Guía del Modelo de Dominio Textual de OOH4RIA

Diseño de Sistemas Software

2015-2016

Índice

- Creación de un proyecto
  - Modelo de dominio Textual
  - Clase de Domino
  - Atributos
  - Operaciones (CRUD, Custom y Customized)
  - Relaciones de asociación y herencia
  - Tipos Enumerados
  - Generación de código
- Ejercicio con el modelo textual

Características principales

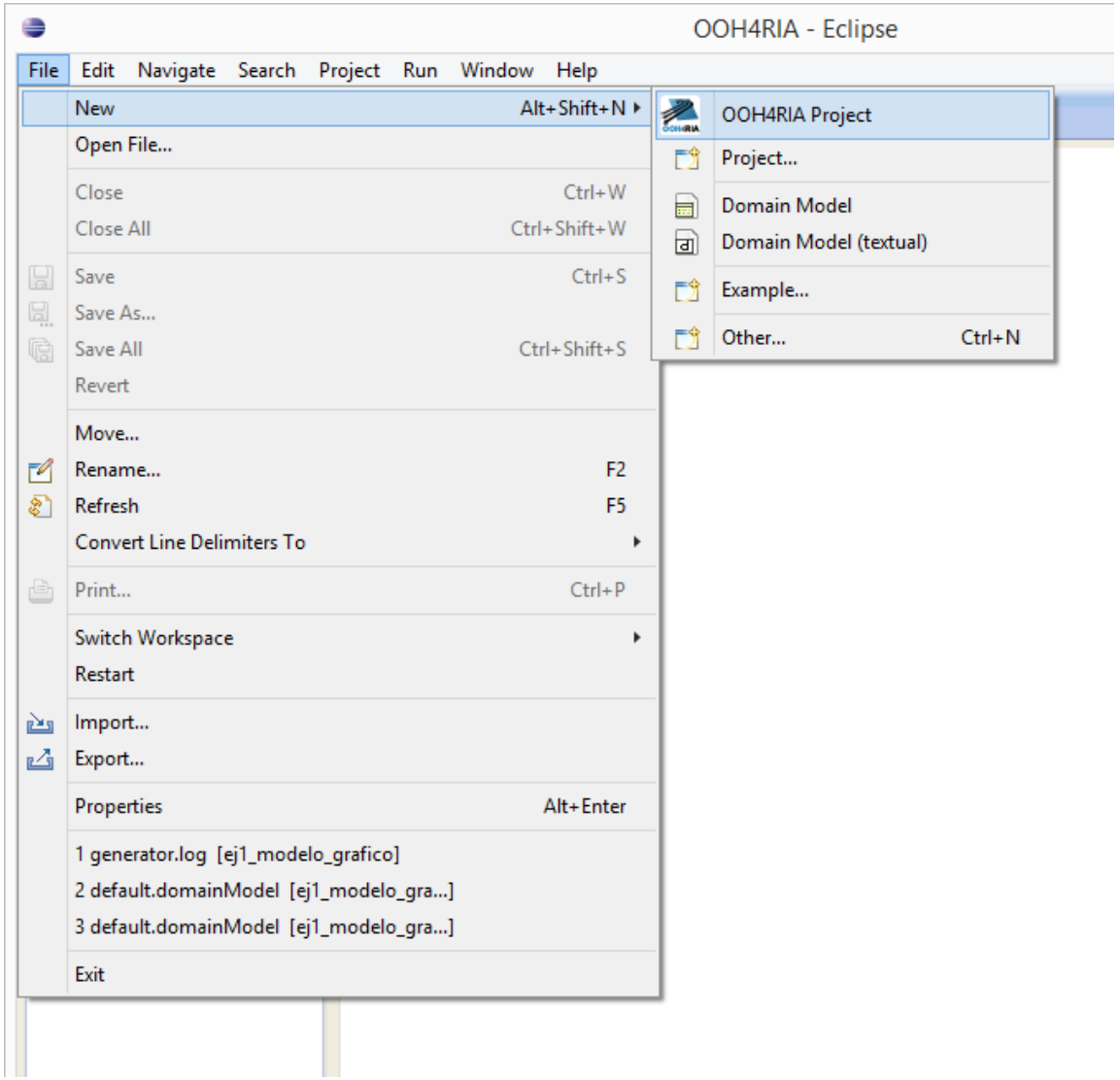
- OOH4RIA proporciona para su modelo de dominio una sintaxis textual con un sencillo lenguaje orientado a objetos equivalente semánticamente a la notación gráfica
- La notación textual proporciona mecanismos de plantillas para acelerar la escritura, autocompletado para elegir valores válidos (todos se activan con control + espacio)
- Posee mecanismos de validación (igual que la notación gráfica) para aspectos semánticos del propio modelo.
- Permite la misma generación de código de una lógica de negocio basada en NHibernate

