# NFP121, Cnam/Paris
# Cours 9
# Injection de dépendance

jean-michel Douin, douin au cnam point fr
version : 1 Décembre 2016

**Notes de cours**

# Sommaire

- **A la recherche du Couplage faible …**
- **Injection de dépendance**
  - **Différentes approches**
  - **Patrons de conception**
- **Inversion de contrôle et/ou Injection de dépendance**
  - **Cf. le cours 3-2, l'article de Martin Fowler**
- **Configuration et utilisation**
  - **XML, texte et/ou API**
- **Le « framework Cnam » femtoContainer**
- **Le patron Builder en annexe**
- *Framework sur ce principe*
  - *Spring, Struts,…*
  - *À l'aide d'annotations (@Inject..) (cf. JSR330)*
    - *Google Guice, GWT, EJB 3.0,…*

# Principale bibliographie

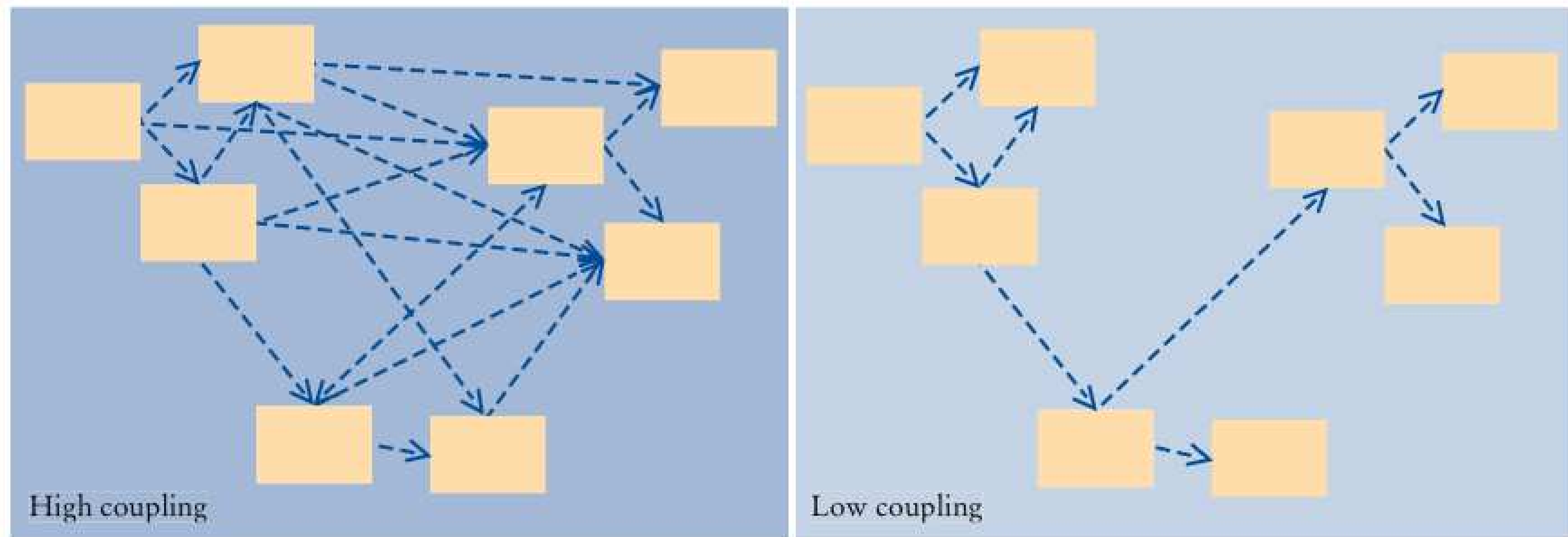
- **Ces deux références sont à lire impérativement**
  - **Martin Fowler**
  - **Inversion of Control Containers and the Dependency Injection pattern**
    - **http://martinfowler.com/articles/injection.html**

**Best-Practice Software Engineering**

  - **Une bonne adresse : le site de l'université de Vienne où tout est en anglais ☺**
    - **http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/dependency_injection.html**
      - **sauf <<The book is in German …>>**

- Quelques Framework/Canevas
  - **http://www.springsource.org/**
  - **http://picocontainer.org/**
  - **http://code.google.com/p/google-guice/**
    - **http://google-guice.googlecode.com/svn/trunk/javadoc/packages.html**
  - **@Inject javax.inject.Inject**
    - **http://download.oracle.com/javaee/6/api/javax/inject/package-summary.html**
  - **Un exemple qui vaut le détour**
    - **http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection**

# Couplage faible




- Source : https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/MyBigJava/Ch07/ch07.html

- Comment ?
- Quels outils de mesure ?

