

diseño de sistemas

utn
frlp

U3 – Modelado Estático – Propiedades avanzadas de Clases



Temario

- Revisión de conceptos...
- UML
- Propiedades avanzadas de Clases
- Asociación (Agregación y Composición)
- Generalización
- Dependencia
- Realización
- Herencia Múltiple
- Resolución de casos prácticos

utn frlp ds



Modelado de tipos primitivos

Clases enumerativas

<< enumeration >> Boolean
false
true

<< enumeration >> TipoDocumento
dni
libretaCivica
libretaEnrolamiento

<< enumeration >> Estado
desocupado
activado
error

utn frlp ds 

Visibilidad de atributos y operaciones

¿Cómo podemos proteger los datos?

El encapsulado es el principio para ocultar datos
(*data hiding*)

La visibilidad puede ser:

- Privada (-):** acceden los objetos donde está definido el atributo u operación.
- Protegida (#):** acceden objetos de las subclases.
- Pública (+):** acceden objetos de cualquier clase.

utn frlp ds 

Clase: nombre de la clase

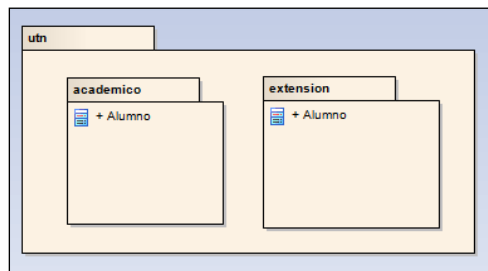
Sintaxis

Nombre simple: **NombreDeClase**
Nombres de camino: **paq1::paq2::NombreDeClase**

Ejemplos

Alumno

utn :: academico :: Alumno



utn frlp ds

Clase: Atributo

Sintaxis

[visibilidad] nombre [multiplicidad] [:tipo] [=valor inicial] [{lista de propiedades}]

Propiedades:

changeable
addOnly
frozen

Ejemplos:

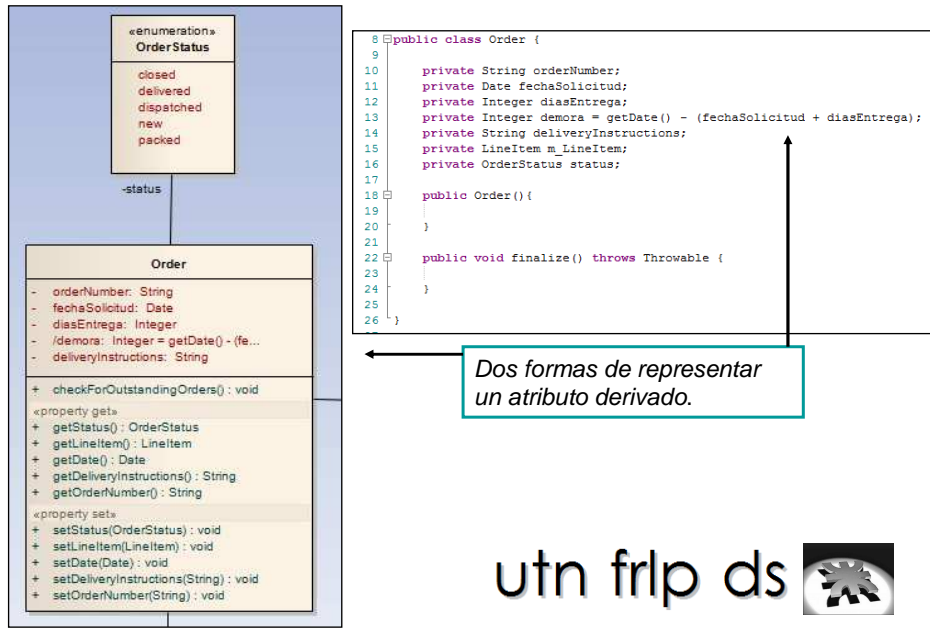
ObraTeatral
id
nombre
responsable
genero

ObraTeatral
- id: Integer {frozen}
+ nombre:String
responsable [1..2]:String
genero: String=musical

```
public class ObraTeatral {  
    private final int id;  
    public String nombre;  
    public java.util.Collection responsable;  
    protected String genero = musical;  
    //código  
}
```

utn frlp ds

Clase: Atributo derivado



Organización de atributos y operaciones

ObraTeatral
<<constructor>> nuevaRep(); nuevaRep(nombre:String)
<<consultas>> nombreRep()

Para organizar listas largas de atributos y operaciones, se pueden utilizar estereotipos. Se antepone a cada grupo una categoría descriptiva.

Usamos el estereotipo **<<accessors>>** para caracterizar (getters y setters) a través de los cuales podemos controlar el acceso a las variables de instancia.

