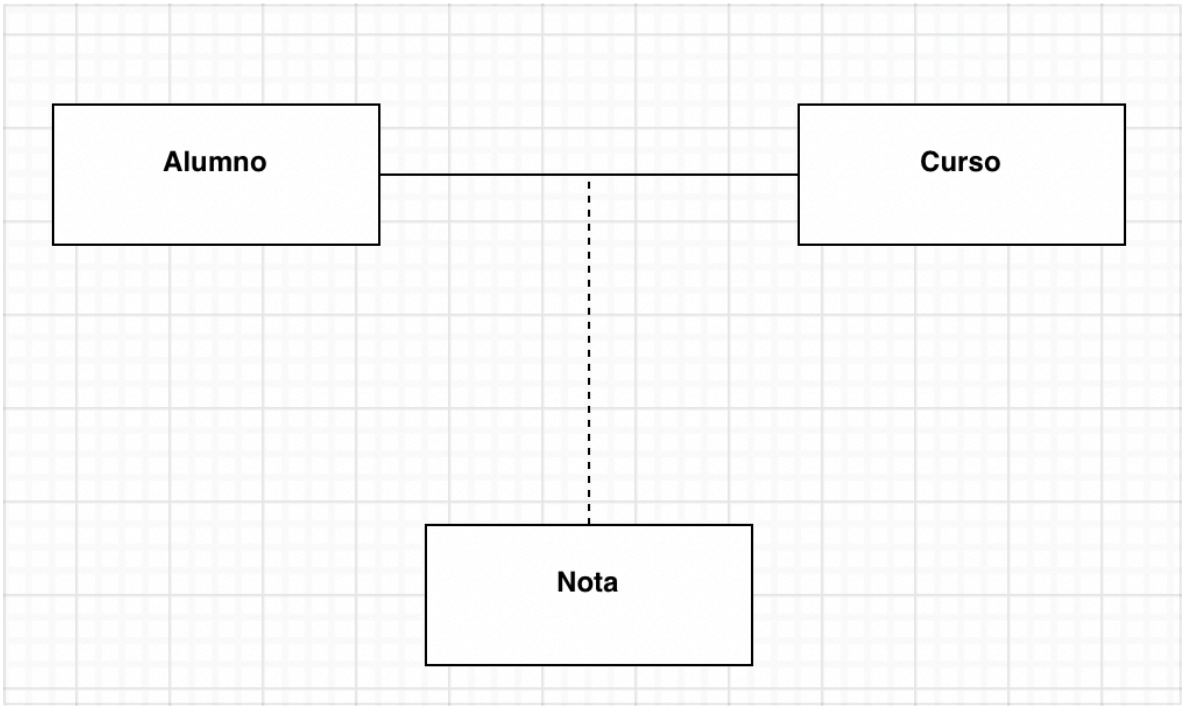



```
public class Libro {
      public List<Libreria> libreria { get; set; }
}
```

```
public class Libreria {
	public List<Libro> libro { get; set; }
}
```

Constraints: {condicion}

Clase de asociación



Diagramas de interacción

- Secuencia: se ve la linea de tiempo de los objetos y sus llamadas a métodos.
- Comunicación: se ve la relación entre objetos (participantes).

Resumen intenso:
https://aulas.ort.edu.uy/pluginfile.php/444295/mod_resource/content/2/Diagramas%20de%20interacción.pdf

```
public class A
{
	public void DoOne(B myB)
	{
		myB.doTwo();
	}
}
```

```
public interface Payment
{
    public void sale();
}

public class CreditPayment : Payment
{
    public void sale()
    {
    ...
    }
}
```