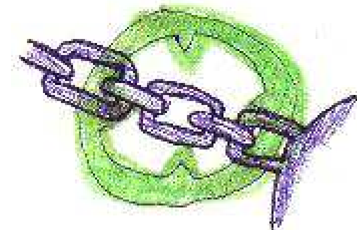
# Couplage faible, pourquoi faire ?

- **Le bon sens**
  - **A dépend de B qui dépend de C …**
  - **A dépend de B qui dépend de A …**
    - **Cohésion à revoir …**

- **Avantages escomptés**
  - **Maintenance**
  - **Substitution d'une implémentation par une autre**
  - **Tests unitaires facilités**

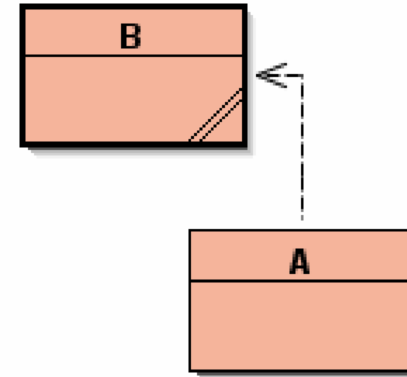
- **Usage de patrons**
  - **Une solution**

- Un outil de mesure du couplage (parmi d'autres) : DependencyFinder
  - http://depfind.sourceforge.net/
  - *Un exemple les classes A et B sont en couple*



# Exemple : les classes A et B

```
public class A{
  private B b;

  public A(){
    this.b = new B();
  }
  public void m(){
    this.b.q();
  }
}
public class B{}
```




**A dépend de B → couplage fort de ces classes …**

**(A extends B idem …)**

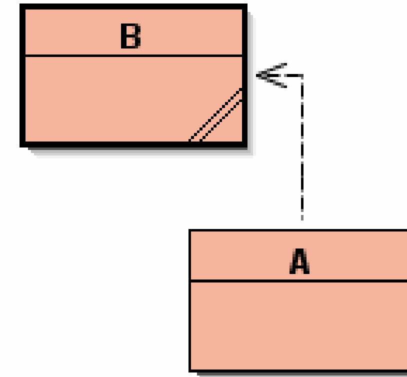
# Des mesures, outil DependencyFinder

```
    A
        --> java.lang.Object *
        A()
            --> A.b
            --> B *
            --> B.B() *
            --> java.lang.Object.Object() *
        b
            <-- A.A()
            --> B *
    B *
        <-- A.A()
        <-- A.b
        B() *
            <-- A.A()
java.lang *
    Object *
        <-- A
        Object() *
            <-- A.A()
```

- **A dépend de B … et de Object** *on le savait déjà…*

# A dépend de B

```
public class A{
  private B b;

  public A(){
    this.b = new B();
  }
}
public class B{}
```




**Comment A pourrait ne plus dépendre de B**

**Comment supprimer ce *défaut* ?**

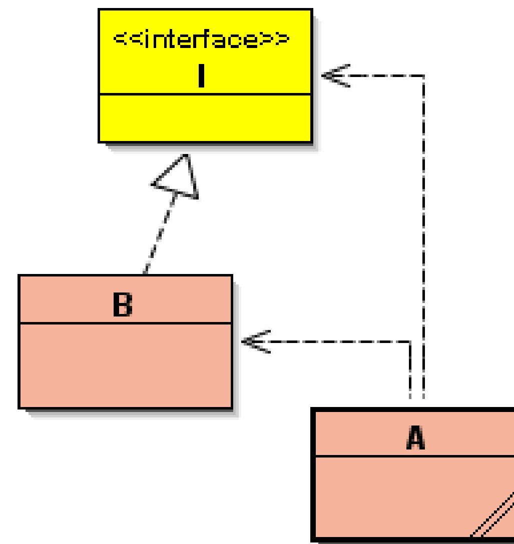
**Est-ce un *défaut* ?**

**cf. le patron délégation**

**-> une solution: la notion d'interface en java**

# Une interface I, les classes A et B

```
public class A{
  private I i;

  public A(){
    this.i = new B();
  }
}

public class B implements I{}
public interface I{}
```