Otros ejemplos en un modelo de BD : <<columna>> <<Pk>> <<FK>>

Clase: Operaciones

Sintaxis – signature

[visibilidad] nombre ([lista de parámetros]) [:tipo de retorno] [{propiedades}]

Declaración de un parámetro:

[dirección] nombre: tipo [multiplicidad][=valor]

- Donde dirección puede ser:
 - **in**: la operación puede modificar el parámetro y el que llama no necesita volver a verlo.
 - **out**: la operación coloca o cambia el parámetro y lo devuelve al que llama.
 - **inout**: la operación utiliza el parámetro y puede cambiarlo para devolverlo.

utn frlp ds 

Clase: Operaciones

Propiedades

- isQuery
- sequential
- guarded
- concurrent

Las operaciones isQuery difieren en complejidad a las de modificación y persistencia.
Sequential, guarded y concurrent refieren a la secuencia y forma de ejecución de varios envíos de un mensaje

Ejemplos

ObraTeatral
nueva() nombre() estaVigente()

ObraTeatral
+nueva(in nombre:String) #nombre() estaVigente(): Boolean {isQuery}

```
public class ObraTeatral {  
  
    public nueva(String nombre) {  
        //código  
    }  
    protected void nombre() {  
        //código  
    }  
    public Boolean estaVigente() {  
        //código  
        return null;  
    }  
}
```

utn frlp ds 

Clase: responsabilidades

¿Qué es responsabilidad de una clase?

Es un contrato u obligación de una clase.

Descripción textual.

Ejemplo

CuentaCorriente
depositar() extraer()
--notificar saldo deudor --permitir depósitos -- permitir extracciones --rechazar extracciones c/saldo deudor encima del límite --notificar a Gte. rojo no cubierto

utn frlp ds 

Diagrama de clases: relaciones

¿Qué son las relaciones entre clases?

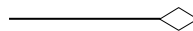
Una relación es una **conexión** semántica entre objetos. Proveen un camino de **comunicación** entre ellos.

- **Notación:**

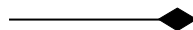
- Asociación



- Agregación



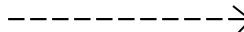
- Composición



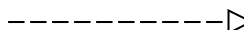
- Generalización



- Dependencia



- Realización



utn frlp ds 

Diagrama de clases: relaciones

¿Qué es una Asociación?

Es una relación **estructural** que especifica que los objetos de un elemento están conectados con los objetos de otro.



Rol: son las caras que presentan las clases a las demás.

Navegabilidad (Conocimiento): sirve para limitar la navegación a una sola dirección.

utn frlp ds 

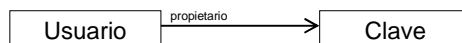
Diagrama de clases: asociación - propiedades

Ejemplo:



```
public class Espectador {
    public Reserva reserva;
    //código
}
```

```
public class Reserva {
    public Espectador propietario;
    //código
}
```



```
public class Usuario {
    public Clave clave;
    //código
}
```

```
public class Clave {
    //código
}
```

utn frlp ds 

Diagrama de clases: asociación - propiedades

Multiplicidad: indica “cuántos” objetos pueden conectarse a través de una instancia de una asociación.

Se puede indicar una multiplicidad de:

Exactamente Uno: A — 1 B

Cero o Uno: A — 0..1 B


Muchos: A — 0..* B

Uno o Más: A — 1..* B

Número Exacto: A — 3 B

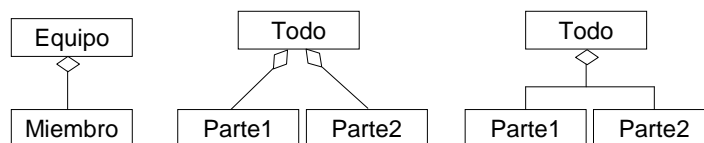
Lista: A — 0..1,3,5,7,9 B

Rango: A — 3..6 B

utn frlp ds 

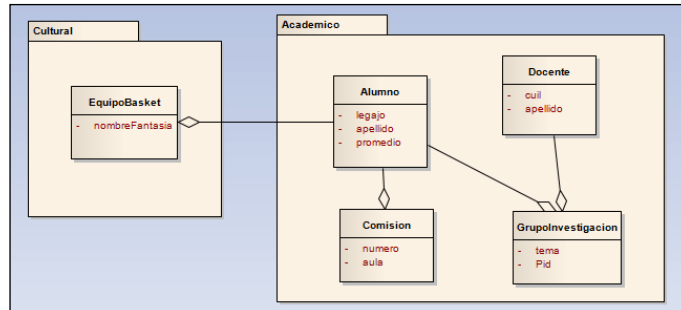
Asociación: Agregación

Agregación: es una asociación especial, una relación del tipo “todo/parte” dentro de la cual una o más clases son partes de un conjunto.



utn frlp ds 

Asociación: Agregación



```
public class GrupoInvestigacion {
    public Docente docente;
    public Alumno alumno;
    //código
}
```

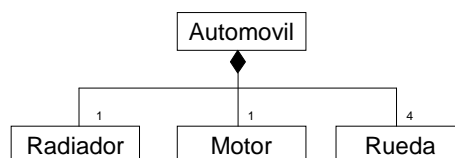
```
public class Alumno {
    public GrupoInvestigacion grupoInvestigacion ;
    public Comision comision;
    .....
    //código
}
```

utn frlp ds 

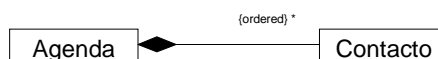
Asociación: Composición

Composición: es una forma de agregación, con fuerte sentido de posesión y tiempo de vida coincidentes de las partes con el conjunto.

