

**R3.02**

# **Développement efficace**

---

**Cours 3 – arbres n-aires**










**Hervé Blanchon**

Université Grenoble Alpes

IUT 2 – Département Informatique

# Plan du cours

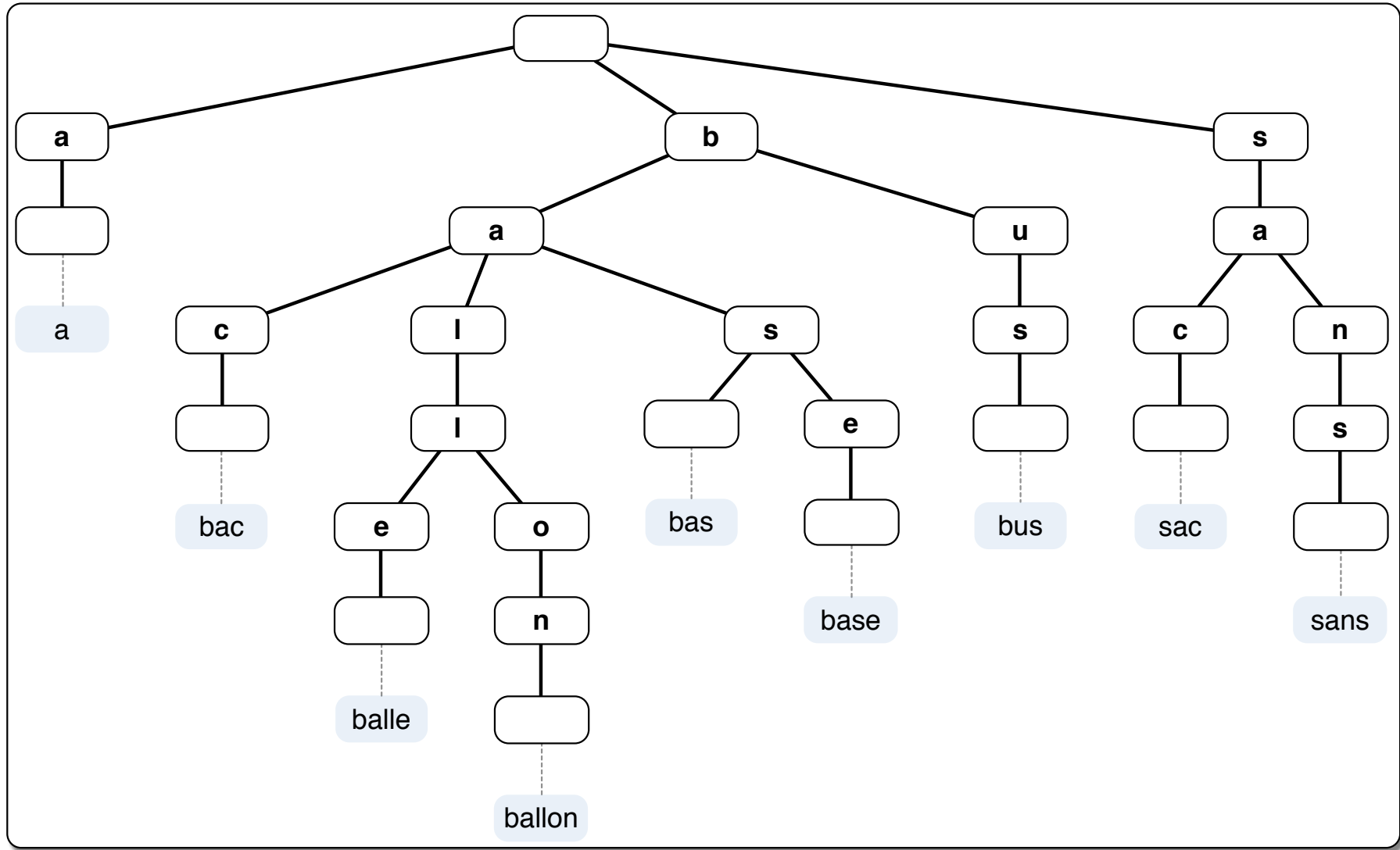
---

-  Notion d'arbre n-aire
-  Formalisation et représentation
-  Représentation persistante et en mémoire
-  Manipulation de document XML sous forme d'arbre n-naire
-  La classe Mondial
-  Exemples de méthodes codées en **itératif**
  -  Compter le nombre d'aéroports
  -  Afficher tous les codes d'identification des pays
  -  XMLElement d'un pays identifier par son nom

