

---

# Le patron Décorateur

## Sérialisation en java et en XML

jean-michel Douin, douin au cnam point fr  
version : 14 Octobre 2015

**Notes de cours**

---

# Sommaire pour les Patrons

---

- **Classification habituelle**

- **Créateurs**

- Abstract Factory, Builder, Factory Method, Prototype Singleton

- **Structurels**

- Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy

- **Comportementaux**

- Chain of Responsibility. Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

# Les patrons déjà vus en quelques lignes ...

---

- **Adapter**
  - Adapte l'interface d'une classe conforme aux souhaits du client
- **Proxy**
  - Fournit un mandataire au client afin de contrôler/vérifier ses accès
- **Observer**
  - Notification d'un changement d'état d'une instance aux observateurs inscrits
- **Template Method**
  - Laisse aux sous-classes une bonne part des responsabilités
- **Iterator**
  - Parcours d'une structure sans se soucier de la structure interne choisie
- **Composite**
  - Définition d'une structure de données récursives
- **Interpréteur**
  - Un calcul, une interprétation du noeud d'un composite
- **Visiteur**
  - Parcours d'une structure Composite

# Sommaire

---

- **Comportement dynamique d 'un objet**
  - Le pattern Décorateur
- **Entrées/Sorties, paquetage java.io,**
- **XML : SAX et JDOM (une introduction)**

# Principale bibliographie

---

- **GoF95**

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Design Patterns, Elements of Reusable Object-oriented software Addison Wesley 1995

+

- <http://www.eli.sdsu.edu/courses/spring98/cs635/notes/composite/composite.html>
- <http://www.patterndepot.com/put/8/JavaPatterns.htm>

+

- <http://www.javaworld.com/javaworld/jw-12-2001/jw-1214-designpatterns.html>
- [www.oreilly.com/catalog/hfdesignpat/chapter/ch03.pdf](http://www.oreilly.com/catalog/hfdesignpat/chapter/ch03.pdf)

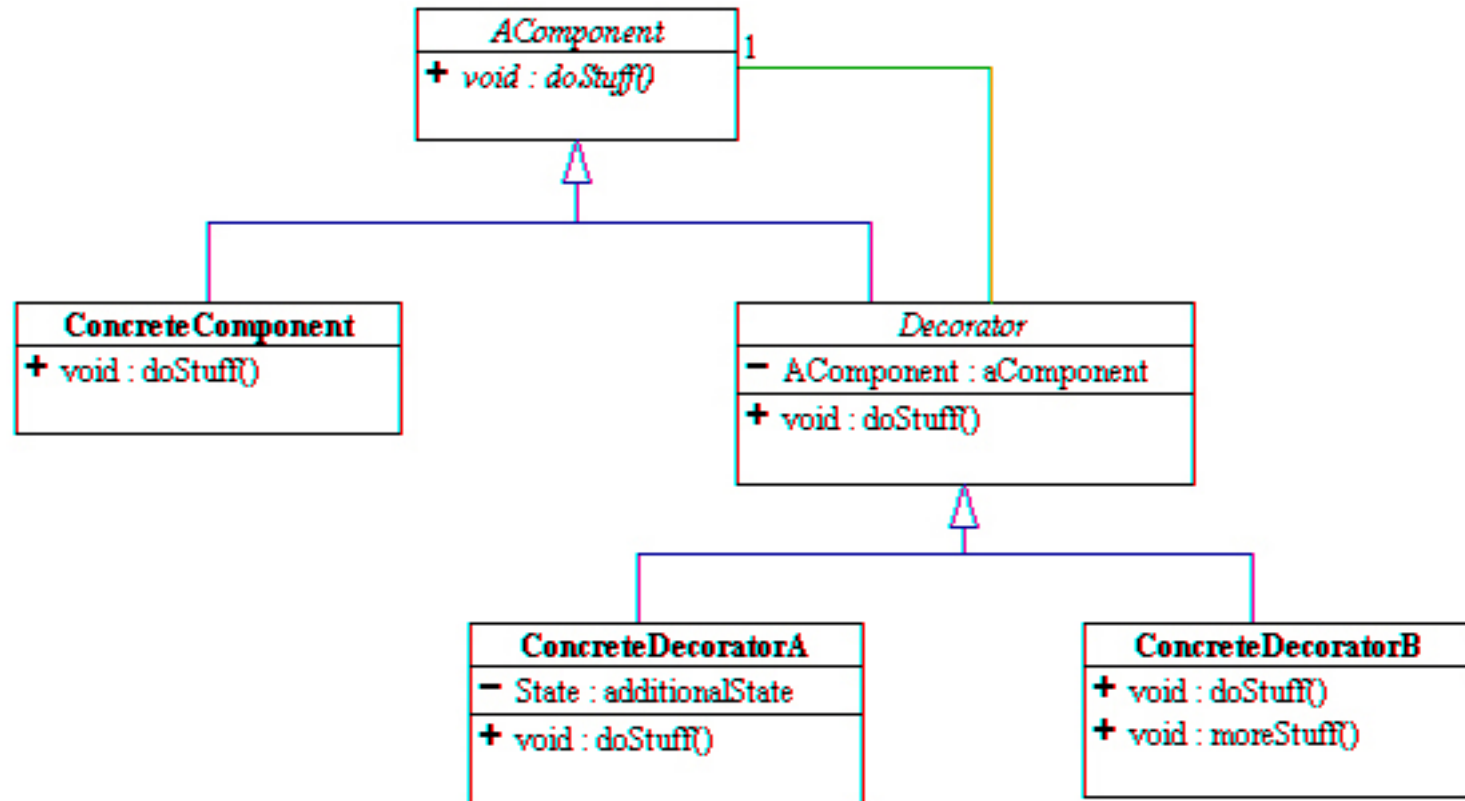
# Le Pattern Décorateur

---

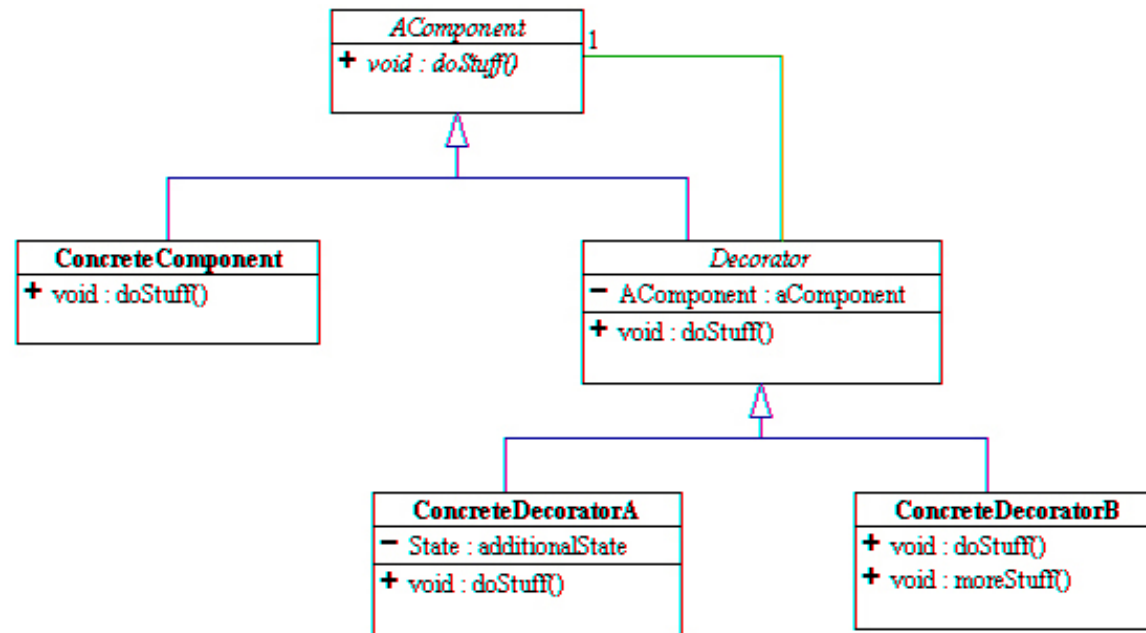
- Ajout dynamique de responsabilités
- Alternative à l'héritage
- Transparent au client
- Lire <http://oreilly.com/catalog/hfdesignpat/chapter/ch03.pdf>

# le Pattern Décorateur

- Ajout dynamique de responsabilités à un objet



# Le Pattern : mise en œuvre



- *AComponent* interface ou classe abstraite
- **ConcreteComponent** implémente\* *AComponent*
- *Decorator* implémente *AComponent* et contient une instance de *AComponent*
- Cette instance est décorée
- **ConcreteDecoratorA**, **ConcreteDecoratorB** héritent de *Decorator*
- \* implémente ou hérite de



# Quelques déclarations

---

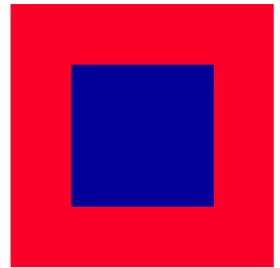
## Un composant standard

- `AComponent a = new ConcreteComponent();`
- `a.doStuff();`



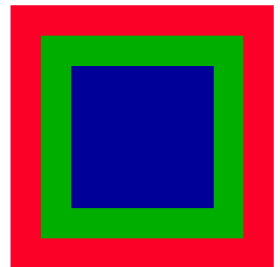
## Un composant décoré

- `AComponent a = new ConcreteDecoratorA(new ConcreteComponent());`
- `a.doStuff();`

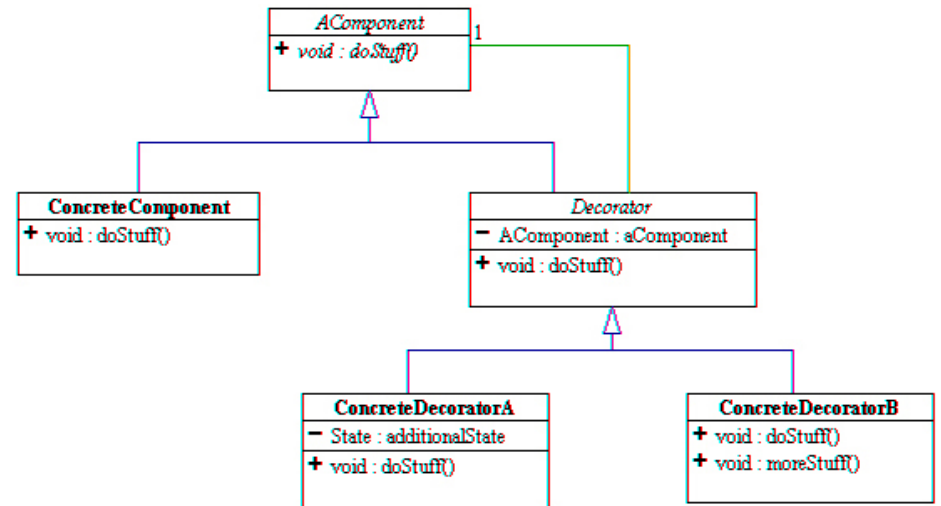


## Un composant décoré

- `AComponent a = new ConcreteDecoratorA(  
•                                   new ConcreteDecoratorB(new ConcreteComponent());`
- `a.doStuff();`



# public abstract Decorator

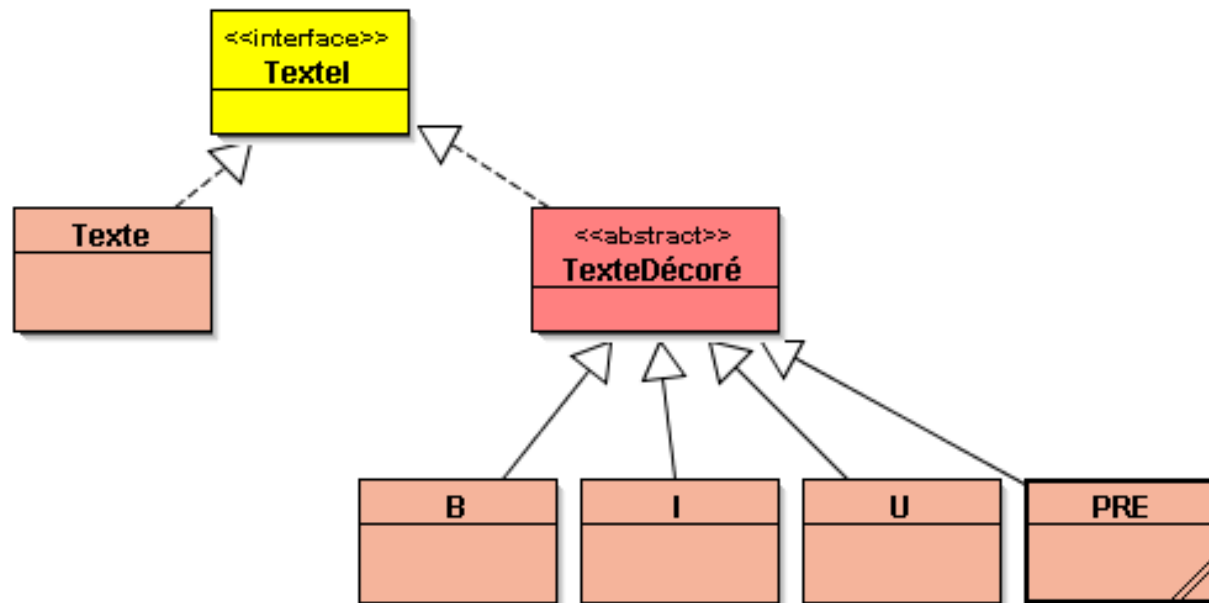


```
public abstract Decorator implements AComponent{
    private AComponent aComponent;
    public Decorator(AComponent aComponent){
        this.aComponent = aComponent;
    }

    public void doStuff(){
        aComponent.doStuff();
    }
}
```

# Un exemple de texte décoré

- Un exemple : un texte décoré par des balises HTML
  - `<b><i>exemple</i><b>`



# Le TexteI, Texte et TexteDécoré

---

```
public interface TexteI{  
    public String toHTML();  
}
```

```
public class Texte implements TexteI{  
    private String texte;  
    public Texte(String texte){this.texte = texte;}  
    public String toHTML(){return this.texte;}  
}
```

```
public abstract class TexteDécoré implements TexteI{  
    private TexteI unTexte;  
  
    public TexteDécoré(TexteI unTexte){  
        this.unTexte = unTexte;  
    }  
    public String toHTML(){  
        return unTexte.toHTML();  
    }  
}
```

# B, I, U ...

---

```
public class B extends TexteDécoré{

    public B(TexteI unTexte){
        super(unTexte);
    }

    public String toHTML(){
        return "<B>" + super.toHTML() + "</B>";
    }
}
```

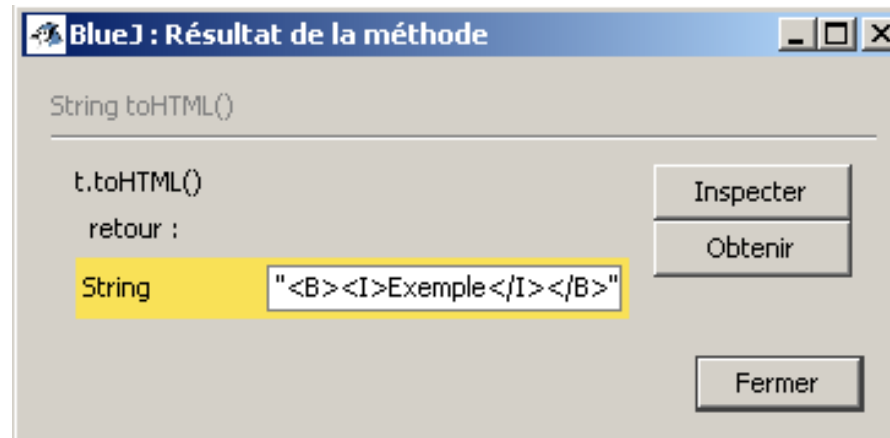
```
public class I extends TexteDécoré{

    public I(TexteI unTexte){
        super(unTexte);
    }

    public String toHTML(){
        return "<I>" + super.toHTML() + "</I>";
    }
}
```

# ***Exemple***

- `Textel t = new B( new I( new Texte("Exemple")));`
- `String s = t.toHTML();`



- **Démonstration/ Discussion**
  - Liaison dynamique ....