Una parte puede pertenecer solamente a una composición (un *todo*). Cuando el todo desaparece, también lo hacen sus partes.

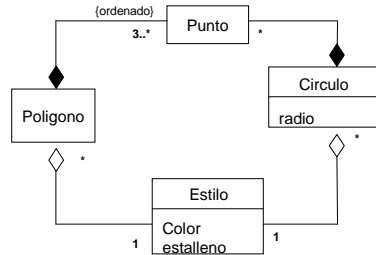


Analizamos otro ejemplo:



utn frlp ds 

Asociación: Composición



Reflexionar sobre las condiciones de validez del modelo presentado.

```

public class Punto {
    public Circulo circulo;
    public Poligono poligono;
    //código
}
    
```

```

public class Circulo {
    public Float radio;
    public Estilo estilo;
    public java.util.Collection punto = new java.util.TreeSet();
    //código
}
    
```

```

public class Poligono {
    public Estilo estilo;
    public java.util.Collection punto = new java.util.TreeSet();
    //código
}
    
```

```

public class Estilo {
    public String color;
    public Boolean estalleno;
    public java.util.Collection circulo = new java.util.TreeSet();
    public java.util.Collection punto = new java.util.TreeSet();
    //código
}
    
```

utn frlp ds

Asociación: Visibilidad

Visibilidad:

Sirve para limitar la visibilidad a través de esta asociación relativa a los objetos externos a ella.

Privada (-)
Protegida (#)
Pública (+)



```

public class GrupoUsuarios {
    public java.util.Collection usuario = new java.util.TreeSet();
    //código
}
    
```

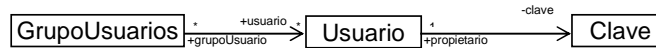
```

public class Usuario {
    public java.util.Collection grupoUsuarios = new java.util.TreeSet();
    private Clave clave;
    //código
}
    
```

utn frlp ds

Asociación: Visibilidad

Visibilidad: sirve para limitar la visibilidad a través de esta asociación relativa a los objetos externos a ella.



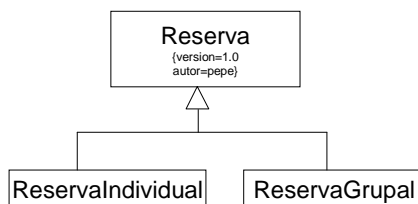
```
public class GrupoUsuarios {
    public java.util.Collection usuario = new java.util.TreeSet();
    //código
}
```

```
public class Usuario {
    private Clave clave;
    //código
}
```

utn frlp ds 

Generalización

Es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo).



```
public class Reserva {
    /*
    *version=1.0
    *autor=pepe
    */
    //código
}
```

```
public class ReservaIndividual extends Reserva {
    //código
}
```

```
public class ReservaGrupal extends Reserva {
    //código
}
```

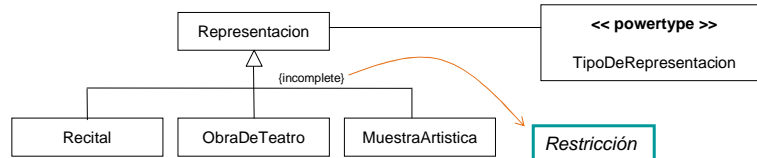
Observar cómo se transforma un valor etiquetado en la representación del código.

utn frlp ds 

Generalización

En UML puede enriquecerse el diagrama de Herencia (Generalización) con discriminantes y restricciones

Ejemplos:



El estereotipo <<implementation>> especifica que el hijo hereda la implementación del padre, pero no hace públicas ni soporta sus interfaces, y por ello viola la semántica de sustitución.

Restricciones:

complete: no se permiten hijos adicionales

incomplete: permite hijos adicionales

En contexto de herencia múltiple

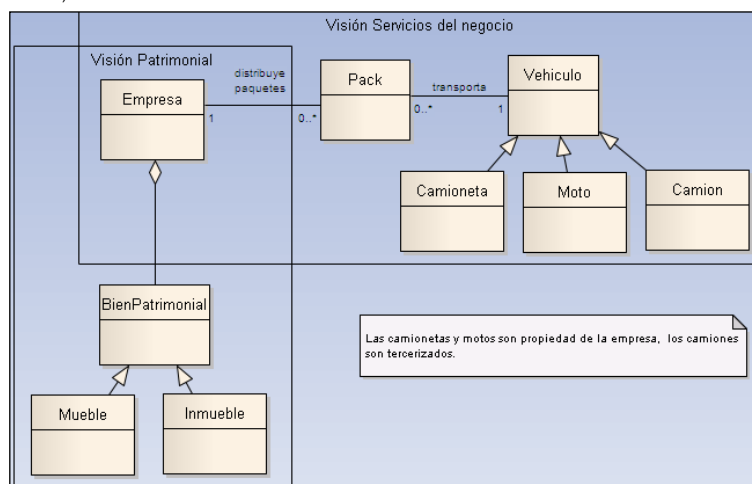
disjoint: los objetos del padre no pueden tener más de uno de los hijos como tipo.


