

Protégé pentru construirea ontologiilor

Cuprins

Protégé pentru construirea ontologiilor.....	1
Cuprins.....	1
Instalare Protégé	2
Introducere	2
Componetele unei ontologii OWL versus componentele unei ontologii Protégé.....	2
Indivizi.....	3
Clasele	3
Proprietățile	3
Crearea unei ontologii OWL	5
Convenții de denumire	9
Interfața grafică.....	10
Ierarhia de clase și ierarhia de proprietăți.....	14
Stabilirea caracteristicilor proprietăților OWL	22
Stabilirea domeniilor și codomeniilor pentru proprietățile OWL.....	26
Restricții de cuantificare	32
Restricții de cuantificare de tip existențial	32
Restricții de cuantificare de tip universal	44
Restricții de cardinalitate	47
Proprietăți-atribut	50
Convenția OWL privind Lumile Deschise în Protege	57
Crearea membrilor unei clase (indivizi).....	61
Crearea unei clase prin restricții de tip hasValue.....	62
Crearea unei clase prin enumerarea membrilor săi	64
Crearea interogărilor	66
Bibliografie	66

Instalare Protégé

Universitatea Stanford, California

Google.com → <http://protege.stanford.edu/> =>

Introducere

Ontologiile sunt utilizate în Semantic Web pentru a reprezenta cunoștințele specifice dintr-un domeniu de interes. Ele descriu concepțele din domeniu și relațiile dintre aceste concepțe.

Față de **OWL**, limbajul standard pentru ontologii creat de **Grupul de Lucru privind Ontologiile Web** din **W3C, Protégé**:

- se bazează pe un alt tip de model logic ⇒ permite
 - atât definirea cât și descrierea conceptelor ⇒ construirea de concepte complexe pronind de la definiția conceptelor simple;
 - utilizarea unui instrument de verificare (*reasoner*) capabil să aprecieze consistența globală a ontologiei (oricare două concepte sunt mutual necontradictorii) și compatibilitatea definițiilor cu concepțele pe care le definesc ⇒ acest instrument asigură permanent corectitudinea ierarhiei (lucru esențial mai ales pentru ontologiile care comportă clase care au mai mult de o singură clasă-părinte);
- oferă facilități suplimentare: are un set mai bogat de operatori (*not*, *and*, *or*). În continuare, ne vom axa pe OWL DL.

Componentele unei ontologii OWL versus componentele unei ontologii Protégé

Componentele unei ontologii OWL	Componentele unei ontologii Protégé
Indiviși	Instanțe
Clase	Clase
Proprietăți	Sloturi

Indivizii

Indivizii (**OWL**) sau instanțele **Protégé** reprezintă obiectele din domeniul modelat. Convenția grafică utilizată: romburi, ca în Figura 1.

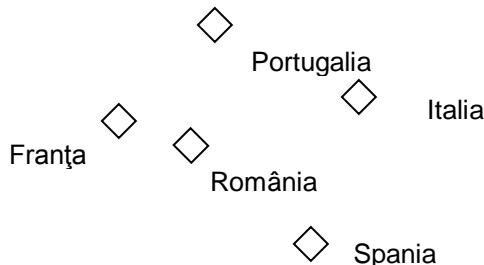


Figura 1 Reprezentarea indivizilor (instantelor)

O diferență importantă între **Protégé** și **OWL** este legată de convenția **UNA = Unique Name Assumption**, ignorată de **OWL** dar recunoscută de **Protégé**.

În **OWL** trebuie specificat explicit faptul că doi indivizi (sau două clase) sunt identici (cu clauze `owl:sameIndividualAs` sau `owl:sameAs`) sau diferiți (cu clauze `owl:differentFrom` sau `owl:allDifferent` împreună cu clauza `owl:distinctMembers`). Altfel, în absența unor precizări suplimentare, utilizarea unor nume diferite îndreptățește atât concluzia că este vorba despre unul și același individ (sau clasă) cât și concluzia că sunt indivizi (clase) diferiți.

Clasele

Clasele sunt interpretate ca multimi de indivizi sau ca o reprezentare concretă a conceptelor din domeniul discursului. În **OWL DL**, instrumentul de verificare (*reasoner*) poate ‘calcula automat’ taxonomia domeniului (ierarhia de clase și supraclase).

Clasele sunt reprezentate prin cercuri sau elipse, într-un mod asemănător diagramelor **Venn**.

Proprietățile

În **OWL** există următoarele tipuri de proprietăți

- proprietăți-legătură, care coreleză obiectele!! care compun clasele (de exemplu: pictează, compune etc.);
- proprietăți-atribut, care coreleză obiectele cu valorile atributelor lor (de exemplu: culoareOchi, dataNașterii, dataApariției, înaltime, greutate etc.).

Acste valori pot fi:

- valori **XML Schema DataType**;
- literalii **rdf**,

- proprietăți-de-adnotare, care adaugă informații (metadate = date despre date) claselor, indivizilor, respectiv proprietăților-obiect sau proprietăților-tip-de-date. Pot fi:
 - proprietățile-de-adnotare-legătură,
 - proprietățile-de-adnotare-atribut.

Proprietățile sunt reprezentate prin arce orientate de la clasa / individul domeniu de definiție la clasa / individul sau la literalul domeniu de valori.



Figura 5.2 O proprietate-legătură care corelează doi indivizi

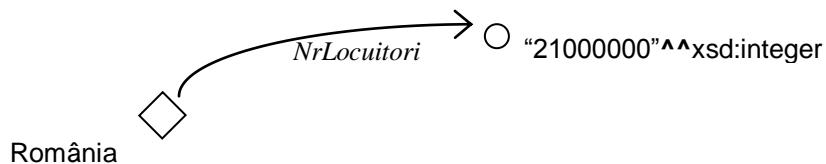


Figura 3 O proprietate-atribut care corelează un individ cu un literal de tip
xml:integer

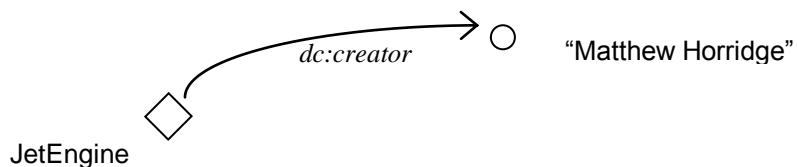


Figura 4 O proprietate-adnotare care corelează clasa ‘JetEngine’ cu literalul (stringul)
‘Matthew Horridge’

Proprietățile (în **OWL**) se mai numesc și sloturi (în **Protégé**), roluri (în logicile descriptive), relatii (în **UML**), attribute (în **GRAIL**).

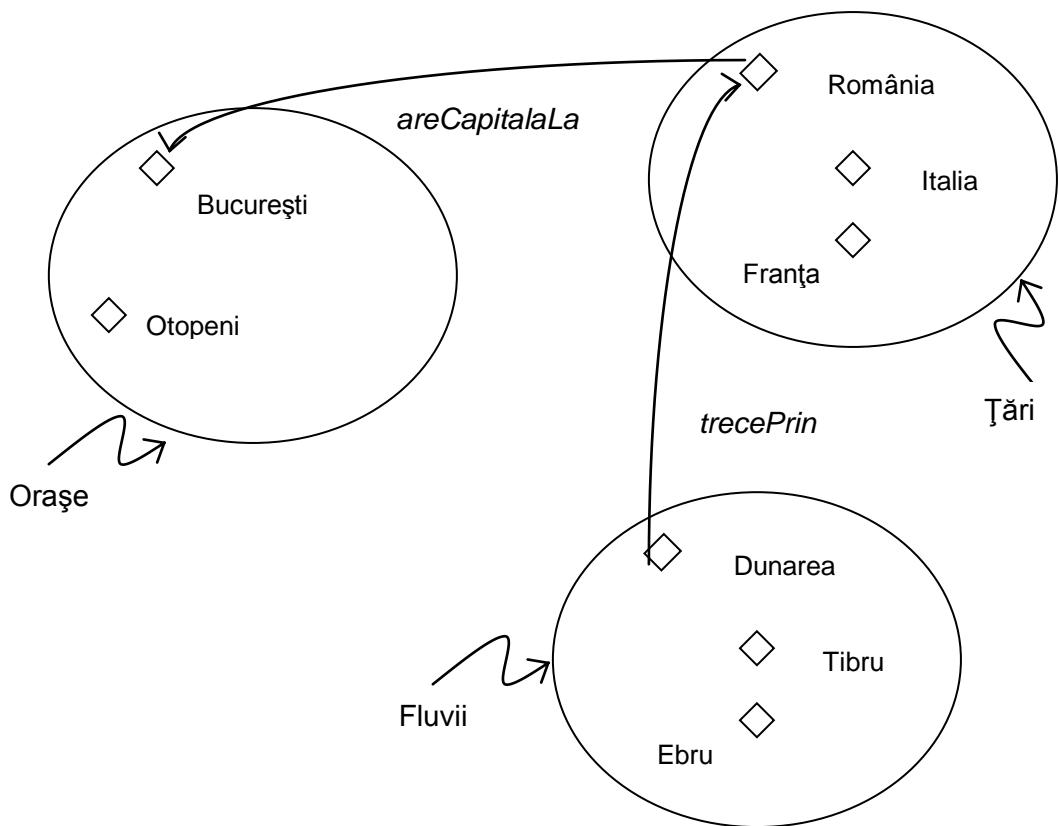


Figura 5.5. Reprezentarea indivizilor, claselor și proprietăților

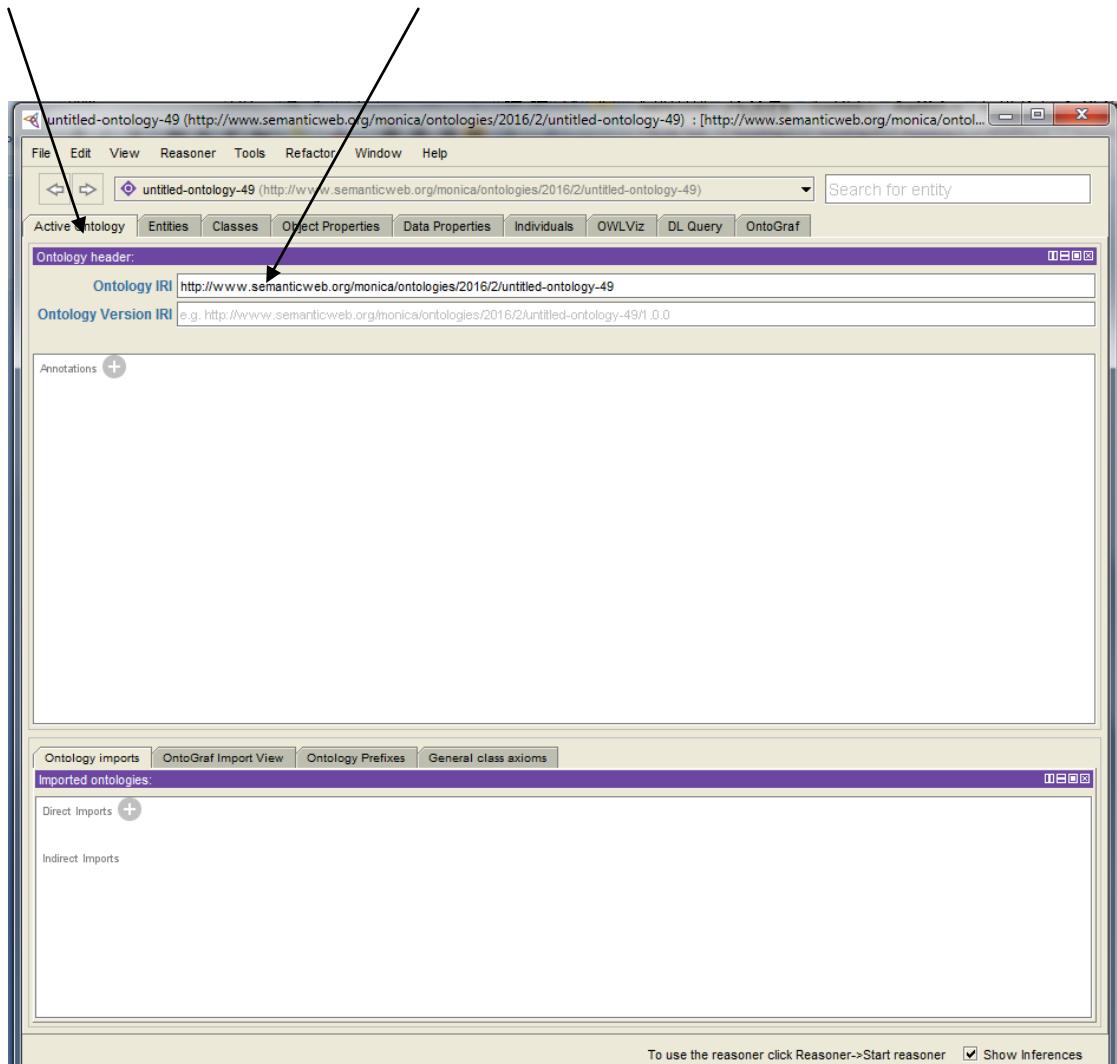
Crearea unei ontologii *OWL*

Vom prezenta modul de creare a unei ontologii; exemplificarea se face pe o ontologie privind pizza (vezi bibliogr.).

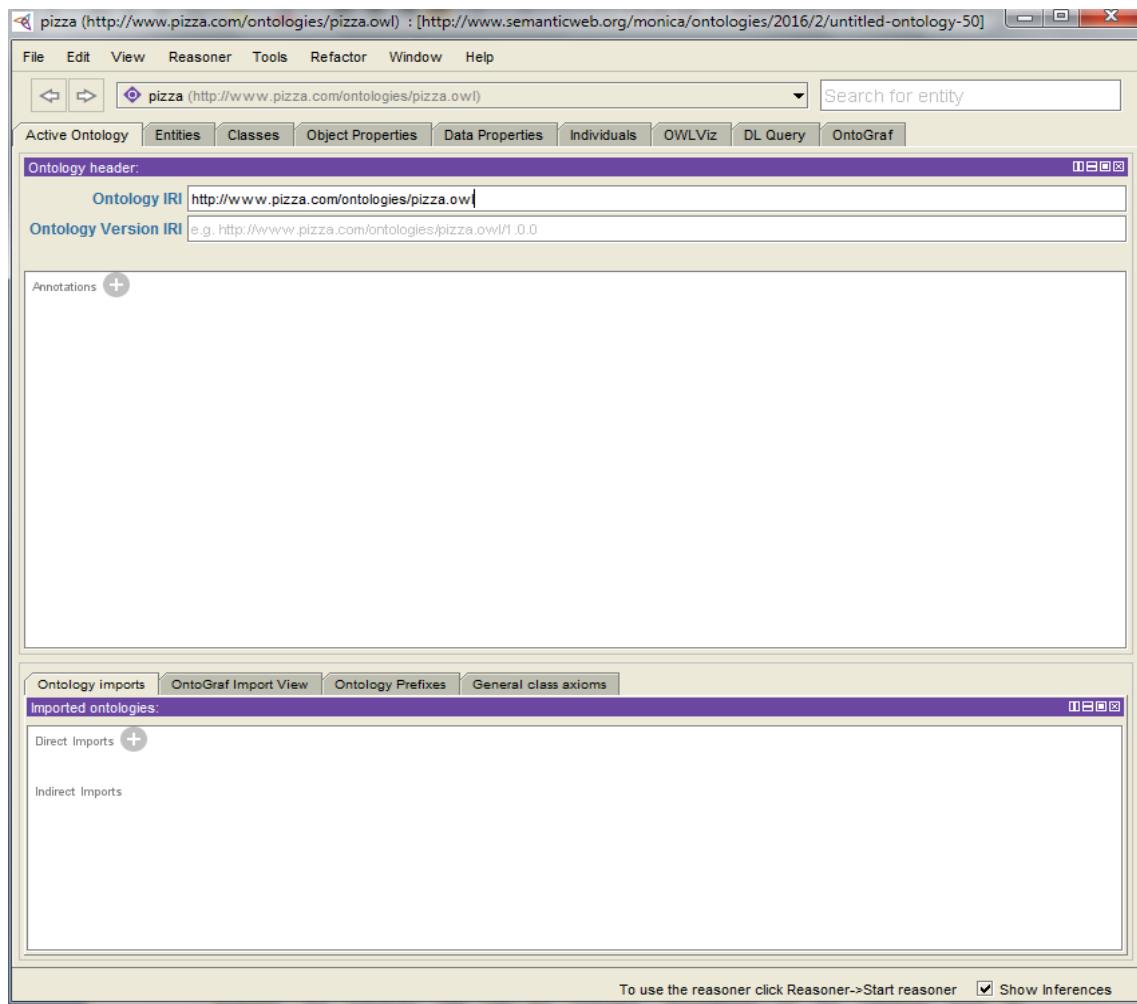
Tema 2: Lansarea aplicatiei si crearea unui project OWL

- se lansează aplicația **Protégé 4.3** ⇒ fereastră-dialog (fd) de tip *NEW*
-> în linia *Ontology URI* se tastează numele ontologiei care trebuie creată

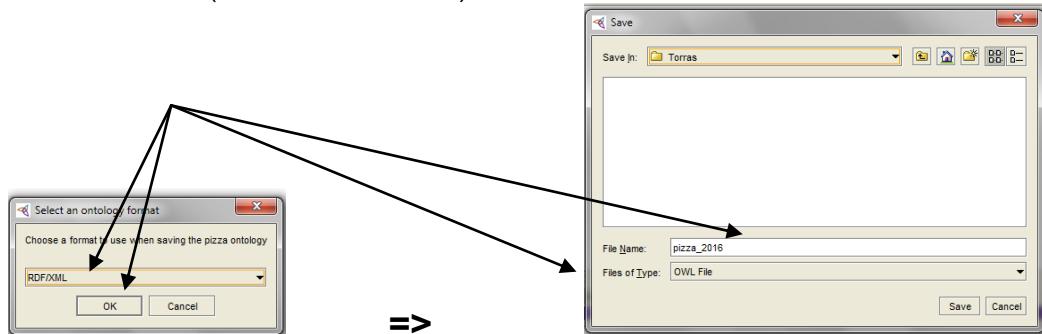
tabul activ numele ontologiei



Editorul Protégé pentru crearea ontologiilor



- se salvează (formatul RDF/XML)=>

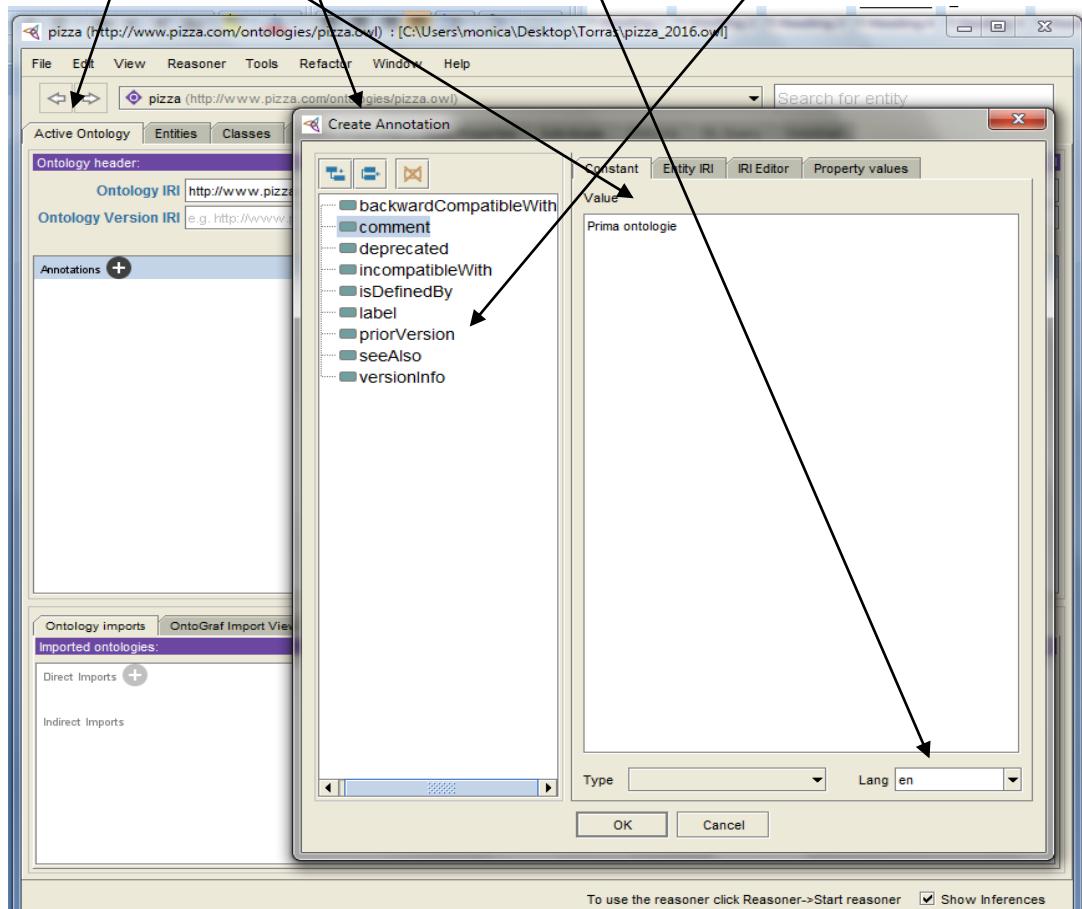


- optional se adaugă un comentariu:

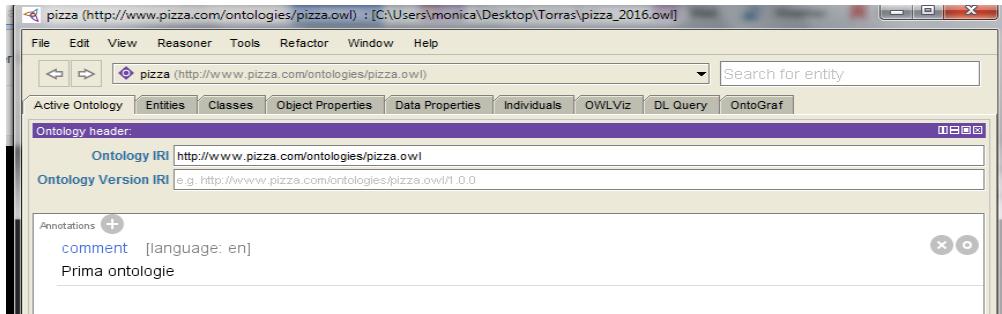
tabul *Active Ontology* → panelul *Annotations* -> iconul *Add* 

=> fd *Create Annotation* (apar toate opțiunile din elementul <owl ontology> din antetul unui document OWL (vezi Curs4: OWL)

-> se tastează un text (optional: se alege limba) -> OK



⇒ fd:



Convenții de denumire

De evitat caracterele speciale (este permis, eventual, underscore)

- ✓ Clasele: substantive la singular, scrise cu initiala majuscula și fară spații (PizzaBase). indiferent de alegere, e necesară coerenta pt ca un identificator scris greșit (vezi Case-sensitive) poate genera erori
- ✓ Proprietățile-legătură: verbe sau locuțiuni verbale care conțin verbele a fi / a avea; utilizarea majusculelor pentru inițiale – cu excepția primului cuvânt, evitarea spațiilor, a caracterelor speciale etc. (esteCondusDe; esteScrisDE);
- ✓ Proprietățile-atribut: substantive, locuțiuni verbale (greutate; areGreutateaDe).

