

# Ingeniería de ontologías

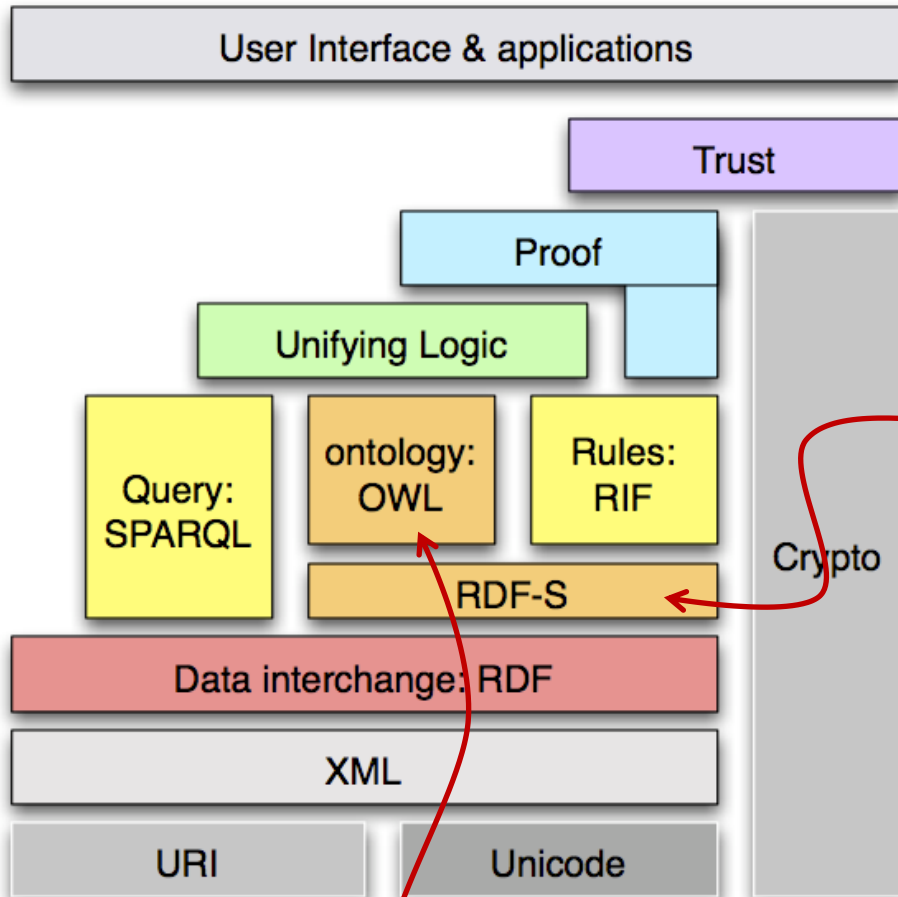
## Ontologías – OWL

Octubre 2022

# Agenda

- **Revisión ontologías - lógica descriptiva**
- **Lenguaje OWL**
- **OWL – Ejercicios Protégé**

# Arquitectura de la Web Semántica



Para modelar un dominio de interés o una aplicación:

Se necesita un modelo que permita **clasificar los recursos** descritos en RDF.



**Ontologías**: mayor expresividad

“Livianas”: **RDF-Schema**

(jerarquías de clases de recursos, jerarquías de propiedades, dominio y rango)

“Pesadas”: **OWL**

(imponer restricciones, chequeos de consistencia, inferencias) → **Lógica**

# ONTOLOGIA = Una especificación formal y explícita de una conceptualización compartida [Studer98]

- **Conceptualización:** modelo abstracto de un objeto de estudio
- **Formal:** representable en un lenguaje interpretable por agentes automáticos
- **Explícita:** conceptos, relaciones y axiomas que restringen la interpretación de los conceptos son explícitamente declarados
- **Compartida:** se representa conocimiento consensuado

# Bloques de construcción de una ontología

- **Instancias o individuos:** constantes

Elementos, objetos atómicos del dominio

*María, Uruguay*

- **Conceptos ó clases:** predicados unarios

Conjuntos de elementos del dominio

*Persona, Estudiante, País*

- **Relaciones ó roles:** predicados binarios

Conjuntos de pares de elementos del dominio

Ej.:  $vive \subseteq Persona \times Pais$

- **Axiomas:** sentencias que son siempre verdaderas

Afirmaciones sobre individuos, conceptos y roles

*Estudiante es una subclase de Persona*

***Estudiante  $\sqsubseteq$  Persona***

*María es una instancia de Persona*

***Persona(María)***

*María vive en Uruguay*

***vive(María, Uruguay)***

# Lógica Descriptiva - Sintaxis

Partimos de un conjunto de **nombres de conceptos atómicos**  $A, B, \dots$  y **nombres de roles atómicos**  $R, S, \dots$ , **nombres de individuos**  $a, b, \dots$

Cada lógica permite construir conceptos y axiomas con diferente expresividad:

$\mathcal{ALC}$ :  $\top, \perp, \sqcap, \sqcup, \exists, \forall, \neg$

$\mathcal{S}$ :  $\mathcal{ALC}$  + **roles transitivos**  $\text{Trans}(R)$

A  $\mathcal{S}$  se agregan constructores que se representan por diferentes letras:

$\mathcal{H}$ : inclusión de roles  $\mathcal{O}$ : nominales  $\{a\}$   $\mathcal{I}$ : roles inversos

$\mathcal{N}$ : restricciones numéricas  $\mathcal{Q}$ : restricciones numéricas calificadas

$\mathcal{R}$ :  $\text{Dis}(R, S)$  roles disjuntos  $\text{Irr}(R)$  roles irreflexivos

Aserciones de negación de roles:  $(\text{John}, \text{Mary}) : \neg\text{likes}$ ,

Axiomas de inclusión de roles complejos:  $R \circ S \sqsubseteq Q$ , universal role  $U$ ,  $\exists R.\text{Self}$