overlapping: los objetos del padre pueden tener más de uno de los hijos como tipo.

utn frlp ds 

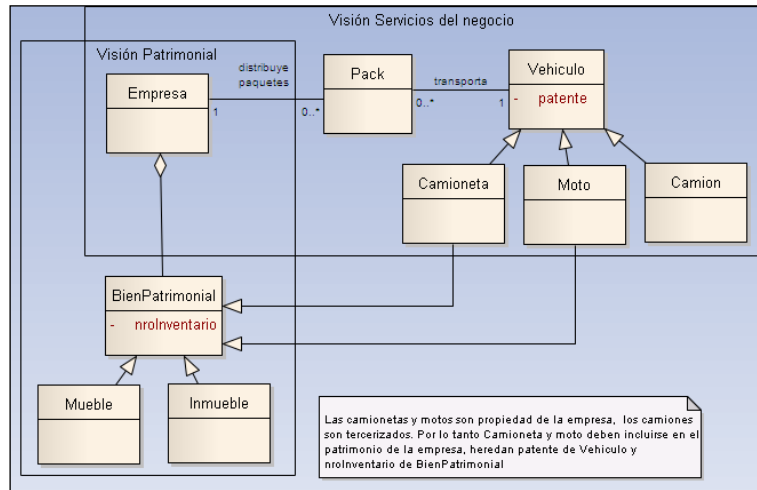
Herencia múltiple - Construimos un ejemplo

Empresa de logística: presentación del modelo de clases integrando dos perspectivas de la empresa. (Primera aproximación)



utn frlp ds 

Herencia múltiple - Construimos un ejemplo

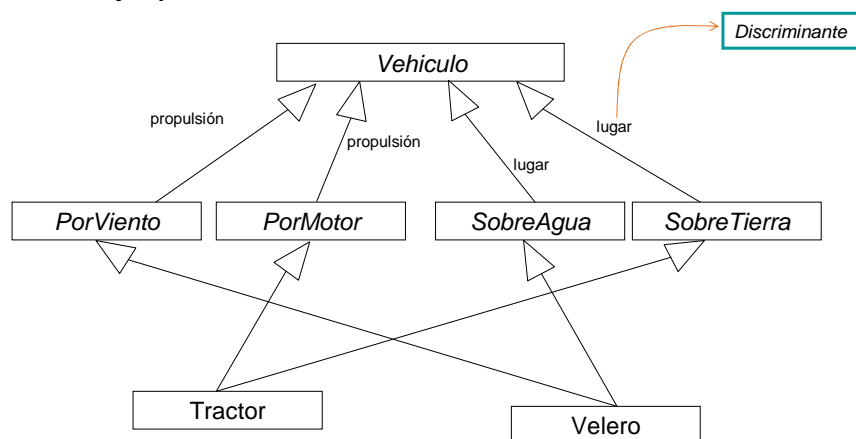


Modelo refinado (segunda aproximación)

utn frlp ds 

Herencia múltiple

Otros ejemplos:



Identificar y diferenciar Clases abstractas y concretas.

utn frlp ds 

Herencia múltiple

Mismo ejemplo, otra representación:

```

public abstract class Vehiculo {
    private lugar;
    private propulsion;
    //código
}

public abstract class PorViento extends Vehiculo {
    //código
}

public abstract class PorMotor extends Vehiculo {
    //código
}

public abstract class SobreTierra extends Vehiculo {
    //código
}

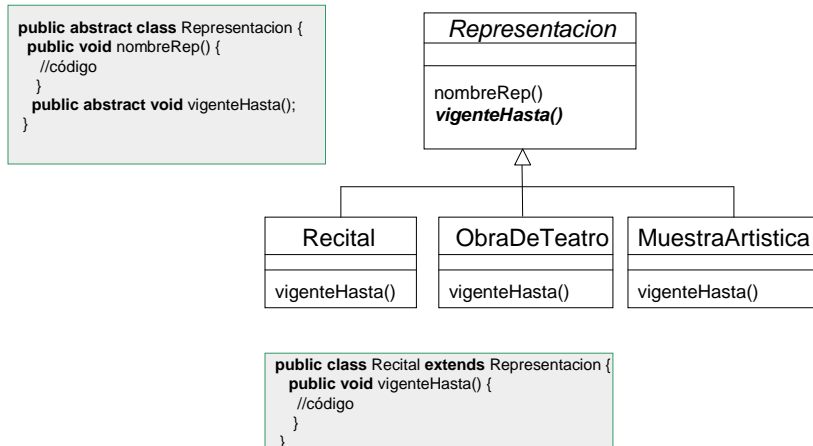
public abstract class SobreAgua extends Vehiculo {
    //código
}

public class Velero extends PorViento, SobreAgua {
    //código
}

public class Tractor extends PorMotor, SobreTierra {
    //código
}
    
```

