

ENGINYERIA DEL SOFTWARE II

Orientació a Objectes

GEINF - GDDV

Departament Informàtica i Matemàtica Aplicada
Escola Politècnica Superior
Universitat de Girona

Curs2025/26

IMPORTANT

Aquest document conté les *diapositives* que faig servir a classe. No està pensat ni com uns apunts ni, molt menys, com un capítol d'un *llibre de text*.

Només us ho deixo al Moodle per si us és útil per seguir les classes i caldria que ho complementéssiu amb les explicacions de classe o la bibliografia recomanada.

Prèvia

Sobre aquest tema...

Dedicarem dues o tres sessions a repassar diversos aspectes del disseny i programació OO i del llenguatge UML que farem servir durant l'assignatura.

No vol ser, ni molt menys, una explicació completa d'OO ni d'UML.

Taula de continguts

- 1 Conceptes Orientació a Objectes
 - 2 *Unified Modeling Language (UML)*
 - Pinzellades UML
 - 3 Recordem alguns diagrames UML
 - Diagrames de classes
 - Una mica de codi
 - Casos d'ús
 - Diagrames de casos d'ús
 - Fitxes de casos d'ús
 - Diagrama d'activitats
 - Diagrames d'interacció
 - Diagrames de seqüència i comunicació

Classes i objectes

Classes Una classe la podem veure com un mòdul que gestiona el seu estat. Una classe té atributs, operacions i associacions.

Objectes Un objecte és una instància d'una classe. Un objecte és un estat (dades) i un comportament (operacions)

A grosso modo...

Una classe defineix. Un objecte executa.

Conceptes claus OO

ABSTRACCIÓ

ENCAPSULATION

HERÈNCIA

POLIMORFISME

Conceptes claus OO: Encapsulament



- Visibilitat atributs i operacions
- *getters* i *setters*
- ...

Conceptes claus OO: Herència



- Especialització/Generalització
- Classificació simple vs classificació múltiple
- Herència múltiple
- ...

Conceptes claus OO: Polimorfisme



- Una instància pot ser dues classes a l'hora?
- Podem assignar instàncies d'una classe subclasse a una variable del tipus d'una superclasse?
- Sobreescriptura de mètodes. Quina versió s'executa?
- ...

Taula de continguts

- 1 Conceptes Orientació a Objectes
- 2 *Unified Modeling Language (UML)*
 - Pinzellades UML
- 3 Recordem alguns diagrames UML
 - Diagrames de classes
 - Una mica de codi
 - Casos d'ús
 - Diagrames de casos d'ús
 - Fitxes de casos d'ús
 - Diagrama d'activitats
 - Diagrames d'interacció
 - Diagrames de seqüència i comunicació

Unified Modeling Language (UML)

Referència

- Estàndard UML gestionat pel *Object Management Group*(OMG) <https://www.omg.org/>
 - Informació sobre UML: <https://www.uml.org/>.
 - L'última versió de l'estàndard d'UML existent maig del 2023 és la 2.5.1 (desembre 2017)

Unified Modeling Language (UML)

Referència

The screenshot shows a web browser displaying the OMG (Object Management Group) website at <https://www.omg.org/spec/UML/#spec-versions>. The page lists the historical versions of the UML specification, their adoption dates, and URLs. The columns are labeled VERSION, ADOPTION DATE, and URL. The data is presented in a table with 11 rows, starting from version 1.1 (desembre 1997) up to version 2.5.1 (desembre 2017). Each row contains a link to the specific UML specification document.

VERSION	ADOPTION DATE	URL
2.5.1	de desembre 2017	https://www.omg.org/spec/UML/2.5.1
2.4.1	de juliol 2011	https://www.omg.org/spec/UML/2.4.1
2.3	de maig 2010	https://www.omg.org/spec/UML/2.3
2.2	de gener 2009	https://www.omg.org/spec/UML/2.2
2.1.2	d'octubre 2007	https://www.omg.org/spec/UML/2.1.2
2.0	de juliol 2005	https://www.omg.org/spec/UML/2.0
1.5	de març 2003	https://www.omg.org/spec/UML/1.5
1.4	de setembre 2001	https://www.omg.org/spec/UML/1.4
1.3	de febrer 2000	https://www.omg.org/spec/UML/1.3
1.2	de juliol 1999	https://www.omg.org/spec/UML/1.2
1.1	de desembre 1997	https://www.omg.org/spec/UML/1.1

Unified Modeling Language (UML)

UML en el procés de desenvolupament del software

- UML el podem veure com un llenguatge a partir del qual podríem arribar a generar codi (plànot, *blueprint*) o com una ajuda per representar idees (esborrany, *sketch*). **Nosaltres ens quedem amb la segona opció, però respectant la sintaxi del llenguatge.**
- **Mètodes iteratius vs cascada** cascada divideix un projecte en 4 fases (anàlisi, disseny, implementació i prova) i iteratius divideix el projecte en subprojectes, cadascun amb les 4 fases. **UML l'aplicarem on ens sigui útil amb independència del mètode que seguim.**

Unified Modeling Language (UML)

Diagrames definits per UML: **Diagrames d'estructura**

Diagrama		Des de versió
de classes	Class	UML1
de components	Component	UML1
de desplegament	Deployment	UML1
d'estructura composta	Composite structure	UML2
d'objecte	Object	UML2 (no oficial a UML1)
de paquets	Package	UML2 (no oficial a UML1)

Unified Modeling Language (UML)

Diagrames definits per UML: **Diagrames de Comportament**

Diagrama		Des de versió
d'activitats	Activity	UML1
de casos d'ús	Use case	UML1
d'estats	State machine	UML1
D'interacció	Interaction	(taula següent)

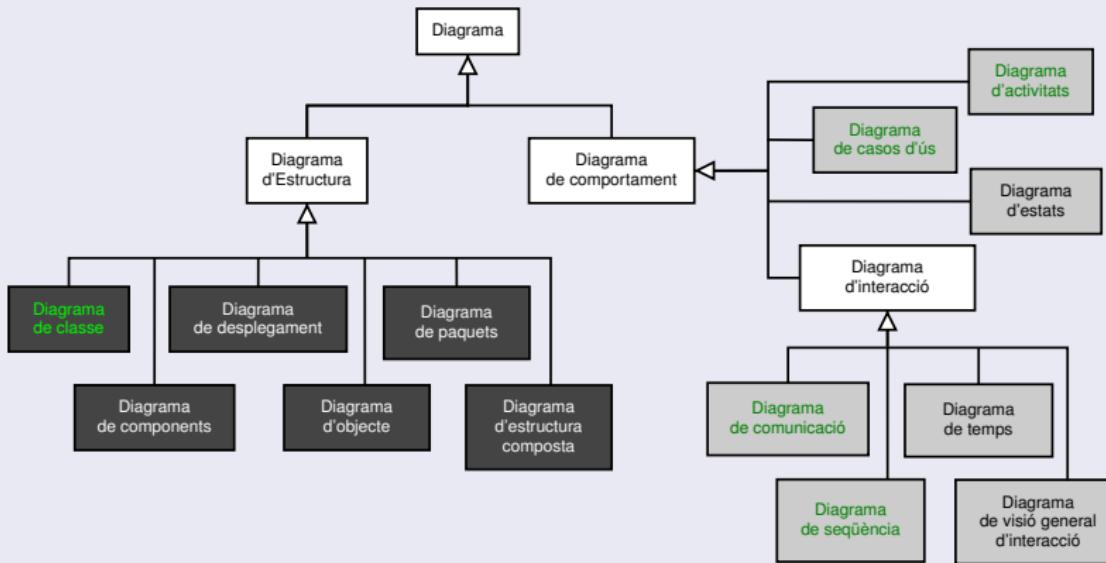
Unified Modeling Language (UML)

Diagrames definits per UML: **Diagrames de Comportament (Interacció)**

Diagrama		Des de versió
de comunicació	Communication	UML2 (diagrama col·laboració, UML1)
de seqüència	Sequence	UML1
de temps	Timing	UML2
de visió general d'interacció	Interaction overview	UML2

Unified Modeling Language (UML)

Diagrames definits per UML



Informació addicional: What is Unified Modeling Language (UML) (web de Visual Paradigm)

Unified Modeling Language (UML)

Ús de l'UML en les diferents fases (anàlisi)

Anàlisi de requisits. Busquem determinar el que el nostre client i/o els usuaris volen que faci el sistema. Podem fer servir:

- Diagrama de casos d'ús
- Diagrama de classes (visió domini)
- Diagrama d'activitats (per descriure fluxes d'accions)
- Diagrama d'estats (per descriure cicles de vida d'objectes)

Sempre tenint present que és vital poder-se entendre amb els clients/usuaris (fem servir una notació comprensible)

Unified Modeling Language (UML)

Ús de l'UML en les diferents fases (disseny)

Disseny Refinem l'anàlisi per apropar-lo a la implementació

- Diagrama de classes (visió *software*)
- Diagrama de seqüències (pels casos d'ús més importants)
- Diagrama de paquets (per projectes grans)
- Diagrama d'estats
- Diagrama de desplegament

Taula de continguts

- 1 Conceptes Orientació a Objectes
 - 2 *Unified Modeling Language (UML)*
 - Pinzellades UML
 - 3 Recordem alguns diagrames UML
 - Diagrames de classes
 - Una mica de codi
 - Casos d'ús
 - Diagrames de casos d'ús
 - Fitxes de casos d'ús
 - Diagrama d'activitats
 - Diagrames d'interacció
 - Diagrames de seqüència i comunicació

Diagrams de classes: notació UML

Elements del diagrama d'una classe

- CLASSES I INTERFÍCIES
- ATRIBUTS
- OPERACIONS
- TIPUS I ENUMERATIONS
- RELACIONS
- ALTRES RESTRICCIONS
- HERÈNCIA I INTERFÍCIES

Classes i Interfícies, atributs i operacions: notació UML

Sintaxi per una classe abstracta

Una classe és abstracta si té al menys un mètode abstracte.

```
NomDeLaClasseAbstracta
+_atributPublic : double
-_atributPrivat : string
-_atributPrivatMultiple : int
+_atributPublicOpcional : char
#_atributProtegit : string
-_atributPrivatEstatic : int = -1
+ / _atributPublicDerivat : string

+accioPublica(param1 : double, param2 : int)
-accioPrivada()
#accioProtegida()
+funcioPublica() : int
+accioPublicaEstatica()
+funcioPublicaAbstracta() : string
```

Classes i Interfícies, atributs i operacions: notació UML

Sintaxi per una classe abstracta pura

Una classe és abstracta pura si tots els seus mètodes són abstractes i no té atributs (això últim a vegades es relaxa.)

```
NomDeLaClasseAbstractaPura  
+funcioPublicaAbstracta1() : string  
+accioPublicaAbstracta2()  
+_____()
```

Classes i Interfícies, atributs i operacions: notació UML

Sintaxi per una Interfície

Una classe és abstracta pura sense atributs es pot pensar també com interfície (representen, però, coses diferents).

```
<<Interface>>
Nom de la interficie
+funcioPublicaAbstracta1() : string
+accioPublicaAbstracta2()
+___()
```

Powered by Visual Paradigm Community Edition

Tipus i enumerations: notació UML

Tipus i *Enumerations*

- UML defineix uns tipus bàsics (char, int, float, double, ...) i permet definir-ne d'altres.
- DataType serveix per definir valors de dades estructurades.
- Enumeration serveix per definir un tipus de dades a partir dels valors que pot prendre Mesos, Comarques...

Tipus i enumerations: notació UML

Data Type i enumeratio: definició i ús

<<Data Type>>

Data

```
dia : int
mes : NomsMes
any : int
nomDia : NomsDia
```

<<enumeration>>

NomsDia

```
dilluns
dimarts
dimecres
dijous
divendres
dissabte
diumenge
```

<<enumeration>>

NomsMes

```
gener
febrer
març
abril
maig
juny
juliol
agost
setembre
octubre
novembre
desembre
```

Pacient

```
-codi : string
-DNI : string
-nom : string
-cognoms : string
-dataNaixement : Data
+edat() : int
+nom() : string
```

Relacions: notació UML

Podem parlar de quatre tipus de relacions entre classes, de més generals a més concretes:

Dependència. $A \rightarrow B$ A depèn de B si canvis en la definició de B poden implicar modificar el codi d'A.

