

MODELADO Y DISEÑO DEL SOFTWARE

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

Álvaro Valencia Villalón
Alba de la Torre Segato
Pablo Astudillo Fraga
Carla Serracant Guevara
Pablo Alarcón Carrión
INGENIERÍA DEL SOFTWARE

PRÁCTICA 2

CONTEXTO

Para comenzar vamos a explicar un poco el contexto y situación del porque de este diagrama. Se nos ha pedido crear un diagrama o estructura de una empresa que vende diferentes productos, cada empresa puede tener un precio diferente de producto. También tenemos clientes, que pueden ser vip o no. Estos realizan pedidos de los productos a las empresas siempre que superen un valor mínimo. Además también debemos añadir la gente que trabaja en cada empresa, diferenciada entre director, gerente y lo que podríamos llamar un trabajador base, cada cargo con un salario diferente y con sus respectivas restricciones de personal. (Para más información véase el archivo adjunto con el enunciado).

Una vez resumido el contexto vamos a pasar a adjuntar los modelos en los diferentes entornos requeridos (USE, Visual Paradigm y Papyrus). Que posteriormente explicaremos.

DIAGRAMAS

USE

DIAGRAMA DE CLASES

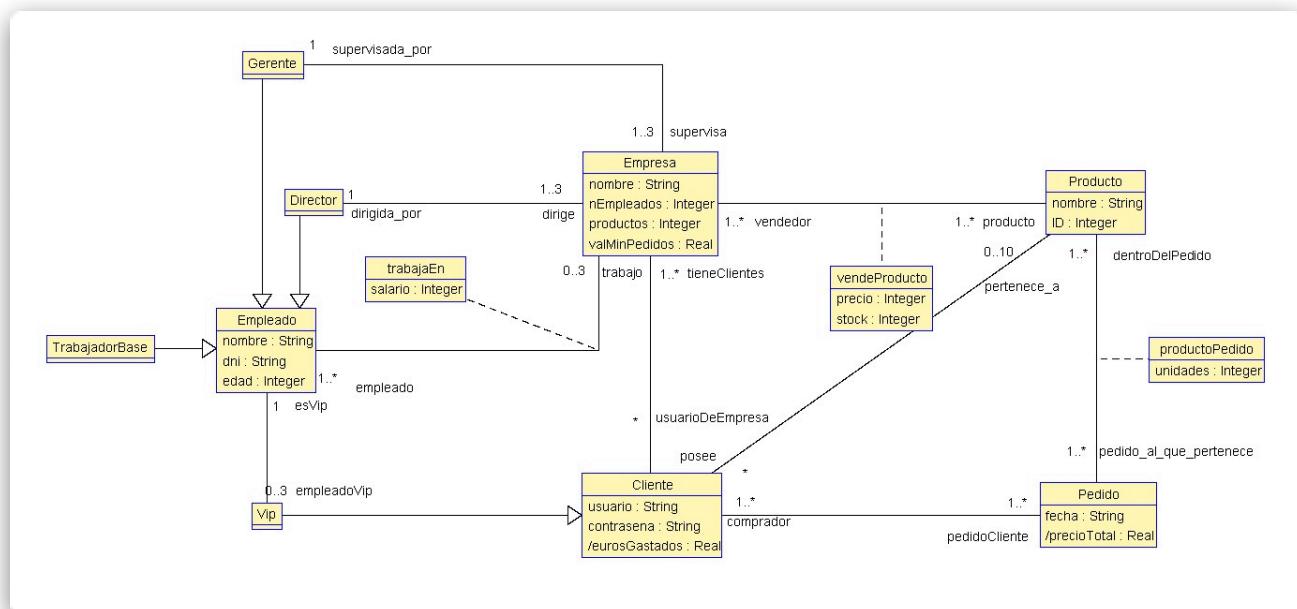
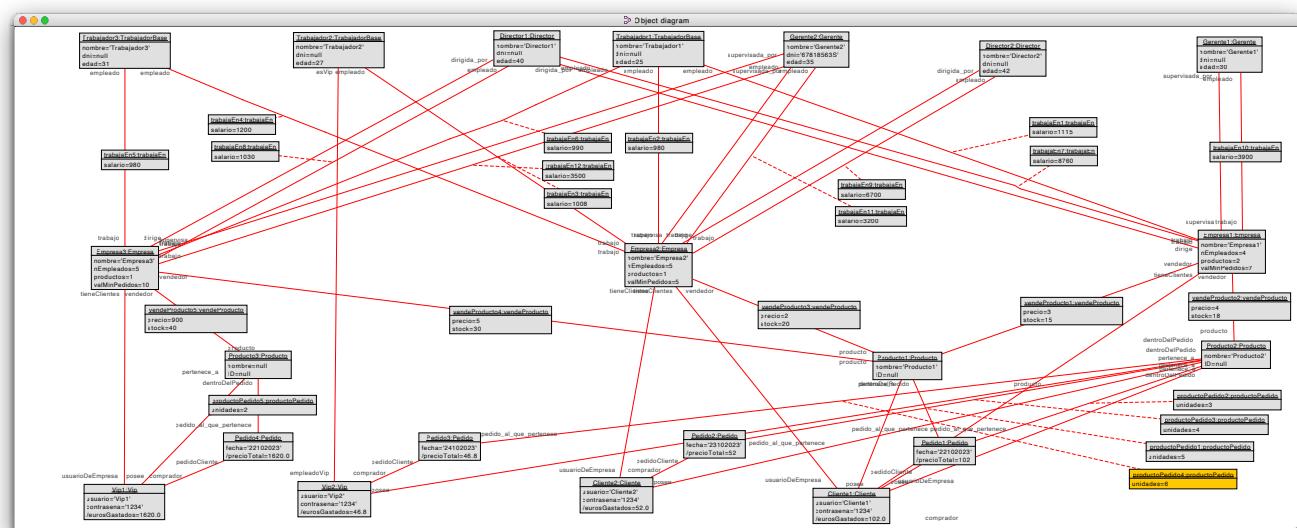
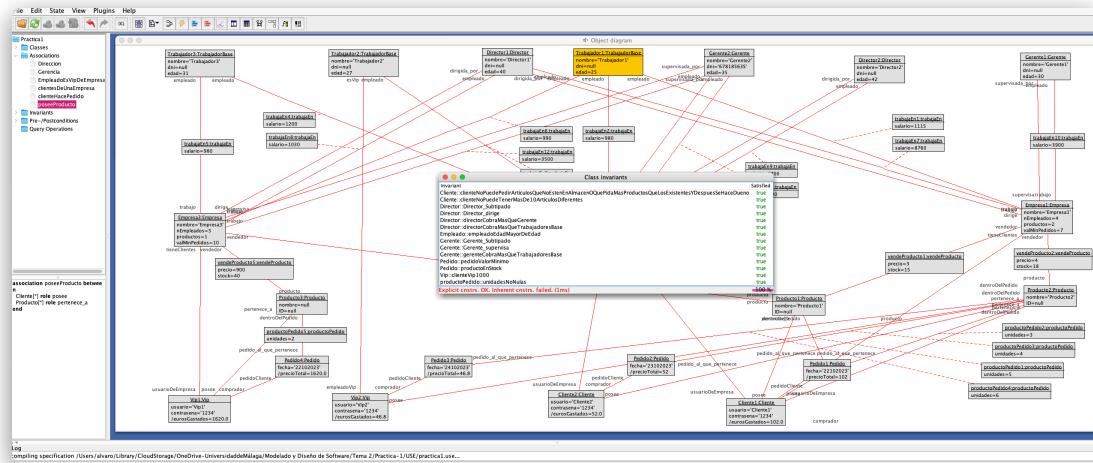