Creación de un proyecto OOH4RIA

- Seleccionamos la perspectiva de OOH4RIA
- Vamos al menú File y seleccionamos

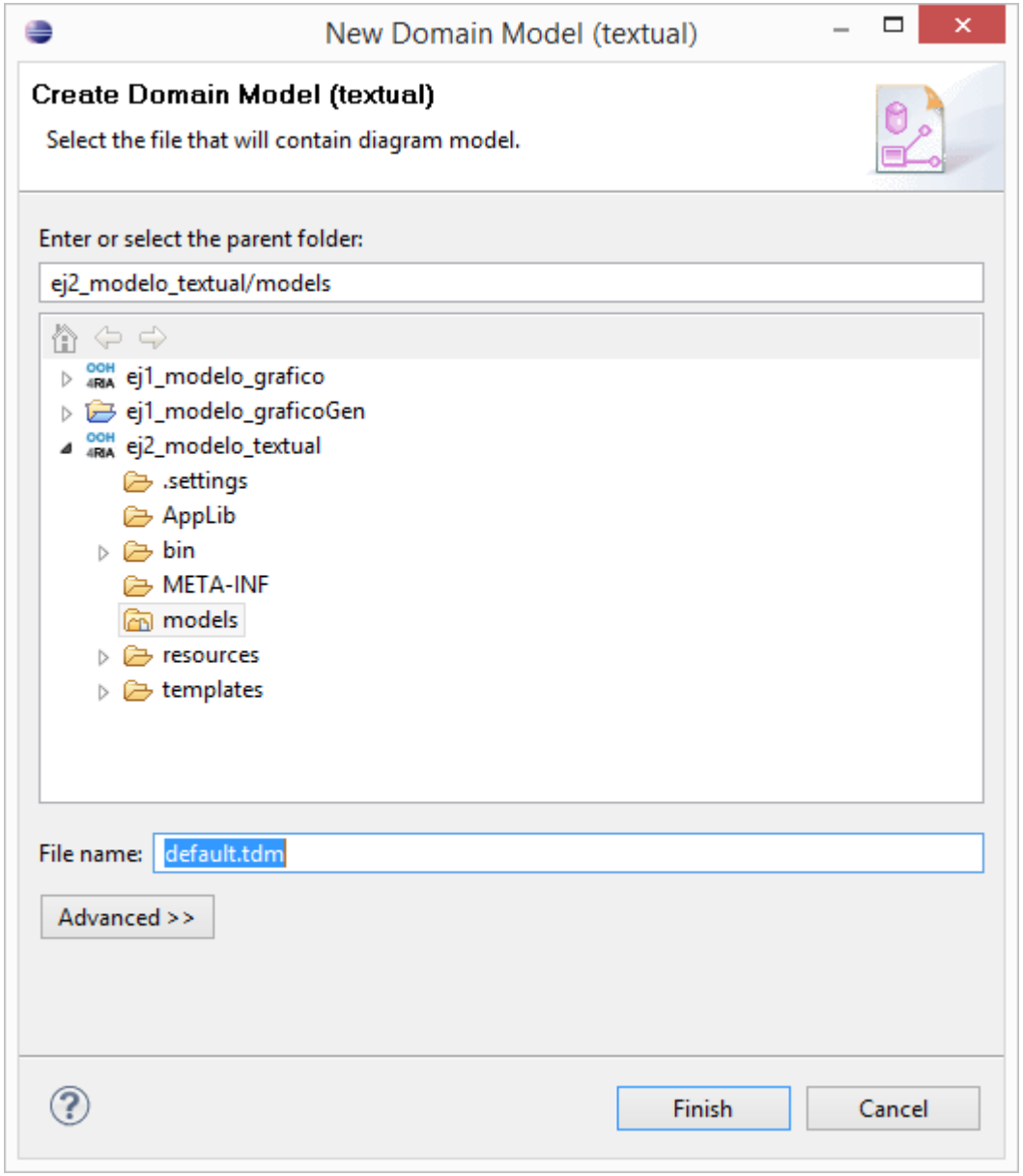
New -> OOH4RIA Project

- Creamos un proyecto nuevo para el modelo textual

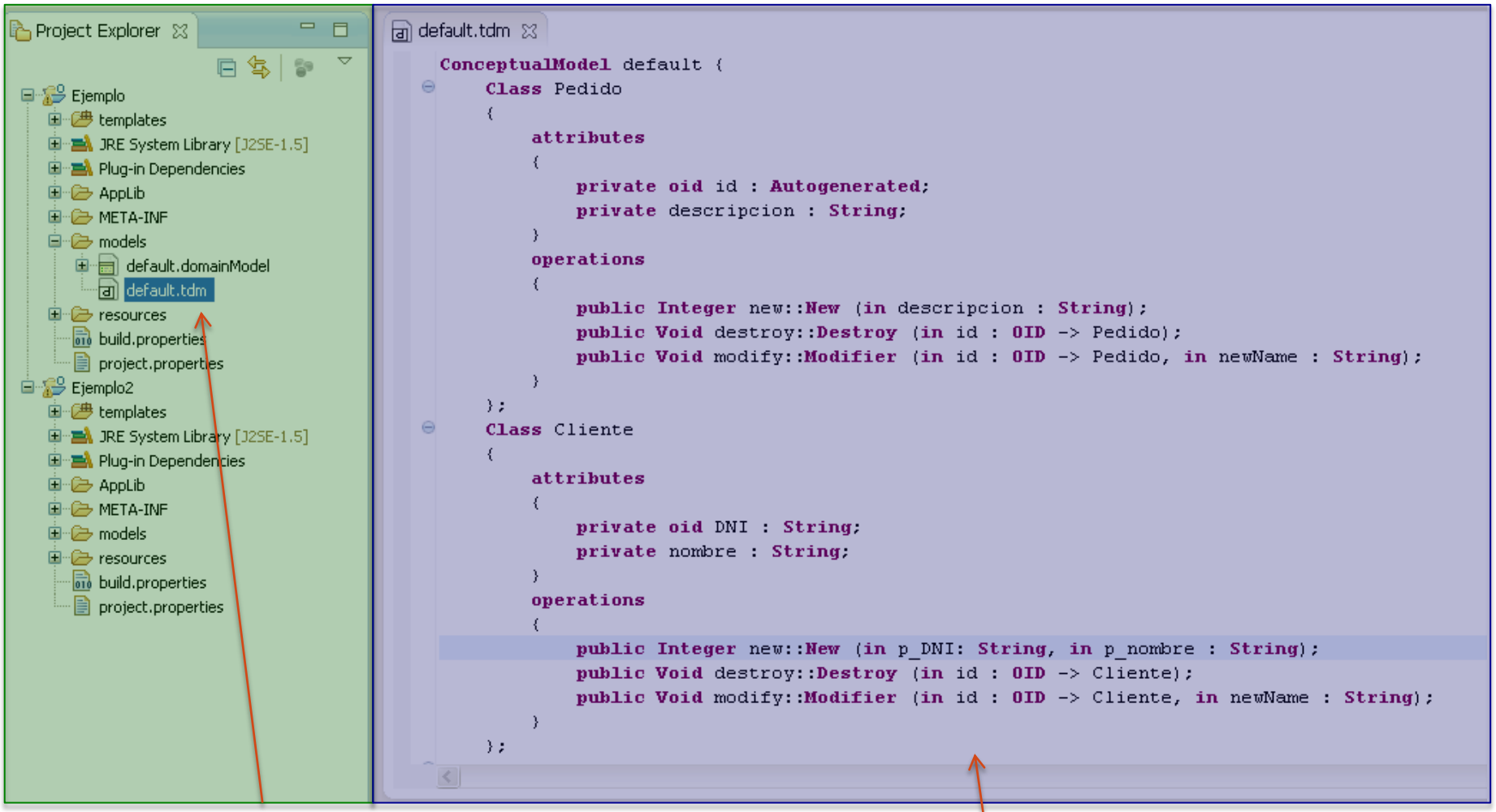


Creamos un modelo de dominio textual

- Seleccionamos el proyecto y se crea el modelo de dominio llamado por defecto **default.tdm (textual domain model)**
- La ubicación recomendada es la carpeta **models**



Representación del Modelo de dominio textual



Proyecto OOH4RIA que contiene el modelo de dominio textual (default.tdm)

Fichero de texto (.tdm) donde se define el modelo de dominio textual

Creación de una clase en el modelo textual

- Una clase se compone de atributos y operaciones
- Como ocurre en la notación gráfica, el atributo OID y la operación new son obligatorias en la clase

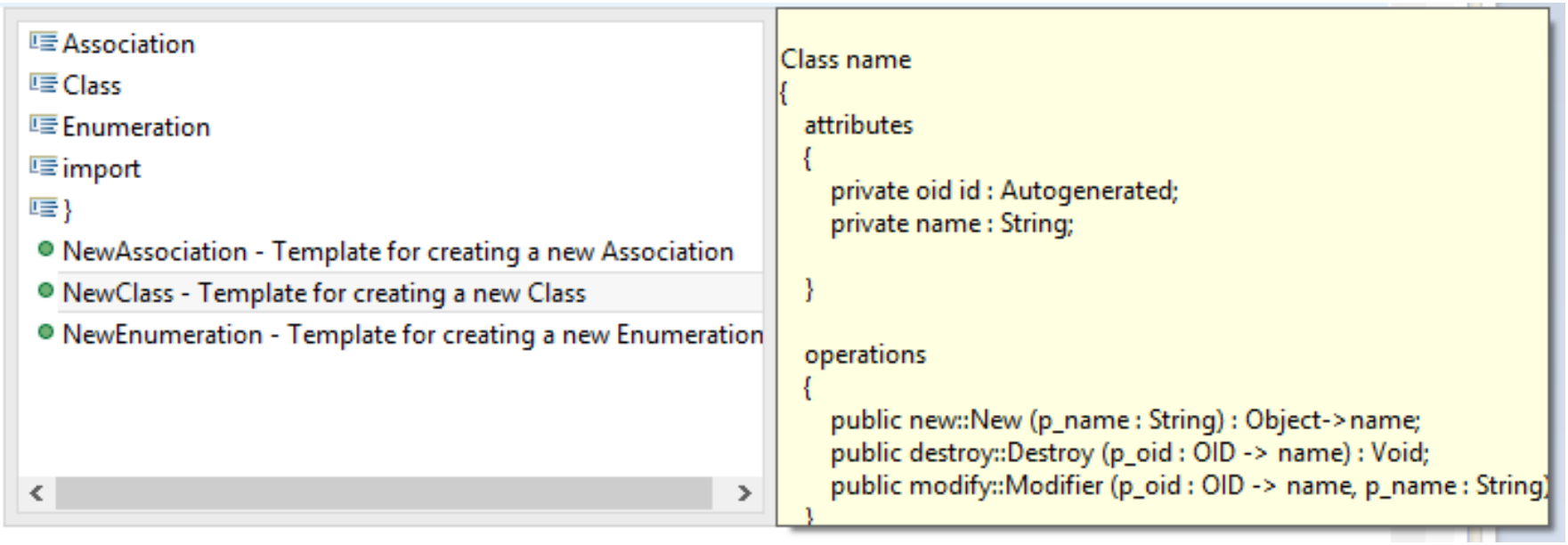
```
Class Customer
{
    attributes {
        private oid id : Autogenerated;
        private name : String;
        private surname : String;
        private postalCode : Integer;
        private telephone : String;
    }

    operations {
        public new::New (p_name : String, p_surname : String,
            p_postalCode : Integer, p_telephone : String) : Object->Customer;
        public modify::Modifier (p_oid : OID -> Customer, p_name : String, p_su
            p_postalCode : Integer, p_telephone : String) : Void;
        public destroy::Destroy (p_oid : OID -> Customer) : Void;
        public readAll::ReadAll (/* No arguments */) : List<Object->Customer>;
    }
};
```

