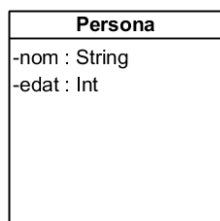


# De l'Especificació al Disseny: Diagrama de Classes i Contractes

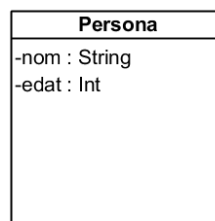
## Quina diferència hi ha?

### Especificació



- És la representació d'una **persona real**, de carn i ossos.

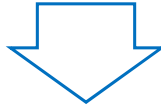
### Disseny



- És la representació d'una **classe de codi**.

## De l'Especificació al Disseny

- Al disseny hi tenim **components de software** i no conceptes de domini.
- Limitació tecnològica
  - No podem implementar directament tots els conceptes que hem usat a l'especificació



- Cal una **transformació** prèvia dels diagrames d'especificació
  - Obtenció del diagrama de classes de disseny:
    - Eliminar elements no compatibles amb la tecnologia
  - Obtenció dels contractes de disseny de les operacions
    - Controlar les restriccions d'integritat i precondicions
    - Tractar la informació derivada.

3

## Transformació del diagrama de classes

- El diagrama de classes d'especificació és més expressiu que el diagrama de classes de disseny, per tant, per caldrà:
  - Mantenir aquells elements que siguin compatibles:
    - Classes no Associatives
    - Generalitzacions/Especialitzacions Disjoint.
    - Associacions binàries
    - Atributs
  - Eliminar aquells elements que no siguin compatibles i reemplaçar-los per d'altres que sí que ho siguin:
    - Classes Associatives
    - Associacions N-àries
    - Altres casos de Generalització / Especialització
    - Classes Especials (com Data, Hora, ...)
    - Atributs derivats
  - Afegir aquells conceptes de disseny que no són necessaris a especificació.
    - Visibilitat
    - Àmbit
    - Excepcions

4

## Elements Compatibles

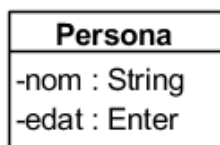
- Classes no associatives
- Generalitzacions/especialitzacions disjoint.
- Associacions binàries
- Atributs

5

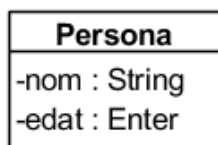
## Classes no associatives

- Les classes d'especificació són compatibles, per tant, es mantenen a disseny amb la mateixa forma.

### Especificació



### Disseny



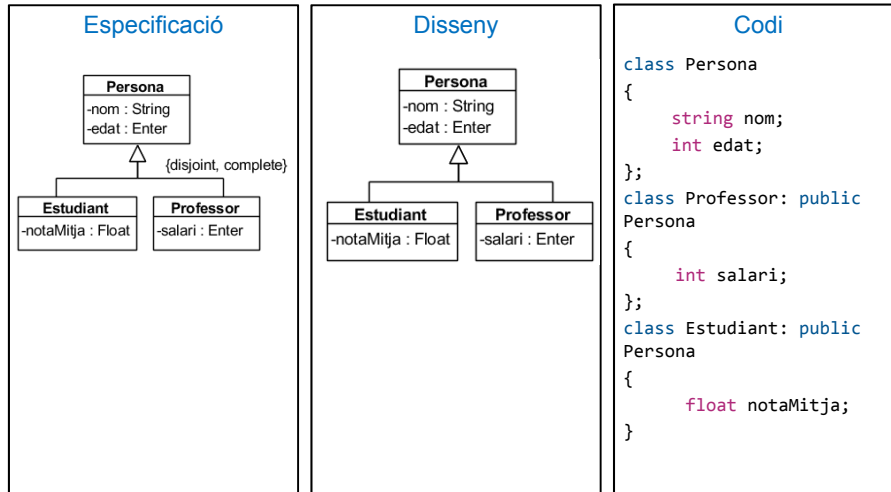
### Codi (C++)

```
class Persona
{
    string nom;
    int edat;
};
```