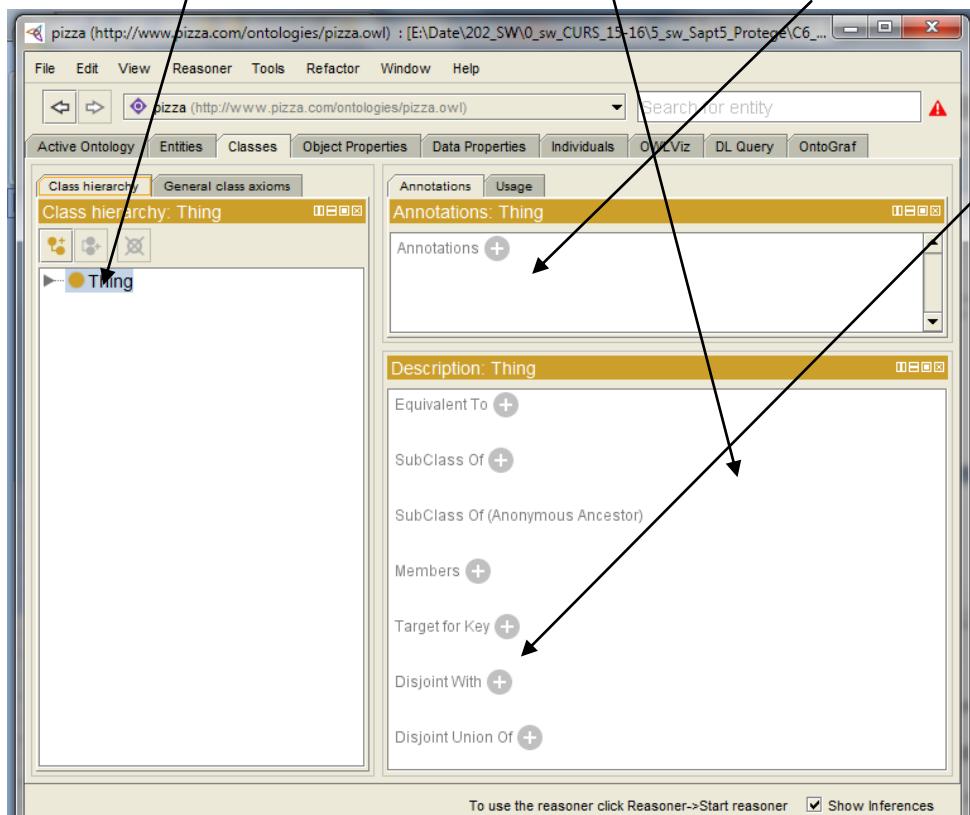
Interfață grafică

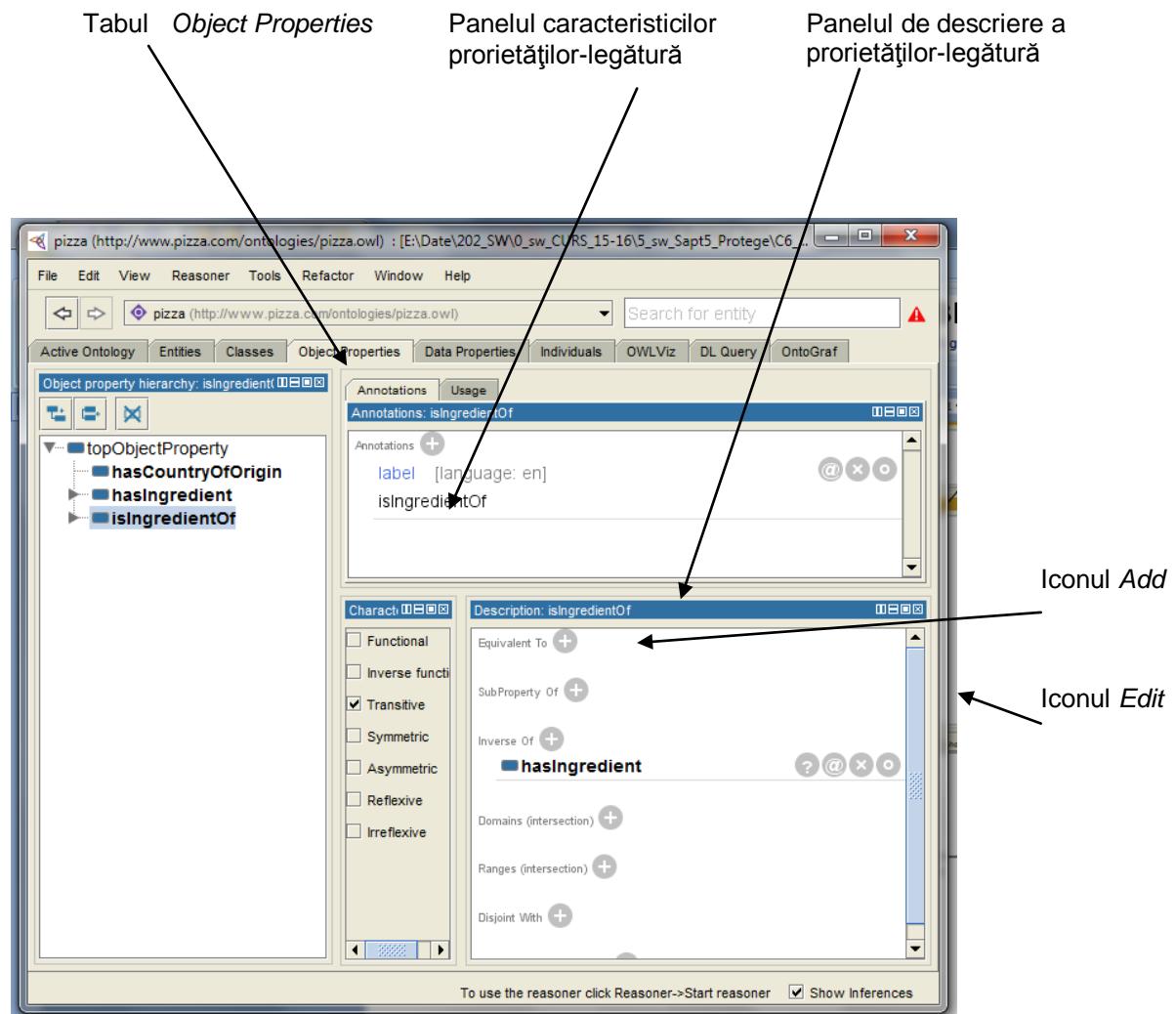
După lansarea aplicației și deschiderea ontologiei se deschide fereastra-dialog de mai jos:

Tabul *Classes; Thing*: clasa cea mai cuprinzătoare

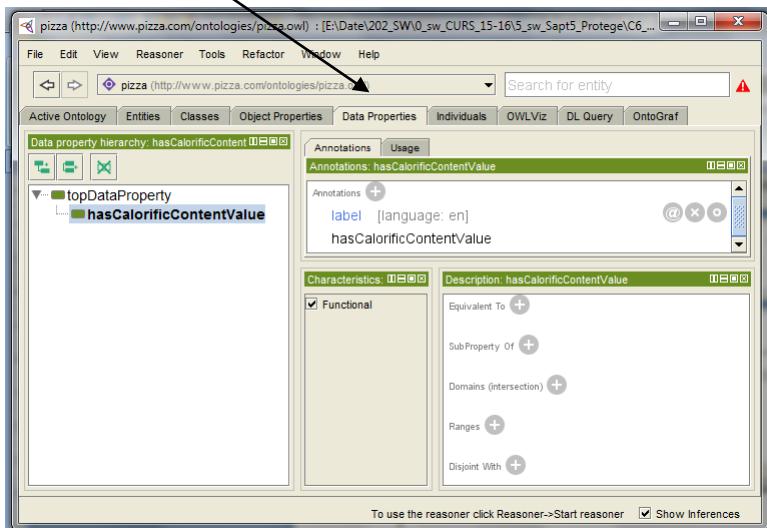
Panelul de descrierea proprietăților clasei selectate;
Panelul de adnotare

Iconul *Add*

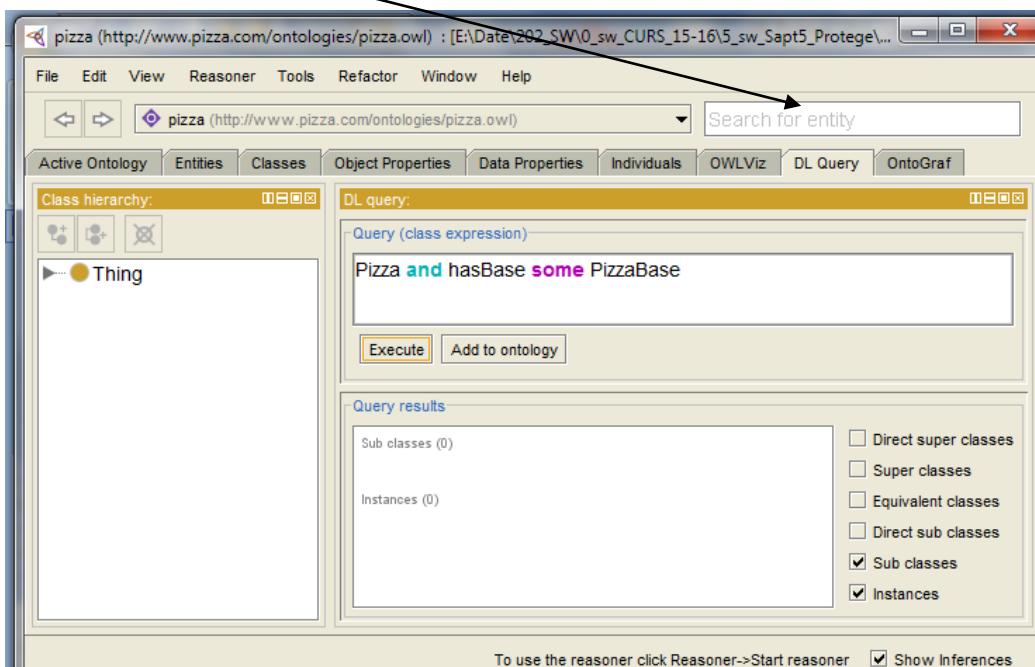




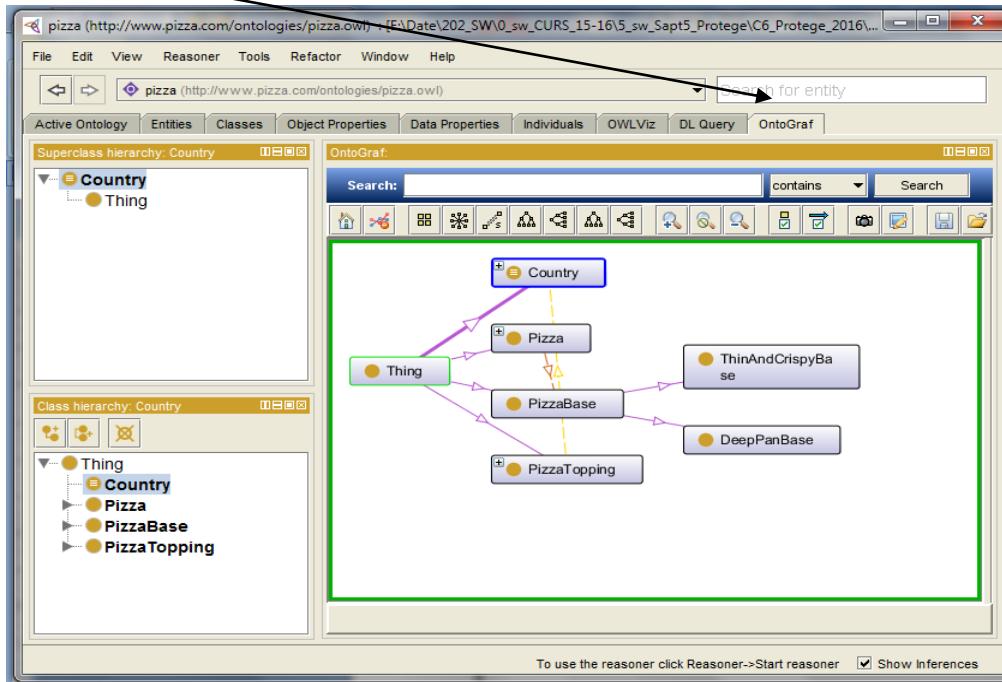
Tabul *Data Properties*



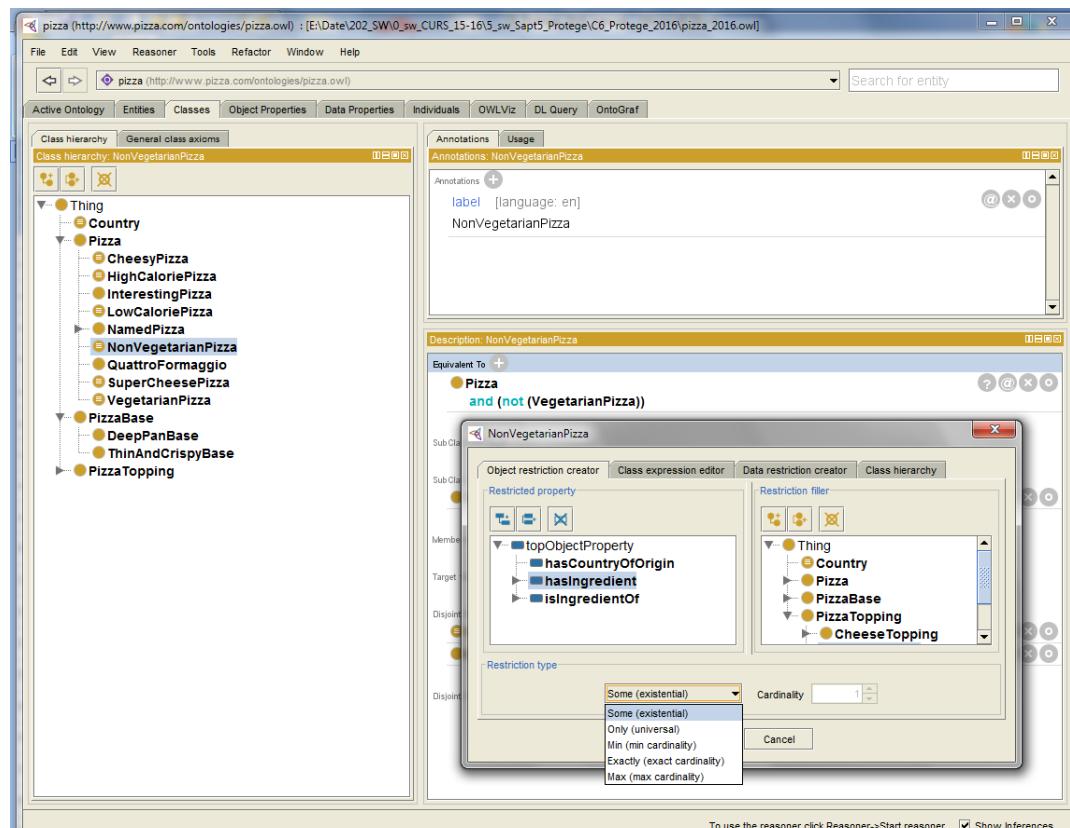
Tabul *DL Query*



Tabul *OntoGraph*



Fereastra-dialog de definire (nu doar de declarare) a claselor (vom reveni)



Observații

Pt afișarea/ascunderea taburilor:

meniuul *Window* → comanda *Tabs* → numele tabului care trebuie afișat/ascuns

Pentru schimbarea ordinii claselor/proprietăților:

meniu *View* → *Render by label*

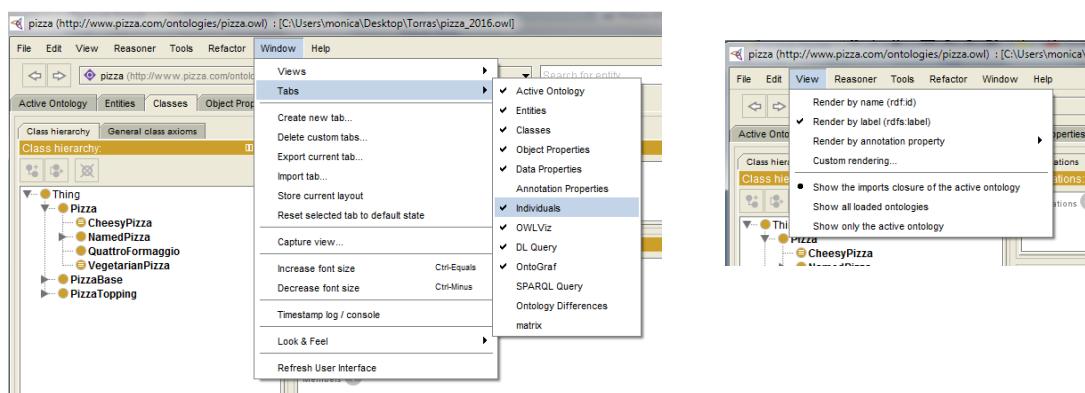
Pentru schimbarea dimensiunii fontului:

meniu *View* → *Custom rendering*

Pentru schimbarea numelui claselor / proprietăților etc.:

meniu *Refactor* → *Re mane entity*

Protege dispune de o facilitate *AutoComplete* care se activează cu tasta TAB.



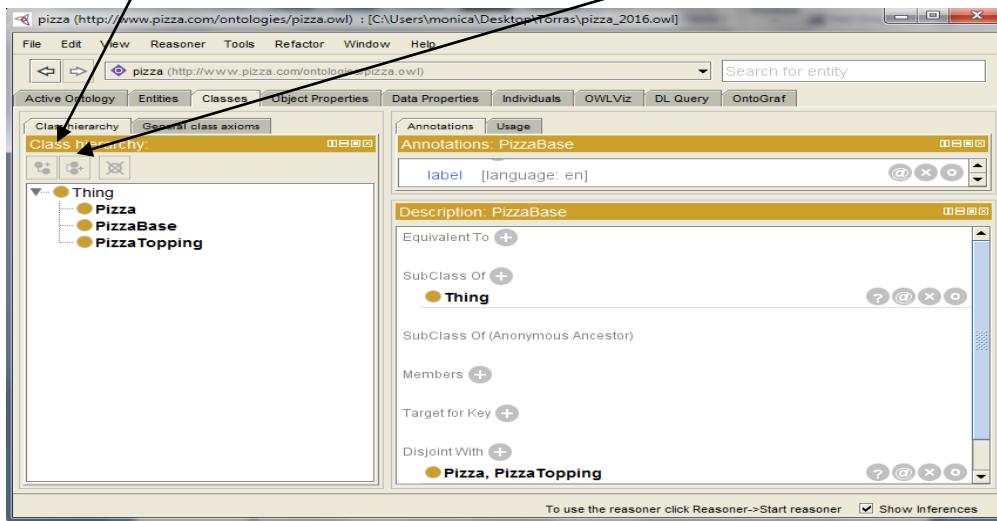
Ierarhia de clase și ierarhia de proprietăți

Tema 4: Crearea unei ierarhii de clase

Orice clasă este subclasă a clasei *Thing*

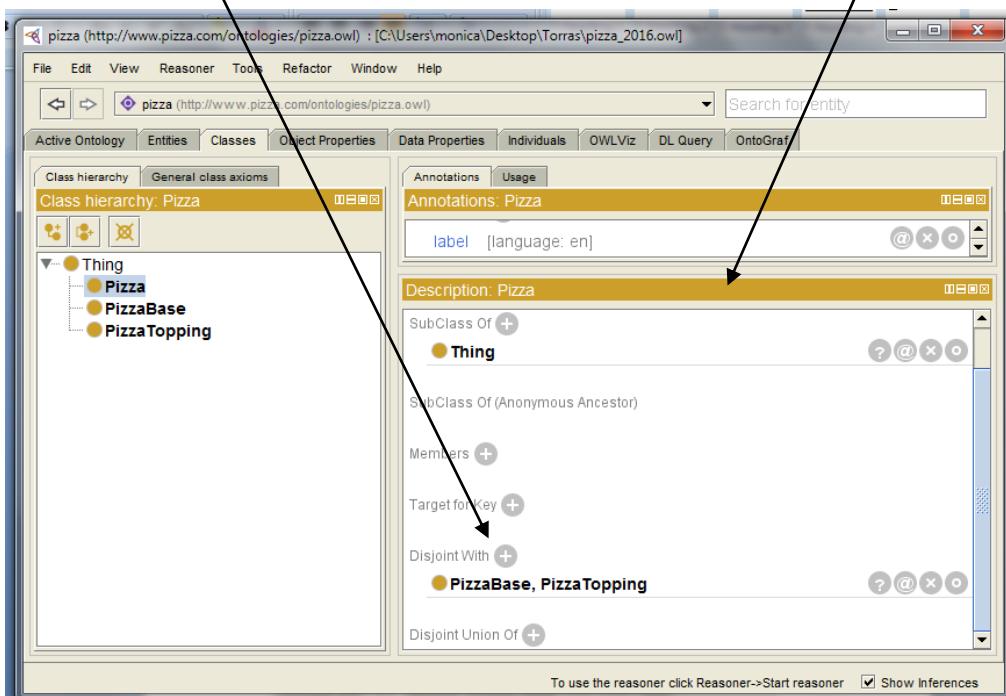
Vom incepe prin a crea clasa **Pizza** cu subclasele **PizzaTopping** și **PizzaBase**:

tabul *Classes* activ → se folosesc succesiv iconii *Add subclass* și *Add sibling class* în funcție de clasa selectată (supraclasă / clasă "soră")



Tema 5: Clase disjuncte

se selectează clasa **Pizza** → panelul *Description numeClasă* (aici *Description Pizza*)
→ directiva *Disjoint With* → clik pe iconul *Add* ⇒ TOATE cele 3 clase "surori" sunt disjuncte.



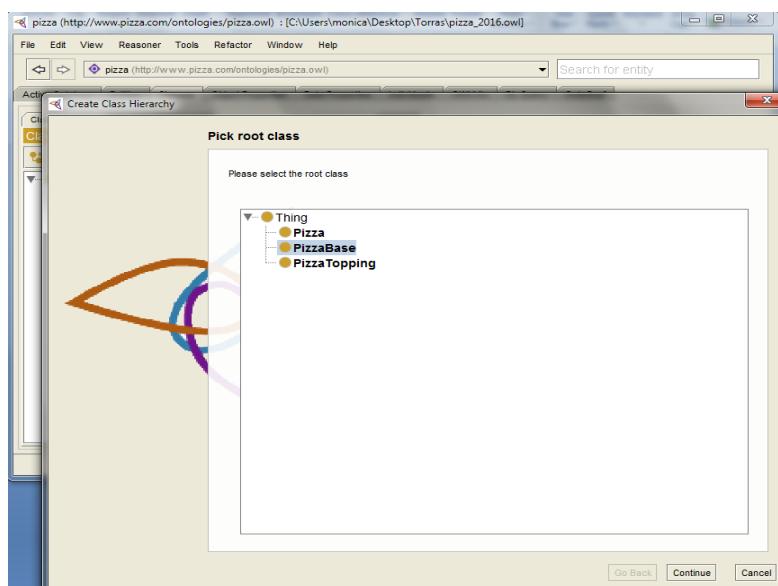
Observație

Declaratia *Disjoint With* este necesară pt că în OWL clasele sunt apriori nedisjuncte. ca atare, nu se poate presupune că $x \notin A$ numai pt că NU s-a afirmat că x nu face parte din A. Pt a separa clasele dintr-o taxonomie OWL trebuie să le declarăm EXPLICIT ca fiind disjuncte.

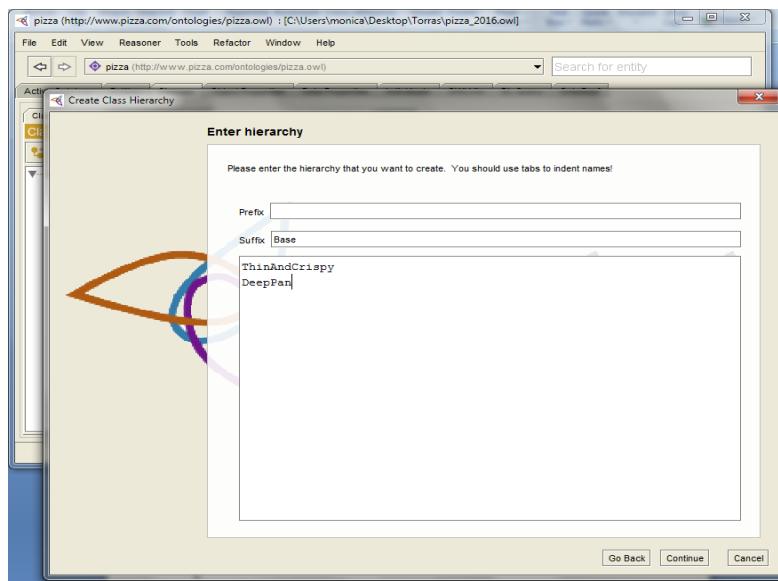
Tema 6: Crearea rapidă a unei ierarhii de subclase a unei clase

se selectează supraclasa (aici **PizzaBase**) →

→ meniul *Tools* → comanda *Create class hierarchy...* ⇒ procedura de asistență



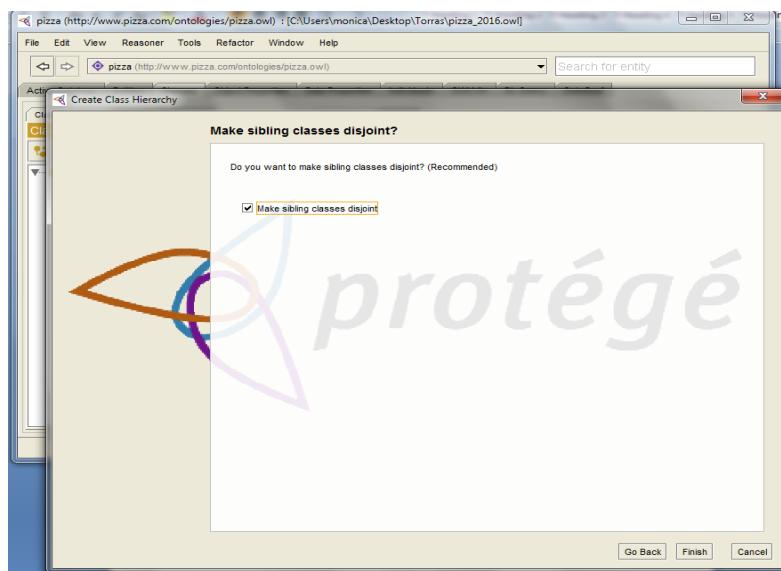
→ butonul de comandă *Continue* ⇒ fd



→ putem introduce numele subclaselor (cu ENTER) și un prefix și/ sau un sufix (după caz) → butonul de comandă *Continue* ⇒ fd

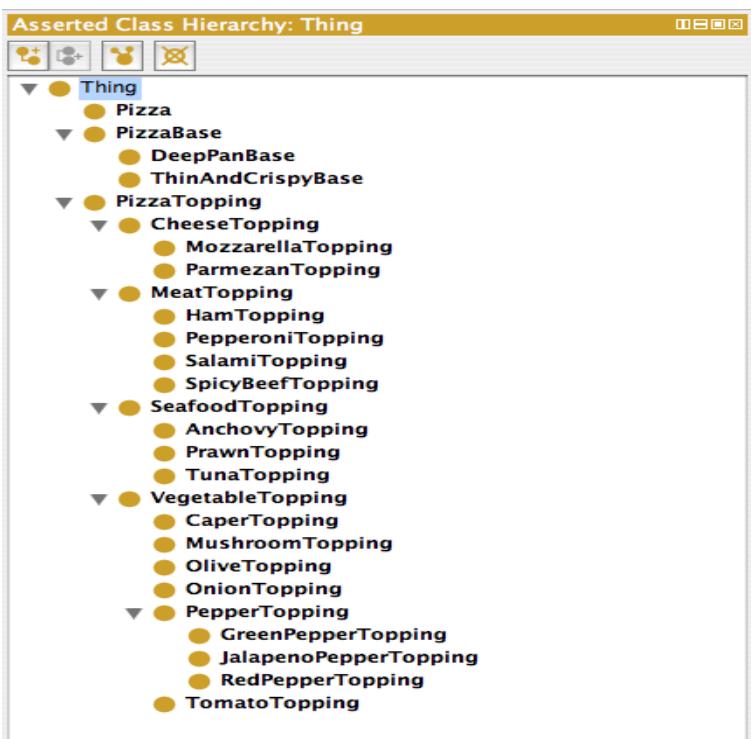
Observație

Cu TAB se poate crea și o subclasă a unei clase.