Creación de la clase en el modelo textual

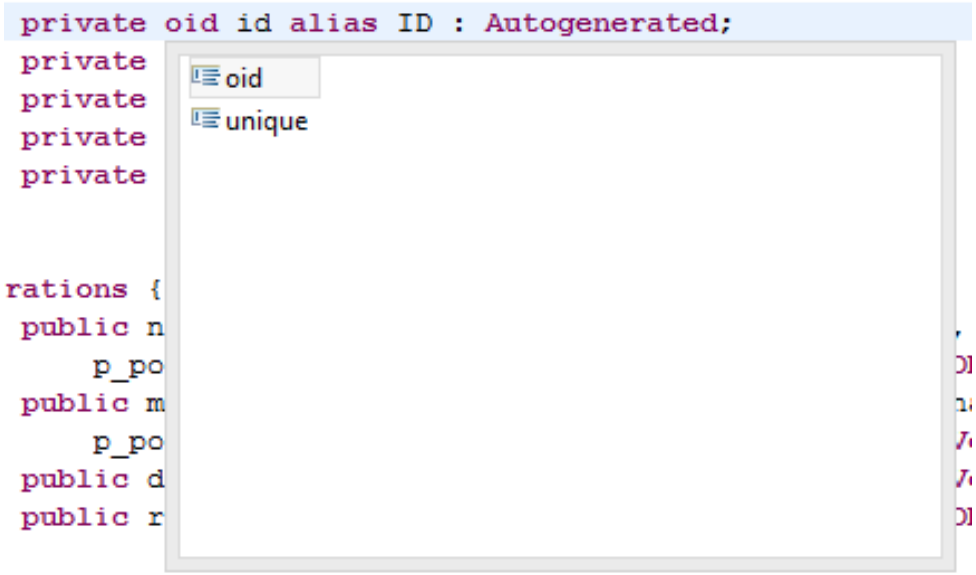
- Los templates o plantillas permiten introducir rápidamente el OID y operaciones de una clase
- Mediante el template NewClass creamos los atributos id (oid), name y las operaciones New, destroy y modify
- Las plantillas y el autocompletado salen con:

Control + Espacio



Creación del atributo

- Definición del atributo es muy similar a cualquier lenguaje orientado a objetos
  - La visibilidad se fija siempre a **private** por restricción de nhibernate (atributos privados y propiedades públicas)
  - Cada clase debe tener un atributo oid, y podrá tener N atributos **unique** (no admite nulos ni repetidos)



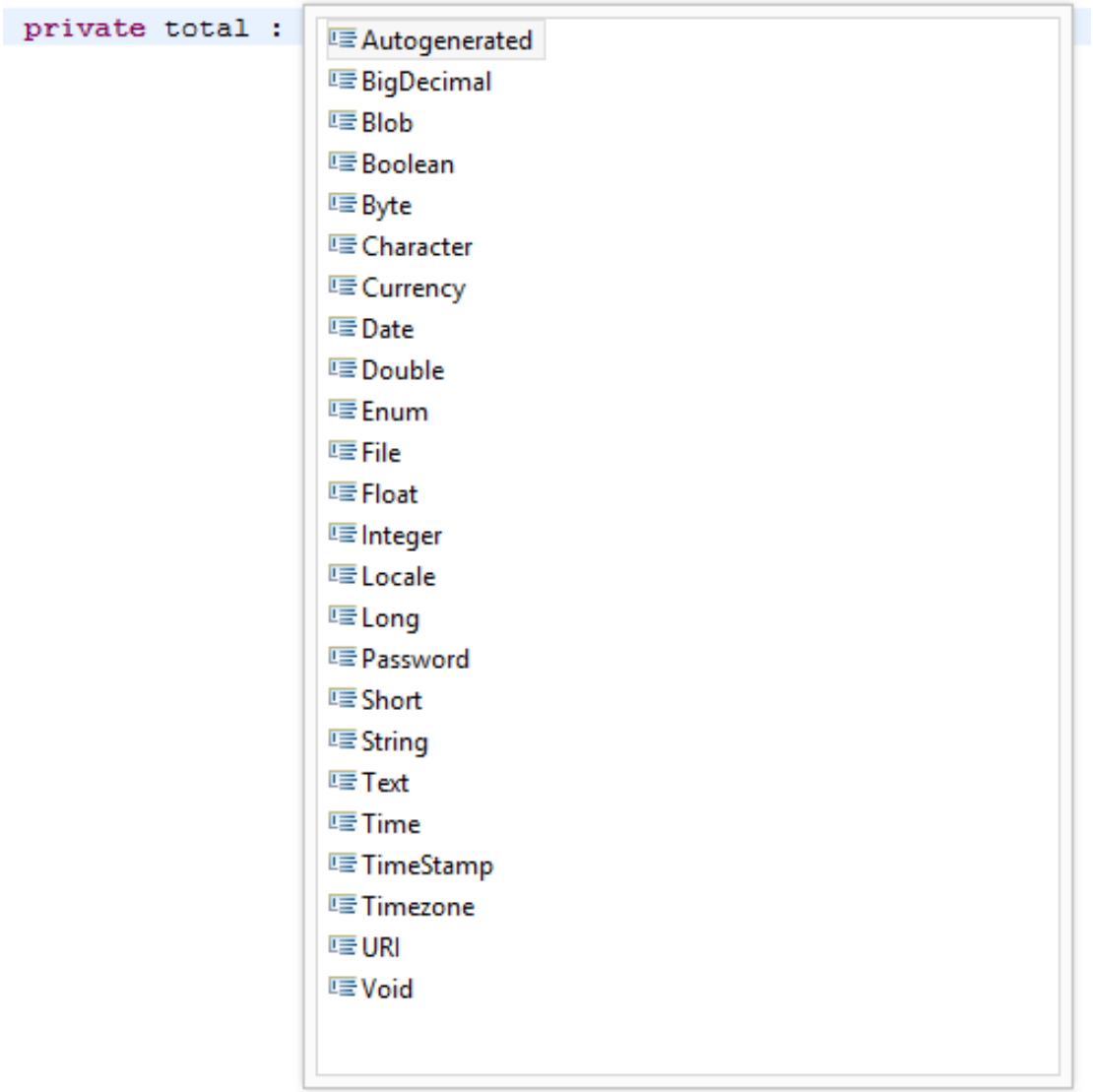
Creación del atributo

- Se puede expresar explícitamente la cardinalidad mínima (0 ó 1) para indicar que el atributo admite o no valores nulos
- La cardinalidad máxima (1 o *), podrá fijar si se trata de un único valor primitivo (1) o de una colección de valores primitivos (*) (teléfonos es una Lista de Strings)
- Por defecto, las cardinalidad son minima = 1 y máxima= 1 -> (“1”, “1”).

```
private unique dateSent : Date ("1","1");
private telephones : String ("1","*");
```