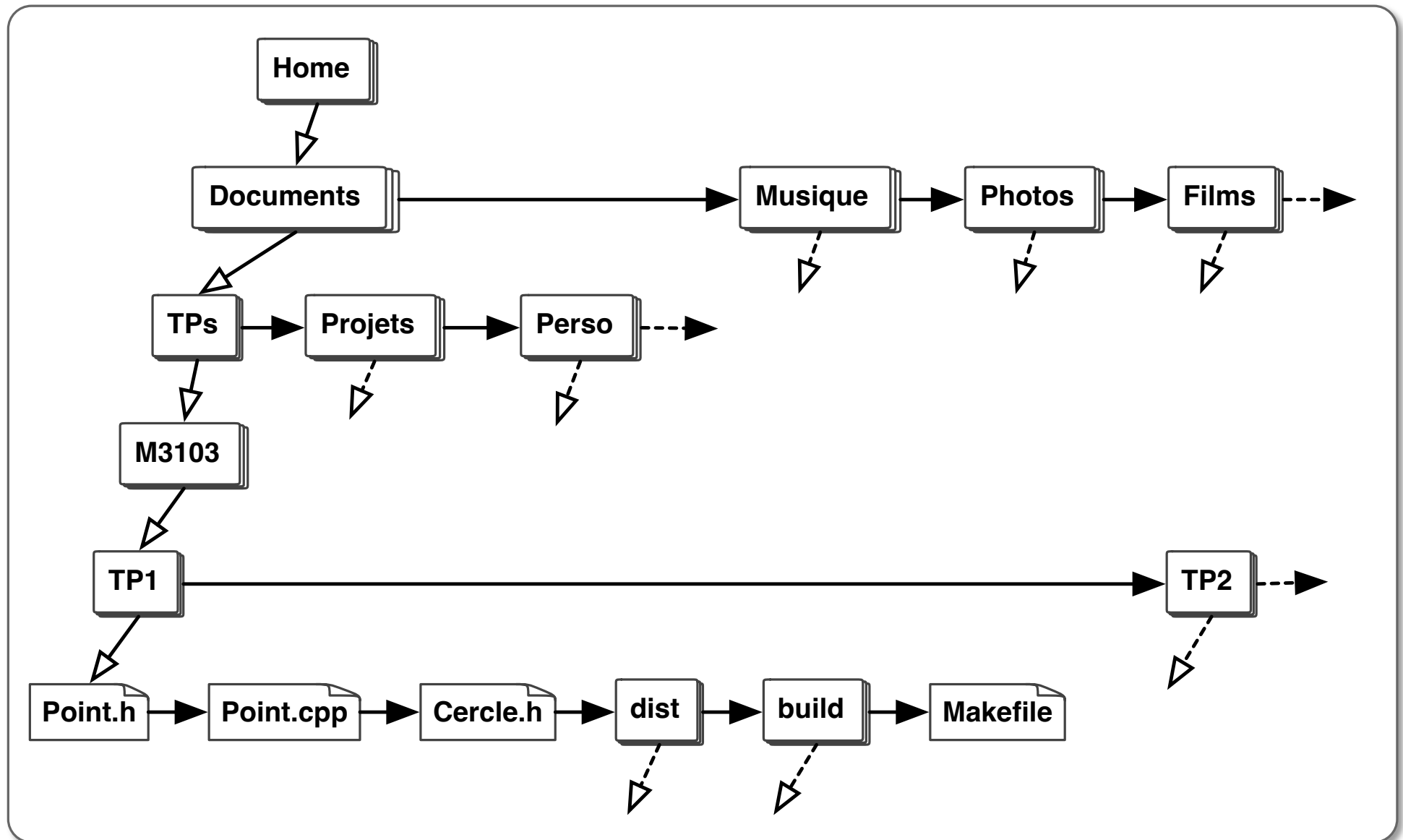
Où je comprends l'intérêt des arbres à travers quelques exemples !

