LABORATORI 7
DISSENY DEL DCD I PAS A CODI

Disseny de Software 2016-17

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica, Universitat de Barcelona

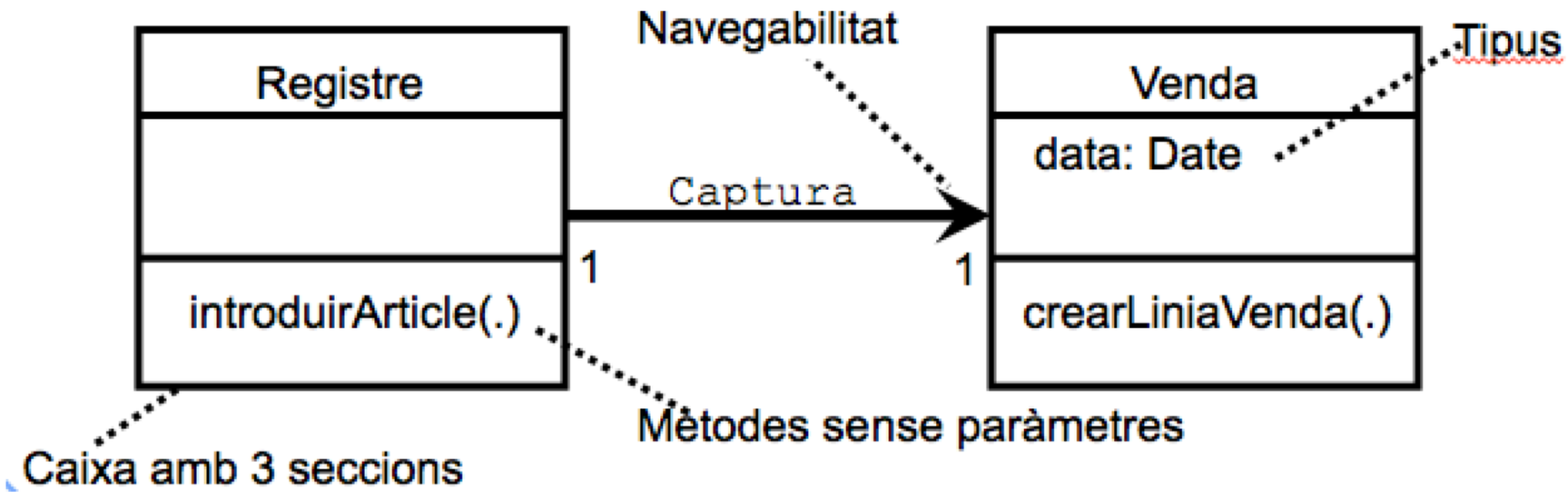
Contingut

1. Diagrama/es de Classes de Disseny
   - Visibilitat
   - Àmbit
   - Atributs
   - Associacions
   - Operacions
2. Relació amb DS
3. Pas a codi

1. Diagrama de Classes

Què és?

Un **diagrama de classes de disseny** (DCD) il·lustra les especificacions per classes software i interfícies en una aplicació



Es desenvolupa en paral·lel amb els diagrames d'interacció

Passos per crear un DCD

1. Identificar classes i il·lustrarles
2. Afegir els noms dels mètodes
3. Afegir informació de tipus i visibilitat (atributs i paràmetre
4. Afegir associacions i navegabilitat
5. Afegir relacions de dependència

Visibilitat

- **Visibilitat**: quins objectes tenen drat a consultar i modificar informació declarada en un DCD. Hi ha visibilitat sobre: **atributs** d'una classe, **operacions** i **rols**.

    - Tenim un element **x definit a la classe C**:

        - *Public (+)*: qualsevol classe que veu C, veu x

        - *Private (-)*: x només és visible des de C

        - *Protected (#)*: x és visibles des de C i des de tots els descendents de C

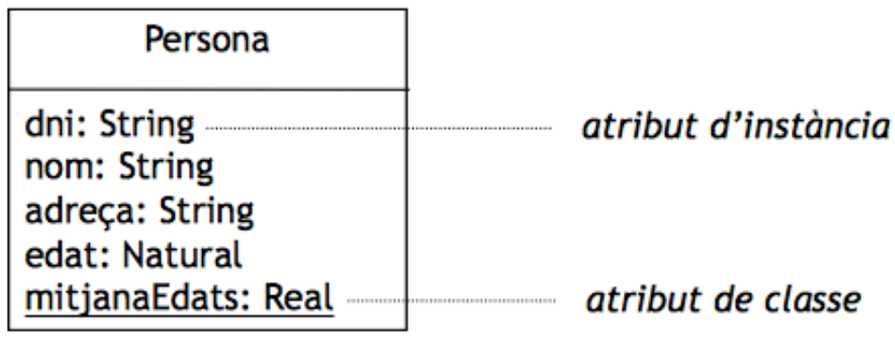
Àmbit

- **Àmbit**: ens diu si hi ha elements de disseny aplicables a objectes individuals o a la classe classe que defineix els elements. Àmbit sobre: **atributs** i **operacions** d'una classe