Tipos de datos

- Soporta los mismos tipos que el modelo gráfico



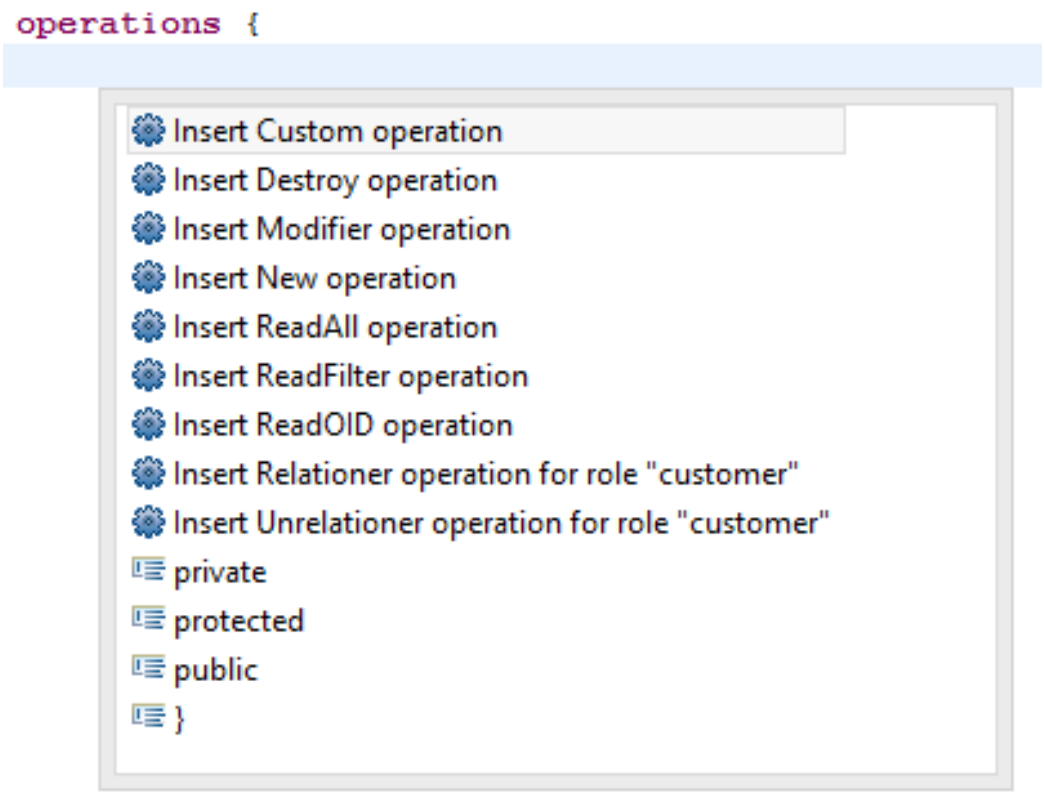
Creación de una Operación

- Seguimos una notación muy similar a un Lenguaje OO para definir la operación. La estructura es:
  - Visibilidad Nombre (::tipo Operación) argumentos : valor de retorno {filtro}
    - La visibilidad puede ser public o private (no hay herencia de comportamiento solo de estado)
    - El tipo de operación por defecto es custom

```
public readFilter::ReadFilter (customerId : String) : List<Object->CustomerOrder>
 { filter = "FROM CustomerOrderEN c where c.Customer = :customerId" };
```

Creación rápida de Operaciones

- Se ha introducido un mecanismo de autocompletado que permite crear las operaciones CRUD completas en función de su tipo



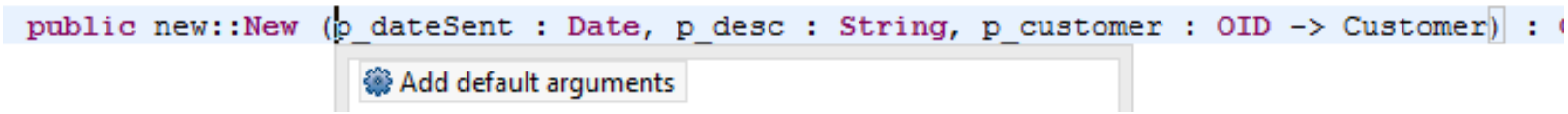
Customización de operaciones

- Se añade la palabra customized detrás del tipo de la operación CRUD
- Se genera entonces un archivo separado en C# que podremos modificar (el mismo mecanismo que hemos visto en el modelo visual)

```
public modify::Modifier customized (p_oid : OID -> CustomerOrder,
    p_dateSent : Date, p_telephones : List <String>,
    p_desc : String) : Void;
```

Actualización de las operaciones

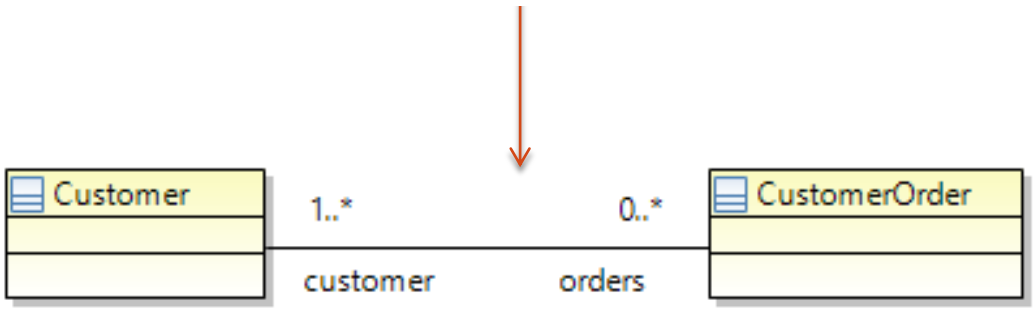
- Cuando actualicemos los atributos o roles de una clase. El new y el modifier no se actualizan solos (como en el visual)
- Debemos actualizar con el mecanismo de autocompletado “Add default arguments”