Descripción de conceptos en lógica  $\mathcal{ALCQ}$ :

$C, D ::= \perp \mid \top \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C \mid \geq nR.C$

Concepto vacío      Todo el Universo

# Lógica Descriptiva - Sintaxis

Conceptos en lógica *ALCQ*:

$C, D := \perp | T | A | \neg C | C \sqcap D | C \sqcup D | \forall R.C | \exists R.C | \geq nR.C | \leq nR.C$

Conceptos atómicos: *Persona, Madre, Padre, Mujer*      Rol atómico: *tieneHijo*

$\neg$ *Persona*: conjunto de todos los elementos que **no** satisfacen el predicado *Persona* (no pertenecen a ese conjunto)

*Persona*  $\sqcap$  *Mujer*: conjunto de todos los elementos que satisfacen el predicado *Persona* y satisfacen el predicado *Mujer* (pertenecen a ambos conjuntos)

*Padre*  $\sqcup$  *Madre*: conjunto de todos los elementos que satisfacen el predicado *Padre* (pertenecen a ese conjunto) ó satisfacen el predicado *Madre*

$\exists$ *tieneHijo.Persona*: conjunto de todos los elementos que **están vinculados** a algún elemento del concepto *Persona* a través del rol *tieneHijo*

$\forall$ *tieneHijo.Persona*: conjunto de todos los elementos que, **si están vinculados** a algún elemento a través del rol *tieneHijo*, este elemento **debe pertenecer al concepto *Persona***. Si NO está vinculado a ningún elemento a través de *tieneHijo*, también satisface la condición.

$\geq 2$ *tieneHijo.Persona*: conjunto de todos los elementos que **están vinculados a por lo menos 2 elementos** del concepto *Persona* a través del rol *tieneHijo*

# Lógica Descriptiva – Sintaxis - Rbox

**ALC**:  $\top, \perp, \sqcap, \sqcup, \exists, \forall, \neg$

**S**: **ALC** + roles transitivos  $\text{Trans}(R)$

A **S** se agregan constructores que se representan por diferentes letras:

**H**: inclusión de roles **O**: nominales  $\{a\}$  **I**: roles inversos

**N**: restricciones numéricas **Q**: restricciones numéricas calificadas

**R**:  $\text{Dis}(R, S)$  roles disjuntos  $\text{Irr}(R)$  roles irreflexivos

Aserciones de negación de roles:  $(John, Mary) : \neg\text{likes}$ ,

Axiomas de inclusión de roles complejos:  $R \circ S \sqsubseteq Q$ , universal role  $U$ ,  $\exists R.\text{Self}$

**H**:  $R \sqsubseteq S$  **tieneHijo**  $\sqsubseteq$  **tieneDescendiente**

**O**:  $\{a\}$   $\{Uruguay\}$  **I**:  $S = R^{-}$  **Hijo** = **Padre**<sup>-</sup>

**N**:  $\geq nR.T$   $\geq 2\text{tieneHijo}.T$

**R**:  $\text{Dis}(R, S)$  **Dis(esAmigoDe, esEnemigoDe)**  $\text{Irr}(R)$  **Irr(tieneHijo)**

$R \circ S \sqsubseteq Q$  **esHermano**  $\circ$  **esPadre**  $\sqsubseteq$  **esTio**

Roles transitivos:  $\text{Trans}(R) \rightarrow R \circ R \sqsubseteq R$



# Lógica Descriptiva - Sintaxis

Base de conocimiento  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$

**TBox  $\mathcal{T}$** : Conocimiento básico, descripción intensional sobre el dominio de un problema.  $C \sqsubseteq D$

**ABox  $\mathcal{A}$** : Conocimiento sobre individuos, descripción extensional sobre el dominio de un problema.

$C(a)$   $R(a, b)$   $a = b$   $a \neq b$

$\mathcal{T} = \{Mujer \sqsubseteq Persona, Persona \equiv Mujer \sqcup Hombre,$

$Madre \equiv Mujer \sqcap \exists tieneHijo. Persona\}$

$C \equiv D \rightarrow C \sqsubseteq D$  y  $D \sqsubseteq C$

$\mathcal{A} = \{Mujer(maria), tieneHijo(maria, diego)\}$

# Lógica Descriptiva - Sintaxis

Base de conocimiento  $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$

**TBox  $\mathcal{T}$** : Conocimiento básico, descripción intensional sobre el dominio de un problema.  $C \sqsubseteq D$

**ABox  $\mathcal{A}$** : Conocimiento sobre individuos, descripción extensional, sobre el dominio de un problema.  $C(a)$   $R(a, b)$   $a = b$   $a \neq b$

**RBox  $\mathcal{R}$** : Conocimiento básico, descripción intensional sobre el conjunto de pares de elementos del dominio.  $R \sqsubseteq S$   $\text{Dis}(R, S)$   $R \circ S \sqsubseteq Q$

*Mujer*  $\sqsubseteq$  *Persona*

*tieneHijo(maria, diego)*

*mariajose = majo*

*esPadre*  $\circ$  *esPadre*  $\sqsubseteq$  *esAbuelo*

# Web Ontology Language: OWL

Lenguajes de ontologías permiten escribir una **conceptualización formal y explícita de un modelo de dominio**.