## **NOTION D'ARBRE N-AIRE**

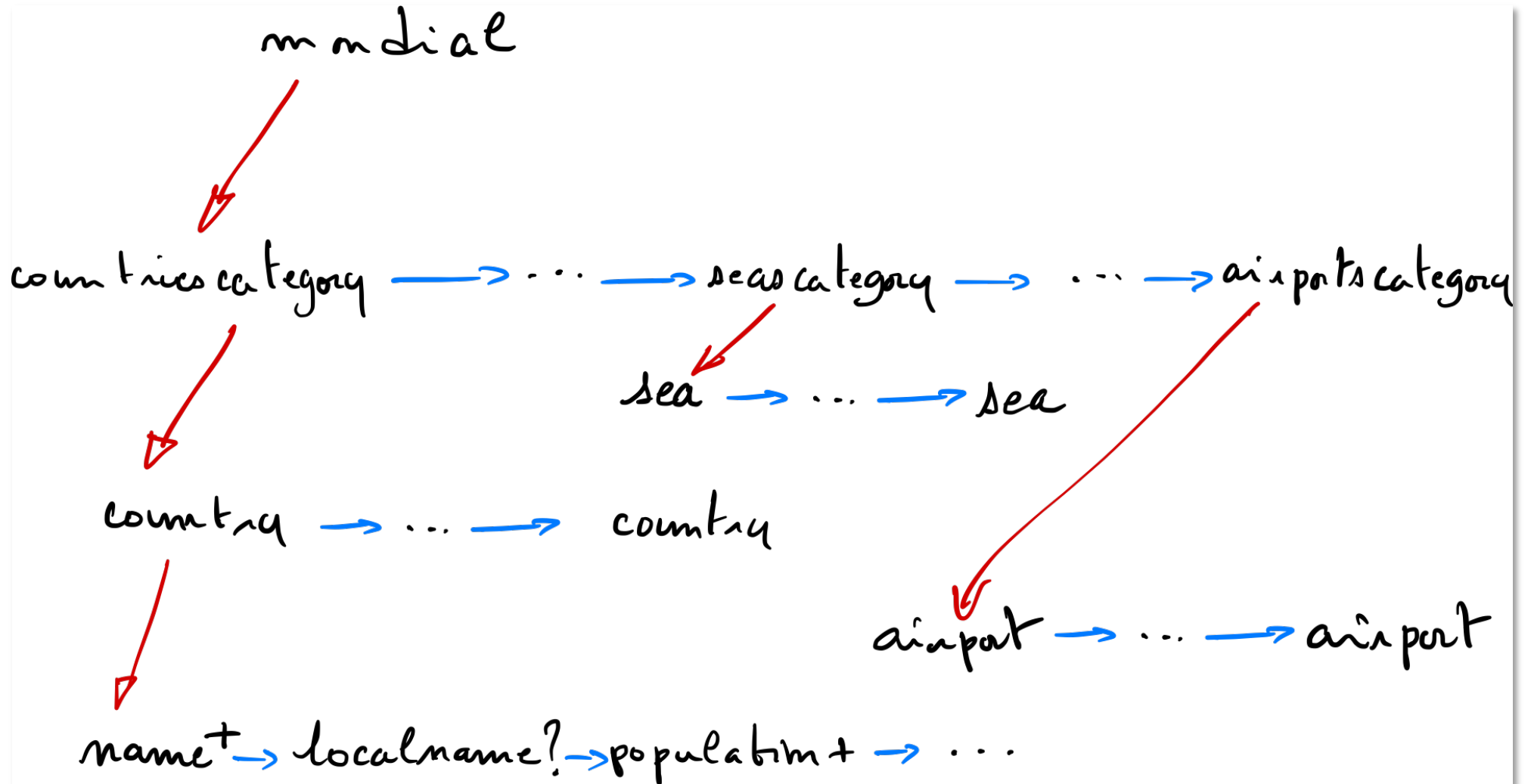
# Un dictionnaire



# Hiérarchie de dossiers/documents



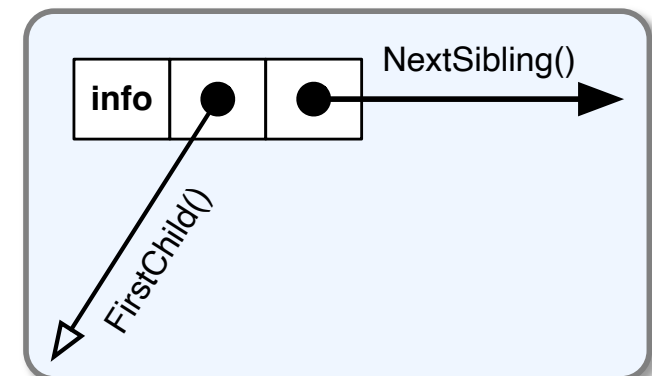
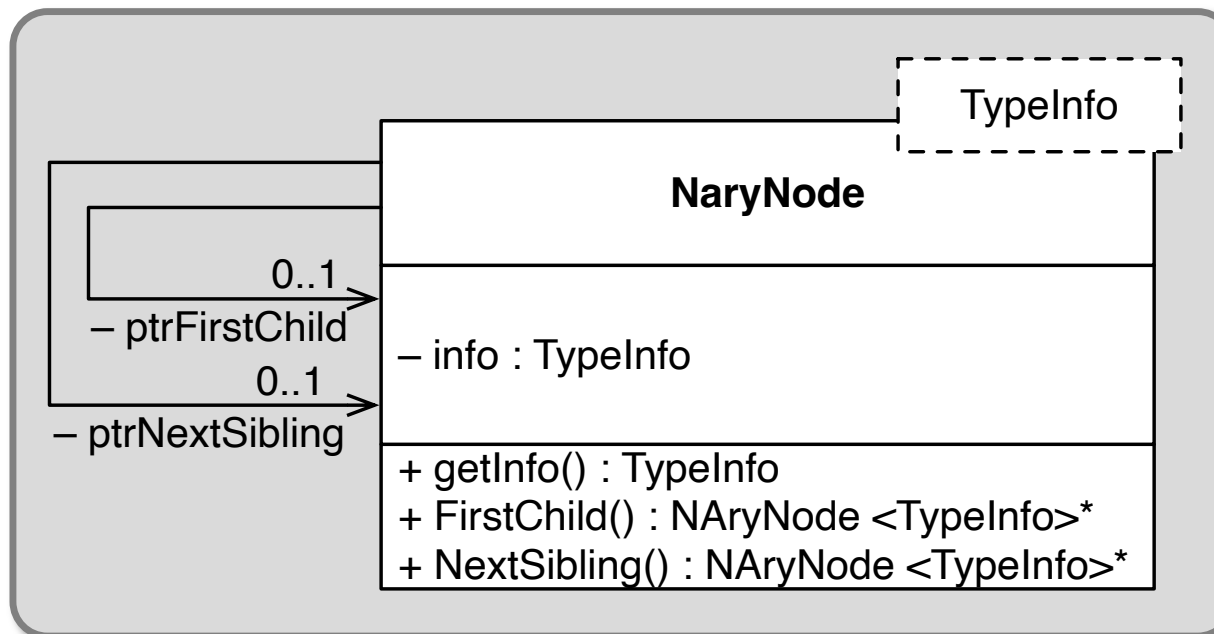
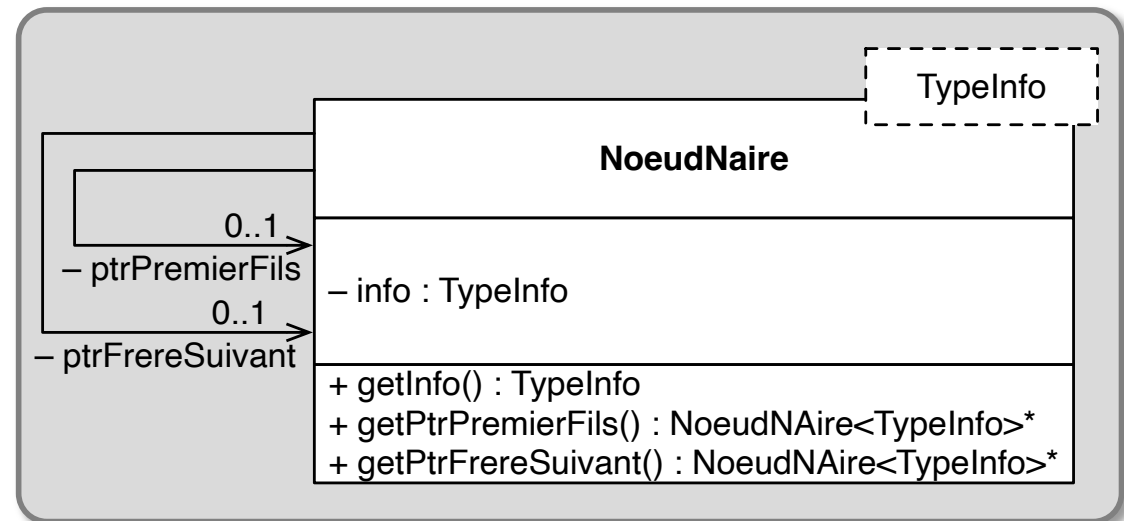
# Des données géographiques



# FORMALISATION & REPRÉSENTATION

# La classe nœud n-aire

- Une information
- Un premier fils
- Un frère suivant














un fichier xml

# **REPRÉSENTATION PERSISTANTE ET EN MÉMOIRE**

# XML en bref

---

-  XML (eXtensible Markup Language) est un langage de balisage généraliste comme HTML
-  Un document XML est constitué d'éléments
  -  cf. planche suivante
-  L'organisation, la hiérarchie et le contenu des éléments sont définis dans un fichier de spécification .dtd (document type definition)
  -  une dtd définit le vocabulaire et la structure d'un document
  -  on peut définir des types de document
    -  catalogue de produits
    -  données géographiques, ...
  -  cf. planche suivante

# Élément XML

---

## Élément « encadré » par des balises

```
<Nom_de_l_Element>  
    contenu de l'élément  
</Nom_de_l_Element>
```

- `<Nom_de_l_Element>` s'appelle la balise ouvrante (de début) de l'Élément,
- `</Nom_de_l_Element>` s'appelle la balise fermante (de fin) de l'Élément.