Relaciones de Asociación

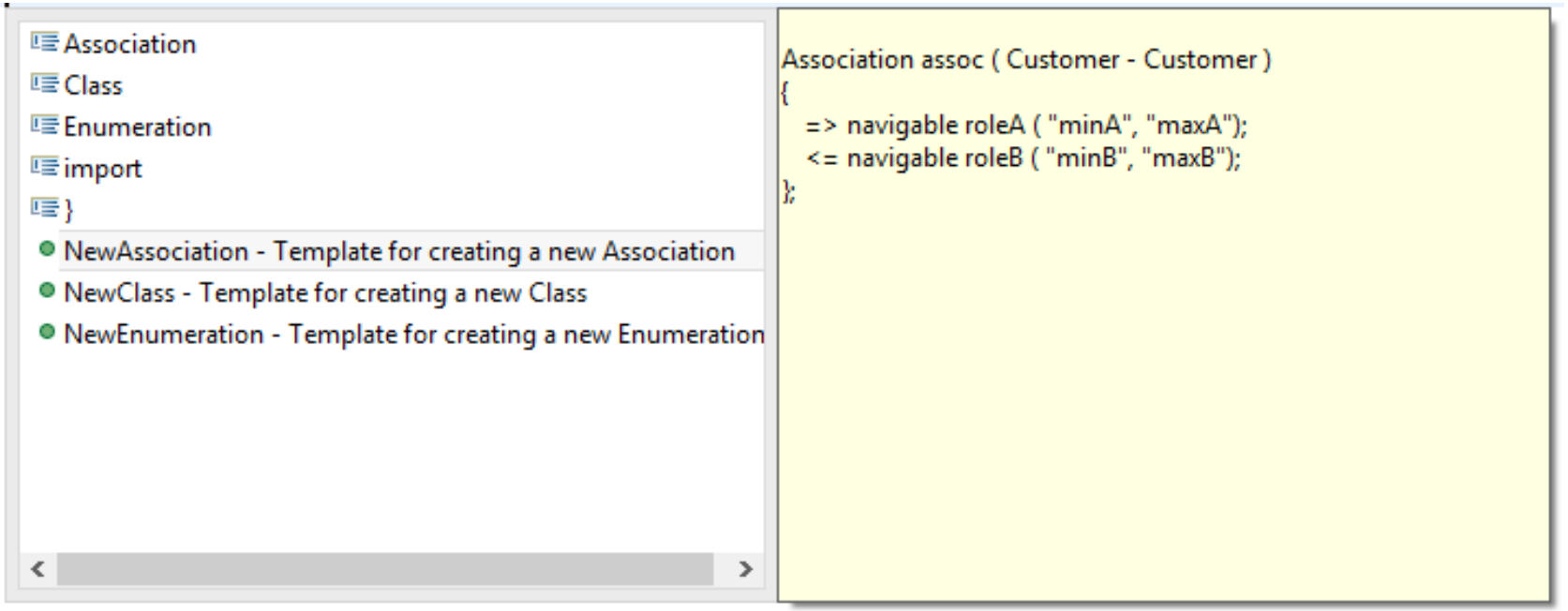
- Se define un elemento Association, que permite establece una relación de asociación (o agregación o composición) entre dos clases

```
Association OrderCustomer (CustomerOrder - Customer) {
    => navigable customer ("1", "1");
    <= navigable orders ("0", "*");
};
```



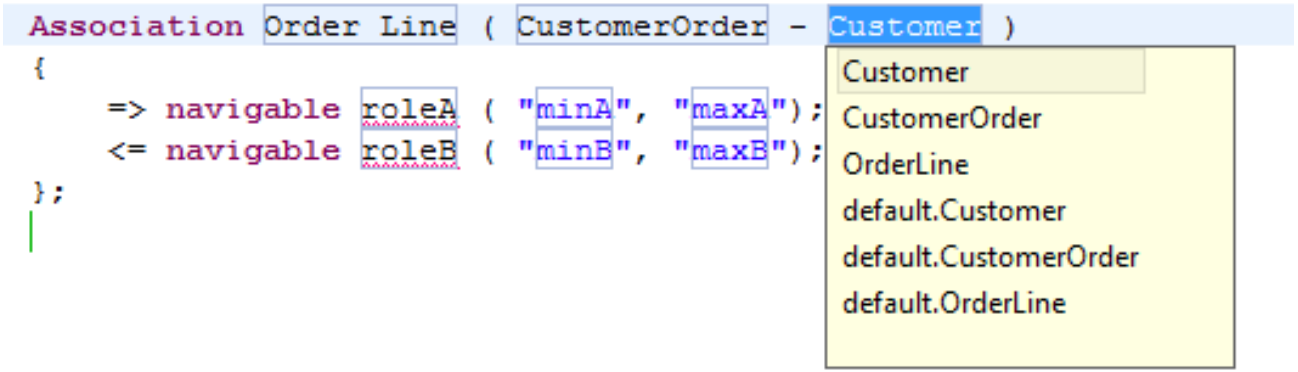
Creación de asociación

- Utilizando un template (control+Espacio), fuera de cualquier elemento (al mismo nivel de clase) se nos ofertará la plantilla para crear una asociación

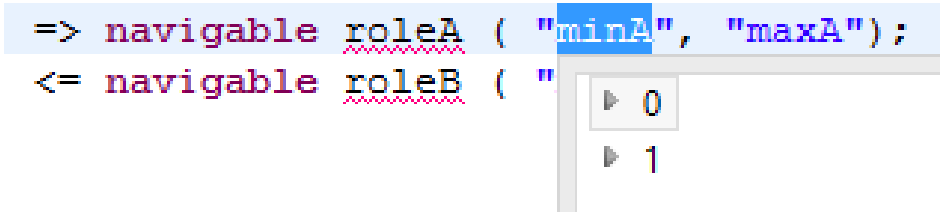


Creación de Asociación

- Se seleccionan las clases que se relacionan, mostrándose las clases definidas hasta ahora



- Las cardinalidad mínima (0 ó 1) y máxima (1 ó *) son también ofertadas mediante autocompletado

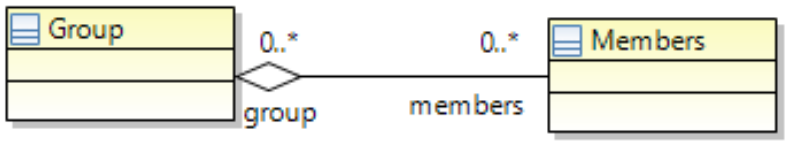


Relaciones de Asociación

- Por defecto, la relación es None (asociación). Si queremos establecer agregación o composición hay que explicitar después del nombre del rol los tipos :Share (agregación) ó Composite (composición)

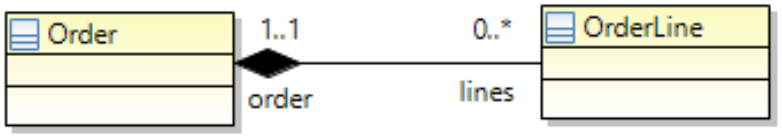
Agregación:

```
Association BelongsGroups ( Group - Member )
{
    => navigable members:Share ( "0", "*");
    <= navigable groups ( "0", "*");
};
```



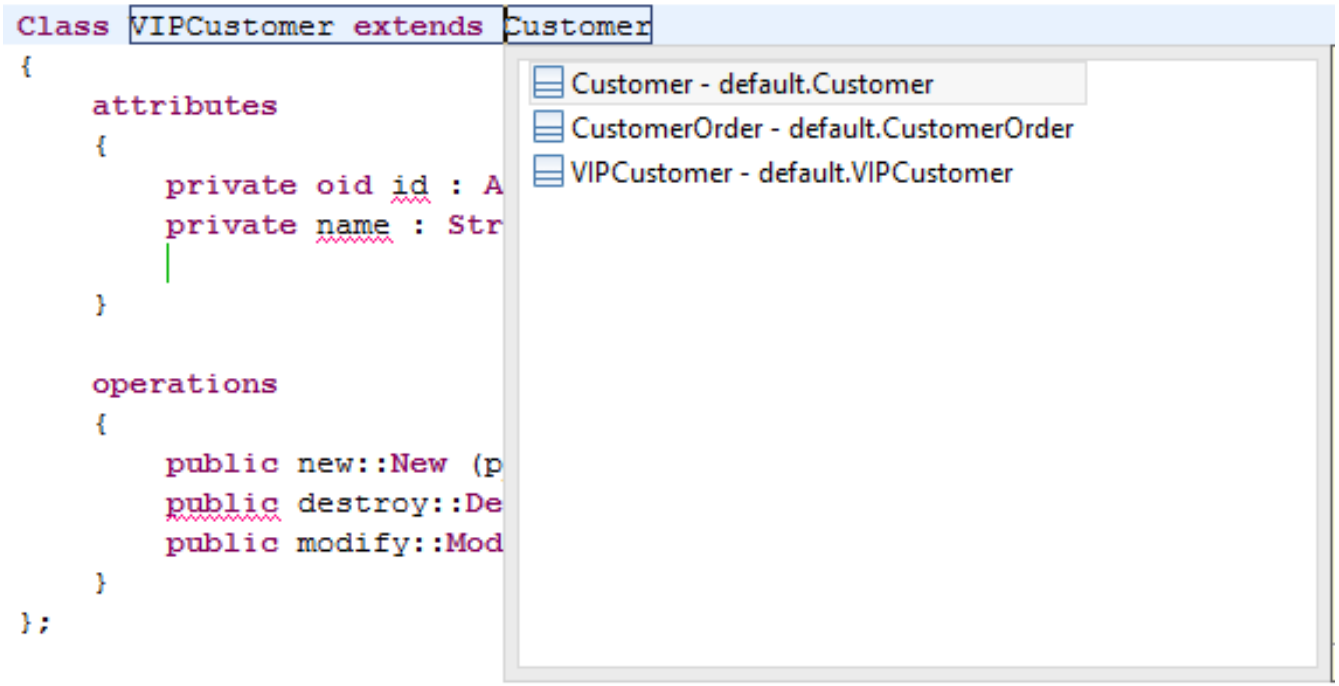
Composición:

```
Association Order_Line ( CustomerOrder - OrderLine )
{
    => navigable lines:Composite ( "0", "*");
    <= navigable order ( "1", "1");
};
```