Requerimientos principales:

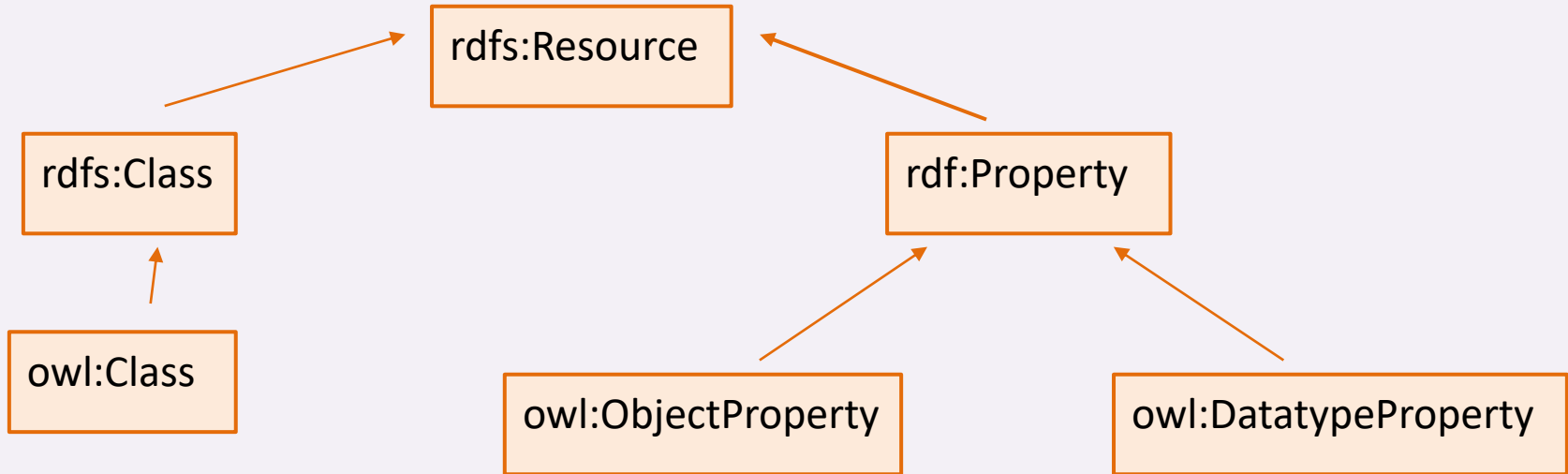
- **Sintaxis bien definida**
  - **Semántica bien definida**
  - **Soporte eficiente de razonamiento**
  - **Poder expresivo**
- Consistencia
  - Subsumption de clases
  - Clasificación de instancias

# Web Ontology Language: OWL

**OWL es un lenguaje de la Web Semántica para representar conocimiento complejo acerca de “cosas” y relaciones entre “cosas”**

**Es un lenguaje basado en lógica, tal que es posible aplicar algoritmos de razonamiento para verificar consistencia y hacer explícito conocimiento implícito.**

# Constructores OWL



rdfs:subClass  
rdfs:domain  
rdfs:range  
rdf:type

## Instance

owl:differentFrom  
owl:AllDifferent  
owl:sameAs

## Class

owl:disjointWith  
owl:equivalentClass  
owl:complementOf  
owl:unionOf  
owl:intersectionOf  
owl:oneOf

## Property

owl:Restriction  
owl:minCardinality  
owl:someValuesFrom  
owl:allValuesFrom  
owl:hasValue  
owl:TransitiveProperty  
owl:inverseOf  
owl:equivalentProperty  
owl:inverseFunctionalProperty  
owl:SymmetricProperty  
owl:FunctionalProperty

# Constructores OWL - DL

OWL Constructor	DL
SubClassOf	$C \sqsubseteq D$
intersectionOf	$C \sqcap D$
unionOf	$C \sqcup D$
complementOf	$\neg C$
oneOf	$\{a_1, a_2, \dots, a_n\}$
someValuesFrom	$\exists R.C$
allValuesFrom	$\forall R.C$
minCardinality	$\geq n.R$
maxCardinality	$\leq n.R$
cardinality	$= n.R$
hasValue	$\exists R.\{a\}$

# OWL: Web Ontology Language

- 2 familias de OWL, difieren en la expresividad de la DL

inclusión de roles:  $R \sqsubseteq S$

nominales:  $\{a\}$

– **OWL 1** - *SHOIN* number restrictions:  $\geq 3 \text{ hasChild}$

roles inversos:  $R^{-}$

Conceptos, roles (atómicos),  $\top$ ,  $\perp$ ,  $\sqcap$ ,  $\sqcup$ ,  $\exists$ ,  $\forall$ ,  $\neg$ , roles transitivos

– **OWL 2**  $\rightarrow$  *SRHIQ* qualified number restrictions:  $\geq 3 \text{ hasChild.Male}$

$\text{Dis}(R, S)$ , (ir)reflexive roles, negated role assertions:  $\neg \text{likes}(\text{John}, \text{Mary})$ , complex role inclusion axioms:  $R \circ S \sqsubseteq Q$ , universal role  $U$ , constructs:  $\exists R.\text{Self}$

# OWL1: profiles

- **OWL Lite:** Fácil de usar y de implementar herramientas
- **OWL DL:** Decidible (todas las conclusiones son realizables en un tiempo finito)
- **OWL Full:** Máxima expresividad sin garantía de computabilidad.



# OWL2: profiles

- OWL 2 EL: Para ontologías de gran tamaño, complejidad polinomial

$\top, \perp, \sqcap, \exists, \{a\}, \exists R.Self, keys, R \circ S \sqsubseteq Q$

- OWL 2 RL: Más expresivo que OWL 2 EL, complejidad polinomial

$\top, \perp, \sqcap, \sqcup, \exists, \{a\} \sqsubseteq \top, \perp, \sqcap, \forall, \neg, \leq 1R.C$

$R \circ S \sqsubseteq Q$

- OWL 2 QL: Orientado a consultas de grandes volúmenes de instancias, complejidad polinomial

$\top, \perp, \exists R.T \sqsubseteq \top, \perp, \sqcap, \neg, \exists$

$R \sqsubseteq S, R^-$

# Protégé: OWL Ontology Development Environment

- Versions:
  - 3.X → **OWL 1**
  - 4.X, 5.X → **OWL 2**
- Download: <http://protege.stanford.edu/>

