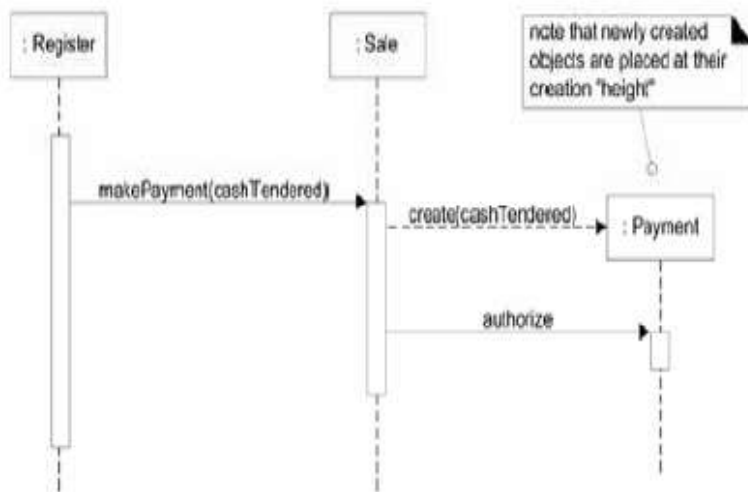


Diagrama de clases



Categorías DOO



- **Modelos de Objetos Dinámicos:**
- Ayudan a diseñar la lógica, el comportamiento del código o los cuerpos de los métodos
- Ej. Diagramas de Interacción



- **Modelos de Objetos**
Estáticos: Ayudan a diseñar la definición de paquetes, nombres de clases, atributos
- Ej. Diagrama de Clases

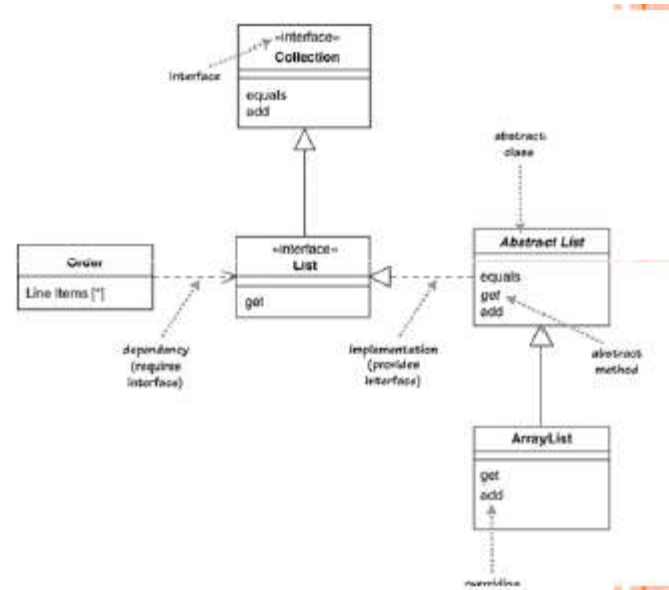
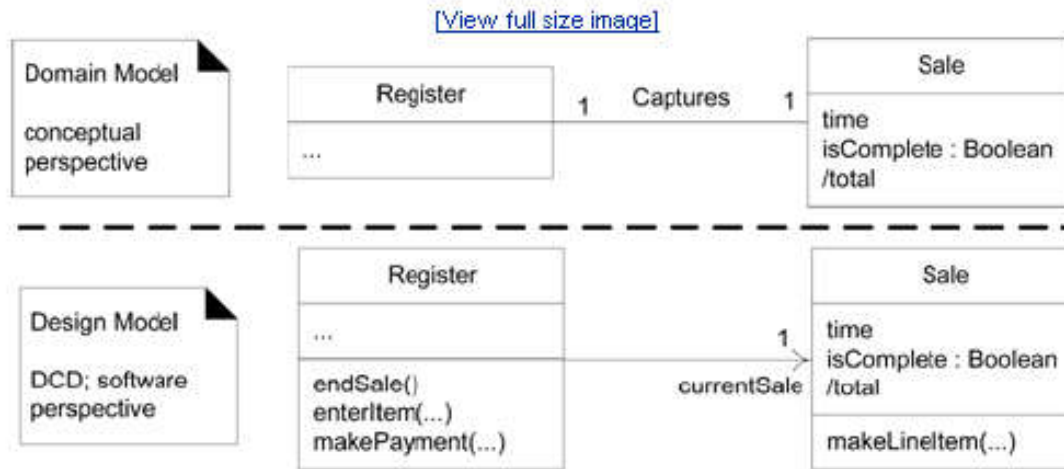