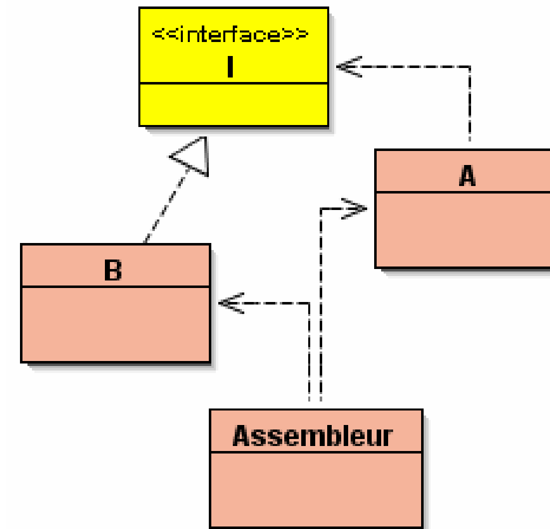

**A dépend toujours de B,**
**-> la notion d'interface … ne suffit pas,**

**à moins que**

nous puissions ajouter un paramètre de type I au constructeur de A (cela implique une *délocalisation* de B…)

# Exemple suite : Une interface I, les classes A, B et +

```
public class A{
  private I i;

  public A(I i){
    this.i = i;
  }
}

public class B implements I{}
```




**A ne dépend plus de B,**

Nécessité d'un assembleur,

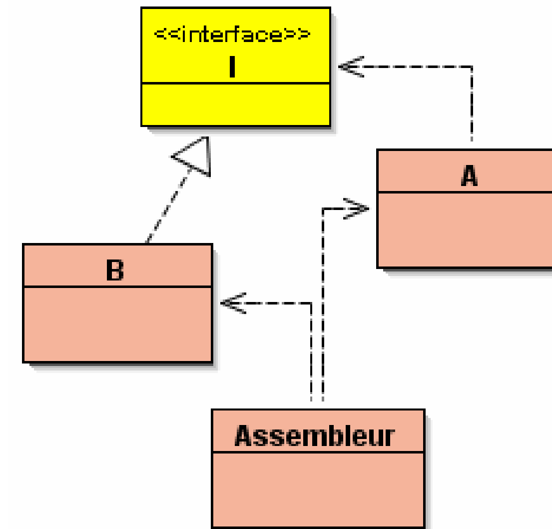
```
I référence = new B();
a = new A( référence);
```

Délocalisation de B, `I référence = new B()`,

`I` comme injecteur de référence …

# Exemple suite : Une interface I, les classes A, B et +

```
public class A{
  private I i;

  public A(I i){
    this.i = i;
  }
}

public class B implements I{}

public class Assembleur{
  Assembleur(){
    A a = new A( new B());
  }
}
```




-> Nous injectons une référence lors de l'appel du constructeur de A

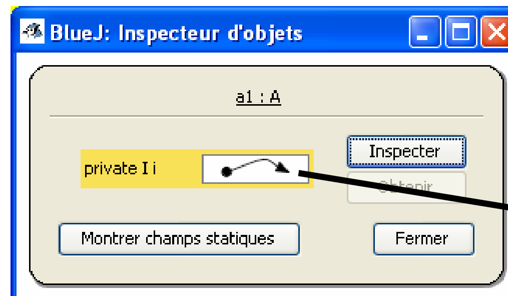
*-> Nous créons donc une dépendance en injectant cette référence*

*Serait-ce alors une injection de dépendance ?*

# Injection en images animées

1) new Assembleur()








2) new A( new B())

injection !