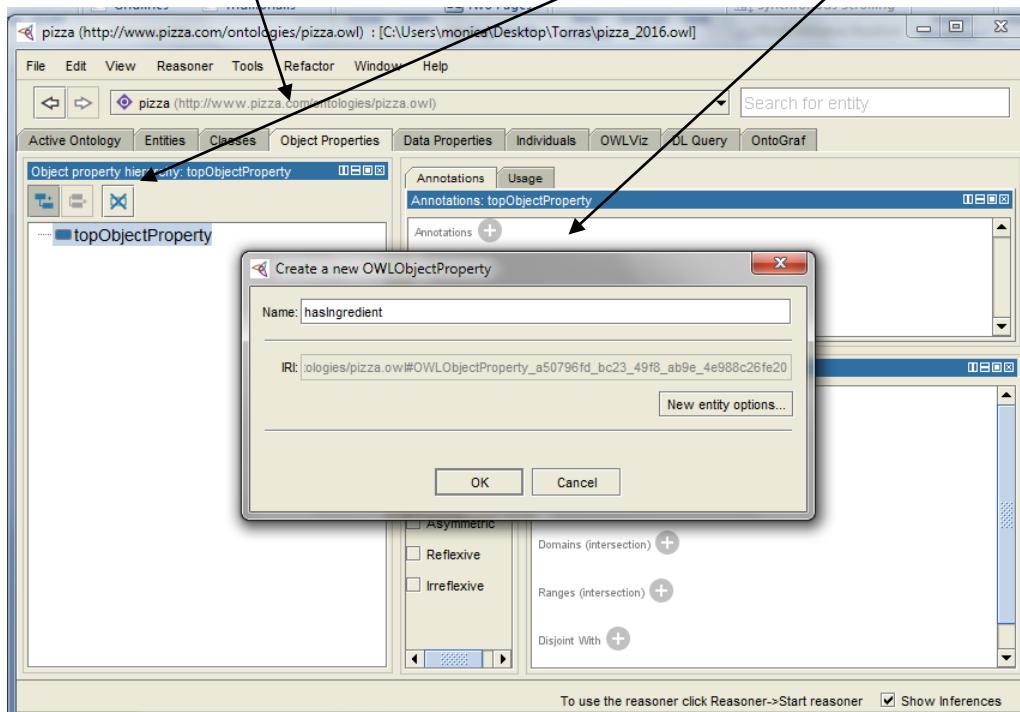
→ eventual, deselectăm caseta de opțiune *Make sibling classes disjoint*
→ butonul de comandă *Finish*
⇒ s-au creat subclasele disjuncte **DeepPanBase** și **ThinAndCrispyBase** ale clasei **PizzaBase**.

Analog creăm subclase ale clasei **PizzaTopping** ⇒ ierarhia de clase va arăta astfel:



Tema 8: Crearea unei proprietăți-legătură (hasIngredient)

tabul Object Properties activ → butonul de comandă Add subproperty sau Add sibling property (în funcție de proprietatea selectată) ⇒ fd Create a new OWLObjectProperty → se tastează numele proprietății → OK ⇒ ⇒ proprietatea apare în panelul Object Property iar în panelul Description: Property apare ca subproprietatea lui **topObjectProperty**



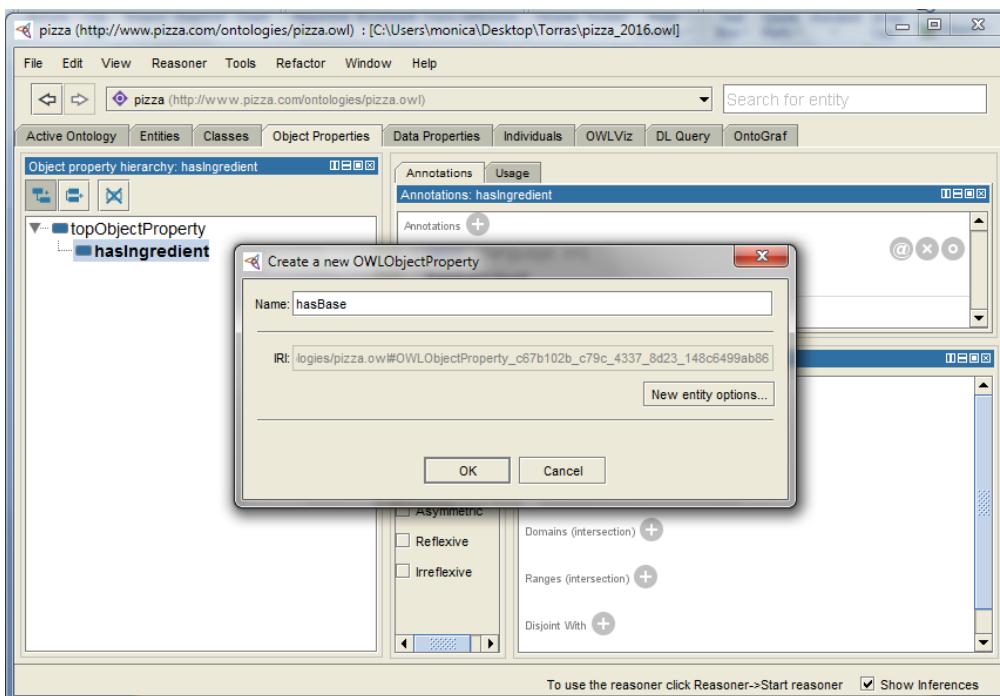
Observație

- Proprietățile-legătură coreleză doi indivizi în timp ce proprietățile-atribut descriu un individ.

Tema 9: Crearea subproprietătilor-legătură (hasBase și hasTopping) a unei proprietăți-legătură (hasIngredient)

se selectează supraproprietatea (aici **hasIngredient**) → clik pe butonul de comandă Add subproperty ⇒ fd Create a new OWLObjectProperty → se tastează numele subproprietății (aici: **hasBase**) → OK

Idem pt **hasTopping**.



Alte exemple de ierarhii de proprietăți: **esteUnchiPentru**, **esteMatusaPentru**, **esteVărul** sunt subproprietăți ale proprietății **esteRudăPentru**.

Observație

1. și proprietățile-atribut admit subproprietăți.
2. Tipurile proprietăților nu se pot amesteca: pentru proprietăți-legătură putem crea numai subproprietăți de tip legătură nu și subproprietăți-atribut.

Tema 10: Crearea proprietăților inverse

se creează proprietatea-legătură **isIngredientOf** ca mai sus →

→ panelul *Property Description* (aici *Description: isIngredientOf*) →

→ directiva *Inverse Of* → clik iconul *Add* ⇒

⇒ fd cu numele proprietății selectate și care conține ierarhia de proprietăți-legătură →

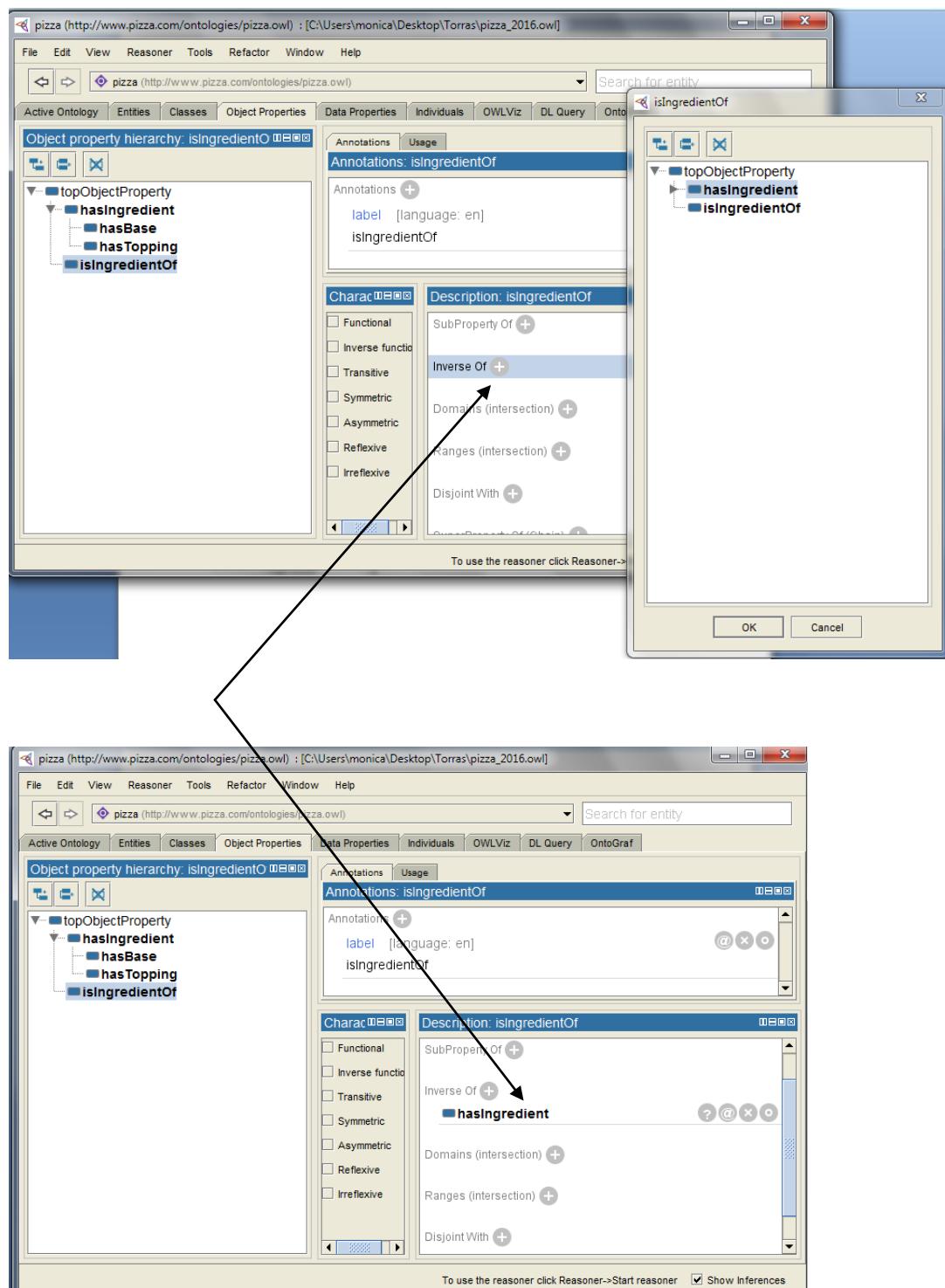
→ se selectează proprietatea-legătură care trebuie declarată ca inversă a celei selectate (aici: **hasIngredient**) → OK ⇒

⇒ cele 2 proprietăți apar ca inverse una alteia în panelul *Description: Property*

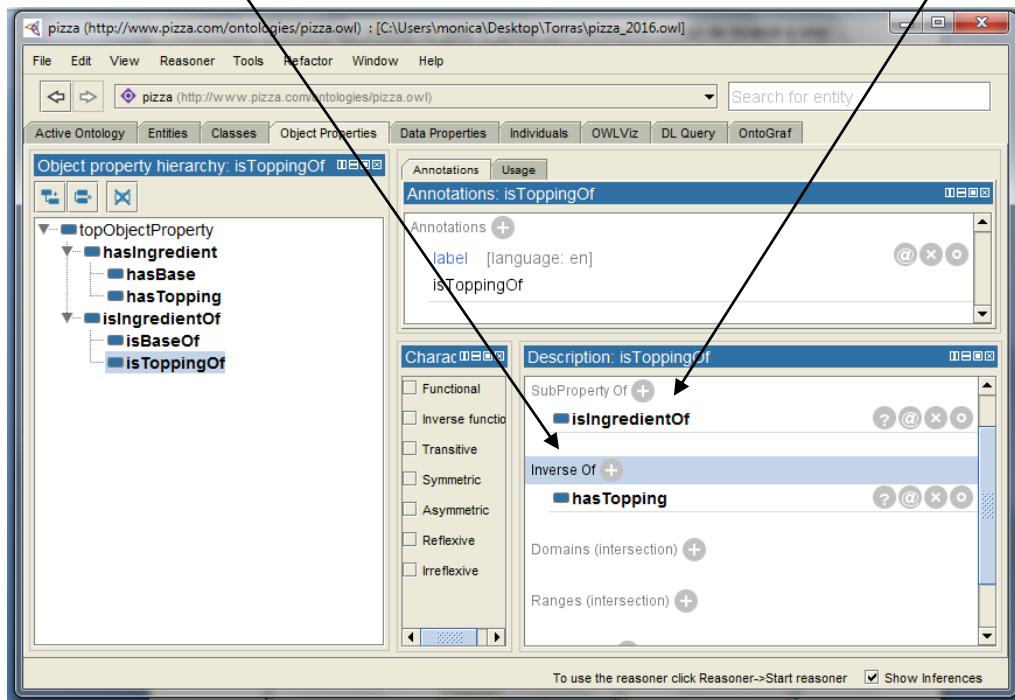
Observație

O proprietate-legătură inversă nu poate fi creată prin declararea calității de inversă a unei proprietăți deja create: trebuie întâi creată ea ca proprietate-legătură și apoi specificată proprietatea pe care ea este inversă .

Editorul Protégé pentru crearea ontologiilor



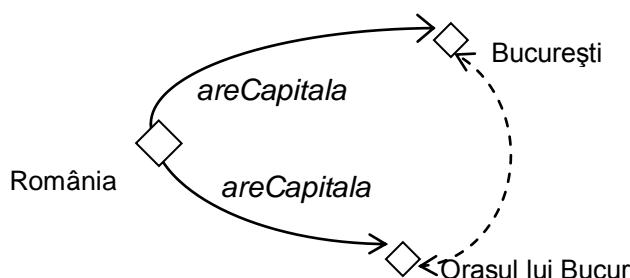
⇒ ierarhia de proprietăți-legătură arată astfel: (a se vedea indicarea supraproprietăților și a proprietăților inverse).



Stabilirea caracteristicilor proprietăților OWL

Proprietăți funcționale

Se mai numesc și proprietăți cu valori unice sau caracteristici. De exemplu:

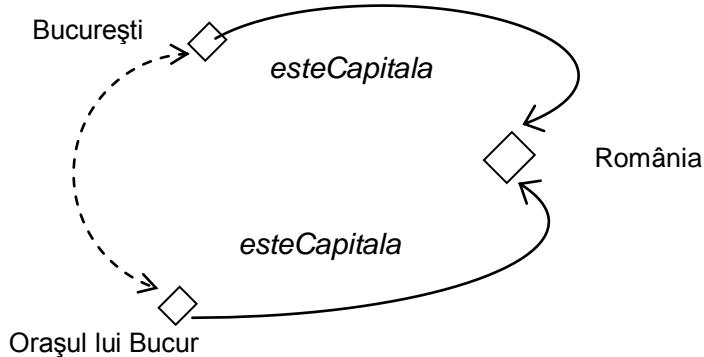


Observație 5

Întrucât o țară nu poate avea decât o singură capitală, din reprezentarea grafică de mai sus (și din traducerea ei în **OWL DL**) rezultă că București și 'Orasul lui Bucur' sunt unul și același. Pe de altă parte, dacă în cadrul ontologiei s-ar fi specificat faptul că 'București' și 'Orasul lui Bucur' sunt două entități distincte (de exemplu, cu elementul `owl:differentFrom`) atunci am fi avut de a face cu o inconsistență a ontologiei (chiar *reasoner-ul* ar fi semnalat-o).

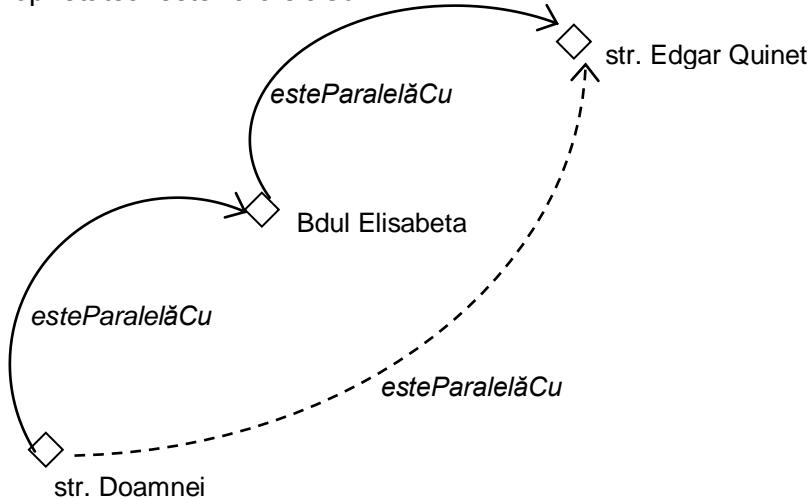
Proprietăți invers funcționale

Proprietatea-legătură $P(x, y)$ este o proprietate invers funcțională dacă ea este inversa proprietății-legătură $Q(y, x)$ iar $Q(y, x)$ este funcțională. De exemplu: dacă afirmăm că ‘București’ este capitala României și că ‘orașul lui Bucur’ este de asemenea capitala României, putem deduce că este vorba despre același oraș.



Proprietăți tranzitive

Fie proprietatea ‘esteParalelăCu’.

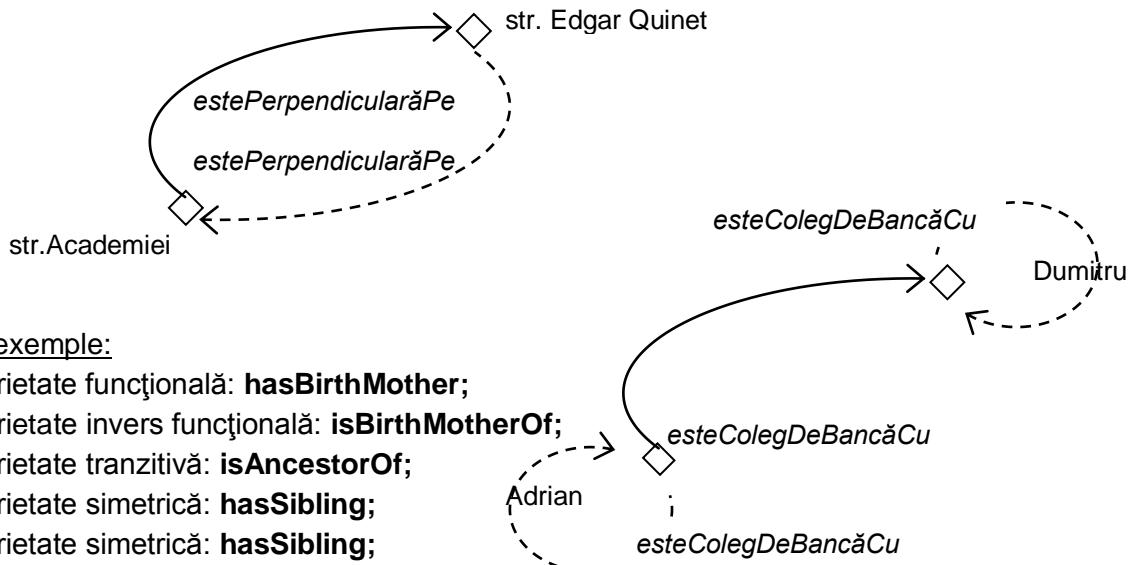


Observații:

Amintim 2 teoreme utile aici:

- fie P o proprietate tranzitivă și fie Q inversa ei $\Rightarrow Q$ este tranzitivă;
- fie P o proprietate tranzitivă $\Rightarrow P$ nu poate fi funcțională (i.e. injectivă: are mai multe imagini pentru același argument).

Proprietăți simetrice



Alte exemple:

Proprietate funcțională: **hasBirthMother**;

Proprietate invers funcțională: **isBirthMotherOf**;

Proprietate tranzitivă: **isAncestorOf**;

Proprietate simetrică: **hasSibling**;

Proprietate simetrică: **hasSibling**;

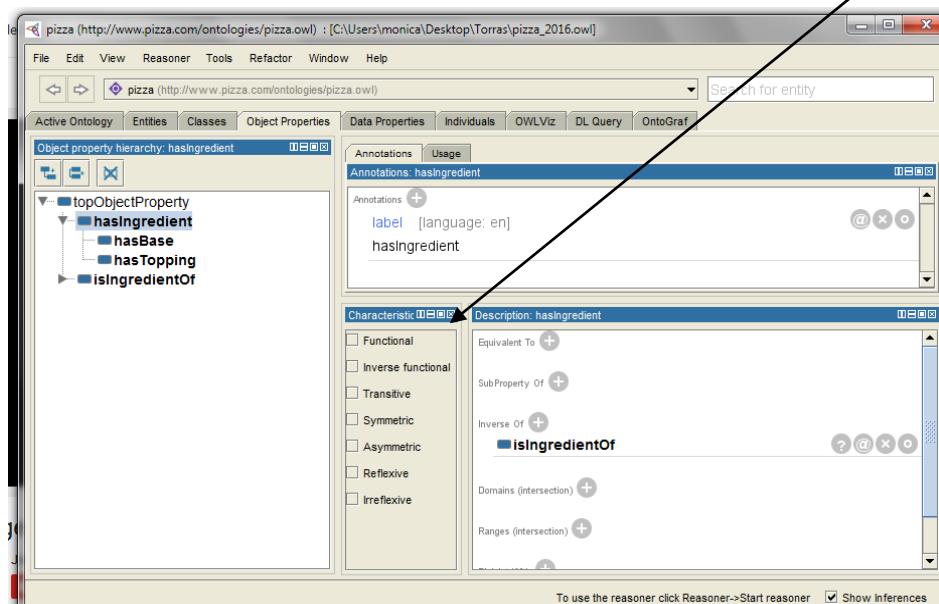
Proprietate asimetrică: **hasChild**; **isChildOf**

Proprietate reflexivă: **knows**; **esteColegDeBancăCu**;

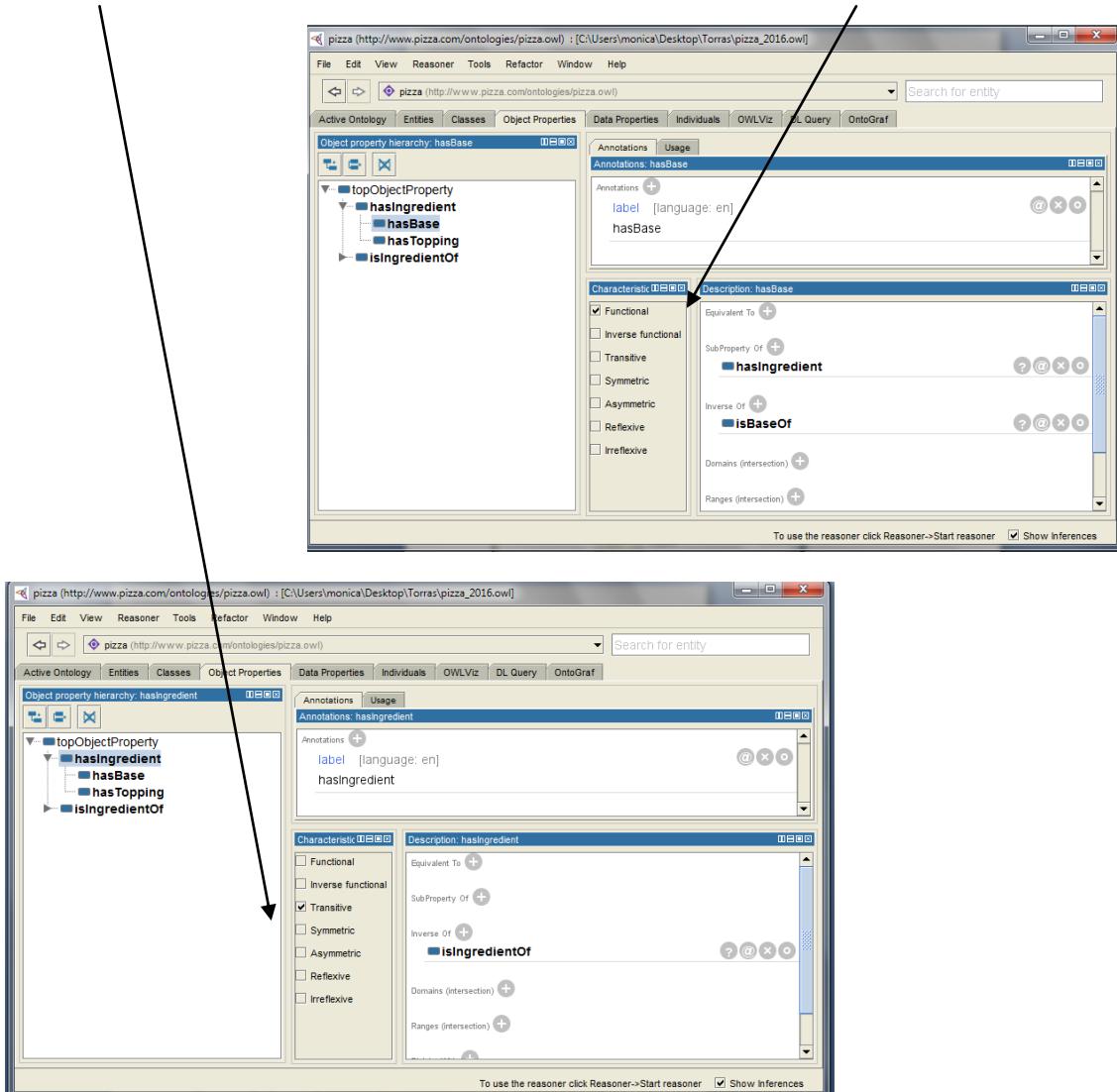
Proprietate nereflexivă: **motherOf**.

Stabilirea caracteristicilor proprietăților

Se realizează prin selectarea casetelor de opțiune din panelul *Characteristics*, după selectarea proprietății respective.



Tema 11, 12: Să se stabilească tranzitivitatea proprietății *hasIngredient* (precum și a *isIngredientOf* !!) și injectivitatea proprietății *hasBase*.



Observații

1. Dacă P1 și P2 sunt inverse una alteia și dacă P1 a fost definită ca tranzitivă Protégé nu stabilește automat că și P2 este tranzitivă (cf. teoremei). **Acest lucru trebuie făcut manual.** Totuși, reasonnerul NU va semnala eroare dacă nu se setează manual P2 ca tranzitivă
2. **OWL DL** nu acceptă proprietăți-atribut tranzitive sau simetrice; ca urmare numai caseta de opțiune *Functional* este vizibilă.

Stabilirea domeniilor și codomeniilor pentru proprietățile OWL

În **OWL** stabilirea domeniilor și codomeniilor unei proprietăți nu trebuie privită ca o restricție ce trebuie verificată ci ca un set de axiome ce pot fi folosite pentru efectuarea automată a raționamentelor. Să presupunem, de exemplu, că:

- pe lângă clasa **Pizza** am definit și clasa **IceCream**;
- definim clasa **Pizza** ca domeniu de definiție al proprietății **hasTopping**;
- aplicăm proprietatea **hasTopping** clasei **IceCream**.

Urmarea: NU se generează o eroare ci se conchide că **IceCream** este o subclasă a clasei **Pizza**!! Se generează o eroare (de către un *reasoner*) numai dacă am stabilit în prealabil că **IceCream** și **Pizza** sunt clase disjuncte.