6

## Generalitzacions/Especialitzacions disjoint

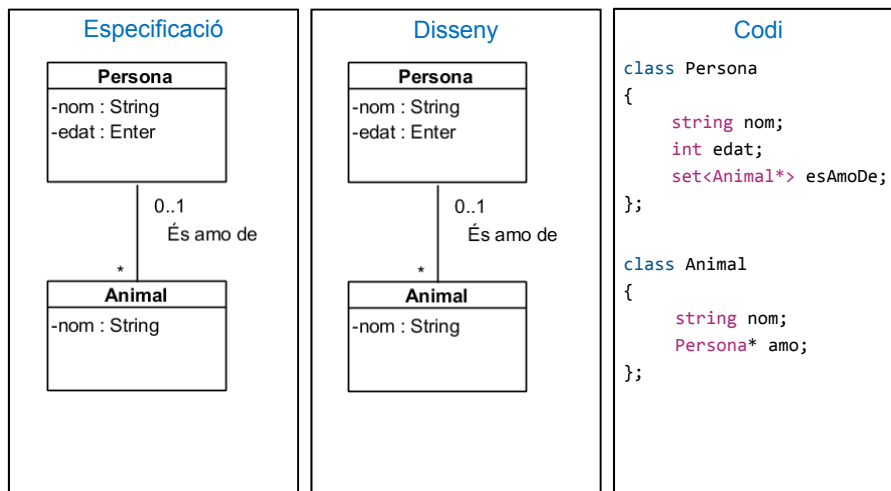
- Les herències disjointes són compatibles, per tant es mantenen igual



7

## Associacions binàries

- Les associacions binàries es mantenen igual, tot i que a disseny els haurem d'afegir **navegabilitat** (al proper tema).



8

## Elements No Compatibles

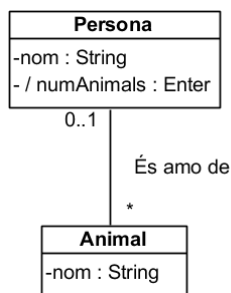
- Atributs derivats
- Classes especials (com Data, Hora, ...)
- Classes associatives
- Associacions N-àries
- Altres casos de generalització / especialització

9

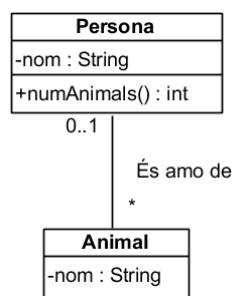
## Atributs Derivats (I)

- Els atributs derivats es poden convertir en una funció nova que calcula el valor. Si són així, direm que són atributs **Calculats**

### Especificació



### Disseny



### Codi

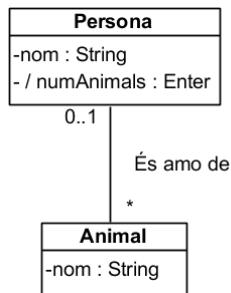
```
class Persona
{
    string nom;
    set<Animal*> esAmoDe;
    int numAnimals()
    {
        return esAmoDe.size();
    }
};
```

10

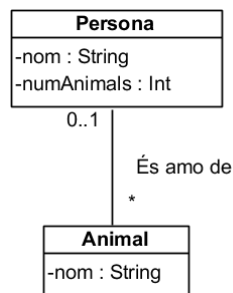
## Atributs Derivats (II)

- Els atributs derivats es poden convertir en un nou atribut "real" que s'haurà de mantenir actualitzat. Si són així, direm que s'han **Materialitzat**

### Especificació



### Disseny



### Codi

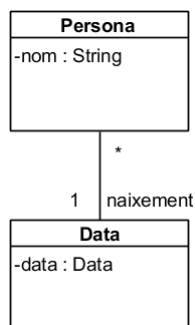
```
class Persona
{
    string nom;
    set<Animal*> esAmoDe;
    int numAnimals;
};
```

11

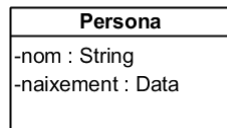
## Classes Especials (Data, Hora, ...)

- Les classes especials esdevenen atributs amb el tipus corresponent, per tant, s'ha d'eliminar la classe original i afegir com a un atribut.