de cette dépendance

- **Assembleur comme injecteur**

# A ne dépend plus de B, la preuve…

- **Graphe pour A**

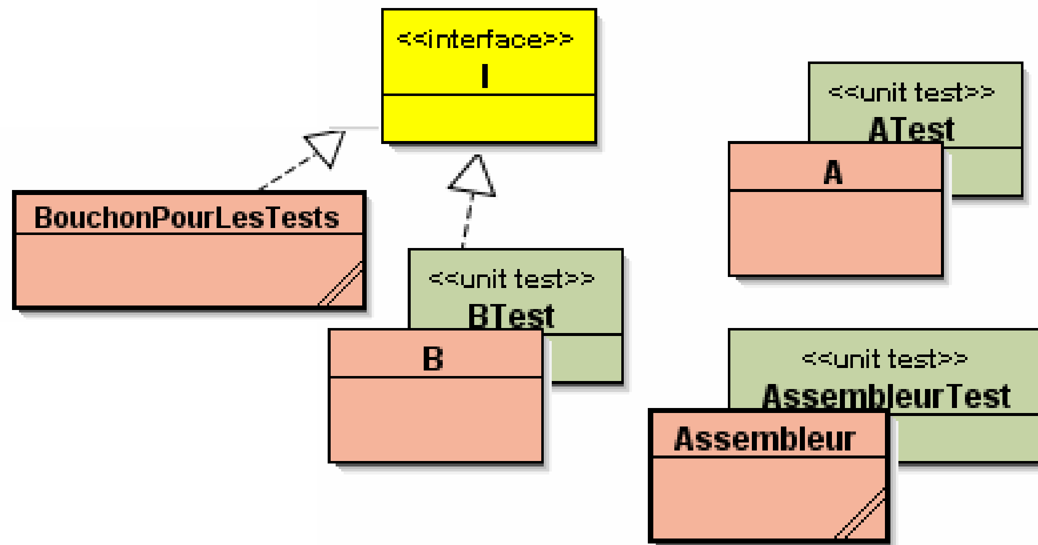
```
    A
        --> java.lang.Object *
        A(I)
            --> A.i
            --> I *
            --> java.lang.Object.Object() *
        i
            <-- A.A(I)
            --> I *
    I *
        <-- A.A(I)
        <-- A.i
java.lang *
    Object *
        <-- A
        Object() *
            <-- A.A(I)
```

- **Graphe pour Assembleur**

```
    A *
        A(I) *
            <-- Assembleur.Assembleur()
    Assembleur
        --> java.lang.Object *
        Assembleur()
            --> A.A(I) *
            --> B.B() *
            --> I *
            --> java.lang.Object.Object() *
    B *
        B() *
            <-- Assembleur.Assembleur()
    I *
        <-- Assembleur.Assembleur()
java.lang *
    Object *
        <-- Assembleur
        Object() *
            <-- Assembleur.Assembleur()
```

- **DependencyFinder**

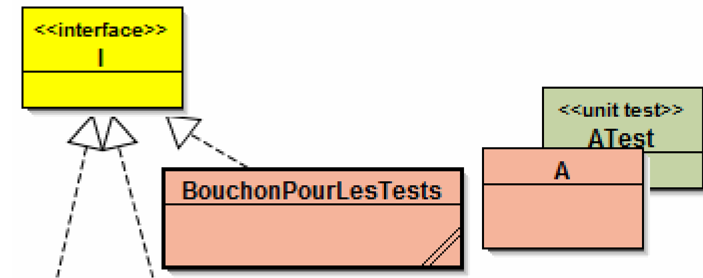
# Premier gain : les tests unitaires sont plus simples




- **De vrais tests à l'unité …**

  Des *bouchons* pour les tests peuvent être utilisés
  [http://easymock.org/](http://easymock.org/)

# Premier gain: Tests unitaires




En attendant la classe B, tests unitaires de A

```
public class ATest extends junit.framework.TestCase{

  A a = new A( new BouchonPourLesTests());
```

# Gain suite : substitution d'une implémentation

Un « assembleur » contient ce code …

```
A a = new A( new B());
```

La substitution d'une classe par une autre est réussie !