utn frlp ds 

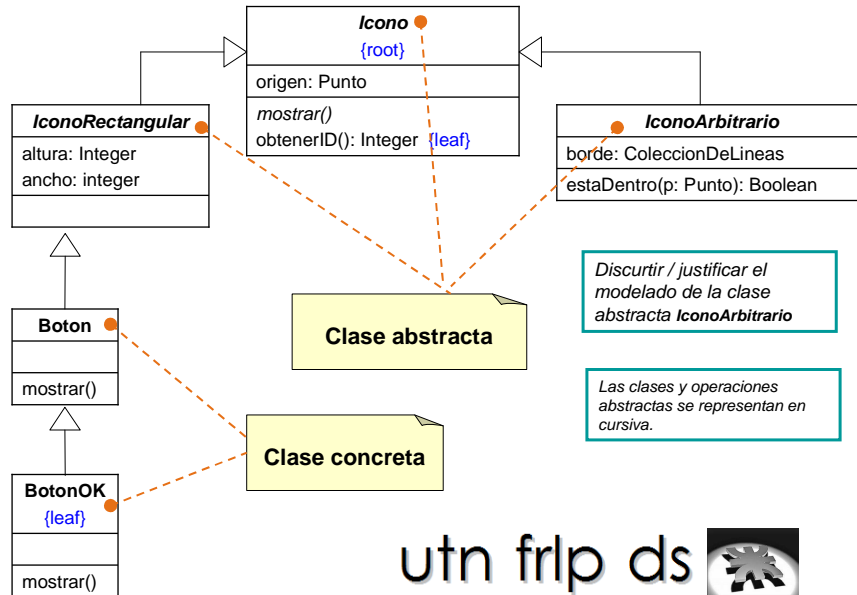
Herencia (abstracción de clases y operaciones)



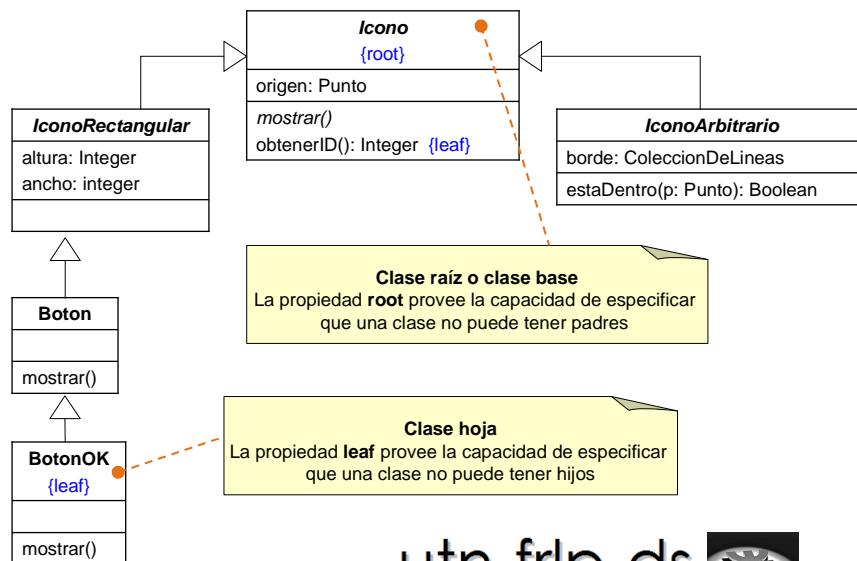
Una clase abstracta es aquella que funciona como base de la herencia

