Master 2 LITL
Programmation pour le TAL (SLT0905V)
POO : la notion d'héritage

Franck Sajous/CLLE-ERSS

Séance 3

http://fsajous.free.fr/

Étude de cas : extracteur de triplets syntaxiques

À partir d'une analyse syntaxique...

1	Si	si	CS	17	mod	
2	les	les	DET	n=p|	3	det
3	puces	puce	NC	n=p|g=f|	7	suj
4	et	et	CC	3	coord	
5	patrons	patron	NC	n=p|g=m|	4	dep_coord
6	ponctuationnels	_	ADJ	_	5	mod
7	constituent	constituer	V	n=p|t=PS|p=3|	1	sub
8	le	le	DET	n=s|g=m|	9	det
9	type	type	NC	n=s|g=m|	7	obj
10	d'	de	P	9	dep	
11	indice	indice	NC	n=s|g=m|	10	prep
12	prédominant	prédominant	ADJ	n=s|g=m|	11	mod
13	,	,	PONCT	12	ponct	
14	les	les	DET	n=p|	16	det
15	autres	autre	ADJ	n=p|	16	mod
16	catégories	catégorie	NC	n=p|g=f|	17	suj
17	sont	être	V	n=p|t=P|p=3|	0	root
18	également	également	ADV	_	17	mod
19	bien	bien	ADV	20	mod	
20	présentes	présent	ADJ	n=p|g=f|	17	ats

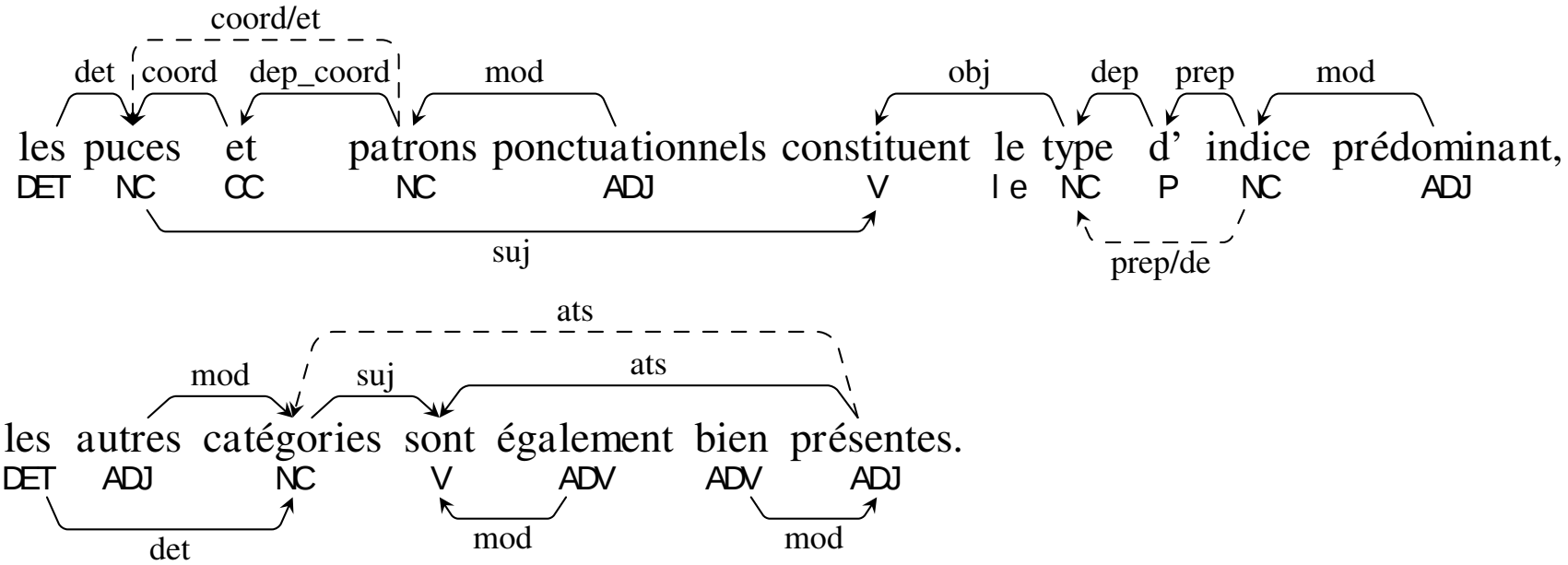
Étude de cas : extracteur de triplets syntaxiques

À partir d'une analyse syntaxique...

1	Si	si	CS	17	mod	
2	les	les	DET	n=p|	3	det
3	puces	puce	NC	n=p|g=f|	7	suj
4	et	et	CC	3	coord	
5	patrons	patron	NC	n=p|g=m|	4	dep_coord
6	ponctuationnels	_	ADJ	_	5	mod
7	constituent	constituer	V	n=p|t=PS|p=3|	1	sub
8	le	le	DET	n=s|g=m|	9	det
9	type	type	NC	n=s|g=m|	7	obj
10	d'	de	P	9	dep	
11	indice	indice	NC	n=s|g=m|	10	prep
12	prédominant	prédominant	ADJ	n=s|g=m|	11	mod

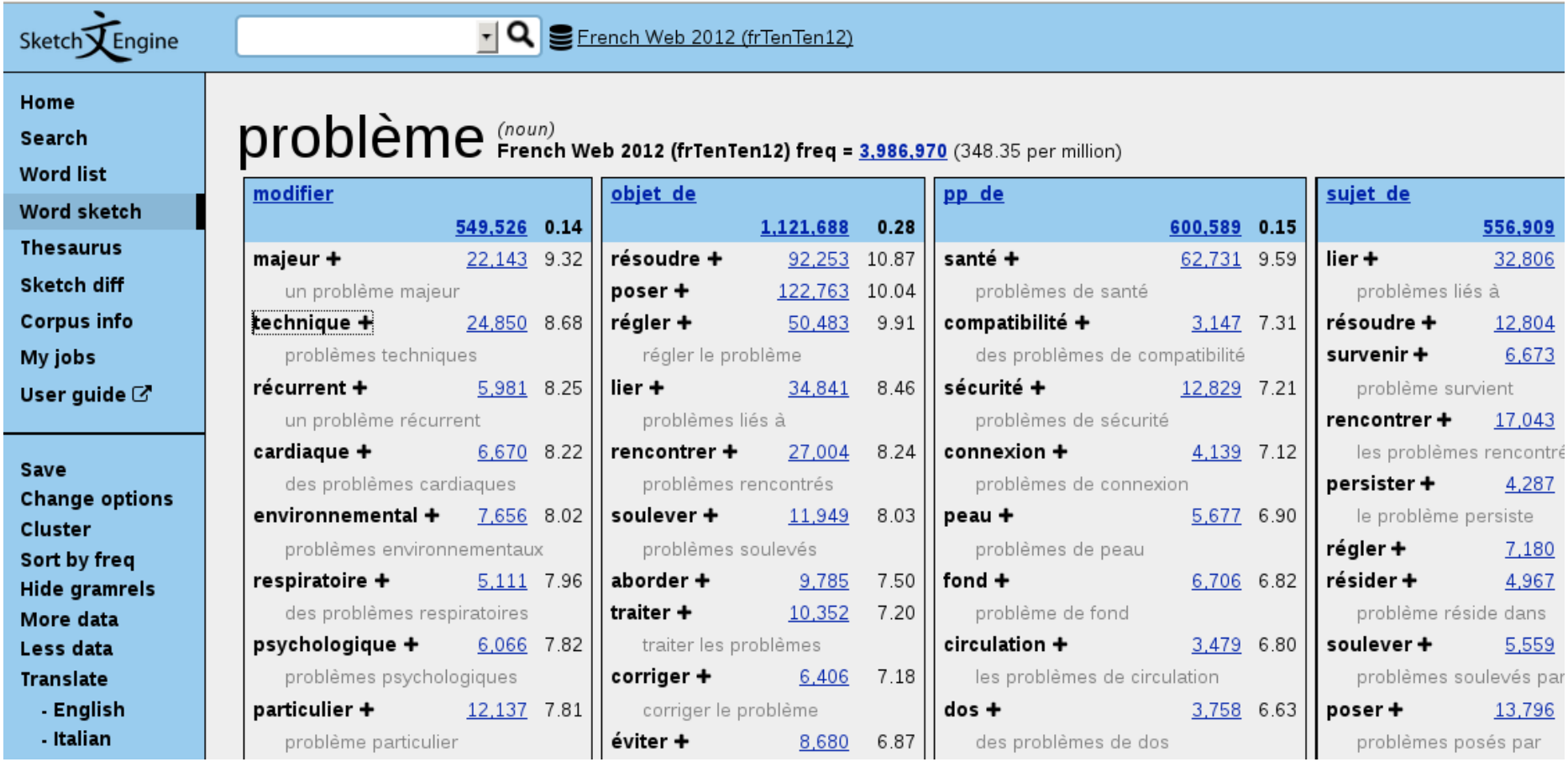
...on veut extraire tous les « triplets syntaxiques »

(i.e. <gouverneur, relation, dépendant>)



Motivation : word sketch

Recherche de collocations (e.g. *problème*)



Sketch Engine — French Web 2012 (frTenTen12)

Home · Search · Word list · Word sketch · Thesaurus · Sketch diff · Corpus info · My jobs · User guide

Save · Change options · Cluster · Sort by freq · Hide gramrels · More data · Less data · Translate · - English · - Italian

problème *(noun)* French Web 2012 (frTenTen12) freq = 3,986,970 (348.35 per million)

modifier	549,526	0.14
majeur +	22,143	9.32
un problème majeur		
technique +	24,850	8.68
problèmes techniques		
récurrent +	5,981	8.25
un problème récurrent		
cardiaque +	6,670	8.22
des problèmes cardiaques		
environnemental +	7,656	8.02
problèmes environnementaux		
respiratoire +	5,111	7.96
des problèmes respiratoires		
psychologique +	6,066	7.82
problèmes psychologiques		
particulier +	12,137	7.81
problème particulier		

objet_de	1,121,688	0.28
résoudre +	92,253	10.87
poser +	122,763	10.04
régler +	50,483	9.91
régler le problème		
lier +	34,841	8.46
problèmes liés à		
rencontrer +	27,004	8.24
problèmes rencontrés		
soulever +	11,949	8.03
problèmes soulevés		
aborder +	9,785	7.50
traiter +	10,352	7.20
traiter les problèmes		
corriger +	6,406	7.18
corriger le problème		
éviter +	8,680	6.87

pp_de	600,589	0.15
santé +	62,731	9.59
problèmes de santé		
compatibilité +	3,147	7.31
des problèmes de compatibilité		
sécurité +	12,829	7.21
problèmes de sécurité		
connexion +	4,139	7.12
problèmes de connexion		
peau +	5,677	6.90
problèmes de peau		
fond +	6,706	6.82
problème de fond		
circulation +	3,479	6.80
les problèmes de circulation		
dos +	3,758	6.63
des problèmes de dos		

sujet_de	556,909
lier +	32,806
problèmes liés à	
résoudre +	12,804
survenir +	6,673
problème survient	
rencontrer +	17,043
les problèmes rencontré	
persister +	4,287
le problème persiste	
régler +	7,180
résider +	4,967
problème réside dans	
soulever +	5,559
problèmes soulevés par	
poser +	13,796
problèmes posés par	

Autre application : extracteur automatique de syntagmes : N-adj, N-N, N-de-N, V-SP, etc.