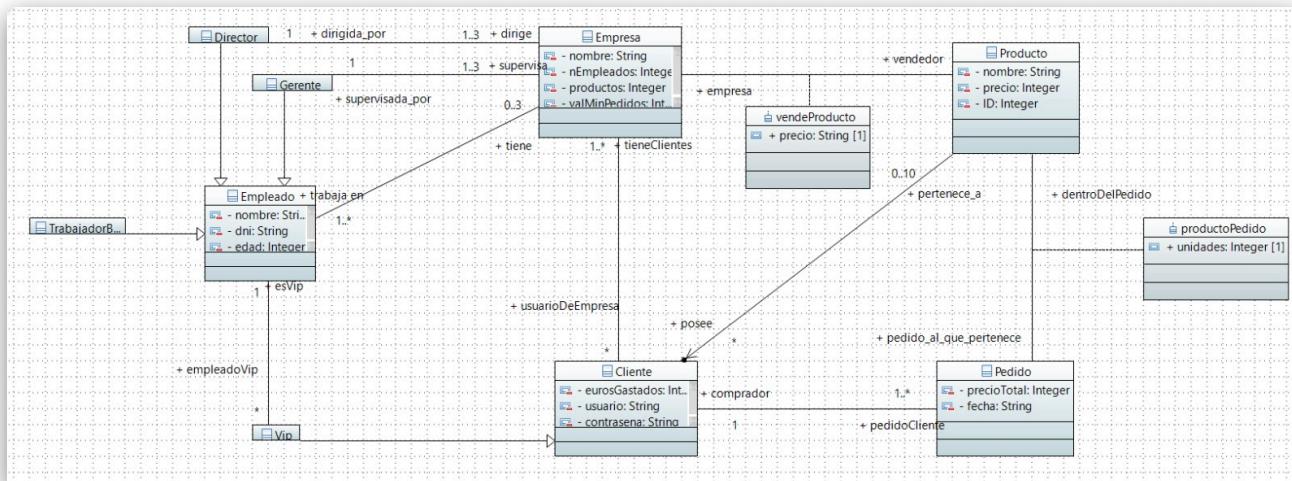
DIAGRAMA DE OBJETOS



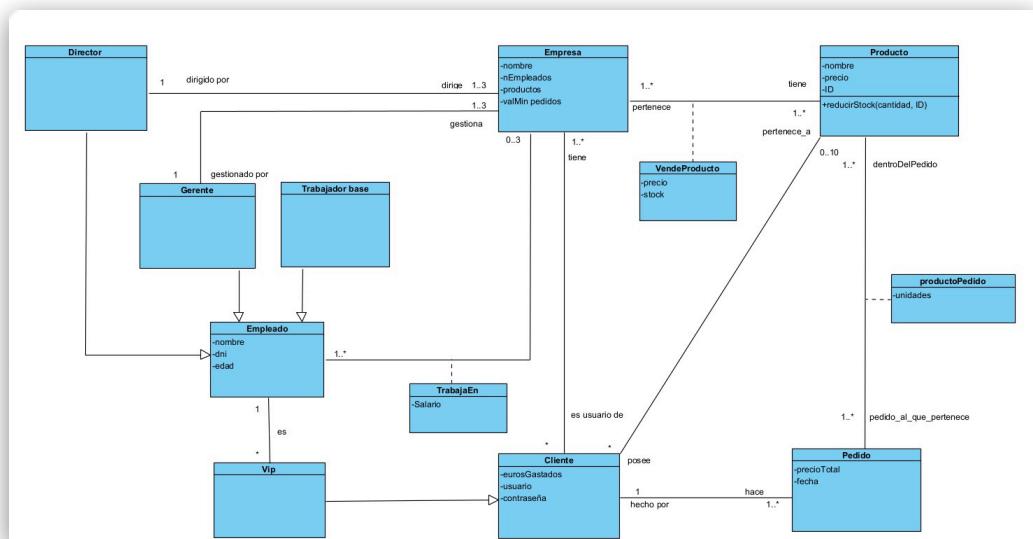
RESTRICCIONES CUMPLIDAS



PAPYRUS



VISUAL PARADIGM



EXPLICACIÓN

CLASES

Para comenzar la explicación empezaremos a explicar y describir las diferentes clases del modelo.