Associació. $A \rightarrow B$, A utilitza o està interrelacionat amb B

Agregació. $A \diamond \rightarrow B$ Una instància de A(tot) conté n instàncies de B(part).

Composició. $A \blacklozenge \rightarrow B$ Una instància d'A està formada per n instàncies de B i cada instància de B només té sentit com a part d'una instància d'A.

Fletxes Les fletxes $A \rightarrow B$, $A \leftarrow B$, $A \leftrightarrow B$ marquen la "navegabilitat" de la relació. $A - B$ és equivalent a $A \leftrightarrow B$.

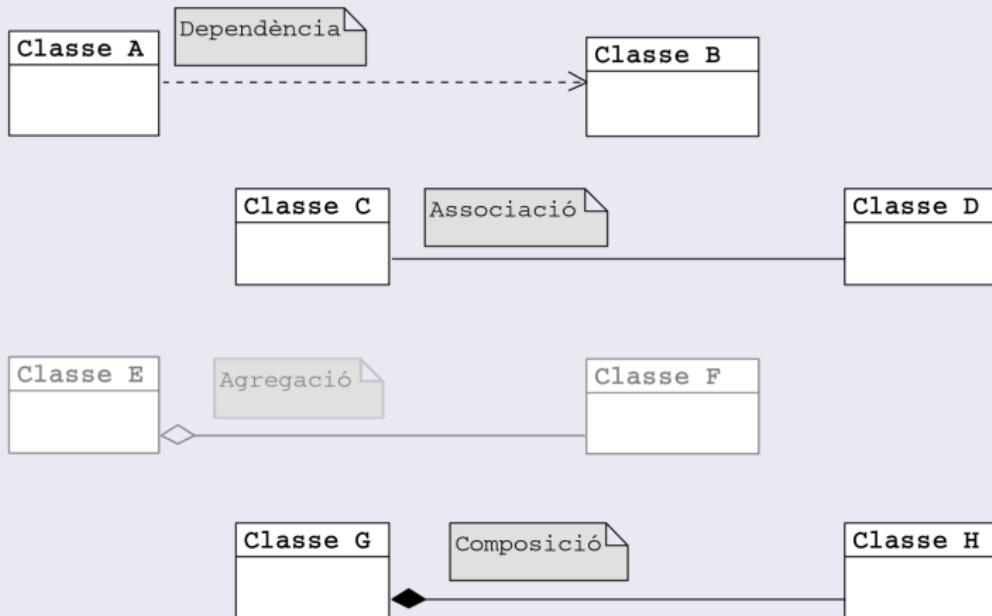
Relacions: notació UML

Ús dels diferents tipus de relacions

- La més usada és l'associació $A - B$, $A \leftarrow B$, $A \rightarrow B$.
- La dependència només es fa servir per destacar possibles efectes colaterals.
- L'agregació quasi mai es fa servir (és equivalent a una associació 1:n o n:n). Ve a ser un element de disseny *placebo*.
- La composició $A \blacklozenge \rightarrow B$ es pot fer servir quan sigui important poder indicar que les instàncies de B només tenen sentit si estan relacionades amb una de A .

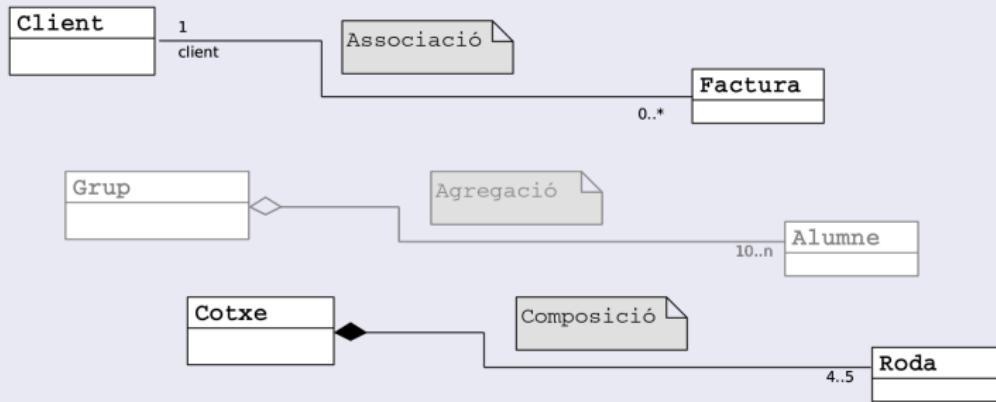
Relacions entre classes: notació UML

Sintaxi relacions entre classes



Relacions entre classes: notació UML

Sintaxi relacions amb multiplicitats entre classes



Powered By Visual Paradigm Community Edition

Relacions entre classes: notació UML

Sintaxi relacions amb multiplicitats i rols entre classes



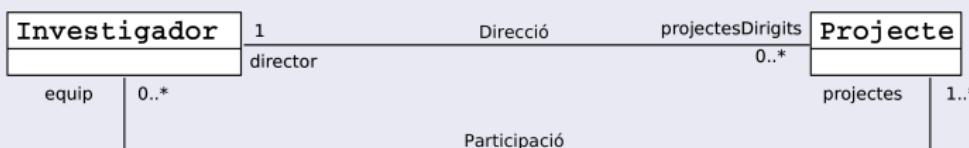
Photo by Dennis van de Water on Unsplash

Associació Direcció, projectes dirigits per un investigador

- Des de **Projecte**, l'investigador que el dirigeix s'anomena **director**
- Des d'**Investigador**, els projectes que té associats s'anomena **projectesDirigits**

Relacions entre classes: notació UML

Sintaxi relacions amb multiplicitats i rols entre classes



Modelos de software - UML - 2013-2014 - 10

Associació Participació, projectes on un investigador treballa

- Des de Projecte, els investigadors que hi treballen són equip
- Des d'Investigador, els projectes on treballa són projectes

Relacions entre classes: notació UML

Sintaxi relacions amb multiplicitats i rols entre classes

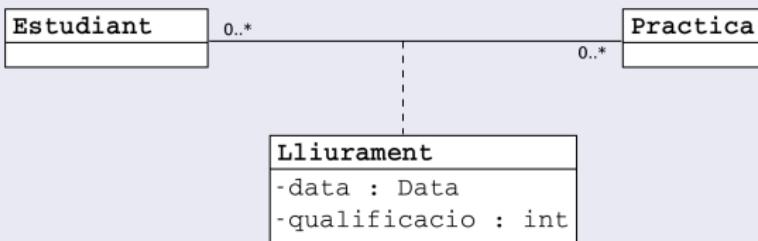


Els rols serviran com a nom d'atributs a la implementació

- La implementació de **Projecte** tindrà un atribut **director** de tipus **Investigador** i un atribut **equip** que serà un contenidor d'**Investigadors**.
- La implementació d'**Investigador** tindrà dos atributs contenidors de **Projectes**: **projectesDirigits** i **projectes**.

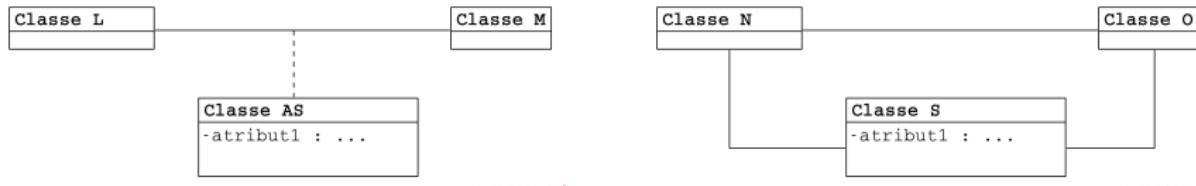
Relacions entre classes: notació UML

Classes associatives



Permeten incorporar atributs i comportaments a les associacions.

Relacions entre classes: notació UML



COMPTE! Classes associatives vs associacions

- Cada parella d'instàncies Classe L / Classe M **només pot tenir una** instància de la classe associativa Classe AS associada.
- Cada parella d'instàncies Classe N / classe O **pot tenir diverses** instàncies de la classes Classe S associades.

Relacions entre classes: notació UML

Navegabilitat de les associacions: bidireccionals

Powered by Visual Paradigm Community Edition

De Maquina a Operari i d'Operari a Maquina

Associació amb fletxa als dos costats indica navegabilitat en els dos sentits: cada classe contindrà una referència a la instància (o instàncies) a la que està associada:

- Maquina tindrà un atribut **responsable** de tipus Operari
- Operari tindrà un atribut **opera**, contenidor de Maquina.

Relacions entre classes: notació UML

Navegabilitat de les associacions: unidireccionals

Powered ByVisual Paradigm Community Edition

Jugador a Pilota, però no de Pilota a Jugador

Associació amb una fletxa indica navegabilitat només entre la classe origen i la classe destí (la classe origen tindrà una referència a la classe destí):

- Jugador tindrà un atribut pilota de tipus Pilota
- Pilota no té cap referència a cap Jugador

Relacions entre classes: notació UML

Navegabilitat de les associacions:



Powered by Visual Paradigm Community Edition

Sense cap fletxa pot representar dues coses:

- Que la navegabilitat no és important en aquesta associació.
- Que hi ha una navegabilitat bidireccional (notació per defecte). **Millor marcar explícitament la bidireccionalitat amb \longleftrightarrow**

Relacions entre classes: notació UML

COMPTE: o atribut o associació. **NO les dues coses**

Un error habitual és posar al diagrama de classes els atributs que es faran servir per representar les associacions quan s'implementi.
No s'hi han de posar.

És un error equivalent a, quan dissenyem una base de dades amb el model ER, posar com atributs de les entitats les claus foranes que farem servir per representar les interrelacions quan ho passem al model relacional.

Altres restriccions

Els elements del diagrama de classe, entre altres coses, recullen restriccions (multiplicitats, tipus, etc.) però no sempre es podran expressar totes.

Exemples d'altres restriccions

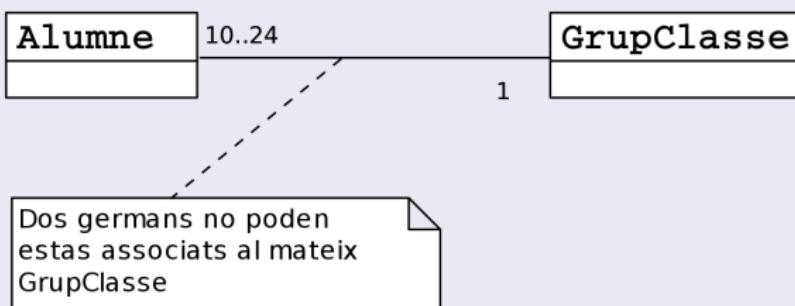
- L'atribut `horesDesdeUltimaAturada` d'una `Maquina` no pot ser més gran de 10000 (per exemple per forçar parades programades cada 10000 hores)
- Dos germans no poden anar a la mateixa classe (per exemple en un programa per assignar alumnes a grups de classe d'una escola que vol que els germans no vagin junts).
- ...

Aquestes restriccions es posen com a notes de text dins el diagrama (o com un text acompañant al diagrama)

Altres restriccions

Els elements del diagrama de classe, entre altres coses, recullen restriccions (multiplicitats, tipus, etc.) però no sempre es podran expressar totes.