## Exemples :

 `<localnames>la République française </localname>`




 `<government>republic</government>`

## Élément de balise auto-fermante

```
<Nom_de_l_Element />
```



# Attributs de balise

---

-  Une balise ouvrante ou auto-fermante peut avoir des attributs
-  La valeur d'un attribut est une chaîne de caractères qu'il faudra éventuellement convertir
-  Attributs d'une balise ouvrante

```
<Nom_de_l_Element attr1="val_attr1"  
                  attr2="val_attr2"  
                  ...  
                  attrn="val_attrn">
```

## Exemples

-  `<population year="1990" measured="census">`
-  `<city id="cty-France-22" country="F" province="prov-France-11">`


# Attributs de balise


---

## Attributs d'une balise auto-fermante

```
<Nom_de_l_Element attr1="val_attr1"  
                  attr2="val_attr2"  
                  ...  
                  attrn="val_attrn »/>
```

## Exemples

 `<border country="AND" length="60"/>`

 `<located_at watertype="river" river="river-Rhein"/>`

# Contenu d'un élément...

 ... du **texte** (à convertir éventuellement)



```
<localname>la République française </localname>
```



```
<population measured="census" year="1946">40502513</population>
```

 ...des **éléments** (ce sont ses fils)



```
<city id="cty-France-Strasbourg" country="F" province="prov-France-2">  
  <name>Strasbourg</name>  
  <latitude>48.58</latitude>  
  <longitude>7.75</longitude>  
  <elevation>132</elevation>  
  <population year="1990" measured="census">252338</population>  
  <population year="1999" measured="census">264115</population>  
  <population year="2006" measured="admin.">276867</population>  
  <population year="2011" measured="admin.">272222</population>  
  <located_at watertype="river" river="river-Rhein"/>  
  <located_at watertype="river" river="river-Ill"/>  
</city>
```

# Déclarations dans une dtd


---

 déclaration des éléments : ELEMENT

 déclaration des balises et leur organisation

 *exemples*

 `<!ELEMENT mondial (country*,continent*,organization*,  
sea*,river*,lake*,island*,mountain*,desert*,airport*)>`


 `<!ELEMENT country (name+,localname?,population+,  
population_growth?,infant_mortality?,gdp_total?,  
gdp_agri?,gdp_ind?,gdp_serv?,inflation?,  
unemployment?,(indep_date|dependent)?,  
government?,encompassed*,ethnicgroup*,religion*,  
language*,border*,(province+|city+))>`

 `<!ELEMENT name (#PCDATA)>`

**#PCDATA**  
**chaîne de caractère**

# Déclarations dans une dtd

 déclaration des listes d'attributs : ATTLIST déclaration des attributs des balises *examples*


 <!ATTLIST country car\_code ID #IMPLIED  
 area CDATA #IMPLIED  
 capital IDREF #IMPLIED  
 memberships IDREFS #IMPLIED>



❏ <!ATTLIST population year CDATA #IMPLIED  
measured CDATA #IMPLIED>

## CDATA

### chaîne de caractère



# Opérateurs syntaxiques

---



**symboles** (placés juste après les noms des éléments) :



pas de symbole signifie « 1 et 1 seul élément »



**+** signifie « 1 ou plusieurs éléments », l'élément est donc obligatoire et répétable



**\*** signifie « 0,1 ou plusieurs éléments », l'élément est donc optionnel et répétable



**?** signifie « 0 ou 1 élément », l'élément est donc optionnel



**connecteurs** (placés entre deux éléments) :



**,** les différents éléments séparés par une virgule doivent apparaître dans l'ordre donné



**|** (le pipe = disjonction) signifie qu'un seul des 2 éléments séparés par ce symbole doit apparaître dans le fichier XML



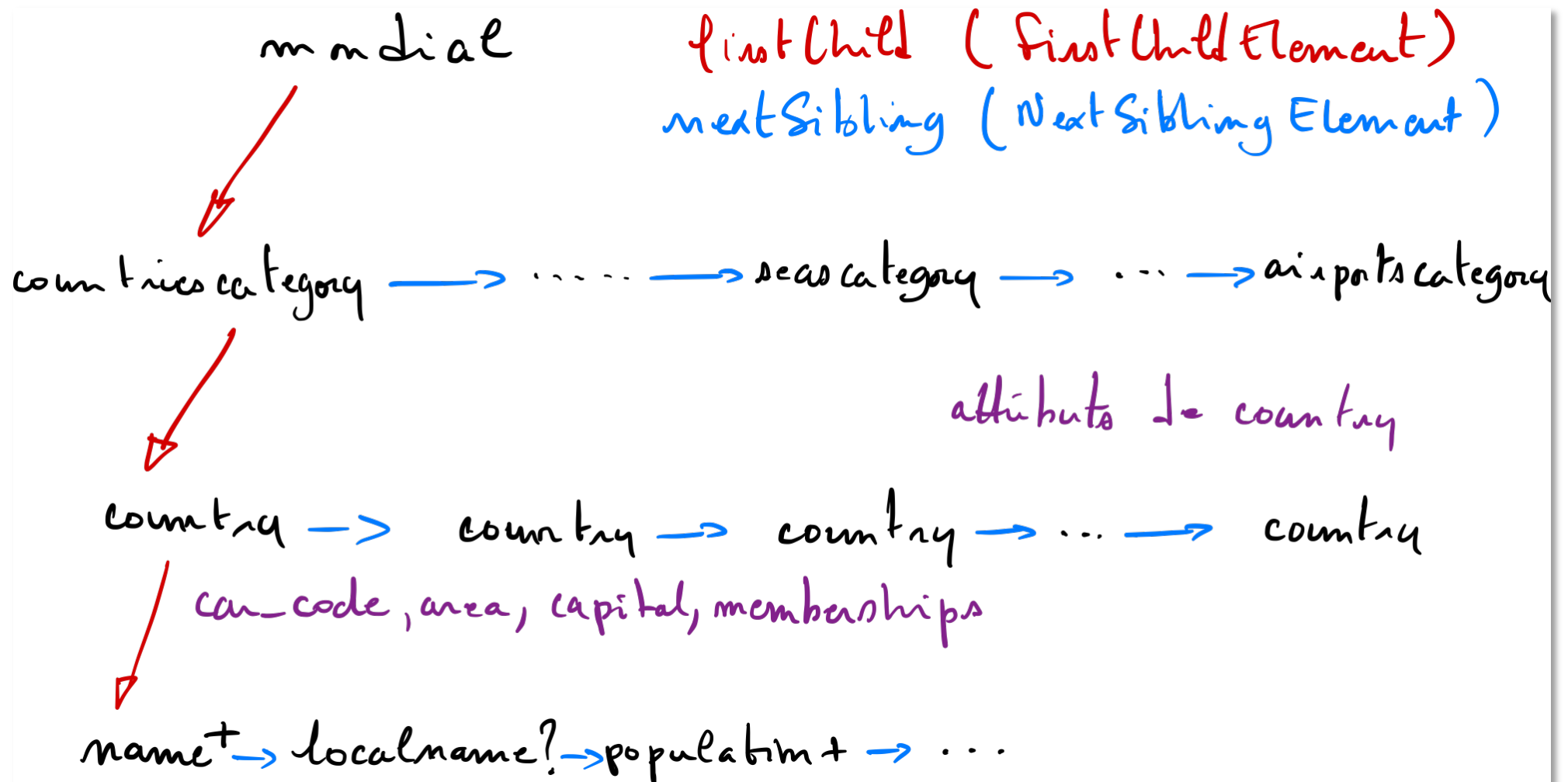
**&** les différents éléments séparés par une esperluette doivent apparaître mais pas forcément dans l'ordre donné