# Sintaxis OWL-DL-Manchester

OWL Constructor	DL	Manchester Syntax (Protégé)
SubClassOf	$C \sqsubseteq D$	SubClassOf
intersectionOf	$C \sqcap D$	$C$ and $D$
unionOf	$C \sqcup D$	$C$ or $D$
complementOf	$\neg C$	not $C$
oneOf	$\{a_1, a_2, \dots, a_n\}$	$\{a_1, a_2, \dots, a_n\}$
someValuesFrom	$\exists R.C$	$R$ some $C$
allValuesFrom	$\forall R.C$	$R$ only $C$
minCardinality	$\geq n.R$	$R$ min $n$
maxCardinality	$\leq n.R$	$R$ max $n$
cardinality	$=n.R$	$R$ exactly $n$
hasValue	$\exists R.\{a\}$	$R$ value $a$

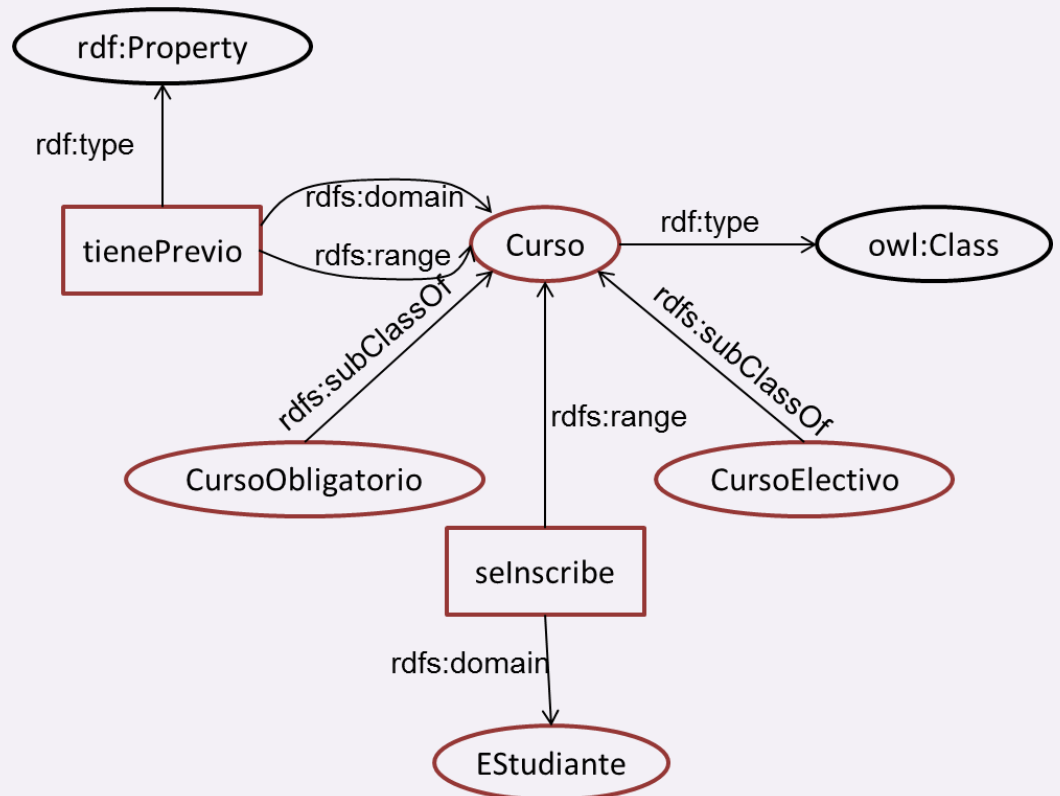
# Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.



# Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

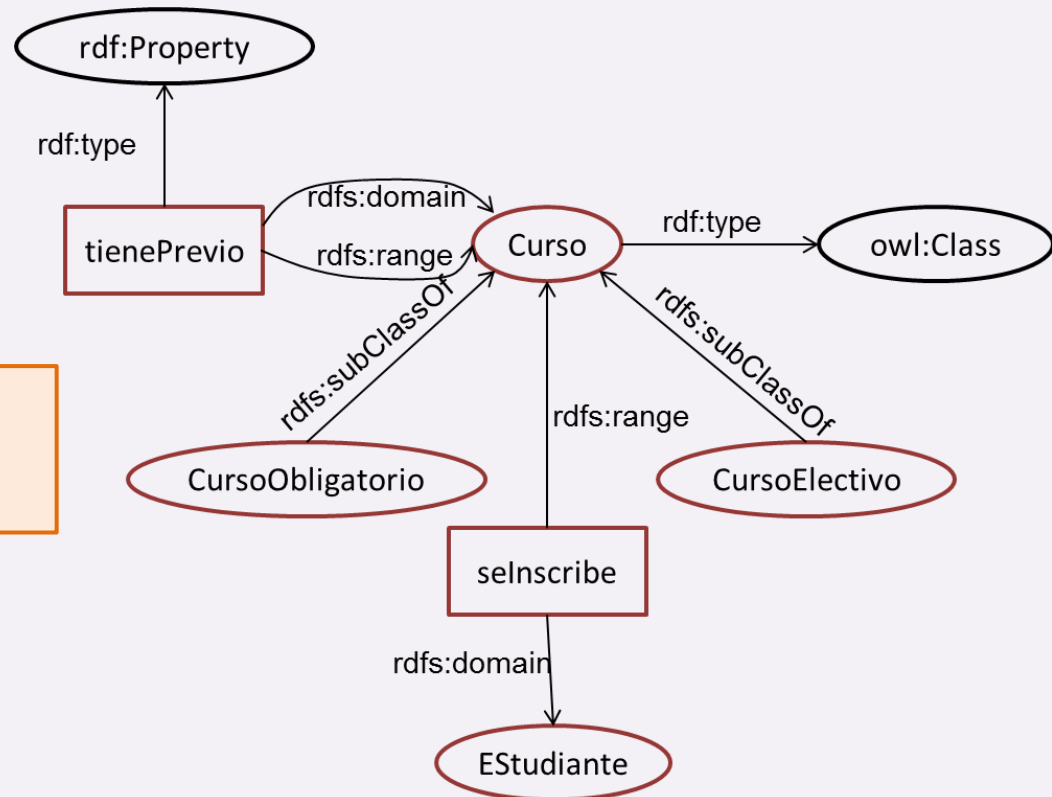
Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.