    - Tenim un element **x definit a la classe C**:

        - *D'instància (no estàtic):* x està associat als objectes de C. **obj.x —> obj és una instància de C**

        - De classe (estàtic): x està assiciat a C. **C.x**



Atributs: sintaxi completa

[visibilitat] nom [: tipus] [multiplicitat] [= valor-inicial] [{propietats}]

Univaluat (per defecte)
Admet valors nuls: [0..1]
Multivaluat: [1..*]
etc.

No les usem en aquesta assignatura

Public (+): qualsevol classe que veu la classe, veu l'atribut
Protected (#): només la pròpia classe i els seus descendents veuen l'atribut
Private (-): només la pròpia classe veu l'atribut

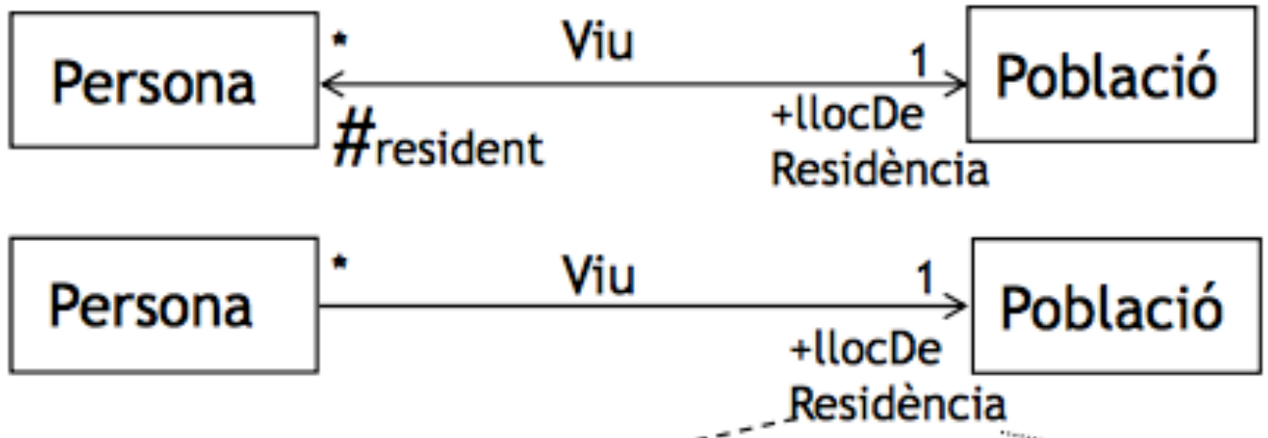
Suposarem que els atributs són privats, a no ser que especifiquem una altra visibilitat

Persona
+ dni: String # nom: String - adreça: String = "C/Nou" telèfon : Integer [0..1]

operacions *getter* i *setter*

obtéAdreça(): String
posaAdreça(n: String)

Associacions



Navegabilitat

Persona a Població
Població a Persona

Persona a Població

visibilitat: public, protected, private (com abans)

Suposarem que els rols són privats, a no ser que especifiquem el contrari

Navegabilitat: resultat del procés de disseny

- Indica si és possible o no travessar una associació binària d'una classe a una altra:
  - si *A* és navegable cap a *B*, des d'un objecte d'*A* es poden obtenir els objectes de *B* amb els què està relacionat
  - efecte: *A* té un pseudo-atribut amb la mateixa multiplicitat del rol
- Les associacions no navegables en cap sentit podrien desaparèixer de la vista lògica
  - però les podem deixar si el disseny és incomplet o per motius de canviabilitat (extensibilitat)

Operacions: sintaxi completa

No les usem a aquesta assignatura

Signatura d'una operació:

[visibilitat] nom [(llista-paràmetres)] [: tipus-retorn] [{propietats}]

Public (+): l'operació pot ser invocada des de qualsevol objecte
Protected (#): pot ser invocada pels objectes de la classe i els seus descendents
Private (-): només els objectes de la pròpia classe poden invocar l'operació

Suposarem que les operacions són públiques, a no ser que especifiquem una altra visibilitat

Paràmetres d'una operació:

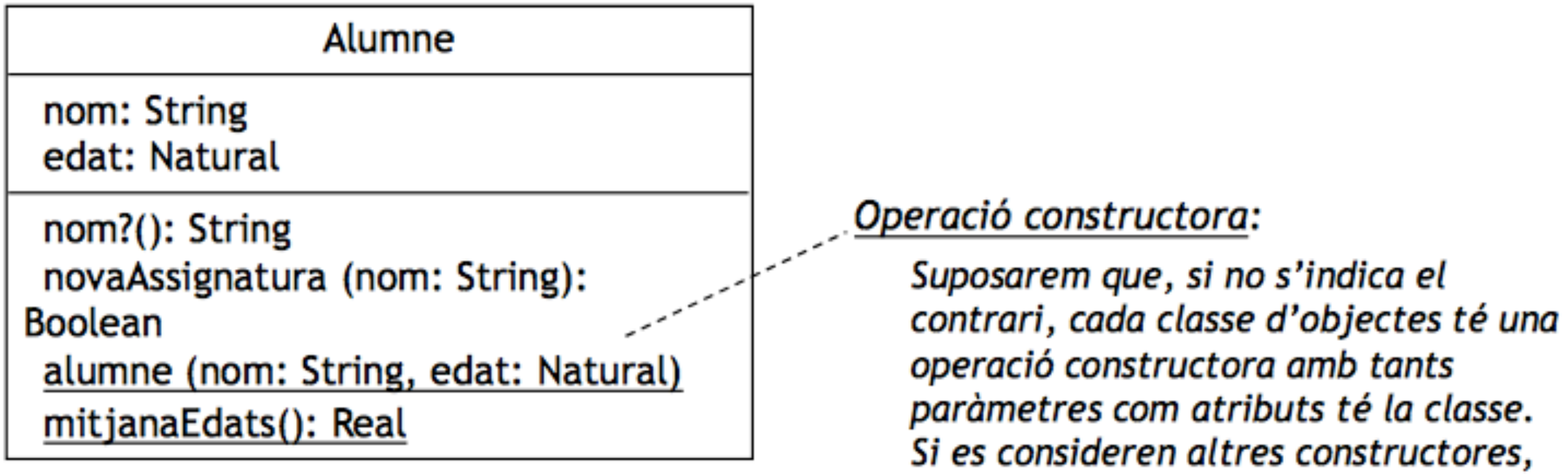
[direcció] nom: tipus [multiplicitat] [= valor-per-defecte]