Relación de herencia

- La herencia se especifica de la misma forma que en un Lenguaje Orientado a Objeto. Utilizamos la palabra **extends**
- Recordar que no admite herencia múltiple ni ciclos en entre las clases de la jerarquía

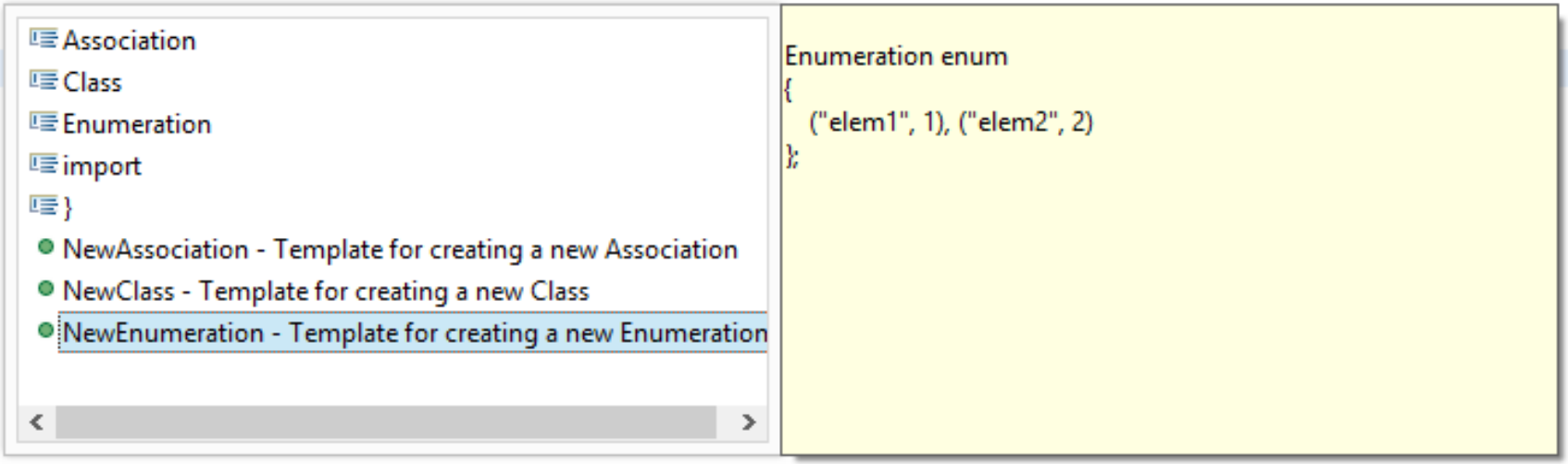


Relación de Herencia

- Recordad:
  - La clase hija de una relación de herencia, no puede definir su propio OID sino que lo obtiene del padre
  - No puede contener atributos ni roles que tengan el mismo nombre que los heredados del padre
  - La operación new incorporará los atributos y roles del padre y los que defina la propia clase hija

Clase Enumerada

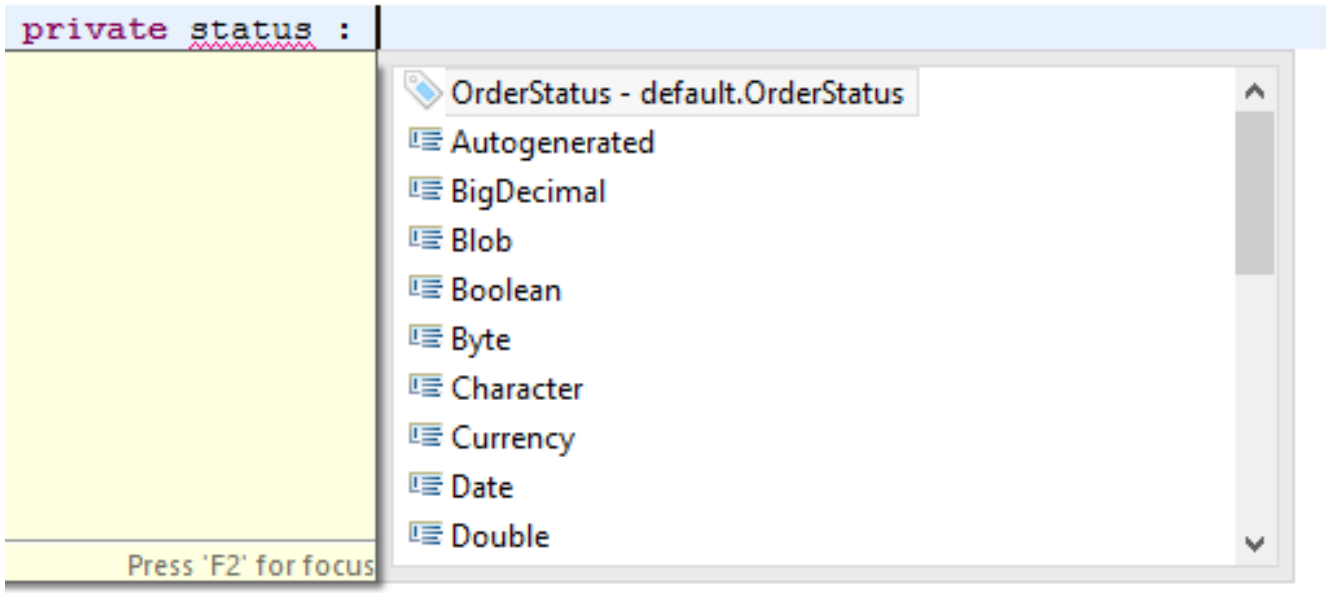
- Se define también como un elemento principal del modelo de dominio textual
- Utilizamos una plantilla para la creación rápida de la Enumeración




```
Enumeration OrderStatus
{
    ("pending", 1), ("sent", 2), ("received", 3), ("rejected", 4)
};
```