Observație

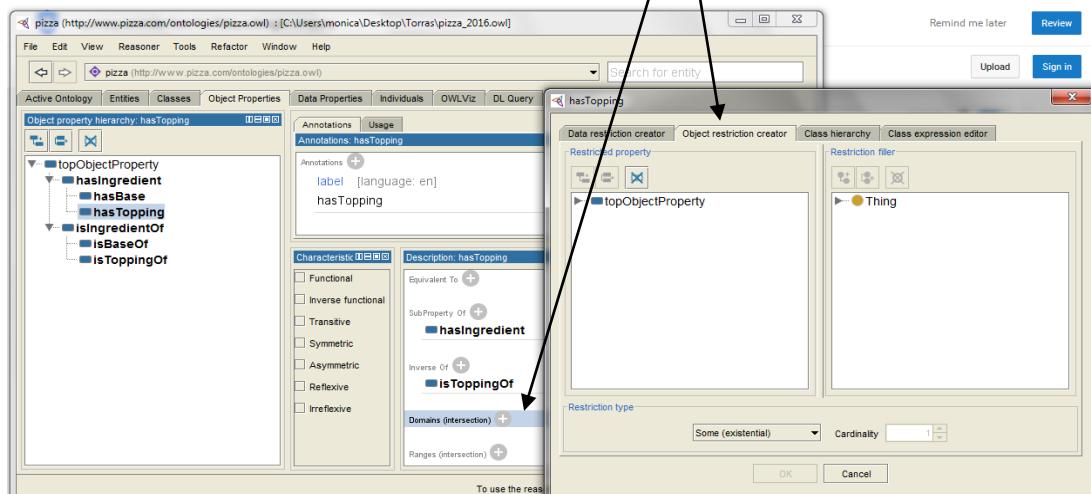
În versiunile anterioare ale **Protégé 3.1** stabilirea domeniilor și codomeniilor proprietăților se putea realiza atât la nivel de individ (instantă) cât și la nivel de clasă (prin selectarea corespunzătoare a opțiunilor grafice din interfață). Specificarea la nivel de clasă nu este recomandată (uneori este chiar considerată o greșală!) deoarece în acest fel clasa este tratată ea-însăși ca un individ, ori acesta este un tip de ‘meta-asserțiune’ care face ca ontologia să nu mai fie în **OWL DL** ci în **OWL FULL**. Motivul: se consideră că instanțele proprietăților-legătură coreleză instanțe ale unei clase cu instanțe ale altrei clase. Incepând cu v. 3.1 nu se mai face însă această distincție.

Tema 13, 14: Stabilirea domeniului și codomeniului proprietății hasTopping

se selectează proprietatea (aici: **hasTopping**) → directiva *Domains (intersection)* →

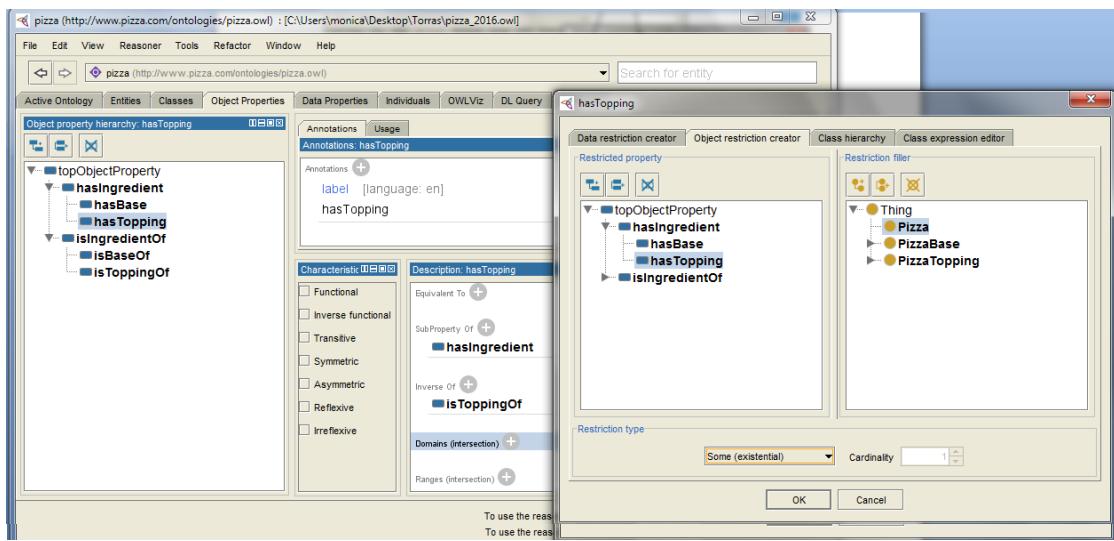
→ clik pe iconul *Add* ⇒

⇒ fd → se selectează tabul *Object Restriction Creator* →

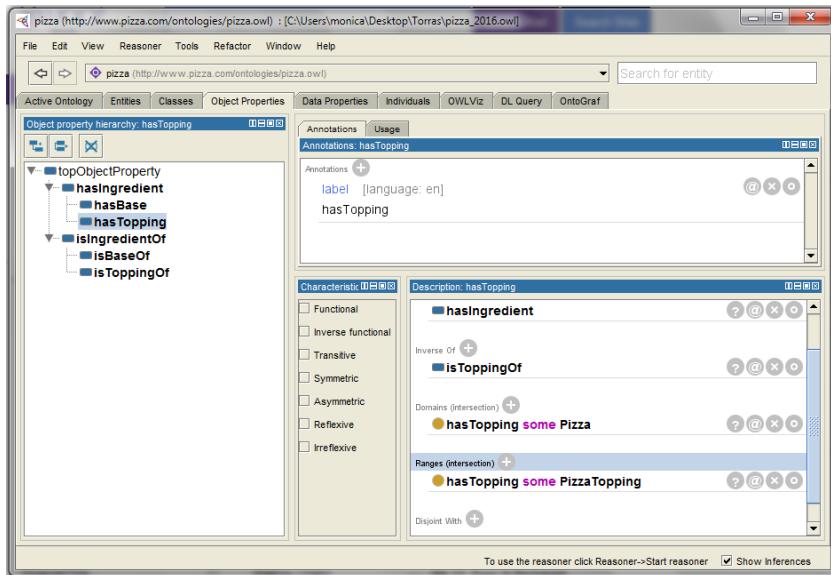


→ se expandează ierarhia de proprietăți și ierarhia de clase și se aleg proprietatea și clasa dorite → OK

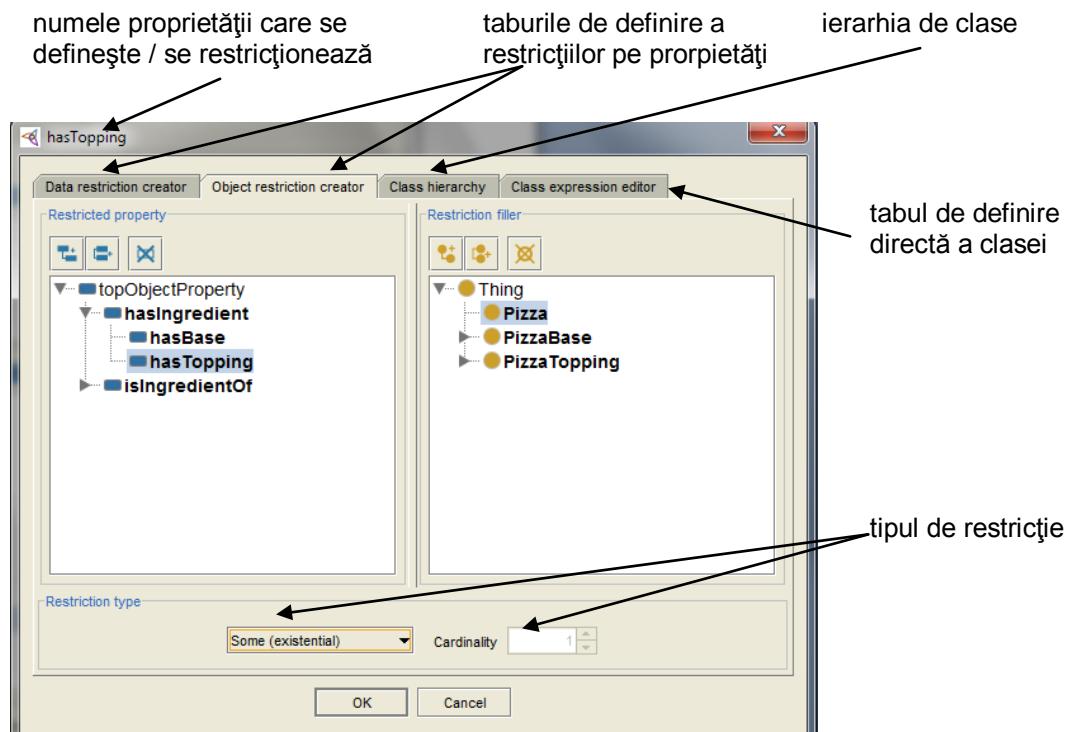
Editorul Protégé pentru crearea ontologiilor



Analog pt codomeniul proprietății **hasTopping** ⇒

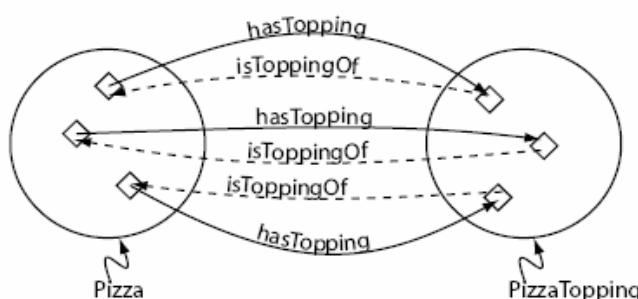


Fereastra-dialog de definire a claselor, afişată la click pe orice icon Add (vom reveni):



Observații

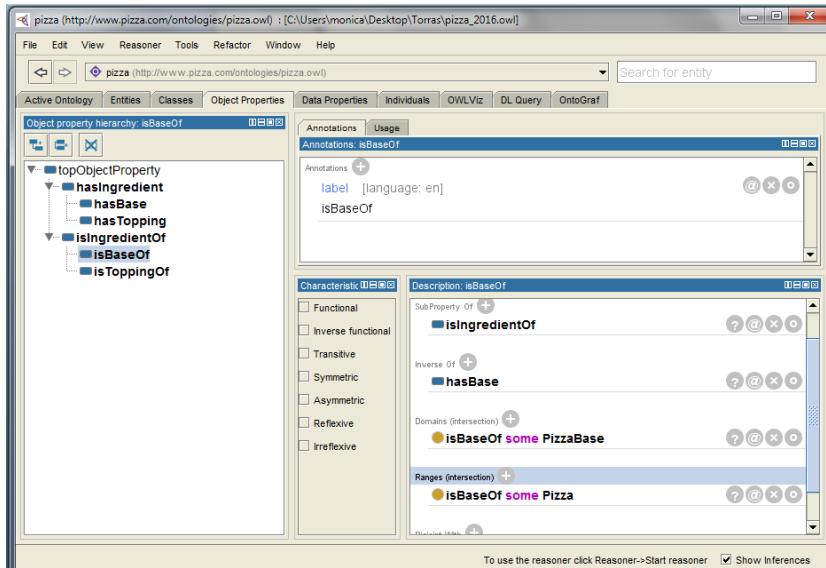
- Ca urmare a acestor specificări, orice nume care apare ‘în membrul stâng/drept’ al proprietății `hasTopping` va fi considerat ca fiind numele unei instanțe a clasei `Pizza/PizzaTopping` (chiar dacă acest lucru nu fusese deja specificat în mod explicit).
- În versiunea 3.1 a **Protégé**, specificarea domeniului și/sau a codomeniului unei proprietăți care a fost anterior definită ca inversabilă atrage automat după sine definirea corespunzătoare a domeniului și codomeniului proprietății inverse (aici: ca



urmare a rezolvării Temei 7, clasele `PizzaTopping` și respectiv `Pizza` au fost automat definite ca domeniu și respectiv codomeniu ale proprietății `isToppingOf` și apar în fereastra-dialog atunci când se selectaează proprietatea `isToppingOf`. În

versiunile anterioare ca și în ***Protégé 4.3*** !!! acest lucru trebuie realizat explicit de către utilizator.

Tema 15: Să se definească domeniul și codomeniul proprietăților *hasBase* și *isBaseOf*.



Observații 9

Reamintim: în general, specificarea domeniilor și codomeniilor proprietăților NU este recomandabilă: faptul că acestea nu funcționează ca restricții ci ca ‘axiome’ pentru fundamentarea raționamentelor și că pot genera clasificări ‘derutante’ poate avea efecte secundare neașteptate și poate conduce la erori dificil de depistat (mai ales într-o ontologie de mari dimensiuni!).

Observație

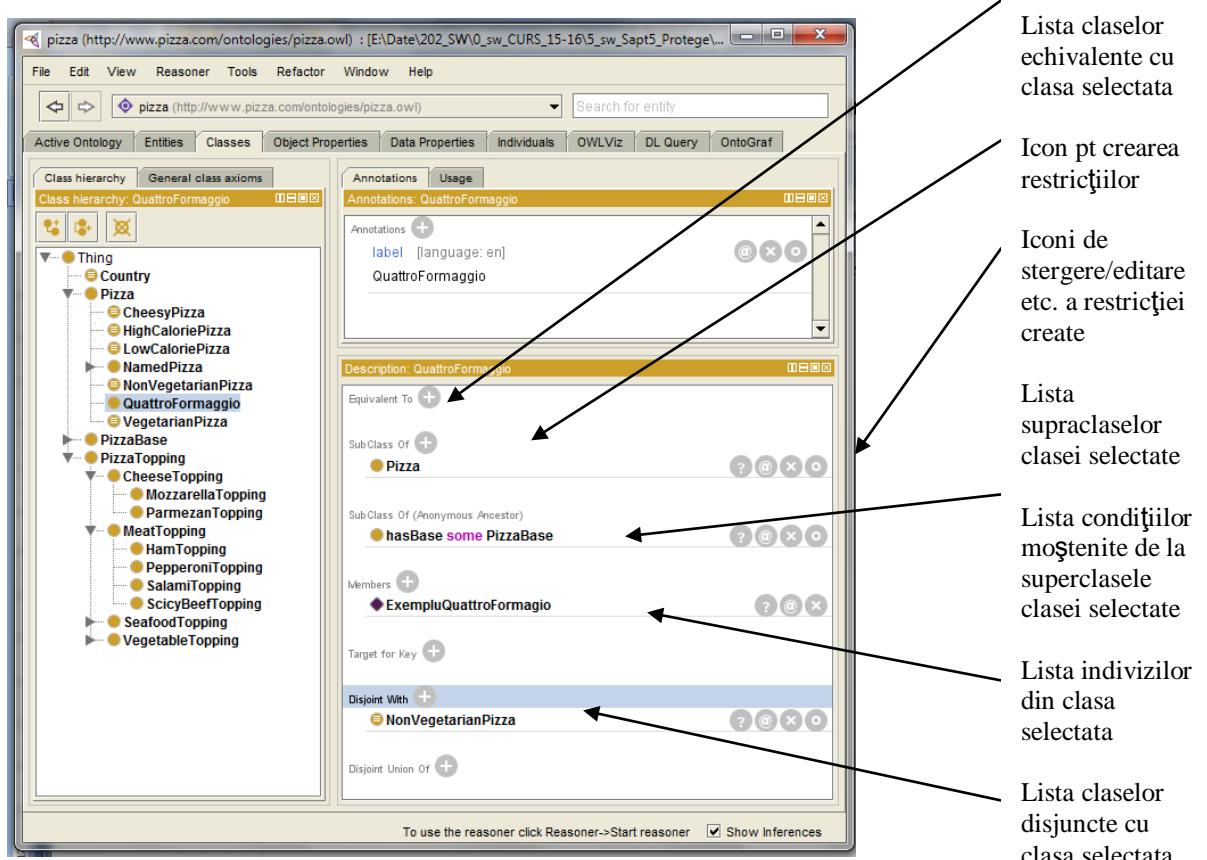
Reamintim: în **OWL**, proprietățile descriu relații binare:

- proprietățile-legătură (*Object properties*) descriu relații între 2 indivizi;
- proprietățile-atribut (*Datatype properties*) descriu relații între indivizi și valori ale caracteristicilor lor.

Proprietățile-legătură și proprietățile-atribut din **OWL** servesc la crearea restricțiilor; acestea sunt aplicate indivizilor în scopul descrierii apartenenței lor la o anumită clasă (i.e.: o restricție descrie o clasă de indivizi cu ajutorul relațiilor la care participă indivizii). Se creează astfel supraclase/subclase/clase echivalente clasei respective (eventual anonime). Altfel spus: o clasă de indivizi este descrisă/definită prin relațiile la care participă indivizii săi ⇒ o clasă poate fi:

- o clasă creată ca atare (cu directiva `owl: class [owl: subclass of]`; în *Protégé* este o “named class”)
- o restricție a unei proprietăți.

Restrictiile în *Protégé* sunt create și modificate cu ajutorul iconilor din panelul *Description* din tabul *Classes* și al taburilor din fereastra-dialog astfel afișată:



The screenshot illustrates several methods for creating class restrictions in the Protégé editor:

- Iconul Add subclass**: Points to the tree view on the left where a new subclass of **Pizza** named **QuattroFormaggio** is being created.
- Tabul pt crearea de restricții asupra proprietății-atribut selectate**: Points to the "Data restriction creator" tab of the dialog for **QuattroFormaggio**, which shows a restriction on the **hasBase** property.
- Tabul pt crearea de restricții asupra proprietății-legătură selectate**: Points to the "Object restriction creator" tab of the same dialog, showing a restriction on the **hasTopping** property.
- Alegerea clasei**: Points to the "Restriction filler" panel on the right, which lists various classes like **Thing**, **Country**, **Pizza**, and **PizzaBase**.
- Definirea unui tip de restricție (de cuantificare de**: Points to the "Restriction type" section of the dialog, which specifies "Some (existential)" as the quantifier.
- Tabul pt crearea de restricții prin tastarea codului**: Points to the "Class expression editor" tab of the dialog, showing the query **hasBase some PizzaBase**.

Restricțiile **OWL** se clasifică în:

- restricții de cuantificare;
 - de tip existențial (numite și restricții de tip `somevaluesFrom`)
 - de tip universal (numite și restricții de tip `allvaluesFrom`)
- restricții de cardinalitate;
- restricții de tip `hasValue`.

Restricții de cuantificare

Definiție. O restricție de cuantificare se compune din 3 părți:

1. un cuantificator (universal sau existențial);
2. o proprietate-legătură;
3. un argument (*filler*).

Restricții de cuantificare de tip existențial

Distingem două tipuri de restricții existențiale:

- r.e. care definesc condiții necesare pt ca un individ să aparțină unei clase: clasele astfel definite se numesc clase primitive sau partial definite (un individ care îndeplinește acest tip de condiții poate face parte din clasă);
- r.e. care definesc condiții necesare și suficiente pt ca un individ să aparțină unei clase: clasele astfel definite se numesc clase derivate sau complet definite (un individ care îndeplinește acest tip de condiții trebuie să facă parte din clasă).

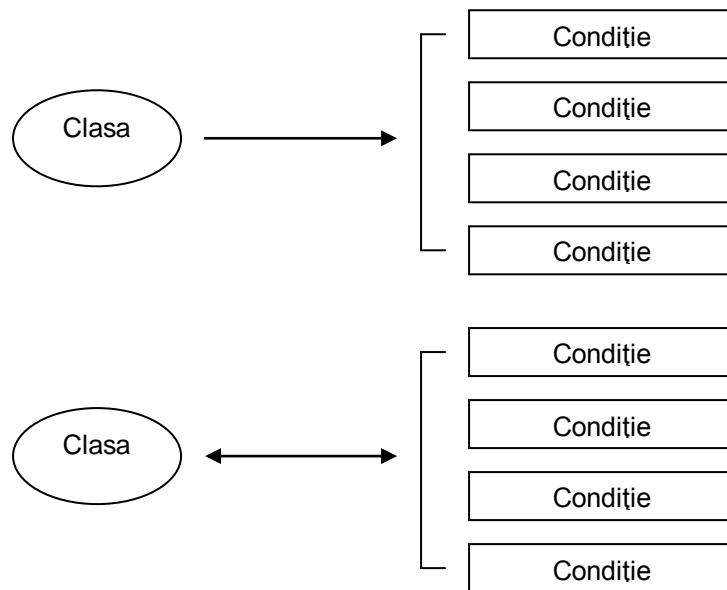
Clasa C se numește primitivă (parțial definită) dacă este caracterizată numai de condiții necesare \Rightarrow

dacă o instanță **a** se află într-o clasă primitivă **C**, atunci știm că **a** are toate caracteristicile clasei **C** dar faptul că **a** are caracteristicile lui **C** nu ne permite să afirmăm că **a ∈ C**.

Clasa C se numește derivată (bine definită) dacă este caracterizată de condiții necesare și suficiente \Leftrightarrow

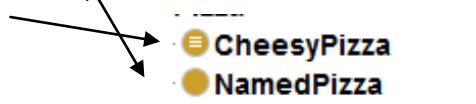
dacă o instanță **a** se află într-o clasă bine definită **C**, atunci știm că **a** are toate caracteristicile clasei **C** și reciproc, știm că dacă **a** are caracteristicile lui **C** atunci **a ∈ C**.

Reprezentarea grafică a condițiilor necesare, respectiv necesare și suficiente este cea de mai jos:



Reprezentarea grafică a claselor în *Protégé*:

- clase primitive (parțial definite)
- clase derivate (complet definite)



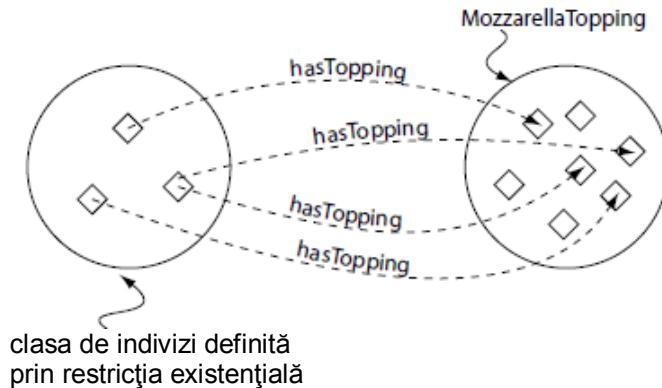
În *Protégé*,

- condițiile necesare \Leftrightarrow superclase
- condițiile necesare și suficiente \Leftrightarrow clase echivalente

Restricțiile existențiale descriu clase de indivizi care au cel puțin o corelație cu indivizii unei anumite clase în raport cu o anumită proprietate-legătură. De exemplu: clasa indivizilor care au (cel puțin) o corelație hasTopping cu membrii clasei MozzarellaTopping (elementul folosit în **OWL** este owl:someValuesFrom; cuvântul-cheie folosit în **Protégé 4** este **some**.).

Restricțiile universale descriu clase de indivizi care, dată fiind o proprietate-legătură, au relații numai în raport cu acea proprietate-legătură cu indivizi care fac parte dintr-o anumită clasă. De exemplu: clasa de indivizi care au numai relații hasTopping cu membri ai clasei VegetableTopping. (elementul folosit în **OWL** este owl:allValuesFrom; cuvântul-cheie folosit în **Protégé 4.3** este **only**.).

Restricțiile existențiale sunt de departe restricțiile cel mai frecvent întâlnite în ontologiile **OWL**. Dacă examinăm din nou restricția `hasTopping some MozzarellaTopping`, observăm că este o restricție de tip **existențial** (se utilizează clauza **some**) definită pe proprietatea-legătură `hasTopping` property având ca argument clasa `MozzarellaTopping`. Putem spune că ea descrie o clasă de indivizi care au cel puțin o corelație `hasTopping` cu un individ din clasa `MozzarellaTopping`. Altfel spus: au cel puțin un topping care este făcut din Mozzarella.



Prin urmare, o restricție existențială descrie o clasă anonimă compusă din toți indivizii care satisfac respectiva restricție, adică au proprietățile necesare pentru a fi membri ai acesteia. Deși aplicate proprietăților, restricțiile definesc clase anonime: ele constau din acei indivizi care satisfac restricția impusă. Atunci când o restricție este folosită pentru a descrie o clasă este definită de fapt o supraclasă a clasei respective (aici: este definită clasa tuturor ‘lucrurilor’ care au cel puțin un ‘topping’ de Mozzarella).

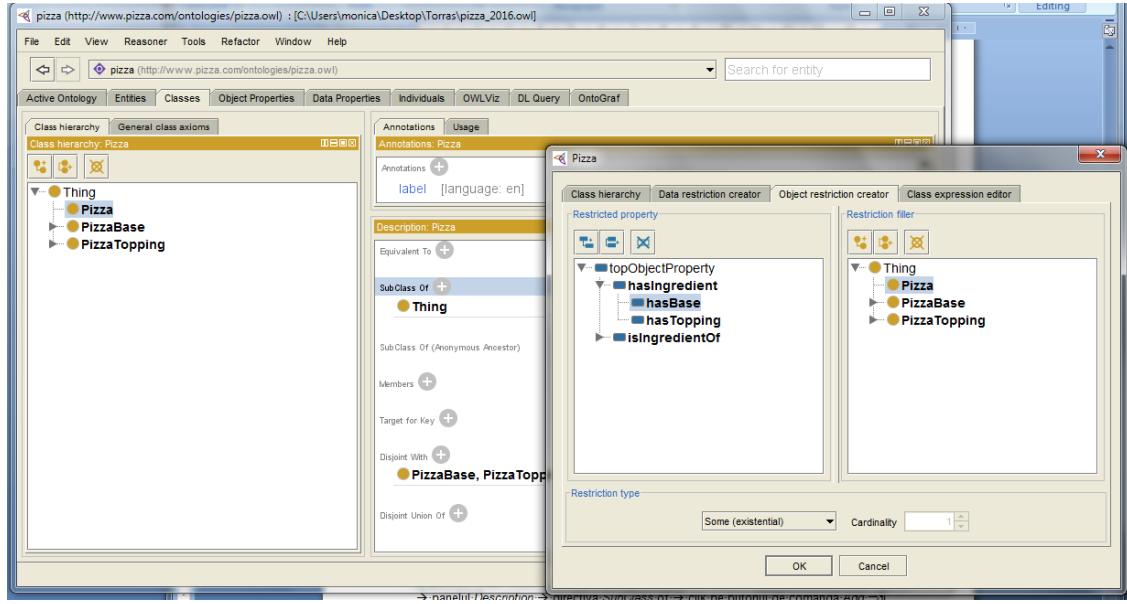
Ne ocupăm mai întâi de clasele primitive, deci de condițiile necesare

Crearea claselor primitive (partial definite)

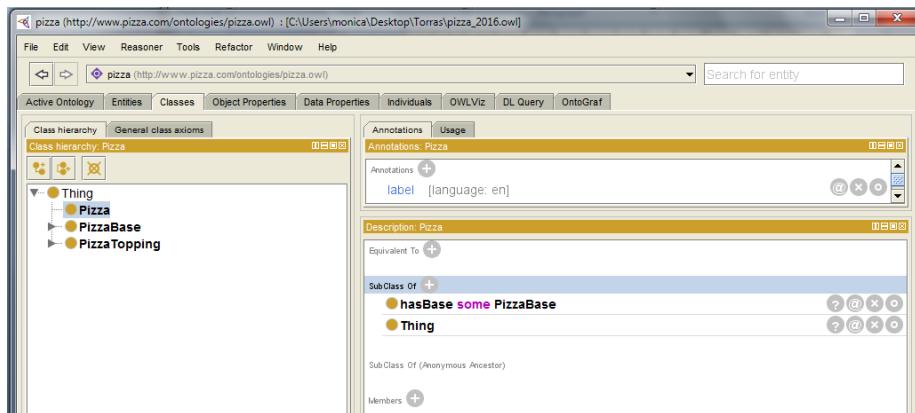
Se face cu ajutorul unei restricții de cuantificare de tip existențial , prin definirea (unui set de) condiții necesare.