utn frlp ds 

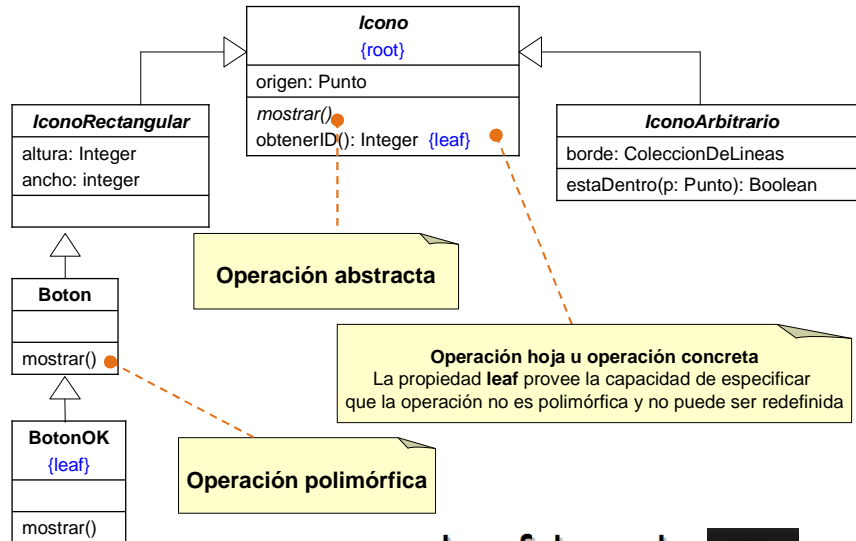
Definiendo clase raíz y clases hojas



Definiendo clase raíz y clases hojas



Definiendo clase raíz y clases hojas

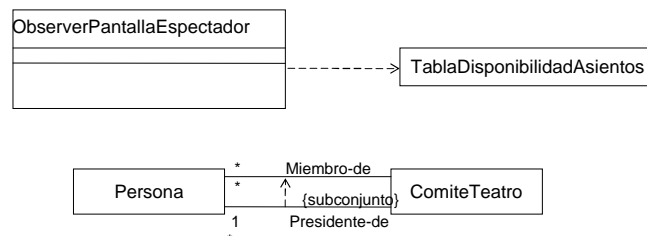


utn frlp ds 

Dependencia

¿Qué es una Dependencia?

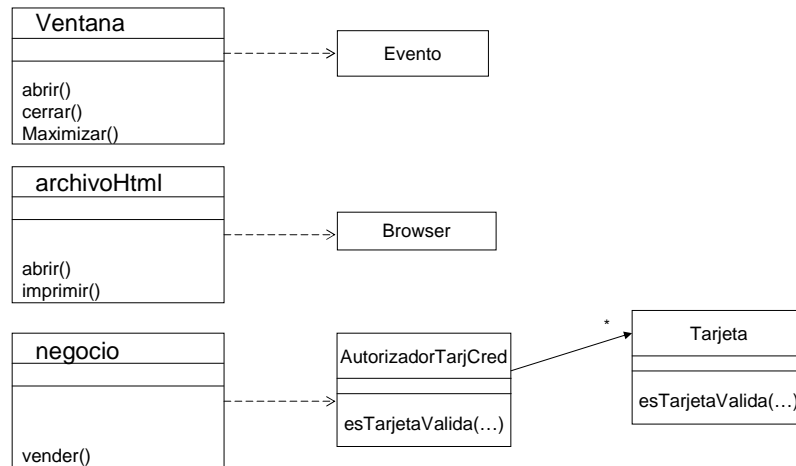
Es una relación de **uso** que declara que un cambio en la especificación de un elemento puede afectar a otro elemento que la utiliza, pero no necesariamente a la inversa.



utn frlp ds 

Dependencia

Otros ejemplos:

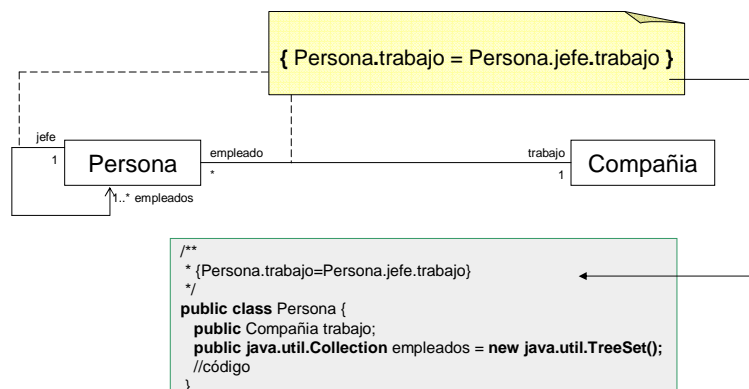


utn frlp ds 

Anotaciones

¿Qué es una nota?

Es un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos



Interpretación grupal del contenido de la nota


utn frlp ds 

Diagrama de clases: interfaces

¿Qué es una Interfaz?

Es una colección de operaciones que se utiliza para especificar un servicio de una clase o componente.

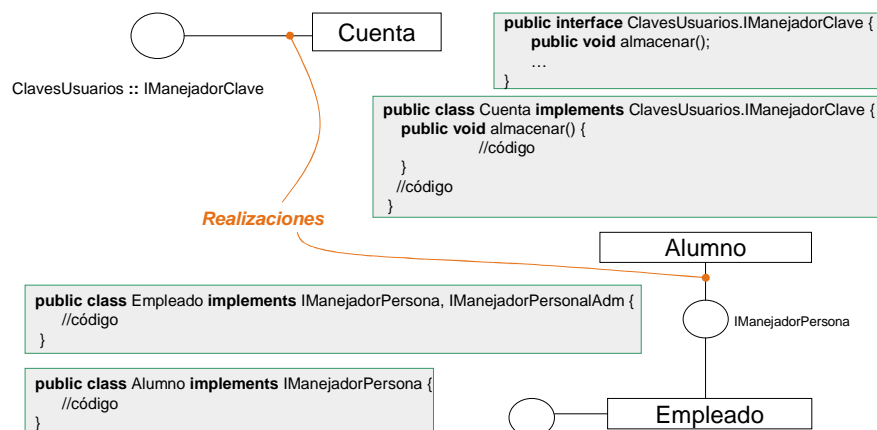
Notación

- Nombre:** sirve para distinguirla del resto de las interfaces.
- Operaciones:** se utilizan para especificar un servicio.
- Relaciones:** participan en las relaciones de asociación, generalización y dependencia.