### Especificació



### Disseny



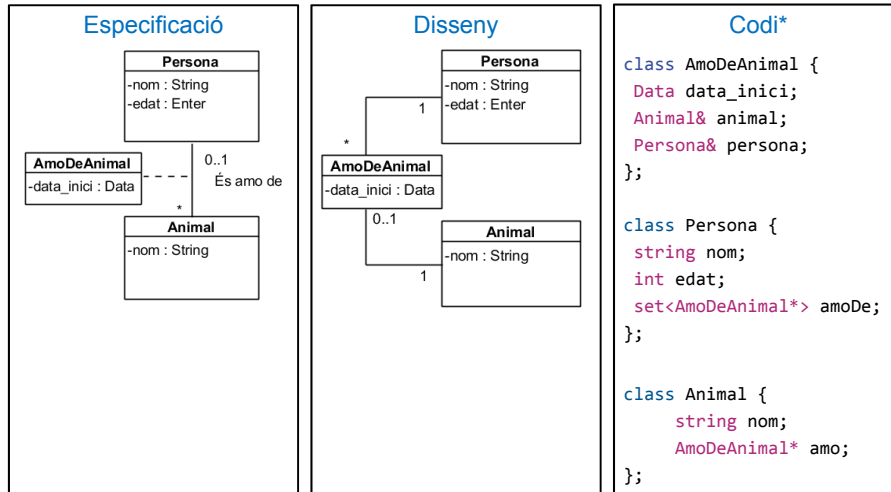
### Codi

```
class Persona
{
    string nom;
    Data naixement;
};
```

12

## Classes Associatives (I)

- Les classes associatives es converteixen en classes no associatives i es relacionen amb les dues originals...



13

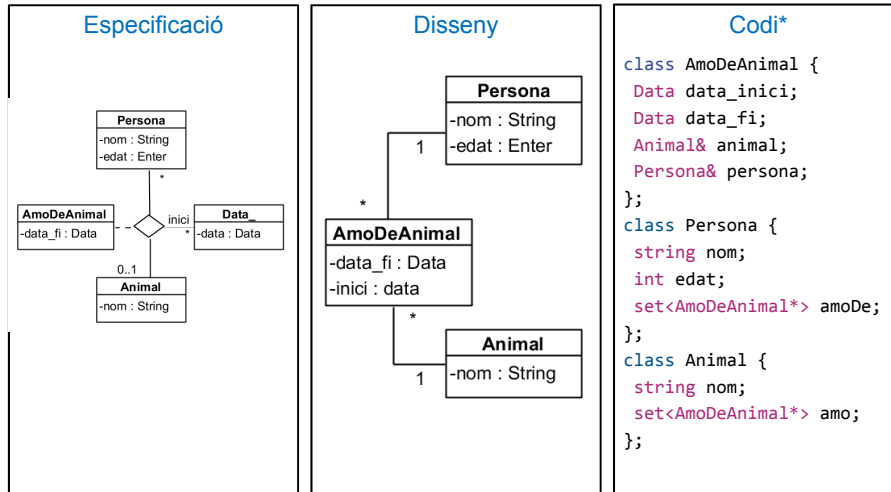
## Classes Associatives (II)

- El diagrama obtingut ha de tenir la mateixa semàntica que l'original, per tant, haurem d'afegir restriccions textuais per a compensar la pèrdua d'informació.
  - No hi pot haver dos "AmoDeAnimal" amb els mateixos Persona i Animal

14

## Associacions N-àries (I)

- Les associacions N-Àries s'han de canviar per una classe associativa relacionada amb les N originals...