Tema 16: Să se definească o clasă de pizza care să specifice faptul că: “o pizza este ceva care trebuie să aibă o <bază>”

Intrucât o restricție existențială acționează asupra unei proprietăți **P** și definește o superclasă (anonimă) a clasei **C** care este domeniu de definiție pt **P**, procedăm astfel:



în tabul *Classes* se selectează clasa **C** (aici: *Pizza*) → panelul *Description* → directiva *SubClass of* → iconul *Add* ⇒
 ⇒ fd → tabul *Object restriction creator* → se expandează ierarhiile de proprietăți și clase → se alege proprietatea **P** (aici *hasBase*) și clasa codomeniu a acesteia (aici: *PizzaBase*) →
 → se alege cuantificatorul existențial (aici: *some*, care este, de altfel, implicit) → OK
 ⇒ în fd *Description* a apărut, alături de *Thing*, noua superclasă a clasei *Pizza*.

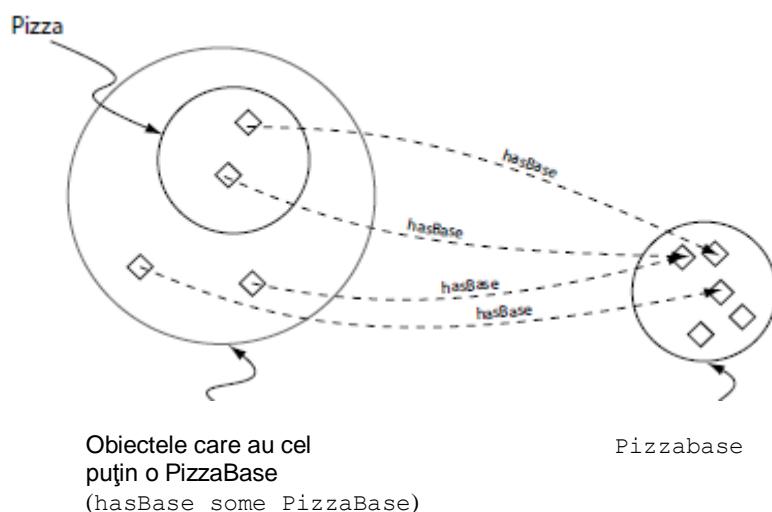


Observație

Am definit clasa **Pizza** ca fiind o subclasă a clasei **Thing** precum și o subclasă a clasei tuturor obiectelor care au ca bază un tip oarecare de **PizzaBase**. Toate acestea sunt însă numai condiții necesare: pentru ca un obiect să fie o **Pizza**, el trebuie:

- ✓ să fie membru al clasei **Thing** (în **OWL**, orice obiect este membru al clasei **Thing**),
- ✓ să aibă o **PizzaBase** oarecare

Reluând observația de mai sus: condiția necesară ca un obiect să fie o **Pizza** este ca acesta să aibă o corelație cu un membru din clasa **PizzaBase** pe baza proprietății **hasBase** (cf. figură de mai jos).



Crearea unor rețete speciale de pizza

Vom îmbogăți ierarhia cu câteva rețete speciale de pizza: Margherita, Americana, Soho etc. Pentru bună ordine, le vom grupa într-o clasă specială: **NamedPizza**.

Tema 18: Să se creeze o subclasă a clasei Pizza, numită NamedPizza și o subclasă a ei, numită MargheritaPizza

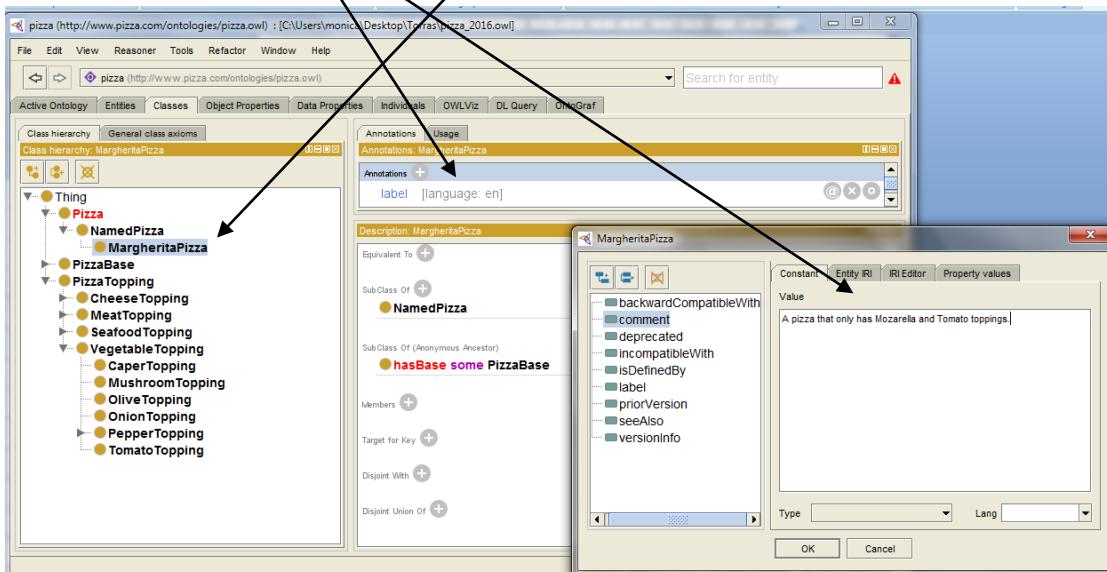
Se procedează astfel:

panelul Classes → se selectează clasa Pizza → click pe iconul 1: Create subclass

⇒ în fd se tastează numele: NamedPizza → OK →

→ analog se creează subclasa MargheritaPizza a clasei NamedPizza →

→ se introduce un comentariu pt MargheritaPizza: Margherita este o pizza care are 2 topping-uri: mozzarella și roșii



Tema 19: Definim MargheritaPizza cf. comentariului de mai sus

⇒ vom crea 2 restricții existențiale asupra proprietății hasTopping cu valorile MozzarellaTopping și TomatoTopping

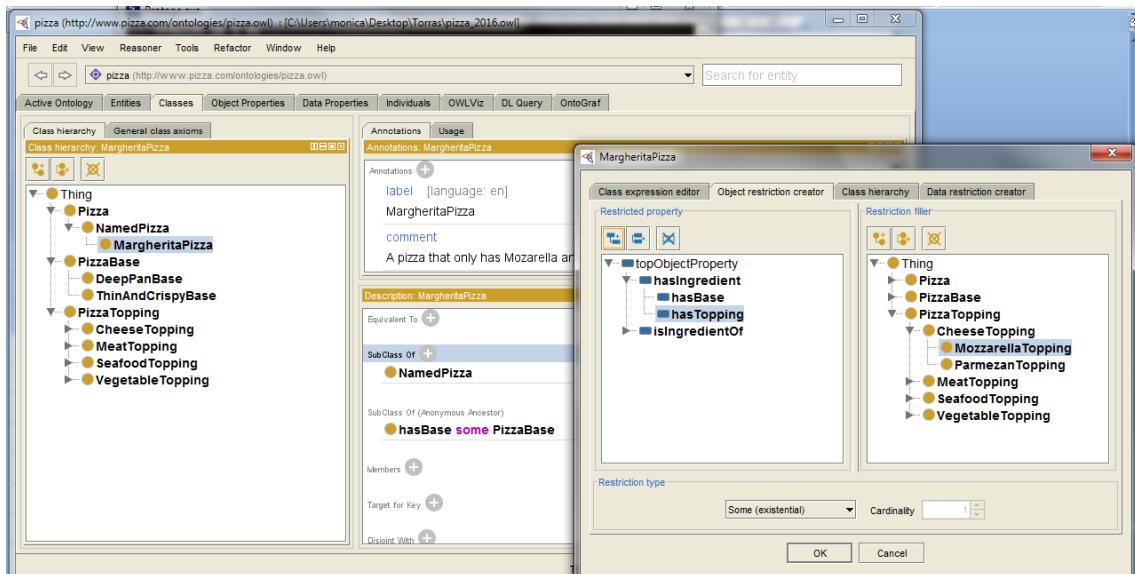
tabul Classes → în panelul Class Hierarchy se selectează MargheritaPizza

→ panelul Description → directiva SubClass Of → iconul Add ⇒ fd

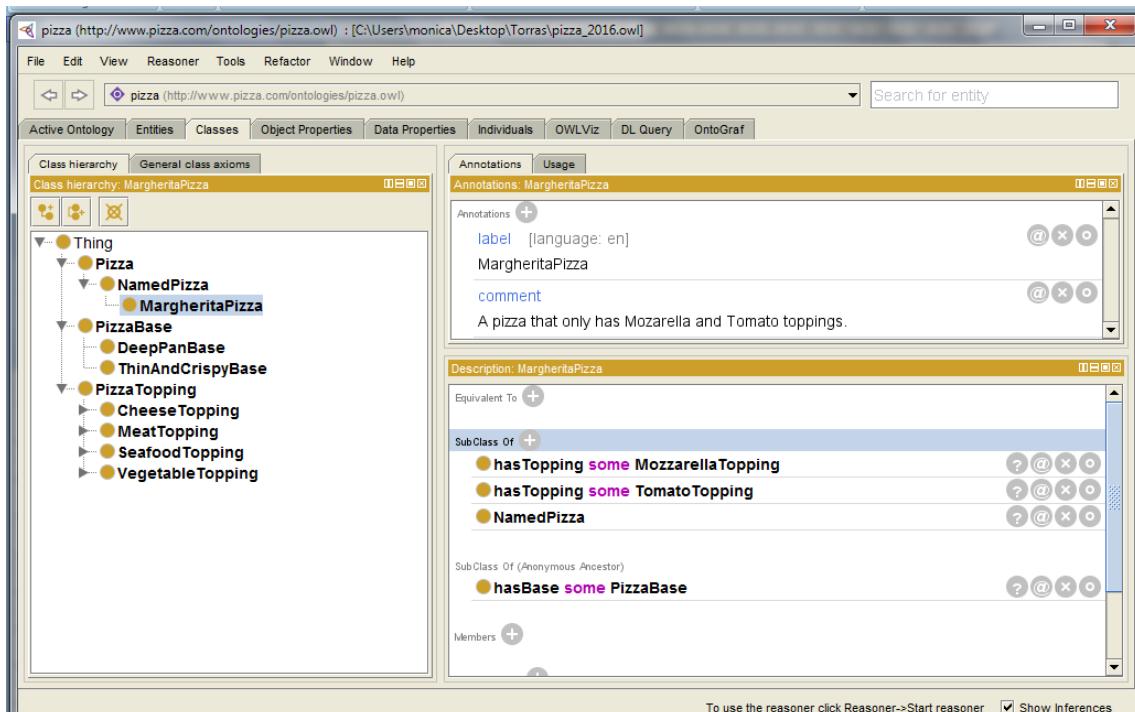
→ tabul Object restriction creator → se aleg proprietatea de restrictionat (aici: hasTopping) și valoarea restricției (aici: MozzarellaTopping) → OK

→ analog pt al doilea topping → implicit operatorul logic dintre cele 2 restricții este AND

Editorul Protégé pentru crearea ontologiilor



⇒



Observație

Am atașat clasei MargheritaPizza două restricții pentru a specifica faptul ca o MargheritaPizza este o NamedPizza care are cel puțin un MozzarellaTopping și cel puțin un TomatoTopping. Formalizat (examinând rând cu rând caseta SubClass Of): pentru ca un obiect să fie membru al clasei MargheritaPizza, el trebuie:

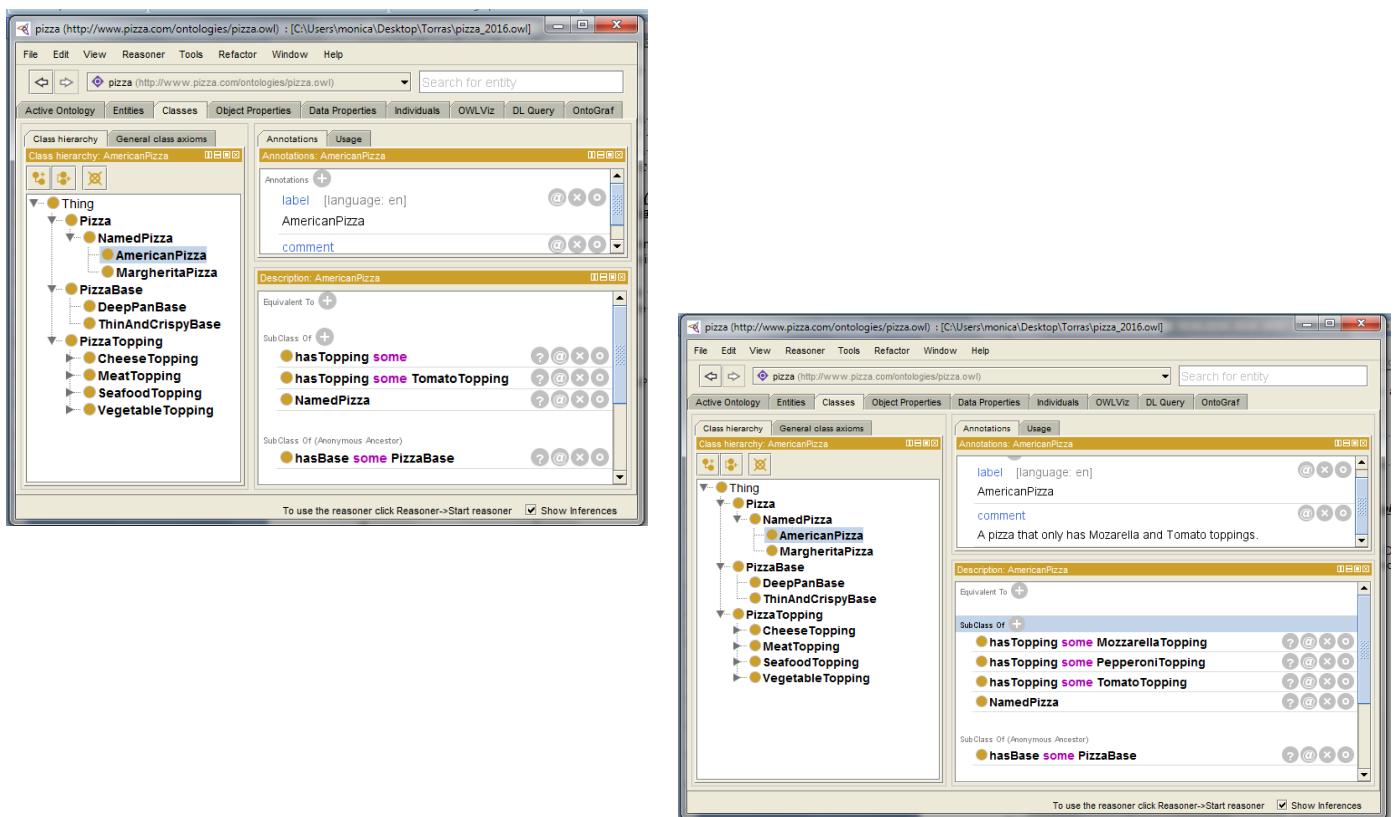
- să fie membru al clasei NamedPizza și

- să fie membru al clasei anonte care este corelată cu cel puțin un membru al clasei MozzarellaTopping prin intermediul proprietății hasTopping și
- să fie membru al clasei anonte care este corelată cu cel puțin un membru al clasei TomatoTopping prin intermediul proprietății hasTopping.

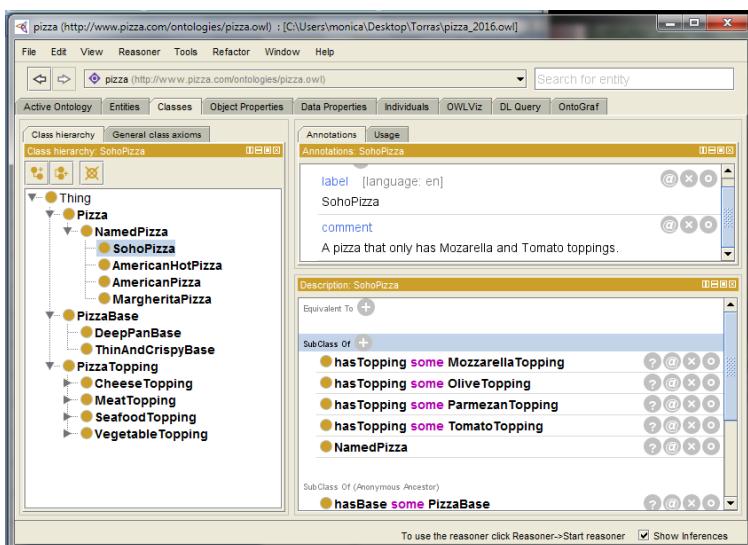
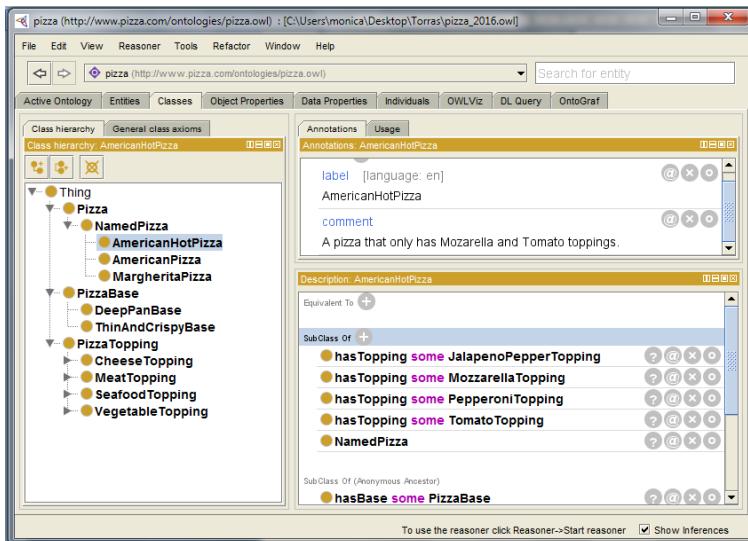
Tema 21: Să se creeze clasa *AmericanPizza* (o pizza care are 3 topping-uri: mozzarella, roșii și pepperoni) prin clonarea și modificarea clasei *MargheritaPizza* cu care seamănă f mult

se selectează clasa MargheritaPizza → meniul *Edit* → comanda *Duplicate selected class* ⇒ fd în care se introduce numele noii clase și se mai pot face și alte setări → OK ⇒ noua ierarhie de clase este:

se adaugă (ca mai sus) restricția necesară (al treilea topping, cel cu pepperoni)



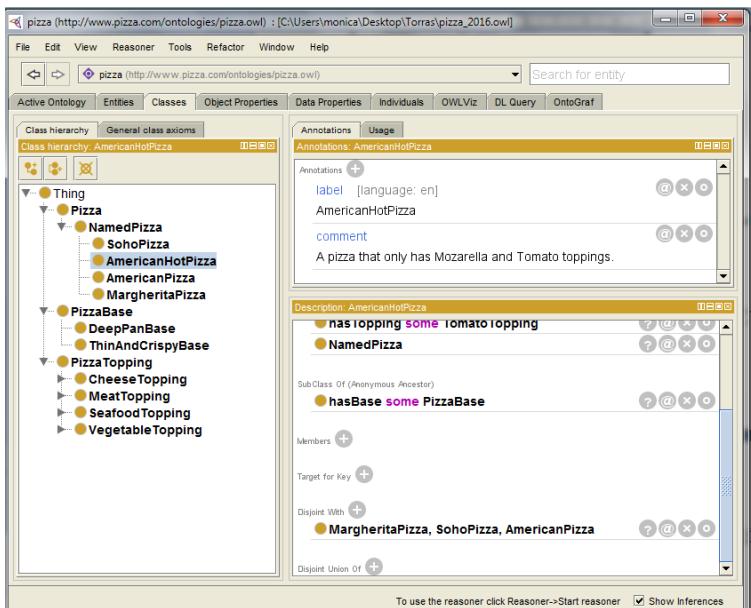
Tema 22: Să se creeze clasa *AmericanHotPizza* (o *AmericanPizza* care are un al 4-lea topping: Jalapeno peppers) și o *SohoPizza* (o *MargheritaPizza* care are încă 2 toppingsuri: măslini și parmezan) tot prin clonare



Tema 23: Toate subclasele clasei *NamedPizza* sunt disjuncte

Intrucât sunt 4 subclase procedăm astfel:

se selectează una dintre subclase (ex.: *MargheritaPizza*) → meniul *Edit* → comanda *Make primitive siblings disjoint* ⇒



Utilizarea Reasonner-ului

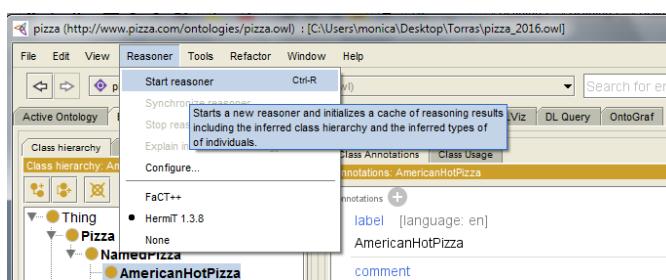
Avantaj al ontologiilor bazate pe **OWL DL**:

pot fi procesate cu un *reasonner*.

Un *reasonner* poate face verificări de:

- clasificare (verifică dacă o clasă este o subclasă a alteia) => *reasonner-ul* furnizează o ierarhie calculată (*inferred ontology class hierarchy* vs. *asserted hierarchy*);
- consistență (verifică dacă o clasă are cel puțin o instanță)

Protege 4.3 poate incorpora diverse alte aplicații de tip *reasonner* pe lângă Fact++. (și HermiT). Ele se lansează din meniul *Reasonner*.



Ca urmare a testelor făcute de *reasonner*:

- dacă o clasă a fost reclasificată (are acum o altă superclasă) → numele ei apare în culoare albastră;
- dacă o clasă este găsită ca inconsistentă (nu conține instanțe) → numele ei apare în culoare roșie;

Crearea claselor deriveate (complet definite)

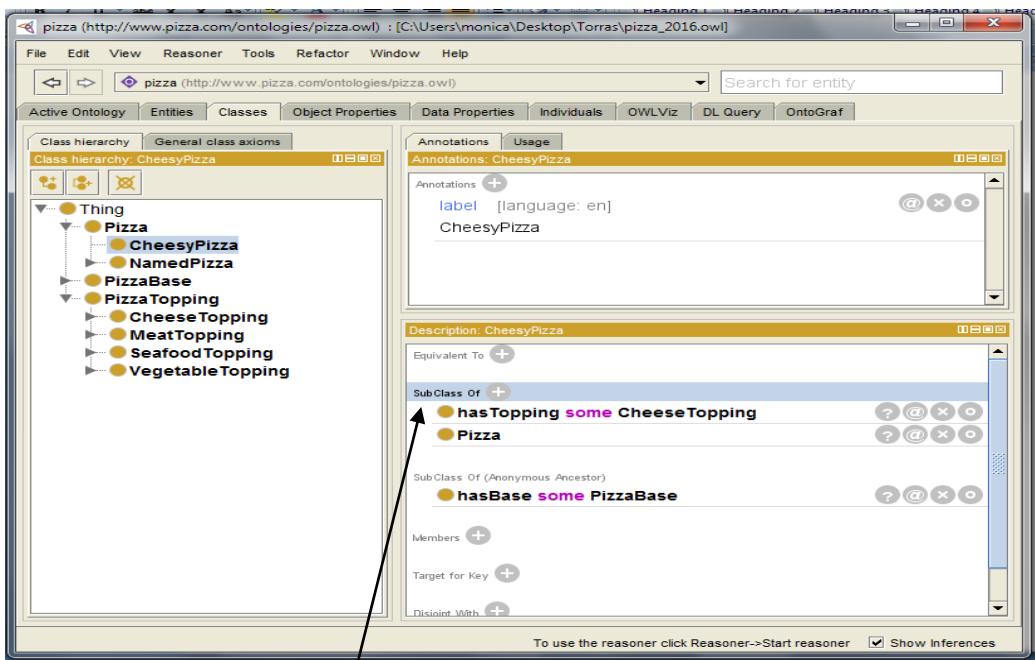
Se face cu ajutorul unei restricții de cuantificare de tip existențial , prin definirea (unui set de) condiții necesare și suficiente.