# <B><B> un texte </B></B> non merci

---

```
AbstractTexte texte = new B( new I( new Texte("ce texte")));  
System.out.println(texte.enHTML());
```

```
AbstractTexte textel = new B(new I(new B( new I(new Texte("ce texte")))));  
System.out.println(textel.enHTML());
```

```
AbstractTexte texte2 = new B(new B(new B( new I(new Texte("ce texte")))));  
System.out.println(texte2.enHTML());
```

```
<B><I>ce texte</I></B>  
<B><I>ce texte</I></B>  
<B><I>ce texte</I></B>
```

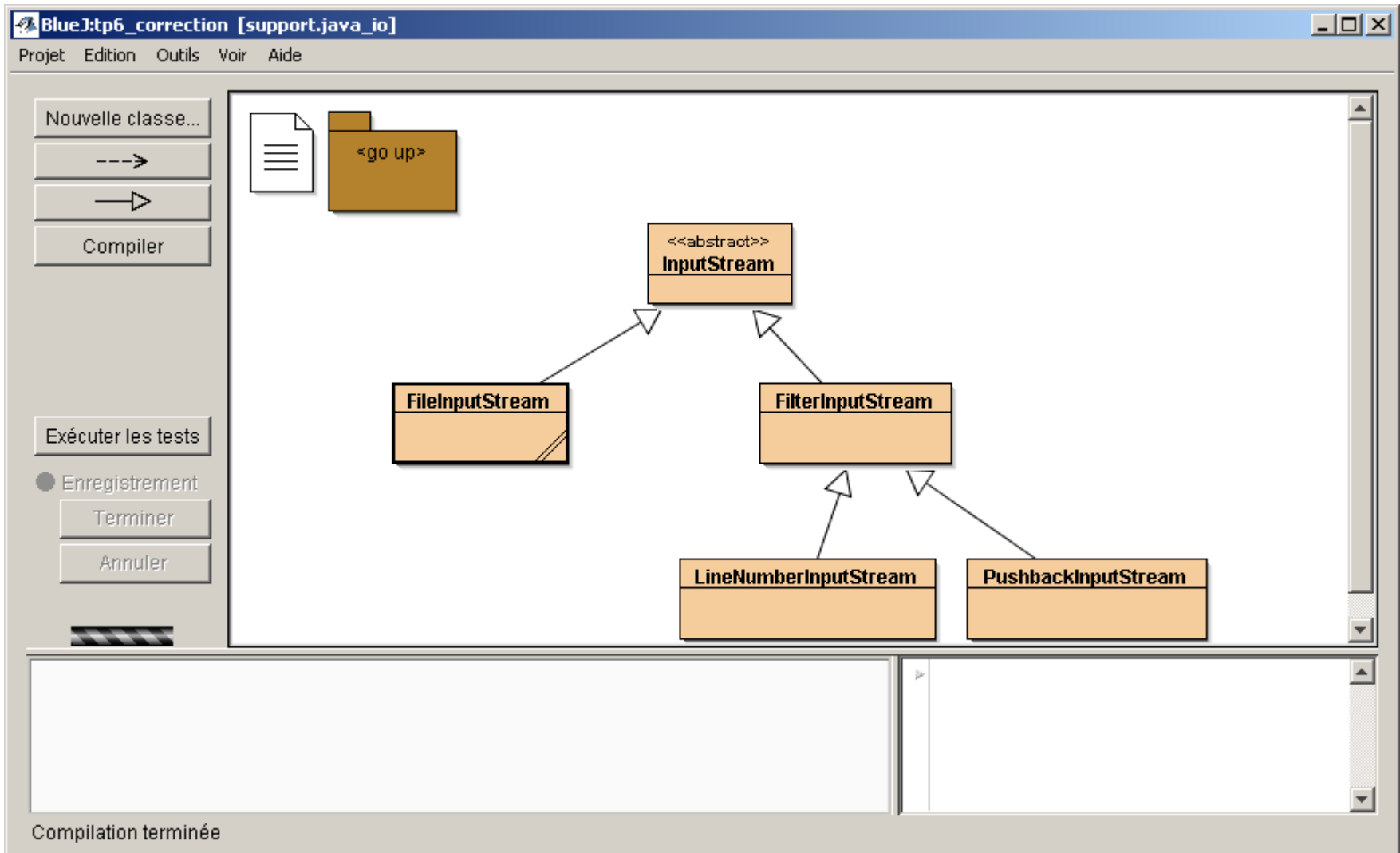
- **Comment ?**
- **En exercice ?**
  - Une solution (peu satisfaisante) est en annexe ...

# Démonstration, Discussion

---

- **Est-ce une alternative à l'héritage ?**
  - Ajout de nouvelles fonctionnalités ?
  - Comportement enrichi de méthodes héritées
- **Instance au comportement dynamique**
  - En fonction du contexte
- **Un exemple plus complet en annexe**





- Le Décorateur est bien là

# Quelques déclarations

---

## Un fichier standard

- `InputStream a = new FileInputStream("fichier.txt");`
- `a.read();`

## Un fichier décoré

- `InputStream a = new LineNumberInputStream(new FileInputStream("fichier.txt"));`
- `a.read();`

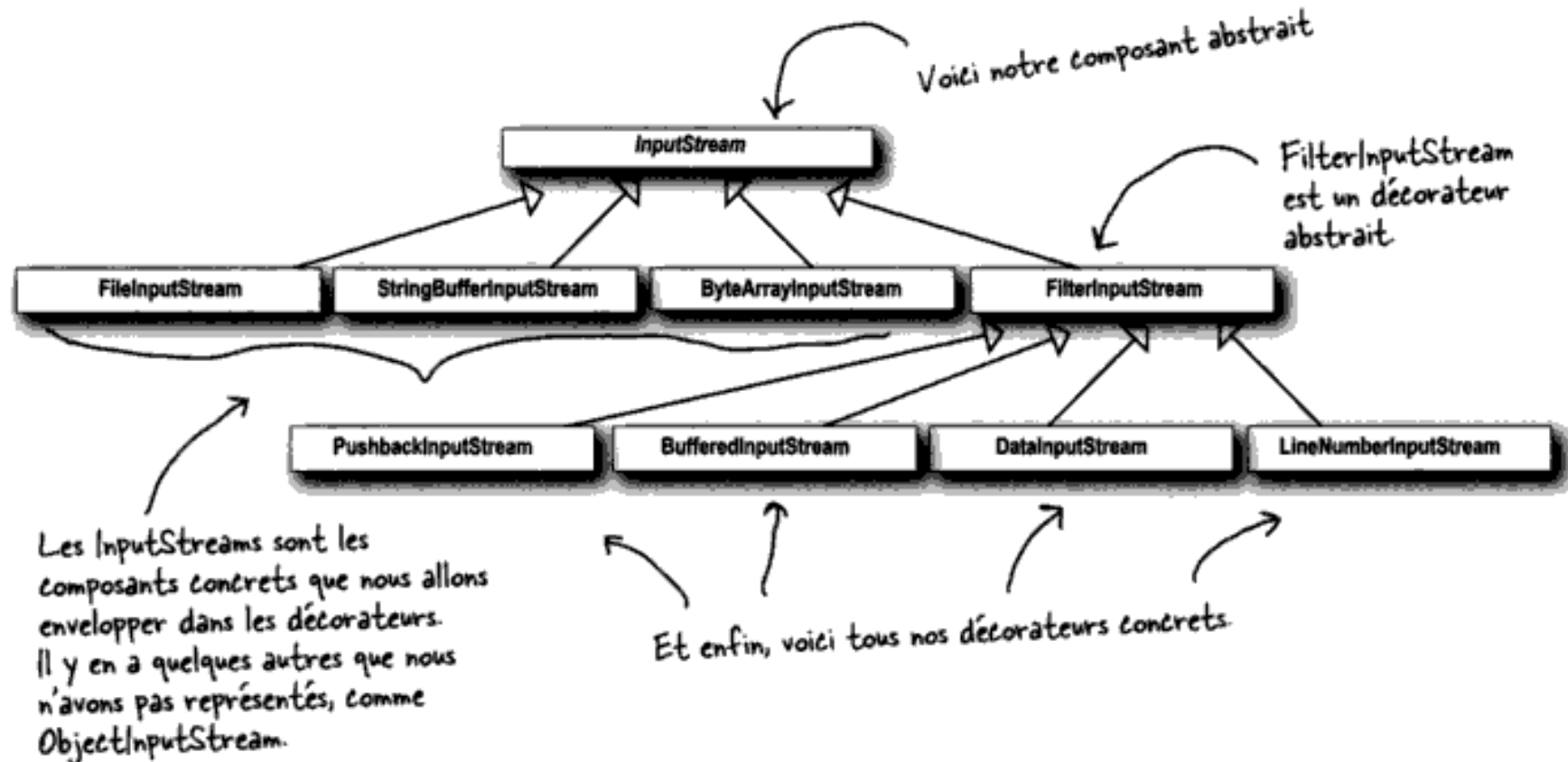
## Un autre fichier décoré

- `InputStream a = new LineNumberInputStream(  
 new PushbackInputStream(new FileInputStream("fichier.txt"))`
- `a.read();`

# Extrait de java : tête la première

le pattern Décorateur

## Décoration des classes de **java.io**



- **Sommaire**

- **Au format Java**

- **Implements java.ioSerializable**
    - **writeObject, readObject**

- **Au format XML**

- **API JDOM, SAX**

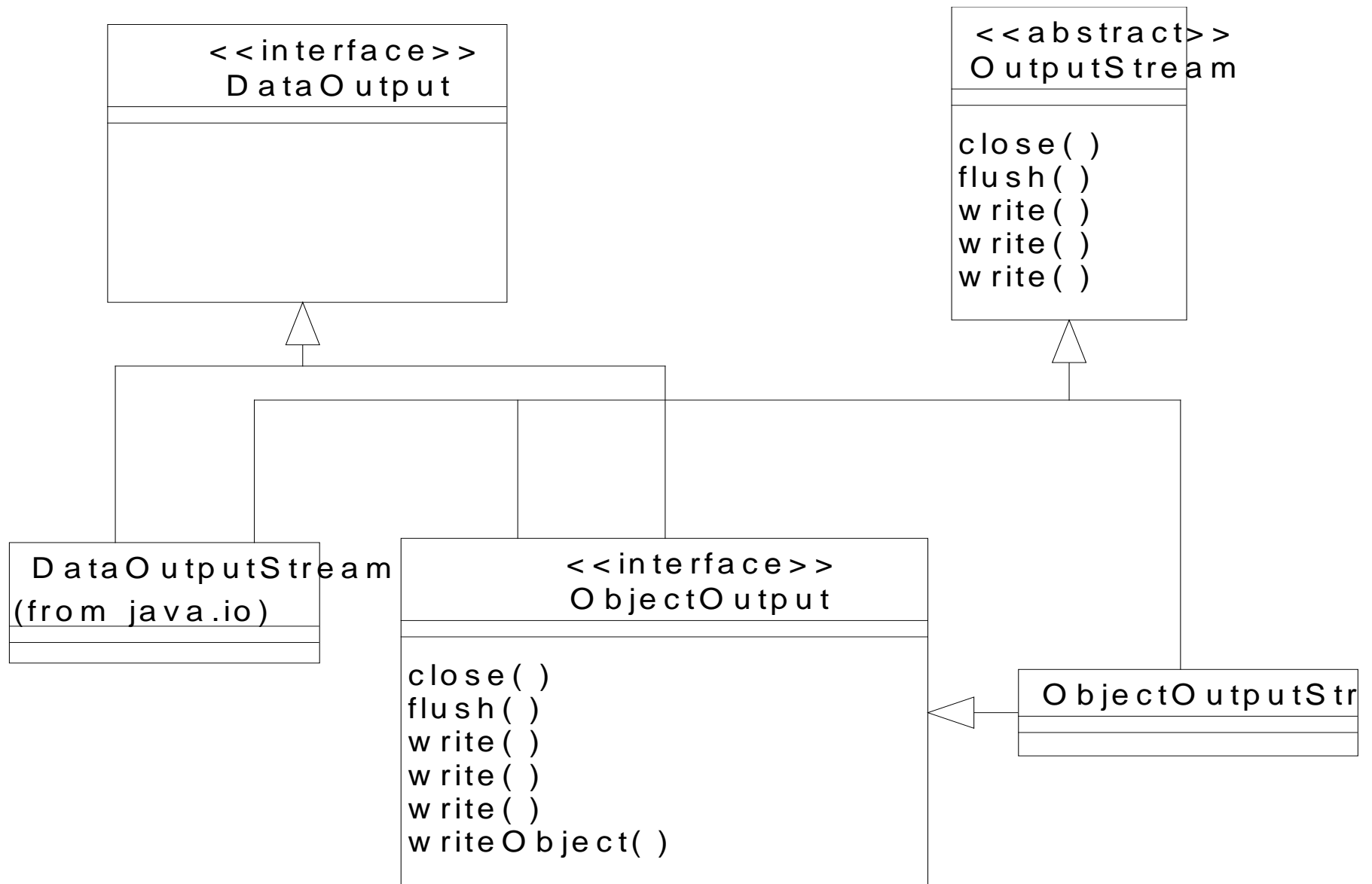
# Persistance et S rialisation

---

Entr es/Sorties d'objets quelconques.

- >la "*Serialization*" qui assure ainsi la persistance de l' tat d'un objet.
- > copie par valeur, quelque soit la complexit  de l'instance

# Serialisation



# Sauvegarde d'un objet sur fichier

---

```
FileOutputStream fout = new FileOutputStream(fileName);
```

```
ObjectOutputStream out = new ObjectOutputStream(fout);
```

```
out.writeObject(obj);
```

# Exemple

---

```
class EtudiantEsiee implements Serializable{  
    private String nom;  
    private List<Unité> liste = ...  
  
    public String toString(){  
        return "nom :"+nom+" liste: "+liste;  
    }  
}
```



# instance non sérialisable

---

Les objets dont l'état dépend de celui de l'environnement d'exécution ne sont pas sérialisables tels quels

exemples : identificateur de thread ou de process, descripteur de fichiers, socket réseau, ... .

```
private transient Thread t;
```

```
private transient Password pass;
```

# ObjectOutputStream

---

Seules des classes implémentant `Serializable` ou `Externalizable` permettent la sérialisation de leurs objets.

Le fichier contient

le nom de cette classe,

sa signature ainsi que la valeur de tous les champs non 'static' et non 'transient' ,

ceci récursivement pour tous les objets sérialisables définis dans les attributs.

# Démonstration

---

```
Unité it4101e = new Unité("IT4101E",6);  
it4101e.inscrire(new EtudiantEsiee("bbb","1234"));  
it4101e.inscrire(new EtudiantEsiee("aaa","321"));  
it4101e.inscrire(new EtudiantEsiee("ccc","456"));  
it4101e.inscrire(new EtudiantEsiee("zzz","888"));  
it4101e.inscrire(new EtudiantEsiee("yyyy","888"));
```

```
ObjectOutputStream oos=  
    new ObjectOutputStream(new FileOutputStream("it4101e.ser"));
```

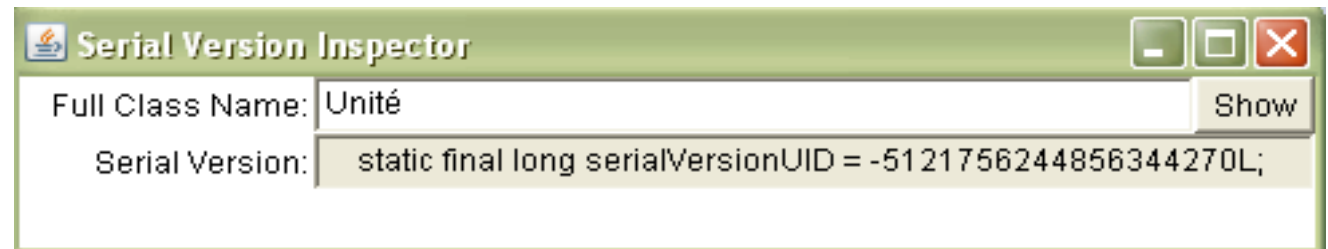
```
oos.writeObject( it4101e);
```

```
oos.close
```

# Compatibilité ascendante

- **Un scénario :**
  1. **Persistence d'une instance dans un fichier**
  2. **modification de la classe de cette instance,**
    - **du source Java (ajout d'un attribut)**
  3. **Lecture de ce fichier, création d'une instance**
    - **Compatibilité ascendante ? Valeur du nouvel attribut ?**
    - **Exception ! Mauvaise version de la classe, sauf si ...**

- **>serialver –show**



- **Une estampille de votre .class**
  - **private static final serialVersionUID = xxxx;**
  - **À installer dans le source de votre classe**

– **Démonstration**

# ObjectOutputStream

---

La sauvegarde par défaut ne convient pas

Sécurité, données cryptées, ...