Motivation : analyse distributionnelle

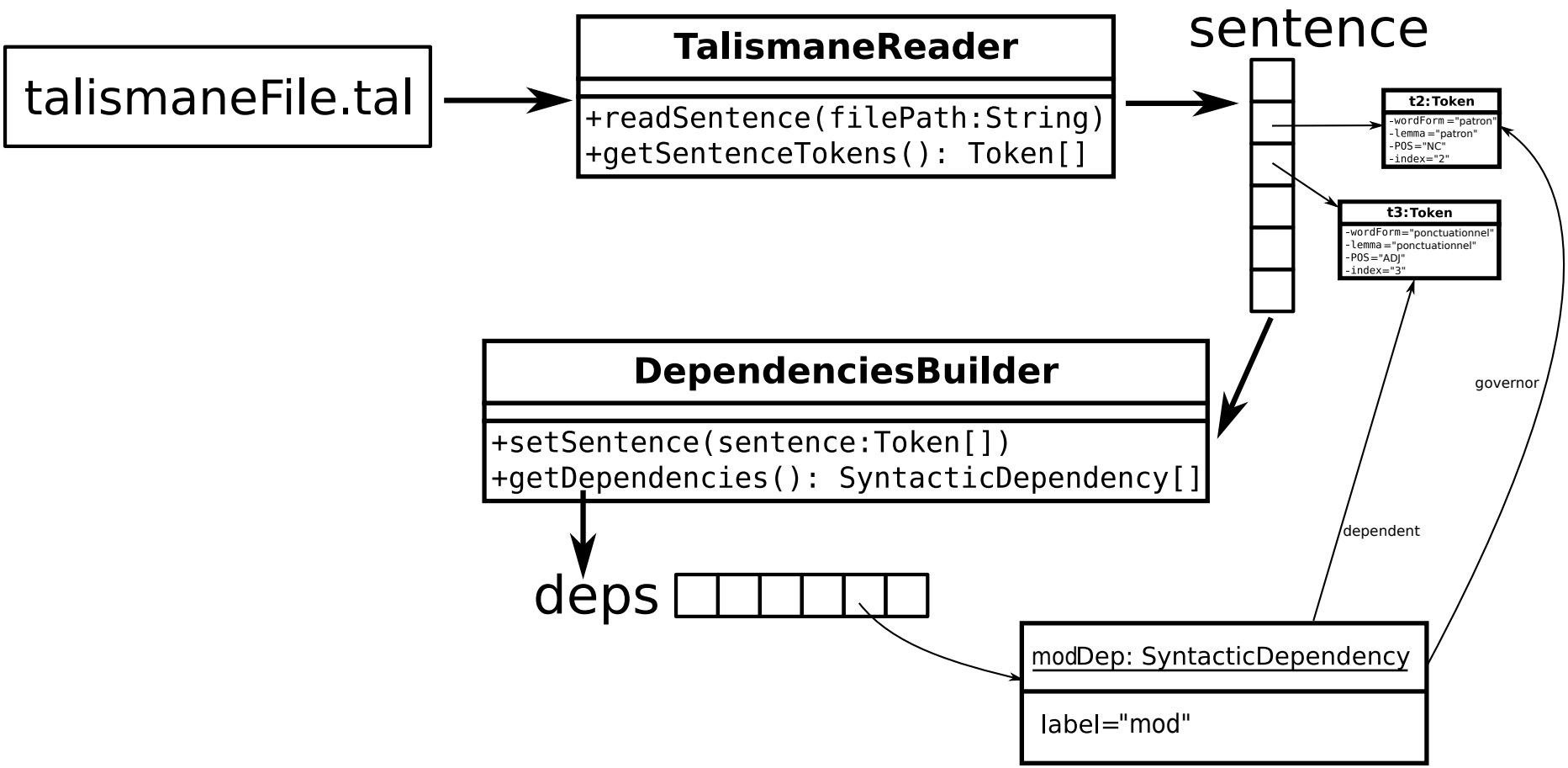
Voisins de *problème*

- difficulté
- souci
- conflit
- préoccupation

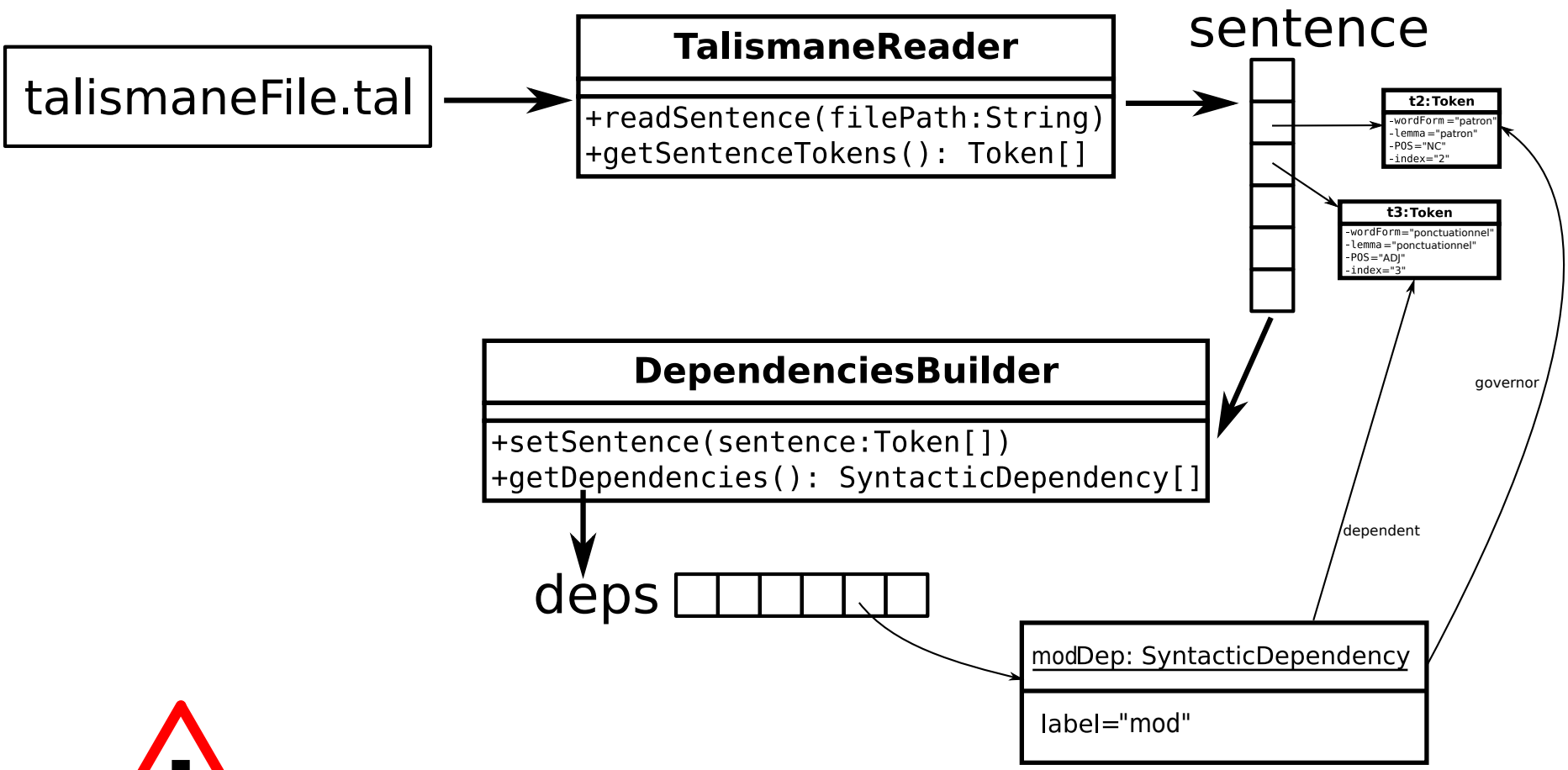
Cooccurrents syntaxiques

- modifieurs : majeur, technique, récurrent...
- objets de : résoudre, poser, régler, rencontrer...
- prép/de : santé, compatibilité...
- sujet de : survenir, persister, résider...

Principe d'architecture

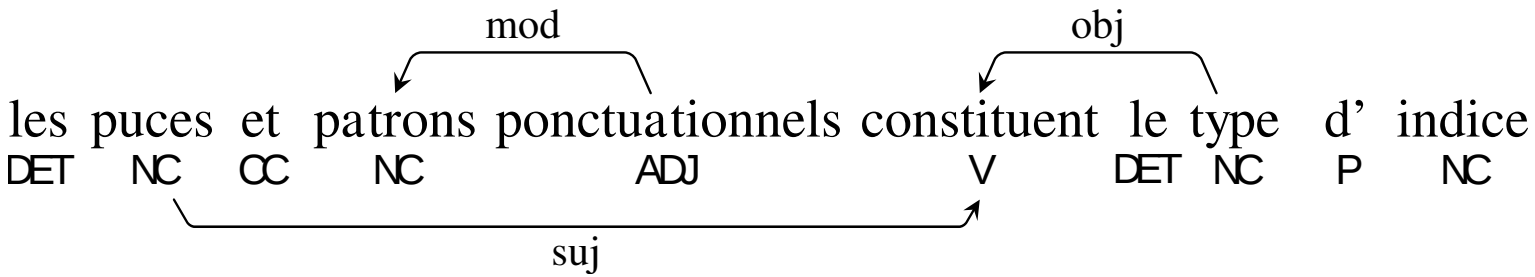


Principe d'architecture

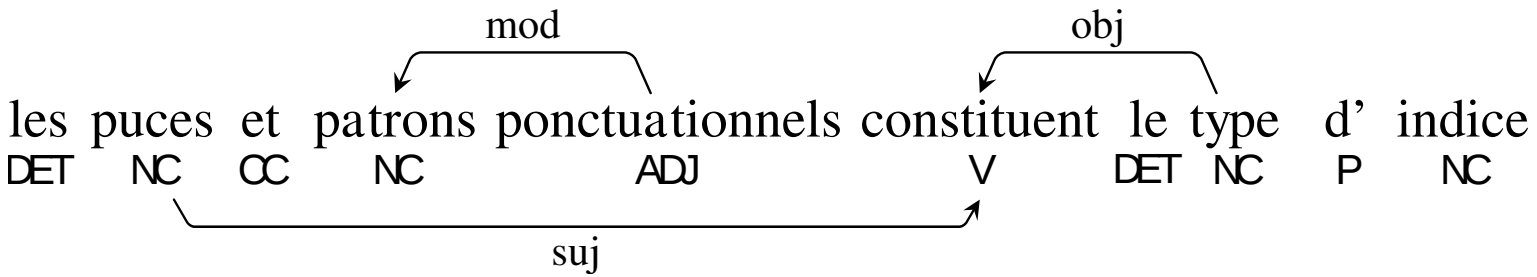


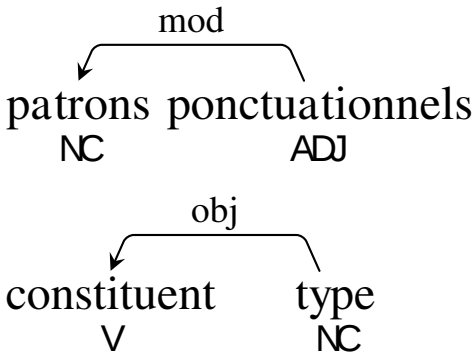
Ce schéma « coin de table » ne suit aucun formalisme

