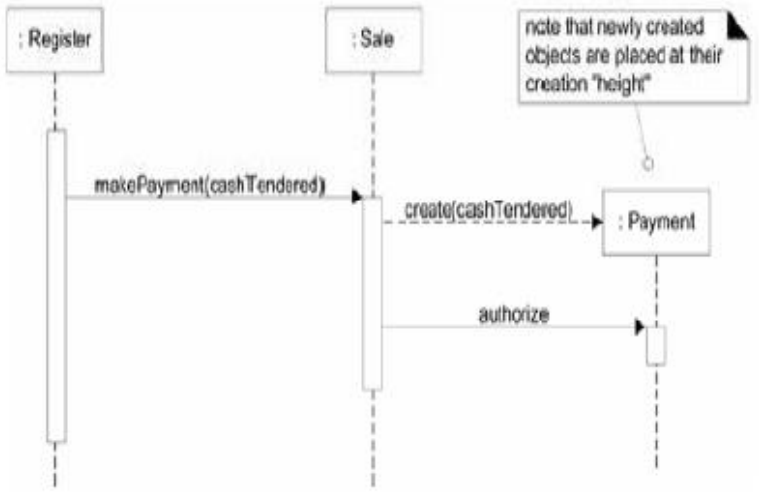
Diagrama de clases

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

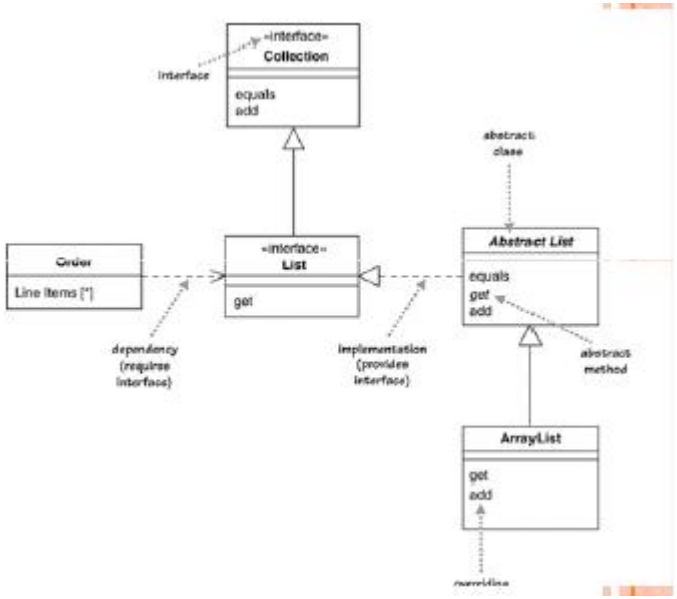
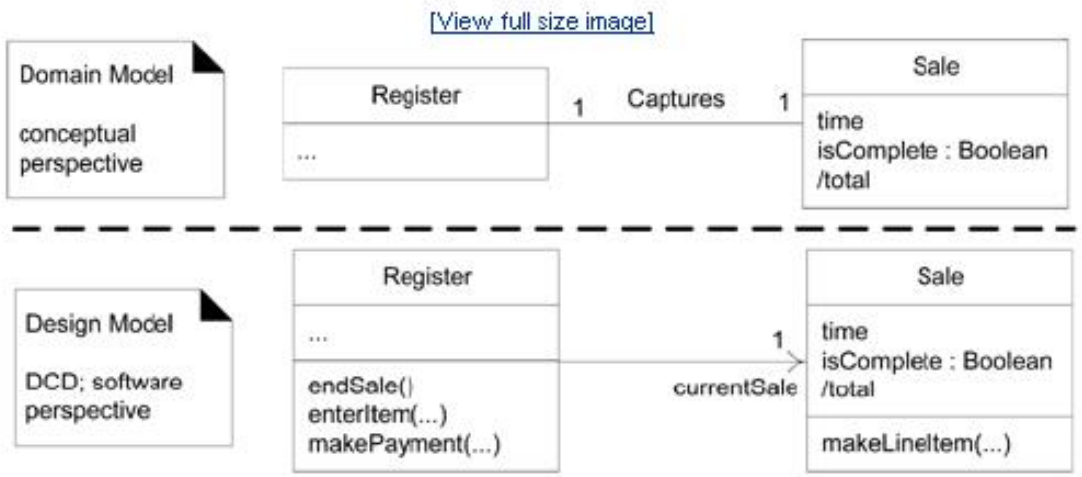