Exemples d'altres restriccions

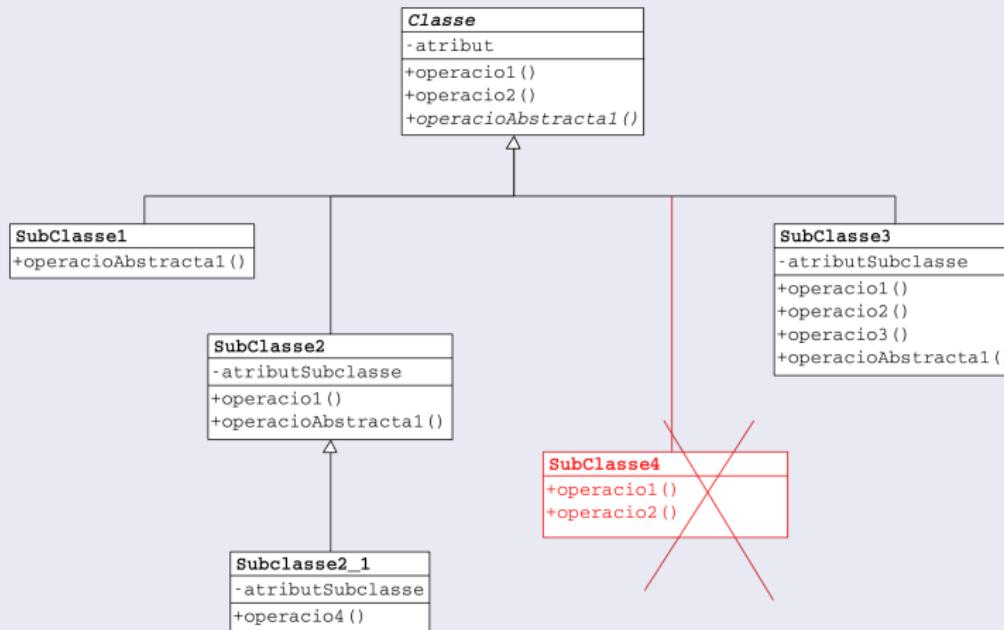
Powered ByVisual Paradigm Community Edition

Aquestes restriccions es posen com a notes de text dins el diagrama (o com un text acompañant al diagrama)

Herència i Interfície: notació UML

Herència

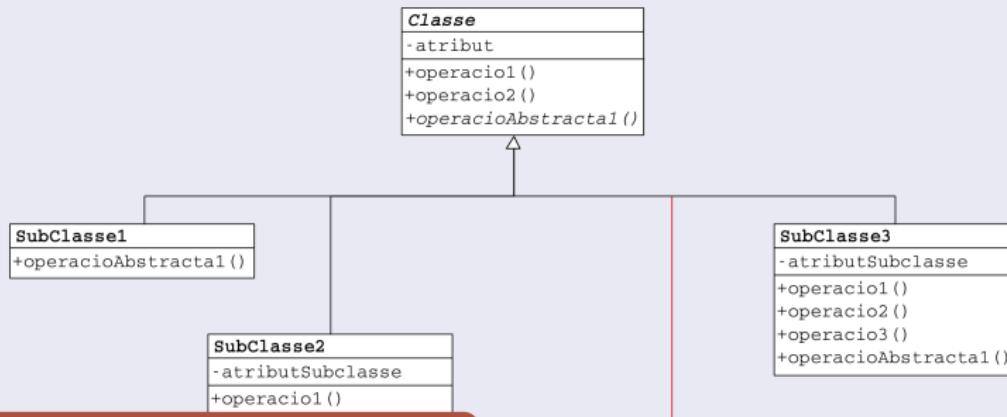
Línia contínua acabada amb fletxa sòlida que apunta cap a la superclasse:



Herència i Interfície: notació UML

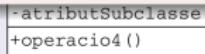
Herència

Línia contínua acabada amb fletxa sòlida que apunta cap a la superclasse:



Error

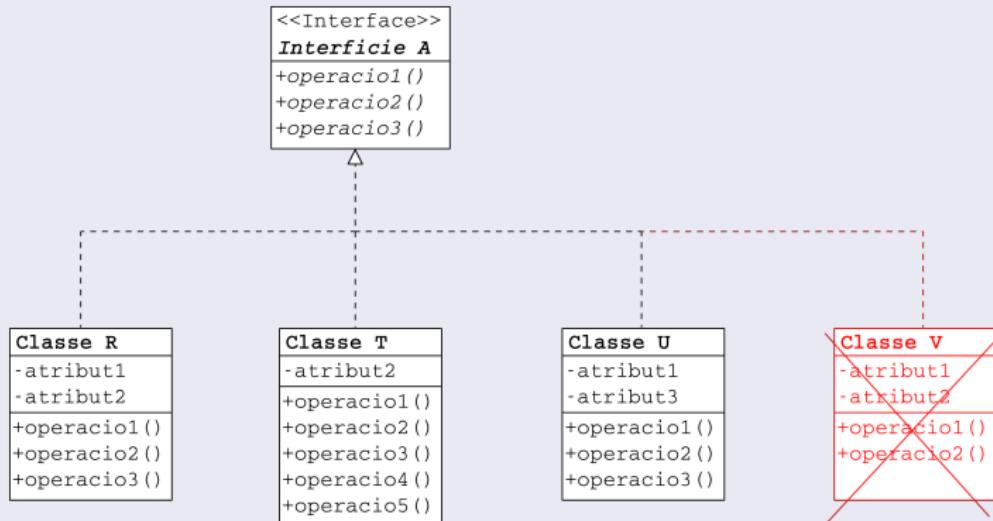
SubClasse4 és errònia perquè no ofereix l'operació *operacioAbstracta1()* que hereta de Classe: o bé caldria definir SubClasse4 com abstracte, o bé hauria d'ofrir l'operació *operacioAbstracta1()*.



Herència i Interfícies: notació UML

Interfícies

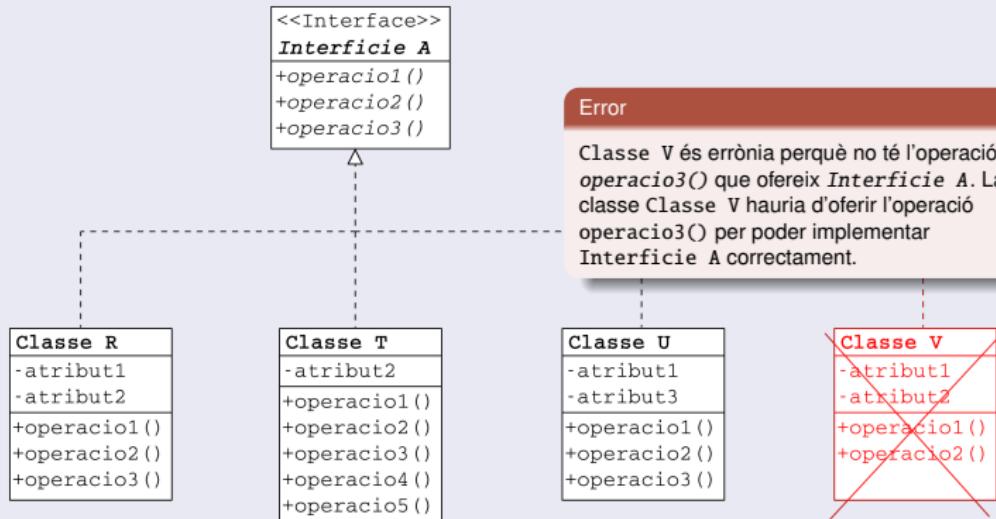
Línia discotínua acabada amb fletxa sòlida que apunta cap a la interfície:



Herència i Interfícies: notació UML

Interfícies

Línia discotínua acabada amb fletxa sòlida que apunta cap a la interfície:



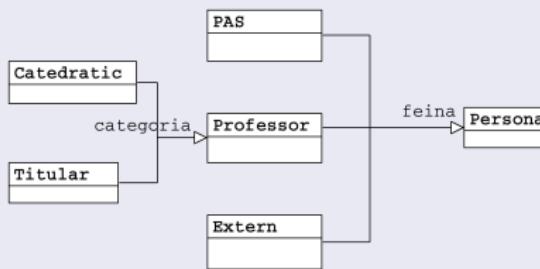
Error

Classe V és errònia perquè no té l'operació **operacio3()** que ofereix **Interfície A**. La classe Classe V hauria d'ofereir l'operació **operacio3()** per poder implementar **Interfície A** correctament.

Herència i Interfície: notació UML

Classificació simple

Un objecte és descrit per un tipus:

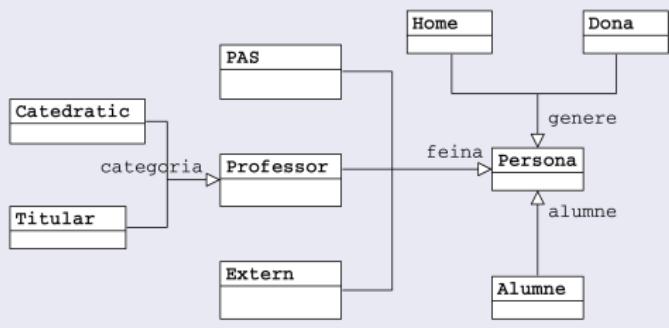


- Una Persona pot ser d'una de les 5 classes que hereten d'ella (i, per polimorfisme, de les superclasses). Per exemple:
 - Pot ser un Professor.
 - Pot ser un Catedratic (que serà, també, Professor per polimorfisme).
 - No pot ser Professor i PAS perquè són de la mateixa generalització.

Herència i Interfície: notació UML

Classificació múltiple

Una instància pot ser descrita per diversos tipus a l'hora:



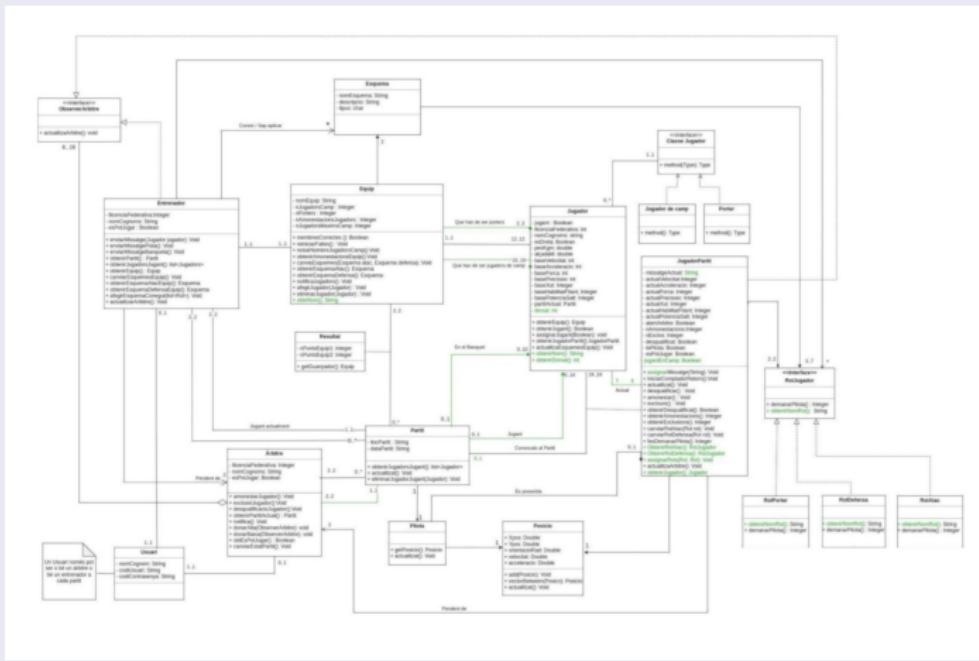
- Una Persona pot ser, a l'hora, d'una de les 5 classes de la generalització feina, d'una de les 2 de genere i alumne (i, per polimorfisme, de les superclasses). Per exemple
 - Pot ser un Professor, Home i Alumne
 - Pot ser un Catedratic (i Professor per polimorfisme) i Dona
 - No pot ser Professor i PAS perquè són de la mateixa generalització.