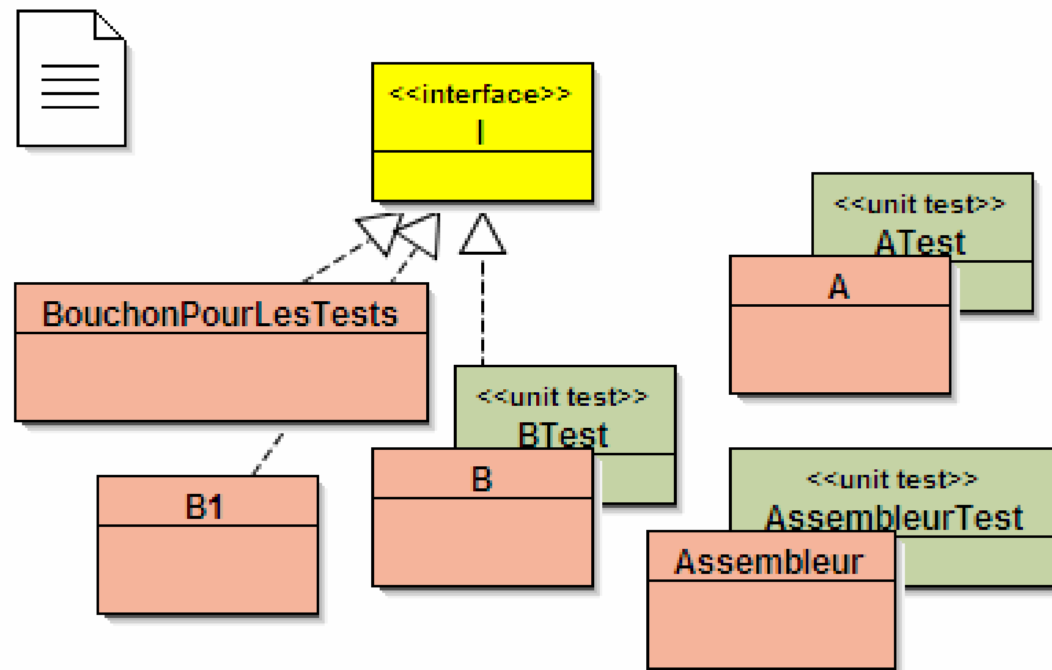
il suffit d'écrire

pour une nouvelle classe B1,

```
A a = new A( new B1());
```

etc…

# Démonstration



# Assembleur et configuration

- La substitution d'une classe par une autre
  *nécessite une modification du code de l'assembleur* …

- Peut-on supprimer cette modification du source Java ?

Par: Un fichier de configuration …

un fichier XML ? CSV ? Excel …

un fichier texte ?

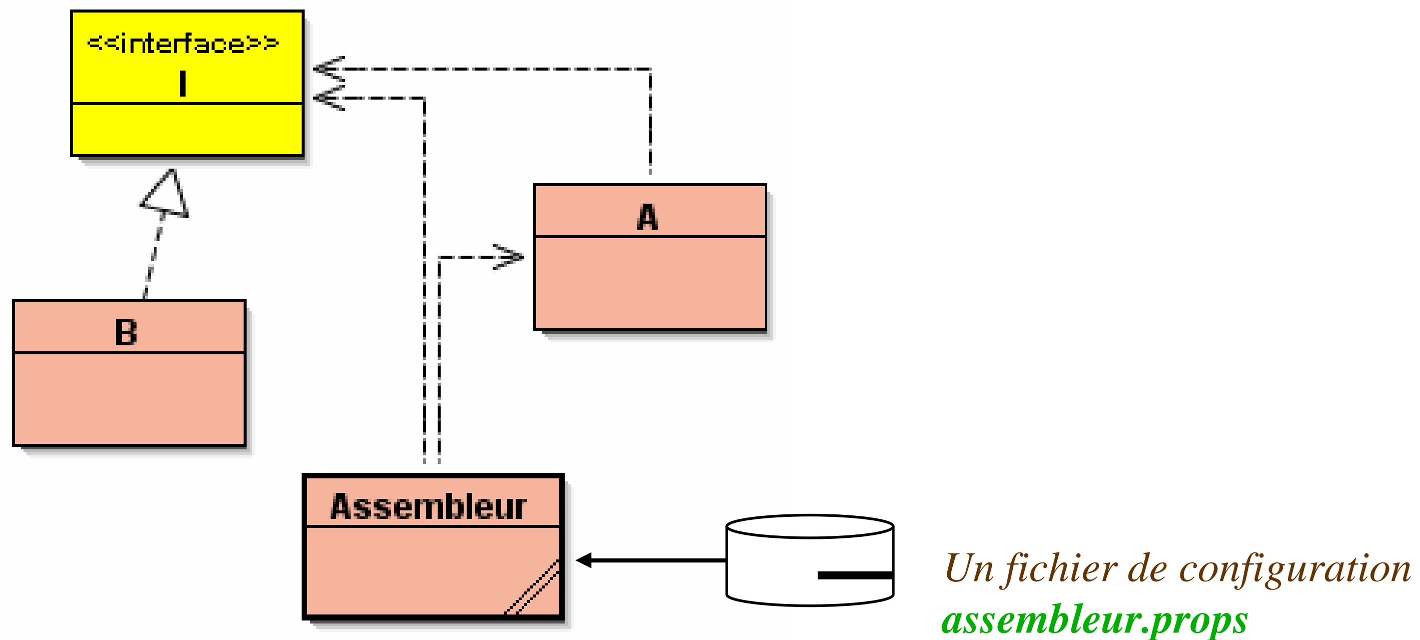
Un fichier de propriétés, par exemple:

```
implementation_i_class=B1
```