DIAGRAMA DE CLASES DE DISEÑO (DCD)

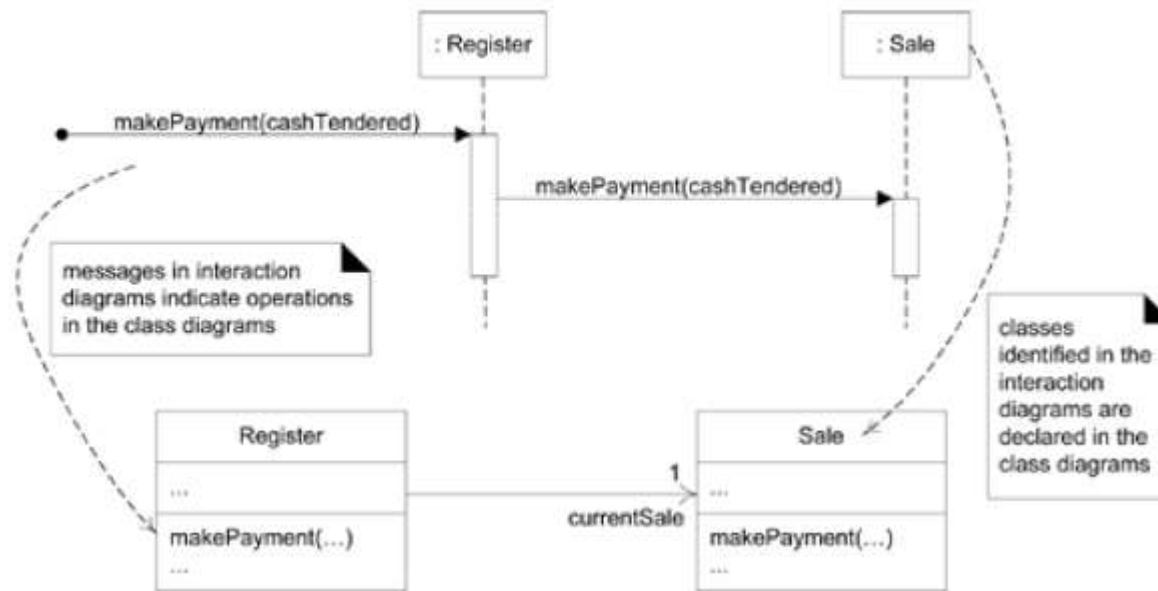
- Ilustra Clases, métodos, Interfaces, asociaciones, navegabilidad y dependencias
- Es la base para futuras etapas del diseño (Ej Diagrama de Componentes)
- Parten del modelo de dominio y de los diagramas de interacción como fuente primaria