in, out, inout

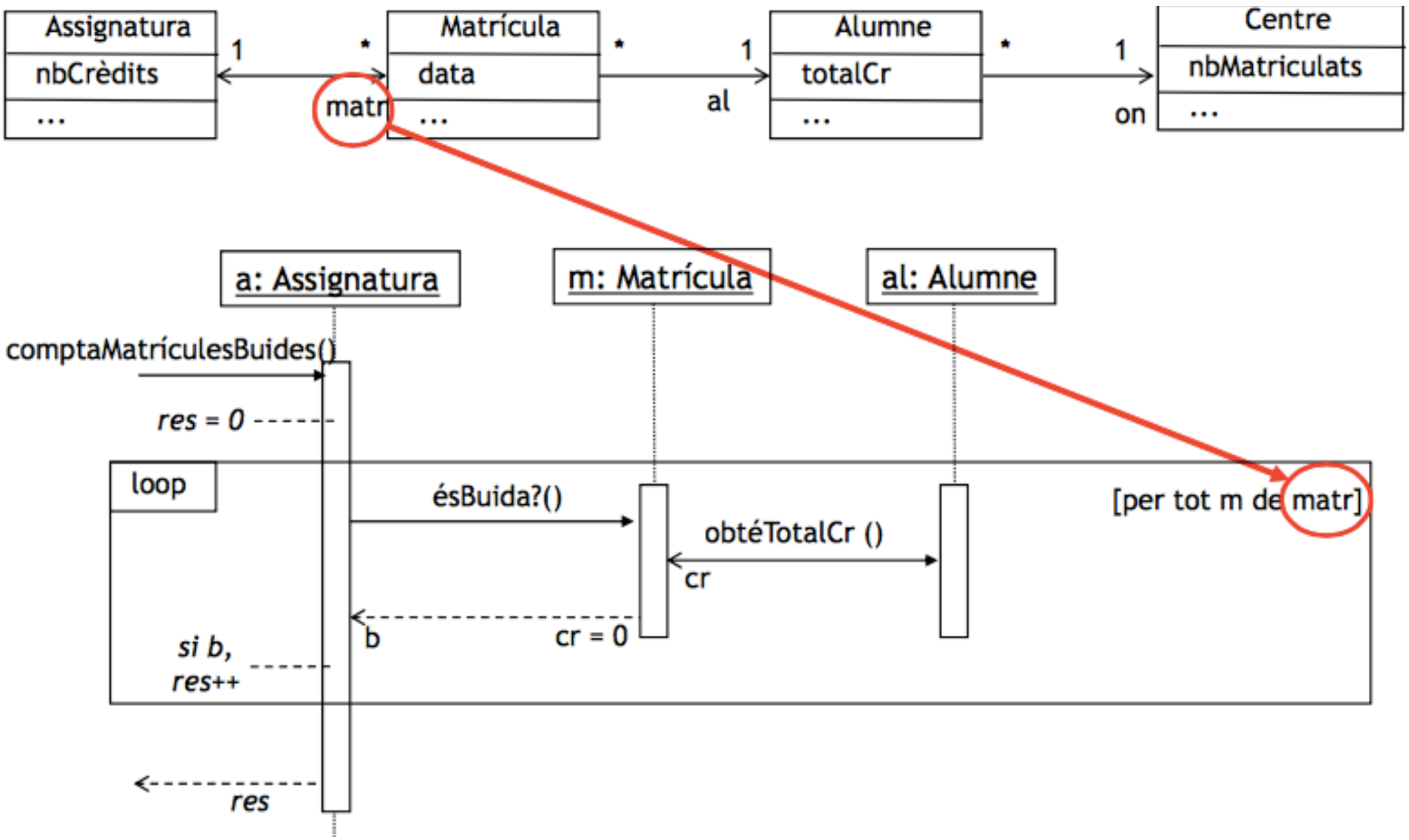
Valor per defecte

Operacions: àmbit

- Operació d'instància:
  - l'operació és invocada sobre objectes individuals
- Operació de classe:
  - l'operació s'aplica a la classe pròpiament dita
  - exemple: operacions constructores que serveixen per donar d'alta noves instàncies d'una classe



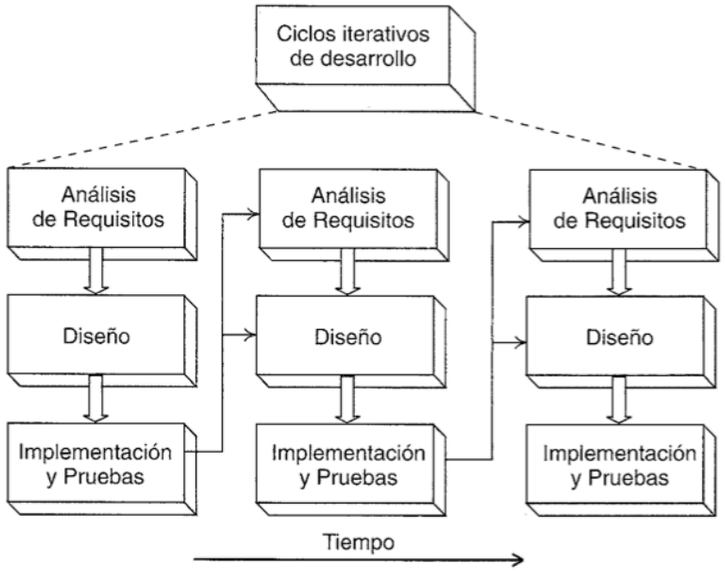
2. Rleació de DCD amb el DS



3. Generació de codi

DCD + DS

Beneficis de fer servir un sistema iteratiu.