Votre classe propose ces deux méthodes privées

```
private void readObject(ObjectOutputStream stream)
    throws IOException, ClassNotFoundException
```

```
private void writeObject(ObjectOutputStream stream)
    throws IOException
```

... oos.writeObject(et4101e);

déclenche la méthode privée (writeObject) de la classe Unité ( Hum, Hum...)

# interface Externalizable

---

hérite de Serializable

permet de contrôler la totalité de la sauvegarde, y compris de celle des super classes (seule l'identification de la classe de l'objet est traitée automatiquement)

```
void readExternal(ObjectInput oi);
```

```
void writeExternal(ObjectOutput oo);
```

Cette interface ne supporte pas automatiquement la gestion des versions de classes.

- **Sérialisation d'objet**
- **Avec un format lisible XML**
  - Standard, ...
- **Analyse d'un fichier XML**
  - Au fil de l'eau ... -> SAX
  - Un arbre en mémoire ... -> JDOM

# XML pourquoi faire ?

## Structuration des données

Titre

Auteur

Section

Paragraphe

Paragraphe

Paragraphe

XML: Des BD aux Services Web

Georges Gardarin

### 1. Introduction

Ces dernières années ont vu l'ouverture des systèmes d'information à l'Internet. Alors que depuis les années 1970, ces systèmes se développaient souvent par applications plus ou moins autonomes, le choc Internet ...

Ainsi, on a vu apparaître une myriade de technologies nouvelles attrayantes mais peu structurantes voir perturbantes. Certaines n'ont guère survécues. D'autres ont laissé des systèmes peu fiables et peu sécurisés. ...

L'urbanisation passe avant tout par la standardisation des échanges : il faut s'appuyer sur des standards ouverts, solides, lisibles, sécurisés, capable d'assurer l'interopérabilité avec l'Internet et les systèmes d'information. XML, "lingua franca" ...



# Vue Balisée en XML

<Livre>

<Titre> XML : Des BD aux Services Web </Titre>

<Auteur>Georges Gardarin</Auteur>

<Section titre = "Introduction">

<Paragraphe>Ces dernières années ont vu l'ouverture des systèmes d'information à l'Internet. Alors que depuis les années 1970, ces systèmes se développaient souvent par applications plus ou moins autonomes, le choc Internet ... </Paragraphe>

<Paragraphe>Ainsi, on a vu apparaître une myriade de technologies nouvelles attrayantes mais peu structurantes voir perturbantes. Certaines n'ont guère survécues. D'autres ont laissé des systèmes peu fiables et peu sécurisés. ...</Paragraphe>

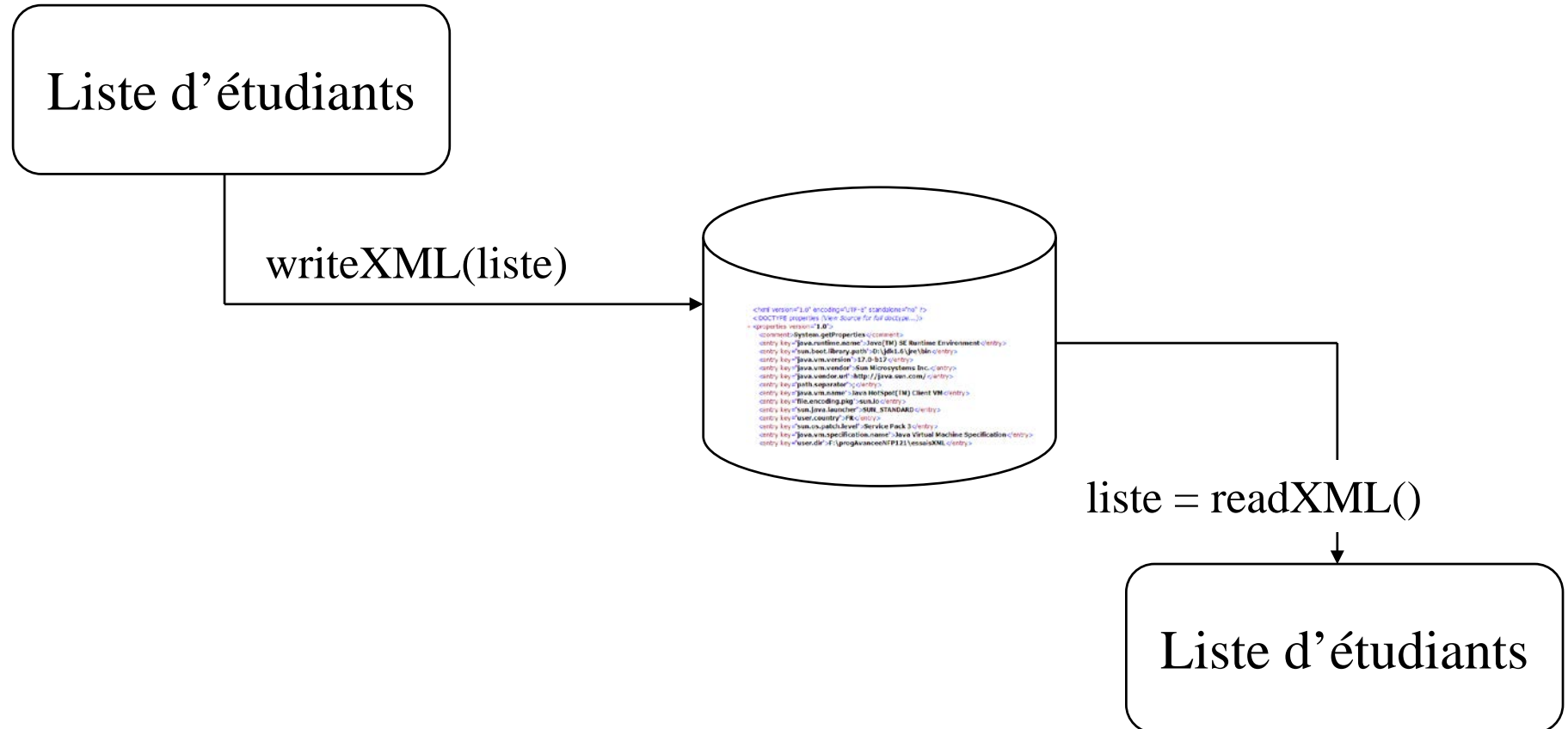
<Paragraphe>L'urbanisation passe avant tout par la standardisation des échanges : il faut s'appuyer sur des standards ouverts, solides, lisibles, sécurisés, capable d'assurer l'interopérabilité avec l'Internet et les systèmes d'information. XML, "langua franca" ... </Paragraphe>

</Section>

</Livre>

# XML pourquoi faire ?

## Persistance d'un objet et "lisible"



# Exemple : Properties de java.util

## Écriture

```
Properties props = System.getProperties();  
props.storeToXML(new FileOutputStream(new File("props.xml")), "System.getProperties");
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>  
<!DOCTYPE properties (View Source for full doctype...)>  
- <properties version="1.0">  
  <comment>System.getProperties</comment>  
  <entry key="java.runtime.name">Java(TM) SE Runtime Environment</entry>  
  <entry key="sun.boot.library.path">D:\jdk1.6\jre\bin</entry>  
  <entry key="java.vm.version">17.0-b17</entry>  
  <entry key="java.vm.vendor">Sun Microsystems Inc.</entry>  
  <entry key="java.vendor.url">http://java.sun.com/</entry>  
  <entry key="path.separator">;</entry>  
  <entry key="java.vm.name">Java HotSpot(TM) Client VM</entry>  
  <entry key="file.encoding.pkg">sun.io</entry>  
  <entry key="sun.java.launcher">SUN_STANDARD</entry>  
  <entry key="user.country">FR</entry>  
  <entry key="sun.os.patch.level">Service Pack 3</entry>  
  <entry key="java.vm.specification.name">Java Virtual Machine Specification</entry>  
  <entry key="user.dir">F:\progAvanceeNFP121\essaisXML</entry>
```

Le fichier  
**props.xml**  
ici lu par  
un  
Navigateur

## Lecture

```
Properties props = new Properties();  
props.loadFromXML(new FileInputStream(new File("props.xml")));  
assert True(System.getProperties().equals(props));
```

# XML : la famille

---

- **Né : fin 96**
- **Père : W3C**
- **Petit-fils de SGML (ISO-1986)**
- **Cousin d'HTML**
- **Reconnu le : 10/02/98 – version 1.0**
- **Descendance – XHMTL, MathML, ANT...**

# X comme eXtensible

---

- **HTML : nombre fini de balises**
- **XML : possibilité de définir les balises**
- **HTLM : balises pour formater**
- **XML : balises pour structurer**
- **DTD ou Schéma pour définir les balises**

# Règles syntaxiques

---

- Commencer par une déclaration XML
- Balisage sensible à la casse
- La valeur des attributs doit être quotée
- Balises non vides appariées `<br></br>`
- Balises vides fermées `<br/>`
- Les éléments ne doivent pas se chevaucher
  - `<jour> <mois> </jour> </mois>` **interdit**
- Un élément doit encapsuler tous les autres
- Ne pas utiliser les caractères `<` et `&` seuls

# Le Prologue

---

- **Une déclaration XML**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- **Instructions de traitement (PI Processing Instruction)**

- Une indication de traitement est destinée aux applications qui manipulent les documents XML

- **Une déclaration de type de document**

- indique le type de document auquel se conforme le document en question (ex. DTD)

```
<!DOCTYPE rapport SYSTEM "rapport.dtd">
```

# Élément (*Element*)

---

- **Composant de base**
- **Identifié par un nom**
- **Délimité par une balise ouvrante et une balise fermante à ce nom**  
`<AUTEUR> Victor Hugo </AUTEUR>`
- **Ou élément vide**  
`<PHOTO Source= "victor.gif" />`
- **Contenu textuel, éléments ou mixte**



# Les attributs (*Attribut*)

---

- Inclus dans la balise ouvrante d'un élément
- Composé d'un nom et d'une valeur

```
<AUTEUR NE="1802" MORT="1885" >
```

Victor Hugo

```
</AUTEUR>
```

# Données

---

- **Constituées par un flot de caractères**
  - tous les caractères sont acceptés sauf le caractère « & » et le caractère « < »
  - **Exemple :** `<auteurs>Victor Hugo<auteurs>`
- **Si l'on souhaite insérer des caractères « spéciaux », il est préférable d'utiliser une section littérale ou CDATA**

```
<![CDATA[<auteurs>S. Fleury & al.</auteurs>]]>
```

**qui se traduit en :**

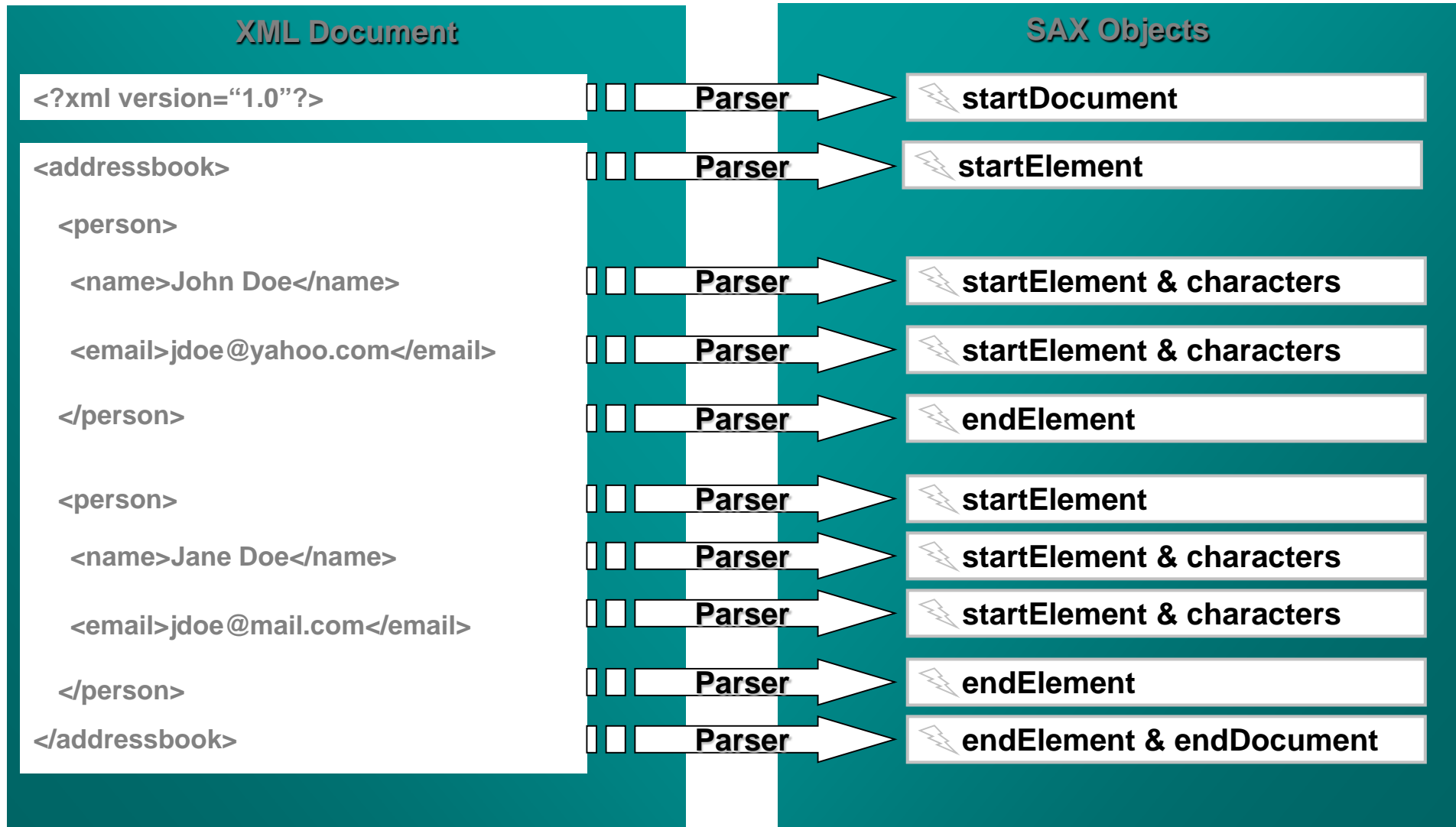
```
<auteurs>S. Fleury & al.</auteurs>
```

# SAX Simple Api for Xml

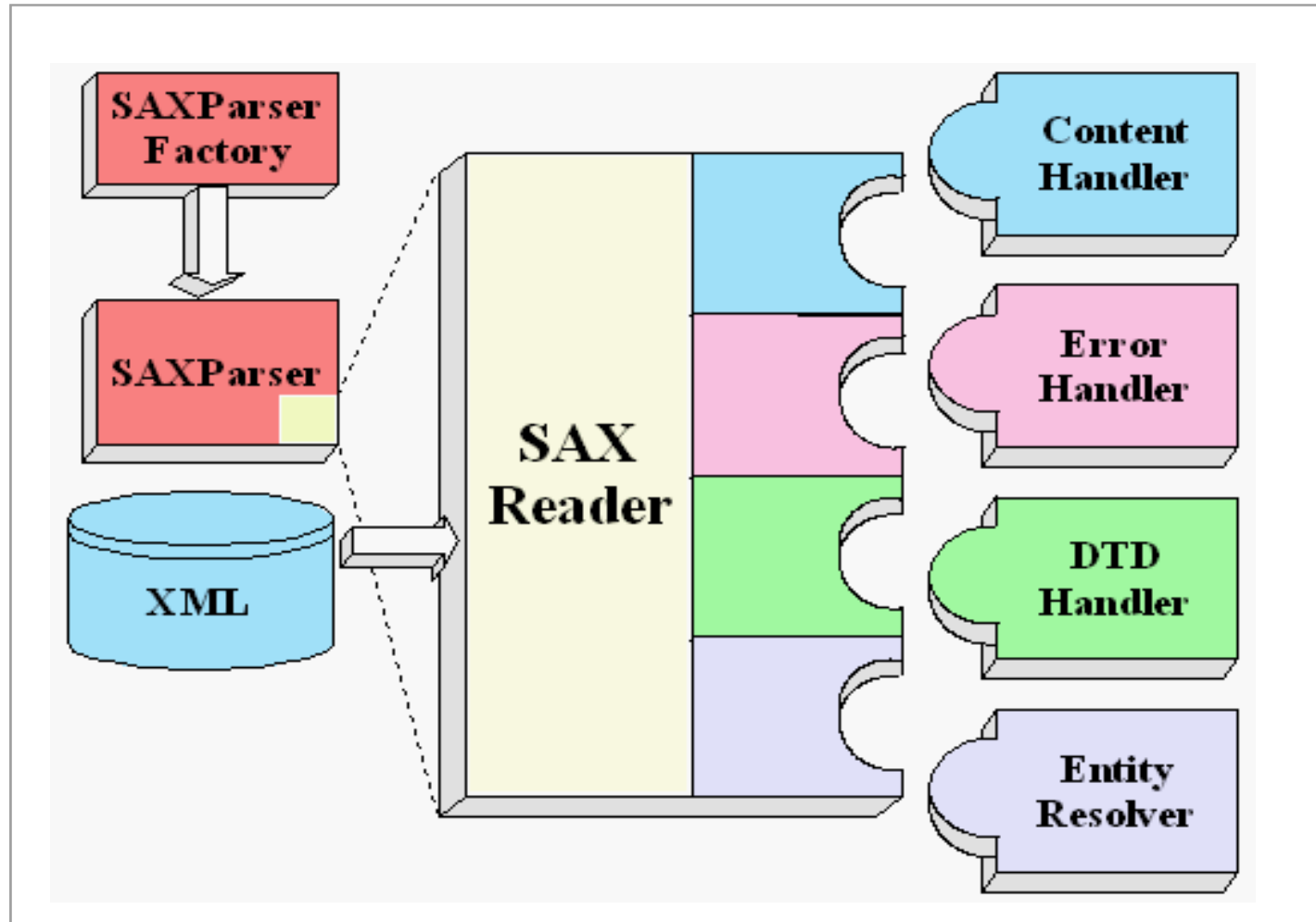
---

- Début des travaux Dec, 1997
- SAX 1.0 Mai, 1998
  - Tim Bray
  - David Megginson
  - ...
- SAX 2.0 Mai, 2000
- SAX 2.0.2 27-April 2004:
- .....