# Notation dtd et représentation XML

- 🌐 L'élément racine des données géographiques est <mondial>, il contient 10 éléments qui sont considérés comme ses fils

Notation DTD	Réalisation XML
<b>&lt;!ELEMENT mondial</b> (countriestcategory, continentcategory, organizationscategory, seascategory, riverscategory, lakescategory, islandscategory, mountaincategory, desertcategory, airportcategory)>	<mondial> <countriestcategory>...</countriestcategory> ... <seascategory>...</seascategory> ... <airportcategory>...</aiportcategory> </mondial>

# Représentation arborescente

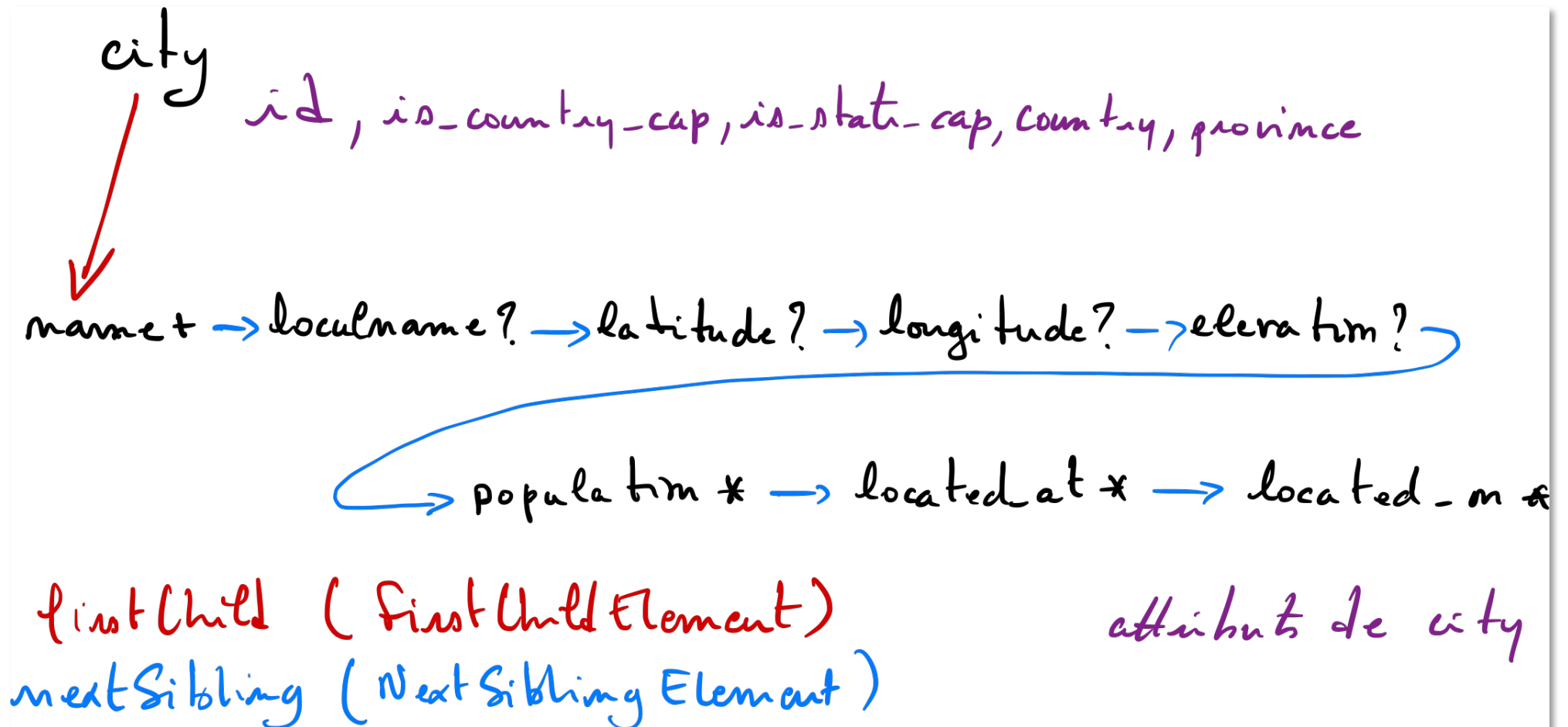


# Notation dtd et représentation XML

## Autre exemple : city

Notation DTD	Réalisation XML
<code>&lt;!ELEMENT city (name+, localname?,latitude?, longitude?,elevation?, popuation*,located_at*, located_on*)&gt;</code>	<pre>&lt;city&gt;   &lt;name&gt;...&lt;/name&gt;           //1 ou plus   &lt;localname&gt;...&lt;/localname&gt; //présent ou non   &lt;latitude&gt;...&lt;/latitude&gt;   //présent ou non   &lt;longitude&gt;...&lt;/longitude&gt; //présent ou non   &lt;elevation&gt;...&lt;/elevation&gt; //présent ou non   &lt;popuation&gt;...&lt;/popuation&gt; //0 ou plus   &lt;located_at/&gt;              //0 ou plus   &lt;located_on/&gt;              //0 ou plus &lt;/city&gt;</pre>

# Représentation arborescente



# **MANIPULATION DE DOCUMENT XML SOUS FORME D'ARBRE N-AIRE**

# Bibliothèque TinyXML

---



Fournit des méthodes pour :



charger en mémoire un fichier XML sous forme d'arbre n-aire



naviguer sur les éléments dans un arbre n-aire



extraire des informations des éléments d'un arbre n-aire



*créer un arbre n-aire*



*modifier un arbre n-aire*



*sauvegarder un arbre n-aire sous forme de fichier XML*

<http://www.grinninglizard.com/tinyxml2/>

# Méthodes utiles de TinyXML

**XML\_Error tinyxml2::XMLDocument::**

**LoadFile(const char\* filename)**

Load an XML file from disk. Returns XML\_NO\_ERROR (0) on success, or an errorID.