Transformarea unei condiții necesare într-o clasa definită (i.e. a unei clase primitive (partial definite) în una derivată (complet definită), i.e. a unei superclase a clasei date într-o clasă echivalentă cu aceasta) se face astfel:

se selectează superclasa → meniu *Edit* → comanda *Convert to defined class* ⇒
⇒ **clasa nu mai este doar descrisă ci și definită.**

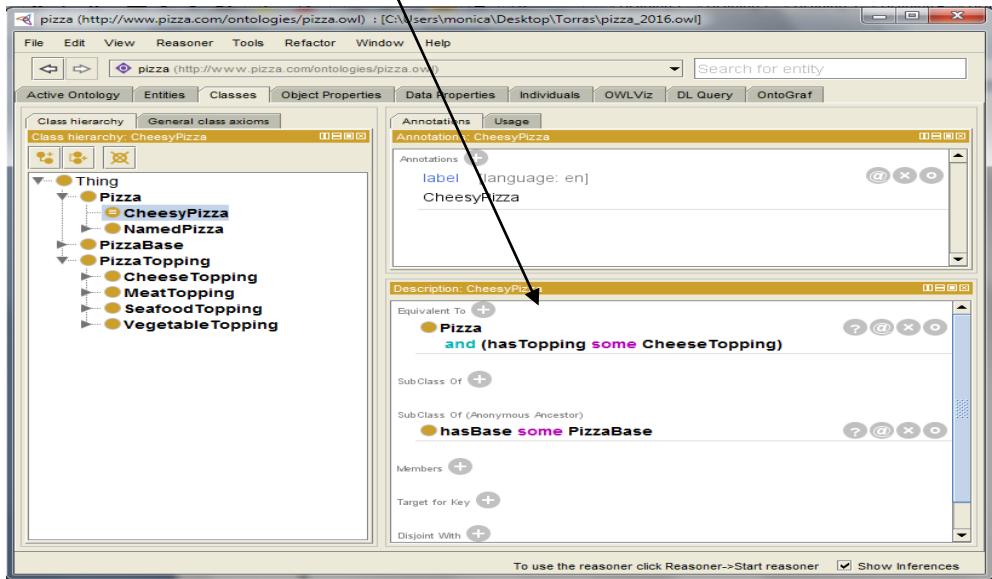
Tema 28: Să se creeze o subclasă a clasei Pizza numită CheesyPizza (i.e. o clasă care are cel puțin un topping de tip CheeseTopping ⇒ o definim prin restricție existențială asupra proprietății hasTopping cu codomeniul CheeseTopping)

tabul *Classes* → panelul *Class hierarchy* → clasa *Pizza* → în acest panel se creează o subclasă a *Pizza* denumită *CheesyPizza* → panelul *Description* → directiva *SubClass Of* → iconul *Add* ⇒ *fd* → panelul *Object restriction creator* → se extind ierarhiile de proprietăți și clase → *OK* ⇒ ierarhia din figura



Tema 29 Să se transforme clasa CheesyPizza dintr-o clasă primitivă într-una bine definită (i.e. să se transforme condițiile necesare pt CheesyPizza în condiții necesare și suficiente)

se selectează clasa CheesyPizza → meniul *Edit* → comanda *Convert to defined class* ⇒



Restricții de cuantificare de tip universal

Restricțiile existențiale nu reprezintă condiții obligatorii de apartenență la o clasă: de exemplu, putem folosi restricția existențială `hasTopping some MozzarellaTopping` pentru a descrie indivizi care au cel puțin o corelație de tip `hasTopping` cu un individ din clasa `MozzarellaTopping` (i.e. indivizi din `Thing` care au cel puțin un topping din clasa `MozzarellaTopping`). Aceasta nu implică însă faptul că toate relațiile de tip `hasTopping` trebuie să implice membri ai clasei `MozzarellaTopping`. (i.e. nu implică faptul că toate toppingurile trebuie să fie din clasa `MozzarellaTopping`).

Pentru a restricina relațiile privitoare la o anumită proprietate-legătură la indivizii dintr-o anumită clasă trebuie să utilizăm restricții universale (în **OWL**: `allValuesFromRestrictions`). De exemplu, restricția universală `hasTopping only MozzarellaTopping` ($\forall \text{ hasTopping } \text{ MozzarellaTopping}$) descrie indivizii din `Thing` care au toppinguri numai din clasa `MozzarellaTopping`.

Fie o proprietate-legătură oarecare `P`; restricțiile universale referitoare la `P` nu statuează existența unei relații în care e implicată `P`. Ele afirmă doar că, dacă există o astfel de corelație privitoare la `P`, atunci aceasta trebuie să implice indivizi care fac parte dintr-o anumită clasă.

⇒

Restricțiile existențiale definite asupra unei proprietăți `P`, care coreleză o clasă `A` cu o clasă `B`, afirmă faptul că indivizii din clasa `A` intră în corespondență de tip `P` cu cel puțin un individ din clasa `B`.

Restricțiile universale definite asupra unei proprietăți `P`, care coreleză o clasă `A` cu o clasă `B`, afirmă faptul că indivizii din clasa `A` intră în corespondență de tip `P` numai cu indivizi din clasa `B`.

Exemplu

$\forall \text{ hasTopping } \text{ CheeseTopping OR VegetableTopping} \Rightarrow$
membrii acestei clase de Pizza nu pot avea un topping decât din clasa `CheeseTopping` sau din clasa `VegetableTopping`.

Atentie la operatorul logic !!!

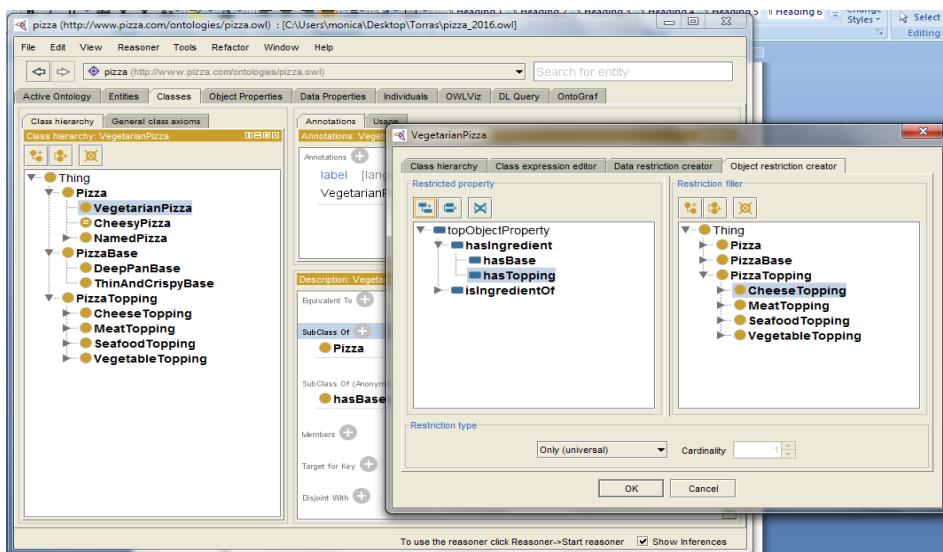
Tema 31: Să se creeze o nouă subclăsă a clasei Pizza, numită VegetarianPizza, prin specificarea faptului că membrii ei pot să aibă un topping numai din clasa CheeseTopping sau din clasa VegetableTopping.

O greșeală frecventă în astfel de situații este utilizarea operatorului logic și în loc de sau. Deși logic incorectă, afirmația nu ar fi fost semnalată ca eroare de reasoner decât dacă cele 2 clase ar fi fost a priori declarate disjuncte.

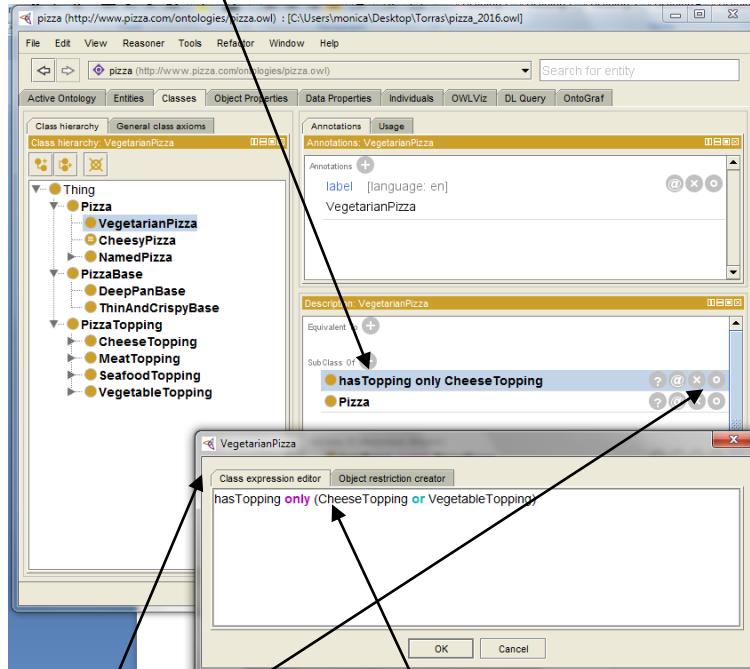
Același lucru s-ar întâmpla și dacă s-ar crea 2 restricții universale, una pentru CheeseTopping și una pentru VegetableTopping. deoarece atunci când se creează mai multe restricții individuale, definiția finală constă din intersecția acestora.

Se procedează astfel:

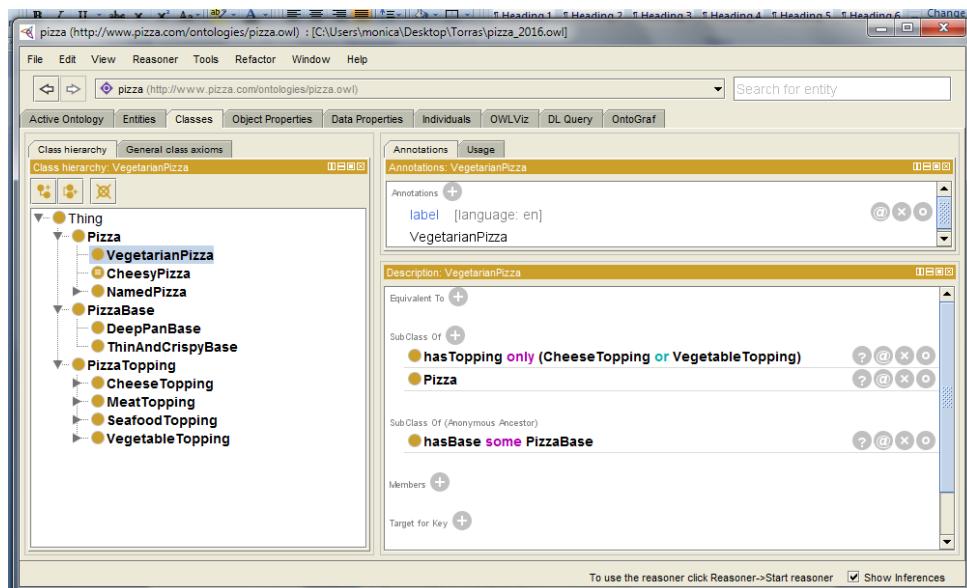
1. se creează subclasa VegetarianPizza a clasei Pizza în panelul *Class hierarchy*
2. se definește clasa prin restricție universală asupra proprietății hasTopping (i.e. în loc să confirmăm valoarea implicită *some* din caseta *Restriction type*, alegem valoarea *only* →OK)



⇒ a apărut o condiție necesară, incompletă



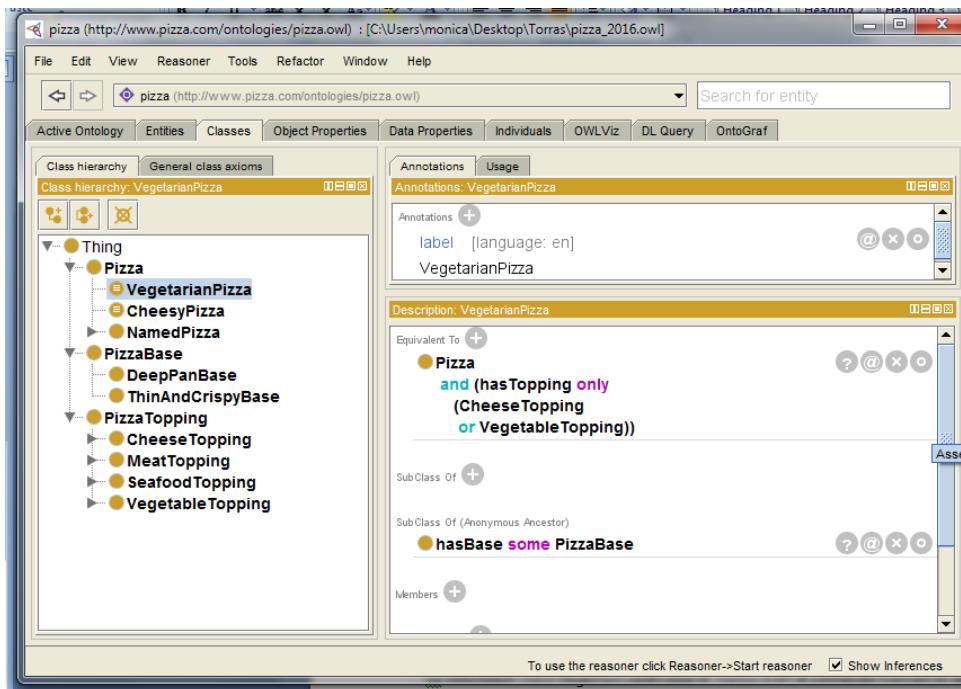
3. clik pe iconul *Edit* al acestei definiții ⇒ se reafisează fd *Description* → comutăm în tabul *Class expression editor* → edităm expresia → OK ⇒



acum definiția clasei primitive *VegetarianPizza* e încheiată.
O putem transforma într-o clasă complet definită (derivată), ca mai sus.

Tema 32: Să se convertească clasa VegetarianPizza într-o clasă complet definită (condiții necesare → condiții necesare și suficiente)

se selectează clasa VegetarianPizza → meniul *Edit* → comanda *Convert to defined class* ⇒



Observație

Această restricție NU putea fi creată și ca sumă a două restricții

- ✓ hasTopping CheeseTopping
- ✓ hasTopping VegetableTopping

pentru că ar rezulta într-o restricție

✓ hasTopping CheeseTopping **AND** VegetableTopping
 ceea ce grammatical este corect dar logic nu, încărcat am declarat cele două clase disjuncte!

Restricții de cardinalitate

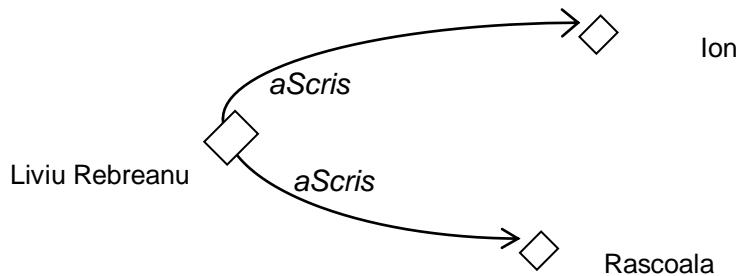
În **OWL** putem descrie o clasă specificând că ea este formată din indivizi care au minimum k, maximum k sau exact k relații cu alți indivizi sau cu alte valori ale unui tip de dată.

Acest tip de restricții se numesc restricții de cardinalitate:

- Minimum Cardinality Restriction indică numărul minim de relații distințe la care trebuie să participe individul ;
- Maximum Cardinality Restriction indică numărul maxim de relații distințe la care trebuie să participe individul ;

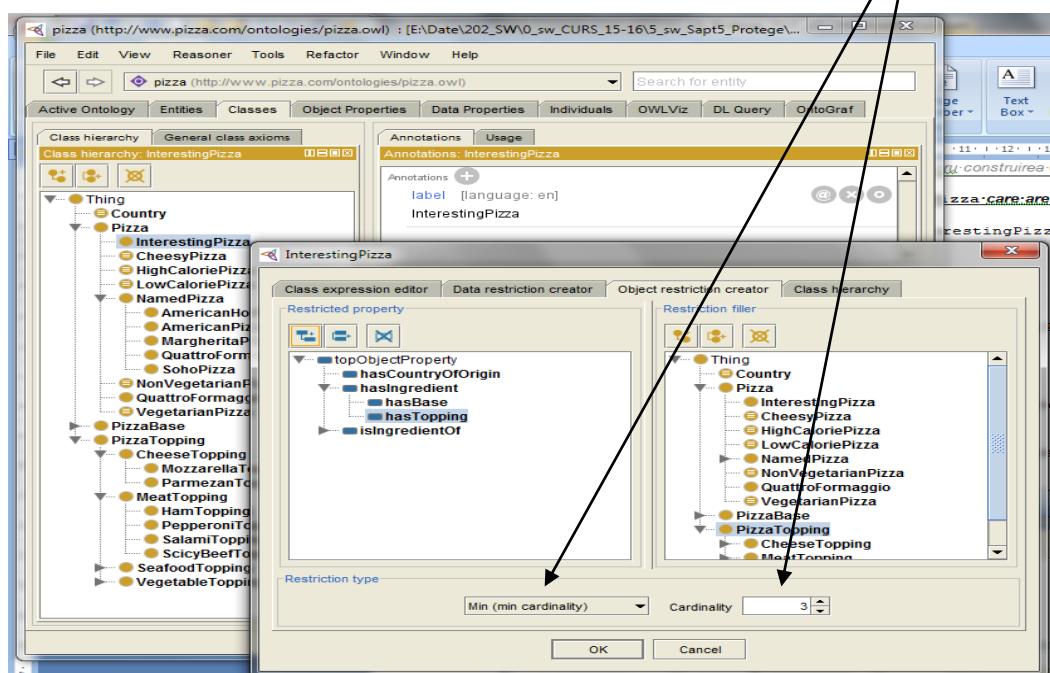
- Cardinality Restriction indică numărul exact de relații distincte la care trebuie să participe individul.

În **OWL** relațiile sunt contabilizate ca diferite numai dacă se poate afirma că indivizi/clasele care constituie argumentele acestora sunt distincți. De exemplu, știm că Liviu Rebreanu a scris și românul „Ion” și romanul „Răscoala” ⇒ putem afirma că scriitorul „Liviu Rebreanu” satisfacă o restricție de cardinalitate minimă 2 în raport cu proprietatea „aScris” numai dacă cele două romane au fost declarate ca indivizi distincți în clasa “Romane”.

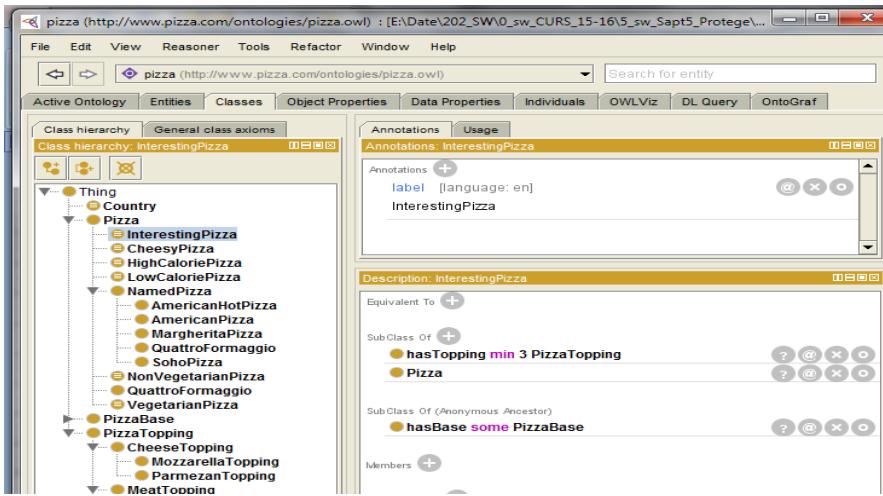


Tema 43: Să se creeze o pizza, numită InterestingPizza care are cel puțin 3 topping-uri.

tabul Classes → clasa Pizza → se creează subclasa InterestingPizza → → directiva SubClass Of → iconul Add ⇒ fd → tabul Object restriction creator → se aleg proprietatea hasTopping și clasa PizzaTopping → se aleg restricția Min și cardinalitatea 3 → OK ⇒



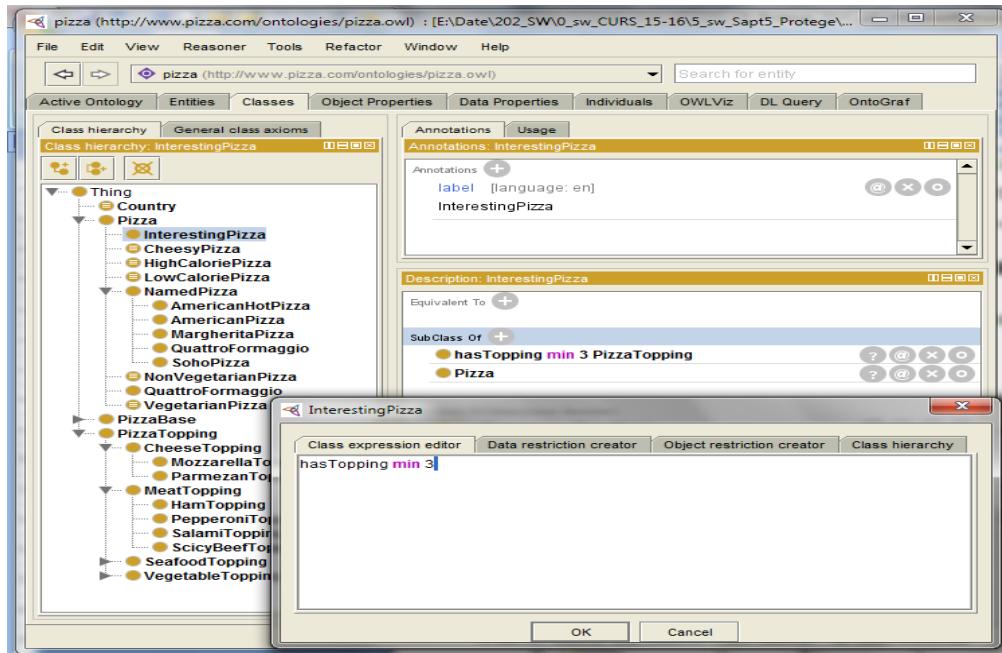
⇒



→ se transformă în clasă derivată

Observație

Un rezultat similar (Pizza and (hasTopping min 3 Thing)) se obține dacă se tastează direct în tabul *Class expression editor* codul hasTopping min 3.

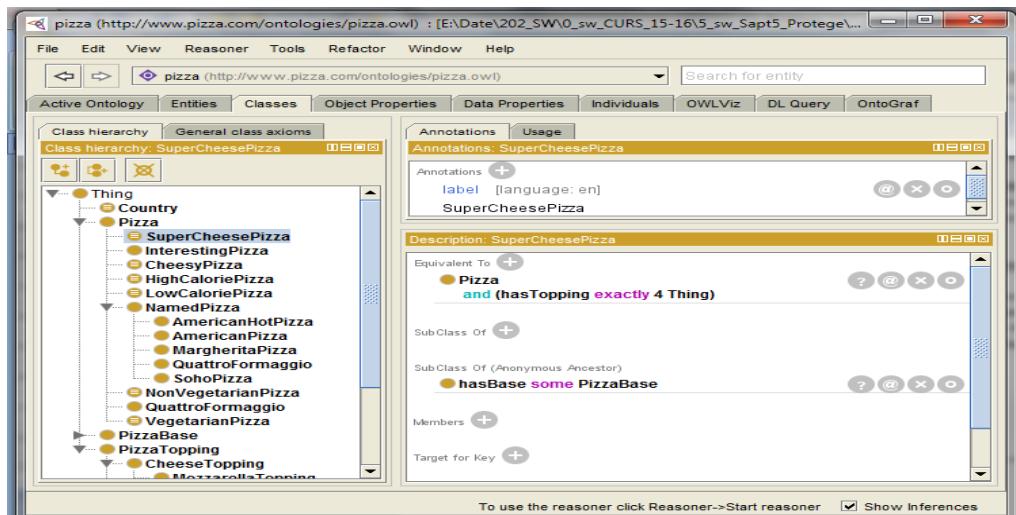


Tema 44: Să se verifice consistența ontologiei cu reasonerul

Rezultatul va fi clasificarea claselor AmericanaPizza, AmericanaHotPizza, SohoPizza ca subclase ale InterestingPizza (nu și MargheritaPizza pt că ea are numai 2 toppinguri!).

Tema 45: Să se creeze o pizza, numită SuperCheesePizza care are exact 4 topping-uri de brânză.

tabul *Classes* → clasa *Pizza* → se creează subclasa *SuperCheesePizza* → → directiva *SubClass Of* → iconul *Add* ⇒ fd → tabul *Object restriction creator* → se aleg proprietatea *hasTopping* și clasa *CheeseTopping* → se aleg restricția *Exactly* și cardinalitatea 4 (echivalent: se tastează direct în tabul *Class expression editor* codul *hasTopping exactly 4*) → OK ⇒



Proprietăți-atribut

Definiție

O proprietate-atribut corelează un individ cu un tip de date din *XML Schema* sau cu un literal *rdf* (i.e. descrie relațiile dintre un individ și valorile atributelor sale (caracteristicile sale)).

Pentru a crea o proprietate-atribut , trebuie folosit tabul *Datatype Properties* sau tabul *Entities*.

Vom utiliza proprietăți-atribut pentru a descrie profilul caloric al diferitelor tipuri de pizza. Vom stabili apoi anumite valori minime/maxime pentru a le incadra în categoriile “inalt calorice”, respectiv “slab calorice”.

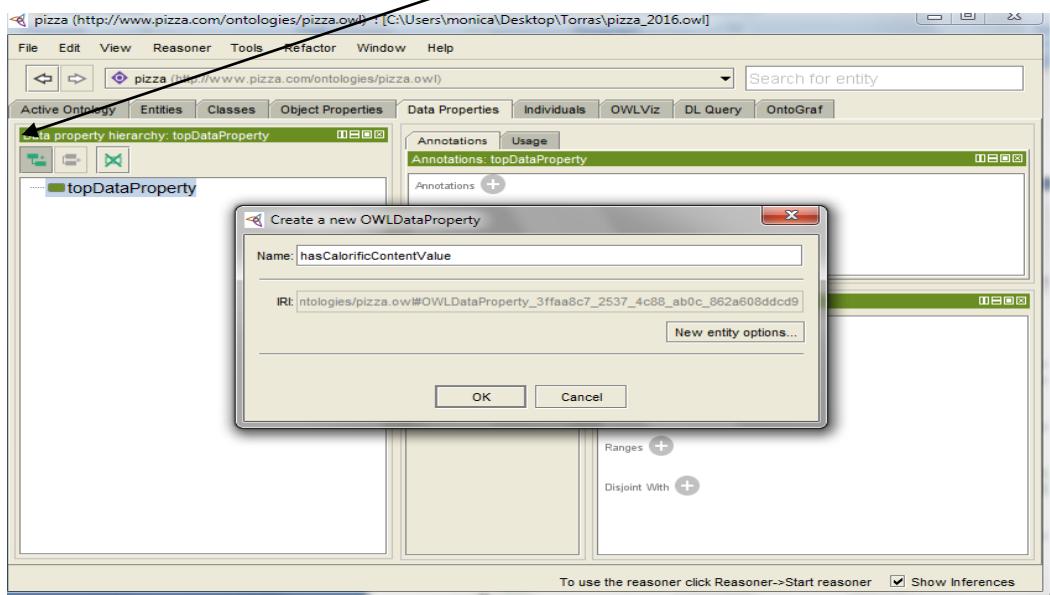
=>

1. vom crea proprietatea-atribut *hasCalorificContentValue* (pentru a stabili numărul de calorii al fiecărui tip de pizza);
2. vom crea mai multe exemplare de pizza cu un anumit număr de calorii
3. vom crea 2 mari clase de pizza: cu număr mare / mic de calorii.