Modélisation

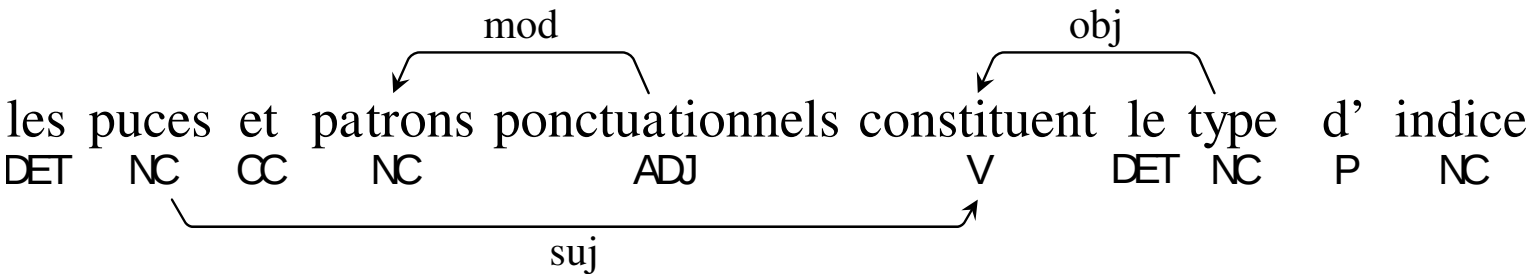


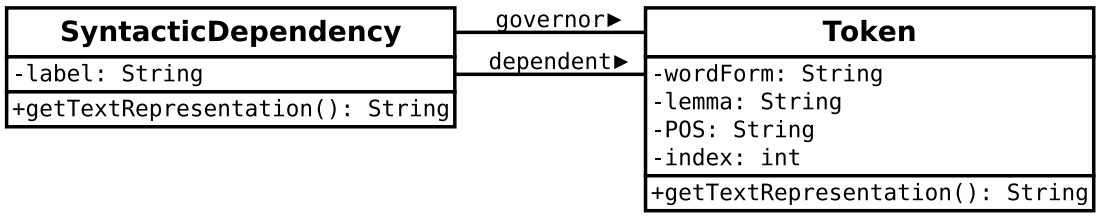
Modélisation

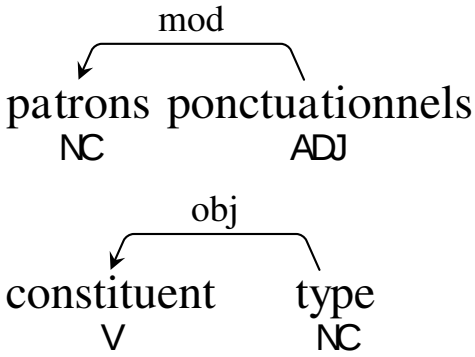




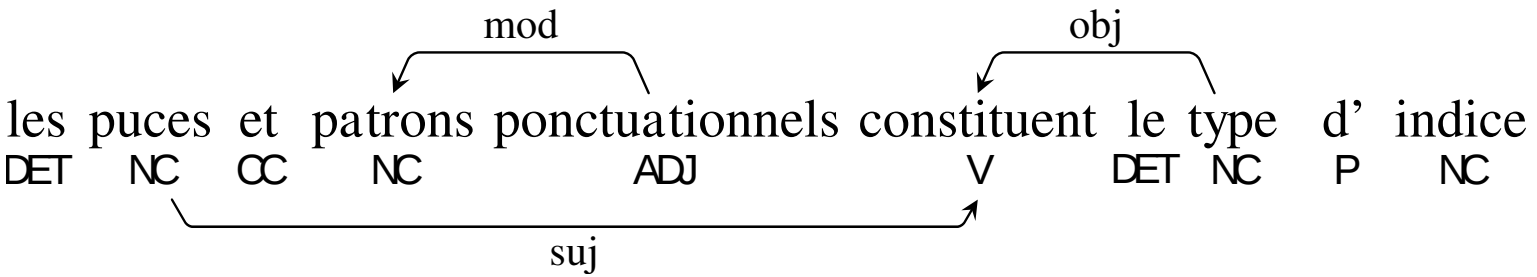
Modélisation

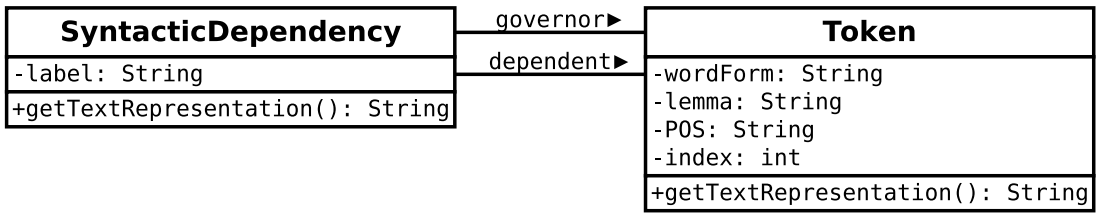


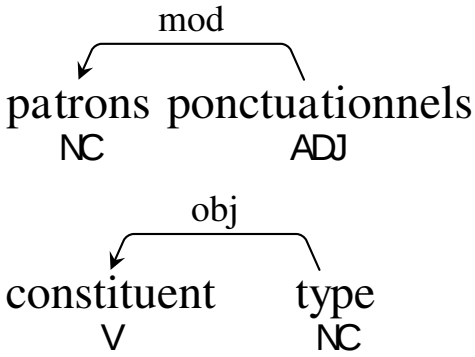




Modélisation





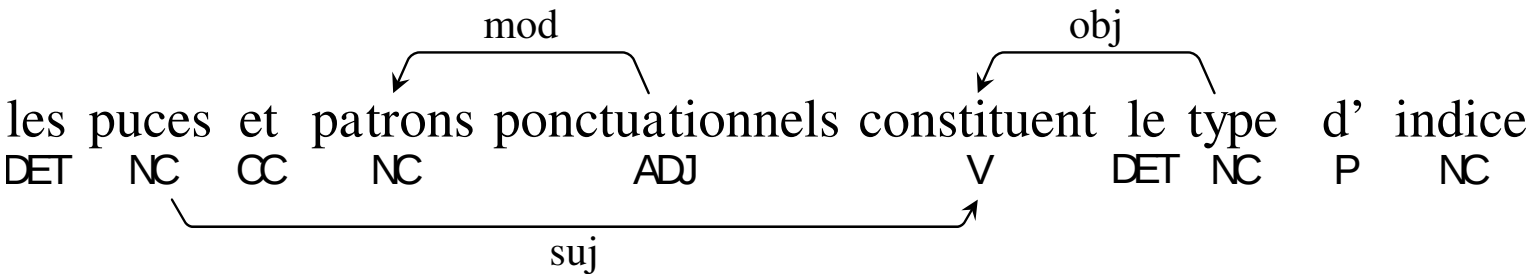


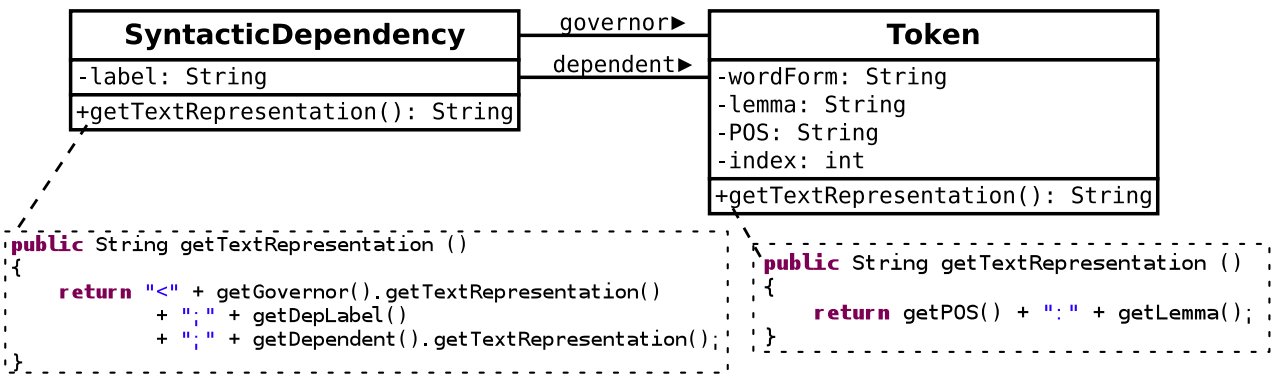
dépendances :