# SAX Comment ?



# Implémenter les Handlers d'évènements du parseur



## DefaultHandler

Il implémente ces différents Handler avec des méthodes vides, de sorte que l'on peut surcharger seulement celles qui nous intéressent.

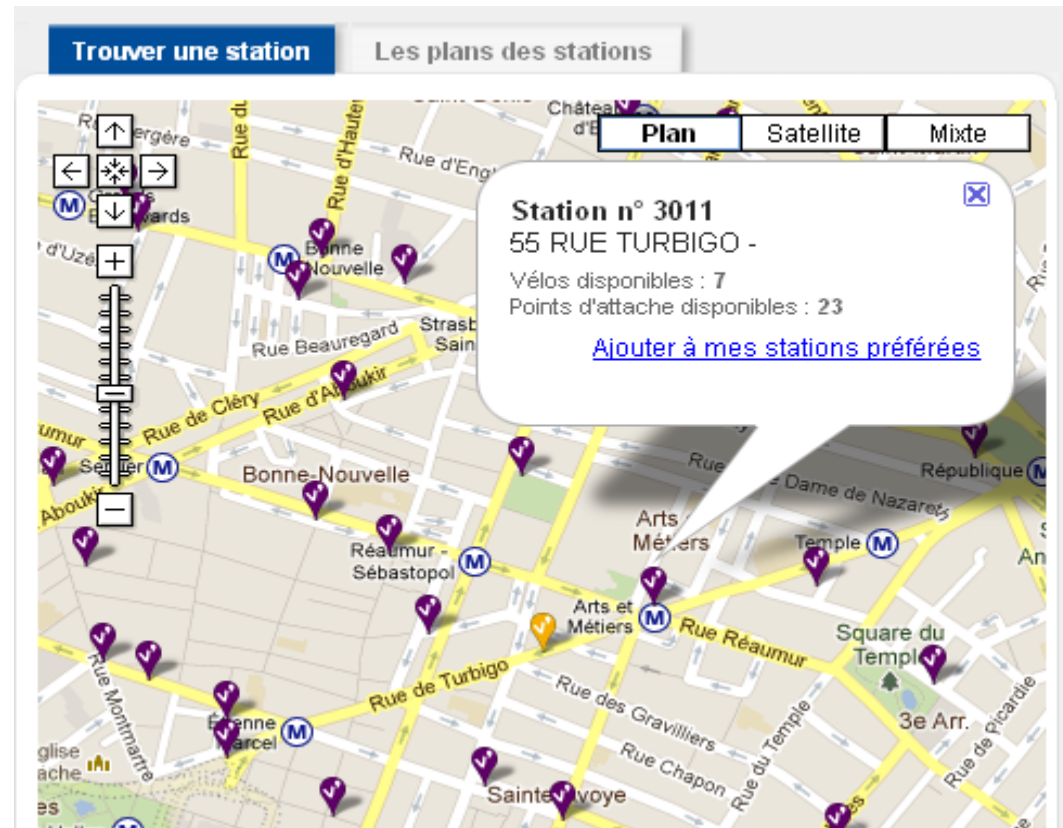
# Structure du main

```
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.XMLReader;

public class SurveyReader extends DefaultHandler{
    public static void main (String args[]) {
        XMLReader xmlReader = null;
        try { SAXParserFactory spfactory = SAXParserFactory.newInstance();
            SAXParser saxParser = spfactory.newSAXParser();
            xmlReader = saxParser.getXMLReader();
            xmlReader.setContentHandler(new SurveyReader());
            InputSource source = new InputSource("surveys.xml");
            xmlReader.parse(source);
        } catch (Exception e) { System.err.println(e); System.exit(1);
        }
    }
}
```

- Toutes les applications SAX doivent implanter un ContentHandler
- Méthodes :
  - public void **startDocument()** throws SAXException
  - public void **endDocument()** throws SAXException
  - public void **startElement**(String nspURI, String localName, String qName, Attributes atts) throws SAXException
  - public void **characters**(char[] ch, int start, int length) throws SAXException
  - ...

# Un exemple les stations Vélib



- <http://www.velib.paris.fr>
- <http://www.velib.paris.fr/service/carto>
- <http://www.velib.paris.fr/service/stationdetails/{number}>



<carto>

<markers>

```
<marker name="00901 - STATION MOBILE 1" number="901"
  address="ALLEE DU BELVEDERE PARIS 19 - 0 75000 Paris -"
  fullAddress="ALLEE DU BELVEDERE PARIS 19 - 0 75000 Paris -
75000 PARIS" lat="48.892745582406675"
lng="2.391255159886939" open="1" bonus="0" />
```

```
<marker name="03011 - TURBIGO" number="3011" address="55 RUE
TURBIGO -" fullAddress="55 RUE TURBIGO - 75003
PARIS" lat="48.86558781525867" lng="2.356094545731025" open
="1" bonus="0" />
```

Analyse des attributs

de la balise **marker**

en SAX -> méthode **startElement**

**<station>**

<available>21</available>

<free>10</free>

<total>31</total>

<ticket>1</ticket>

**</station>**

**Analyse du contenu**

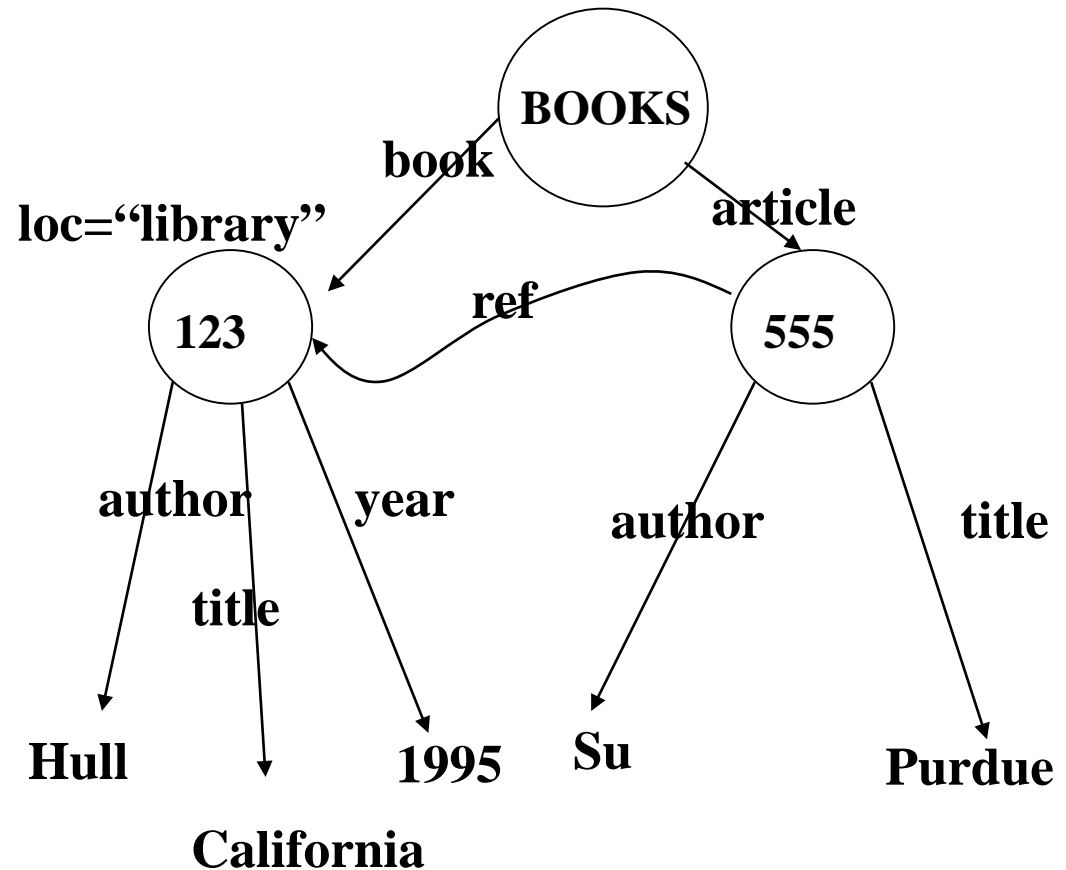
de la balise **station**

en SAX -> des méthodes **startElement, endElement, characters**

# JDOM

## Représentation Arborescente des documents XML

```
<BOOKS>  
<book id="123" loc="library">  
  <author>Hull</author>  
  <title>California</title>  
  <year> 1995 </year>  
</book>  
<article id="555" ref="123">  
  <author>Su</author>  
  <title> Purdue</title>  
</article>  
</BOOKS>
```



# De DOM à JDOM

---

DOM est l'acronyme de ***Document Object Model***.

API pour modéliser, de parcourir et de manipuler un document XML  
DOM fournit une représentation mémoire d'un document XML  
sous la forme d'un arbre d'objets et permet la manipulation  
(parcours, recherche et mise à jour).

DOM est défini pour être indépendant du langage dans lequel il sera implémenté. DOM n'est donc pas spécifique à Java.

**JDOM manipule les éléments d'un Document Object Model spécifique (créé grâce à un constructeur basé sur SAX).**

**JDOM permet donc de construire des documents, de naviguer dans leur structure, d'ajouter, de modifier, ou de supprimer leur contenu.**

## Créer un fichier XML avec JDOM

---

```
<personnes>  
  <etudiant classe="P2">  
    <nom>CynO</nom>  
  </etudiant>  
</personnes>
```

# JDOM1.java

---

```
import java.io.*;
import org.jdom.*;
import org.jdom.output.*;
public class JDOM1{
    static Element racine = new Element("personnes");
    static org.jdom.Document document = new Document(racine);

    public static void main(String[] args){
        Element etudiant = new Element("etudiant");
        racine.addContent(etudiant);
        Attribute classe = new Attribute("classe","P2");
        etudiant.setAttribute(classe);
        Element nom = new Element("nom");
        nom.setText("CynO");
        etudiant.addContent(nom);
        affiche();
        enregistre("Exercice1.xml");
    }
}
```

# Element

```
Element
<<create>> +Element(name: String)
+getName(): String
+setName(name: String): Element
+getValue(): String
+isRootElement(): boolean
+getContentSize(): int
+indexOf(child: Content): int
+getText(): String
+getChildText(name: String): String
+setText(text: String): Element
+getContent(): List
+getContent(filter: Filter): List
+removeContent(): List
+removeContent(filter: Filter): List
+setContent(newContent: Collection): Element
+setContent(index: int, child: Content): Element
+setContent(index: int, collection: Collection): Parent
+addContent(str: String): Element
+addContent(child: Content): Element
+addContent(collection: Collection): Element
+addContent(index: int, child: Content): Element
+addContent(index: int, c: Collection): Element
+cloneContent(): List
+getContent(index: int): Content
+removeContent(child: Content): boolean
+removeContent(index: int): Content
+setContent(child: Content): Element
+isAncestor(element: Element): boolean
+getAttributes(): List
+getAttribute(name: String): Attribute
+getAttributeValue(name: String): String
+getAttributeValue(name: String, def: String): String
+setAttributes(newAttributes: List): Element
+setAttribute(name: String, value: String): Element
+setAttribute(attribute: Attribute): Element
+removeAttribute(name: String): boolean
+removeAttribute(attribute: Attribute): boolean
+getDescendants(): Iterator
+getDescendants(filter: Filter): Iterator
+getChildren(): List
+getChildren(name: String): List
+getChild(name: String): Element
+removeChild(name: String): boolean
+removeChildren(name: String): boolean
```

Créer un élément

Lui affecter un texte

Ajouter un fils

Ajouter un attribut

**<personnes>**

**<etudiant classe="P2">**

**<nom>CynO</nom>**

**<etudiant>**

**<personnes>**

# Document

## Document

```
<<create>>+Document()
<<create>>+Document(rootElement: Element, docType: DocType, baseURI: String)
<<create>>+Document(rootElement: Element, docType: DocType)
<<create>>+Document(rootElement: Element)
<<create>>+Document(content: List)
+getContentSize(): int
+indexOf(child: Content): int
+hasRootElement(): boolean
+getRootElement(): Element
+setRootElement(rootElement: Element): Document
+detachRootElement(): Element
+getDocType(): DocType
+setDocType(docType: DocType): Document
+addContent(child: Content): Document
+addContent(c: Collection): Document
+addContent(index: int, child: Content): Document
+addContent(index: int, c: Collection): Document
+cloneContent(): List
+getContent(index: int): Content
+getContent(): List
+getContent(filter: Filter): List
+removeContent(): List
+removeContent(filter: Filter): List
+setContent(newContent: Collection): Document
+setBaseURI(uri: String)
+getBaseURI(): String
+setContent(index: int, child: Content): Document
+setContent(index: int, collection: Collection): Document
+removeContent(child: Content): boolean
+removeContent(index: int): Content
+setContent(child: Content): Document
+toString(): String
+equals(ob: Object): boolean
+hashCode(): int
+clone(): Object
+getDescendants(): Iterator
+getDescendants(filter: Filter): Iterator
+getParent(): Parent
+getDocument(): Document
+setProperty(id: String, value: Object)
+getProperty(id: String): Object
```



# Attribute

## Attribute

```
<<create>>+Attribute(name: String, value: String, namespace: Namespace)
<<create>>+Attribute(name: String, value: String, type: int, namespace: Namespace)
<<create>>+Attribute(name: String, value: String)
<<create>>+Attribute(name: String, value: String, type: int)
+getParent(): Element
+getDocument(): Document
+detach(): Attribute
+getName(): String
+setName(name: String): Attribute
+getQualifiedName(): String
+getNamespacePrefix(): String
+getNamespaceURI(): String
+getNamespace(): Namespace
+setNamespace(namespace: Namespace): Attribute
+getValue(): String
+setValue(value: String): Attribute
+getAttributeType(): int
+setAttributeType(type: int): Attribute
+toString(): String
+equals(ob: Object): boolean
+hashCode(): int
+clone(): Object
+getIntValue(): int
+getLongValue(): long
+getFloatValue(): float
+getDoubleValue(): double
+getBooleanValue(): boolean
```

## AttributeList

```
<<create>> ~AttributeList(parent: Element)
~uncheckedAddAttribute(a: Attribute)
+add(obj: Object): boolean
+add(index: int, obj: Object)
~add(index: int, attribute: Attribute)
+addAll(collection: Collection): boolean
+addAll(index: int, collection: Collection): boolean
+clear()
~clearAndSet(collection: Collection)
+get(index: int): Object
~get(name: String, namespace: Namespace): Object
~indexOf(name: String, namespace: Namespace): int
+remove(index: int): Object
~remove(name: String, namespace: Namespace): boolean
+set(index: int, obj: Object): Object
~set(index: int, attribute: Attribute): Object
+size(): int
+toString(): String
```

# Éditer le document

---

```
static void affiche(){  
    try{  
        XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());  
        sortie.output(document, System.out);  
    }catch (java.io.IOException e){}  
}
```