:Register
——> :Payment
———> :CreditPayment. ❌

:Register
——> :CreditPayment ✅

:Register
——> :DebitPayment ✅

:Register
———> :Payment ✅

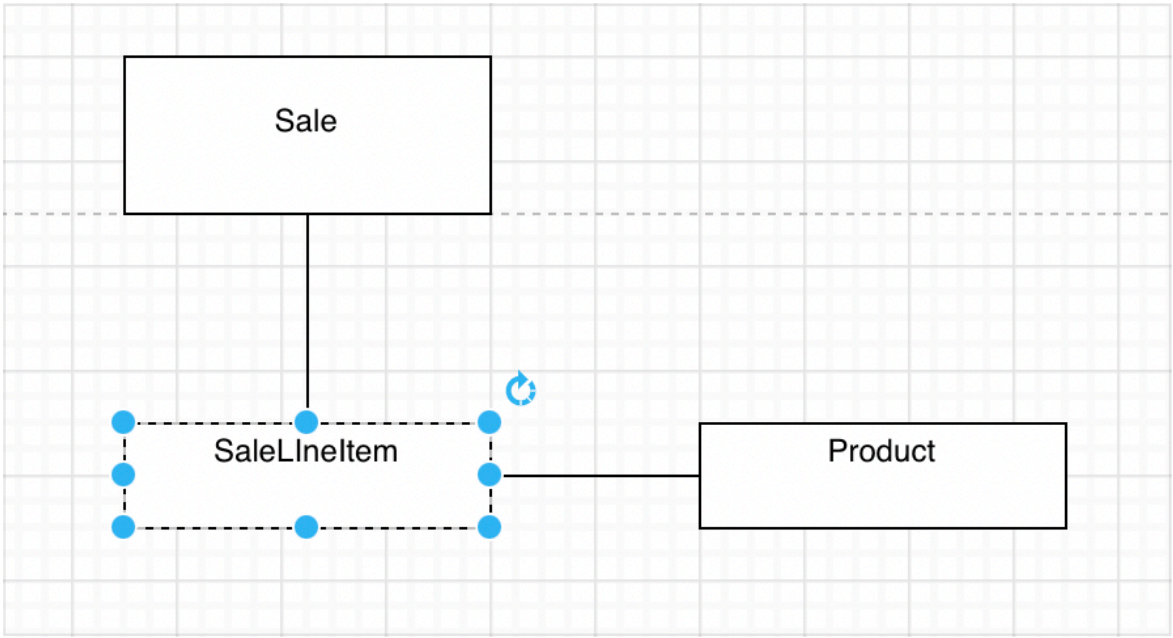
Principios SOLID:

- (S) Responsabilidad única: Clases con un único motivo por el cual cambiar. Contra ejemplo de la navaja suiza, que tenía muchas herramientas, pero si mañana debemos cambiar alguna podría impactar en las demás herramientas.
- (O) Abierto cerrado: Abierto a la extensibilidad y cerrado a la modificación. Ejemplo del abrigo que no es práctico abrirse le cuerpo y ponerlo por dentro, sino que debemos extenderlo por fuera del mismo.

- (L) Sustitución de Liskov: Hay que hacer énfasis que en la abstracción este siempre bien definida y tenga lo necesario para sus implementaciones. Ejemplo del pato y el juguete.
- (I) Segregación de interfaces: Hacer que las interfaces solamente expongan o expresen lo que los clientes o quién las usen realmente necesitan y no de más. Ejemplo de los auriculares con muchos conectores.
- (D) Inversión de dependencias: Módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones. Ejemplo del soldador y el enchufe.

Patrones GRASP:
Como asignar responsabilidad, o ¿dónde debería ir el código?
Es un acronimo.

- Experto: Debo poner el código en la clase donde tengo todas las herramientas para llevarlo a cabo.
- Creador: Quién debe encargarse de crear nuevas instancias.



- Fabricación pura: Son clases que no pertenecen al dominio pero las necesito para encapsular funcionalidades que no se dónde ponerlas. Ejemplo los helpers, (pasear una fecha, funciones matemáticas)
- Alta cohesión: Todo lo que tenga que ver este junto.
- Bajo acomplamiento: Una clase debería estar relacionada con

la menor cantidad clases. Si lo logramos, tenemos mejor mantenibilidad, porque un cambio impactaría en una menor cantidad de clases o paquetes.
- Indirección: Clases intermedias para desacoplar implementaciones
- Controlador: Esta relacionado más a un cliente (windows form, webapi)
- Variaciones protegidas (no hables con extraños): Solo llamar a métodos que conoces y no hacer una transitiva.
- Poliformismo: No preocuparnos de la implementación. En el caso de las interfaces, nos permite limitar la funcionalidad.

FIN DEL REPASO DE DISEÑO DE APLICACIONES I