1) **Empresa:** cómo puede interpretarse por el nombre, esta clase representa la empresa. Sus atributos son:

a) nombre: atributo que representa el nombre de la empresa, es de tipo *String*.

b) nEmpleados: la empresa llevará un recuento de los empleados que tiene, para contabilizar y registrar ese dato utilizamos este atributo de tipo *Integer*.

c) productos: hemos querido representar el número de diferentes tipos de productos que tiene la empresa, para ello hemos utilizado este atributo de tipo *Integer*.

d) valMinPedidos: para que un cliente realice un pedido en la empresa es necesario que supere un valor mínimo o un precio del pedido mínimo. Como cada empresa tiene su valor mínimo hemos decidido añadir este atributo de tipo *Double*.

2) **Producto:** como vemos por el nombre de nuevo, esta clase representa el producto que vende la empresa. Sus atributos son:

a) nombre: atributo para representar el nombre del producto, es de tipo *String*.

b) ID: hemos querido dar un ID identificativo de cada producto, para representarlo hemos utilizado este atributo de tipo *Integer*.

3) **VendeProducto:** esta es una clase asociada a una relación entre una empresa y su producto, que representa el precio del producto asignado por la empresa y el stock del producto que tiene la misma. Sus atributos son los ya mencionados.

a) precio: atributo que representa el precio del producto asignado por la empresa, es de tipo *Double*.

b) stock: atributo que representa el stock que tiene la empresa del producto, es de tipo *Integer*.

4) **Cliente:** esta es una clase que representa, como bien dice el nombre, a un cliente. Sus atributos son:

a) eurosGastados: nos han pedido que cuando un cliente haya gastado un mínimo de euros pase a ser cliente vip, por lo que necesitaremos llevar un recuento y para eso tenemos este atributo de tipo *Double*.

b) usuario: para que alguien pueda realizar pedidos a una empresa debe ser un cliente registrado de la misma, por lo cual tendrá un nombre de usuario asignado que se almacenará en este atributo de tipo *String*.

c) contraseña: por la misma razón que la del atributo usuario utilizamos este atributo para asignar al cliente una contraseña, este atributo será de tipo *String*.

5) **Pedido:** esta es una clase que representa el pedido de un cliente. Sus atributos son:

a) fecha: atributo de tipo *String* que representa la fecha del pedido con formato *ddmmyyyy*.

b) precioTotal: atributo de tipo *Real* que representa el precio total del pedido realizado.

6) **productoPedido:** es una clase asociada a la relación entre *Pedido* y *Producto* que nos dirá mediante su atributo de tipo *Integer* la cantidad de unidades de producto que quiere el cliente.

7) **Empleado:** esta clase representa al empleado de la empresa. Sus atributos son:

a) nombre: representamos el nombre del cliente con este atributo de tipo *String*.

b) dni: para diferenciar los empleados que puedan tener el mismo nombre utilizamos este atributo del tipo *String* que almacena su número del DNI.

c) edad: hemos añadido este atributo para conocer la edad del empleado. Este atributo es de tipo *Integer*.

8) **trabajaEn:** es una clase asociada a la relación entre *Empresa* y *Empleado* la cual nos indica que el empleado trabaja para una empresa con un salario que nos dice el atributo de tipo *Integer* llamado *salario*.

9) **Vip:** clase sin atributos que nos indica cuando un cliente es *Vip* heredando así de *Cliente*.