utn frlp ds 

Diagrama de clases: interfaces

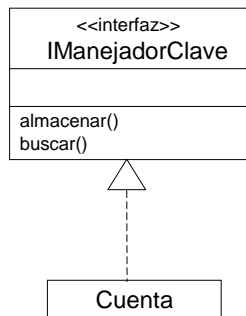
Notación simple para Interfaces:



utn frlp ds 

Interfaces: Realización

Notación expandida para Interfaces



```
public interface IManejadorClave {
    public void almacenar();
    public void buscar();
    ...
}
```

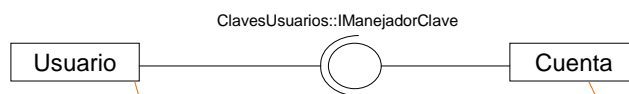
```
public class Cuenta implements IManejadorClave {
    public void almacenar(){
        //código
    }
    public void buscar(){
        //código
    }
    //código
}
```

La clase Cuenta realiza (implementa) la interfaz IManejadorClave

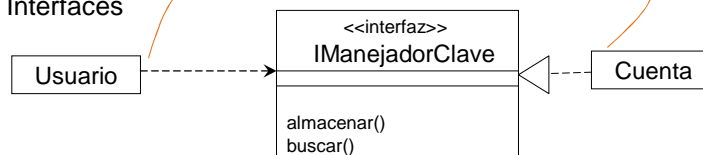
utn frlp ds 

Interfaces: Realización

Notación simple



Notación expandida para Interfaces



utn frlp ds 

Interfaces en el Diagrama de Clases

Las interfaces se diferencian de las clases abstractas porque ellas no representan un concepto abstracto, solamente manifiestan un conjunto de operaciones soportadas por los objetos que la realicen.

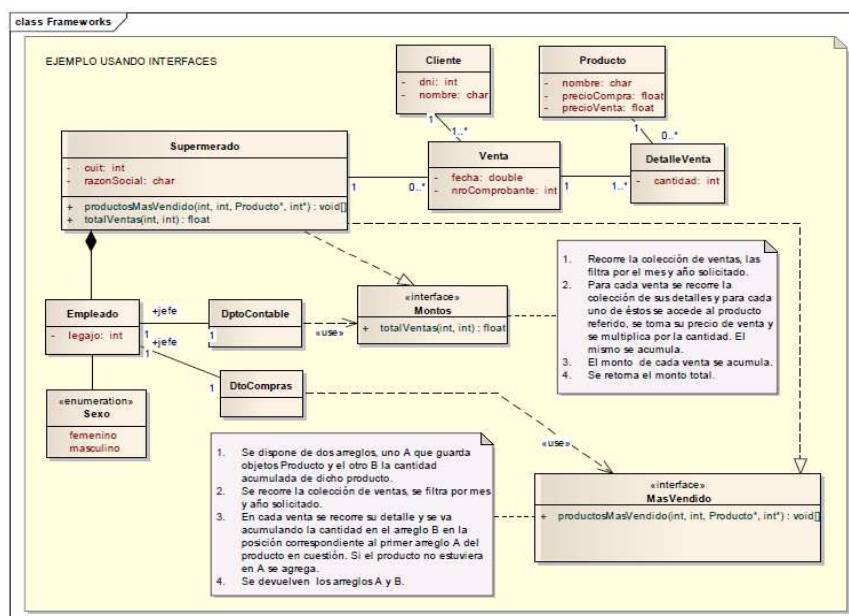
Su diferencia puede decirse que es semántica.

Notar que en las clases de las interfaces, todas las operaciones aparecen sin implementación.

Las clases tienen atributos, las interfaces no!

“Se presenta un ejemplo de aplicación de interfaces”

utn frlp ds 



utn frlp ds 

Caso práctico...

Se solicita modelar el suministro de medicamentos de una Obra Social. Existe un catálogo compuesto de medicamentos del cual se necesita guardar la fecha de edición.

Los medicamentos tienen nombre comercial, precio de compra y venta. Además se sabe la droga principal que contiene (genérico), éste está codificado en un nomenclador. Cada medicamento está vinculado a varias enfermedades para el cual está indicado y varias en las cual está contraindicado.

Cada medicamento tiene un único proveedor que puede ser un laboratorio o farmacia, ambos tienen razón social y cuit, en el primer caso además requiere código de industria y en el segundo el farmacéutico responsable.