$CursoObligatorio \sqcap CursoElectivo \sqsubseteq \perp$   
 $Curso \equiv CursoObligatorio \sqcup CursoElectivo$

$Estudiante \sqsubseteq \leq 2 \text{seInscribe}.CursoElectivo$



# Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.

*CursoObligatorio*  $\sqcap$  *CursoElectivo*  $\sqsubseteq \perp$   
*Curso*  $\equiv$  *CursoObligatorio*  $\sqcup$  *CursoElectivo*

*Estudiante*  $\sqsubseteq \leq 2 \text{seInscribe} . \text{CursoElectivo}$

A los efectos de producir datos estadísticos en bedelía se necesita obtener la siguiente información:

- Una lista de todos los cursos que tengan tres o más cursos previos.
- Un lista con todos los estudiantes inscriptos a cursos obligatorios y a cursos electivos.

# Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.

$CursoObligatorio \sqcap CursoElectivo \sqsubseteq \perp$   
 $Curso \equiv CursoObligatorio \sqcup CursoElectivo$

$Estudiante \sqsubseteq \leq 2 \text{ seInscribe. } CursoElectivo$

A los efectos de producir datos estadísticos en bedelía se necesita obtener la siguiente información:

- Una lista de todos los cursos que tengan tres o más cursos previos.
- Un lista con todos los estudiantes inscriptos a cursos obligatorios y a cursos electivos.

$CursoMinTresPr \equiv Curso \sqcap \geq 3 \text{ tienePrevio. } Curso$

$EstudianteOblElect \equiv Estudiante \sqcap \exists \text{ seInscribe. } CursoObligatorio \sqcap \exists \text{ seInscribe. } CursoElectivo$

# Ejercicio

## **Ejecutar el razonador.**

Ingresar individuos para: estudiantes, cursos obligatorios y cursos electivos.

Para cualquiera de los individuos de estudiantes, ingresar una inscripción para un curso obligatorio y otra para un curso electivo.

Para cualquiera de los individuos de estudiantes, ingresar inscripciones para más de dos cursos electivos.

Para cualquiera de los individuos de cursos (obligatorios o electivos), ingresar varias instancias que lo vinculen a 3 o más cursos previos.

## **Ejecutar el razonador y observar las inferencias generadas.**

**Se obtuvieron las inferencias esperadas?**



# Ejercicio (mundo abierto)

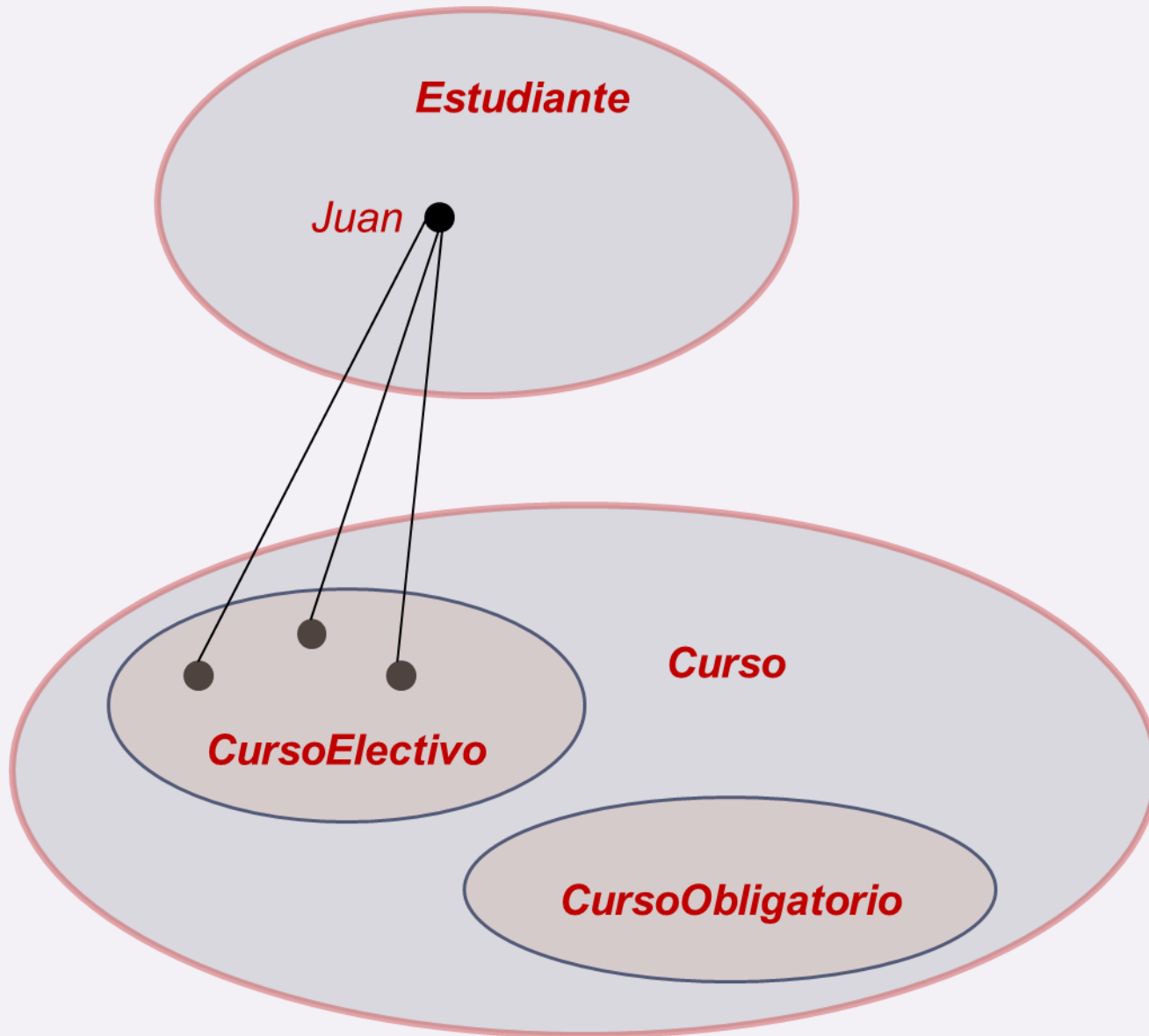
**No se detectó inconsistencia:**

“Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos”