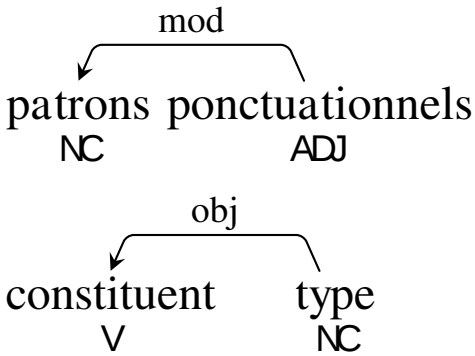
<NC:patron; mod; ADJ:ponctuationnel>

<V:constituer; obj; NC:type>

Modélisation





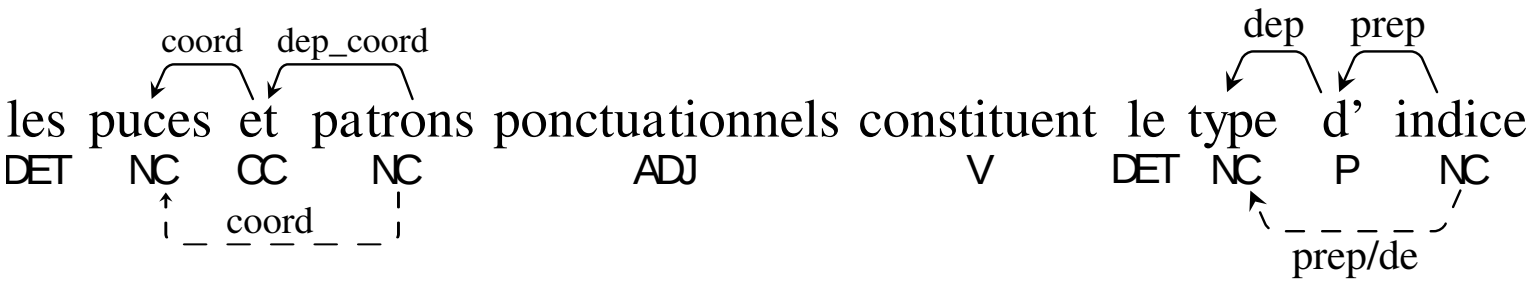


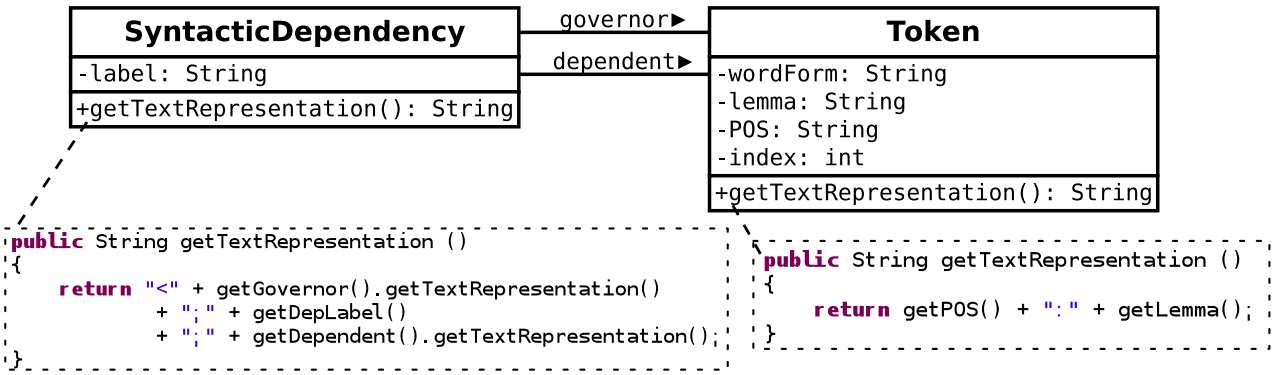
dépendances :

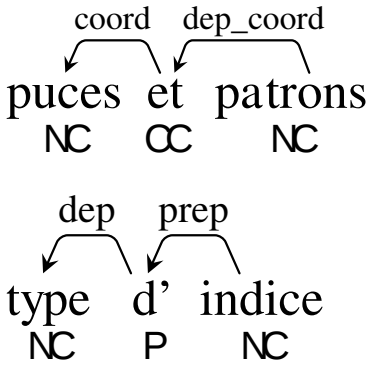
<NC:patron; mod; ADJ:ponctuationnel>

<V:constituer; obj; NC:type>

Modélisation





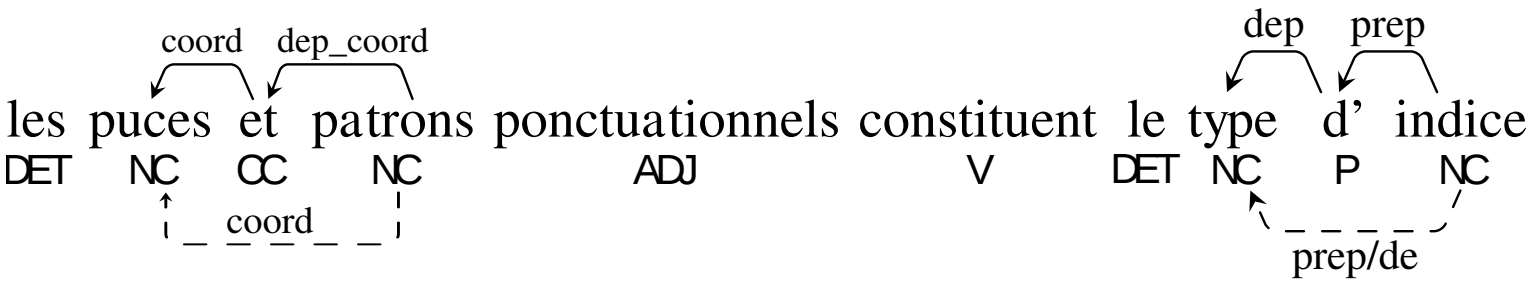


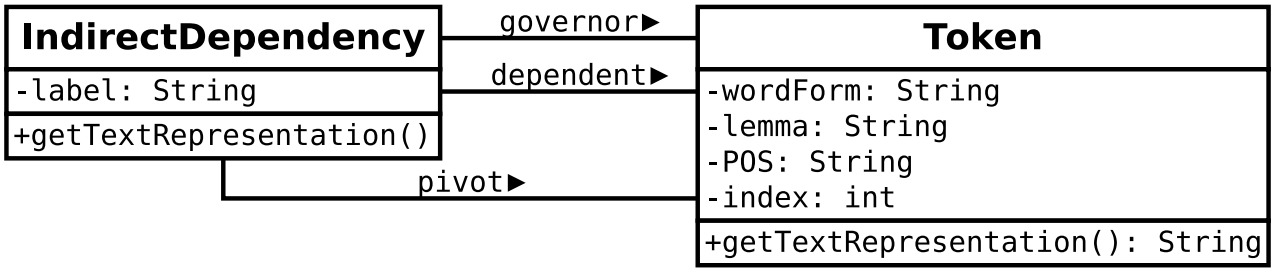
dépendances :

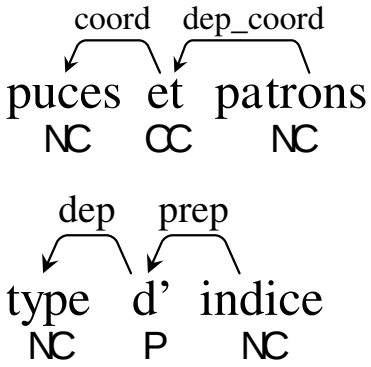
<NC:patron; coord; NC:puce>

<NC:type; prep/de; NC:indice>

Modélisation





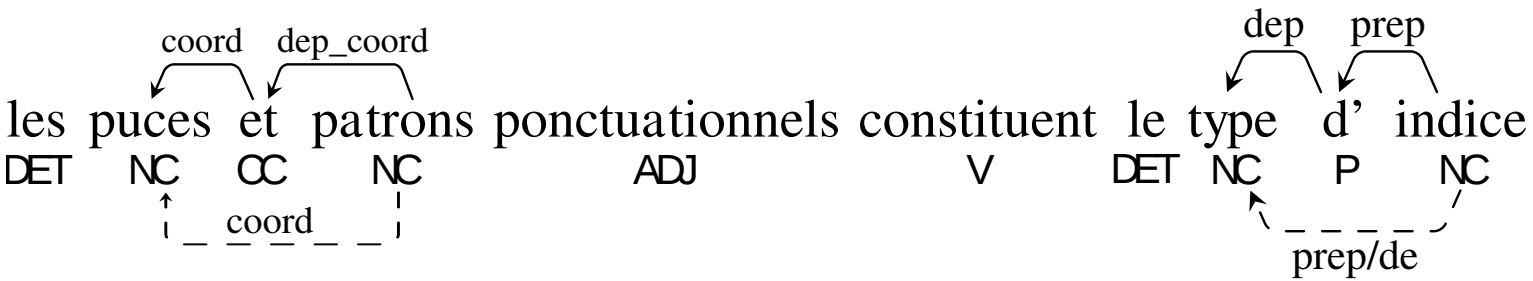


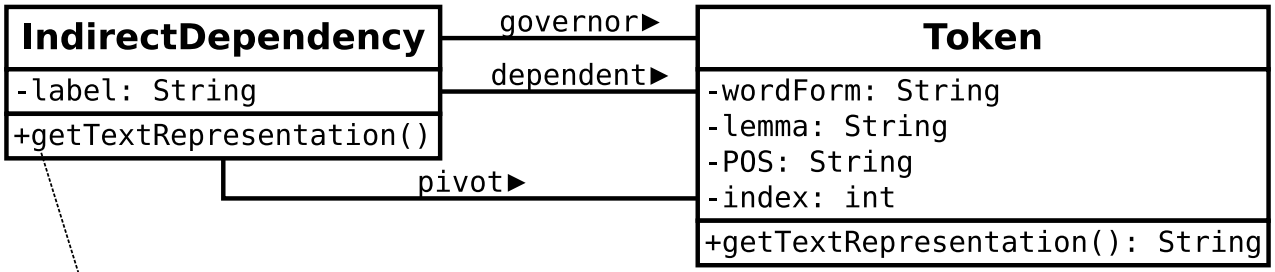
dépendances :

<NC:patron; coord; NC:puce>

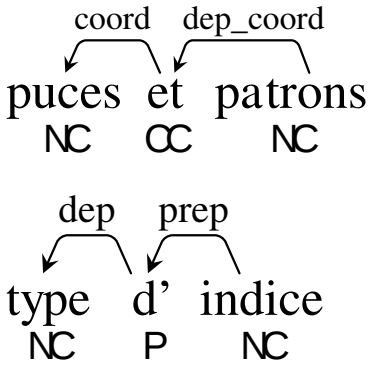
<NC:type; prep/de; NC:indice>

Modélisation






```
public String getTextRepresentation ()
{
    return "<" + getGovernor().getTextRepresentation()
           + ":" + getDepLabel() + "/" + getPivot().getLemma()
           + ":" + getDependent().getTextRepresentation();
}
```