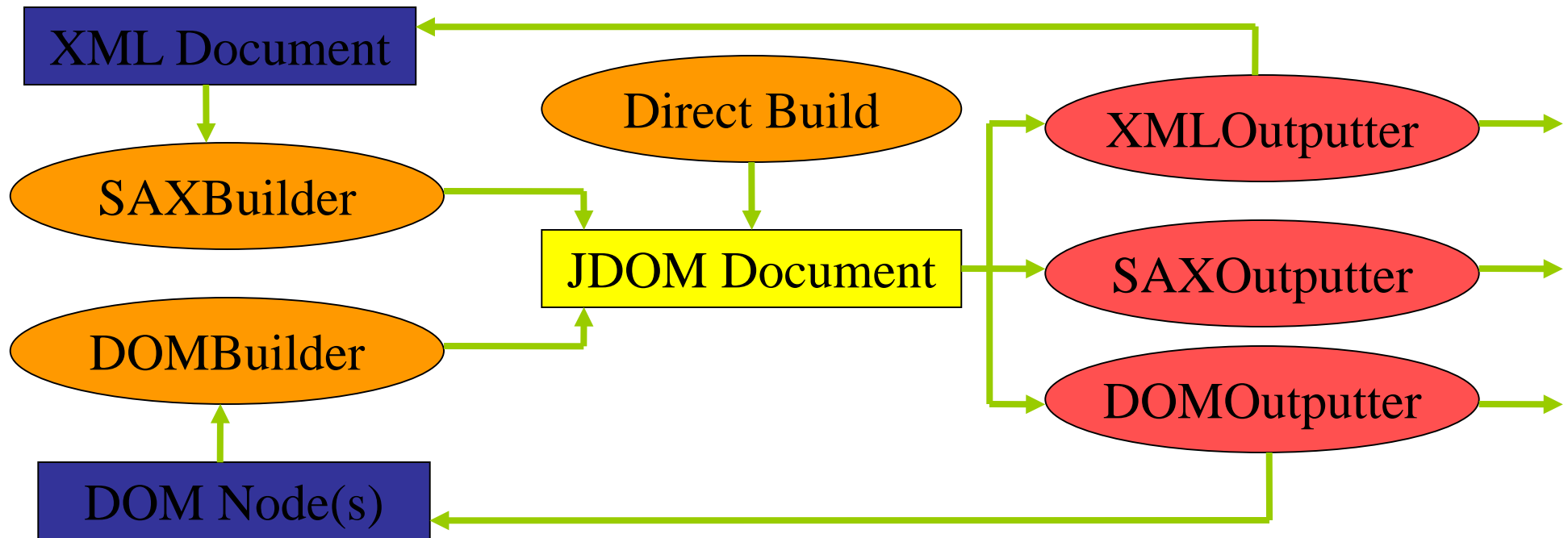
```
static void enregistre(String fichier){  
    try{  
        XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());  
        //Remarquez qu'il suffit simplement de créer une instance de  
        // FileOutputStream avec en argument le nom du fichier  
        // pour effectuer la sérialisation.  
        sortie.output(document, new FileOutputStream(fichier));  
    }catch (java.io.IOException e){}  
}
```

## XMLOutputter

```
<<create>>+XMLOutputter()
<<create>>+XMLOutputter(format: Format)
<<create>>+XMLOutputter(that: XMLOutputter)
+setFormat(newFormat: Format)
+getFormat(): Format
+output(doc: Document, out: OutputStream)
+output(doctype: DocType, out: OutputStream)
+output(element: Element, out: OutputStream)
+outputElementContent(element: Element, out: OutputStream)
+output(list: List, out: OutputStream)
+output(cdata: CDATA, out: OutputStream)
+output(text: Text, out: OutputStream)
+output(comment: Comment, out: OutputStream)
+output(pi: ProcessingInstruction, out: OutputStream)
+output(entity: EntityRef, out: OutputStream)
+output(doc: Document, out: Writer)
+output(doctype: DocType, out: Writer)
+output(element: Element, out: Writer)
+outputElementContent(element: Element, out: Writer)
+output(list: List, out: Writer)
+output(cdata: CDATA, out: Writer)
+output(text: Text, out: Writer)
+output(comment: Comment, out: Writer)
+output(pi: ProcessingInstruction, out: Writer)
+output(entity: EntityRef, out: Writer)
+outputString(doc: Document): String
+outputString(doctype: DocType): String
+outputString(element: Element): String
+outputString(list: List): String
+outputString(cdata: CDATA): String
+outputString(text: Text): String
+outputString(comment: Comment): String
+outputString(pi: ProcessingInstruction): String
+outputString(entity: EntityRef): String
+escapeAttributeEntities(str: String): String
+escapeElementEntities(str: String): String
+clone(): Object
+toString(): String
```

# Structure générale

- XML Document -> SAXBuilder -> XMLOutputter



# JDOM : Un Document XML

---

- **Création**

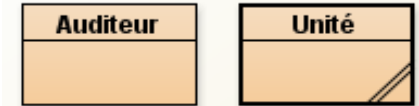
- **Element e** = new Element("addressbook");
- e.setAttribute("lang", "eng");
- **Document** doc = new **Document**(e);

- **Lecture**

- Builder builder = new SAXBuilder();
- Document doc = builder.build(url);

# Un exemple ... une unité d'enseignement, 3 auditeurs

```
public class Unité{
```



```
    private String intitulé;           // Attribut JDOM
    private int nombreDeCrédits;       // Attribut JDOM
    private SortedSet<Auditeur> inscrits; // Element(s) JDOM
```

... •

```
public Element toXML(){
    Element unité = new Element("unite");
    unité.setAttribute("intitule",this.intitulé);
    unité.setAttribute("nombreDeCredits",""+nombreDeCrédits);
    Element liste = new Element("inscrits");
    for(Auditeur a : inscrits){
        liste.addContent(a.toXML());
    }
    unité.addContent(liste);
    return unité;
}
```

## Un exemple ... une unité, 3 auditeurs

---

```
public class Auditeur implements Comparable<Auditeur>{  
    private String nom;  
    private String matricule;
```

...

```
public Element toXML(){  
    Element nom = new Element("nom");  
    nom.setText(this.nom);  
    Element matricule = new Element("matricule");  
    matricule.setText(this.matricule);  
  
    Element elt = new Element("auditeur");  
    elt.addContent(nom);  
    elt.addContent(matricule);  
    return elt;  
}
```



# Le fichier créé

```
Unité nfp121 = new Unité("NFP121",6);  
nfp121.inscrire(new Auditeur("bbb","1234"));  
nfp121.inscrire(new Auditeur("aaa","321"));  
nfp121.inscrire(new Auditeur("ccc","456"));
```

```
Document document = new Document(nfp121.toXML());  
XMLOutputter out = new XMLOutputter(Format.getPrettyFormat());  
out.output(document, System.out);  
out.output(document, new FileOutputStream(new File("nfp121.xml")));
```

```
<?xml version="1.0" encoding="UTF-8" ?>  
- <unite intitule="NFP121" nombreDeCredits="6">  
- <inscrits>  
- <auditeur>  
  <nom>aaa</nom>  
  <matricule>321</matricule>  
</auditeur>  
- <auditeur>  
  <nom>bbb</nom>  
  <matricule>1234</matricule>  
</auditeur>  
- <auditeur>  
  <nom>ccc</nom>  
  <matricule>456</matricule>  
</auditeur>  
</inscrits>  
</unite>
```

# Lecture du fichier, reconstruction de l'arbre

---

```
SAXBuilder sxb = new SAXBuilder();  
Document documentLu = sxb.build(new File("nfp121.xml"));
```

Principe :

La racine est un Element

```
Element racine = documentLu.getRootElement();
```

et possède des « enfants »

```
Element inscrits = racine.getChild("inscrits");  
for(Object o : inscrits.getChildren("auditeur")){  
    ...  
}
```

# Au Complet

---

```
SAXBuilder sxb = new SAXBuilder();
Document documentLu = sxb.build(new File("nfp121.xml"));

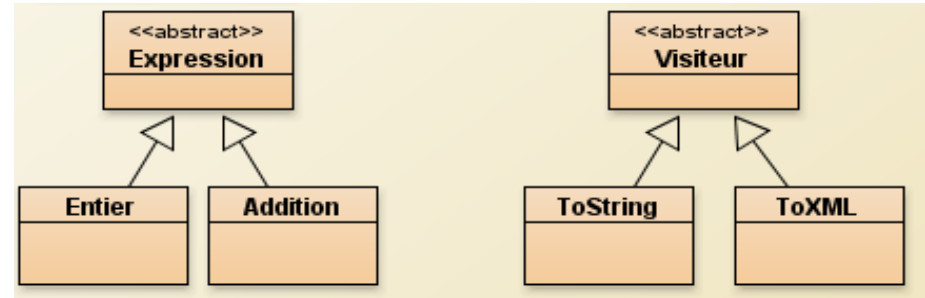
out = new XMLOutputter(Format.getPrettyFormat());
System.out.print("document lu : " );out.output(documentLu, System.out);

Element racine = documentLu.getRootElement();
Unité unité = new Unité(racine.getAttribute("intitule").getValue(),
Integer.parseInt(racine.getAttribute("nombreDeCredits").getValue()));

Element inscrits = racine.getChild("inscrits");
for(Object o : inscrits.getChildren("auditeur")){
    Element elt = (Element)o;
    unité.inscrire(new Auditeur(elt.getChild("nom").getValue(), elt.getChild("matricule").getValue()));
}
```

## Démonstration

# Composite + Visiteur



```
public class ToXML extends Visiteur<Element>{
    public Element visiter(Entier e){
        Element elt = new Element("Entier");
        elt.setAttribute(new Attribute("class",e.getClass().getName()));
        elt.addContent(Integer.toString(e.getValeur()));
        return elt;
    }

    public Element visiter(Addition a){
        Element elt = new Element("Addition");
        elt.setAttribute(new Attribute("class",a.getClass().getName()));
        elt.addContent(new Comment("addition et visiteur XML, NFP121"));
        elt.addContent(a.getOp1().accepter(this));
        elt.addContent(a.getOp2().accepter(this));
        return elt;
    }
}
```

ici le nom de la classe  
est un attribut

# Composite + Visiteur

- Génération du document

```
Document document = new Document(add.accepter(new toXML()));  
XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());  
sortie.output(document, System.out);
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<Addition class="expression.Addition">  
  <Addition class="expression.Addition">  
    <Entier class="expression.Entier">3</Entier>  
    <Entier class="expression.Entier">2</Entier>  
  </Addition>  
  <Addition class="expression.Addition">  
    <Addition class="expression.Addition">  
      <Entier class="expression.Entier">3</Entier>  
      <Entier class="expression.Entier">2</Entier>  
    </Addition>  
    <Entier class="expression.Entier">5</Entier>  
  </Addition>  
</Addition>
```

# Conclusion

---

- **XML incontournable**
- **XML comme langage**
- **XML comme configuration**
- **XML uniforme, description arborescente**
- **SAX, JDOM Outils de bas niveau**
- **JAXB adéquation Java/XML**

- **Décorateur et les pizzas**

# Un autre exemple extrait de Head first

---

- inspiré de [http://jfod.cnam.fr/NFP121/Chapter03\\_Head\\_First.pdf](http://jfod.cnam.fr/NFP121/Chapter03_Head_First.pdf)
- **Confection d'une Pizza à la carte**
  - 3 types de pâte
  - 12 ingrédients différents, (dont on peut doubler ou plus la quantité)
  - si en moyenne 5 ingrédients, soit 792\* combinaisons !
  - ? Confection comme décoration ?
  - Une description de la pizza commandée et son prix