**const XMLElement\* tinyxml2::XMLNode::**

**FirstChildElement(const char\* value = 0) const**

Get the first child element, or optionally the first child element with the specified name (value) of this node.

**const XMLElement\* tinyxml2::XMLNode::**

**LastChildElement(const char\* value = 0) const**

Get the last child element or optionally the last child element with the specified name of this node.

**const XMLElement\* tinyxml2::XMLNode::**

**NextSiblingElement(const char\* value=0) const**

Get the next (right) sibling element, or optionally the next sibling element with the specified name (value) of this node.



# Méthodes utiles de TinyXML

```
const char* tinyxml2::
```

```
XMLElement::GetText() const
```

Convenience function for easy access to the text inside an element.

This is a convenient method for getting the text of simple contained text:

```
<foo>This is text</foo>
```

```
    const string str = fooElement->GetText();
```

str will have the value "This is text".

# Méthodes utiles de TinyXML

```
const char* tinyxml2::XMLElement::  
Attribute(const char* name, const char* value = 0) const
```

Given an attribute name, Attribute() returns the value for the attribute of that name, or null if none exists.

For example: `const char* value = ele->Attribute("foo");`

The value parameter is normally null. However, if specified, the attribute will only be returned if the name and value match.

This allow you to write code: `if (ele->Attribute("foo", "bar")) callFooIsBar();`

```
const XMLAttribute* tinyxml2::XMLElement::  
FirstAttribute() const
```

Return the first attribute in the list for this element.

```
const XMLAttribute* tinyxml2::  
XMLAttribute::Next() const
```

The next attribute in the list.

# LA CLASSE MONDIAL

```

class Mondial {
public:
    ...
private:
    // Un XMLDocument permet d'accéder aux services de la bibliothèque
    // Il peut être sauvegardé, chargé, et affiché à l'écran.
    // Si le document est supprimé, tous ses noeuds sont aussi supprimés.
    XMLDocument imageMondial;

    // XMLElement est une classe container.
    // Un élément possède une valeur, un nom et contenir d'autres éléments,
    // du texte, des commentaires, des inconnus
    // Un élément peut avoir un nombre arbitraire d'attributs
    // racineMondial est un élément de la classe <mondial> racine de l'arbre
    XMLElement* racineMondial;

    ...

    // Procédures de découpage des chaînes en mots
    template<typename Out>
    void split(string& s, char delim, Out result) const;

    vector<std::string> split(string& s, char delim) const;
};

```

# Code de la classe non montré ici

---



**Il comporte des méthodes publiques**



elles permettront de répondre aux différentes questions qui font de l'affichage de données demandées



**Il comporte des méthodes privées**



des méthodes de services qui permettent d'atteindre des éléments ciblés dans l'arbre n-aire

exemple 1

# **COMPTER LE NOMBRE D'AÉROPORTS**

# Vers le code

---



## Besoin



compter le nombre d'aéroports



## Où sont les données désirées ?



Atteindre le premier aéroport et les parcourir tous en les comptant



premier fils de `mondial` ( `countriesCategory` )



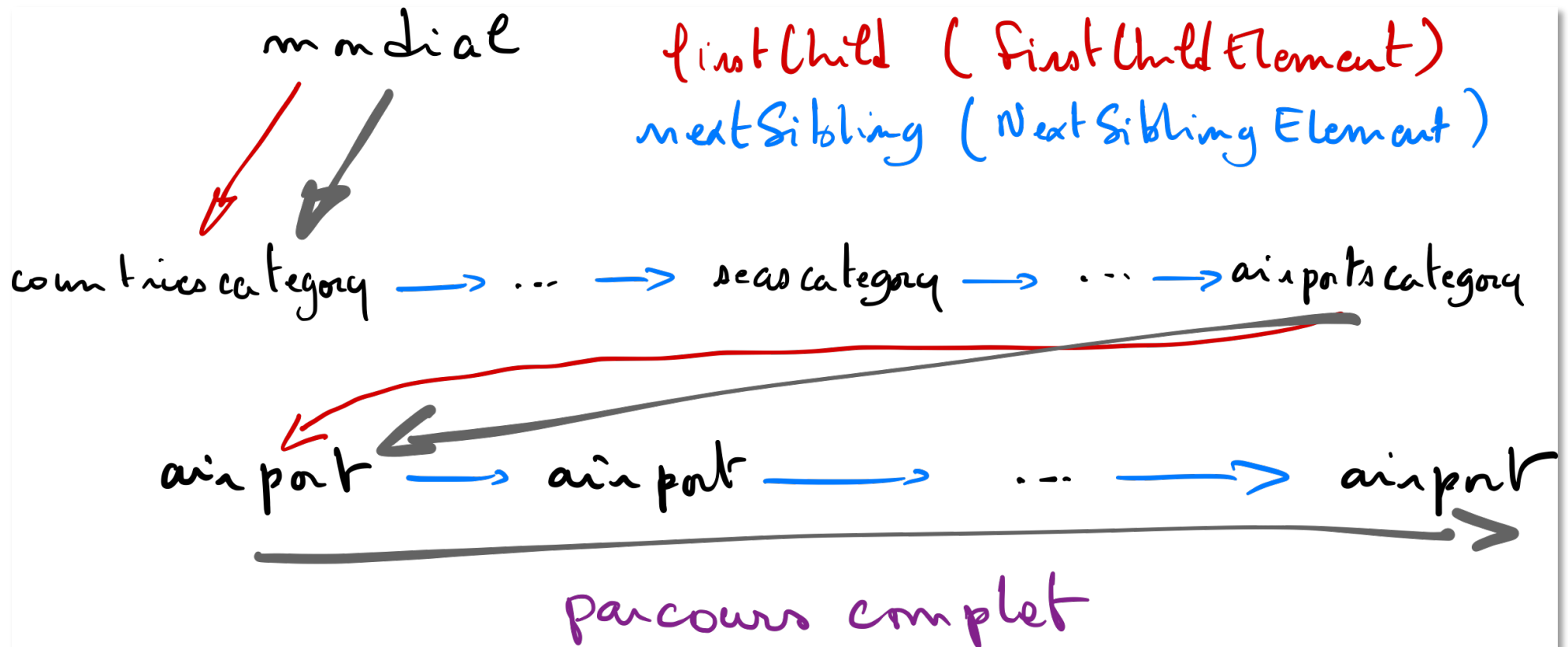
atteindre la catégorie `airportcategory`



parcourir tous les `airports` fils de `airportcategory`

# Vers le code : parcours

- Quel chemin parcourir pour atteindre les données désirées ?