**La clef** (implementation_i_class) — **Le nom de la classe** (B1)

# Une variante de l'assembleur/configurateur

- **Depuis un fichier, l'assembleur extrait le nom de la classe**
  - **Depuis un fichier texte, ou fichier de configuration**

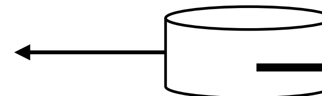


# Séparation Configuration / Utilisation

– **Un exemple :**

- **le fichier *assembleur.props* contient cette ligne**

  – **implementation_i_class=B1**



*Un fichier*
***assembleur.props***

  » **implementation_i_class est la clef, B1 est la valeur**

  » **Cf. props.load(un_fichier);**

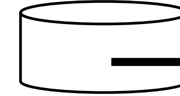
– A la lecture de ce fichier,

- *Classe java.util.Properties, méthode load*

– Une instance de B1 est créée par introspection

- *Class<?> c = Class.forName(" B1 " );*
- *Object o = c.newInstance(); // appel du constructeur par défaut ici B1()*

# Fichier de configuration et Assembleur



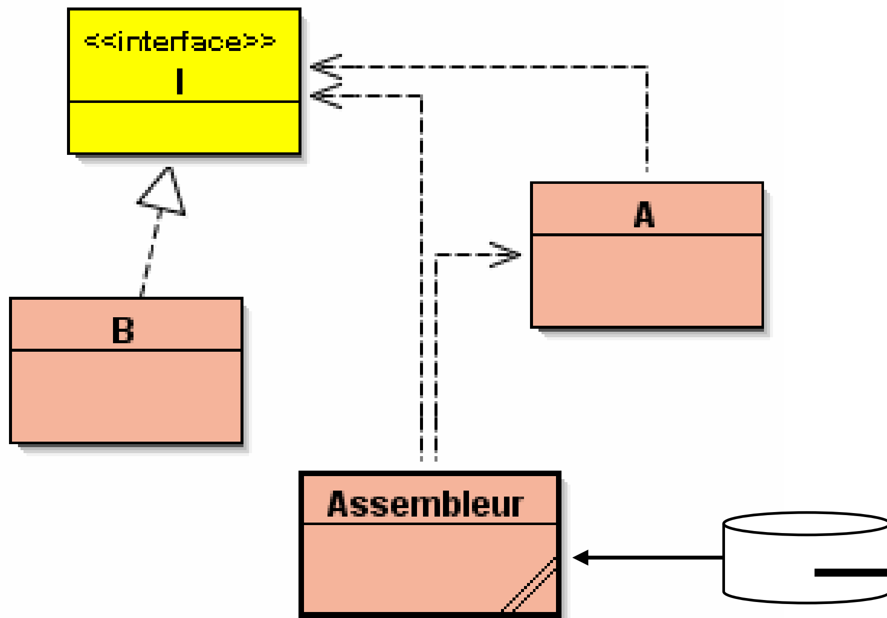
*Un fichier*
*assembleur.props*

```
public Assembleur() throws Exception{
    Properties props = new Properties();

// chargement de la table des propriétés par une lecture du fichier
    props.load(new FileInputStream(new File("assembleur.props")));

// chargement de la classe, à partir de la clef
    Class<?> c =
        Class.forName(props.getProperty("implementation_i_class"));

// création d'une instance, appel du constructeur par défaut
    I i = (I)c.newInstance();

// injection de cette instance, à la création de a
     A a = new A(i);
  }
```

# Démonstration

- **Une nouvelle classe B1 est proposée …**
- **-> Modification du fichier de configuration**
  - Et c'est tout
- **Couplage faible, maintenance aisée, est-ce réussi ?**
- **Discussion …**