\*  $n$  parmi  $k$ ,  $n! / k!(n-k)!$



# 3 types de pâte

---

- **Pâte solo, (très fine...)**
- **Pâte Classic**
- **Pâte GenerousCrust©**



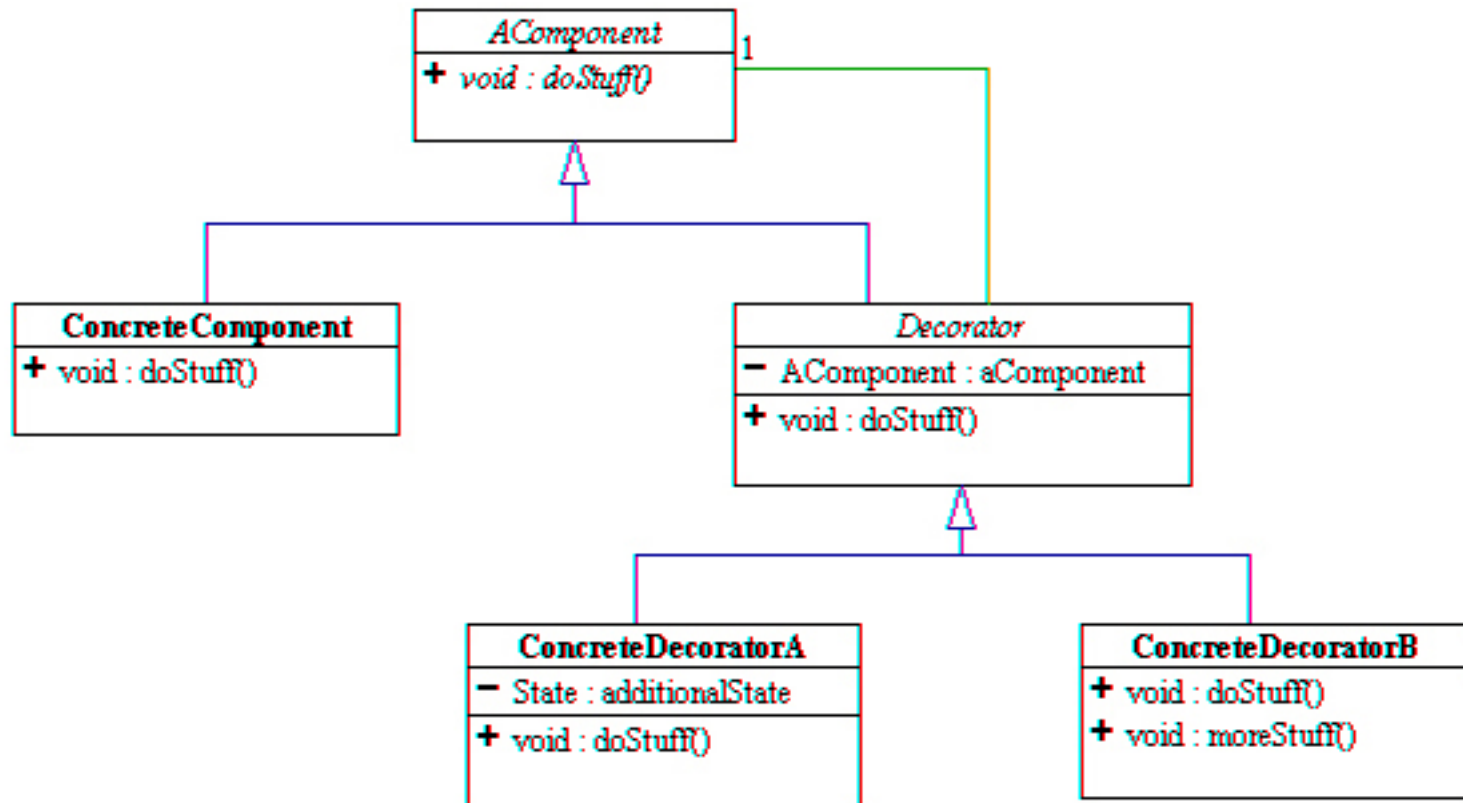
# 12 ingrédients différents

---

- **Mozarella, parmesan, Ham, Tomato, Mushrooms, diced onion, etc...**

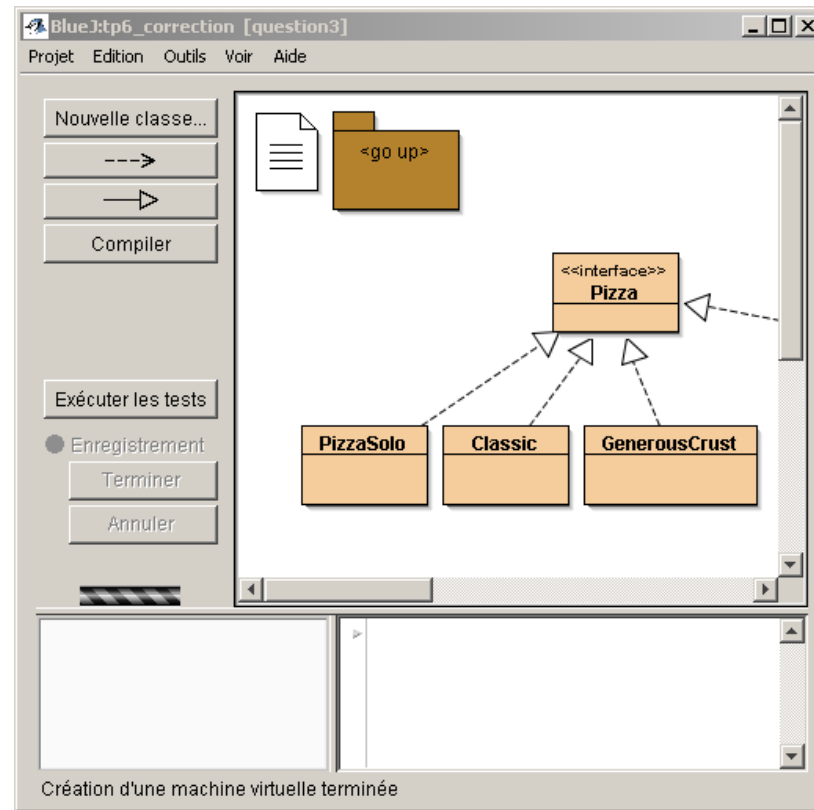


# Le décorateur de pizza



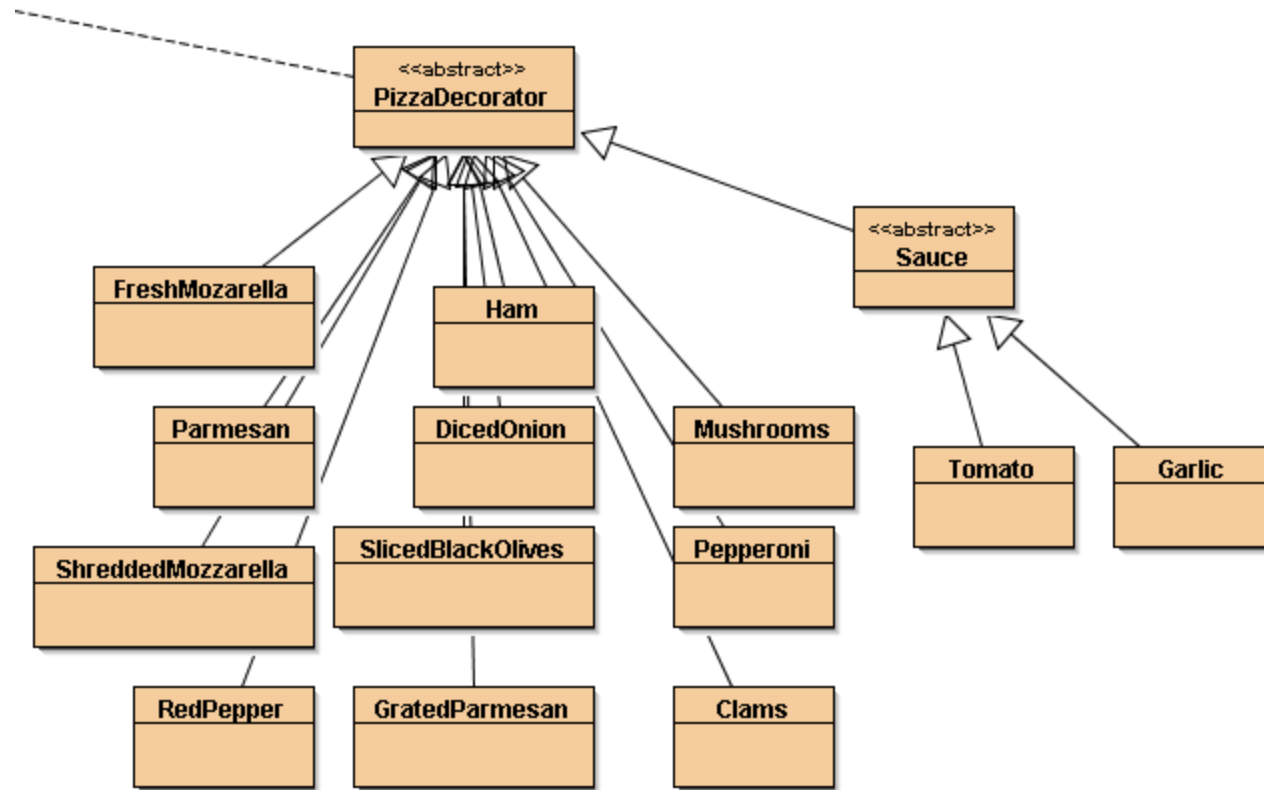
- AComponent --> une interface Pizza
- ConcreteComponent --> les différentes pâtes
- Decorator l'ingrédient, la décoration
- ConcreteDecorator Parmesan, Mozzarella, ...

# 3 types de pâte



```
public interface Pizza{  
    abstract public String getDescription();  
    abstract public double cost();  
}
```

# Les ingrédients



• !

# PizzaDecorator

---

```
public abstract class PizzaDecorator implements Pizza{  
    protected Pizza pizza;  
    public PizzaDecorator(Pizza pizza){  
        this.pizza = pizza;  
    }  
    public abstract String getDescription();  
    public abstract double cost();  
}
```

# Ham & Parmesan

---

```
public class Ham extends PizzaDecorator{
    public Ham(Pizza p){super(p);}
    public String getDescription(){
        return pizza.getDescription() + ", ham";
    }
    public double cost(){return pizza.cost() + 1.50;}
}
```

```
public class Parmesan extends PizzaDecorator{
    public Ham(Pizza p){super(p);}
    public String getDescription(){
        return pizza.getDescription() + ", parmesan";
    }
    public double cost(){return pizza.cost() + 0.75;}
}
```

# Pizza Solo + Mozzarella + quel coût ?

---

```
double coût = new Parmesan(  
    new FreshMozzarella(  
        new PizzaSolo()))).cost();  
assert coût == 5.8;
```

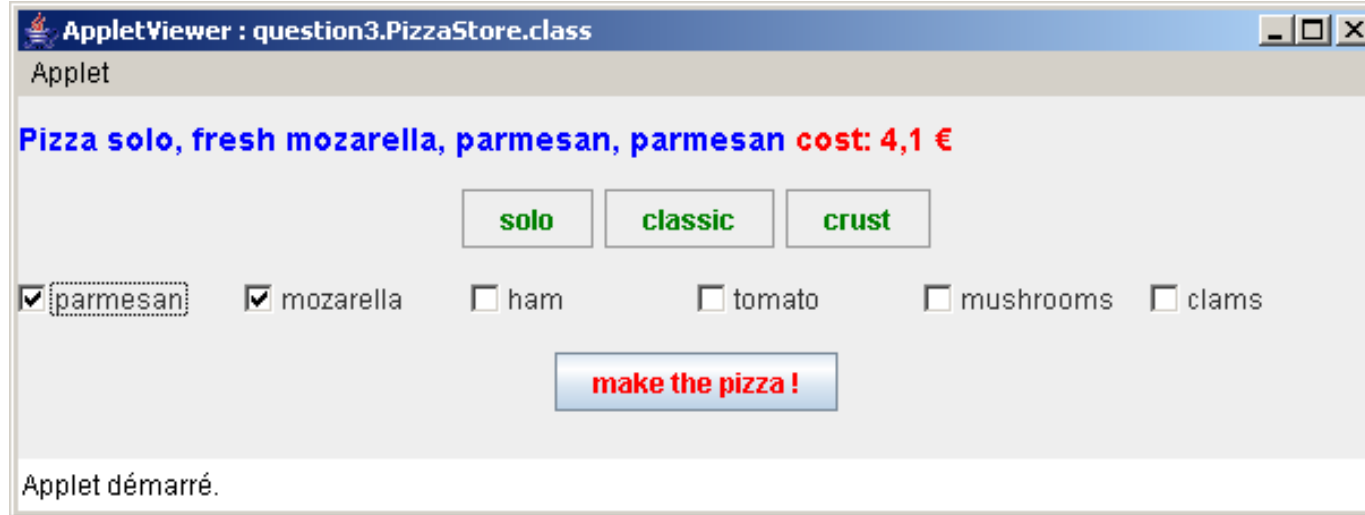
## ***Une pizza aux 2 fromages***

```
Pizza p=new Mozzarella(new Mozzarella(new Parmesan(new PizzaSolo()))));
```

***Magique ou liaison dynamique ???***



# L 'IHM du pizzaiolo



- **Pizza p; // donnée d 'instance de l 'IHM**
- **choix de la pâte, ici solo**

```
boutonSolo.addActionListener(  
    new ActionListener(){  
        public void actionPerformed(ActionEvent ae){  
            pizza = new PizzaSolo();  
            validerLesDécorations();  
        }  
    });
```

# L'IHM : les ingrédients ici Ham\*2



```
ham.addItemListener(new ItemListener(){
    public void itemStateChanged(ItemEvent ie){
        if(ie.getStateChange() == ItemEvent.SELECTED)
            pizza = new Ham(pizza);
        afficherLaPizzaEtSonCoût();
    }
});
```

L'applette est ici, mais la livraison n'est pas garantie

<http://jfod.cnam.fr/progAvancee/tp8/question1.PizzaStore.html>

# TexteDécoré

Texte décoré mais une seule fois par décoration ... une solution

```
public class B extends TexteDécoré{
    private static boolean décoré = false;

    public B(AbstractTexte texte){
        super(texte);
    }

    public String enHTML(){

        if(B.décoré){
            return super.enHTML();
        }else{
            B.décoré = true;
            String réponse = "<B>" + super.enHTML() + "</B>";
            B.décoré = false;
            return réponse;
        }
    }
}
```

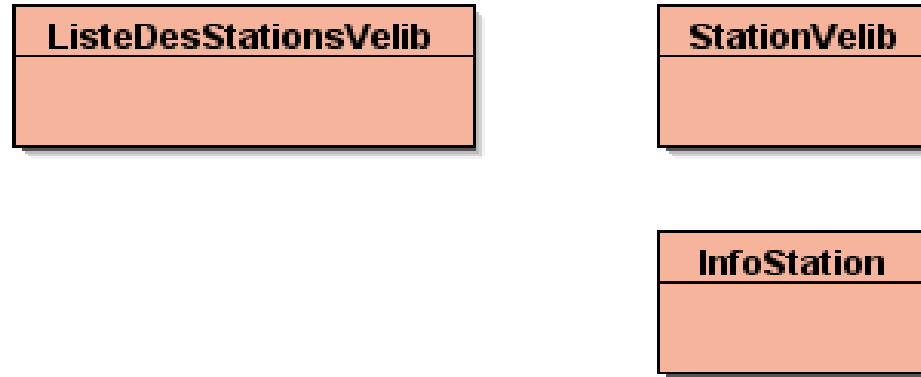
# Annexe Velib

---

---

# Les stations Vélib

---



- Les Classes, *un premier découpage*
  - **StationVelib**,
    - toutes les infos d'une station, (adresse, longitude, latitude,...)
  - **InfoStation**,
    - les informations comme le nombre de vélo et d'emplacements disponibles,...
  - **ListeDesStationsVelib**
    - La gestion de la liste des stations
  - <http://www.velib.paris.fr/service/carto>
  - <http://www.velib.paris.fr/service/stationdetails/{number}>

# Initialisation du « parser »

---

```
class ParserXML extends DefaultHandler {  
  
    public ParserXML(InputStream in)  
        throws Exception {  
  
        SAXParserFactory spf =  
            SAXParserFactory.newInstance();  
        SAXParser sp = spf.newSAXParser();  
  
        XMLReader xr = sp.getXMLReader();  
        xr.setContentHandler(this);  
        xr.parse(new InputSource(in));  
    }  
}
```

# startElement un extrait

---

**// Création d'une instance de la classe StationVelib  
// depuis XML en Java**

```
public void startElement(String uri, String localName, String qName, Attributes
    attributes) throws SAXException {

    super.startElement(uri, localName, qName, attributes);

    if(qName.equals("marker")){
        StationVelib station = new StationVelib();

        station.setName(attributes.getValue("name"));
        station.setNumber(Integer.parseInt(attributes.getValue("number")));
        station.setAddress(attributes.getValue("address"));
        station.setLatitude(Double.parseDouble(attributes.getValue("lat")));
        station.setLongitude(Double.parseDouble(attributes.getValue("lng")));

    }
```



# Démonstration

---

ListeDesStationsVelib

StationVelib

InfoStation

# Une Info à chaque Station

---

```
class ParserXML extends DefaultHandler {  
  
    private StringBuffer current; // la valeur  
  
    public ParserXML(int ID){  
        URL url = new URL(URL_VELIB_INFO + ID);  
  
        SAXParserFactory spf = SAXParserFactory.newInstance();  
        SAXParser sp;  
        sp = spf.newSAXParser();  
        XMLReader xr = sp.getXMLReader();  
        xr.setContentHandler(this);  
        xr.parse(new InputSource(url.openStream()));  
    }  
}
```

# A chaque noeud

---

```
public void startElement (String uri, String localName, String qName,  
    Attributes attributes) throws SAXException {  
    super.startElement(uri, localName, qName, attributes);  
    current = new StringBuffer();  
}
```

```
public void characters (char[] ch, int start, int length) throws SAXException {  
    super.characters(ch, start, length);  
    current.append(new String(ch, start, length));  
}
```

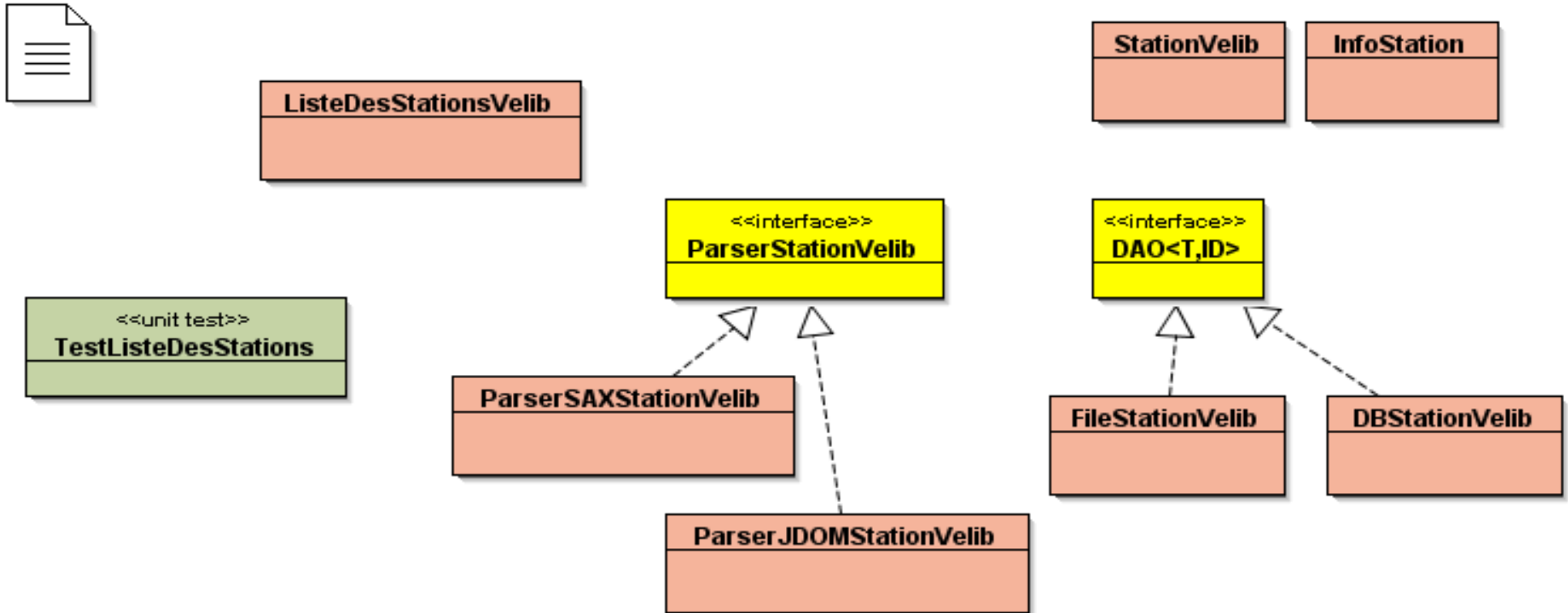
```
public void endElement (String uri, String localName, String qName)  
    throws SAXException {  
    super.endElement(uri, localName, qName);  
    if(qName.equals("available")){  
        available = Integer.parseInt(current.toString());  
        ...  
    }
```

# Démonstration ...

---

- **Un vélo est-il disponible ?**
  - **place d'Italie**
    - **N° 13008, 13010**
  - **55 rue Turbigo**
    - **N° 3011**

# Architecture Vélib refactoring



- Interface et couplage faible...
- Discussion

# Annexe

---

# Un autre exemple : la décoration de source Java

nhat.minh.le@huoc.org (Nhat Minh Lê)

```
33
34 @Invariant("size() >= 0")
35 public interface Stack<T> {
```

Returns the number of elements in this stack.

```
38
39 public int size();
```

Returns the topmost element of this stack without removing it.

```
43
44 @Requires("size() >= 1")
45 public T peek();
```

Pops the topmost element off this stack.

```
49
50 @Requires("size() >= 1")
51 @Ensures({
52     "size() == old (size()) - 1",
53     "result == old (peek())"
54 })
55 public T pop();
```

Pushes an element onto the stack.

```
59
60 @Ensures({
61     "size() == old (size()) + 1",
62     "peek() == old (obj)"
63 })
64 public void push(T obj);
65 }
```

- @Invariant
- @Requires
- @Ensures
- ???