15

## Associacions N-Àries(II)

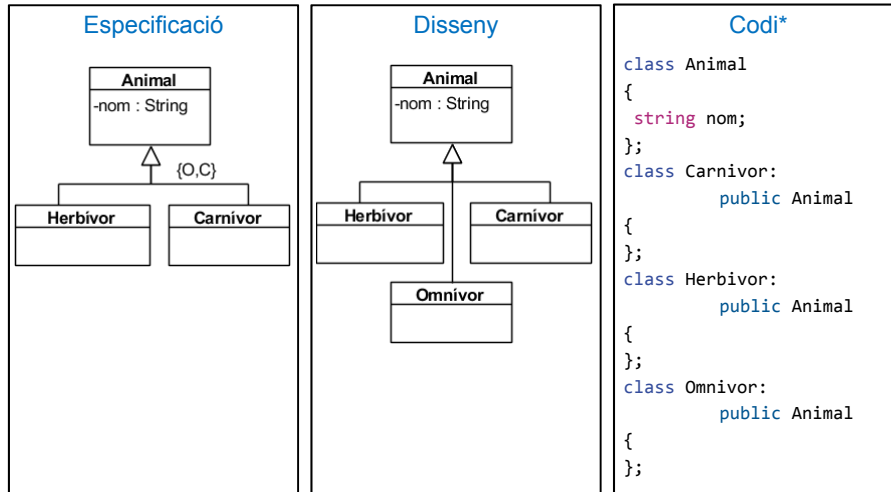
- El diagrama obtingut ha de tenir la mateixa semàntica que l'original, per tant, haurem d'afegir restriccions textuais per a compensar la pèrdua d'informació (igual que amb les classes associatives).
  - RT1: No hi pot haver dos "AmodeAnimal" amb els mateixos Persona, Animal i Inici
- També s'han de considerar les multiplicitats de cada un dels membres de la N-ària per a afegir noves restriccions
  - RT2: Donada una persona i una Data, màxim pot esdevenir amo d'un Animal
- I això pot provocar que algunes restriccions textuais siguin redundants.
  - RT1 és redundant amb RT2, per tant, no s'ha d'afegir RT1

16



## Generalització / Especialització No Disjoint

- No hi ha una forma única de fer-ho depèn de cada situació. El producte cartesià és només una de les opcions, però n'hi ha més.



17

## Elements Exclusius de Disseny

- Visibilitat
- Àmbit
- Excepcions

18

## Visibilitat

- Defineix quins objectes tenen dret a consultar i modificar informació declarada en un diagrama de classes
- Pot ser de tres tipus
  - Pública (+)
  - Privada (-)
  - Protegida (#)
- Aplica a:
  - Atributs
  - Operacions
  - Rols
- A IES assumirem que, per defecte, els atributs i rols són privats, i les operacions són públiques.

19

## Visibilitat - Públic

- Donat un element X d'una classe C
  - Si és públic, qualsevol que vegi C, veurà X

```
class Animal
{
    public:
        string nom;
};
```

```
void main()
{
    Animal animal;
    animal.nom = "Nix";
}
```

20

## Visibilitat - Privat

- Donat un element X d'una classe C
  - Si és privat, només C veurà X

```
class Animal                                void main()
{
    private:                                {
        string nom;                        Animal animal;
    };                                     animal.nom = "Nix";
}
```

21

## Visibilitat - Protegit

- Donat un element X d'una classe C
  - Si és protegit, només C o els seus descendents veuran X

```
class Animal {                                void main()
protected:                                {
    string nom;                            Gos animal;
};                                         animal.nom = "Nix";
class Gos: public Animal {                animal.CanviaNom("Nix");
public:                                    }
    void CanviaNom(string nouNom) {
        nom = nouNom;
    }
};
```

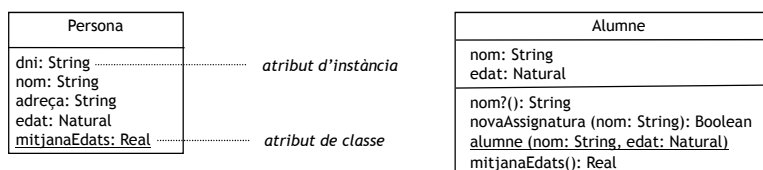
22

## Àmbit

- Determina si els atributs o operacions són aplicables a objectes individuals o a la classe que defineix els elements
- Poden ser:
  - De classe (estàtic)
    - X està associat al C
  - D'instància (no estàtic)
    - X està associat als objectes de C
- Els atributs o operacions de classe (estàtics) es marquen subratllant el nom de la operació / atribut

23

## Àmbit - Exemples



24

## Disseny sense precondicions

- A diferència d'especificació, a disseny no hi ha cap sistema que ens protegeixi de violar restriccions d'integritat, multiplicitats, precondicions o qualsevol protecció que ens doni el diagrama de classes de forma natural.



- Necessitem un mètode de poder controlar i informar de les restriccions que es violin al cridar una funció.
- Les Precondicions ja no tenen sentit!