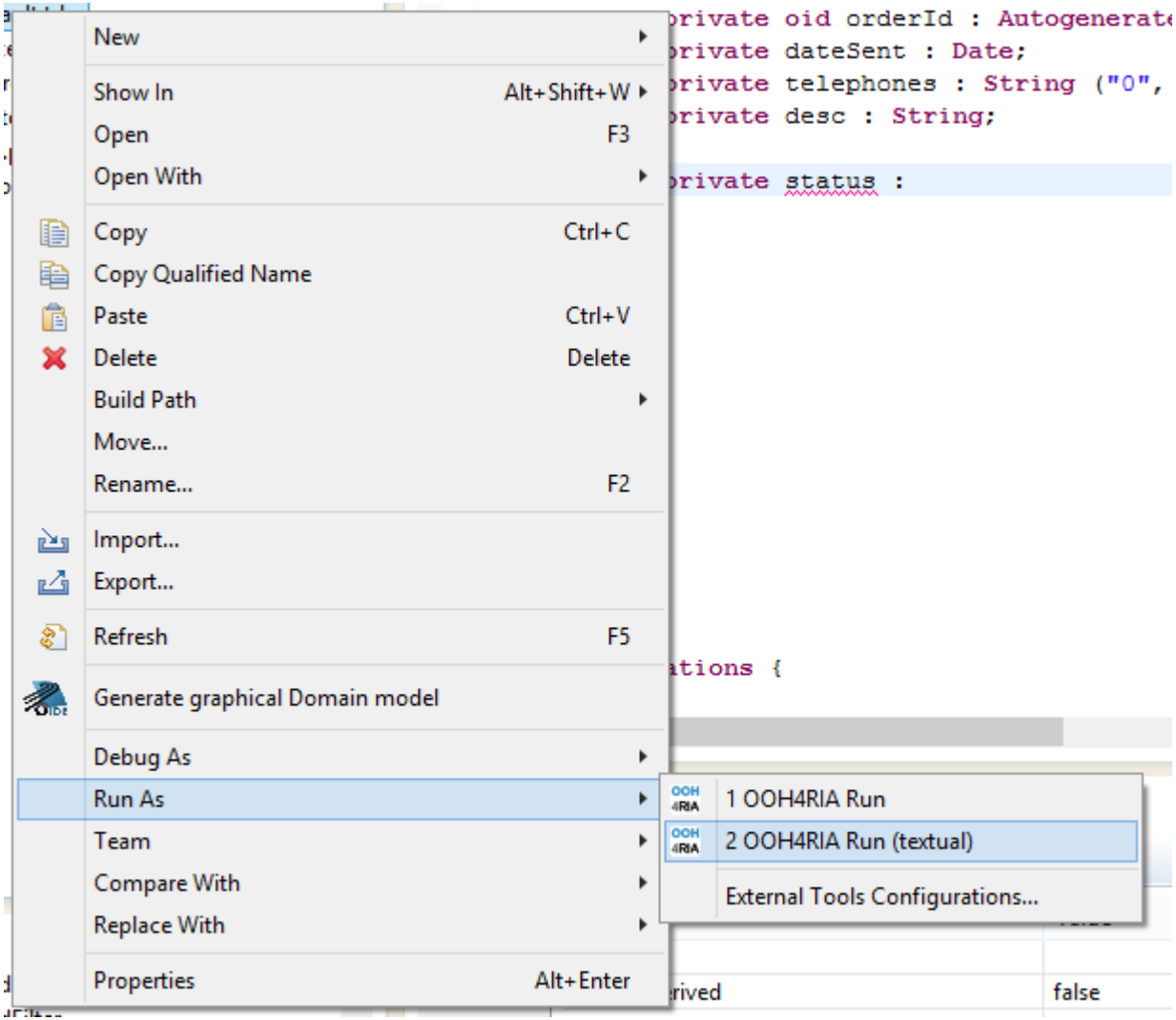
Atributos y argumentos Enumerados

- Una vez definida la clase enumeración los argumentos y atributos pueden tener como tipo dicha enumeración



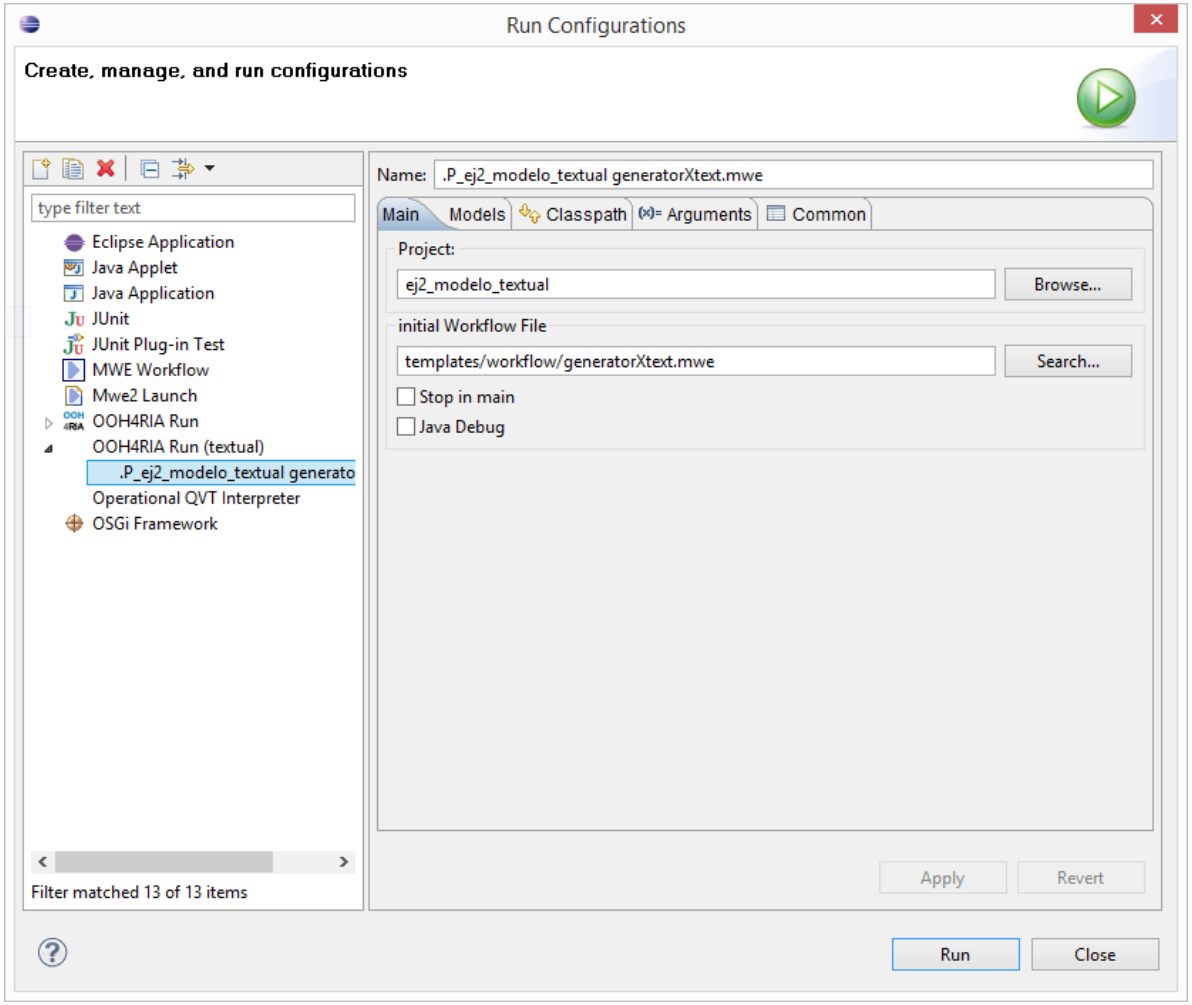
Generar Código modelo Textual

- La primera ocasión se selecciona el proyecto y se pulsa botón derecho Run As->OOH4RIA Run (Textual)