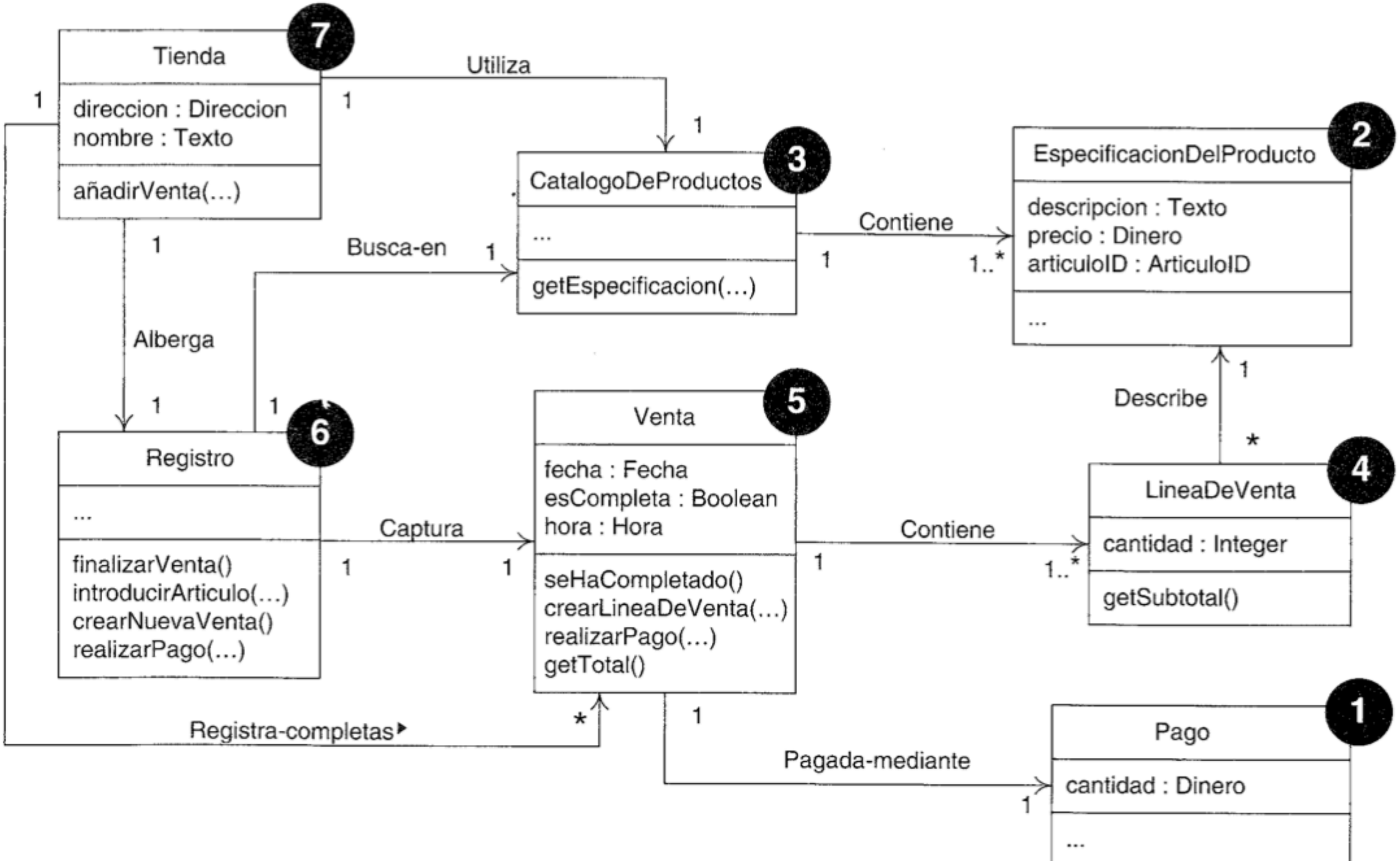
La implementació que fem en una iteració influirà en el disseny de la posterior.

Durant la implementació hi hauràn canvis: actualitzeu els diagrames que hem fet durant l'etapa de disseny !

PASSOS A SEGUIR:

1. De les classes **menys acoblades** a les **més acoblades**
2. Fem **test unitari** de cada classe
    1. Dissenyem tests unitaris (JUnit): *fem programa Prova per fer les proves bàsiques*
3. Escrivim les **classes** corresponents:
    1. Transformem els **rols** descrits en el **DCD** com a atributs de referència de la classe
    2. Fem les **Colleccions** en les relacions 1 - *
    3. Creem els **mètodes** a partir dels **DS**

1. De les classes **menys acoblades** a les **més acoblades**



2. Fem **test unitari** de cada classe

```
public class PruebaVenta extends TestCase
{
    //…
    public void pruebaTotal()
    {
      //inicializa la prueba
      Dinero total = new Dinero(7.5);
      Dinero precio = new Dinero (2.5);
      ArticuloID id = new ArticuloID (1);
      EspecificacionDelProducto espec;
      espec = new EspecificacionDelProducto(id, precio, "producto 1");
      Venta venta = new Venta();

      //añade los artículos
      venta.crearLineaDeVenta(espec, 1);
      venta.crearLineaDeVenta(espec, 2);

      //Comprobar que el total es 7.5
      assertEquals(venta.getTotal(), total);
    }
}
```