DIAGRAMA DE CLASES DE DISEÑO (DCD)

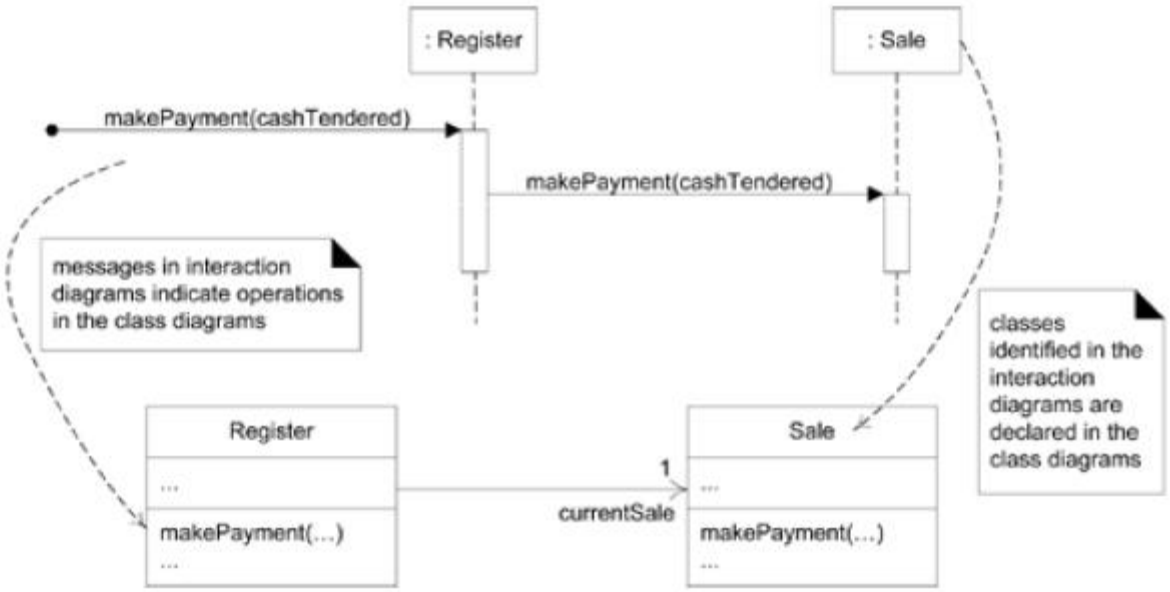
- Ilustra Clases, métodos, Interfaces, asociaciones, navegabilidad y dependencias
- Es la base para futuras etapas del diseño (Ej Diagrama de Componentes)
- Parten del modelo de dominio y de los diagramas de interacción como fuente primaria

DEL MODELO DE DOMINIO AL DCD

[View full size image]