# Une variante la configuration est en xml




*Un fichier*
***assembleur.props.xml***

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>assembleur config</comment>
    <entry key="implementation-i-class">B</entry>
</properties>
```

- **Même principe**
  - **Cf. la méthode props.loadFromXML**

# Constat et questions choisies

- L'assembleur/injecteur contient l'instance de la classe A
  - Un **conteneur** somme toute …
  - Et les paramètres du constructeur de la classe à injecter ???
    - implementation_i_class=B1
    - implementation_i_class.param.1=param_1
    - implementation_i_class.param.2=param_2
    - implementation_i_class.param.3=param_3
      - Adoption d'une convention de nommage des paramètres …

# La suite, sommaire intermédiaire

**Autres syntaxes pour la même chose**
- Injection d'interface
- À l'aide d'un mutateur

- **Un autre exemple**
  - Un cours, des auditeurs …
  - Fichier de configuration en XML

- **Objectif à ne pas perdre de vue:**
  - Vers une <u>séparation effective</u> de la configuration et de l'utilisation

# Autre syntaxe possible : Injection d'interface

```
public interface Inject{
  void inject(I i);
}

public class A implements Inject{
  private I i;

  public void inject(I i){
    this.i = i;
  }
}
public class B implements I{}


public class Assembleur{
  Assembleur{
    A a = new A();
    a.inject(new B());// ou bien à l'aide d'un fichier de configuration
}
```

Une méthode dédiée issue de l'interface,
Affectation de l'attribut comme l'injection par un constructeur

# Un autre exemple…

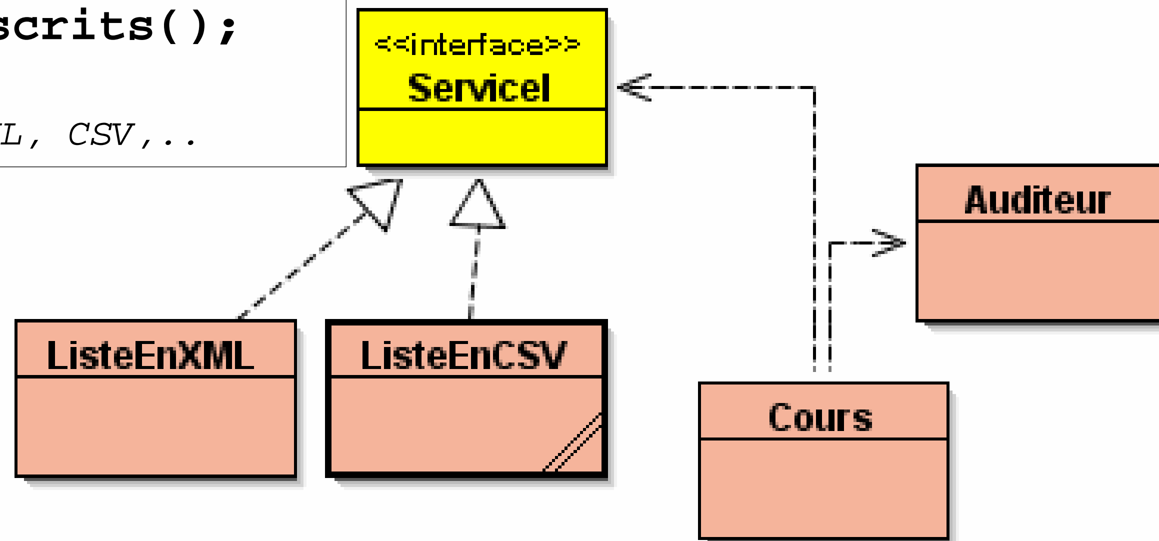
- **Scénario**
  - **Des auditeurs, un cours, une liste d'inscrits à ce cours**

  **Objets métiers:**
  - **Classe Auditeur**
  - **Classe Cours.**
    - **En attribut une liste des auditeurs inscrits …** *un classique*

- **Liste des inscrits fournie par le service de la scolarité**
  - **Texte, CSV, XML, JSON, BDD, … ???**
  - **Ne pas dépendre du format de la liste des inscrits**
    - **Un plus serait de ne pas dépendre de la machine où se trouve ce fichier**
    - **Ou bien de ne pas dépendre du format de fichier et de l'endroit**

# Un rappel ? Par une injection

```java
public interface ServiceI{
  List<Auditeur> lesInscrits();
}// les auditeurs inscrits
 // quelque soit le format XML, CSV,..
```




- **Du classique maintenant …**
  - **Le service retourne une liste d'auditeurs quelque soit le format choisi**
    - **Le couplage est faible, un réflexe …**
    - **n.b. Architecture apparentée patron fabrique…**

  - **Le service est ici un attribut de la classe Cours**

# ServiceI, Cours … Injection