# Le code

```
int Mondial::getNbAirports() const {
    // initialisation du nombre d'aéroports
    int nb = 0;
    // accéder à <airportcategory>, c'est un fils de l'élément <racineMondial>
    XMLElement* airportsCategory =
        racineMondial->FirstChildElement("airportcategory");
    // parcours complet des fils de <airportcategory> en les comptant
    // 1) accéder au premier fils <airport> de <airportcategory>
    XMLElement* currentAirport = airportsCategory->FirstChildElement();
    // 2) parcourir tous les <airport> qui sont des frères
    while (currentAirport != nullptr) {
        // un aéroport supplémentaire
        nb = nb + 1;
        // avancer au frère <airport> suivant de currentAirport
        currentAirport = currentAirport->NextSiblingElement();
    } // currentAirport == nullptr
    // currentAirport n'a plus de frère {}, c'est le dernier
    return nb;
}
```

Exemple 2

**AFFICHER TOUS LES CODES  
D'IDENTIFICATION DES PAYS**

# Vers le code

---



## Besoin



afficher tous les codes d'identifications officiels (`car_code`) des pays



## Où sont les données désirées ?



parcourir tous les country en récupérant la valeur de l'attribut `car_code`



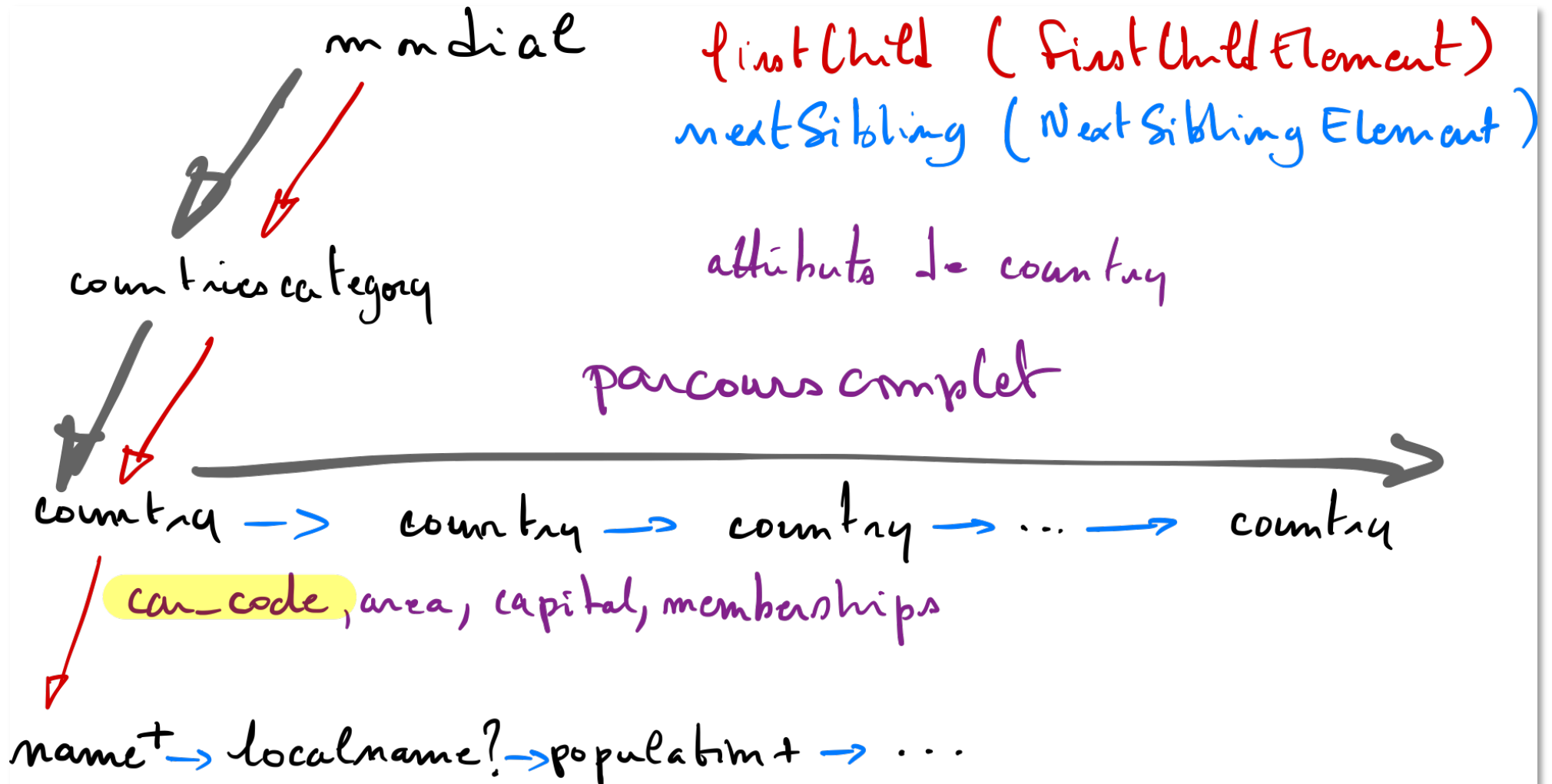
premier fils de mondial (`countriesCategory`)



parcourir tous les country en récupérant la valeur de l'attribut `car_code`

# Vers le code : parcours

- Quel chemin parcourir pour atteindre les données désirées ?