Tema 46: Să se creeze o proprietate-atribut denumită hasCalorificContentValue

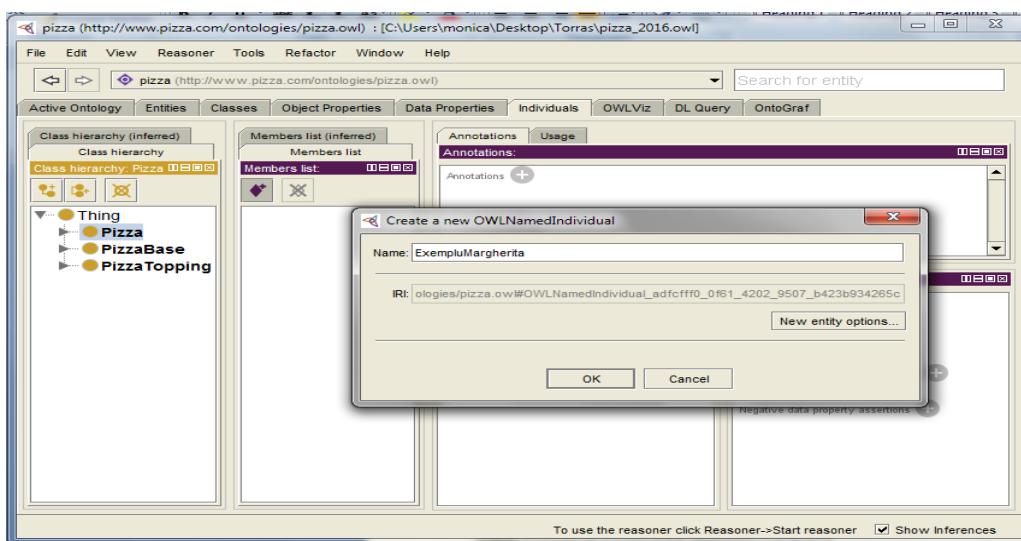
Se procedează astfel:

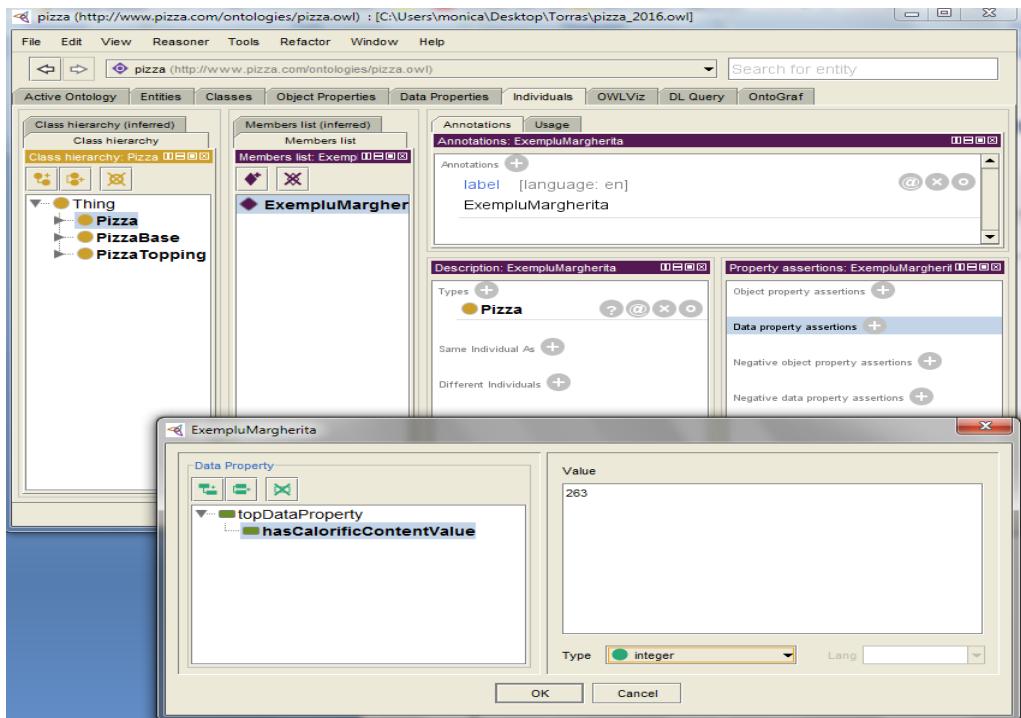
tabul *Datatype Properties* → iconul *Add subproperty* → se tastează numele proprietății →OK



Tema 47: Să se creeze câțiva indivizi pentru clasa Pizza

tabul *Entities* sau tabul *Individuals* → iconul *Add individual* ⇒ fd: se tastează numele (aici: ExempluMargherita)



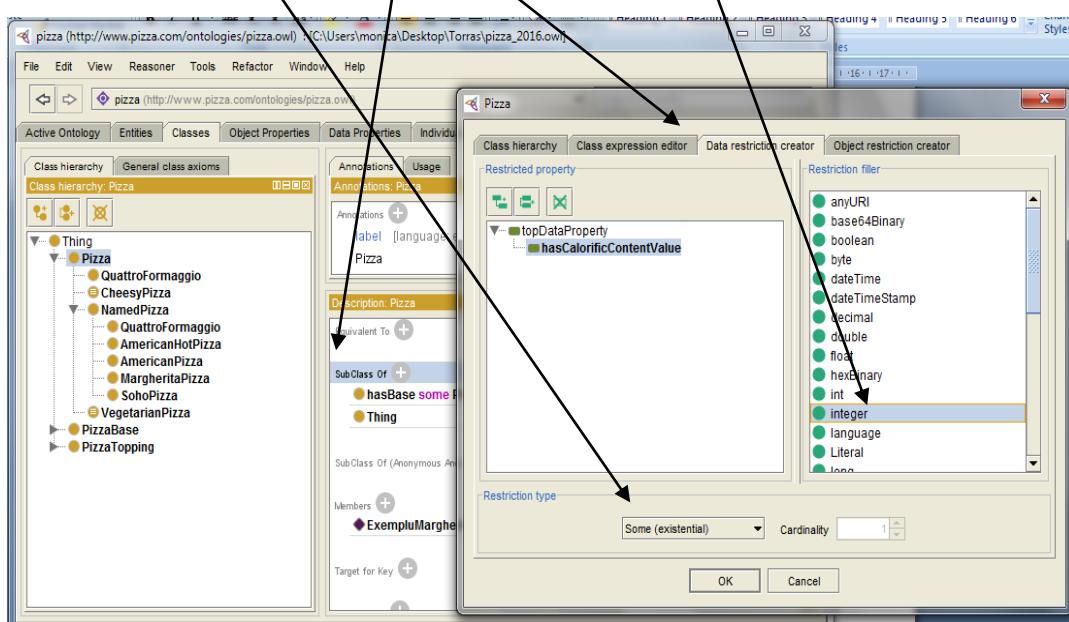


idem pt ***Exemplu2Margherita***, ***ExempluAmericanHotPizza***, ***ExempluQuattroFormaggio*** cu valorile respective: 400, 200, 723 pentru ***hasCalorificContentValue***.

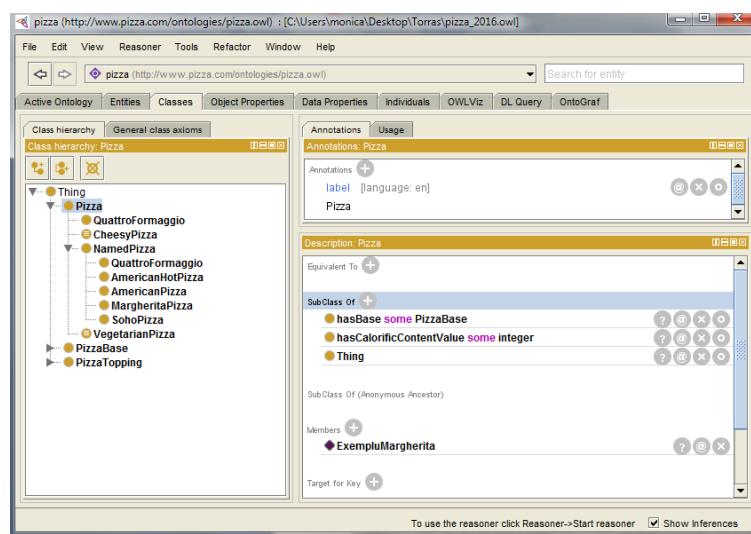
Și proprietățile-atribut pot fi folosite pt restricționare. Ceea ce rezultă este un tip de date restricționat asociat cu clasa-domeniu de definiție a proprietății –atribut.

Tema 48: Să se creeze un tip de date restrictionat care să afirme ca orice Pizza are o valoare calorică.

- tabul Classes → clasa Pizza → subtabul Description →
 → iconul SubClass Of → iconul ADD ⇒ fd →
 → tabul Data restriction creator →
 → în ierarhia de proprietăți-atribut se selectează proprietatea de restricționat (aici hasCalorificContentValue) →
 → în tabul Restriction filler se alege tipul de dată (aici Integer)
 → în zona Restriction type se alege tipul de restricție (aici: de cuantificare existential, deci some) → OK



⇒



Restrictiile de cardinalitate impuse proprietatilor-atribut permit crearea unor clase primitive (parțial definite) prin specificarea unui domeniu de valori pentru atributul respectiv.

Acest domeniu de valori (de fapt, un subdomeniu al unui tip de date predefinit) poate fi specificat prin:

- operatori relationali ($>$, $<$, $=$, \geq , \leq)
- operatorii *min cardinality*, *max cardinality*, *exact cardinality*.

Tema 49: Crearea unei noi clase, `HighCaloriePizza`, care e caracterizata printr-un numar de calorii de cel putin 400

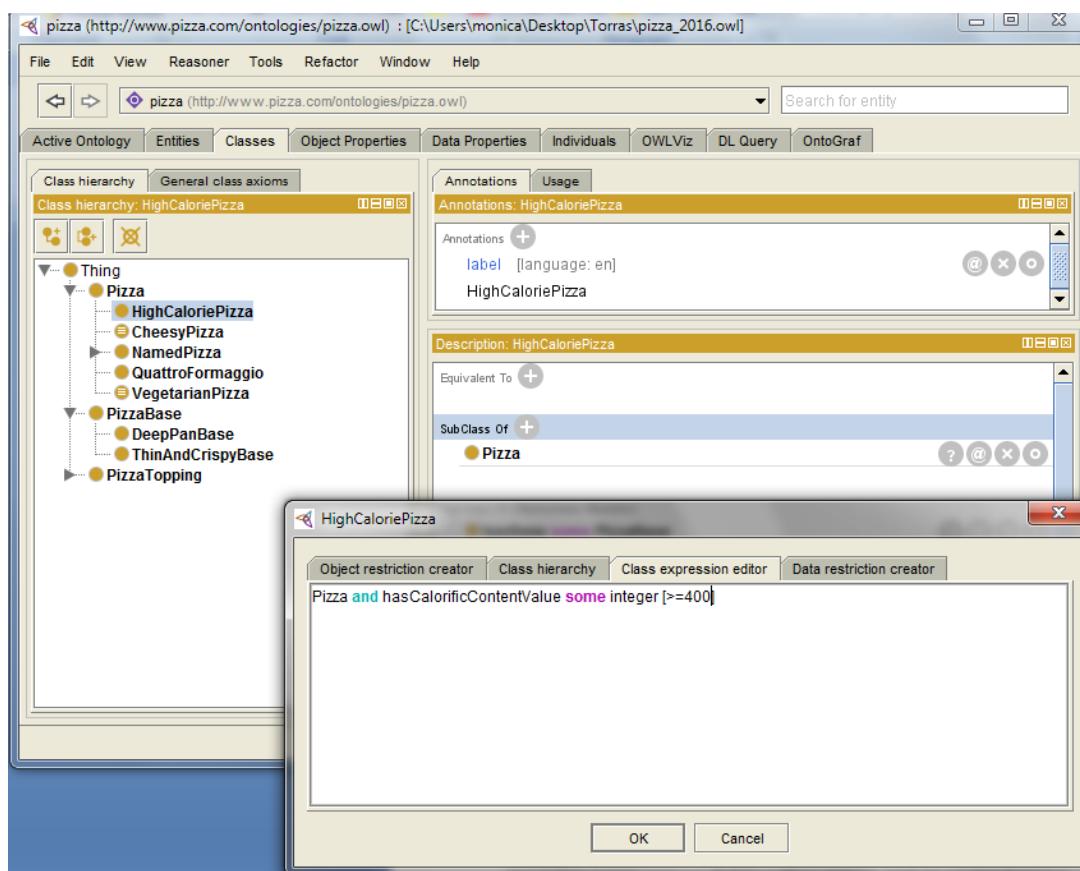
tabul *Classes* → se creează subclasa `HighCaloriePizza` a clasei `Pizza` →

→ subtabul *Description* → iconul *SubClass Of* → iconul *Add* ⇒ fd → tabul *Data restriction creator* →

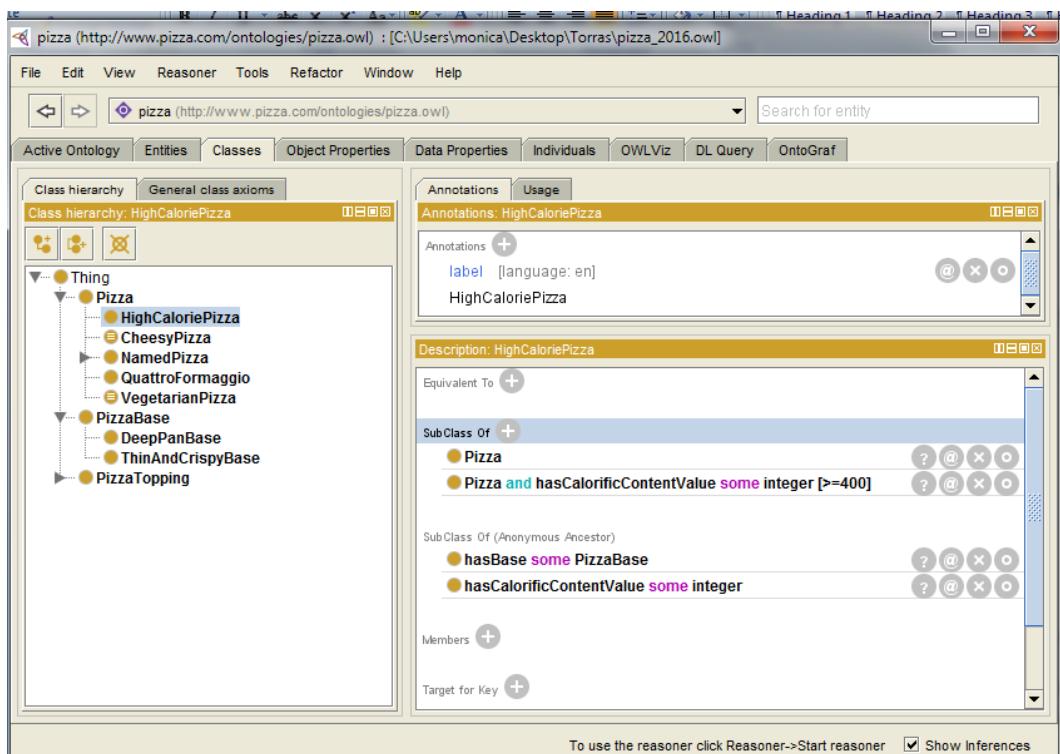
→ în ierarhia de proprietăți- atribut se selectează proprietatea de restricționată (aici `hasCalorificContentValue`) →

→ în tabul *Restriction filler* se alege tipul de dată (aici *Integer*)

→ în zona *Restriction type* se alege tipul de restricție (aici de cuantificare existențial, deci *some*) → OK →

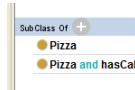


⇒



→ se transformă în clasă derivată (complet definită):

clic pe iconul din stanga descrierii restricției → meniul *Edit* → comanda *Convert to defined class*.



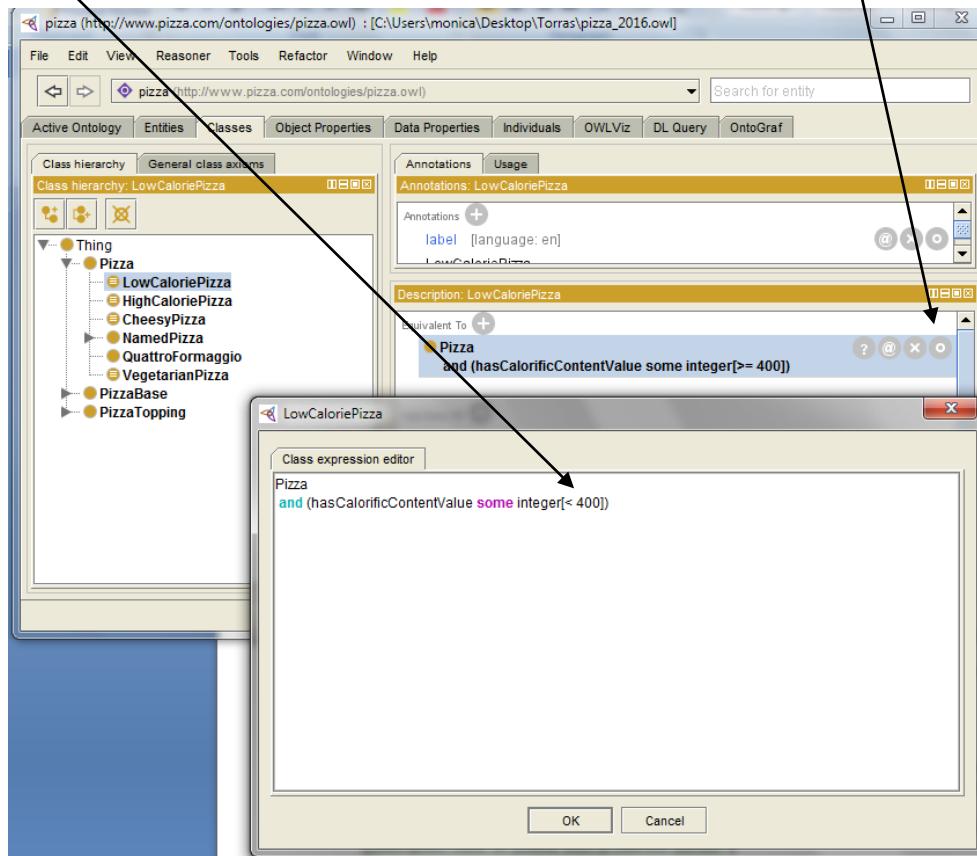
Metodă alternativă:

Se creează restricția "hasCalorificContentValue some integer" ca mai sus (Tema 48) → clic pe iconul *Edit* al restricției ⇒ fd
 → tabul *Class expression editor* → se modifică expresia prin adăugarea limitei (aici: [>=400])

Analog, se creează o clasă, LowCaloriePizza, ca având mai puțin de 400 de calorii:

tabul *Classes* → clic pe clasa HighCaloriePizza → meniul *Edit* → comanda *Duplicate selected class* ⇒ fd *Duplicate class* → se tastează numele noii clase (aici: LowCaloriePizza) → OK

⇒ a apărut noua clasă, identică cu sursa ei → click pe iconul *Edit* din dreptul definiției sale de clasă (derivată) ⇒ fd → se modifică definiția (aici: se înlocuiește operatorul \geq cu $<$) → OK ⇒



Tema 50: Clasificarea rețetelor de pizza în funcție de valoarea lor calorică (i.e. valorile proprietății hasCalorificContentValue)

se lansează *reasonerul* → se selectează pe rând clasele care conțin indivizi pt a vedea "repartiția" acestora → eventual, se afișează și tabul *Individuals*.

Observație

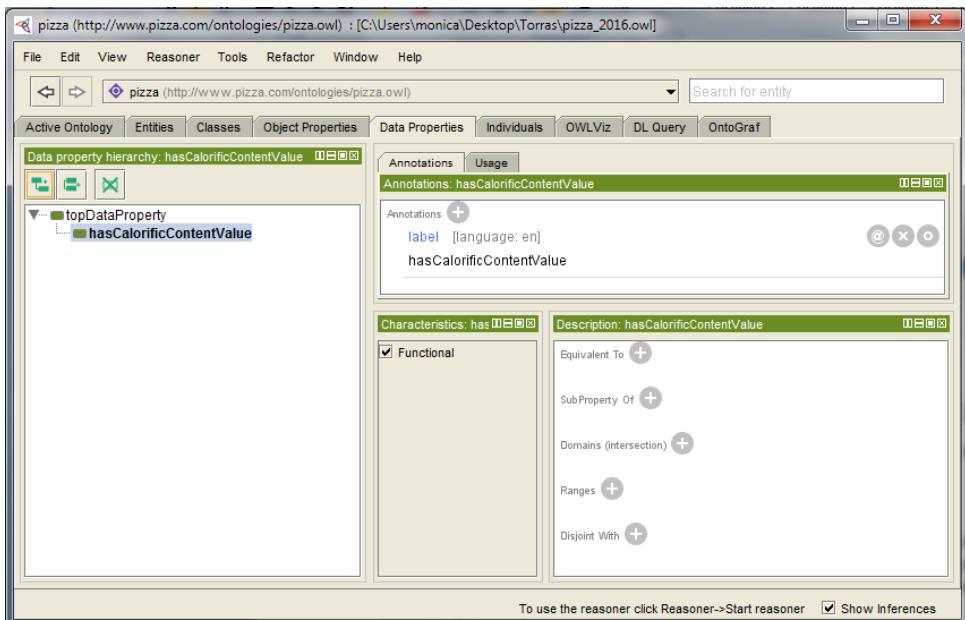
Evident, o rețetă de pizza nu poate avea decât o singură valoare calorică ⇒ trebuie să facem această specificare și în ontologia noastră, i.e. trebuie să definim proprietatea-atribut *hasCalorificContentValue* ca fiind de tip funcție (ia cel mult o valoare pt fiecare argument).

De fapt, *Functional* este singura caracteristică a proprietăților-legătură care are loc și pentru proprietățile-atribut.

Tema 51: Să se definească proprietatea-atribut *hasCalorificContentValue* ca fiind de tip funcție

se selectează proprietatea-atribut *hasCalorificContentValue* → click pe caseta de opțiune *Functional*

→ se poate verifica dacă setarea este corectă creând o nouă rețetă de pizza și dându-i două valori calorice; se lansează apoi *reasoner*-ul care va semnala inconsistenta.



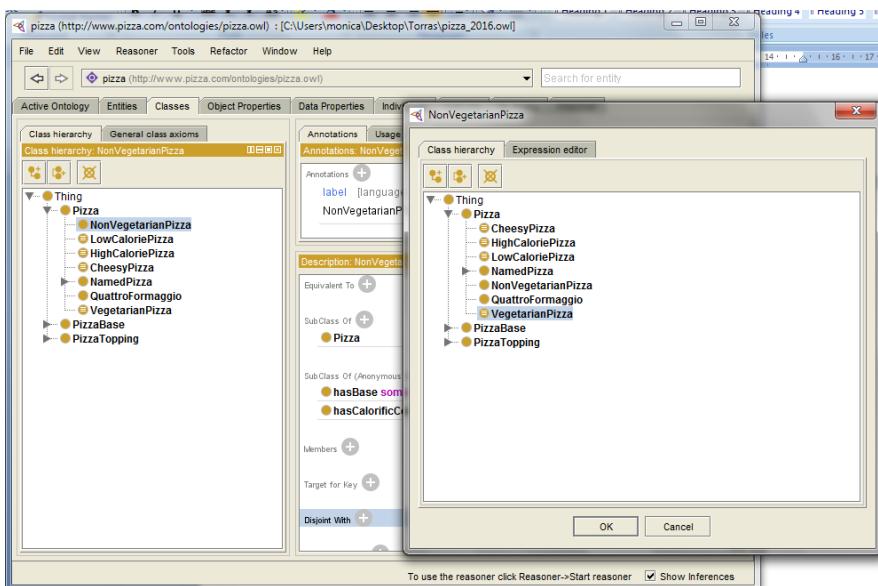
Convenția OWL privind Lumile Deschise în Protege

Vrem să creăm clase prin complementarea unei clase deja create.