25

## Excepcions

- Els llenguatges de programació ens proporcionen diverses maneres de poder controlar els casos en els que la operació no s'ha pogut realitzar per causa d'algun problema.
- La més utilitzada és mitjançant l'ús d'**Excepcions**.

26

## Gestió d'Excepcions

- Segons Wikipedia:
  - La gestió d'excepcions és una tècnica de programació que permet al programador controlar els errors ocasionats durant l'execució d'un programa informàtic. Quan es produeix algun tipus d'error, el sistema reacciona executant un fragment de codi que resolgui la situació, per exemple, retornant un missatge d'error o retornant un valor per defecte.
- Informal
  - És la manera de controlar i gestionar problemes. Quan es produeixi un problema, llencem una excepció per a notificar-ho, això atura la execució.

27

## Com programariem això?

```
Context: Sistema::FesLaResta(  
    num1: Enter Positiu,  
    num2: Enter Positiu  
): Enter Positiu  
Pre: num1 >= num2  
Body: result = num1 - num2;
```

28

## Així?

```
unsigned int FesLaResta(  
    unsigned int num1,  
    unsigned int num2)  
{  
    return num1 - num2;  
}
```

- Si num1 és més petit que num2, això provocarà un error al sistema que no volem que passi. La precondició ens protegia d'això, i ara ja no hi és.

29

## Així!

```
Class Num1Menor: públic std::exception  
{  
  
    unsigned int FesLaResta(  
        unsigned int num1,  
        unsigned int num2)  
    {  
        if (num1 < num2)  
        {  
            throw Num1Menor();  
        }  
        return num1 - num2;  
    }  
}
```

30

## Contractes de les Operacions a Disseny

- S'elimina la secció de precondicions
- S'afegeix una secció d'excepcions
  - Per cada possible problema que la operació pugui provocar crearem una nova excepció. La definirem amb un **nom explicatiu i una descripció**.

31

## Contractes de les Operacions a Disseny

```
Context: Sistema::FesLaResta(  
    num1: Enter Positiu,  
    num2: Enter Positiu  
): Enter Positiu  
Excepcions:  
    - Num1Menor: num1 és menor que num2  
Body: result = num1 - num2;
```

32



## Excepcions possibles

- Les excepcions que es poden produir en una operació poden ser dels següents tipus:
  - Violació d'una precondició
  - Violació d'una restricció textual
  - Violació d'una restricció gràfica
  - Violació d'una restricció del model (implícita)
- Per cada operació haurem de fer una anàlisi i veure quines es poden produir.

33

## Violació d'una precondició

### Especificació

**Context:** AltaPersona(  
    nom: String,  
    edat: Enter,  
    nomCiutat: String)

**Pre:** nomCiutat existeix

**Post:** Es dona d'alta una nova persona amb el nom, i l'edat introduïts, associada a la Ciutat nomCiutat

### Disseny

**Context:** AltaPersona(  
    nom: String,  
    edat: Enter,  
    nomCiutat: String)

#### Excepcions:

- [CiutatNoExisteix] la Ciutat nomCiutat no existeix

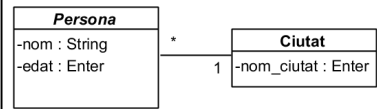
**Post:** Es dona d'alta una nova persona amb el nom, i l'edat introduïts, associada a la Ciutat nomCiutat

- Per cada Precondició que hi hagi a la nostra operació, haurem d'afegir una excepció nova

34

## Violació d'una restricció textual

### Especificació



RT1 (Persona, nom), (Ciutat, nom)  
RT2: edat de Persona >= 18

---

**Context:** AltaPersona(  
    nom: String,  
    edat: Enter,  
    ciutat: String)

**Pre:** nomCiutat existeix

**Post:** Es dona d'alta una persona nova amb el dni, ciutat i edats introduïts.

### Disseny

**Context:** AltaPersona(  
    nom: String,  
    edat: Enter,  
    ciutat: String)

#### Excepcions:

- [CiutatNoExisteix] la Ciutat nomCiutat no existeix
- [PersonaExisteix] Persona amb nom = nom existeix
- [MenorD'Edat] edat < 18

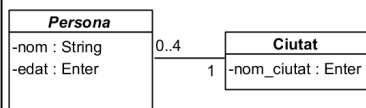
**Post:** Es dona d'alta una persona nova amb el nom, ciutat i edats introduïts.