Herència i Interfícies: notació UML

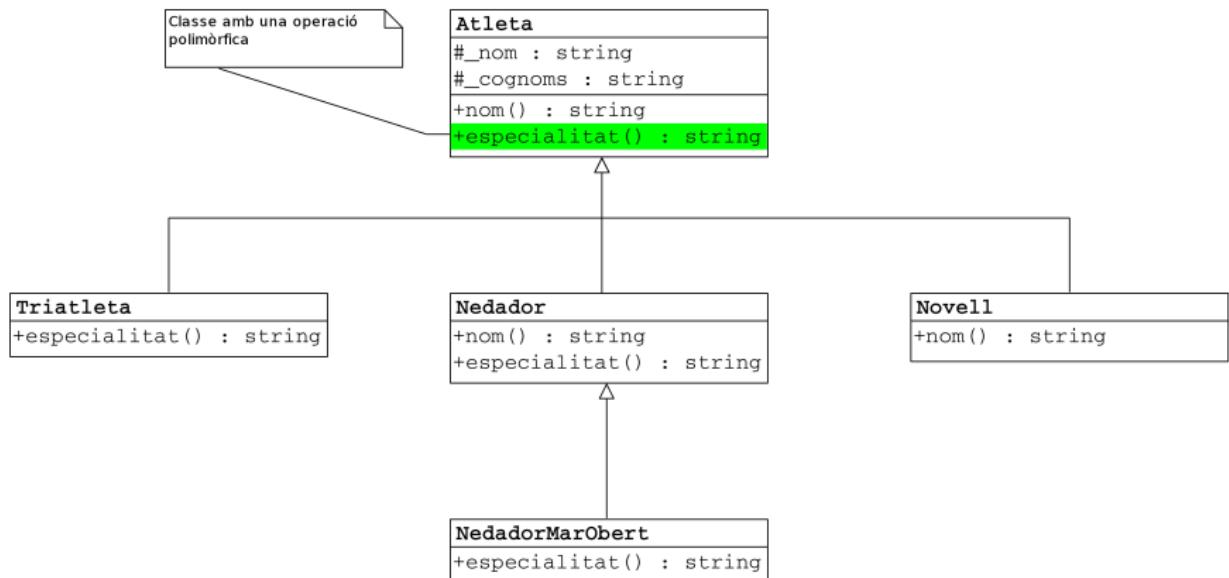
Combinarem tots els elements UML que hem vist per les classes
(Classes, Interfícies, Relacions entre classes, Herència. . .))

Herència i Interfícies: notació UML

exemple diagrama de classes amb diferents elements



Una mica de codi: herència, exemple1

Presentat per Ferran Vilà i Joan Compte Ballester

Una mica de codi: herència, exemple1, C++

Atleta.h (exemple a codiCxx/codiCxx1)

```
1 using namespace std;
2 #include<string>
3 #include <iostream>
4
5 class Atleta {
6     private:
7     protected:
8     string _nom, _cognoms;
9     public:
10    Atleta(){};
11    Atleta(const string &nom, const string &cognoms) : _nom(nom), _cognoms(cognoms) {}
12
13    string nom() const;
14    virtual string especialitat() const;
15    // string especialitat() const;
16
17
18    friend ostream &operator<<(ostream &os, const Atleta &atleta);
19};
```

Una mica de codi: herència, exemple1, C++

Atleta.cpp

```
1 #include "Atleta.h"
2
3 string Atleta::nom() const {
4     return _nom+"_"+_cognoms;
5 }
6
7 string Atleta::especialitat() const {
8     return "CAP";
9 }
10
11 ostream &operator<<(ostream &os, const Atleta &atleta) {
12     os<< atleta.nom() << ":" << atleta.especialitat();
13     return os;
14 }
```

Una mica de codi: herència, exemple1, C++

Triatleta.h, especialitza Atleta

```
1 #include <iostream>
2 #include "Atleta.h"
3
4 class Triatleta : public Atleta {
5     public:
6         Triatleta(const string &nom, const string &cognoms);
7         string especialitat() const override;
8         friend ostream &operator<<(ostream &os, const Triatleta &triatleta);
9     };
```

Una mica de codi: herència, exemple1, C++

Triatleta.cpp

```
1 #include "Triatleta.h"
2
3 Triatleta::Triatleta(const string &nom, const string &cognoms) {
4     _nom=nom;
5     _cognoms=cognoms;
6 }
7
8 string Triatleta::especialitat() const{
9     return "Triatleta";
10 }
11
12 ostream &operator<<(ostream &os, const Triatleta &triatleta) {
13     os<< triatleta.nom() << ":"_ << triatleta.especialitat();
14     return os;
15 }
```

Una mica de codi: herència, exemple1, C++

Nedador.h, especialitza Atleta

```
1 #include <string>
2 #include <iostream>
3 #include "Atleta.h"
4
5 class Nedador : public Atleta {
6     public:
7     Nedador(){}
8     Nedador(const string &nom, const string &cognoms);
9
10    string nom() const override;
11    string especialitat() const override;
12
13    friend ostream &operator<<(ostream &os, const Nedador &nedador);
14};
```

Una mica de codi: herència, exemple1, C++

Nedador.cpp

```
1 #include "Nedador.h"
2
3 Nedador::Nedador(const string &nom, const string &cognoms) {
4     _nom=nom;
5     _cognoms=cognoms;
6 }
7
8 string Nedador::nom() const {
9     return Atleta::_cognoms + ", " + Atleta::_nom;
10 }
11
12 string Nedador::especialitat() const{
13     return "Nedador";
14 }
15
16 ostream &operator<<(ostream &os, const Nedador &nedador) {
17     os<< nedador.nom() << ":" << nedador.especialitat();
18     return os;
19 }
```

Una mica de codi: herència, exemple1, C++

Novell.h, especialitza Atleta

```
1 #include <string>
2 #include <ostream>
3 #include "Atleta.h"
4
5 class Novell :public Atleta{
6     public:
7         Novell(const string &nom, const string &cognoms);
8
9         string nom() const override;
10
11     friend ostream &operator<<(ostream &os, const Novell &novell);
12 };
```

Una mica de codi: herència, exemple1, C++

Novell.cpp

```
1 #include "Novell.h"
2 Novell::Novell(const string &nom, const string &cognoms) {
3     _nom=nom;
4     _cognoms=cognoms;
5 }
6
7 string Novell::nom() const {
8     string nom=_nom+"_"+_cognoms;
9     for (auto &c : nom) c=toupper(c);
10    return nom;
11 }
12
13 ostream &operator<<(ostream &os, const Novell &novell) {
14     os<< novell.nom() << ":" << novell.especialitat();
15     return os;
16 }
```

Una mica de codi: herència, exemple1, C++

NedadorMarObert.h, especialitza Nedador

```
1 #include <string>
2 #include <ostream>
3 #include "Nedador.h"
4
5 class NedadorMarObert : public Nedador{
6     public:
7         NedadorMarObert(const string &nom, const string &cognoms);
8         string especialitat() const override;
9
10        friend ostream &operator<<(ostream &os, const NedadorMarObert &obert);
11    };
```

Una mica de codi: herència, exemple1, C++

NedadorMarObert.cpp

```
1 #include "NedadorMarObert.h"
2
3 NedadorMarObert::NedadorMarObert(const string &nom, const string &cognoms) {
4     _nom=nom;
5     _cognoms=cognoms;
6 }
7
8 string NedadorMarObert::especialitat() const{
9     return "Nedador_mar_obert";
10 }
11
12 ostream &operator<<(ostream &os, const NedadorMarObert &obert) {
13     os<< obert.nom() << ":" << obert.especialitat();
14     return os;
15 }
```

Una mica de codi: herència, exemple1, C++

main.cpp

```
1 #include <iostream>
2 #include <memory>
3 #include <list>
4 #include "Atleta.h"
5 #include "Triatleta.h"
6 #include "Nedador.h"
7 #include "NedadorMarObert.h"
8 #include "Novell.h"
9 using namespace std;
10
11 int main() {
12     list<shared_ptr<Atleta>> atletes;
13     shared_ptr<Triatleta> triatleta;
14     shared_ptr<Nedador> nedador = shared_ptr<Nedador>(new Nedador("Ramona", "Carpa"));
15     shared_ptr<NedadorMarObert> nedadorM0 = make_shared<NedadorMarObert>("Enric", "Anguila");
16     shared_ptr<Novell> novell = make_shared<Novell>("Sara", "Totjustmhiposo");
17     triatleta=make_shared<Triatleta>("Pere", "Hotocounamicatot");
18
19     atletes.push_back(triatleta); atletes.push_back(nedador);
20     atletes.push_back(nedadorM0); atletes.push_back(novell);
21     // posem dues sèries de cout per comprovar la diferència de posar
22     // o no posar virtual davant de l'acció especialitat() a Atleta.h
23     cout << "Atletes:" << endl;
24     for (auto const & a : atletes)
25         cout << "\t[" << a << "] " << *a << endl;
26     cout << endl << "Treballant amb subclasses" << endl;
27     cout << "\t[" << triatleta << "] " << *triatleta << endl;
28     cout << "\t[" << nedador << "] " << *nedador << endl;
29     cout << "\t[" << nedadorM0 << "] " << *nedadorM0 << endl;
30     cout << "\t[" << novell << "] " << *novell << endl;
31
32     return 0;
}
```

Una mica de codi: herència, exemple1, C++

Sortida del programa:

Atletes:

```
[0x558dd37c7ff0] Pere Hotocounamicatot: Triatleta
[0x558dd37c7eb0] Ramona Carpa: Nedador
[0x558dd37c7f30] Enric Anguila: Nedador mar obert
[0x558dd37c7f90] Sara Totjustmhiposo: CAP
```

Treballant amb subclasses

```
[0x558dd37c7ff0] Pere Hotocounamicatot: Triatleta
[0x558dd37c7eb0] Carpa, Ramona: Nedador
[0x558dd37c7f30] Enric Anguila: Nedador mar obert
[0x558dd37c7f90] SARA TOTJUSTMHIPOSO: CAP
```

Una mica de codi: herència, exemple1, C++

COMPTE: operacions virtuals i virtuals pures

En C++ quan una operació preveiem que serà reescrita per alguna subclasse cal definir-les com `virtual` a la superclasse.

Així assegurarem que el polimorfisme funciona correctament i que si hem creat un objecte d'una subclasse sempre farà servir la "seva" implementació de les operacions.

Veiem-ho sobre l'exemple anterior (classe Atleta):

```
// virtual string especialitat() const;  
string especialitat() const;
```

Una mica de codi: herència, exemple1, C++

Atleta.h fent especialitat no virtual

```
1  using namespace std;
2  #include<string>
3  #include <iostream>
4
5  class Atleta {
6      private:
7      protected:
8      string _nom, _cognoms;
9      public:
10     Atleta(){};
11     Atleta(const string &nom, const string &cognoms) : _nom(nom), _cognoms(cognoms) {}
12
13     string nom() const;
14
15
16     // virtual string especialitat() const;
17     string especialitat() const;
18     friend ostream &operator<<(ostream &os, const Atleta &atleta);
19 };
```

Una mica de codi: herència, exemple1, C++

Amb especialitat() **no** virtual:

Atletes:

```
[0x559e99570ff0] Pere Hotocounamicatot: CAP
[0x559e99570eb0] Ramona Carpa: CAP
[0x559e99570f30] Enric Anguila: CAP
[0x559e99570f90] Sara Totjustmhiposo: CAP
```

Treballant amb subclasses

```
[0x559e99570ff0] Pere Hotocounamicatot: Triatleta
[0x559e99570eb0] Carpa, Ramona: Nedador
[0x559e99570f30] Enric Anguila: Nedador mar obert
[0x559e99570f90] SARA TOTJUSTMHIPOS0: CAP
```

Amb especialitat() **virtual**:

Atletes:

```
[0x558dd37c7ff0] Pere Hotocounamicatot: Triatleta
[0x558dd37c7eb0] Ramona Carpa: Nedador
[0x558dd37c7f30] Enric Anguila: Nedador mar obert
[0x558dd37c7f90] Sara Totjustmhiposo: CAP
```

Treballant amb subclasses

```
[0x558dd37c7ff0] Pere Hotocounamicatot: Triatleta
[0x558dd37c7eb0] Carpa, Ramona: Nedador
[0x558dd37c7f30] Enric Anguila: Nedador mar obert
[0x558dd37c7f90] SARA TOTJUSTMHIPOS0: CAP
```

... i completem l'exemple fent **virtual** el nom() d'Atleta

Modifiquem Atleta.h per fer el nom() virtual i tornem a compilar i executar... i veurem que els noms surten igual als dos blocs.