DCD Y LOS DIAGRAMAS DE SECUENCIA



ATRIBUTOS

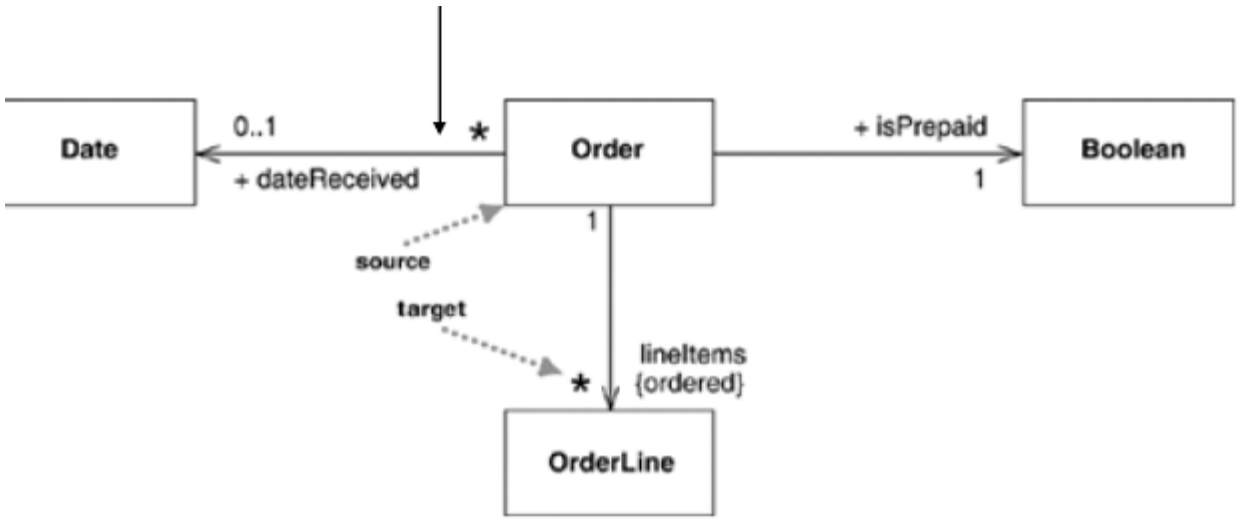

Register
currentSale : Sale
...
Sale
...
...
Registor
...
...
1
currentSale
Sale
...
...
Register
currentSale : Sale
...
1
currentSale
Sale
...
...

ATRIBUTOS: NOTACIÓN TEXTUAL

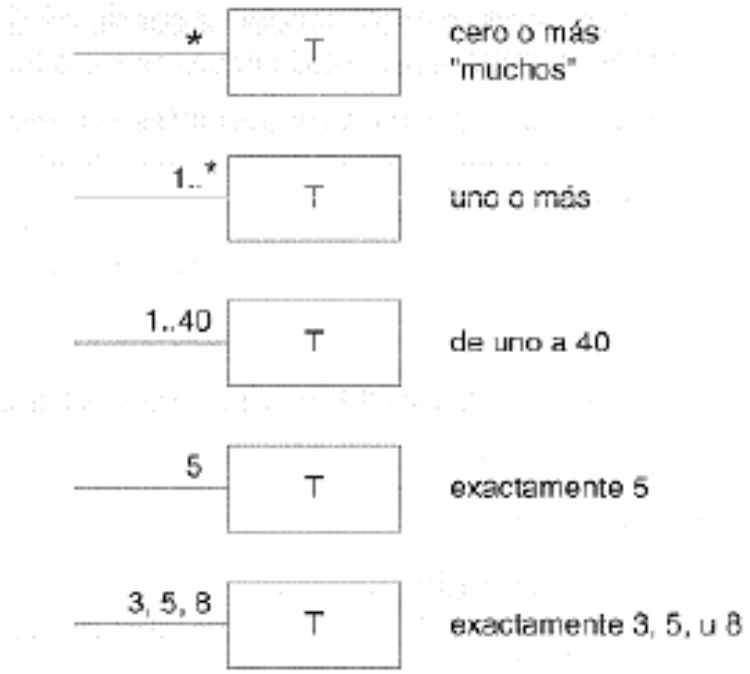
- Describe la propiedad en forma textual en la misma caja de la clase
- visibility name: type multiplicity = default {property-string}
- - name: String [1] = "Untitled“ {readOnly}

ATRIBUTOS: LÍNEA ASOCIACIÓN

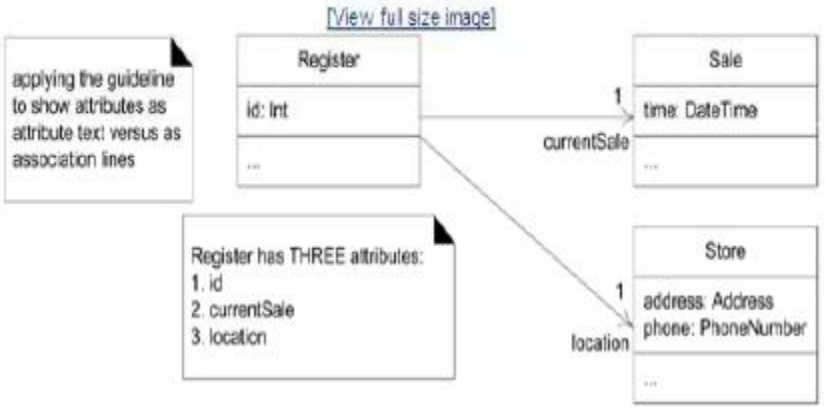
- Línea sólida entre 2 clases. Va de la clase fuente a la clase objetivo