DEL MODELO DE DOMINIO AL DCD



DCD Y LOS DIAGRAMAS DE SECUENCIA



ATRIBUTOS



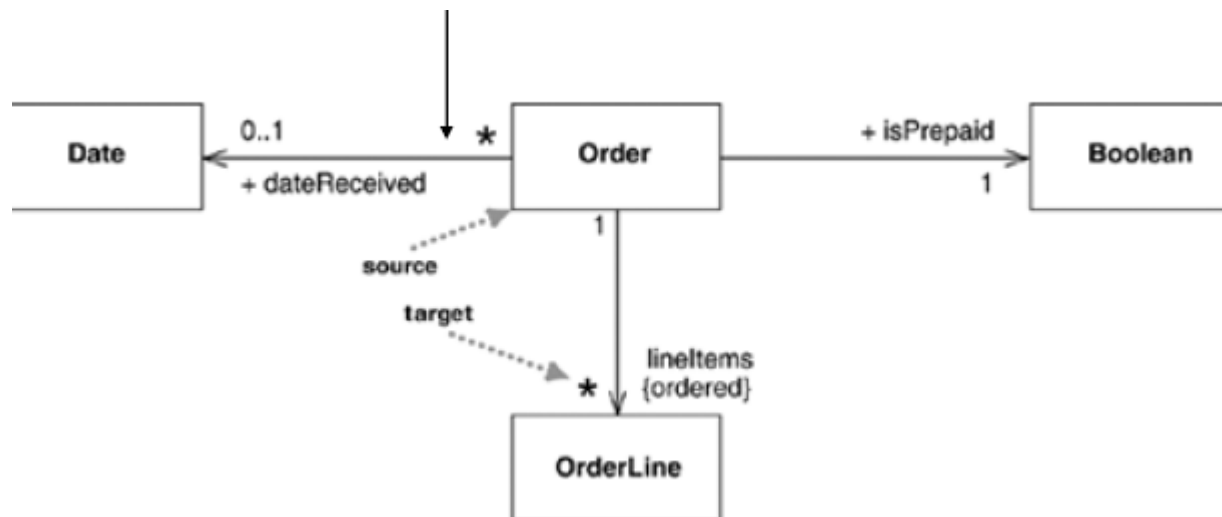
ATRIBUTOS: NOTACIÓN TEXTUAL

- Describe la propiedad en forma textual en la misma caja de la clase
- visibility name: type multiplicity = default {property-string}
- - name: String [1] = "Untitled" {readOnly}



ATRIBUTOS: LÍNEA ASOCIACIÓN

- Línea sólida entre 2 clases. Va de la clase fuente a la clase objetivo

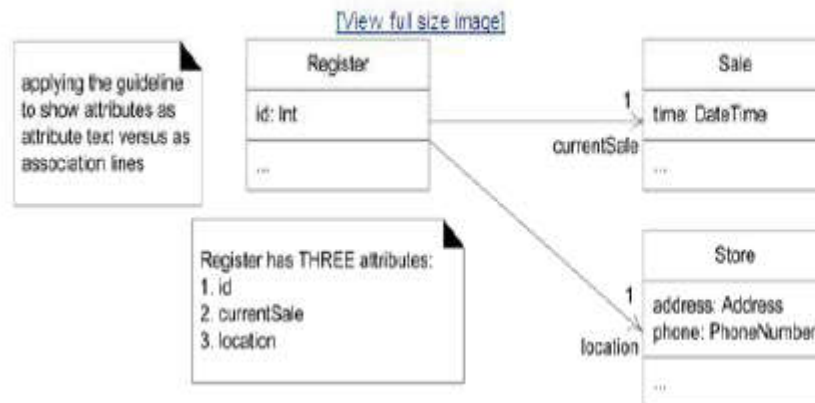


MULTIPLICIDAD

*	T	ceros o más "muchos"
1..*	T	uno o más
1..40	T	de uno a 40
5	T	exactamente 5
3, 5, 8	T	exactamente 3, 5, u 8