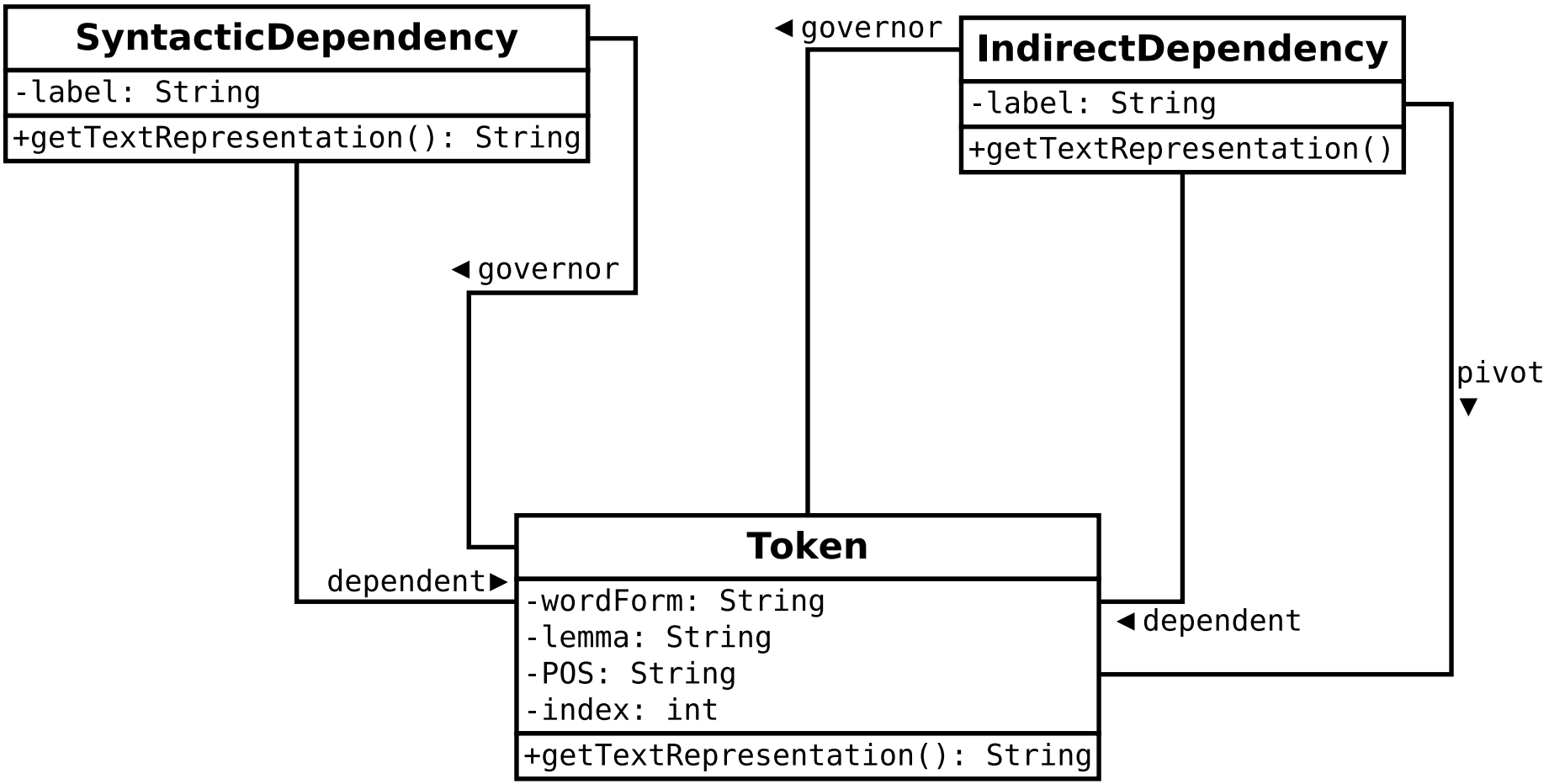


dépendances :

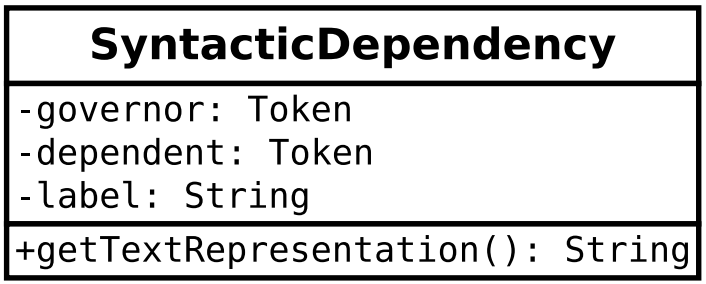
<NC:patron; coord; NC:puce>

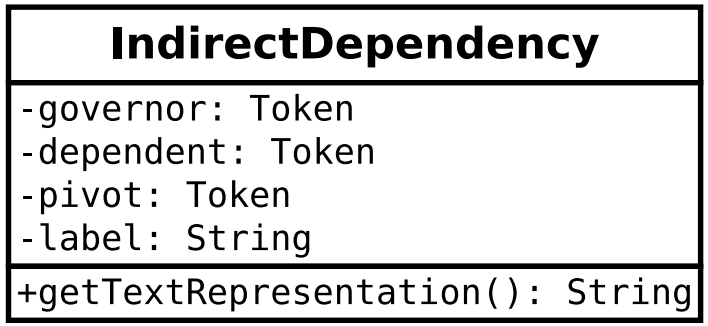
<NC:type; prep/de; NC:indice>

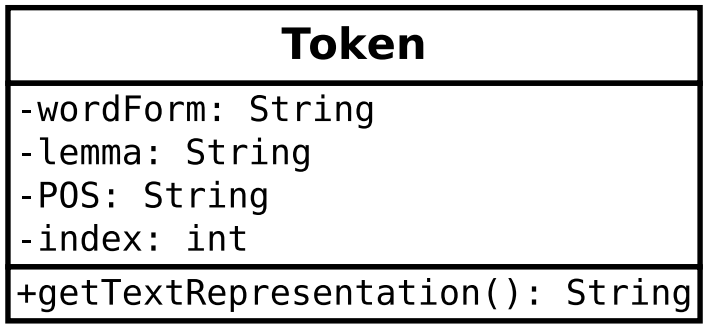
De la modélisation à l'implémentation



De la modélisation à l'implémentation







De la modélisation à l'implémentation

SyntacticDependency

-governor: Token
-dependent: Token
-label: String

+getTextRepresentation(): String

```
public class Token
{
    private String wordForm;
    private String lemma;
    private String POS;

    public String getWordForm()
    {
        return wordForm;
    }

    public void setWordForm(String wf)
    {
        wordForm = wf;
    }

    public void setLemma (String l)
    {
        lemma = l;
    }

    public String getLemma ()
    {
        return lemma;
    }

    public String getPOS()
    {
        return POS;
    }

    public void setPOS(String pOS)
    {
        POS = pOS;
    }

    public String getTextRepresentation ()
    {
        return getPOS() + ":" + getLemma();
    }
}
```

IndirectDependency

-governor: Token
-dependent: Token
-pivot: Token
-label: String

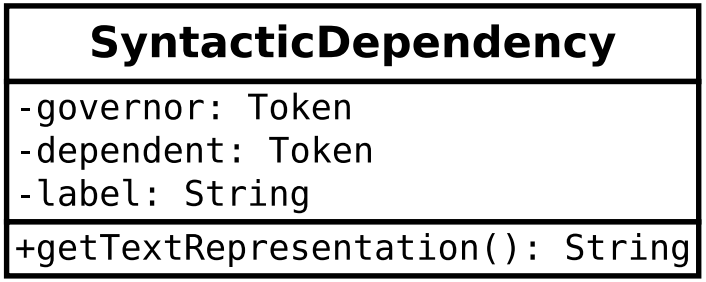
+getTextRepresentation(): String

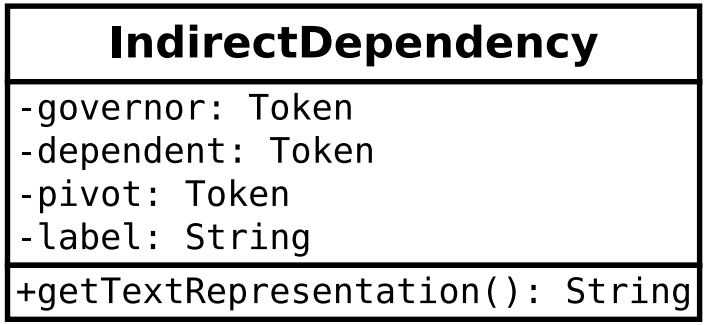
Token

-wordForm: String
-lemma: String
-POS: String
-index: int

+getTextRepresentation(): String

De la modélisation à l'implémentation





De la modélisation à l'implémentation

SyntacticDependency

-governor: Token
-dependent: Token
-label: String
+getTextRepresentation(): String

```
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

IndirectDependency

-governor: Token
-dependent: Token
-pivot: Token
-label: String
+getTextRepresentation(): String

```
public class IndirectDependency
{
    private Token governor;
    private Token dependent;
    private Token pivot;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

De la modélisation à l'implémentation

```java
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
public class IndirectDependency
{
    private Token governor;
    private Token dependent;
    private Token pivot;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

De la modélisation à l'implémentation

```java
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }
    public void setGovernor(Token gov)
    {
        governor = gov;
    }
    public Token getDependent()
    {
        return dependent;
    }
    public void setDependent(Token dep)
    {
        dependent = dep;
    }
    public String getDepLabel()
    {
        return depLabel;
    }
    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }
    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
public class IndirectDependency
{
    private Token governor;
    private Token dependent;
    private Token pivot;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }
    public void setGovernor(Token gov)
    {
        governor = gov;
    }
    public Token getDependent()
    {
        return dependent;
    }
    public void setDependent(Token dep)
    {
        dependent = dep;
    }
    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

Solution : factoriser ?

cf. séance précédente, classe Couleur

oui, mais ici, factoriser quoi ?

Solution : factoriser ?

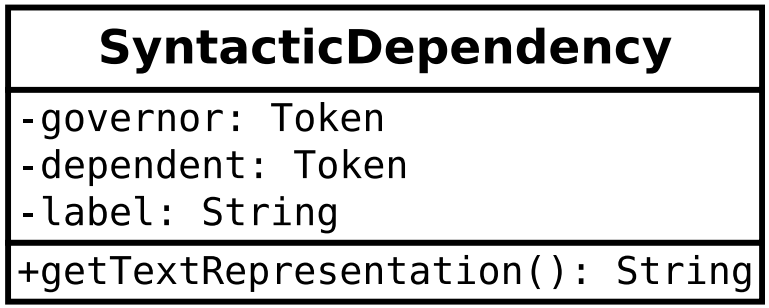
cf. séance précédente, classe Couleur

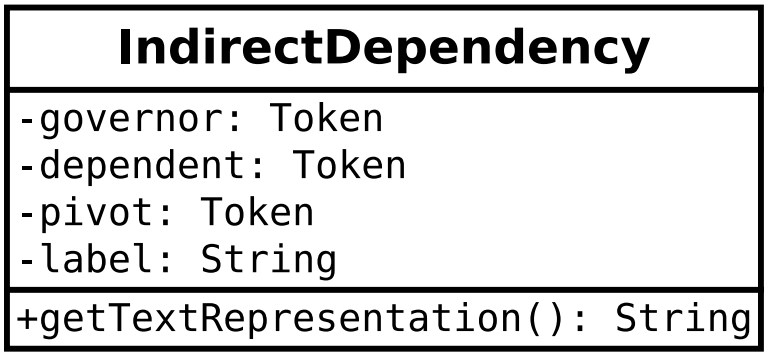
oui, mais ici, factoriser quoi ?
Le fait que les deux types de dépendance aient 3 attributs, gouverneur, dépendant et relation ?