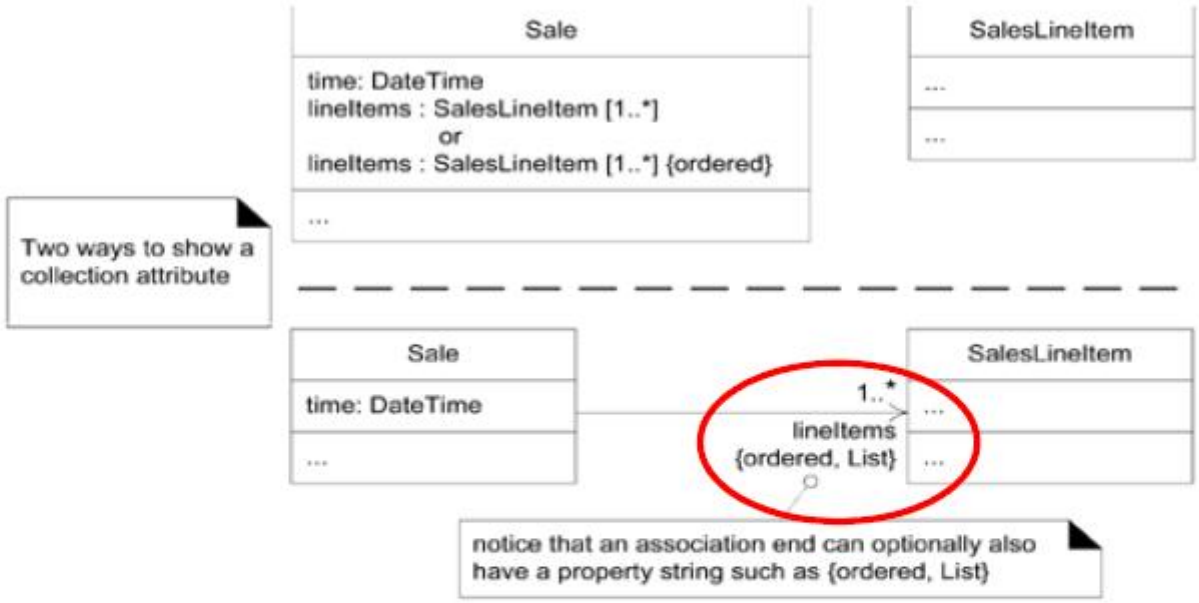
MULTIPLICIDAD



ATRIBUTOS




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



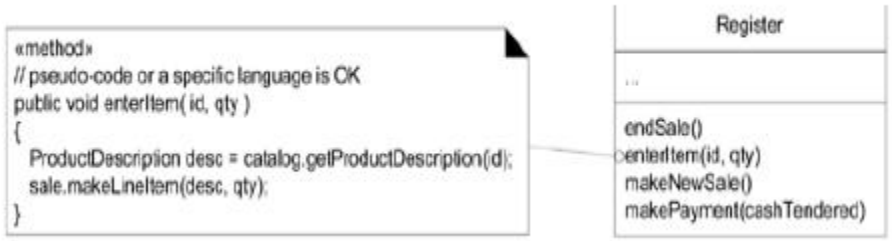

```
public class Sale
{
private List<SalesLineItem> lineItems =
                                new ArrayList<SalesLineItem>();
// …
}
```

OPERACIONES

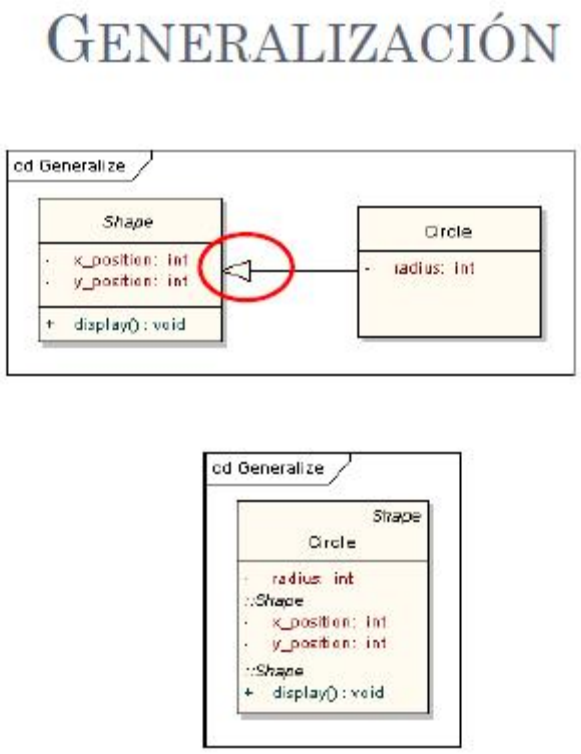
- Acciones que se pueden realizar en la clase
- visibility name (parameter-list) : return-type {propertystring}
- Ej. +getPlayer(name : String) : Player {exception IOException}