Una mica de codi: herència, exemple1, C++

Comentari sobre l'*especificador override*

- Tant C++ com Java es pot explicitar que una operació sobreescriva una operació de la superclasse afegint `override` a la seva declaració.
- **IMPORTANT:** posar-ho no modifica el comportament de l'operació. Té, però, dues utilitats:
 - ➊ Fer el codi més comprensible per un humà.
 - ➋ Evitar errors tipogràfics o de tipus de paràmetres (si es posa `override`) el compilador dóna un error si la superclasse no té cap operació amb el mateix nom i signatura o bé si l'operació no era virtual.
- És recomanable posar-ho al codi quan reescrivim mètodes.

Una mica de codi: herència, exemple1, C++

Exemple a codiCxx/override

Credit.h

```
1 #include <string>
2 using namespace std;
3 class Credit {
4 protected:
5     string _descripcio;
6 public:
7     Credit(string const &desc);
8     virtual string garantiaNecessaria();
9 }
```

Credit.cpp

```
1 #include "Credit.h"
2 Credit::Credit(const string &descripcio) :
3     _descripcio(descripcio) {}
4 string Credit::garantiaNecessaria() {
5     return "Un_quart_de_ronyo";
6 }
```

Hipoteca.h

```
1 #include "Credit.h"
2 class Hipoteca :public Credit{
3 public:
4     Hipoteca(const string &desc);
5     string garantiaNecessaria();
6     //     string garantiaNecessaria() override;
7     //     string garantia_necessaria();
8     //     string garantia_necessaria() override;
9 }
```

Hipoteca.cpp

```
1 #include "Hipoteca.h"
2 Hipoteca::Hipoteca(const string &desc) :
3     Credit(desc) {}
4 string Hipoteca::garantiaNecessaria() {
5     return "La_propietat_immobiliaria_hipotecada
6     per la_vida_l'anima_dels_fills";
7 }
```

Una mica de codi: herència, exemple1, C++

main.cpp

```
1 #include <iostream>
2 #include <memory>
3 #include "Hipoteca.h"
4 int main() {
5     shared_ptr<Hipoteca> h=make_shared<Hipoteca>("Interes_variable");
6     cout << h->garantiaNecessaria()<<endl;
7 }
```

Sortida (l'esperada)

La propietat immobiliària hipotecada i l'ànima dels fills

Una mica de codi: herència, exemple1, C++

Credit.h

```
1 #include <string>
2 using namespace std;
3 class Credit {
4 protected:
5     string _descripcio;
6 public:
7     Credit(string const &desc);
8     virtual string garantiaNecessaria();
9 };
```

Credit.cpp

```
1 #include "Credit.h"
2 Credit::Credit(const string &descripcio) :
3     _descripcio(descripcio) {}
4 string Credit::garantiaNecessaria() {
5     return "Un_quart_de_ronyo";
6 }
```

Hipoteca.h

```
1 #include "Credit.h"
2 class Hipoteca :public Credit{
3 public:
4     Hipoteca(const string &desc);
5     //     string garantiaNecessaria();
6     string garantiaNecessaria() override;
7     //     string garantia_necessaria();
8     //     string garantia_necessaria() override;
9 };
```

Hipoteca.cpp

```
1 #include "Hipoteca.h"
2 Hipoteca::Hipoteca(const string &desc) :
3     Credit(desc) {}
4 string Hipoteca::garantiaNecessaria() {
5     return "La_propietat_immobiliaria_hipotecada
6         . . . . . i_l'anima_dels_fills";
7 }
```

Una mica de codi: herència, exemple1, C++

main.cpp

```
1 #include <iostream>
2 #include <memory>
3 #include "Hipoteca.h"
4 int main() {
5     shared_ptr<Hipoteca> h=make_shared<Hipoteca>("Interes_variable");
6     cout << h->garantiaNecessaria()<<endl;
7 }
```

Sortida (l'esperada)

La propietat immobiliària hipotecada i l'ànima dels fills

Una mica de codi: herència, exemple1, C++

Credit.h

```
1 #include <string>
2 using namespace std;
3 class Credit {
4 protected:
5     string _descripcio;
6 public:
7     Credit(string const &desc);
8     virtual string garantiaNecessaria();
9 };
```

Credit.cpp

```
1 #include "Credit.h"
2 Credit::Credit(const string &descripcio) :
3     _descripcio(descripcio) {}
4 string Credit::garantiaNecessaria() {
5     return "Un_quart_de_ronyo";
6 }
```

Hipoteca.h error en el nom

```
1 #include "Credit.h"
2 class Hipoteca :public Credit{
3 public:
4     Hipoteca(const string &desc);
5     //     string garantiaNecessaria();
6     //     string garantiaNecessaria() override;
7     string garantia_necessaria(); // Error en el nom
8     //     string garantia_necessaria() override;
9 };
```

Hipoteca.cpp

```
1 #include "Hipoteca.h"
2 Hipoteca::Hipoteca(const string &desc) :
3     Credit(desc) {}
4 string Hipoteca::garantia_necessaria() {
5     return "La_propietat_immobiliaria_hipotecada
6         per l'anima_dels_fills";
7 }
```

Una mica de codi: herència, exemple1, C++

main.cpp

```
1 #include <iostream>
2 #include <memory>
3 #include "Hipoteca.h"
4 int main() {
5     shared_ptr<Hipoteca> h=make_shared<Hipoteca>("Interes_variable");
6     cout << h->garantiaNecessaria()<<endl;
7 }
```

Sortida (no és l'esperada)

Un quart de ronyó

Una mica de codi: herència, exemple1, C++

Credit.h

```
1 #include <string>
2 using namespace std;
3 class Credit {
4 protected:
5     string _descripcio;
6 public:
7     Credit(string const &desc);
8     virtual string garantiaNecessaria();
9 };
```

Credit.cpp

```
1 #include "Credit.h"
2 Credit::Credit(const string &descripcio) :
3     _descripcio(descripcio) {}
4 string Credit::garantiaNecessaria() {
5     return "Un quart de ronyo";
6 }
```

Hipoteca.h amb override que provoca error

```
1 #include "Credit.h"
2 class Hipoteca :public Credit{
3 public:
4     Hipoteca(const string &desc);
5     //     string garantiaNecessaria();
6     //     string garantiaNecessaria() override;
7     //     string garantia_necessaria();
8     string garantia_necessaria() override;
9 };
```

Hipoteca.cpp

```
1 #include "Hipoteca.h"
2 Hipoteca::Hipoteca(const string &desc) :
3     Credit(desc) {}
4 string Hipoteca::garantia_necessaria() {
5     return "La propietat immobiliaria hipotecada
6     al l'anima dels fills";
7 }
```

Una mica de codi: herència, exemple1, C++

main.cpp

```
1 #include <iostream>
2 #include <memory>
3 #include "Hipoteca.h"
4 int main() {
5     shared_ptr<Hipoteca> h=make_shared<Hipoteca>("Interes_variable");
6     cout << h->garantiaNecessaria()<<endl;
7 }
```

ERROR DE COMPILACIÓ

Al posar `override` després de la definició d'un mètode que no existeix a la classe mare es genera un error de compilació:

```
$ g++ -o exemple *.cpp
In file included from Hipoteca.cpp:5:
Hipoteca.h:14:12: error: 'std::string Hipoteca::garantia_necessaria()' marked 'override',
but does not override
14 |     string garantia_necessaria() override;
|           ^~~~~~
```

Una mica de codi: herència, exemple1, Java

Atleta (exemple a codiJava/exemple1)

```
1 public class Atleta {  
2     protected String _nom;  
3     protected String _cognoms;  
4  
5     public Atleta(){}  
6     public Atleta(String nom, String cognoms){  
7         _nom=nom;  
8         _cognoms=cognoms;  
9     }  
10    public String nom() {  
11        return _nom+" "+_cognoms;  
12    }  
13    public String especialitat(){  
14        return "CAP";  
15    }  
16    public String toString() {  
17        return nom()+": "+especialitat();  
18    }  
19 }  
20 }
```

Una mica de codi: herència, exemple1, Java

Triatleta, especialitza Atleta

```
1 public class Triatleta extends Atleta{
2     public Triatleta() {
3     }
4
5     public Triatleta(String nom, String cognoms){
6         _nom=nom;
7         _cognoms=cognoms;
8     }
9
10    @Override
11    public String especialitat(){
12        return "Triatleta";
13    }
14
15    @Override
16    public String toString() {
17        return nom()+": "+especialitat();
18    }
19 }
```

Una mica de codi: herència, exemple1, Java

Nedador, especialitza Atleta

```
1  public class Nedador extends Atleta{
2      public Nedador() {
3      }
4
5      public Nedador(String nom, String cognoms){
6          _nom=nom;
7          _cognoms=cognoms;
8      }
9
10     @Override
11     public String nom(){
12         return _cognoms+","+_nom;
13     }
14
15     @Override
16     public String especialitat(){
17         return "Nedador";
18     }
19
20     @Override
21     public String toString() {
22         return nom()+":"+especialitat();
23     }
24 }
```

Una mica de codi: herència, exemple1, Java

Novell, especialitza Atleta

```
1  public class Novell extends Atleta{
2      public Novell() {
3      }
4
5      public Novell(String nom, String cognoms){
6          _nom=nom;
7          _cognoms=cognoms;
8      }
9
10     @Override
11     public String nom() {
12         String t=_nom+" "+_cognoms;
13         return t.toUpperCase();
14     }
15
16     @Override
17     public String toString() {
18         return nom()+":"+_especialitat();
19     }
20 }
```

Una mica de codi: herència, exemple1, Java

NedadorMarObert, especialitza Nedador

```
1 public class NedadorMarObert extends Nedador{
2     public NedadorMarObert() {
3     }
4
5     public NedadorMarObert(String nom, String cognoms){
6         _nom=nom;
7         _cognoms=cognoms;
8     }
9
10    @Override
11    public String especialitat(){
12        return "NedadorMarObert";
13    }
14
15    @Override
16    public String toString() {
17        return nom()+": "+especialitat();
18    }
19 }
```

Una mica de codi: herència, exemple1, Java

Main

```
1 import java.util.ArrayList;
2
3 public class Main {
4     public static void main(String[] args) {
5         ArrayList<Atleta> atletes = new ArrayList<Atleta>();
6         Triatleta triatleta;
7         Nedador nedador = new Nedador("Ramona", "Carpa");
8         NedadorMarObert nedadorMO = new NedadorMarObert("Enric", "Anguila");
9         Novell novell = new Novell("Sara", "Totjustmhiposo");
10        triatleta = new Triatleta("Pere", "Hotocounamicatot");
11
12        atletes.add(triatleta);
13        atletes.add(nedador);
14        atletes.add(nedadorMO);
15        atletes.add(novell);
16        System.out.println("Atletes");
17        for (int i = 0; i < atletes.size(); i++) {
18            System.out.println(atletes.get(i));
19        }
20    }
21 }
```

Una mica de codi: herència, exemple1, Java

Sortida del programa

Atletes:

Pere Hotocounamicatot: Triatleta

Carpa, Ramona: Nedador

Anguila, Enric: NedadorMarObert

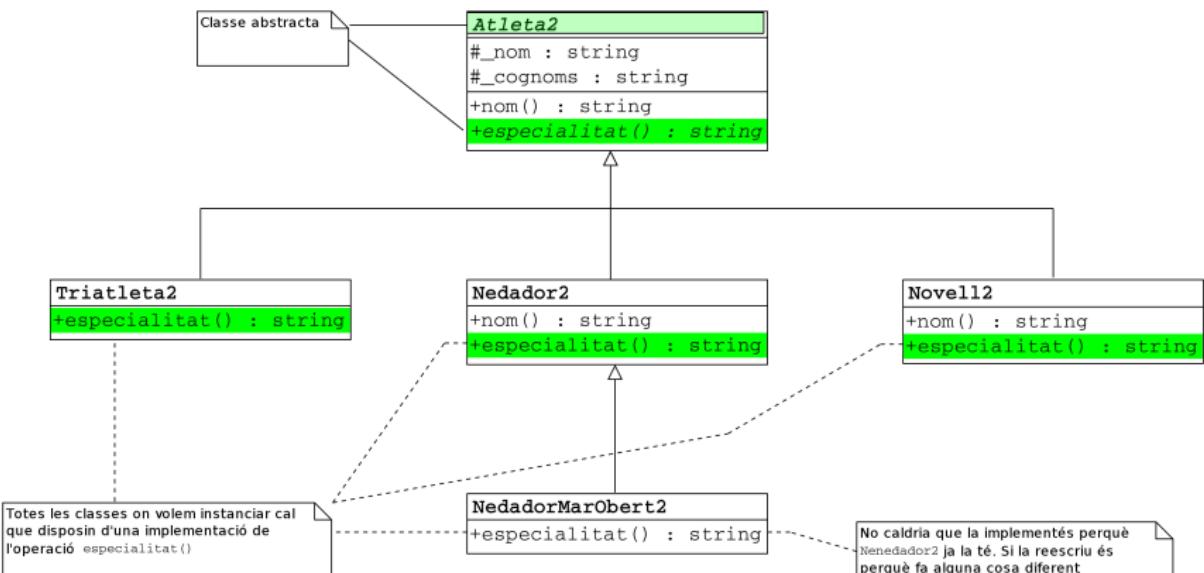
SARA TOTJUSTMHIPOSO: CAP

Una mica de codi: herència, exemple1, Java

El virtual i overrides del C++ en Java

- Java no necessita el `virtual` perquè tot és polimòrfic.
- La comprovació de sobreescritura es fa afegint `@Override` al principi de la definició dels mètodes (a diferència del C++ que es posava `override` al final). Té el mateix efecte i la mateixa utilitat que en C++.