Solution : factoriser ?

cf. séance précédente, classe Couleur

oui, mais ici, factoriser quoi ?
Le fait que les deux types de dépendance aient 3 attributs, gouverneur, dépendant et relation ?
Si on externalise...

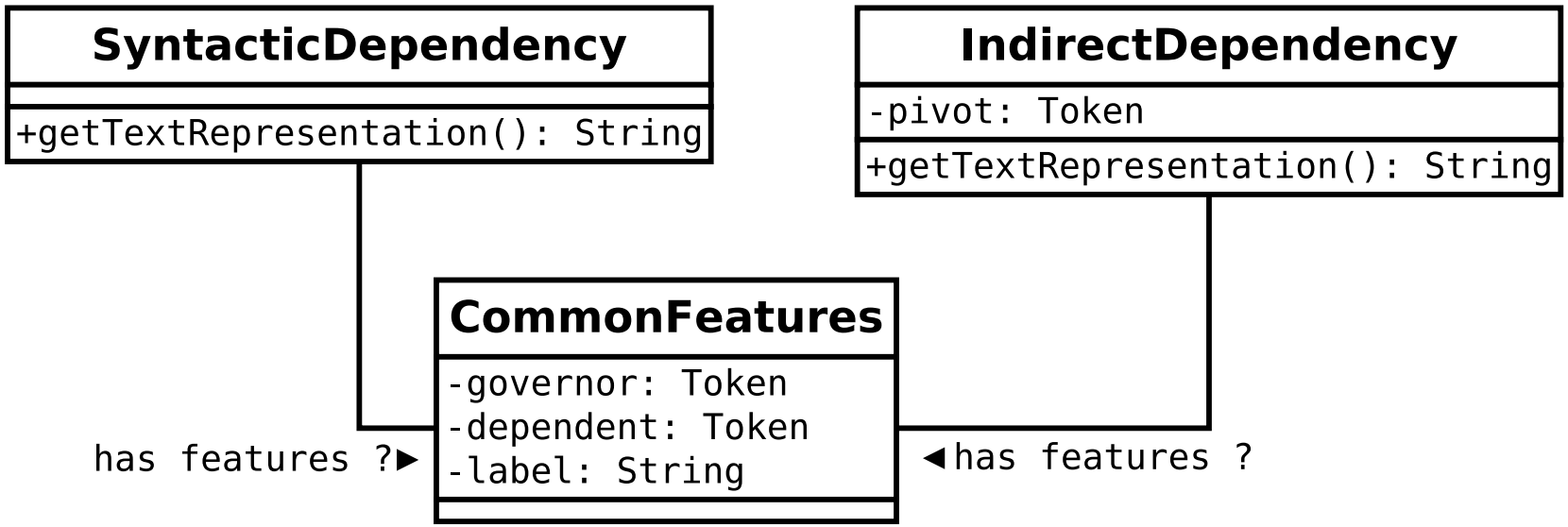




Solution : factoriser ?

cf. séance précédente, classe Couleur

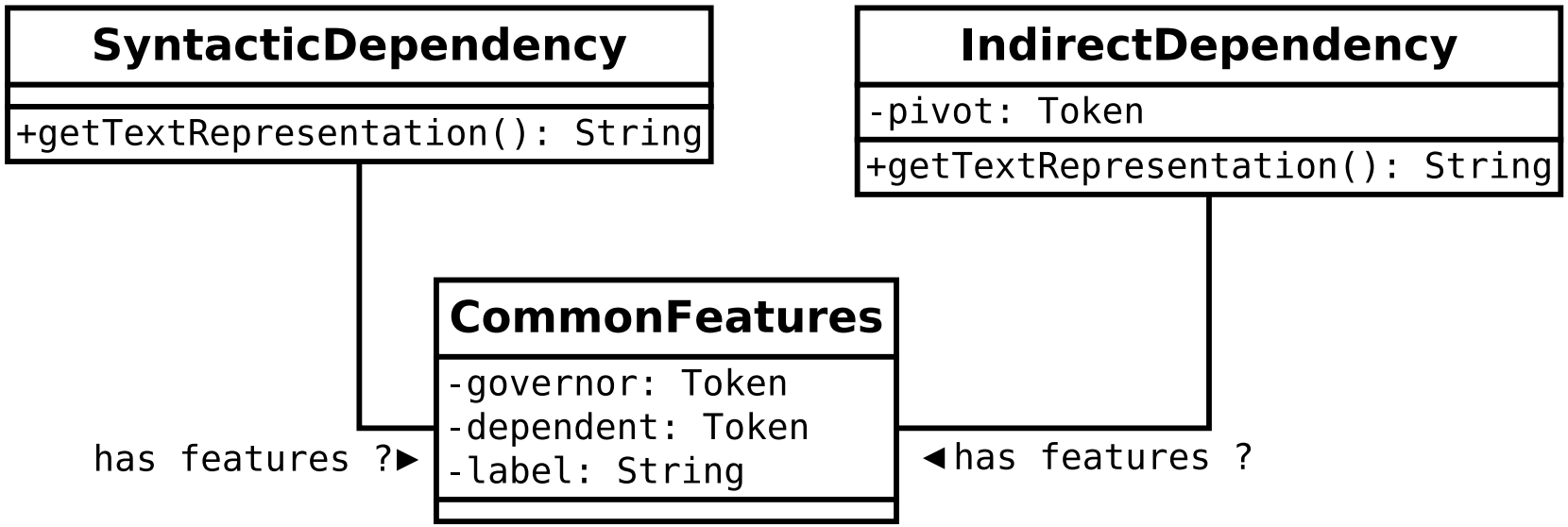
oui, mais ici, factoriser quoi ?
Le fait que les deux types de dépendance aient 3 attributs, gouverneur, dépendant et relation ?
Si on externalise...



Solution : factoriser ?

cf. séance précédente, classe Couleur

oui, mais ici, factoriser quoi ?
Le fait que les deux types de dépendance aient 3 attributs, gouverneur, dépendant et relation ?
Si on externalise...



on externalise l'essence même de la/les classe(s) qu'on veut modéliser.

Que reste-t-il de la classe `SyntacticDependency` ?

Solution : l'héritage

« La classe IndirectDependency, c'est comme la classe SyntacticDependency, sauf qu'elle a un truc en plus »

Solution : l'héritage

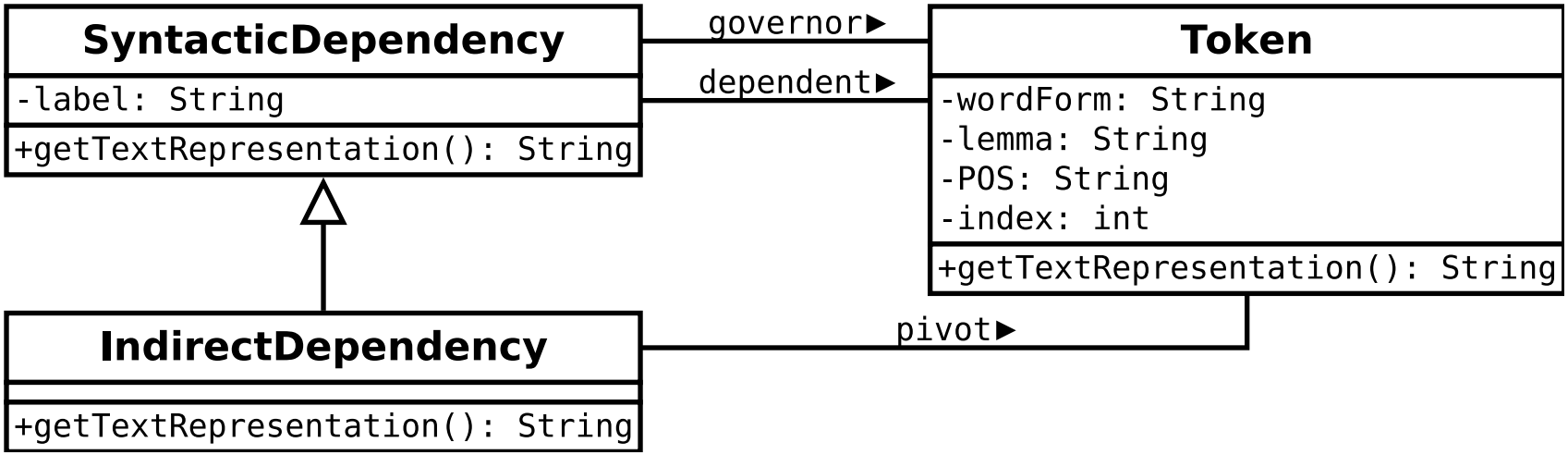
« La classe IndirectDependency, c'est comme la classe SyntacticDependency, sauf qu'elle a un truc en plus »

→ Une *dépendance indirecte* **est une** *dépendance syntaxique* particulière

Solution : l'héritage

« La classe IndirectDependency, c'est comme la classe SyntacticDependency, sauf qu'elle a un truc en plus »

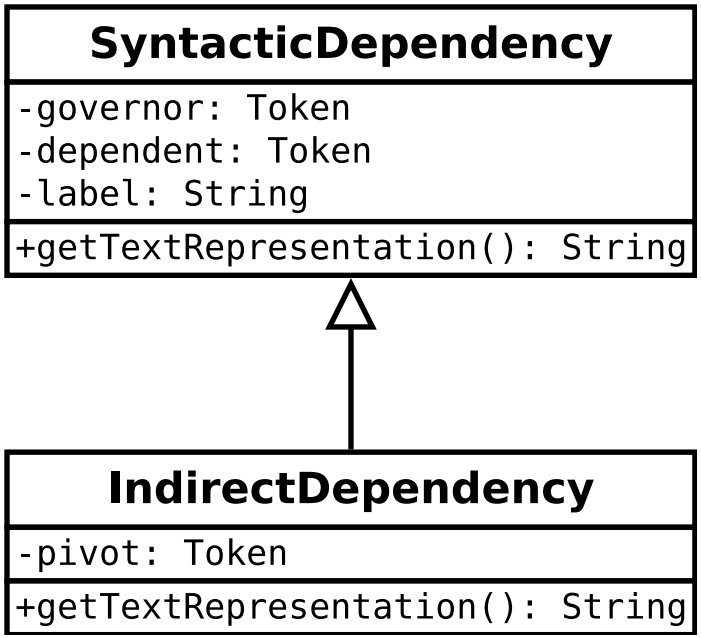
→ Une *dépendance indirecte* **est une** *dépendance syntaxique* particulière



Solution : l'héritage

« La classe IndirectDependency, c'est comme la classe SyntacticDependency, sauf qu'elle a un truc en plus »