NOTAS

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



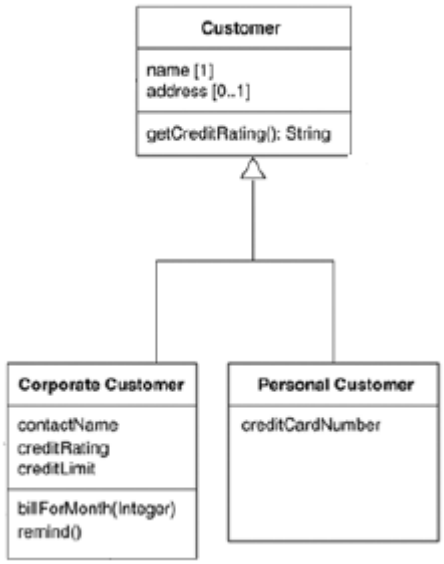
GENERALIZACIÓN



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

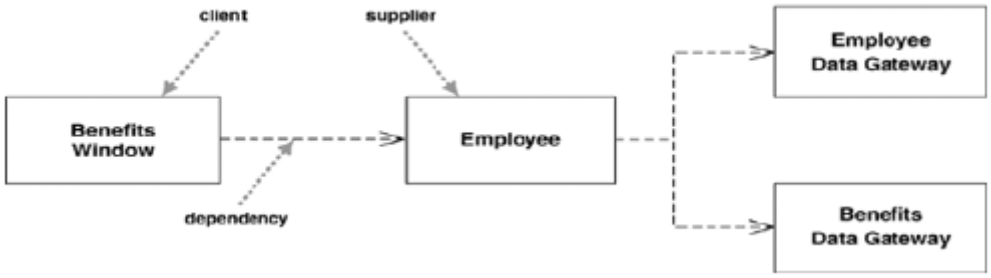
GENERALIZACIÓN

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



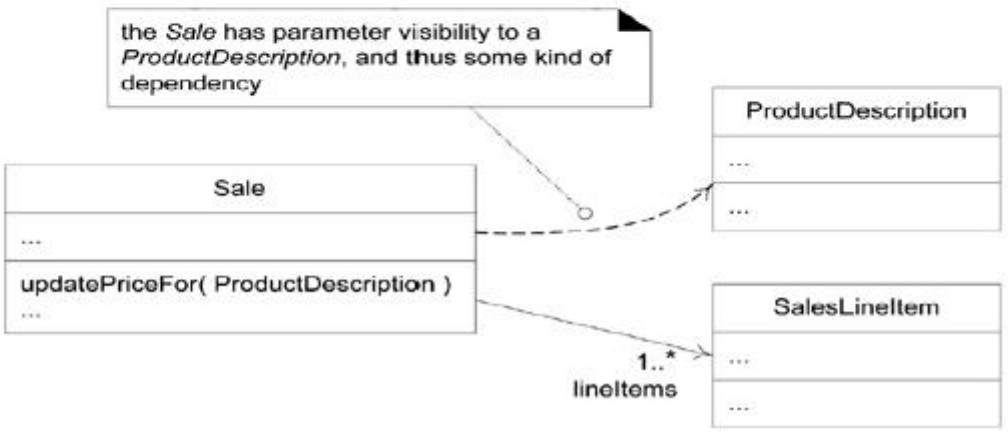
DEPENDENCIAS

- 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

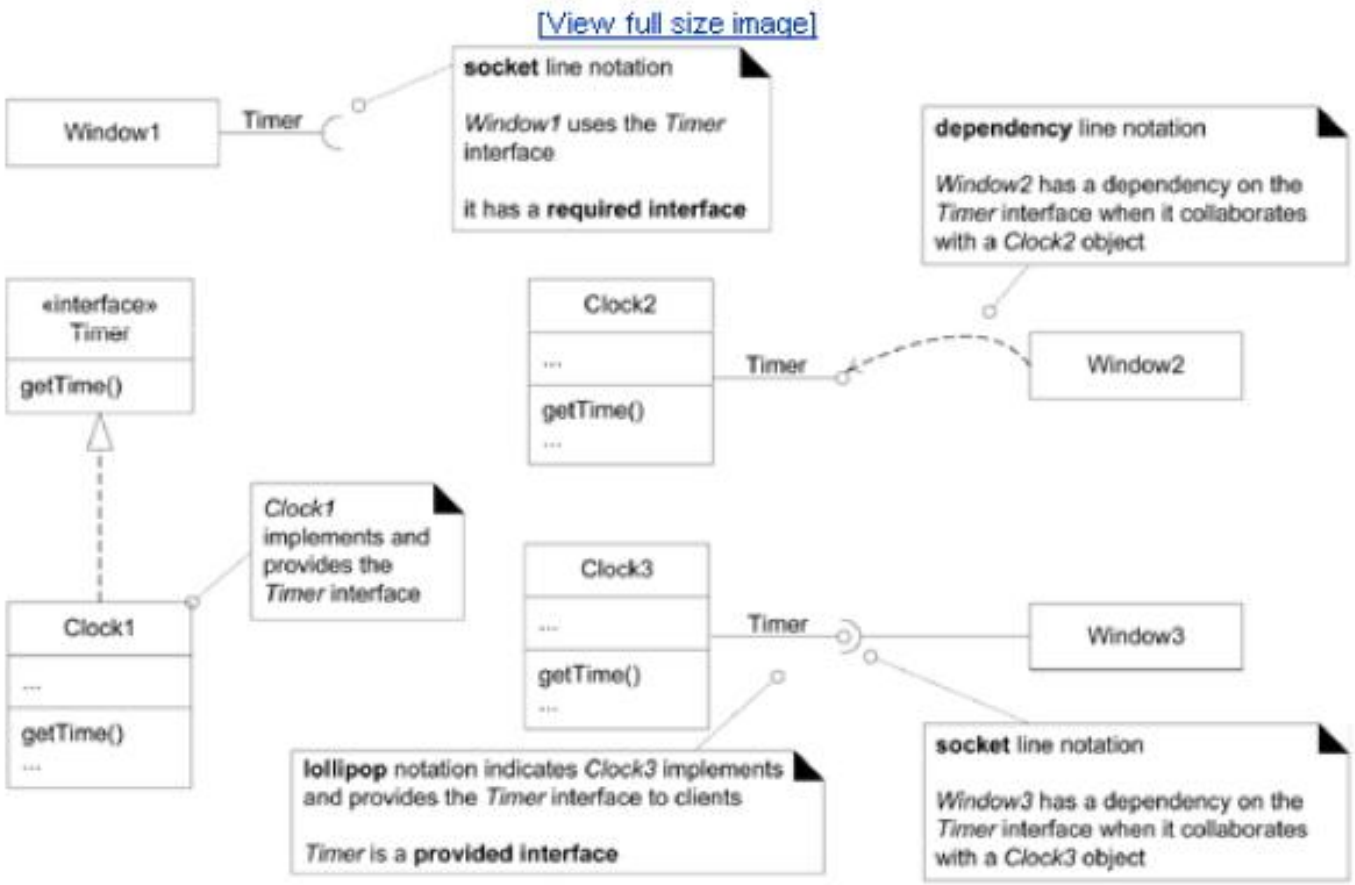


EJEMPLO DEPENDENCIAS

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

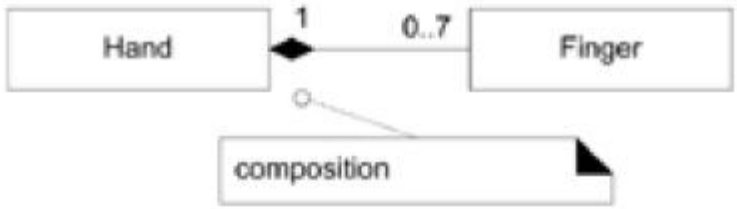


INTERFACES


[View full size image]
socket line notation
Window1 uses the Timer interface
it has a required interface
Timer
Window1
dependency line notation
Window2 has a dependency on the Timer interface when it collaborates with a Clock2 object
«interface»
Timer
getTime()
Clock2
...
getTime()
...
Timer
Window2
Clock1 implements and provides the Timer interface
Clock1
...
getTime()
...
Clock3
...
getTime()
...
Timer
Window3
lollipop notation indicates Clock3 implements and provides the Timer interface to clients
Timer is a provided interface
socket line notation
Window3 has a dependency on the Timer interface when it collaborates with a Clock3 object

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