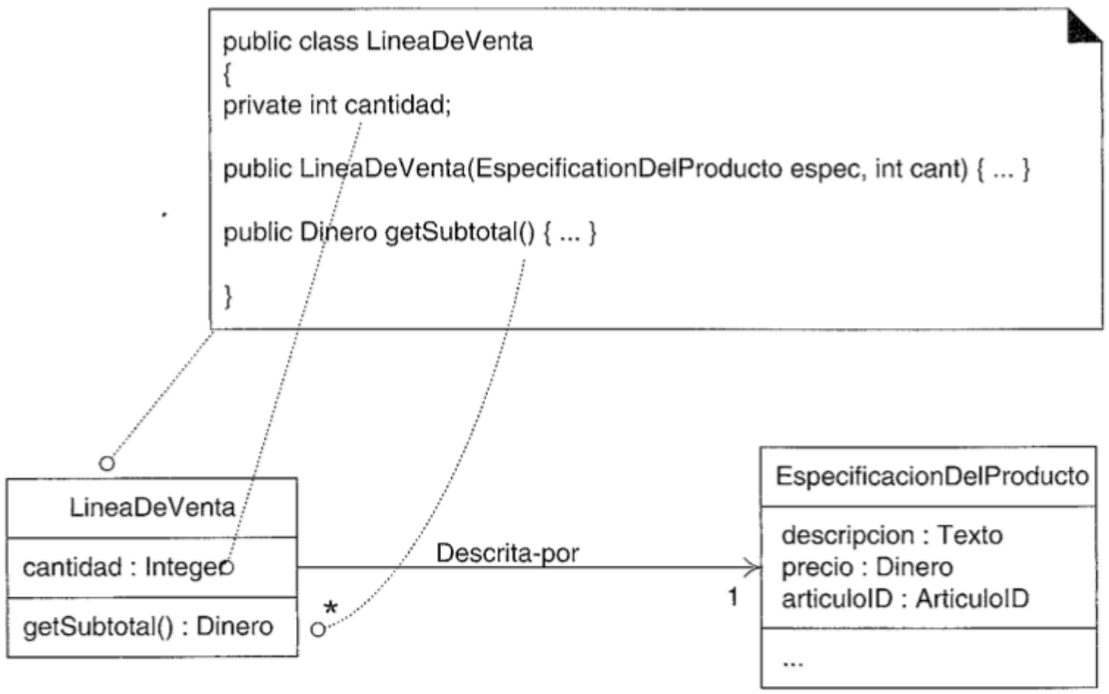
1. Fem esquelet de la classe (amb els metodes sense omplir)
2. Generar automàticament la classe test (també l'esquelet)
3. Omplim el que volem fer en els tests
4. Programem la classe bàsica.

Tools->Create/Update Tests

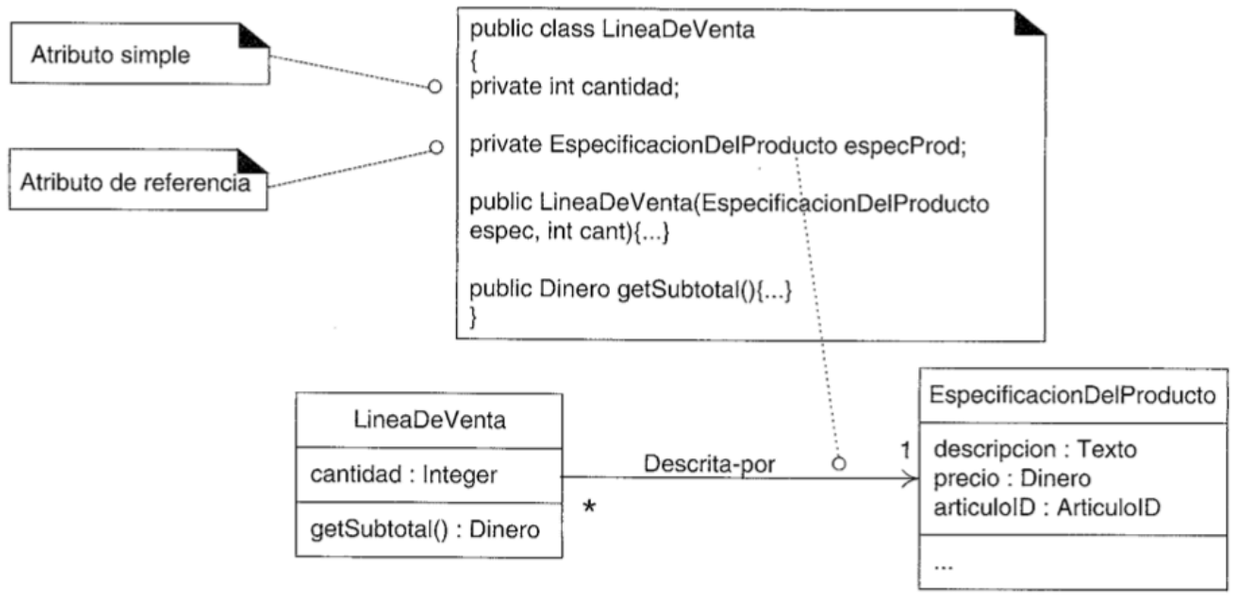
Per executar els test: Test del projecte