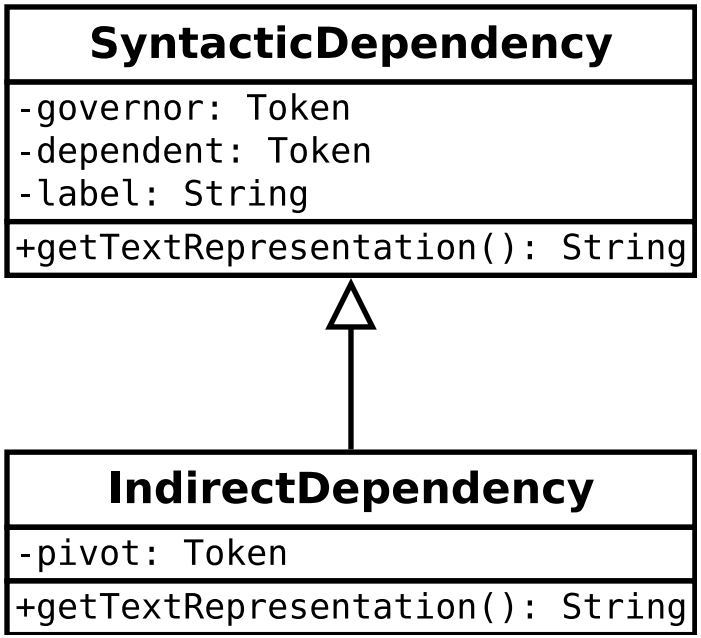
→ Une *dépendance indirecte* **est une** *dépendance syntaxique* particulière



Solution : l'héritage

« La classe IndirectDependency, c'est comme la classe SyntacticDependency, sauf qu'elle a un truc en plus »

→ Une *dépendance indirecte* **est une** *dépendance syntaxique* particulière



L'héritage : une relation de généralisation/spécialisation

- IndirectDependency **hérite de** SyntacticDependency
- IndirectDependency **étend** SyntacticDependency
- IndirectDependency **dérive de** SyntacticDependency
- SyntacticDependency est la **classe mère** de IndirectDependency

Héritage : implémentation

```java
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
public class IndirectDependency extends SyntacticDepency
{
    private Token pivot;

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

Héritage : implémentation

```
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```
public class IndirectDependency extends SyntacticDepency
{
    private Token pivot;

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```
    public static void main (String args[])
    {
        Token prepToken = new Token ();
        prepToken.setLemma("de");
        // ...
        IndirectDependency indDep = new IndirectDependency();
        indDep.setPivot(prepToken);
        indDep.setDepLabel("prep");
    }
```

Héritage : implémentation

```
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```
public class IndirectDependency extends SyntacticDepency
{
    private Token pivot;

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```
    public static void main (String args[])
    {
        Token prepToken = new Token ();
        prepToken.setLemma("de");
        // ...
        IndirectDependency indDep = new IndirectDependency();
        indDep.setPivot(prepToken);
        indDep.setDepLabel("prep");
    }
```

Héritage : implémentation

```java
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
public class IndirectDependency extends SyntacticDepency
{
    private Token pivot;

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
    public static void main (String args[])
    {
        Token prepToken = new Token ();
        prepToken.setLemma("de");
        // ...
        IndirectDependency indDep = new IndirectDependency();
        indDep.setPivot(prepToken);
        indDep.setDepLabel("prep");
    }
```

Héritage : implémentation

```java
public class SyntacticDependency
{
    private Token governor;
    private Token dependent;
    private String depLabel;

    public Token getGovernor()
    {
        return governor;
    }

    public void setGovernor(Token gov)
    {
        governor = gov;
    }

    public Token getDependent()
    {
        return dependent;
    }

    public void setDependent(Token dep)
    {
        dependent = dep;
    }

    public String getDepLabel()
    {
        return depLabel;
    }

    public void setDepLabel(String depLb)
    {
        depLabel = depLb;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
public class IndirectDependency extends SyntacticDepency
{
    private Token pivot;

    public Token getPivot()
    {
        return pivot;
    }

    public void setPivot(Token p)
    {
        pivot = p;
    }

    public String getTextRepresentation ()
    {
        return "<" + getGovernor().getTextRepresentation()
                + ";" + getDepLabel() + "/" + getPivot().getLemma()
                + ";" + getDependent().getTextRepresentation();
    }
}
```

```java
    public static void main (String args[])
    {
        Token prepToken = new Token ();
        prepToken.setLemma("de");
        // ...
        IndirectDependency indDep = new IndirectDependency();
        indDep.setPivot(prepToken);
        indDep.setDepLabel("prep");
    }
```

Héritage : diagramme de classes/diagramme d'objets

Diagramme de classes

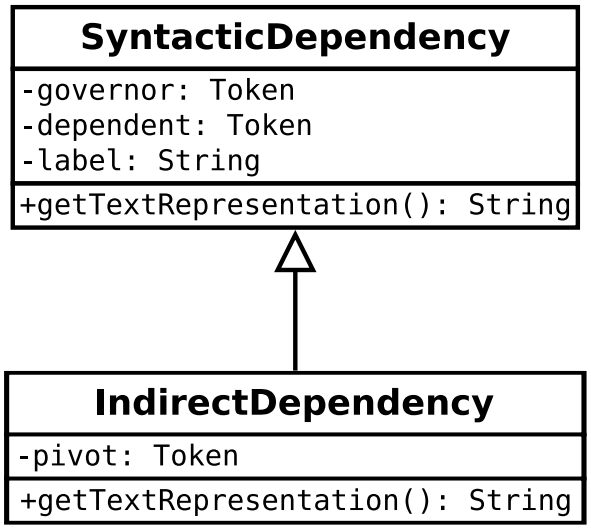
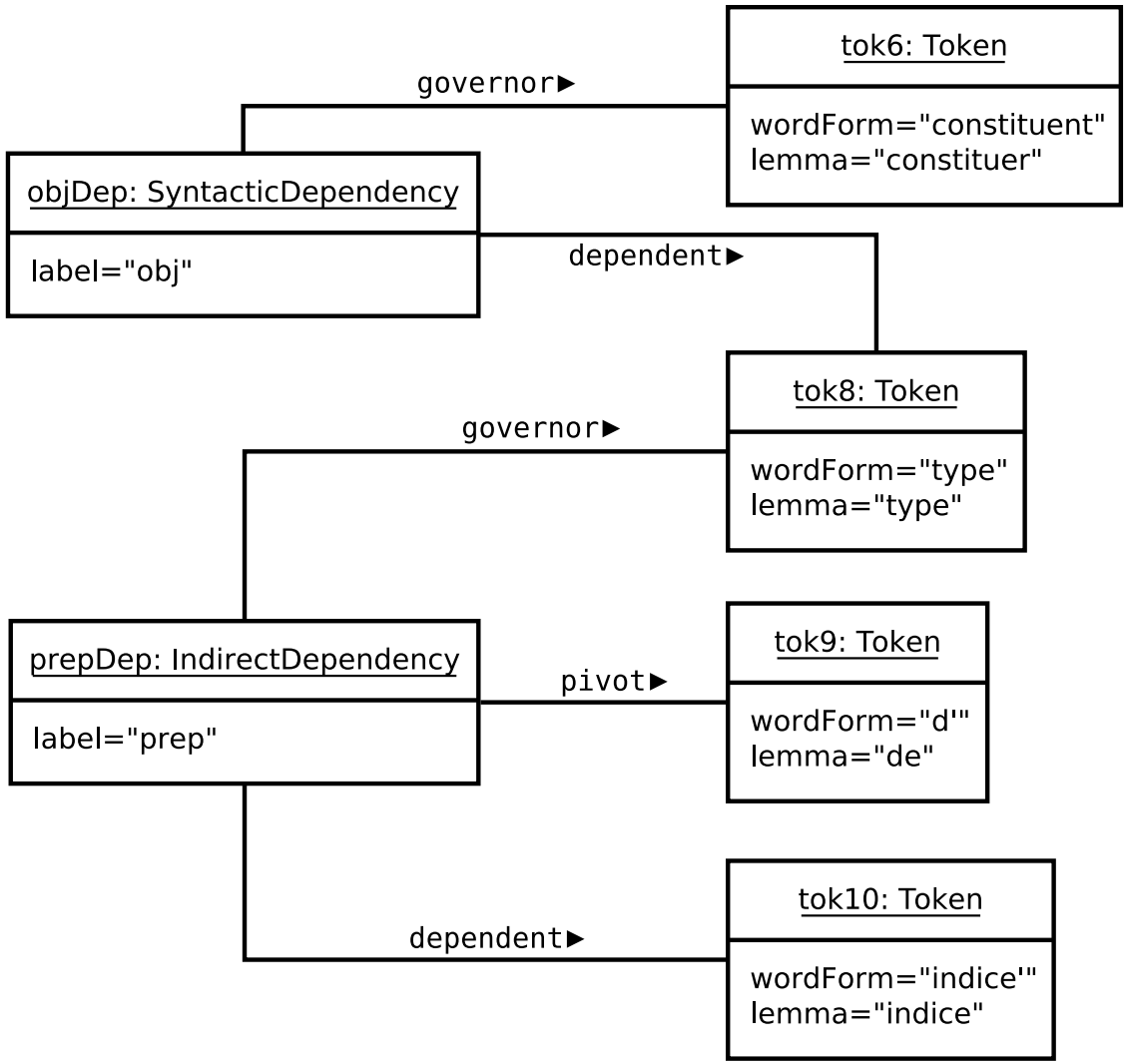
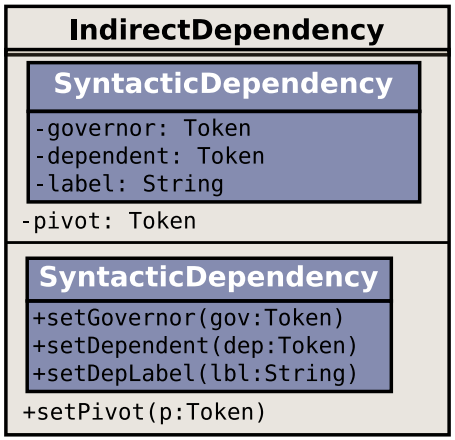


Diagramme d'objets

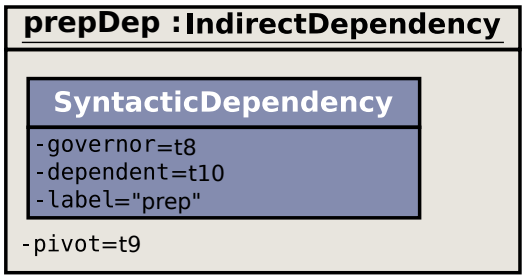


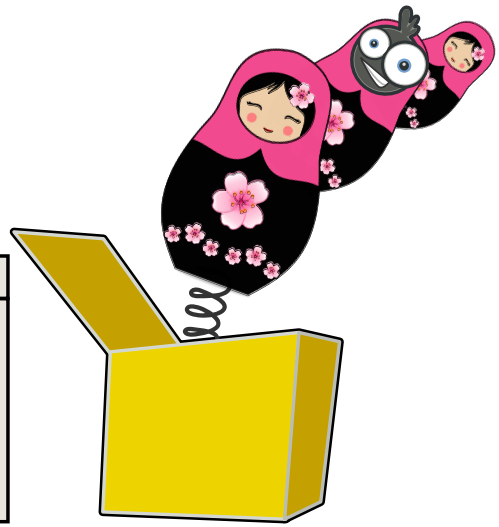
Héritage : poupées russes... ou boîtes à chaussures

classe



objet

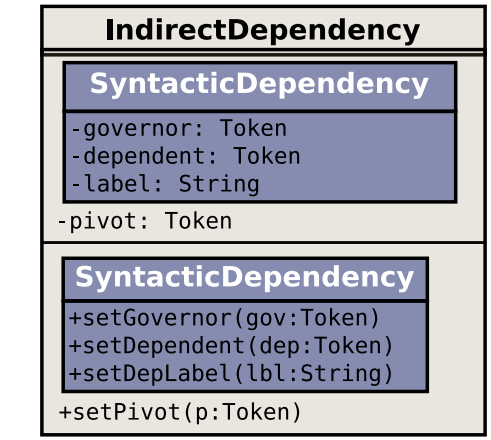




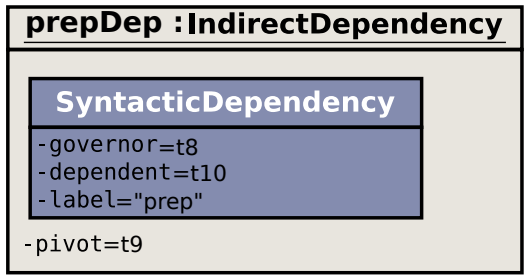
Héritage : poupées russes... ou boîtes à chaussures

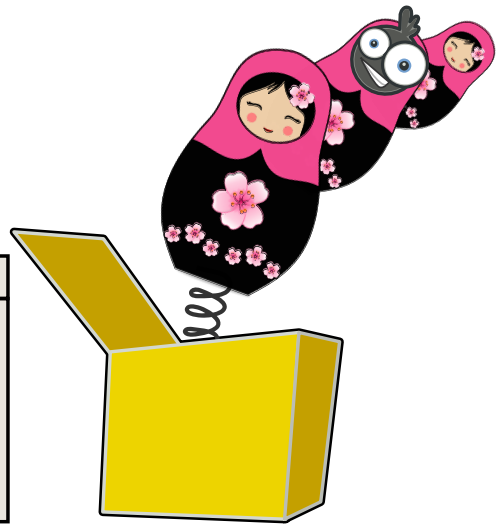
```
IndirectDependency prepDep =
                new IndirectDependency();