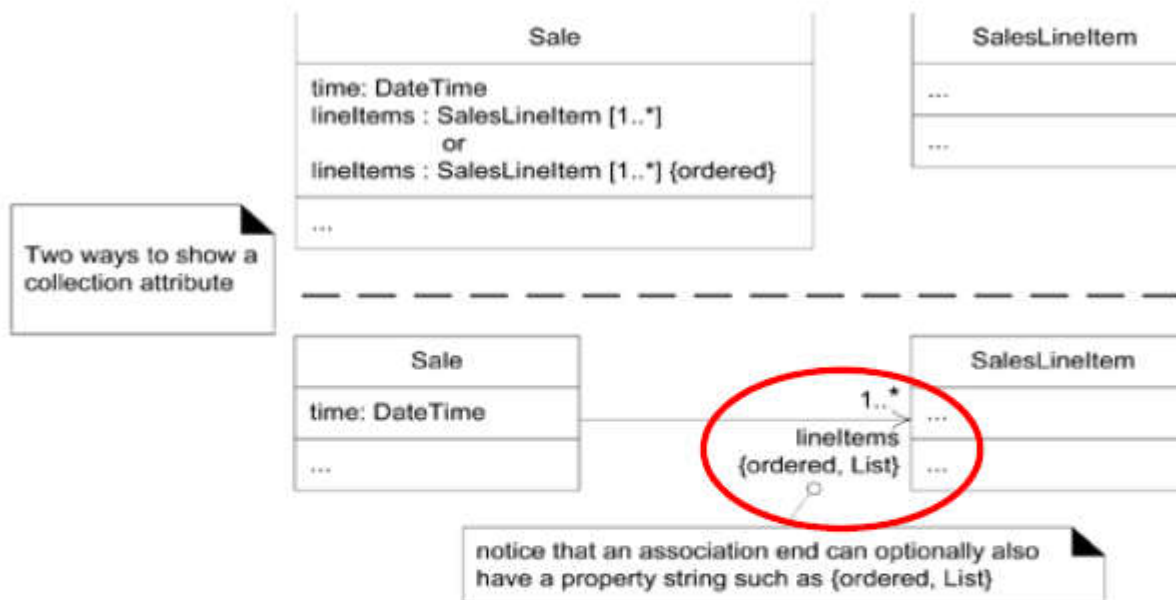


ATRIBUTOS



```
public class Register
{
    private int id;
    private Sale currentSale;
    private Store location;
    // ...
}
```





```

public class Sale
{
private List<SalesLineItem> lineItems =
    new ArrayList<SalesLineItem>();

// ...
}
  
```



OPERACIONES

SuperclassFoo or SuperClassFoo { abstract }
<div><div>- classOrStaticAttribute : int</div><div>+ publicAttribute : String</div><div>- privateAttribute</div><div>assumedPrivateAttribute</div><div>isInitializedAttribute : Bool = true</div><div>aCollection : VeggieBurger { * }</div><div>attributeMayLegallyBeNull : String [0..1]</div><div>finalConstantAttribute : int = 5 { readOnly }</div><div>/derivedAttribute</div></div>
<div><div>+ classOrStaticMethod()</div><div>+ publicMethod()</div><div>assumedPublicMethod()</div><div>- privateMethod()</div><div># protectedMethod()</div><div>- packageVisibleMethod()</div><div>«constructor» SuperclassFoo(Long)</div><div>methodWithParams(parm1 : String, parm2 : Float)</div><div>methodReturnsSomething() : VeggieBurger</div><div>methodThrowsException() (exception IOException)</div><div>abstractMethod()</div><div>abstractMethod2() { abstract } // alternate</div><div>finalMethod() { leaf } // no override in subclass</div><div>synchronizedMethod() { guarded }</div></div>

- Acciones que se pueden realizar en la clase
- visibility name (parameter-list)
: return-type {propertystring}
- Ej. +getPlayer(name : String)
: Player {exception
IOException}



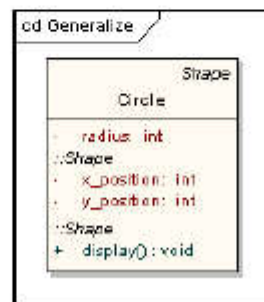
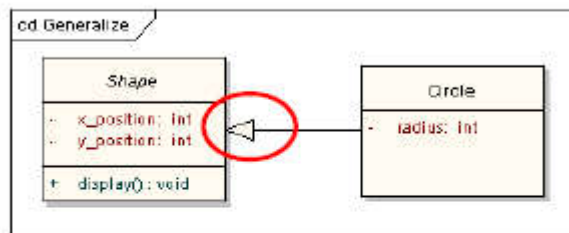
NOTAS

- Múltiples usos, ej:
 - •Comentarios o aclaraciones
 - •Restricciones
 - •Implementación de un método