Una mica de codi: herència, exemple2



Una mica de codi: herència, exemple2

- Suposem que la classe Atleta és una classe abstracta. En C++ això passa sempre que una classe tingui una operació virtual pura (`virtual x() = 0;`) i en Java cal indicar que la classe és abstracta explícitament `abstract class X`

Una mica de codi: herència, exemple2, C++

Atleta.h (exemple a codiCxx/codiCxxAbstracte)

```
1 using namespace std;
2 #include<string>
3 #include <iostream>
4
5 class Atleta {
6 private:
7 protected:
8     string _nom, _cognoms;
9 public:
10    Atleta(){};
11    Atleta(const string &nom, const string &cognoms);
12
13    virtual string nom() const;
14    virtual string especialitat() const = 0;
15    friend ostream &operator<<(ostream &os, const Atleta &atleta);
16};
```

Una mica de codi: herència, exemple2, C++

La classe Novell no reescrivia especialitat() i ara, com que és abstracta a Atleta, cal afegir-la.

Novell.h

```
1 #include <string>
2 #include <ostream>
3 #include "Atleta.h"
4
5 class Novell :public Atleta{
6 public:
7     Novell(const string &nomb, const string &cognoms);
8
9     string nom() const;
10    string especialitat() const override;
11    friend ostream &operator<<(ostream &os, const Novell &novell);
12};
```

Una mica de codi: herència, exemple2, C++

Novell.cpp

```
1 #include "Novell.h"
2 Novell::Novell(const string &nom, const string &cognoms) {
3     _nom=nom;
4     _cognoms=cognoms;
5 }
6
7 string Novell::nom() const {
8     string nom= _nom+" "+_cognoms;
9     for (auto &c :nom) c=toupper(c);
10    return nom;
11 }
12
13 string Novell::especialitat() const {return "CAP";}
14
15 ostream &operator<<(ostream &os, const Novell &novell) {
16     os<< novell.nom() << ":" << novell.especialitat();
17     return os;
18 }
```

Una mica de codi: herència, exemple2, C++

La classe NedadorMarObert reescriu especialitat() aprofitant el codi de la seva superclasse Nedador (per mostrar altres possibilitats...)

NedadorMarObert.cpp accedint al codi de la classe mare

```
1 #include "NedadorMarObert.h"
2
3 NedadorMarObert::NedadorMarObert(const string &nom, const string &cognoms) {
4     _nom=nom;
5     _cognoms=cognoms;
6 }
7
8 string NedadorMarObert::especialitat() const{
9     return Nedador::especialitat()+"...pero_de_mar_obert";
10 }
11
12 ostream &operator<<(ostream &os, const NedadorMarObert &obert) {
13     os<< obert.nom() << ":" << obert.especialitat();
14     return os;
15 }
```

Una mica de codi: herència, exemple2, C++

Sortida del programa

Atletes:

```
[0x55d39c8cbff0] Pere Hotocounamericatot: Triatleta
[0x55d39c8cbebe0] Carpa, Ramona: Nedador
[0x55d39c8cbf30] Anguila, Enric: Nedador... pero de mar obert
[0x55d39c8cbf90] SARA TOTJUSTMHIPOS0: CAP
```

Treballant amb subclasses

```
[0x55d39c8cbff0] Pere Hotocounamericatot: Triatleta
[0x55d39c8cbebe0] Carpa, Ramona: Nedador
[0x55d39c8cbf30] Anguila, Enric: Nedador... pero de mar obert
[0x55d39c8cbf90] SARA TOTJUSTMHIPOS0: CAP
```

Una mica de codi: herència, exemple2, Java

Atleta (exemple a codi Java/exempleAbstracte)

```
1 public abstract class Atleta {  
2     protected String _nom;  
3     protected String _cognoms;  
4  
5     public Atleta(){  
6     public Atleta(String nom, String cognoms){  
7         _nom=nom;  
8         _cognoms=cognoms;  
9     }  
10    public String nom() {  
11        return _nom+" "+_cognoms;  
12    }  
13    public abstract String especialitat();  
14    public String toString() {  
15        return nom()+": "+especialitat();  
16    }  
17 }
```

Una mica de codi: herència, exemple2, Java

La classe Novell no reescrivia especialitat() i ara, com que és abstracta a Atleta, cal afegir-la.

Novell

```
1 public class Novell extends Atleta{
2     public Novell() {
3     }
4
5     public Novell(String nom, String cognoms){
6         _nom=nom;
7         _cognoms=cognoms;
8     }
9
10    @Override
11    public String nom() {
12        String t=_nom+" "+_cognoms;
13        return t.toUpperCase();
14    }
15
16    @Override
17    public String especialitat() {return "Cap";}
18
19    @Override
20    public String toString() {
21        return nom()+": "+especialitat();
22    }
23 }
```

Una mica de codi: herència, exemple2, Java

La classe NedadorMarObert reescriu especialitat() aprofitant el codi de la seva superclasse Nedador (per mostrar altres possibilitats...)

NedadorMarObert accedint al codi de la classe mare

```
1 public class NedadorMarObert extends Nedador{
2     public NedadorMarObert() {}
3
4     public NedadorMarObert(String nom, String cognoms){
5         _nom=nom;
6         _cognoms=cognoms;
7     }
8
9     @Override
10    public String especialitat(){
11        return super.especialitat()+".....pero de mar obert";
12    }
13
14    @Override
15    public String toString() {return nom()+": "+especialitat();}
16 }
```

Una mica de codi: herència, exemple2, Java

Sortida del programa

Atletes

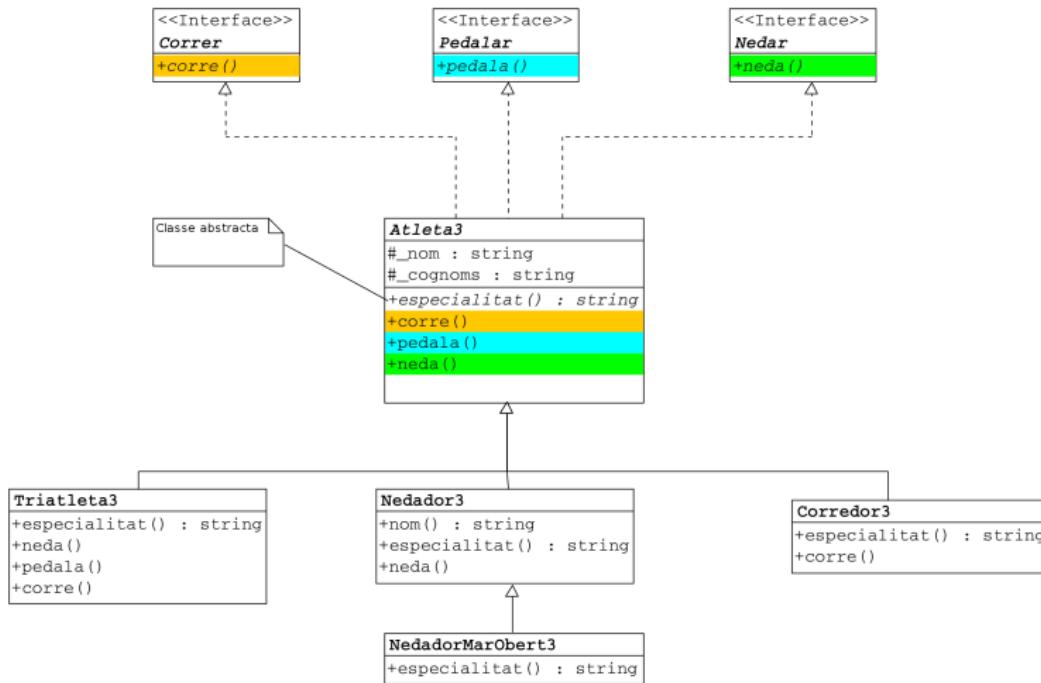
Pere Hotocounamicatot: Triatleta

Carpa, Ramona: Nedador

Anguila, Enric: Nedador pero a mar obert

SARA TOTJUSTMHIPOSÓ: Cap

Una mica de codi: Interfícies (i herència)



Una mica de codi: Interfícies, exemple C++

C++ i les interfícies

- El C++, a diferència del Java, no té el mecanisme d'interfícies i l'haurem de simular amb l'erència múltiple.

Tot seguit no reproduïm totes les classes com abans. Només posarem les que aporten alguna cosa en relació a la implementació de les interfícies.

Una mica de codi: Interfícies, exemple C++

Exemple a codiCxx/codiCxxInterficies

Correr.h (interfície)

```
1 class Correr {  
2     public:  
3         virtual void corre() = 0;  
4 };
```

Pedalar.h (interfície)

```
1 class Pedalar {  
2     public:  
3         virtual void pedala() = 0;  
4 };
```

Nedar.h (interfície)

```
1 class Nedar {  
2     public:  
3         virtual void neda() = 0;  
4 };
```

Una mica de codi: Interfícies, exemple C++

Atleta.h (herència múltiple)

```
1 #include<string>
2 #include<ostream>
3 #include "Nedar.h"
4 #include "Correr.h"
5 #include "Pedalar.h"
6 using namespace std;
7
8 class Atleta : public Nedar, Correr, Pedalar {
9     private:
10     protected:
11     string _nom, _cognoms;
12     public:
13     Atleta(){};
14     Atleta(const string &nom, const string &cognoms) : _nom(nom), _cognoms(cognoms) {}
15
16     virtual string nom() const;
17     virtual string especialitat() const = 0;
18     virtual void neda(); // implementem interficie Nedar
19     virtual void corre(); // implementem interficie Correr
20     virtual void pedala(); // implementem interficie Pedalar
21     friend ostream &operator<<(ostream &os, const Atleta &atleta);
22 };
```

Una mica de codi: Interfícies, exemple C++

Atleta.cpp

```
1 #include "Atleta.h"
2 #include <iostream>
3
4 string Atleta::nom() const {
5     return _nom+"_"+_cognoms;
6 }
7
8 void Atleta::neda(){
9     cout <<"Nedo, pero no es el meu fort"<<endl;
10 }
11 void Atleta::corre(){
12     cout << "Corro, pero no es el meu fort"<<endl;
13 }
14 void Atleta::pedala(){
15     cout << "Pedalo, pero no es el meu fort"<<endl;
16 }
17
18 ostream &operator<<(ostream &os, const Atleta &atleta) {
19     os<< atleta.nom() << ":" << atleta.especialitat();
20     return os;
21 }
```

Una mica de codi: Interfícies, exemple C++

Triatleta.h

```
1 using namespace std;
2
3 class Triatleta : public Atleta {
4     public:
5         Triatleta(const string &nom, const string &cognoms);
6         string especialitat() const override;
7
8         void neda() override; // sobreescrivim neda
9         void corre() override; // sobreescrivim corre
10        void pedala() override; // sobreescrivim pedala
11
12        friend ostream &operator<<(ostream &os, const Triatleta &triatleta);
13    };
```