# Le code

```
void Mondial::printCountriesCode() const {
    int rank = 1; // rang du pays
    string carcodeValue; // valeur de l'attribut "car_cod" du pays courant
    // accéder à <countriescategory>, un fils de l'élément <racineMondial>
    XMLElement* countriesCategory =
        racineMondial->FirstChildElement("countriescategory");
    // parcours complet des fils de <countriescategory> et afficher rang et code
    // 1) accéder au premier fils <country> de <countriescategory>
    XMLElement* currentCountry = countriesCategory->FirstChildElement();
    // 2) parcourir tous les <country> qui sont des frères
    while (currentCountry != nullptr) { // traiter le pays courant
        // 1) récupérer la valeur de l'attribut "car_code »
        carcodeValue = currentCountry->Attribute("car_code");
        // 2) faire l'affichage
        cout << setw(5) << rank << " : " << carcodeValue << endl;
        // avancer au frère <country> suivant de currentCountry
        currentCountry = currentCountry->NextSiblingElement();
        rank = rank + 1; // mettre à jour le rang
    } // currentCountry n'a pas de frère {currentCountry == nullptr}, c'est fini
}
```

Exemple 3

# **XMLEMENT D'UN PAYS IDENTIFIÉ PAR SON NOM**

# Vers le code

---



## Besoin



squelette de la méthode privée itérative qui retourne un pointeur de type `XMLElement*` sur l'élément `<country>` correspondant à un pays identifié par son nom (`countryName`)







## Entête

```
/**
 * élément <country> d'un pays identifié par son nom countryName
 * @param countryName
 * @return pointeur sur l'élément <country> dont la valeur
 *         du fils <name> est égal à countryName, nullptr sinon
 */
XMLElement* Mondial::getCountryXmlElementFromNameIter(
    string countryName) const
```

# Vers le code : squelette

---

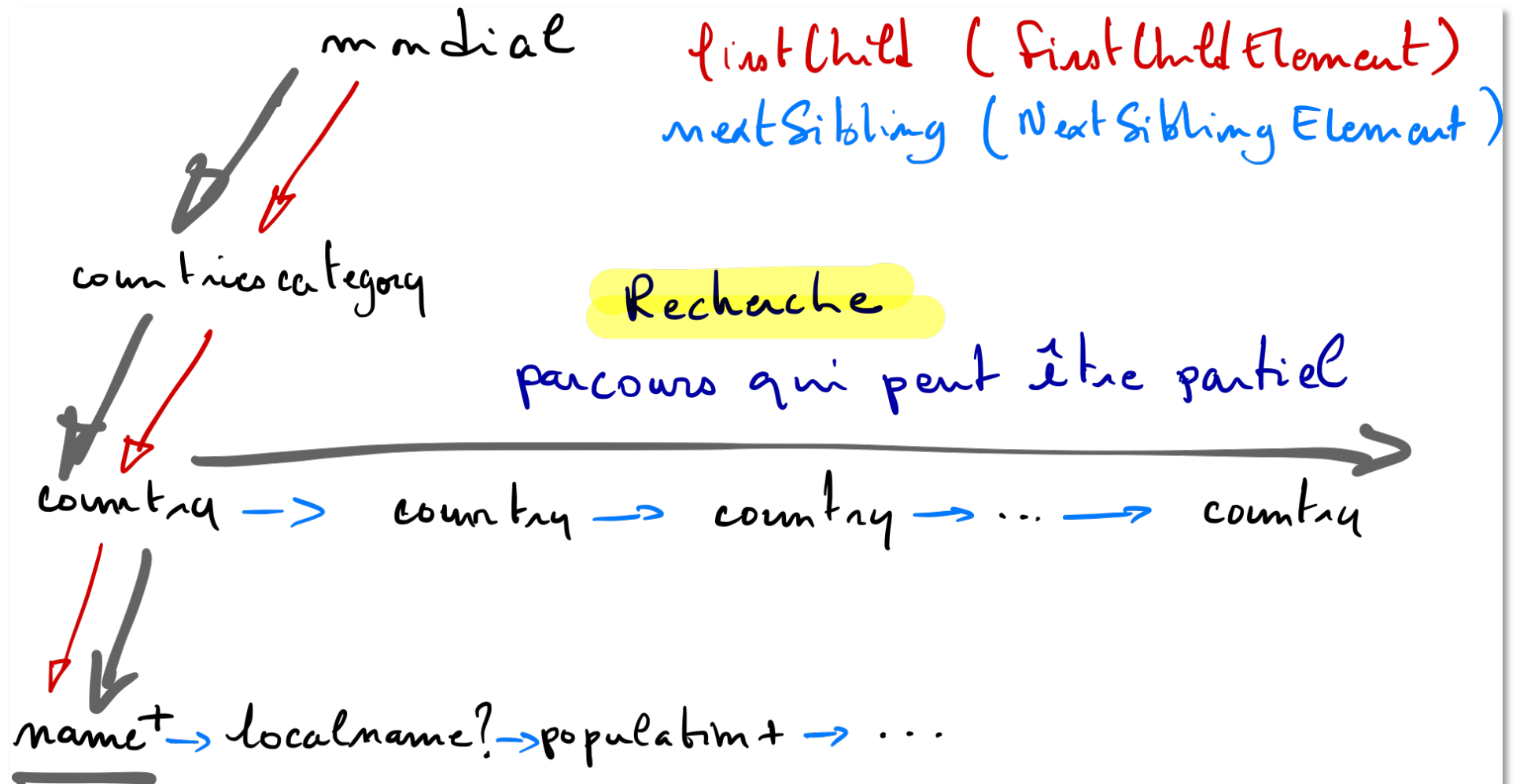
## Où sont les données désirées ?

-  rechercher dans les `country` l'élément dont lequel le fils `name` à la valeur `countryname`
-  premier fils de `mondial` (`countriesCategory`)
-  rechercher dans les `country` celui dont le fils `name` à la valeur `countryname`
-  c'est un parcours qui peut être partiel



# Vers le code : parcours

- Quel chemin parcourir pour atteindre les données désirées ?



# Le code

---

XMLElement\*

```
Mondial::getCountryXmlElementFromNameIter(string countryName) const {  
    // accéder à <countriescategory>, c'est un fils de l'élément <racineMondial>  
    XMLElement* countriesCategory =  
        racineMondial->FirstChildElement("countriescategory");  
    // accéder au premier pays : premier fils <country> de <countriescategory>  
    XMLElement* currentCountry = countriesCategory->FirstChildElement();  
    // parcourir (partiel) les <country> tq le fils <name> ≠ de countryName  
    while ((currentCountry != nullptr)  
        && (currentCountry->FirstChildElement("name")->GetText() != countryName)) {  
        // avancer au frère <country> suivant de currentCountry  
        currentCountry = currentCountry->NextSiblingElement();  
    }  
    // (currentCountry == nullptr) ||  
    // (currentCountry->FirstChildElement("name")->GetText() == countryName)  
    // on retourne l'élément courant dans tous les cas  
    return currentCountry;  
}
```