- Per cada restricció d'integritat que es pugui violar, haurem d'afegir una nova excepció.

35

## Violació d'una restricció gràfica

### Especificació



RT1 (Persona, nom), (Ciutat, nom)

---

**Context:** AltaPersona(  
    nom: String,  
    edat: Enter,  
    nomCiutat: String)

**Pre:** nomCiutat existeix

**Post:** Es dona d'alta una persona nova amb el dni, ciutat i edats introduïts.

### Disseny

**Context:** AltaPersona(  
    nom: String,  
    edat: Enter,  
    ciutat: String)

#### Excepcions:

- [CiutatNoExisteix] la Ciutat nomCiutat no existeix
- [PersonaExisteix] Persona amb nom = nom existeix
- [MassaGent] ciutat ja te 4 persones

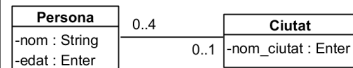
**Post:** Es dona d'alta una persona nova amb el nom, ciutat i edats introduïts.

- Per cada multiplicitat que es pugui violar, cal una nova excepció

36

## Violació d'una restricció del model (implícita)

### Especificació



RT1 (Persona, nom), (Ciutat, nom)

**Context:** AfegirACiutat(  
    nomPersona: String,  
    nomCiutat: String)

**Pre:** nomCiutat existeix  
      nomPersona existeix

**Post:** Es dona d'alta una persona nova  
amb el dni, ciutat i edats introduïts.

### Disseny

**Context:** AfegirACiutat(  
    nomPersona: String,  
    ciutat: String)

#### Excepcions:

- [CiutatNoExisteix] Ciutat amb nom\_ciutat = nomCiutat no existeix
- [PersonaNoExisteix] Persona amb nom = nomPersona no existeix
- [PersonaJaViu] La Persona amb nom = nomPersona ja viu a la ciutat amb nom\_ciutat = nomCiutat

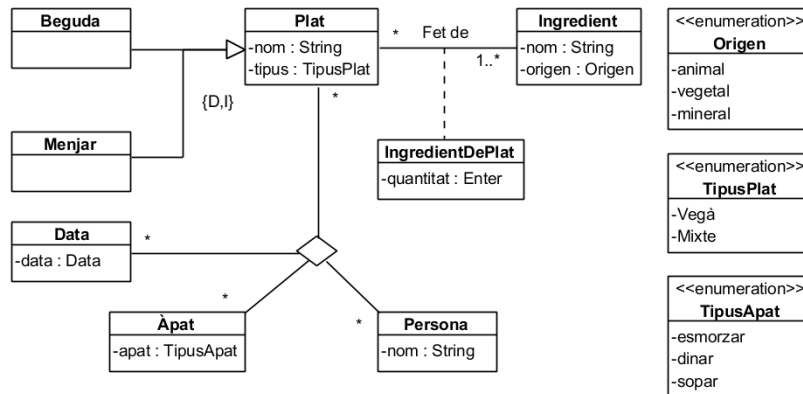
**Post:** Es dona d'alta una persona nova amb el nom, ciutat i edats introduïts.

- Per cada multiplicitat que es pugui violar, cal una nova excepció

37

## Exemple de traducció

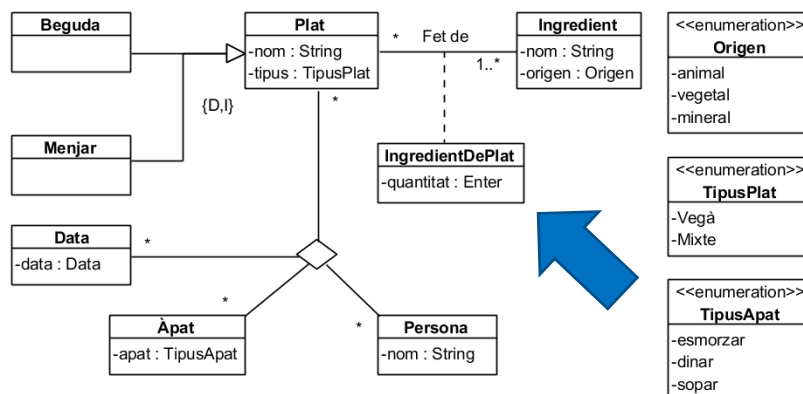
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal