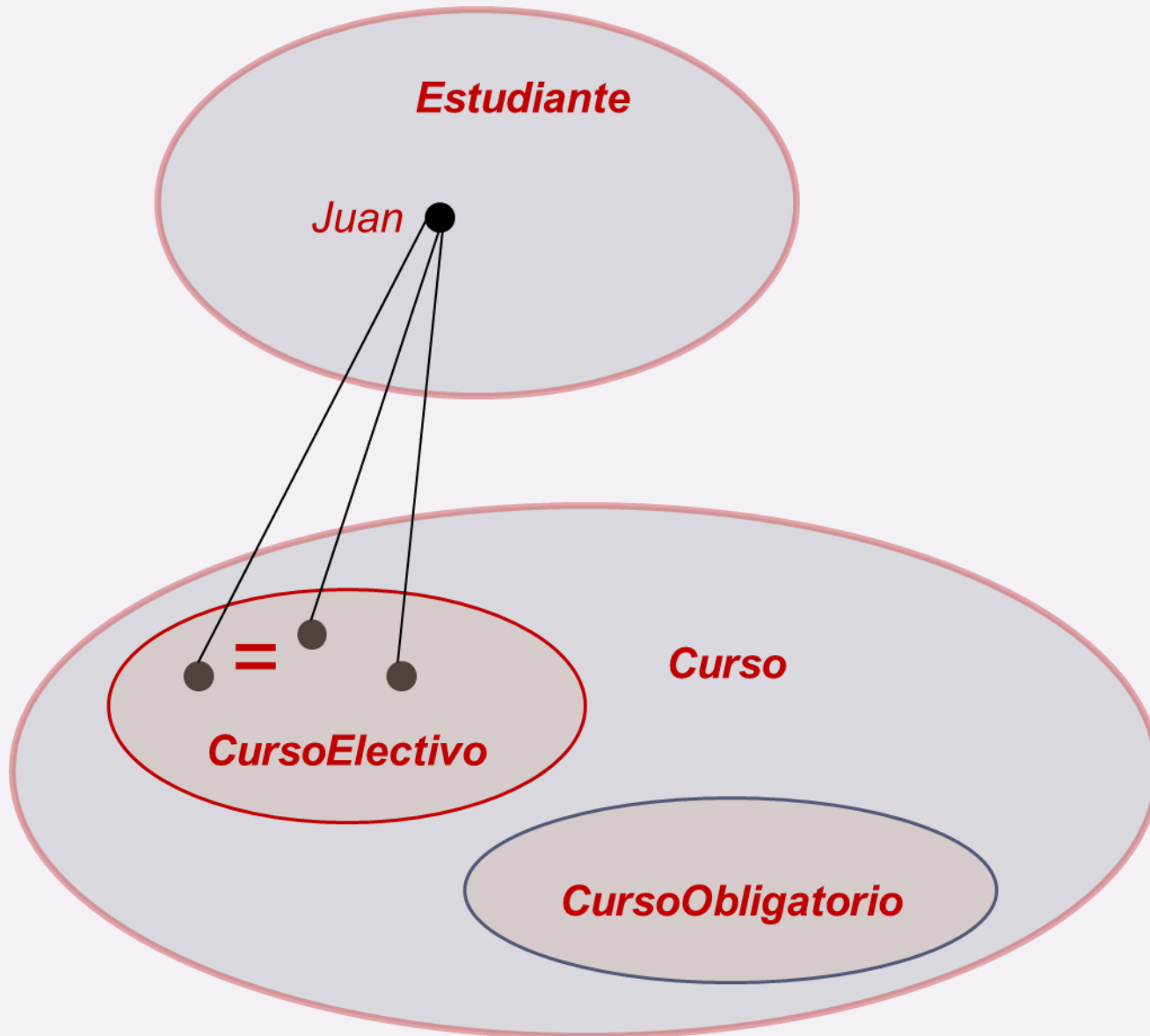
**No se clasificaron los individuos de cursos que satisfacen:**

“Una lista de todos los cursos que tengan tres o más cursos previos”

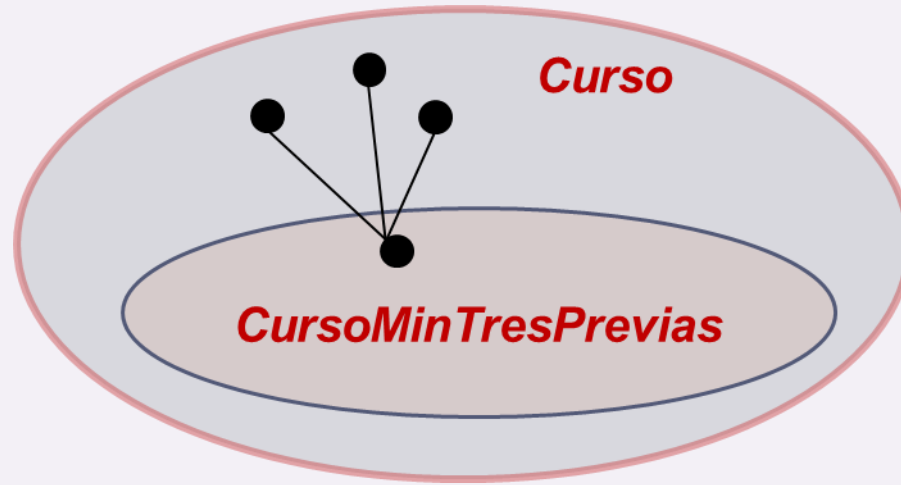
# Ejercicio (mundo abierto)