10) **TrabajadorBase:** clase sin atributos que hereda de *Empleado*, se utiliza para decir cuando un empleado es un trabajador base sin ningún cargo adicional.

11) **Director:** clase sin atributos que hereda también de *Empleado*, con esta clase indicamos que el empleado tiene el cargo de director.

12) **Gerente:** clase sin atributos que hereda también de *Empleado*, con esta clase indicamos tambien el cargo de gerente que tiene el empleado.

RELACIONES

Vamos a empezar a explicar un poco las relaciones que hay en el diagrama con la multiplicidad y roles de cada una.

- 1) **trabajaEn** (Empleado-Empresa): es una clase asociada que nos indica el salario que cobra el empleado por trabajar en esa empresa, además de indicarnos que Empleado trabaja para una Empresa. Un Empleado puede trabajar en ninguna Empresa o hasta 3 Empresas. Una Empresa puede tener empleados de 1 a varios Empleados.
- 2) **Direccion** (Director-Empresa): es una relación utilizada para indicar cuando un empleado trabaja para una empresa como Director. Un Director puede dirigir de 1 a 3 Empresas. Una Empresa puede estar dirigida por un único Director.
- 3) **Gerencia** (Gerente-Empresa): es una relación utilizada para indicar cuando un empleado trabaja para una empresa como Gerente. Un Gerente puede supervisar de 1 a 3 Empresas. Una Empresa puede estar supervisada por un único Gerente.
- 4) **productoPedido** (Pedido-Producto): clase asociada que nos indica que se hace un pedido de con cierta cantidad de producto. Un Pedido puede tener dentro de 1 a varios Productos. Un Producto puede pertenecer a 1 o muchos Pedidos.
- 5) **vendeProducto** (Empresa-Producto): relación que nos indica que productos vende una empresa. Una Empresa puede vender de 1 a muchos Productos. Un Producto puede venderse por 1 o varias Empresas.
- 6) **EmpleadoEsVipDeEmpresa** (Vip-Empleado): relación entre que nos indica que cuando un empleado trabaja para la empresa inmediatamente pasa a ser vip.
- 7) **clientesDeUnaEmpresa** (Cliente-Empresa): relación que nos indica cuando un cliente es cliente de una empresa. Un Cliente puede ser cliente de 1 o muchas Empresas. Una Empresa puede tener muchos varios Clientes.
- 8) **clienteHacePedido** (Cliente-Pedido): relación que nos dice que un cliente hace un pedido. Un Cliente puede hacer de 1 a muchos Pedidos. Un Pedido puede ser pedido por 1 o muchos Clientes.
- 9) **poseeProducto** (Cliente-Pedido): relación entre producto y cliente que nos indica cuando un producto pasa de ser de una empresa al cliente que lo pide. Un Cliente

puede poseer de 0 a 10 Productos. Un Producto puede pertenecer a ninguno o varios Clientes.

CONSTRAINTS

- En otro programa que no fuera USE, esto seria un subtipado de relacion trabajaEn
- Si un director dirige una empresa, en alguna de las empresas en las que trabaja tiene que figurar el como director

context Director **inv** Director_Subtipado:

```
self.trabajo->notEmpty() and
self.trabajo->forAll(empresas_de_un_trabajador | self.dirige-
>exists(empresas_que_dirige | empresas_que_dirige = empresas_de_un_trabajador))
```

- En otro programa que no fuera USE, esto seria un subtipado de relacion
- Si un gerente regenta una empresa, en alguna de las empresas en las que trabaja tiene que figurar el como gerente

context Gerente **inv** Gerente_Subtipado:

```
self.trabajo->notEmpty() and
self.trabajo->forAll(empresas_de_un_trabajador |
self.supervisa->exists(empresas_que_supervisa | empresas_que_supervisa =
empresas_de_un_trabajador)
)
```