GENERALIZACIÓN

GENERALIZACIÓN

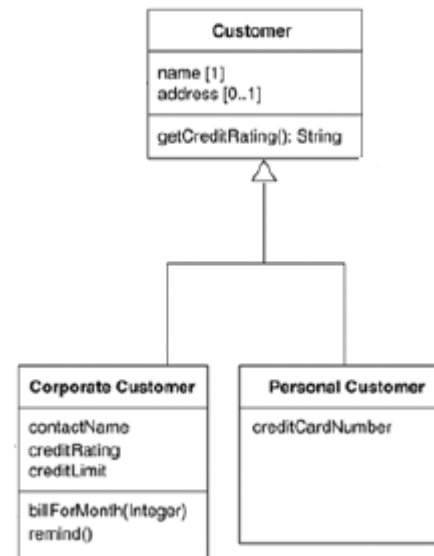


- Muestra una relación entre una clase general y otra más específica
- A nivel de DCD utilizado para representar Herencia



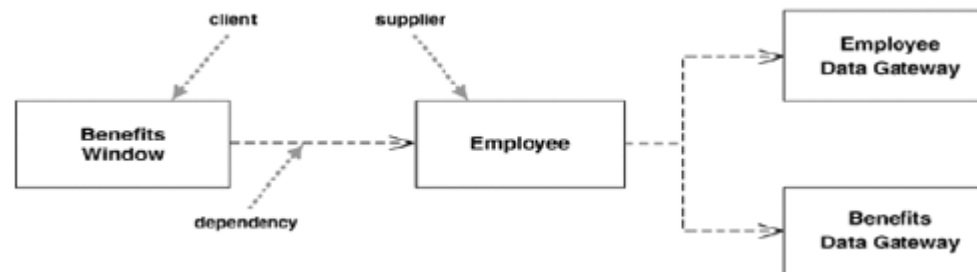
GENERALIZACIÓN

- Una empresa tiene un registro de sus clientes. Estos clientes pueden ser de tipo personas físicas o empresas



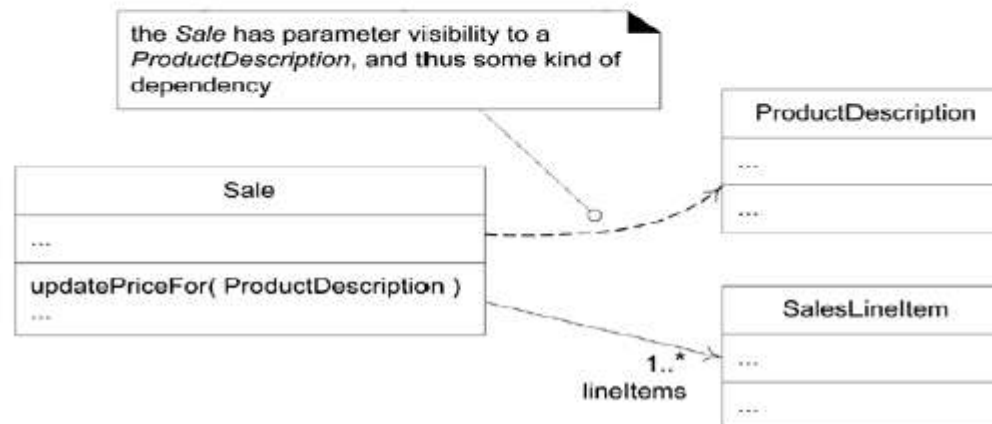
DEPENDENCIAS

- Indica que un elemento tiene conocimiento de otro elemento
- Existe una dependencia de una clase a otra, si cambios en la definición de un elemento puede causar cambios en otra (va en una única dirección)
- Identificada por una línea punteada