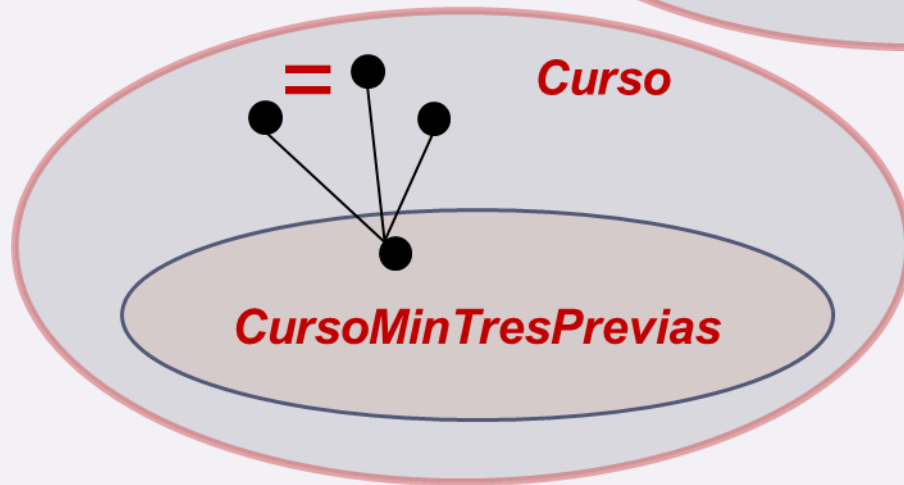
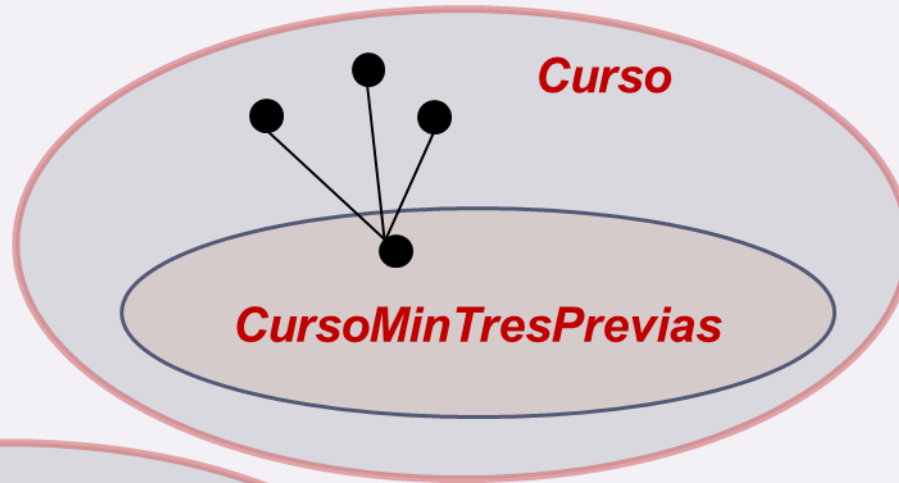
# Ejercicio (mundo abierto)



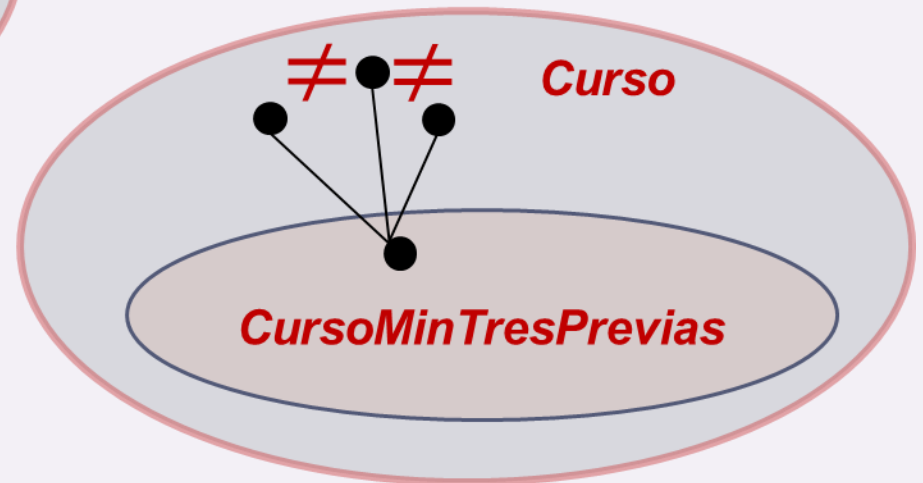
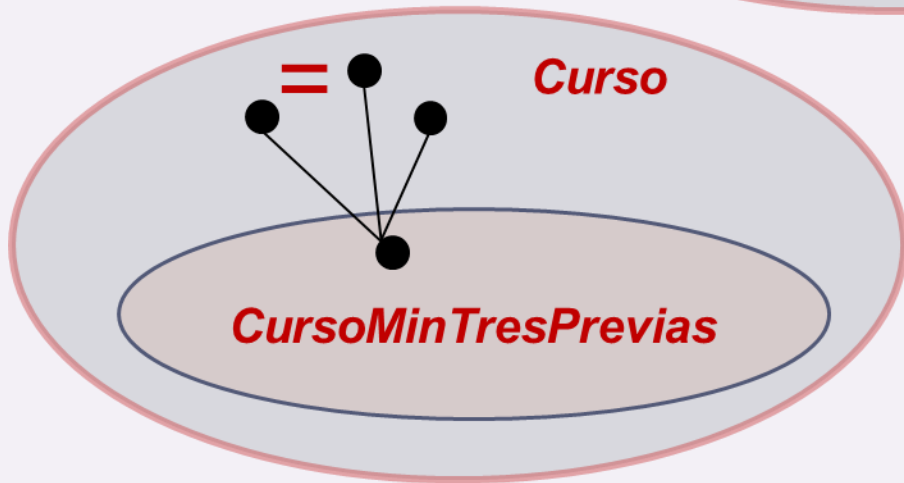
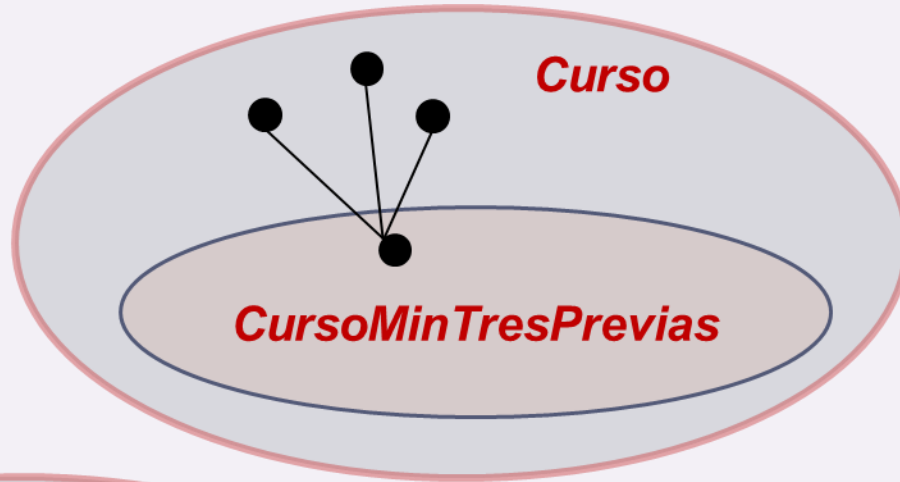
# Ejercicio (mundo abierto)