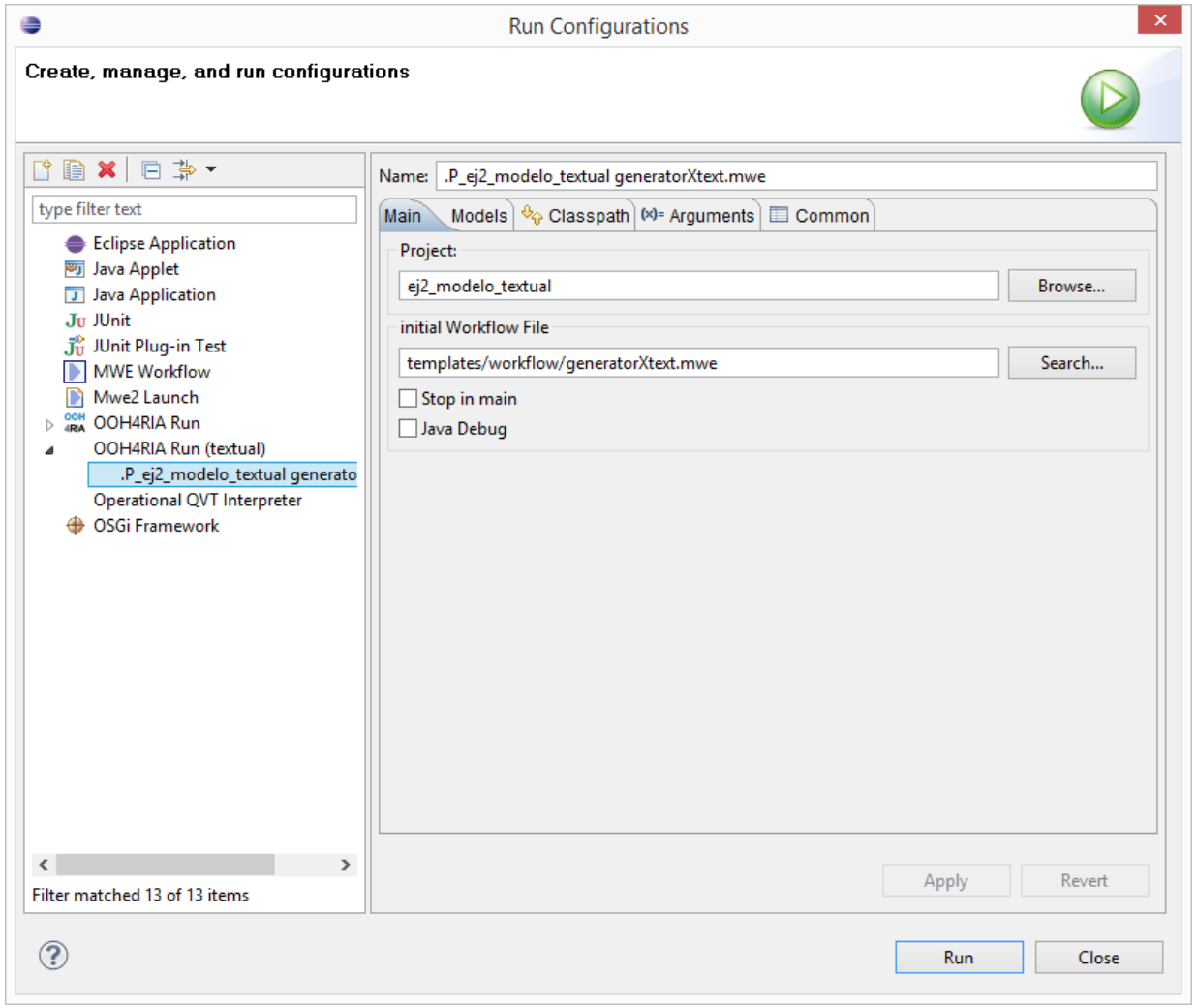
Generar Código

- Se crea un run configuration que apunta al fichero generatorXText.mwe llamado "nombreProyecto" + generatorXtext.mwe
- La próxima vez que lo ejecutemos invocará a dicho fichero
- Hay que elegir la pestaña models cuando haya que configurar los modelos



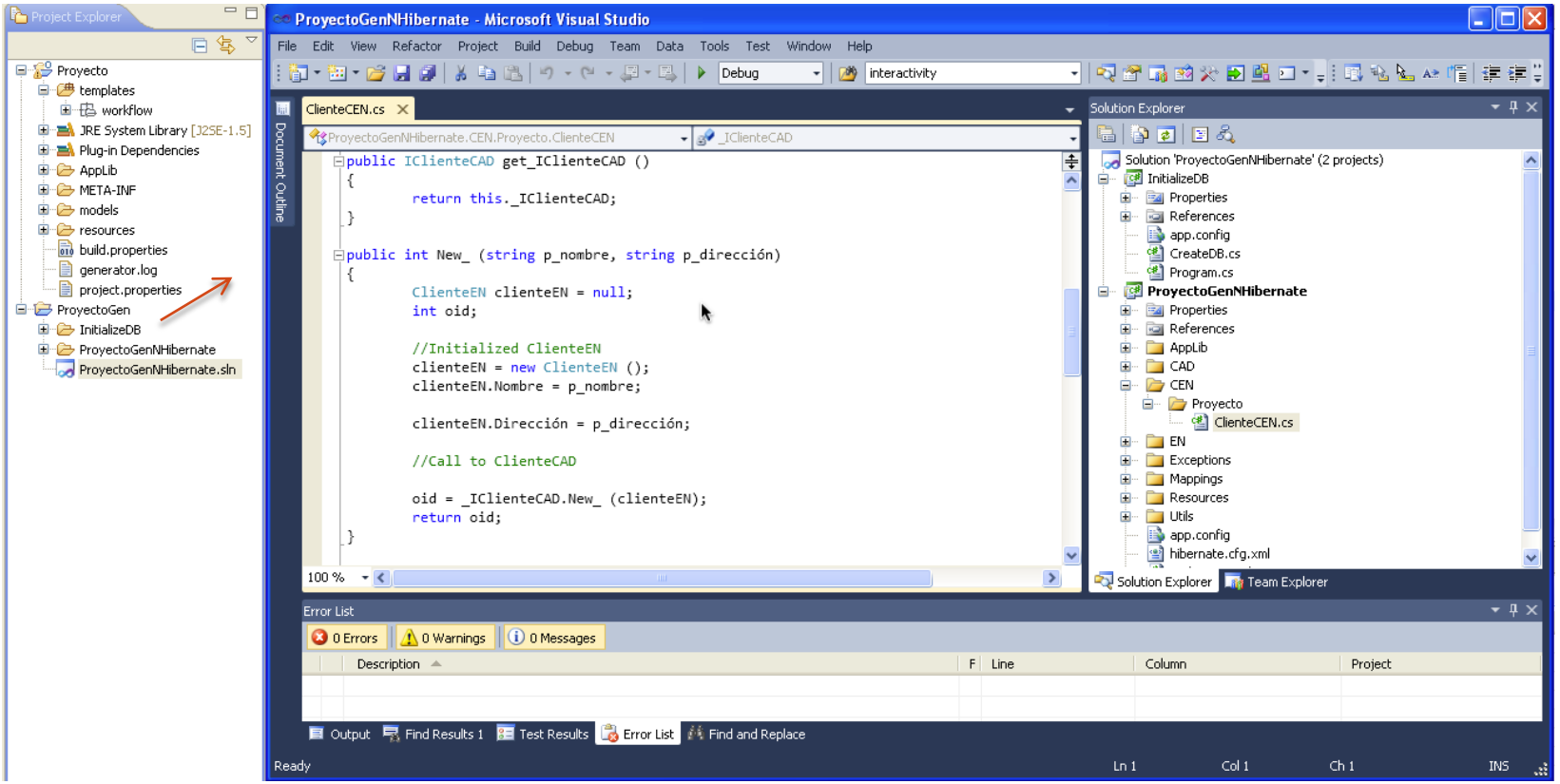
Generar Código

- Por defecto va al directorio models, el modelo por defecto se nombra default.tdm
- Cualquier cambio hay que seleccionar la ubicación con Browse…



Proyecto Generado

- Se genera un proyecto con el mismo nombre que pero con el sufijo "Gen" (igual que el modelo gráfico)
- Hacemos doble-click sobre la solución .sln y se abrirá Visual Studio (previamente instalado)



Ejercicio 2

- Modelar el mismo diagrama de clases con el modelo textual (el mismo que se realizó con el modelo gráfico)