- Un director tiene que dirigir, si no seria trabajador base

context Director **inv** Director_dirige:

```
self.dirige -> notEmpty()
```

- Un gerente tiene que supervisar, si no seria trabajador base

context Gerente **inv** Gerente_supervisa:

```
self.supervisa -> notEmpty()
```

- Un director cobra más que un gerente dentro de una misma empresa 2 como el profesor dice

context Director **inv** directorCobraMasQueGerente:

```
self.trabajaEn.salario->asSequence()->max() >
self.dirige.supervisada_por.trabajaEn.salario->asSequence()->max()
```

--Un gerente cobra más que los trabajadores base dentro de una misma empresa
context Gerente **inv** gerenteCobraMasQueTrabajadoresBase:
self.trabajaEn.salario->asSequence()->max() >
self.supervisa.empleado->select(oclIsTypeOf(TrabajadorBase)).trabajaEn.salario-
>asSequence()->max()

--Un director cobra más que los trabajadores base dentro de una misma empresa
context Director **inv** directorCobraMasQueTrabajadoresBase:
self.trabajaEn.salario->asSequence()->max() >
self.dirige.empleado->select(oclIsTypeOf(TrabajadorBase)).trabajaEn.salario-
>asSequence()->max()

--cliente es vip si ha realizado un pedido de más de 1000 euros
context Vip **inv** clienteVip1000:
self.oclIsTypeOf(Vip) **and** self.esVip->isEmpty() implies self.eurosGastados > 1000

--nadie puede hacer un pedido de artículos que no estén en el almacén y lo hace dueño
de ese producto
context Cliente **inv**
clienteNoPuedePedirArticulosQueNoEstenEnAlmacenOQuePidaMasProductosQueLo
sExistentesYDespuesSeHaceDueño:
self.pedidoCliente->forAll(pedido | pedido.dentroDelPedido->forAll(producto |
pedido.dentroDelPedido->forAll(producto |
producto.vendedor.vendeProducto->exists(p | p.stock <> null implies p.stock
>=0 **and**
producto.productoPedido -> exists(e | e.unidades <= p.stock))) **and**
self.pertenece_a->includes(producto)))

-- Un empleado debe ser mayor de edad
context Empleado **inv** empleadoEdadMayorDeEdad:
self.edad >= 18

--Un pedido debe tener un valor minimo
context Pedido **inv** pedidoValorMinimo:

```

self.precioTotal >= self.dentroDelPedido.vendedor->select(empresa |
empresa.producto.pedido_al_que_pertenece->includes(self)).valMinPedidos-
>asSequence()->first()
--podemos poner first() en el valMinPedidos porque aunque se lea como una
secuencia, un pedido
--va a pertenecer exclusivamente a un solo vendedor. no puede haber un mismo pedido
de dos vendedores distintos

```

--Producto pedido debe estar en stock

context Pedido inv productoEnStock:

```

(self.dentroDelPedido->iterate(producto; acumulador : Integer = 0 | if
producto.vendeProducto->exists(empresa |
empresa.producto.pedido_al_que_pertenece->includes(self)) and
producto.vendeProducto->select(empresa |
empresa.producto.pedido_al_que_pertenece->includes(self)).stock->asSequence()-
>first >=1
then acumulador +1 else acumulador endif)) = self.dentroDelPedido->size()

```

--no puedes pedir 0 unidades de un producto

context productoPedido inv unidadesNoNulas:

self.unidades > 0

--una misma persona no puede tener artículos de más de 10 tipos de productos distintos, independientemente de la empresa en donde los compró

--asSet() hace que se generen objetos únicos con ID diferentes.

context Cliente inv clienteNoPuedeTenerMasDe10ArticulosDiferentes:

self.pedidoCliente->forAll(pedido |

```

    let productosDiferentes = pedido.dentroDelPedido->select(p |
p.productoPedido.unidades <> null and p.productoPedido.unidades->sum() >= 1)-
>asSet()
    in
    productosDiferentes->size() <= 10
)

```