# Ejercicio (mundo abierto)



# Ejercicio (mundo abierto)



# Ejercicio

Agregar al modelo que los estudiantes “aprueban” exámenes y que los exámenes “corresponden a” cursos.

Un estudiante “salva” un curso cuando aprueba el examen correspondiente a dicho curso.

Además, representar que cada estudiante tiene un único número de estudiante.

Para cualquiera de los individuos de estudiantes ingresar una instancia “aprueba” para un individuo de exámenes y para éste ingresar una instancia “corresponde a” para un individuo de cursos.

Para uno de los individuos de estudiantes ingresar una instancia para asignarle un número de estudiante.

**Ejecutar el razonador y observar las inferencias generadas.**

Asociar al otro individuo de estudiantes el mismo número de estudiante.

**Ejecutar el razonador y observar las inferencias generadas.**

# Protégé-OWL API

Open-source Java library para Web Ontology Language y RDF(S).  
Permite cargar y guardar archivos OWL, consultar y manipular OWL data models, y ejecutar razonamiento.

<http://protege.stanford.edu/plugins/owl/api/>

- Protégé-OWL API: guía para el desarrollador

[http://protegewiki.stanford.edu/wiki/ProtegeOWL\\_API\\_Programmers\\_Guide](http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide)

- Ejemplos de construcción de aplicaciones semánticas

<http://protegewiki.stanford.edu/wiki/BuildingSemanticWebApplications>



# Protégé-OWL API

- **Crear un modelo OWL a partir de un archivo .owl.**

```
String uri =  
"file:/c:/Facultad/Postgrado%20Pedeciba/Proyectos/Proyecto%20SALUS/Ontologies/WebSiteSpecialization  
.owl";  
OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
```

- **Obtener una clase del modelo:**

```
OWLNamedClass WebContentClass = owlModel.getOWLNamedClass("http://www.owl-  
ontologies.com/WebSite#WebContent");
```

```
OWLNamedClass AuthorClass = owlModel.getOWLNamedClass("http://www.owl-  
ontologies.com/WebSiteSpecialization#Author");
```

- **Obtener una object property:**

```
OWLObjectProperty hasAuthorProperty =  
owlModel.getOWLObjectProperty("http://www.owl-ontologies.com/WebSiteSpecialization#hasAuthor");
```

- **Crear una instancia:**

```
OWLIndividual AlzheimerWebContent =  
WebContentClass.createOWLIndividual("AlzheimerWebContent");
```

```
OWLIndividual JuanPerezAuthor = AuthorClass.createOWLIndividual("JuanPerezAuthor");
```

- **Asociar la instancia de una propiedad a una instancia dada:**

```
AlzheimerWebContent.setPropertyValue(hasAuthorProperty, JuanPerezAuthor);
```

# Protégé-OWL API

- **Crear una instancia del razonador:**

```
ReasonerManager reasonerManager = ReasonerManager.getInstance();
```

```
ProtegeReasoner reasoner =
```

```
reasonerManager.createProtegeReasoner(owlModel, ProtegePelletOWLAPIReasoner.class);
```

- **Controlar conceptos inconsistentes, clasificar conceptos e instancias:**

```
reasoner.computeInconsistentConcepts();
```

```
reasoner.classifyTaxonomy();
```

```
reasoner.computeInferredIndividualTypes();
```

- **Obtener instancias clasificadas en un concepto:**

```
Collection webContentIndividuals = reasoner.getIndividualsBelongingToClass(WebContentClass);
```

- **Obtener subclases inferidas:**

```
Collection inferredSubclasses = WebContentClass.getInferredSubclasses();
```

# Bibliografía

Handbook on Ontologies. Steffen Staab, Rudi Studer (Editors). 2004.

Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph: Foundations of Semantic Web Technologies, Chapman & Hall/CRC, 2009.

Capítulos 4 y 5.