3. Escrivim les **classes** corresponents: **rols** descrits en el **DCD** com a atributs de referència de la classe

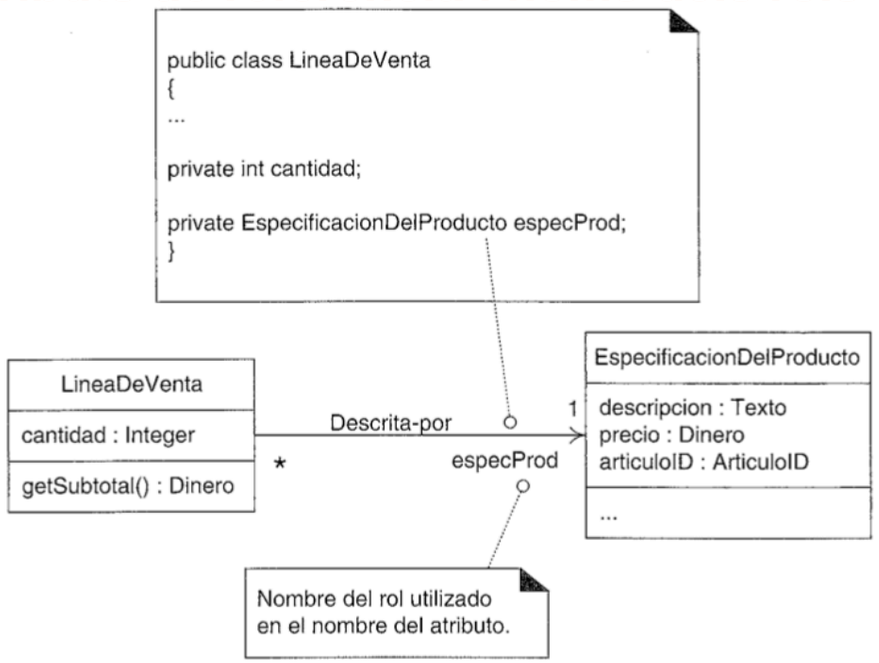
Definició de la classe amb atributs simples



Atributs de referència



Atributs de referència i noms dels rols



3. Escrivim les **classes** corresponents: **Colleccions** en les relacions 1 - *

```
public class Venta
{
...

private List lineasDeVenta = new ArrayList();
}
```

Venta
fecha: Fecha esCompleta : Boolean hora : Hora
seHaCompletado() crearLineaDeVenta() realizarPago() getTotal()