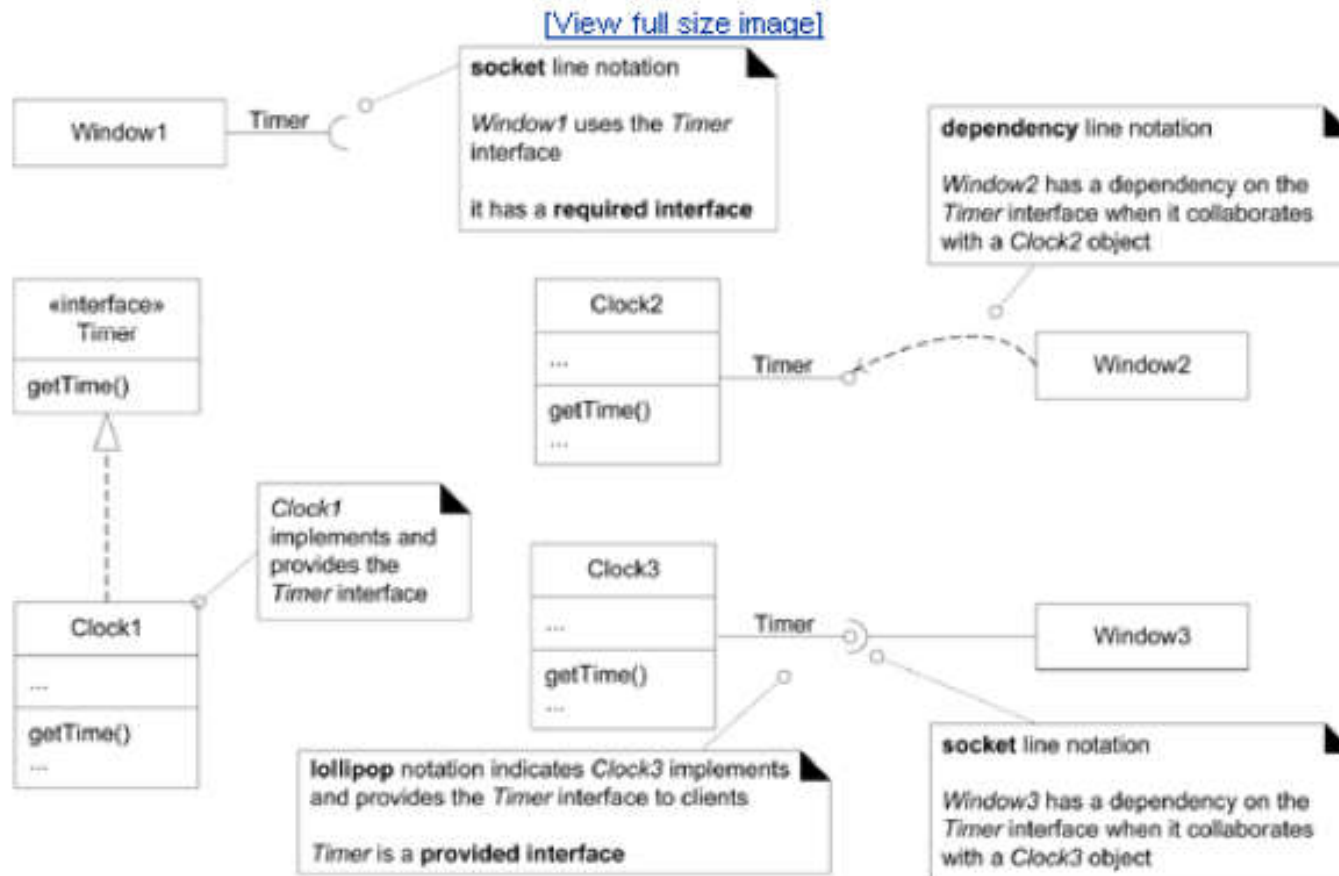


EJEMPLO DEPENDENCIAS

```
public class Sale
{
    public void updatePriceFor( ProductDescription description )
    {
        Money basePrice = description.getPrice();
        //...
    }
    // ...
}
```

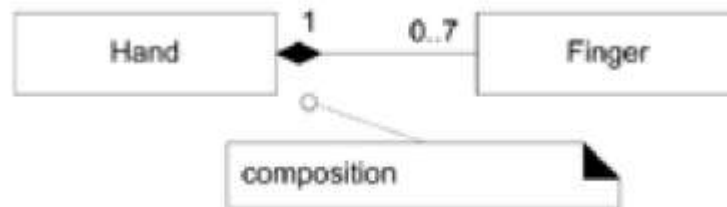


INTERFACES



COMPOSICIÓN

- Variación de una agregación
- Indica que la parte debe siempre pertenecer a un compuesto y este es responsable por la creación y eliminación
- Cada componente de una composición puede pertenecer tan solo al todo



EJEMPLO DIAGRAMA PARCIAL DE CLASES