Una mica de codi: Interfícies, exemple C++

Triatleta.cpp

```
1 ...
2 ...
3 void Triatleta::neda(){
4     cout << "estic_nedant_com_un_peix_globus" << endl;
5 }
6 ...
7 void Triatleta::corre(){
8     cout << "estic_corrent_prou_be" << endl;
9 }
10 ...
11 void Triatleta::pedala(){
12     cout << "hauria_de_posar_un_pinyo_mes_petit_però_no_puc..." << endl;
13 }
14 ...
15 ...
```

Una mica de codi: Interfícies, exemple C++

Programa principal

```
1 #include <iostream>
2 #include <memory>
3 #include <list>
4 #include "Atleta.h"
5 #include "Triatleta.h"
6 #include "Nedador.h"
7 #include "NedadorMarObert.h"
8 #include "Corredor.h"
9
10 using namespace std;
11
12 int main() {
13     list<shared_ptr<Atleta>> atletes;
14     list<shared_ptr<Nedador>> nedadors;
15     shared_ptr<Triatleta> triatleta;
16     shared_ptr<Nedador> nedador = shared_ptr<Nedador>(new Nedador("Ramona", "Carpa"));
17     shared_ptr<NedadorMarObert> nedadorM0 = make_shared<NedadorMarObert>("Enric", "Anguila");
18     shared_ptr<Corredor> corredor = make_shared<Corredor>("Sara", "Xinuxanu");
19     triatleta=make_shared<Triatleta>("Pere", "Hotocounamicatot");
20
21     atletes.push_back(triatleta); atletes.push_back(nedador);
22     atletes.push_back(nedadorM0); atletes.push_back(corredor);
23     nedadors.push_back(nedador); nedadors.push_back(nedadorM0); nedadors.push_back(triatleta);
24
25     cout << "Atletes:" << endl;
26     for (auto const &a : atletes) {
27         cout << *a << endl << "\t" << a->corre() << "\t" << a->neda() << "\t" << a->pedala();
28     }
29     cout << "Nedadors:" << endl;
30     for (auto const &n : nedadors) {
31         cout << "t"; n->neda();
32     }
33     return 0;
34 }
```

Una mica de codi: Interfícies, exemple C++

Sortida del programa

Atletes:

Pere Hotocounamicatot: Triatleta
estic corrent prou bé
estic nedant com un peix globus
hauria de posar un pinyó més petit però no puc...

Carpa, Ramona: Nedador

Corro, però no és el meu fort
estic nedant com un dofí
Pedalo, però no és el meu fort

Anguila, Enric: Nedador... pero de mar obert

Corro, però no és el meu fort
estic nedant com un dofí
Pedalo, però no és el meu fort

Sara Xinuxanu: Corredor

Corro com una llebre
Nedo, però no és el meu fort
Pedalo, però no és el meu fort

Nedadors

estic nedant com un dofí
estic nedant com un dofí
estic nedant com un peix globus

Una mica de codi: Interfícies, exemple Java

(exemple a codiJava/exempleInterficies)

Correr (interfície)

```
1 public interface Correr {  
2     public void corre();  
3 }
```

Pedalar (interfície)

```
1 public interface Pedalar {  
2     public void pedala();  
3 }
```

Nedar (interfície)

```
1 public interface Nedar {  
2     public void neda();  
3 }
```

Una mica de codi: Interfícies, exemple Java

Atleta (implementa interfícies)

```
1  public abstract class Atleta implements Nedar, Correr, Pedalar {
2      protected String _nom;
3      protected String _cognoms;
4
5      public Atleta(){}
6      public Atleta(String nom, String cognoms){
7          _nom=nom;
8          _cognoms=cognoms;
9      }
10     public String nom() {
11         return _nom+" "+_cognoms;
12     }
13     public abstract String especialitat();
14
15
16     @Override
17     public void neda(){
18         System.out.println("Nedo, pero no es el meu fort");
19     }
20
21     @Override
22     public void corre(){
23         System.out.println("Corro, pero no es el meu fort");
24     }
25
26     @Override
27     public void pedala(){
28         System.out.println("Pedalo, pero no es el meu fort");
29     }
30
31     @Override
32     public String toString() {
33         return nom()+":"+especialitat();
34     }
35 }
```

Una mica de codi: Interfícies, exemple Java

Triatleta (herència Atleta, sobreescriv nedar, correr i pedalar)

```
1 public class Triatleta extends Atleta{
2     public Triatleta() {
3     }
4
5     public Triatleta(String nom, String cognoms){
6         _nom=nom;
7         _cognoms=cognoms;
8     }
9
10    @Override
11    public String especialitat(){
12        return "Triatleta";
13    }
14
15    @Override
16    public void neda(){
17        System.out.println("estic_nedant_com_un_peix_globus");
18    }
19
20    @Override
21    public void corre(){
22        System.out.println("estic_corrent_prou_be");
23    }
24
25    @Override
26    public void pedala(){
27        System.out.println("hauria_de_posar_un_pinyo_mes_petit_per_no_puc...");
28    }
29
30
31    @Override
32    public String toString() {
33        return nom()+":"+especialitat();
34    }
35 }
```

Una mica de codi: Interfícies, exemple Java

Programa principal

```
1 import java.util.ArrayList;
2
3 public class Main {
4     public static void main(String[] args) {
5         ArrayList<Atleta> atletes = new ArrayList<Atleta>();
6         ArrayList<Nedar> nedadors = new ArrayList<Nedar>();
7         Triatleta triatleta;
8         Nedador nedador = new Nedador("Ramona", "Carpa");
9         NedadorMarObert nedadorMO = new NedadorMarObert("Enric", "Anguila");
10        Corredor corredor = new Corredor("Sara", "XinuXanu");
11        triatleta = new Triatleta("Pere", "Hotocounamicatot");
12
13        atletes.add(triatleta); atletes.add(nedador);
14        atletes.add(nedadorMO); atletes.add(corredor);
15        nedadors.add(nedador); nedadors.add(nedadorMO);
16        nedadors.add(triatleta);
17        System.out.println("Atletes");
18        for (int i = 0; i < atletes.size(); i++) {
19            Atleta a=atletes.get(i);
20            System.out.println(a);
21            System.out.print("\t");a.corre();
22            System.out.print("\t");a.neda();
23            System.out.print("\t");a.pedala();
24        }
25        System.out.println("Nedadors");
26        for (int i = 0; i < nedadors.size(); i++) {
27            System.out.print("\t");
28            nedadors.get(i).nedada();
29        }
30    }
31 }
```

Una mica de codi: Interfícies, exemple Java

Sortida del programa

Atletes

Pere Hotocounamicatot: Triatleta
estic corrent prou bé
estic nedant com un peix globus
hauria de posar un pinyó més petit però no puc...

Carpa, Ramona: Nedador

Corro, però no és el meu fort
estic nedant com un dofí
Pedalo, però no és el meu fort

Anguila, Enric: Nedador... pero de mar obert

Corro, però no és el meu fort
estic nedant com un dofí
Pedalo, però no és el meu fort

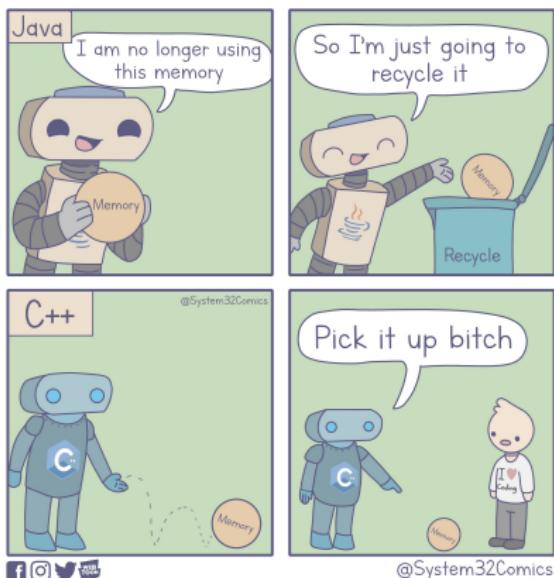
SARA XINUXANU: Cap

Corro com una llebre
Nedo, però no és el meu fort
Pedalo, però no és el meu fort

Nedadors

estic nedant com un dofí
estic nedant com un dofí
estic nedant com un peix globus

Gestió memòria JAVA i C++



<https://twitter.com/System32comicsA/status/1308134334595911681>

Casos d'ús

Descriuen les funcionalitats del sistema des del punt de vista dels actors que hi interaccionen. **La descripció és sobre operacions que ofereix el sistema i no sobre com ho fa.**

Pel disseny dels casos d'ús disposem de les següents eines UML:

- Diagrames de casos d'ús
- Fitxes de cados d'ús
- Diagrames d'activitat

Diagrama de casos d'ús

Elements UML: diagrama casos d'ús

- Elements del diagrama:
 - Actors (personetes) i Casos d'ús (el·lipses)
 - Associació actors/casos d'ús (línies solides sense fletxa)
 - Subcasos d'ús "reutilitzables" (línia discontinua amb etiqueta “<<include>>” i fletxa apuntant al cas inclòs)
 - Si el cas d'ús "A" inclou el cas d'ús "B" quan s'executa el cas d'ús "A" sempre s'executarà el "B".
 - Subcasos d'ús alternatius (línia discontinua amb etiqueta “<<extend>>” i fletxa apuntat cap al cas d'ús extès))
 - Si el cas d'ús "B" exten el cas d'ús "A" quan s'executa el cas d'ús "A" pot ser que s'executi també el "B" (especialitza "A").
- Distingim casos d'ús de context (genèrics) i casos d'ús detallats
- En els diagrames de casos d'ús no hi ha algorísmica: només es visualitzen les funcionalitats disponibles per cada actor.

Diagrama de casos d'ús

Elements UML: diagrama casos d'ús

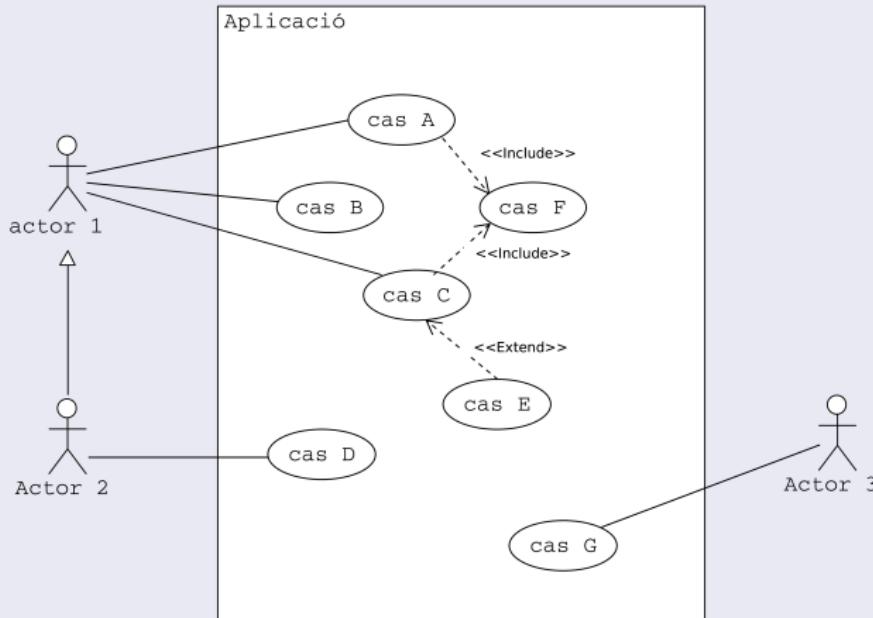


Diagrama de casos d'ús

Elements UML: diagrama casos d'ús

- Un actor 1 disposa dels casos d'ús cas A, cas B i cas C.
- Un actor 2 disposa de tots els casos d'ús de l'actor 1 més el cas d'ús cas D (actor 2 hereta d'actor 1).
- Un actor 3 disposa del cas d'ús cas G
- Quan un actor 1 o un actor 2 executa el cas d'ús cas A també s'executa, com a part d'aquest, el cas F.
- Quan un actor 1 o un actor 2 executa el cas d'ús cas C també s'executa, com a part d'aquest, el cas F.
- Quan un actor 1 o un actor 2 executa el cas d'ús cas C pot executar-se, com a part d'aquest, el cas E.