39

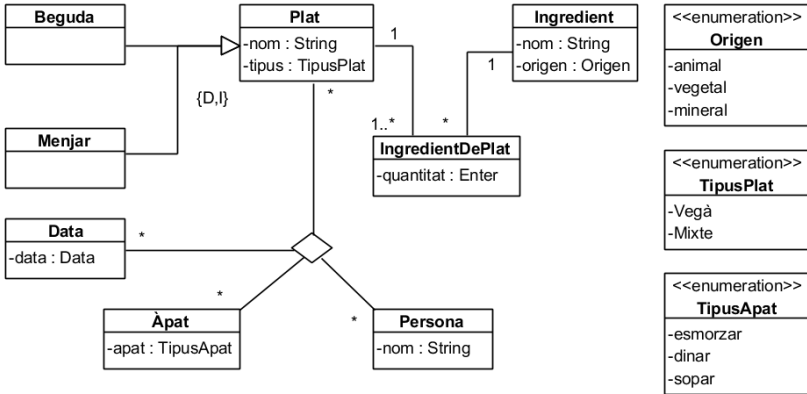
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal

40

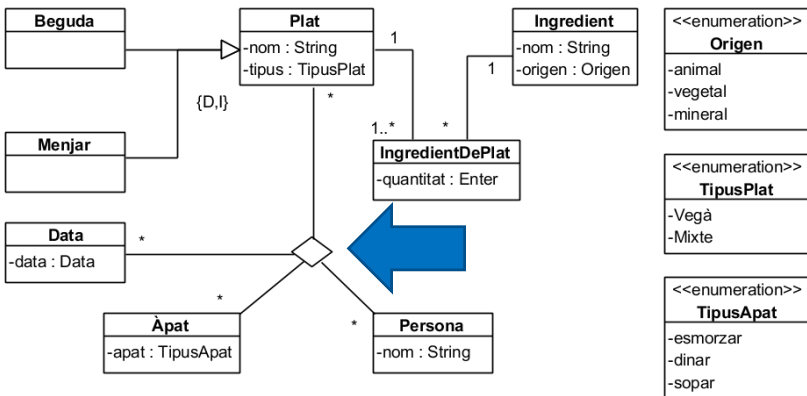
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- **RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat**

41

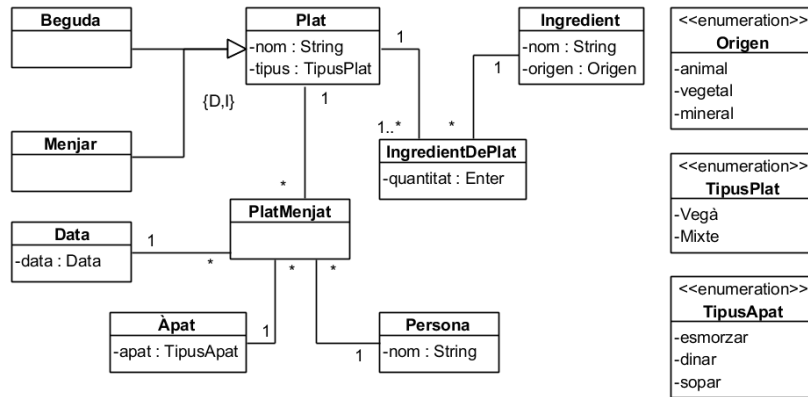
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat

42

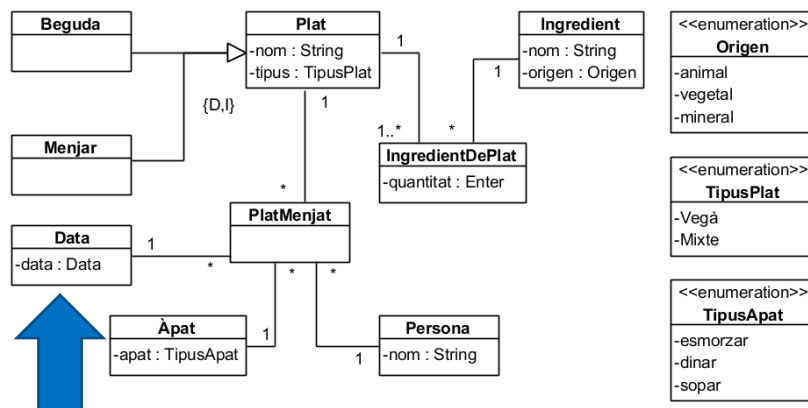
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat
- **RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Data i Persona**