Contiene

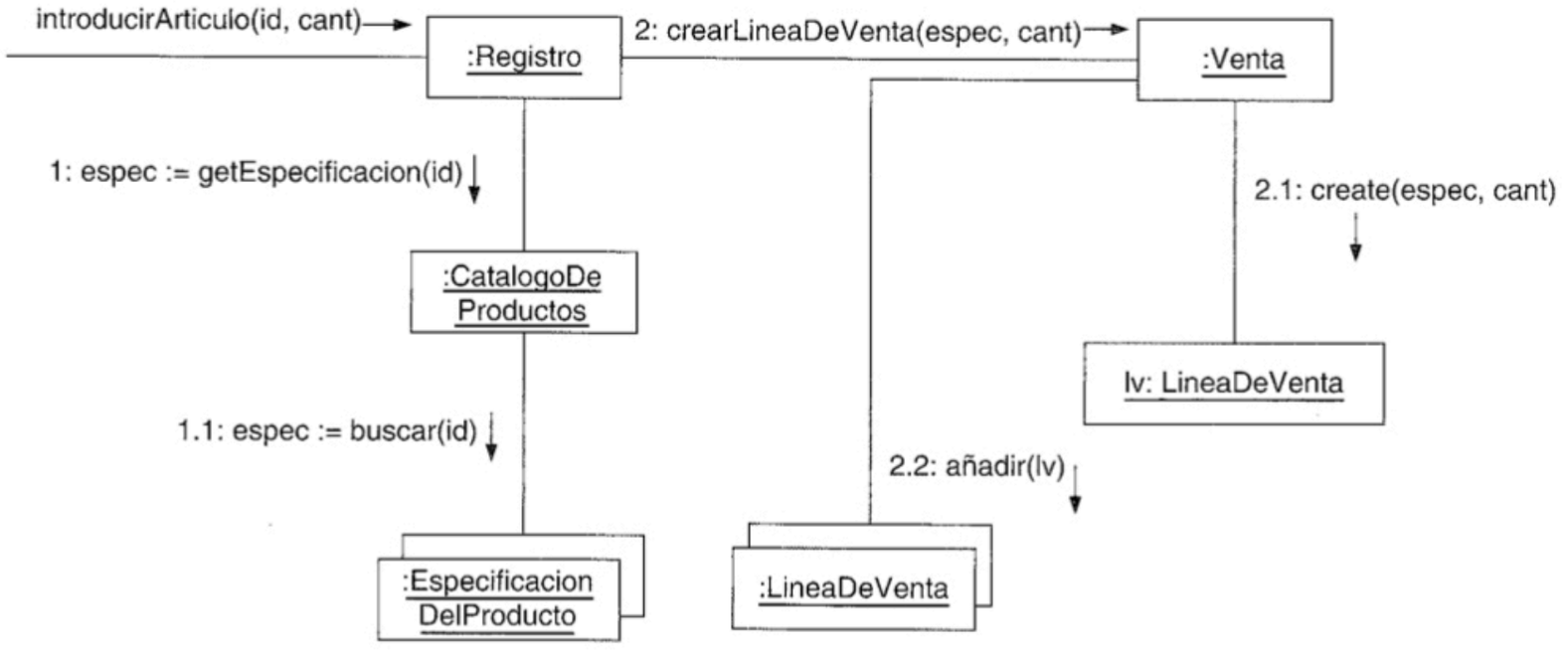
1

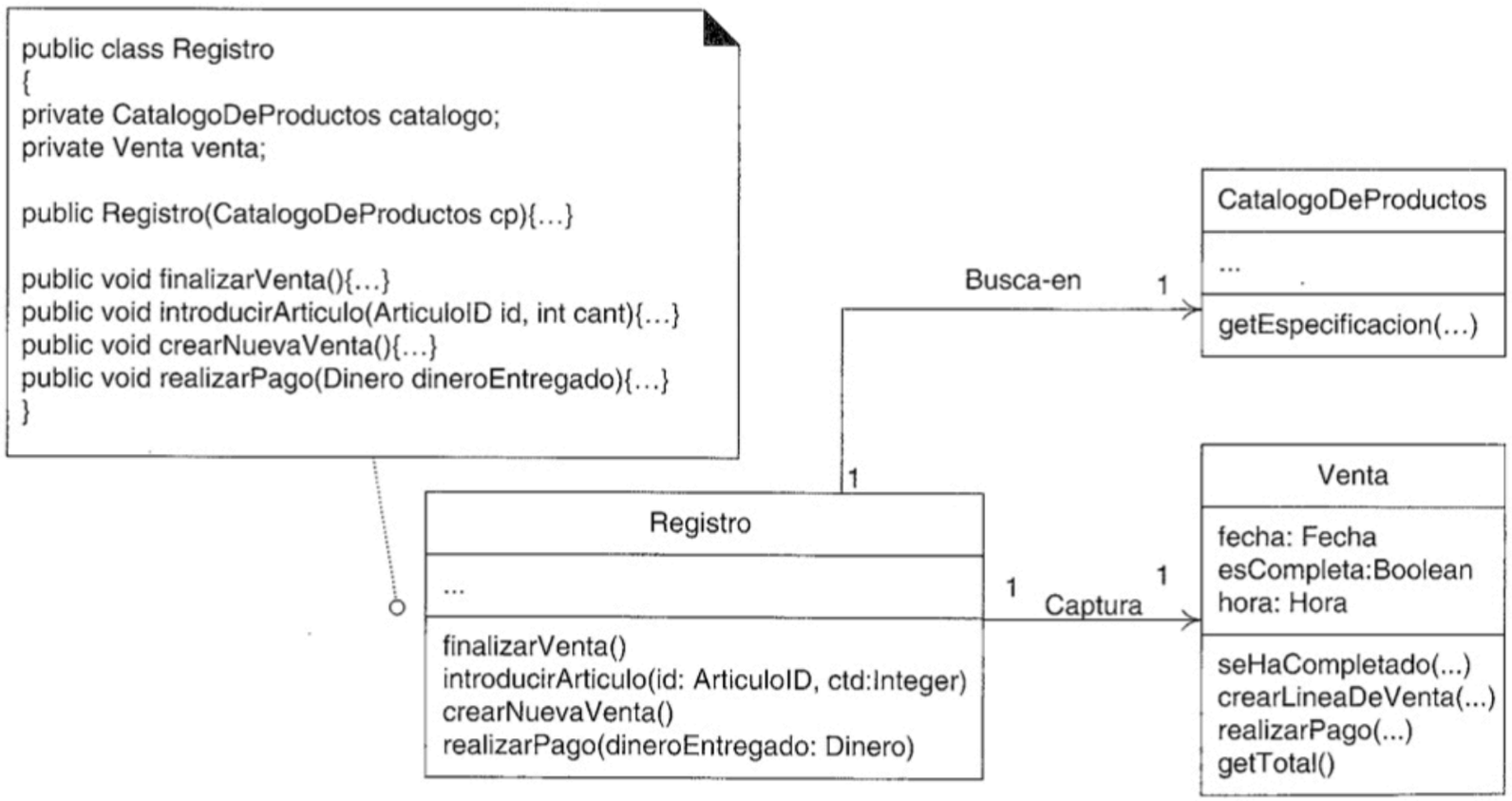
1..*

LineaDeVenta
cantidad : Integer
getSubtotal()

Es necesaria una clase colección para mantener la visibilidad de atributo a todas las LineasDeVenta.

3. Escrivim les **classes** corresponents: Creem els **mètodes** a partir dels **DS**





PLUGINS DE NETBEANS!

- UML de netbeans
- plantUML
- VisualParadigm

DCD KANGUROS