# Programmation par contrats

- **Design by contracts**

- **B. Meyer 1992**

- <https://archive.eiffel.com/doc/manuals/technology/contract/>

The benefits of Design by Contract include the following:

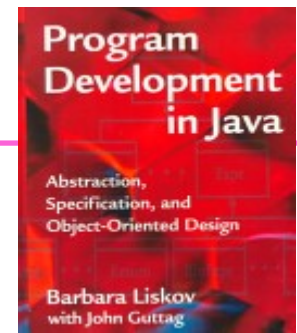
-  A better understanding of the object-oriented method and, more generally, of software construction.
-  A systematic approach to building bug-free object-oriented systems.
-  An effective framework for debugging, testing and, more generally, quality assurance.
-  A method for documenting software components.
-  Better understanding and control of the inheritance mechanism.
-  A technique for dealing with abnormal cases, leading to a safe and effective language construct for exception handling.

- **B.Liskov & J. Guttag program Development in Java**

- **Chapitres 6 et 7**

- **Décoration comme documentation de sources java**





- **Documentation, décoration d'une classe ?**
  - Pour les utilisateurs : javadoc
  - Pour les implémenteurs ? les développeurs ? les successeurs ?
    - B. Liskov dans son livre propose deux notions
      - **Fonction d'abstraction af()**  
et
      - **Invariant de représentation repOk**
- Comme commentaires dans le code ou bien en méthodes
- Page 99 de son livre, page suivante ...

## Aids to Understanding Implementations

In this section, we discuss two pieces of information, the abstraction function and the representation invariant, that are particularly useful in understanding an implementation of a data abstraction.

The *abstraction function* captures the designer's intent in choosing a particular representation. It is the first thing you decide on when inventing the rep: what instance variables to use and how they relate to the abstract object they are intended to represent. The abstraction function simply describes this decision.

The *rep invariant* is invented as you investigate how to implement the constructors and methods. It captures the common assumptions on which these implementations are based; in doing so, it allows you to consider the implementation of each operation in isolation of the others.

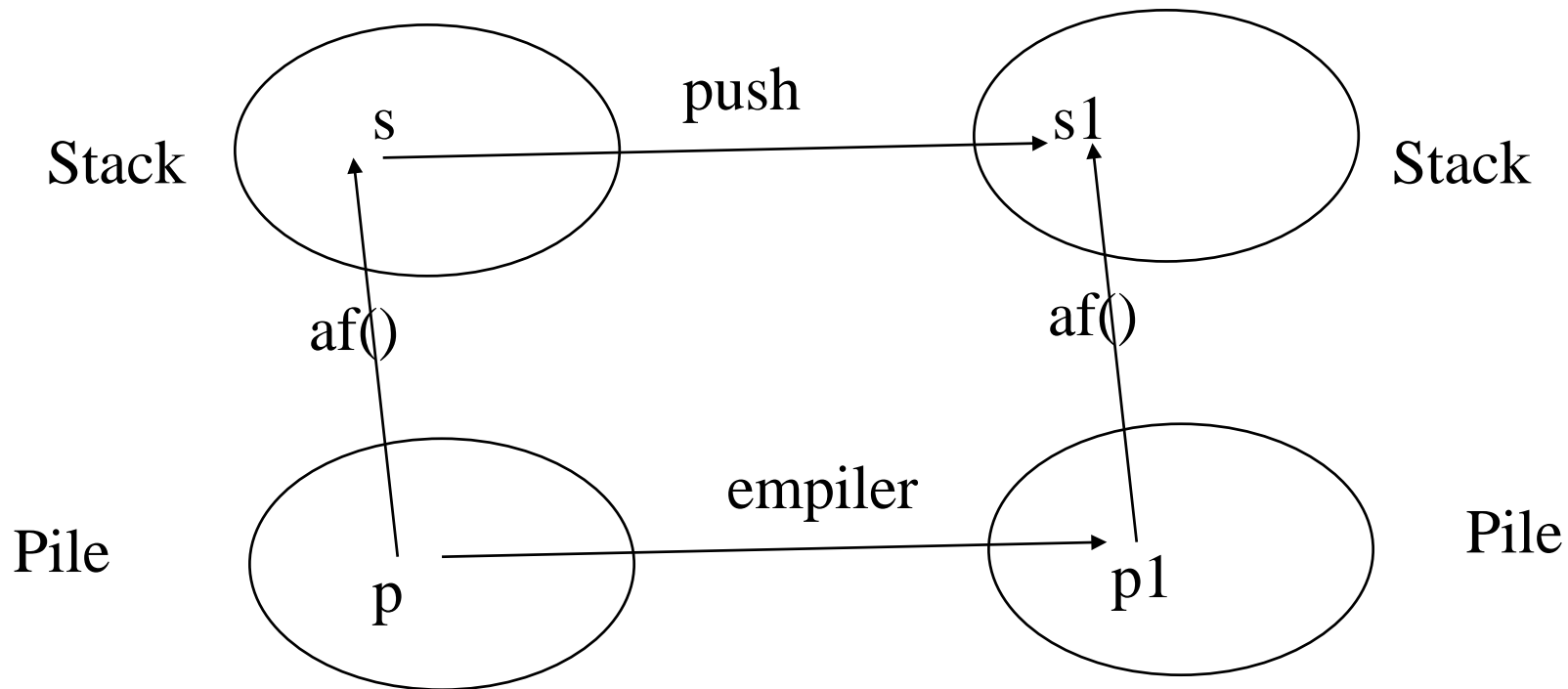
The abstraction function and rep invariant together provide valuable documentation, both to the original implementor and to others who read the code. They capture the reason why the code is the way it is: for example, why the implementation of `choose` can return the *zero<sup>th</sup>* element of `els` (since the elements of `els` represent the elements of the set), or why `size` can simply return the size of `els` (because there are no duplicates in `els`).

Because they are so useful, both the abstraction function and rep invariant should be included as comments in the code. This section describes how to define them and also how to provide them as methods.



# repOk == invariant, af, pre, post

- **repOk**
  - Invariant de représentation, invariant de classe
- **af**
  - Fonction d'abstraction



`p.af().push(44).equals(p.empiler(44).af())`

# Pre Post assertions

---

- **pre, post**
  - **pre\_assertion** vrai à l'appel de la méthode,
  - **post\_assertion** vrai après l'exécution de la méthode,
    - **Pre, post** : un contrat pour l'appelant
- **assert Expression Booléenne** : " un commentaire " ;
  - Un prédéfini java, javac -ea

## Invariant, pre et post assertion, héritage

- Un **invariant** de classe doit être vrai avant et après l'appel de chaque méthode de la classe et à la sortie du constructeur (repOk)
- La **pré assertion** précise le contrat que doit respecter l'appelant
- La **post assertion** garantit le retour pour l'appelant
  - Framework existants
    - Cf. cofoja, JML, Contract4j, Jass-Modern, UML + OCL ...

# pré et post assertions

---

- Exemple : ajout d'un élément dans une liste

```
pre_assertion( element != null );
```

```
void ajouter(E element){ ... }
```

```
post_assertion( tailleAvant + 1 == tailleAprès );
```

- Serait-ce une décoration de la méthode ajouter ?

# Invariant de classe, repOk==invariant, af

---

- Exemple : la liste suite

```
private E[] tableau = .....
```

```
Liste(){ ...}
```

```
invariant( tableau != null)
```

```
invariant( tableau != null)
```

```
void ajouter(E element){ ... }
```

```
invariant( tableau != null)
```

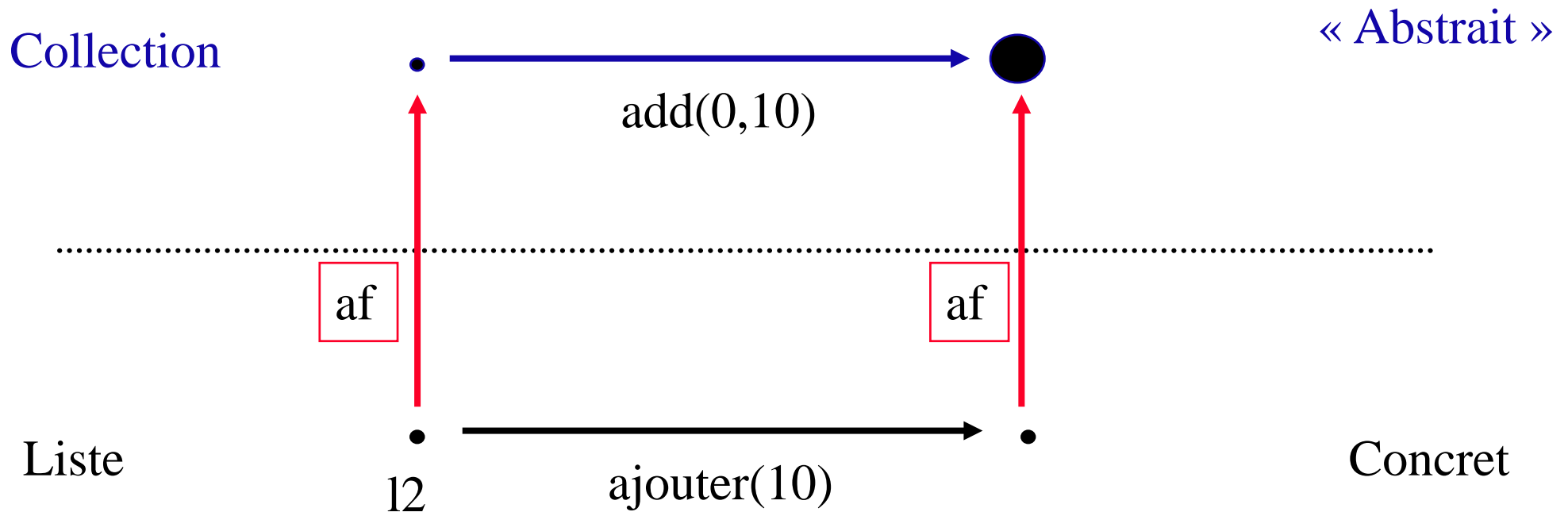
```
// fonction d'abstraction, de la liste en java.util.Collection
```

```
public Collection af(){ return .....
```

- Serait-ce aussi une décoration du constructeur et de la méthode ajouter ?

`af()` ou `l2.af().add(0,10).equals(l2.ajouter(10).af())` ?

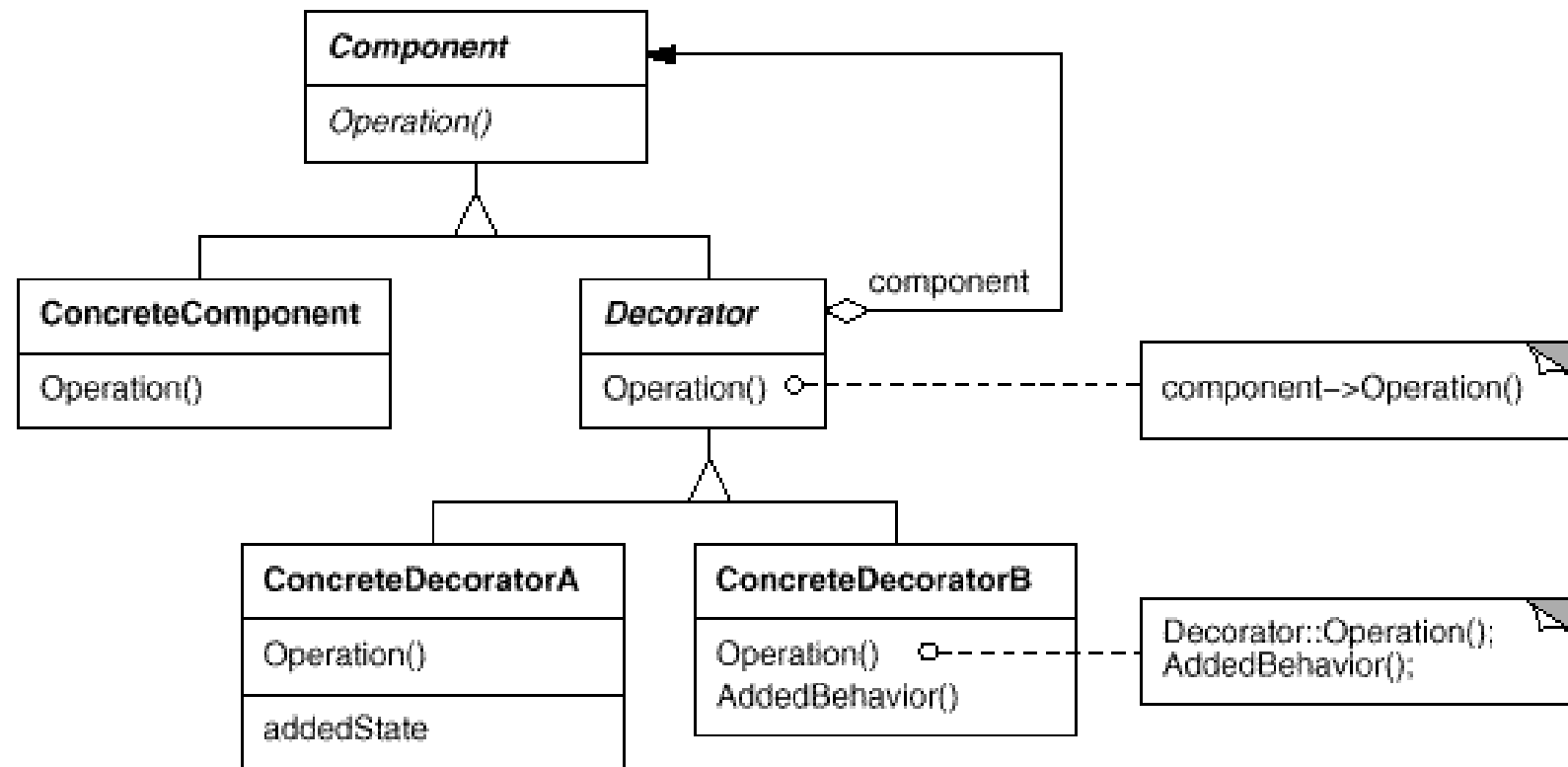
- Pour chaque opération sur la liste :
  - nous devons vérifier que le graphe commute ...