43

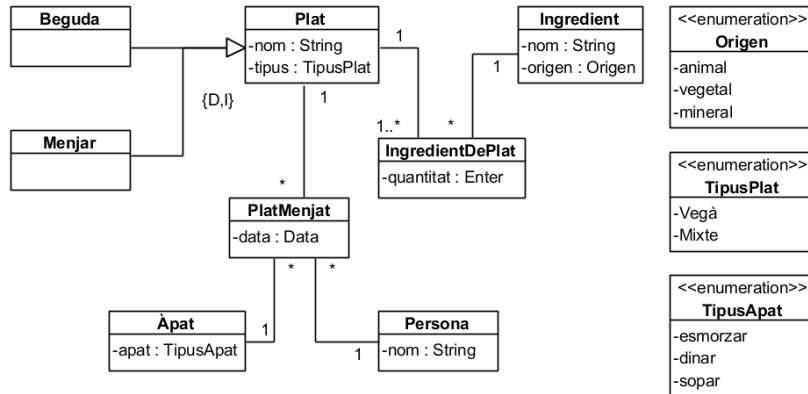
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 ingredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Persona i Data

44

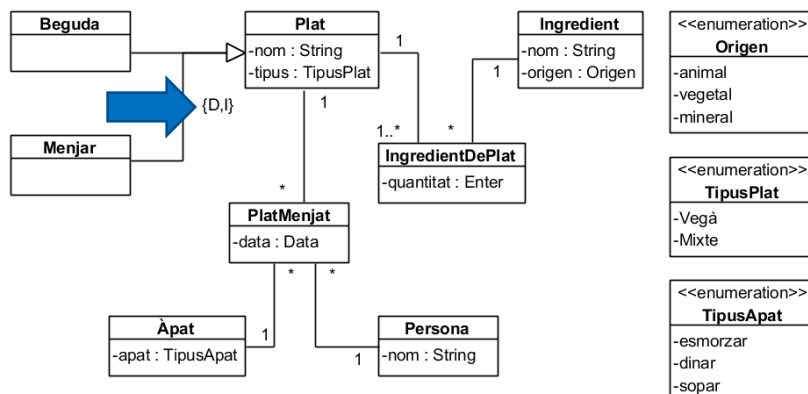
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 ingredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Persona i Data

45

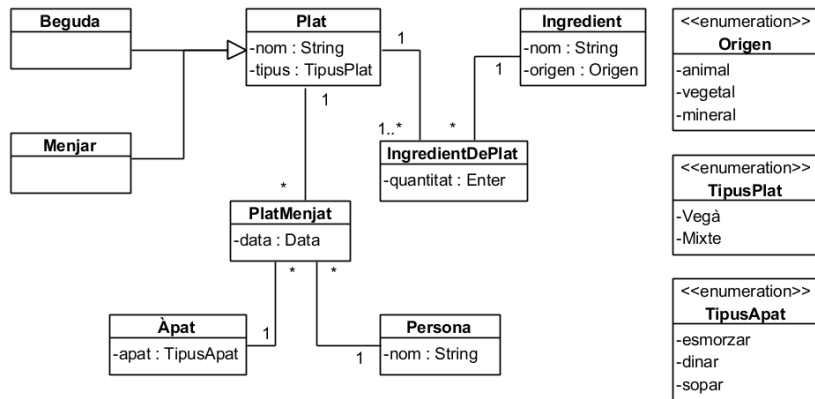
## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 ingredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Persona i Data

46

## Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 ingredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Persona i Data

47

## Bibliografia

- Pressman, R.G. "Software Engineering. A Practitioner's Approach", Mc Graw-Hill, 2015 (8a edició).
- Larman, C. "Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design", Prentice Hall, 2005, (3ª edició)
- Meyer, B. "Object-oriented Software Construction", Prentice Hall, 1997.

48