Fitxes de casos d'ús

Fitxa casos d'ús (camps)

Enginyeria del Software I

A continuació donem un exemple de fitxa d'especificació d'un cas d'ús:

CAS D'US:	nom del cas d'ús		
Versió	Num. Versió	Data	dd/mm/aa
Autors	Autors del document		
Descripció	Descripció informal dels objectius del cas d'ús		
Actors	Actors que intervenen		
Precondició	Condicions que han de complir-se perquè es pugui realitzar el cas d'ús		
Flux principal	Flux principal d'events del cas d'ús		
Subfluxos	Diferents alternatives dins del flux principal		
Fluxos alternatius	Variacions en els fluxos principals o casos d'excepció		
Postcondició	Postcondició del cas d'ús		
Requeriments no funcionals	Llista de restriccions relacionades amb aquest requeriment funcional		
Prioritat	{urgent, normal, no prioritari}		
Comentaris	Comentaris addicionals		

L'especificació dels casos d'ús es va actualitza durant tot el procés de desenvolupament del software.

2012-13

Fitxes de casos d'ús

Fitxa casos d'ús (visió usuari)

Enginyeria del Software I

Exemple de la fitxa del cas d'ús **Gestionar Préstec Exemplar** referit a la gestió de la biblioteca universitària:

CAS D'ÚS:	Gestionar Préstec Exemplar
Versió	Visió usuari
Descripció	Un usuari vol treure un exemplar d'un llibre en préstec
Actors	Bibliotecari, CapBiblioteca
Precondició	Usuari i exemplar donats d'alta al sistema
Flux principal	<ol style="list-style-type: none">Identificar usuariComprovar usuari no sancionatComprovar usuari no té en préstec el màxim nombre d'exemplars autoritzatsIdentificar exemplarComprovar exemplar en préstecComprovar llibre no reservat altra usuariSi llibre reservat per propi usuari cancel·lar reservaFer préstec
Flux alternatiu	<ol style="list-style-type: none">Si usuari sancionat o usuari té en préstec màxim exemplars autoritzats o exemplar no en préstec o llibre reservat altre usuari llavors refusar préstec.
Postcondició	Si tot OK préstec fet altrament préstec refusat

L'exemple de fitxa de cas d'ús està fet des del **punt de vista de l'usuari**.

2012-13

Fitxes de casos d'ús

Fitxa casos d'ús (visió informàtic)

Enginyeria del Software I	
CAS D'ÚS:	Gestionar Préstec Exemplar
Versió	Visió anàlisi
Descripció	Un usuari vol treure un exemplar d'un llibre en préstec
Actors	Bibliotecari, CapBiblioteca
Precondició	Escenari principal: <ul style="list-style-type: none">• usuari i exemplar donats d'alta al sistema;• usuari no sancionat i no té en préstec el màxim nombre de llibres autoritzats;• exemplar en préstec i llibre no reservat altre usuari.
Flux principal	<ol style="list-style-type: none">1. Buscar usuari2. Comprovar usuari no sancionat3. Comprovar usuari no té en préstec el màxim nombre de llibres autoritzats3. Buscar exemplar4. Comprovar exemplar en préstec5. Comprovar llibre no reservat altra usuari6. Si llibre reservat per propi usuari<ul style="list-style-type: none">6.1 Eliminar reserva de la llista de reserves del usuari6.2 Eliminar reserva de la llista de reserves del llibre7. Crear préstec amb usuari i exemplar8. Afegir préstec a llista préstecs usuari9. Afegir préstec a llista préstecs exemplar
Postcondició	Préstec creat i llistes de reserves i préstecs actualitzades

L'exemple de fitxa de cas d'ús està fet des del **punt de vista informàtic**.

2012-13

Diagrama d'activitats

Elements UML: diagrama activitats

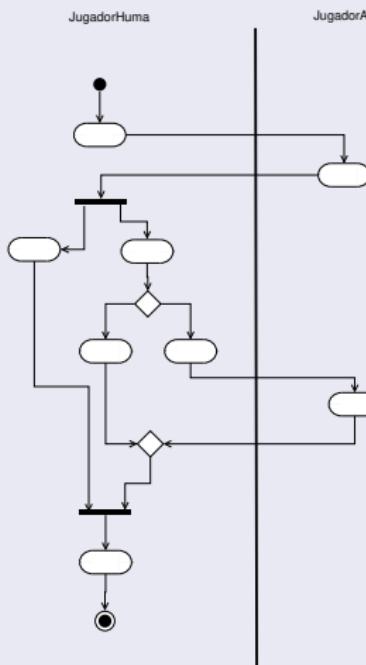


Diagrama d'activitats

Elements UML: diagrama activitats

- Elements del diagrama:
 - Activitats (rectangles arrodonits)
 - Transicions (fletxes entre activitats)
 - *Fork* (línia gruixuda) Una transició es descomposa en n de paral·leles
 - *Join* (línia gruixuda) n transicions hi arriben i en surt una (s'activa quan totes les que hi arriben es compleixen)
 - Decisió (rombe): n'hi arriba una i en surten n, cada una associada a una condició
 - Merge (rombe) n transicions arriben i en surt una (s'activa quan qualsevol de les que hi arriben es compleix)
 - Carrers: organitzen les activitats (un carrer per actor)
- Útils per descriure la part algorítmica de casos d'ús.

Diagrams d'interacció

Els diagrames d'interacció descriuen com col·laboren els objectes que participen en un cas d'ús. Aquesta col·laboració es fa via enviament de missatges entre els objectes

- Ajuden a decidir quines operacions ha de tenir cada classe.
- Disposem de dos diagrames:
 - Diagrames de seqüència (èmfasi en la temporalització)
 - Diagrames de comunicació (èmfasi en els rols dels objectes)

Diagrams de seqüència i comunicació

Diagrama de seqüència

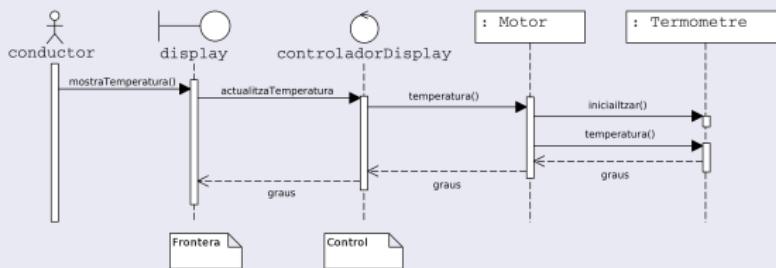
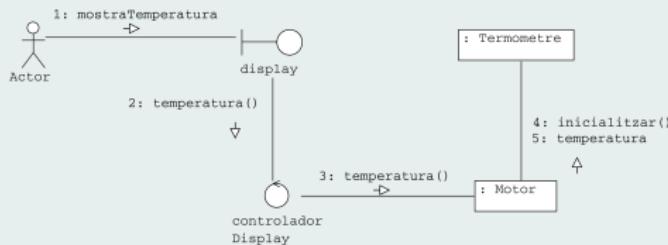


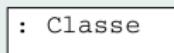
Diagrama de comunicació



Diagrams de seqüència i comunicació

Diverses possibilitats de Línies de Vida

instància sense nom



instància amb nom

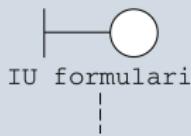


n-element conjunt



estereotips “especials” (per *robustness analysis*)

objecte frontera



objecte control



controlFormulari

objecte entitat



Diagrams de seqüència i comunicació

Estructures algorítmiques en diagrames de seqüència

Podem indicar estructures algorítmiques mitjançant blocs:

loop Bucle

opt Condicional (if then)

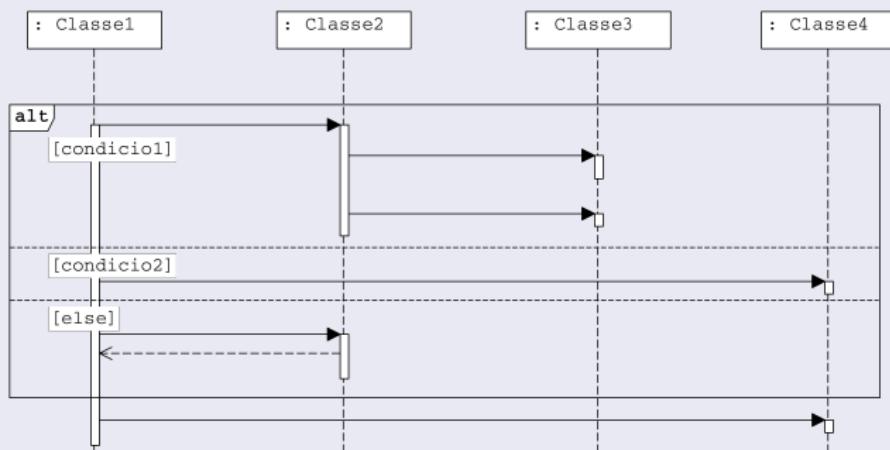
alt Condicional excloent (if then else)

...

Si cal, repasseu el material d'ESI per refrescar la notació

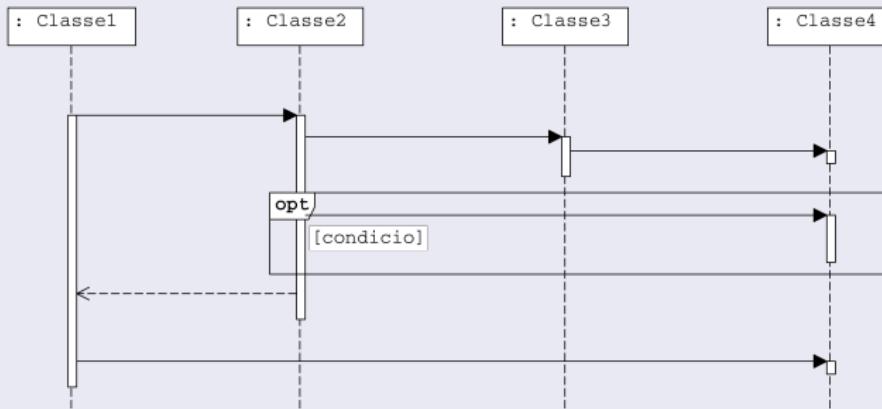
Diagrams de seqüència i comunicació

bloc alt



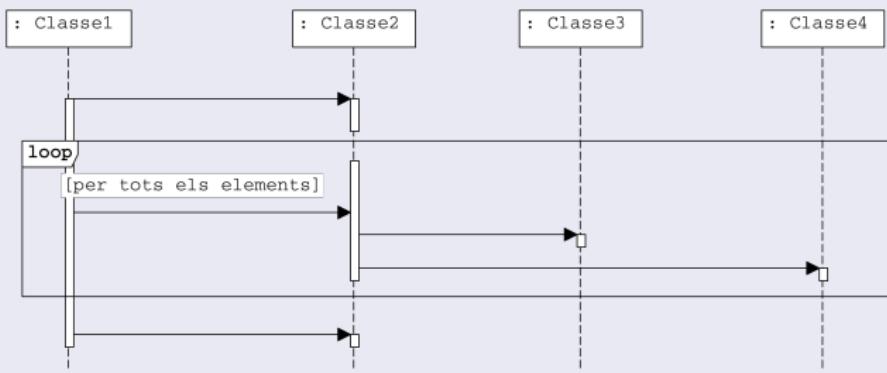
Diagrams de seqüència i comunicació

bloc opt



Diagrams de seqüència i comunicació

bloc loop



Diagrams de seqüència i comunicació

Diagrama de seqüència

- Es mostra clarament la seqüència temporal
- Ofereixen una notació més rica.
- Consumex molt d'espai horitzontal.

Diagrama de comunicació

- Diagrames més compactes
- És més complicat veure la seqüència temporal
- Menys opcions de notació