Cada suministro es la entrega de un medicamento, se quiere conocer el nombre y apellido del afiliado, cantidad en unidades y fecha de dicha entrega.

De cada medicamento además se conoce si es de venta libre o requiere receta.

Identificar Clases mediante método Larman

utn frlp ds 

Caso práctico...

Se solicita modelar el suministro de medicamentos de una **Obra Social**. Existe un **catálogo compuesto** de **medicamentos** del cual se **necesita** guardar la **fecha de edición**.

Los medicamentos **tienen nombre comercial, precio de compra y venta**. Además se sabe la **droga** principal que **contiene** (genérico), éste **está** codificado en un **nomenclador**. Cada medicamento está **vinculado** a varias **enfermedades** para el cual **está indicado** y varias en las cual está **contraindicado**.

Cada medicamento **tiene** un único **proveedor** que puede ser un **laboratorio** o **farmacia**, ambos tienen **razón social y cuit**, en el primer caso además requiere **código de industria** y en el segundo el **farmacéutico responsable**.

Cada **suministro** es la **entrega** de un medicamento, se quiere conocer el **nombre y apellido** del **afiliado**, **cantidad** en unidades y **fecha** de dicha entrega.

De cada medicamento además se conoce si es de **venta libre** o requiere **receta**.

Clases candidatas, relaciones y atributos en diferentes colores. Construcción grupal.

utn frlp ds 

Más casos prácticos...

Recordamos...
Software Requirement
Specification

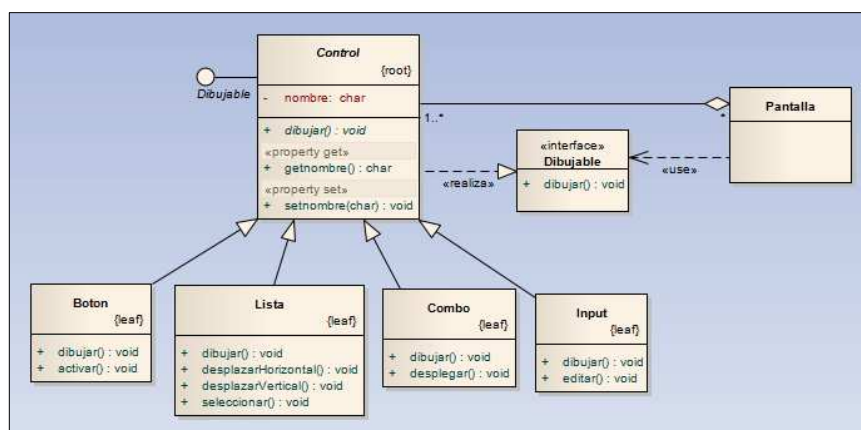
Nos solicitaron un modelo de clases que represente la siguiente SRS:

“Los controles de una pantalla gráfica que pueden ser: botón, lista desplegable, combo e input (caja de texto). Todos los controles pueden dibujarse aunque cada uno tiene su manera especial de hacerlo en la pantalla ya que se le dibujan características diferentes. El botón además se puede activar. La lista se puede desplazar verticalmente y horizontalmente y se puede seleccionar. El combo se puede desplegar y la caja de texto se puede editar (ingresar contenido). La Pantalla es la encargada de dibujar los controles en ella. Los controles además responden al nombre del Control.”

utn frlp ds 

Una posible solución...

Solución gráfica



utn frlp ds 

Una posible solución...

Solución en código Java

```
public abstract class Control implements Dibujable {
    private char nombre;
    public Control(){
    }
    public void finalize() throws Throwable {
    }
    public abstract void dibujar();
    public char getnombre(){
        return nombre;
    }
    public void setnombre(char newVal){
        nombre = newVal;
    }
}
```

```
public final class Boton extends Control {
    public Boton(){
    }
    public void finalize() throws Throwable {
        super.finalize();
    }
    public void dibujar(){
    }
    public void activar(){
    }
}
```

```
public interface Dibujable {
    public void dibujar();
}
public class Pantalla {
    public Control m_Control;
    public Pantalla(){
    }
    public void finalize() throws Throwable {
    }
}
```

utn frlp ds



Y más casos prácticos...

Clasificar las siguientes relaciones como generalizaciones, agregaciones o asociaciones. Modelarlas y discutir opciones.

- Cada país tiene una ciudad capital.
- Un usuario tiene permiso de lectura sobre un archivo.
- Un archivo puede ser un directorio.
- Una tabla contiene registros.
- Un polígono está compuesto por un conjunto ordenado de puntos.
- En un editor gráfico los objetos pueden ser cuadros de texto, objetos geométricos o agrupamientos.
- Una persona programa en un lenguaje para cierto proyecto.
- Modems y teclados son dispositivos de entrada/salida.
- Las clases pueden tener múltiples atributos
- Un jugador participa de un cierto equipo por temporada.

utn frlp ds