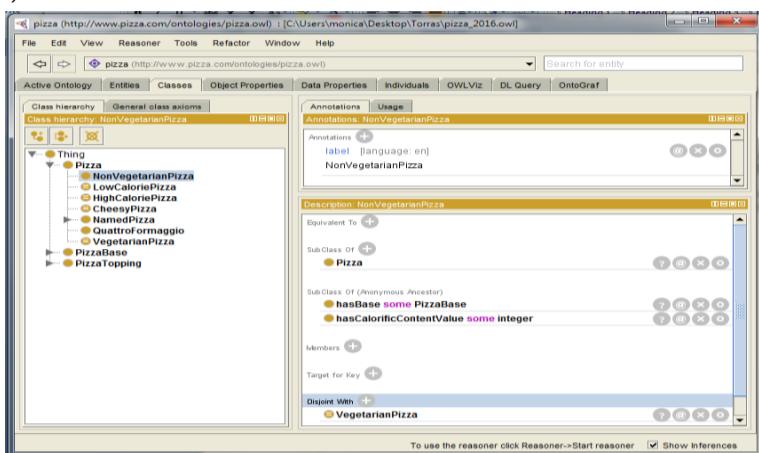
Tema 52: Să se creeze o subclasă a clasei Pizza, cu numele *NonVegetarianPizza* și să se definească a fi disjunctă de clasa *VegetarianPizza*.

se selectează clasa *Pizza* → clik pe iconul *Add subclass* → se tastează numele → directiva *Disjoint With* → clik pe iconul *Add* ⇒ fd → se selectează clasa *VegetarianPizza* → OK

Editorul Protégé pentru crearea ontologiilor

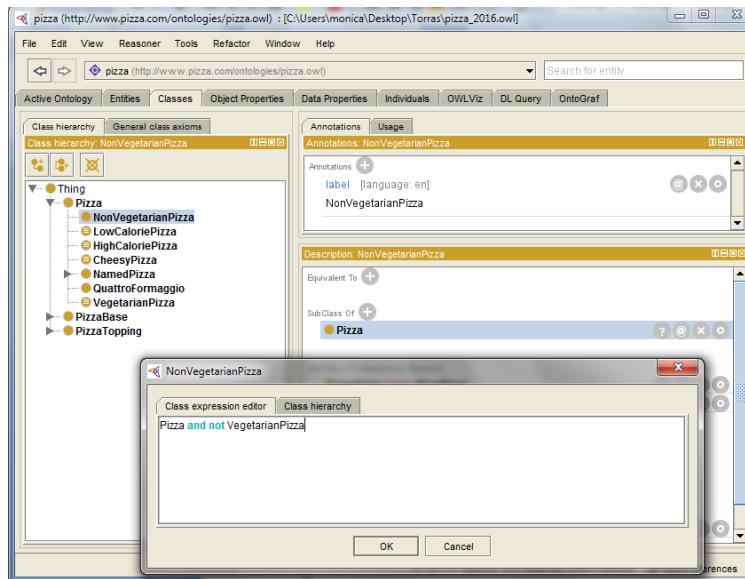


⇒

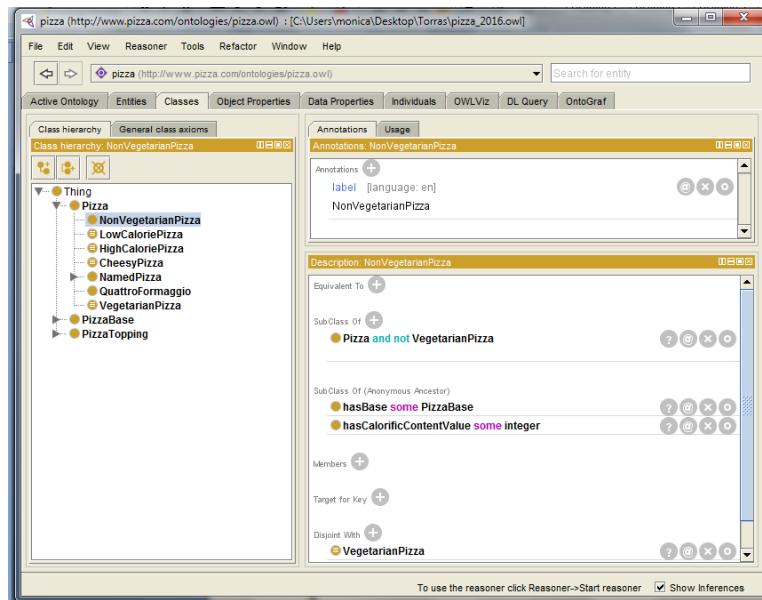


Tema 53: Să se definească *nonVegetarianPizza* drept complement al clasei *VegetarianPizza*

se selectează *nonVegetarianPizza* → click pe iconul *Edit* din linia *SubClass of* *Pizza* ⇒ fd → se editează definiția “*Pizza*” la “*Pizza and not VegetarianPizza*” → OK

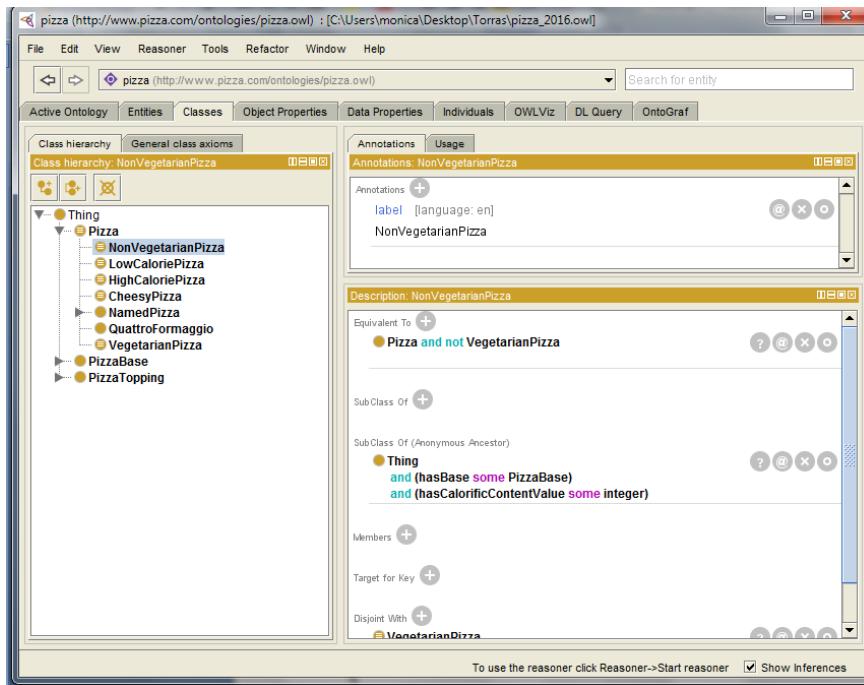


⇒



Tema 54: Să se adauge *Pizza* ca o condiție necesară și suficientă pentru clasa *NonVegetarianPizza*

se selectează clasa *NonVegetarianPizza* → meniu *Edit* → comanda *Convert to defined class* ⇒



Tema 55: Să se clasifice ontologia

se lansează reasoner-ul →

Ierarhia calculată ar trebui să arate ca ea din figura de mai jos, pentru că rețetele MargheritaPizza și SohoPizza au fost clasificate ca subclase ale VegetarianPizza. AmericanaPizza și AmericanaHotPizza au fost clasificate ca subclase ale NonVegetarianPizza.



Crearea membrilor unei clase (indivizi)

OWL permite definirea indivizilor și stabilirea proprietăților lor.

Observație

În "dilema" clasă vs. individ, trebuie avută în vedere ideea: un individ din clasa C este ceva care constituie o instanță a clasei C.

Exemplu:

Creăm clasa ţări și nu clasa Anglia pt că, altfel, un individ din clasa Anglia ar trebui să fie ceva care este o instanță a clasei Anglia.

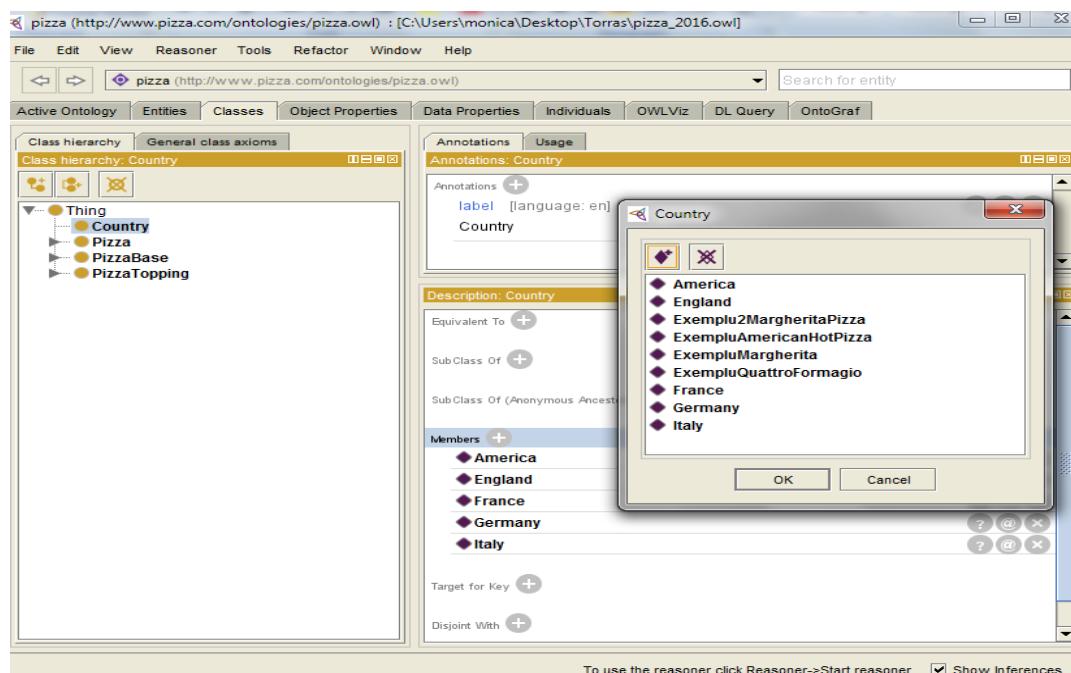
Tema 58: Să se creeze clasa Country și să se populeze cu indivizii America, England, France, Germany, Italy.

tabul Classes → clasa Thing → iconul Add subclass ⇒ fd ⇒ se tastează numele clasei (aici: Country)

→ tabul Individuals → iconul Add individual ⇒ fd ⇒ se tastează numele individului (aici: England) → OK → se reia pt celelalte ţări →

Optional:

tabul Classes → clasa Country → panelul Description → directiva Members → iconul Add ⇒ fd Country → clik pe numele individului care trebuie să facă parte din clasă (aici: America, England, France, Germany, Italy) → OK ⇒



Crearea unei clase prin restricții de tip hasValue

Restricția de tip hasValue

- este notată prin simbolul \exists
- descrie o mulțime de indivizi care au cel puțin o corelație în raport cu o anumită proprietate-legătură cu un individ bine specificat.

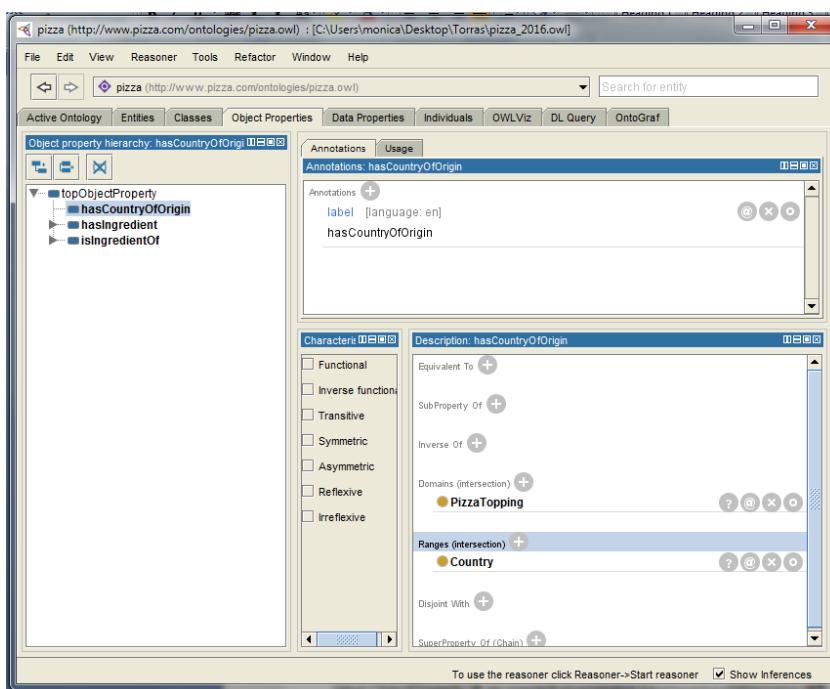
Exemplu

Putem specifica țara de origine pt fiecare ingredient de pizza (ex.: brânza mozzarella folosită pentru MozzarellaTopping provine din Italia):

creăm proprietatea `hasCountryOfOrigin` → definim pt aceasta o restricție de tip `hasValue` pe domeniul de definiție (`Country`) cu valoarea `Italy`.

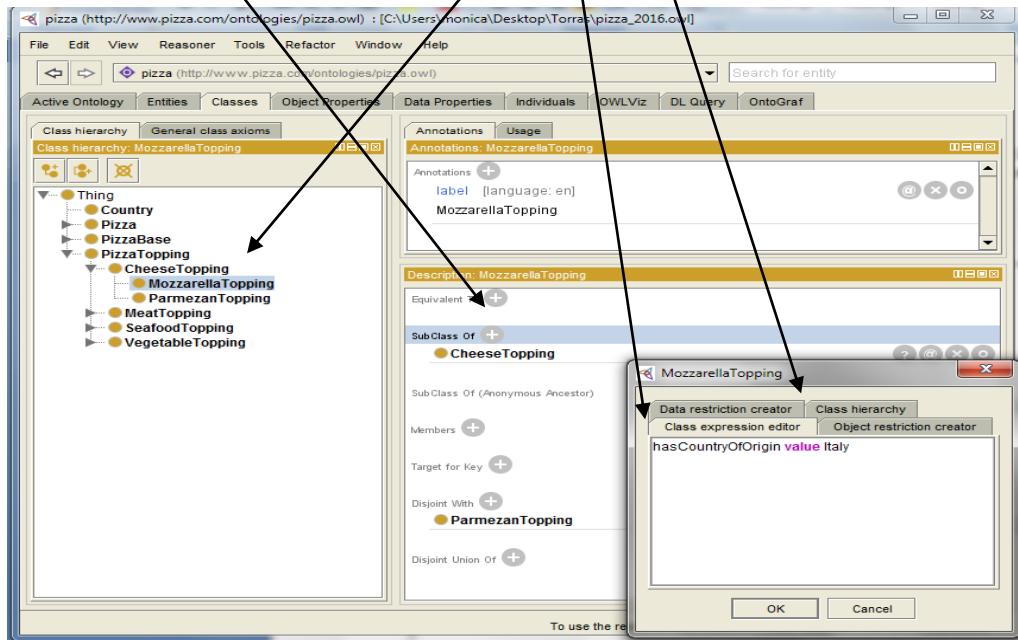
Tema 59: Să se creeze o restricție de tip hasValue pt a specifica faptul că țara de origine pt MozzarellaTopping este Italy.

tabul `ObjectProperty` → se creează proprietatea `hasCountryOfOrigin` cu domeniul `PizzaTopping` și codomeniul `Country` →

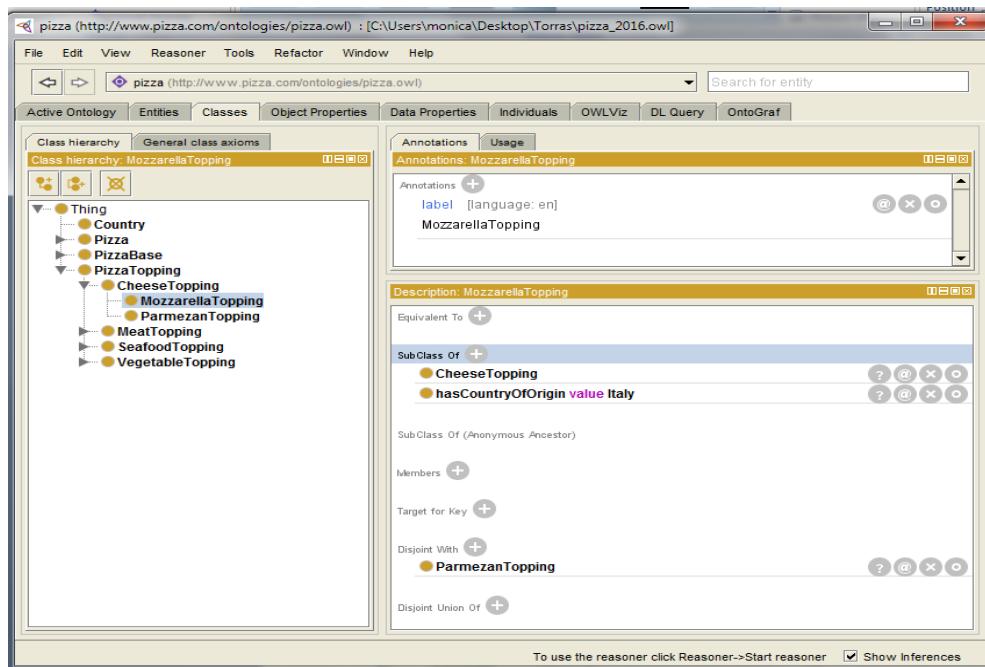


Editorul Protégé pentru crearea ontologiilor

tabul *Classes* → se selectează clasa MozzarellaTopping → directiva *SubClass Of* → iconul *Add* ⇒ fd → tabul *Class expression editor* → se tastează restricția (cuvântul rezervat *value*) aici: *hasCountryOfOrigin value Italy* → OK ⇒



⇒



Crearea unei clase prin enumerarea membrilor săi

Vom redefini clasa **Country** prin enumerarea țărilor componente.

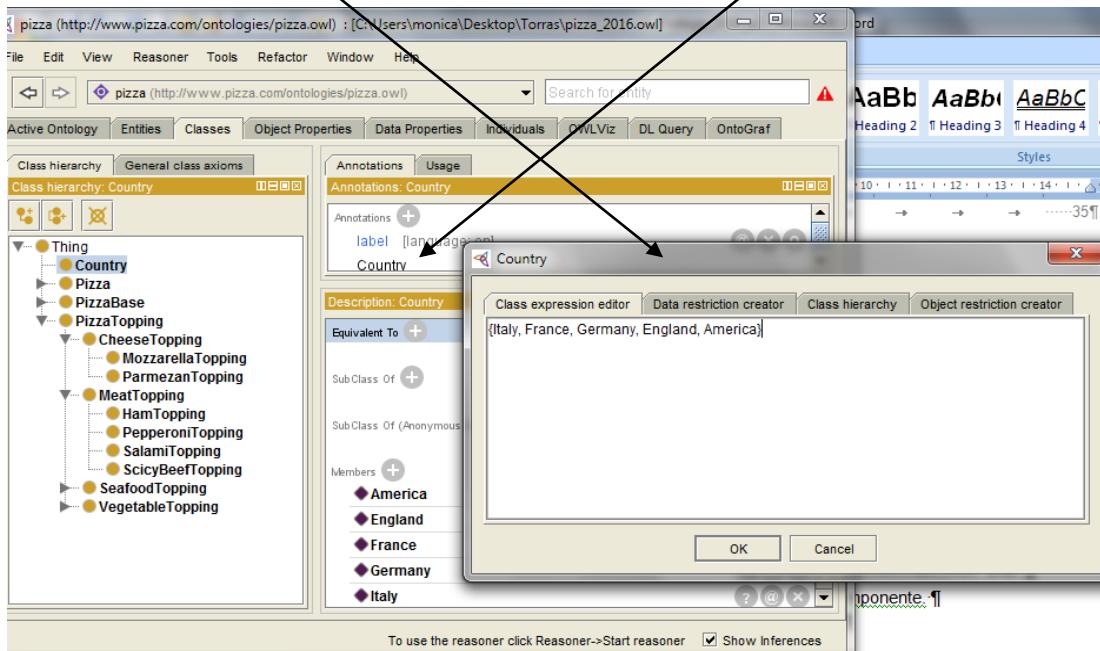
Acestea se includ între acolade și se separă cu virgule.

Obligatoriu: membrii clasei trebuie în prealabil definiti ca indivizi în clasa respectivă. Ei nu pot fi creați aici, pe loc.

Este preferabil ca identificatorii membrilor clasei să apară între ghilimele.

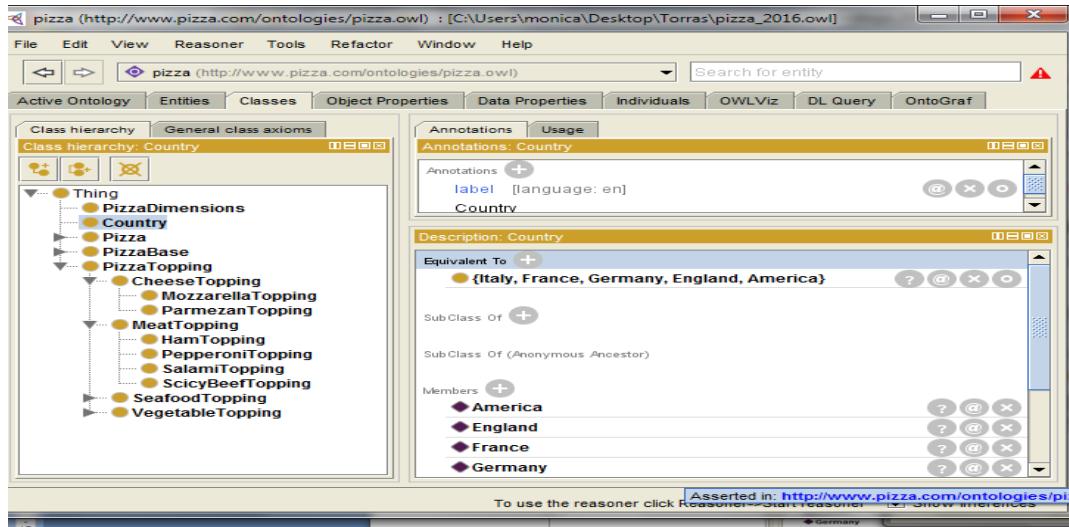
Tema 60: Să se transforme clasa Country într-o clasă enumerată

tabul **Classes** → clasa **Country** → tabul **Description** → directiva **Equivalent To** → iconul **Add** ⇒ fd → tabul **Class expression editor** → se tastează restricția **{Italy, France, Germany, England, America}** → OK ⇒

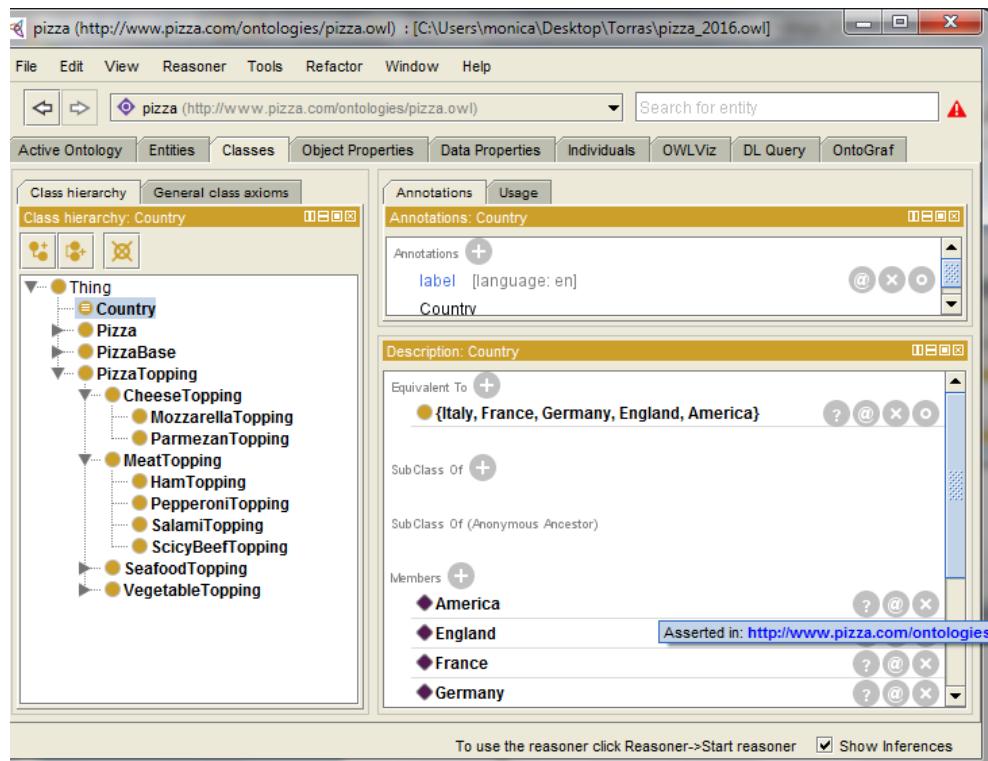


Editorul Protégé pentru crearea ontologiilor

⇒



→ se transformă în clasă derivată ⇒



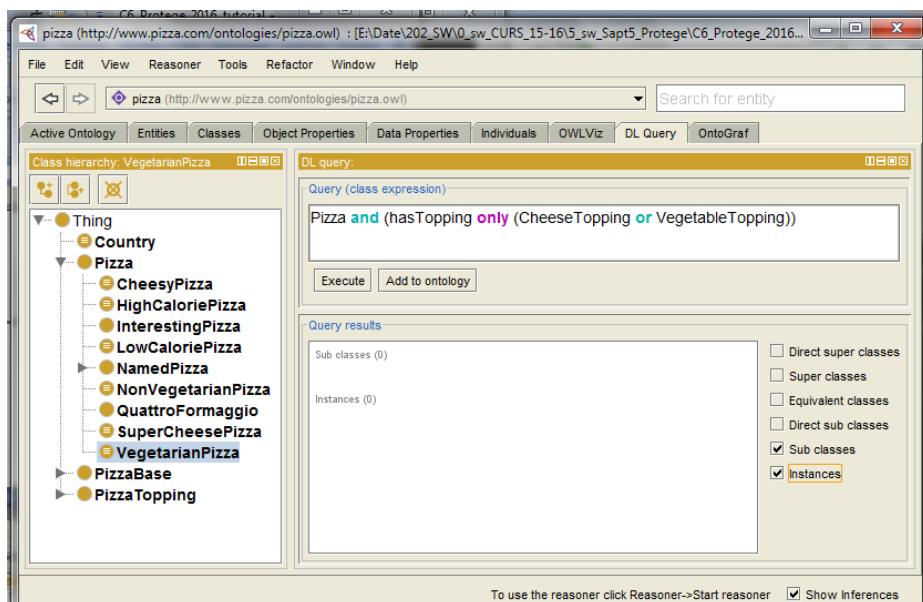
Crearea interogărilor

În tabul *DL Query* se tastează expresia de definire a unei clase (anonyme), echivalentă cu mulțimea indivizilor / subclaselor căutate. Atenție: case sensitive!

Acstea clase se pot salva ca niște clase primitive în ierarhie.

Se poate utiliza **TAB** invocarea facilității *Autocorrect*.

Trebuie selectate – după caz – casetele de opțiune *Subclasses / Individuals*.



Bibliografie

Matthew HORRIDGE, Holger KNUBLAUCH, Alain RECTOR, Robert STEVENS, Chris WROE: *A practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools Edition 1.0*, Copyright©The Manchester University, August 2011, www.co-cde.org/resource/tutorials/protegeOWLTutorial.pdf.