prepDep.setGovernor(t8);
prepDep.setDependent(t10);
prepDep.setDepLabel("prep");
prepDep.setPivot(t9);
```

classe



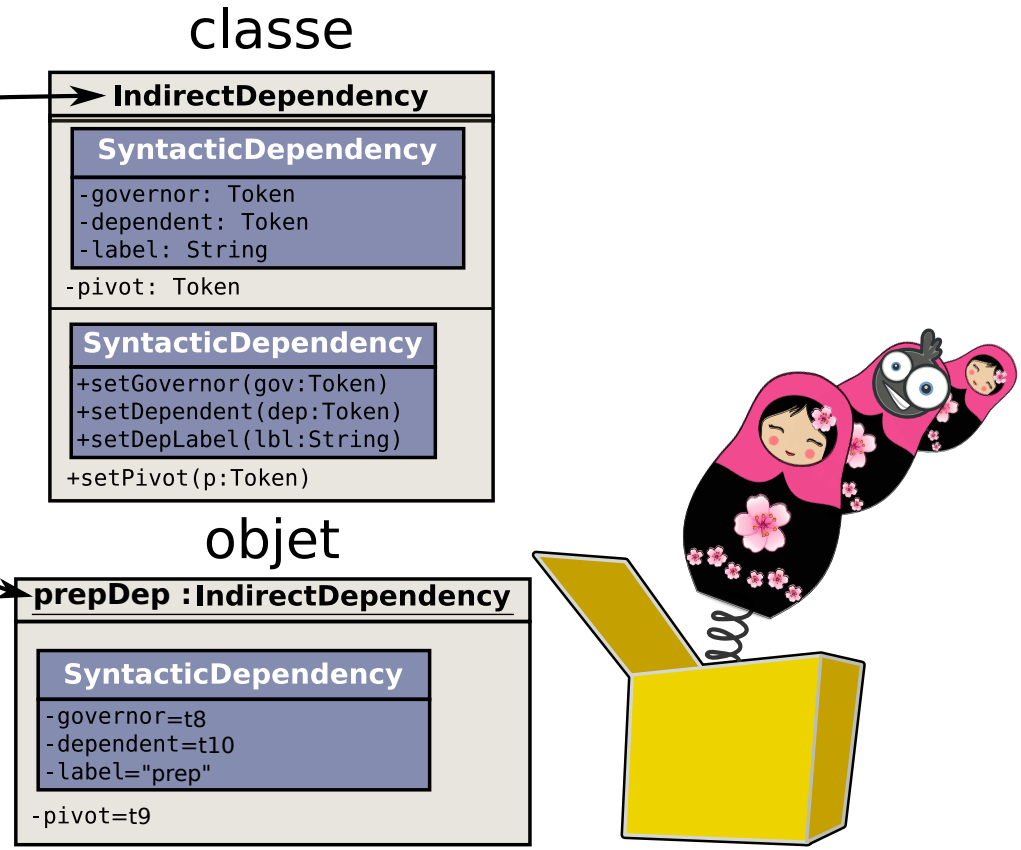
objet





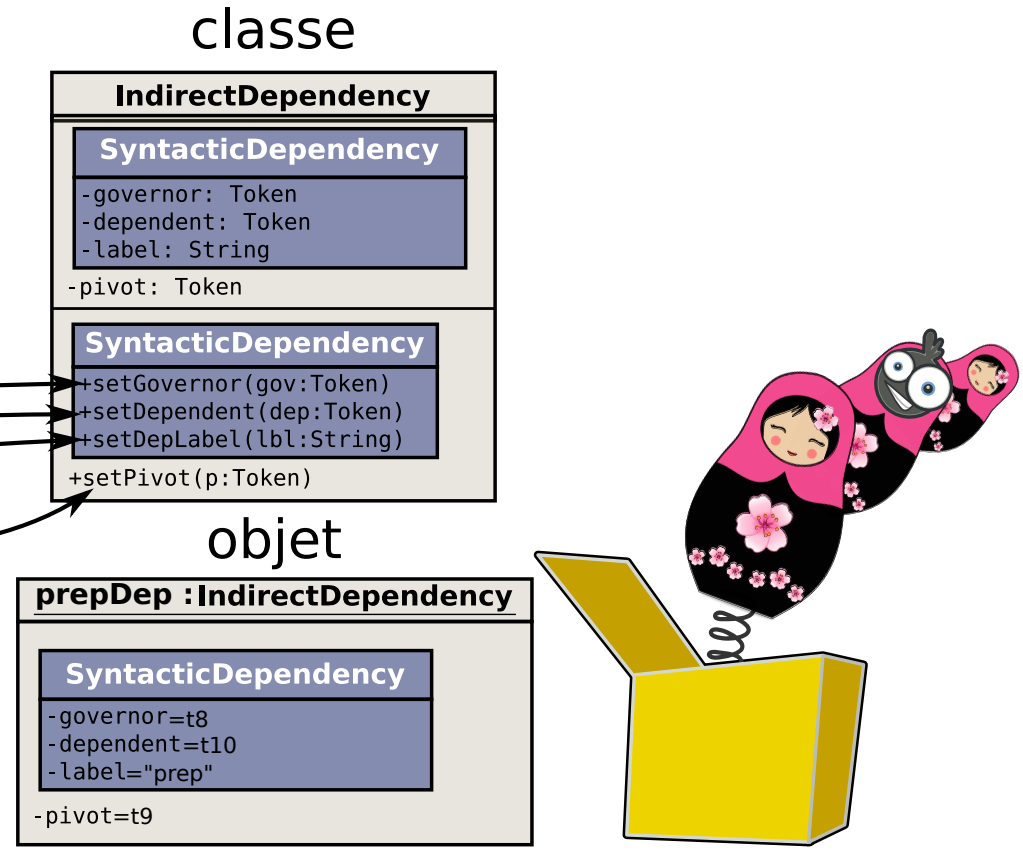
Héritage : poupées russes... ou boîtes à chaussures

```
IndirectDependency prepDep =
                new IndirectDependency();

prepDep.setGovernor(t8);
prepDep.setDependent(t10);
prepDep.setDepLabel("prep");
prepDep.setPivot(t9);
```



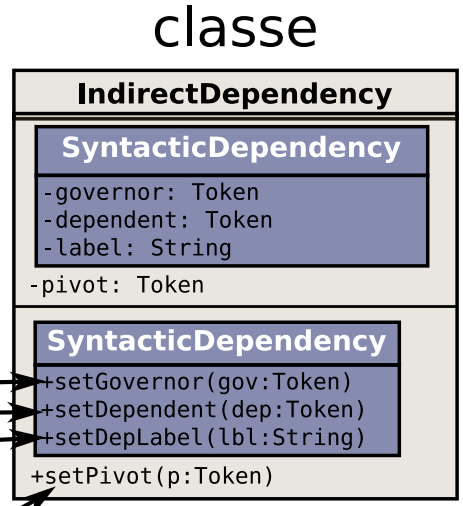
Héritage : poupées russes... ou boîtes à chaussures

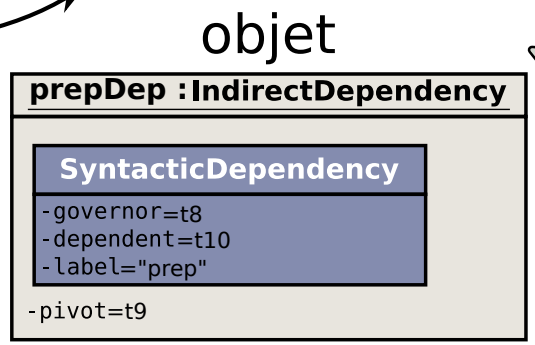
```
IndirectDependency prepDep =
                new IndirectDependency();

prepDep.setGovernor(t8);
prepDep.setDependent(t10);
prepDep.setDepLabel("prep");
prepDep.setPivot(t9);
```



Héritage : poupées russes... ou boîtes à chaussures

```
IndirectDependency prepDep =
                new IndirectDependency();

prepDep.setGovernor(t8);
prepDep.setDependent(t10);
prepDep.setDepLabel("prep");
prepDep.setPivot(t9);
```





encore un schéma... peu académique