- Discussion, décoration, assertion



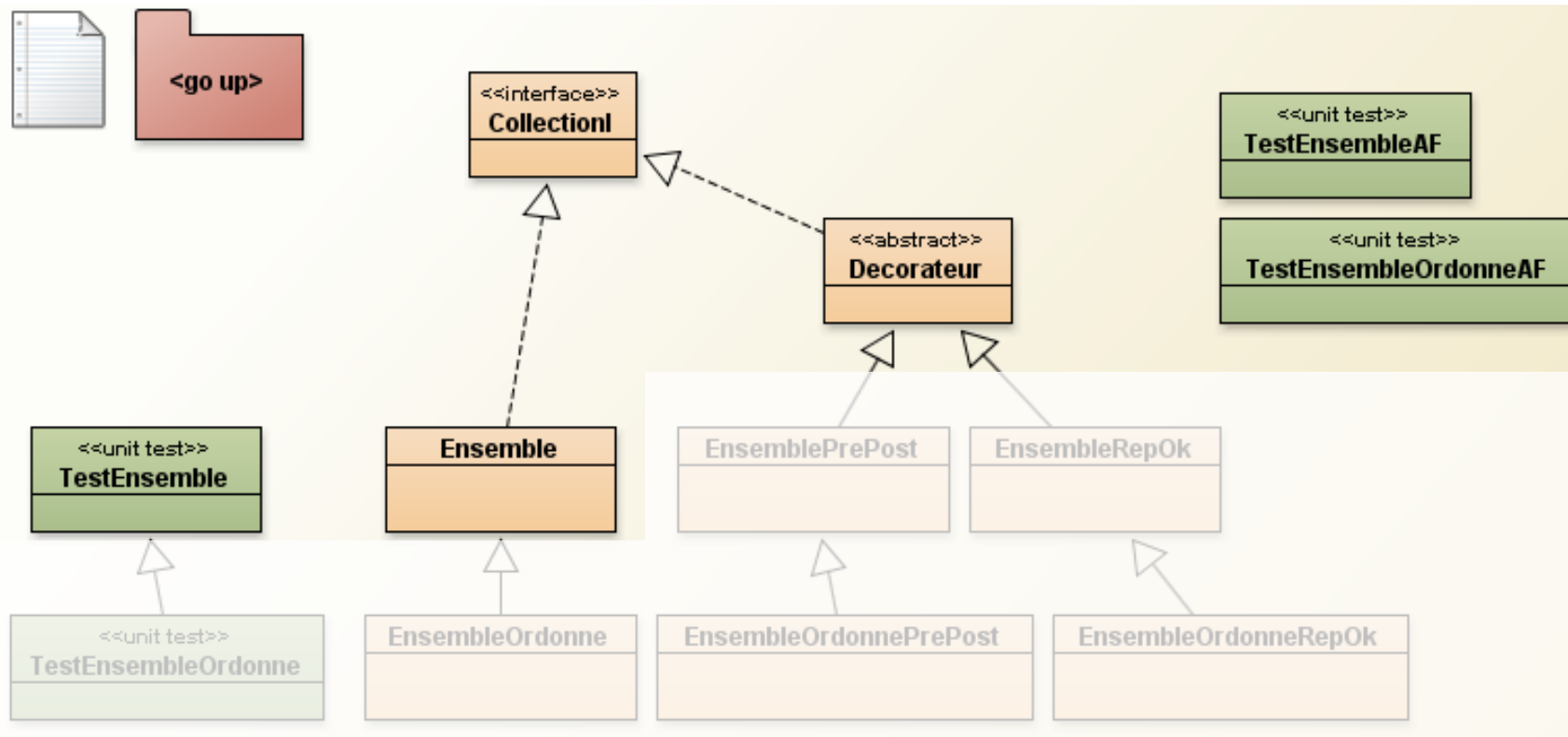
# Usage du Décorateur pour la classe Ensemble



- La classe Ensemble : **ConcreteComponent**
- repOk, pre-post assertion : **ConcreteDecoratorA, B**

# Un exemple : la classe Ensemble

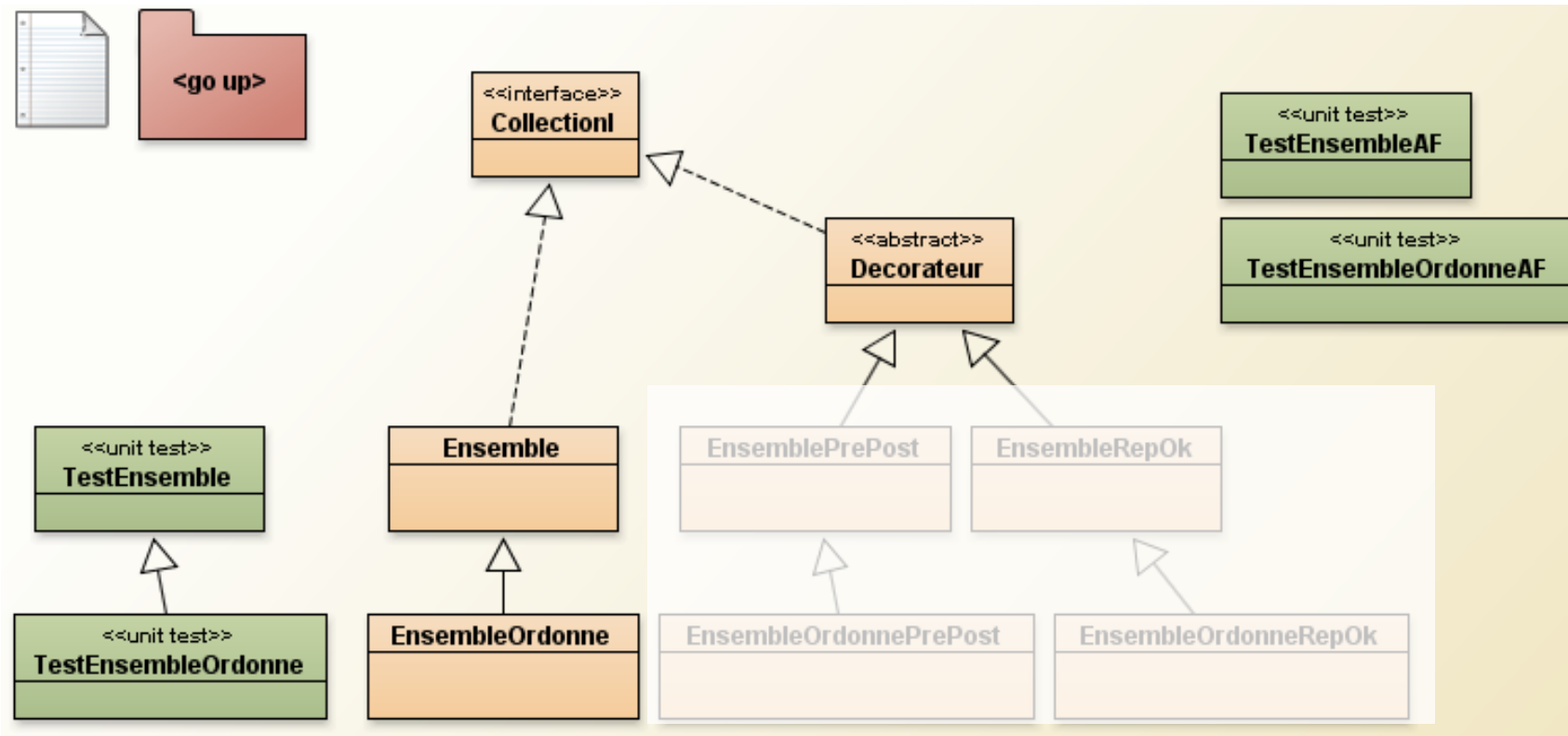
- class Ensemble implements CollectionI



- Discussion

# Un exemple : la classe Ensemble

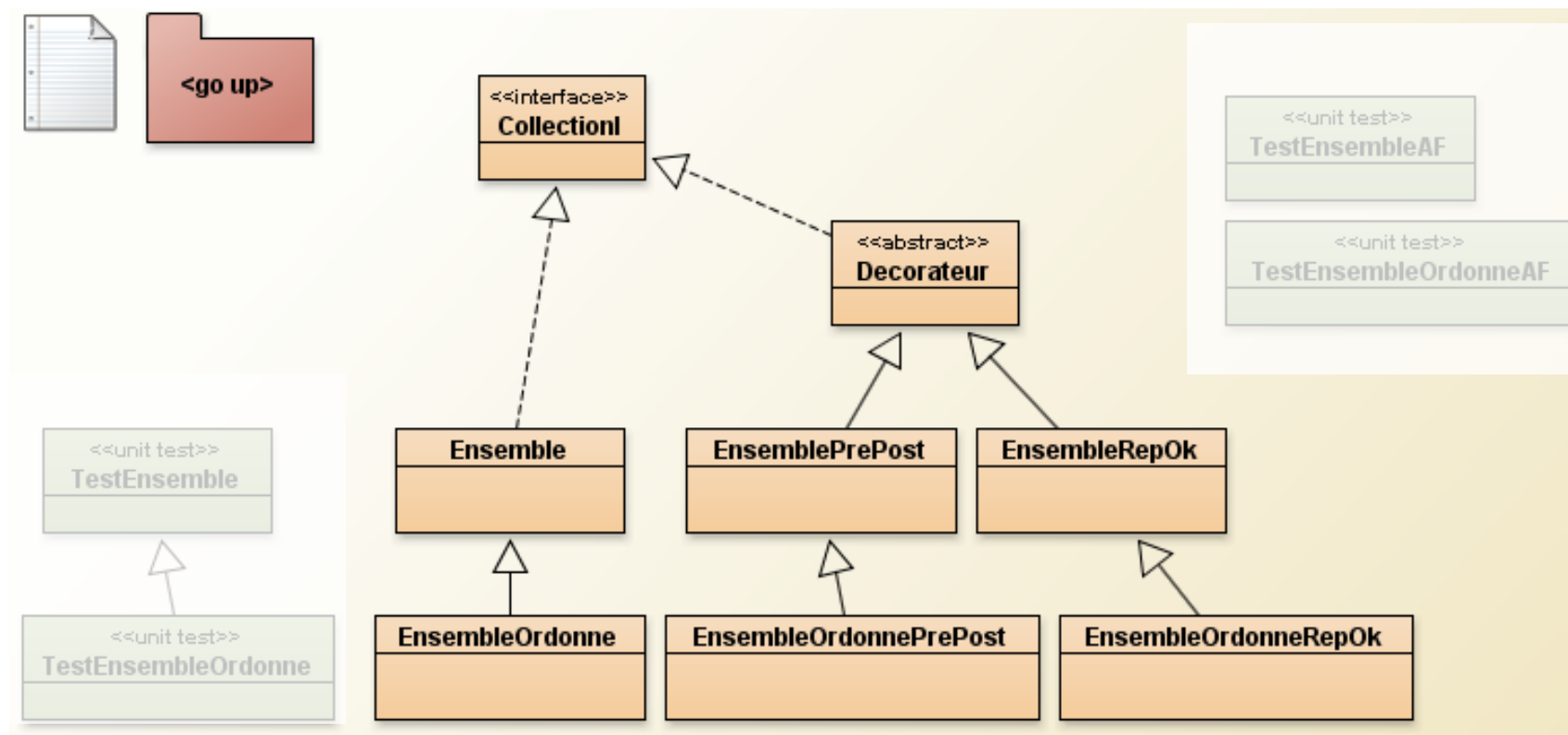
- class Ensemble extends EnsembleOrdonne



- Discussion suite

# Décorations pre-post, invariant

- class Ensemble extends EnsembleOrdonne



- Décorations = pré-post assertions et invariant de classe

# CollectionI

```
3
4 public interface CollectionI extends Iterable<Integer>{
5
6     public void ajouter(int i);
7     public void ajouter(CollectionI I);
8     public void retirer(int i);
9
10    public boolean contient(int i);
11    public int taille();
12
13    /** Invariant de classe. cf. B. Liskov.*/
14    public boolean repOk();
15    /** Fonction d'abstraction. */
16    public Object af();
17
```

- Une collection
  - La classe Ensemble est une collection

# Tests unitaires, TestEnsemble

```
public class TestEnsemble extends junit.framework.TestCase{
    protected CollectionI e,e1;

    protected void setUp(){
        // la collection est décorée
        this.e = new EnsemblePrePost(new EnsembleRepOk(new Ensemble()));

        this.e1 = new EnsemblePrePost(new EnsembleRepOk(new Ensemble()));
    }

    public void testAjouter(){
        e.ajouter(3);
        assertTrue(e.contient(3));
        assertEquals(1, e.taille());
        e.ajouter(3);
        assertEquals(1, e.taille());
        assertTrue(e.contient(3));
        e.ajouter(2);
        assertEquals(2, e.taille());
        assertTrue(e.contient(2));
    }
}
```

# Décoration : EnsembleRepOk

```
3 public class EnsembleRepOk extends Decorateur{
4
5     public EnsembleRepOk( CollectionI c){
6         super(c);
7         assert c.repOk() : " repOk invalide !";
8     }
9
10    public void ajouter(int i){
11        assert super.repOk() : " repOk invalide !";
12        super.ajouter(i);
13        assert super.repOk() : " repOk invalide !";
14    }
15    public void ajouter(CollectionI c){
16        assert super.repOk() : " repOk invalide !";
17        super.ajouter(c);
18        assert super.repOk() : " repOk invalide !";
19    }
```

- **Invariant de classe**
  - Avant et après chaque exécution de méthode
  - À la sortie du constructeur

# Décoration EnsemblePrePost

```
4 public class EnsemblePrePost extends Decorateur{
5
6     protected boolean preajouter(int i){
7         initVariables(i);
8         return true;
9     }
10    protected boolean postajouter(int i){
11        return (tailleAvant+1 == super.taille() && !dejaPresent && super.contient(i)) ||
12               (tailleAvant == super.taille() && dejaPresent && super.equals(ensembleAvant));
13    }
14    public void ajouter(int i){
15        assert preajouter(i) : "pre assertion ajouter invalide !";
16        super.ajouter(i);
17        assert postajouter(i) : "post assertion ajouter invalide !";
18    }
```

- **Pre, post assertions en place**
  - **DejaPresent, tailleAvant, ensembleAvant**
    - **Variables d'instance de la classe de tests**



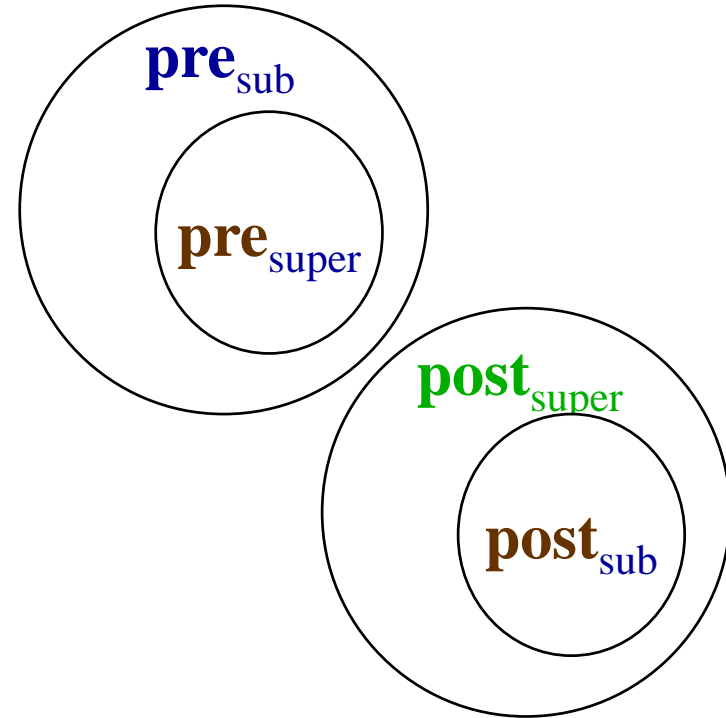
# Héritage

- Héritage d'assertions : héritage de décorations

- Que dit B. Liskov ? Page 176

–  $\text{pre}_{\text{super}} \rightarrow \text{pre}_{\text{sub}}$

–  $\text{pre}_{\text{super}} \ \&\& \ \text{post}_{\text{sub}} \rightarrow \text{post}_{\text{super}}$



- En Eiffel,

–  $\text{pre}_{\text{super}} \parallel \text{pre}_{\text{sub}}$

–  $\text{post}_{\text{super}} \ \&\& \ \text{post}_{\text{sub}}$

- Discussions ... décorons nos décorations

# Les décorations sont héritées

```
1
2
3
4
5 public class EnsembleOrdonnePrePost extends EnsemblePrePost{
6
7     protected boolean pre_ajouter(int i){
8         return true;
9     }
10    protected boolean post_ajouter(int i){
11        return estOrdonne();
12    }
13
14    public void ajouter(int i){
15        // Eiffel, cofoja, etc ...
16        assert super.pre_ajouter(i) | this.pre_ajouter(i);
17        // pre_super ==> post_sub    // page 176 Liskov
18        assert !super.pre_ajouter(i) | this.pre_ajouter(i) : "pre assertion ajouter invalide !";
19        super.ajouter(i);
20        // (pre_super && post_sub) ==> post_super Liskov
21        assert !(super.pre_ajouter(i) & this.post_ajouter(i)) | super.post_ajouter(i) : "post assertion a
22        // Eiffel
23        assert this.post_ajouter(i) & super.post_ajouter(i) : "post assertion ajouter invalide !";
24    }
25 }
```

- $\text{pre}_{\text{super}} \parallel \text{pre}_{\text{sub}}$
- $\text{post}_{\text{super}} \&\& \text{post}_{\text{sub}}$

# démonstration

---

- [http://jfod.cnam.fr/NFP121/assertions\\_et\\_decorateur.jar](http://jfod.cnam.fr/NFP121/assertions_et_decorateur.jar)

# Conclusion édulcorée

---

- **C'était un exemple de Génie logiciel ...**