```
public class Cours{

  private ServiceI service;  // en attribut


  public void setService(final ServiceI service){
    this.service = service;
    List<Auditeur> liste = service.lesInscrits();
    …
  }
```

-> Injection ici lors de l'appel du mutateur setService

*encore une autre forme d'injection de dépendance*

*(rappel déjà vus: injection d'interface, injection par le constructeur)*

# Variante de l'assembleur/conteneur,
## la configuration est maintenant décrite en XML

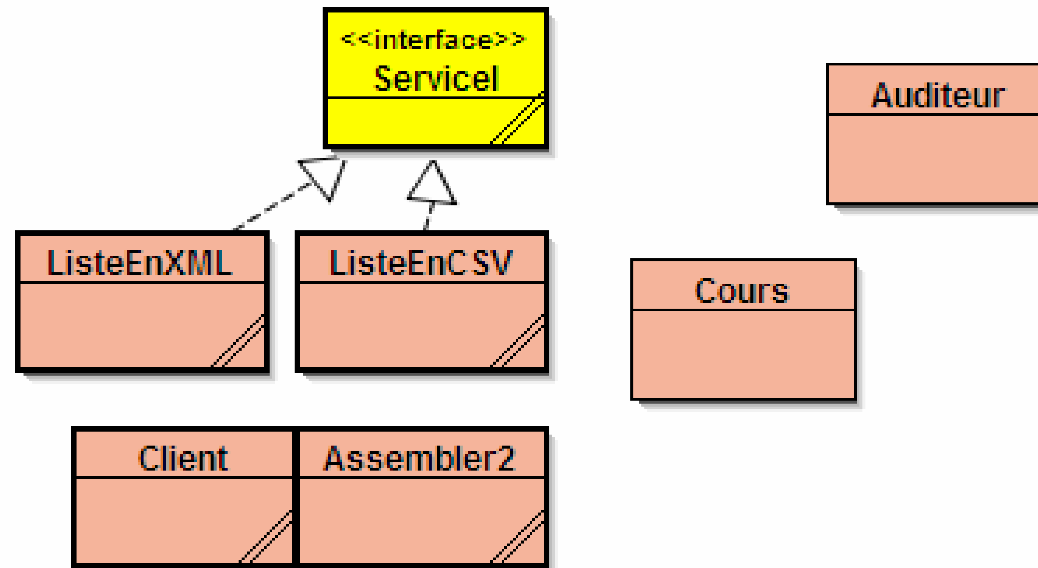
- Injection d'une dépendance par l'appel de setService

- **La configuration est sous la forme d'un fichier XML,**
  *lisible par tous, standard …*

```
// le fichier config.xml contient ici toutes les injections de l'application
<injections>
   <injection application="Cours" inject="ListeEnCSV" />
   <param inject="ListeEnCSV "   param_1=…  param_2=…  />

   <injection …. />
</injections>
```

- **Le conteneur lit le fichier XML et déclenche le mutateur setService**
  - **Réalisation:**
    - **La lecture du fichier XML en utilisant SAX**
    - **L'appel du mutateur setService par introspection**

# Architecture des classes




- **Assembler2 se charge**
  - **De l'analyse du fichier XML**
  - **De la création par introspection des instances**

**Et contient les instances créées**

# Assembler2 : lecture du fichier XML avec SAX

```
public class Assembler2{
   private Map<String,String> injections;  // gestion d'un cache
   private Map<String,String> methodes;

  public Assembler2() throws SAXException, IOException{
    this.injections = new HashMap<String,String>();
    this.methodes = new HashMap<String,String>();
    this.parse();
  }

 private void parse() throws SAXException, IOException{
    XMLReader saxReader = XMLReaderFactory.createXMLReader();
    saxReader.setContentHandler(new DefaultHandler(){
      public void startElement(String uri, String name, String qualif, Attributes at)
   throws SAXException{

      String application=null, inject=null, method=null;
      if(name.equals("injection")){
        for( int i = 0; i < at.getLength(); i++ ){
          if(at.getLocalName(i).equals("application"))
            application = at.getValue(i);
          if(at.getLocalName(i).equals("inject"))
            inject = at.getValue(i);
       }
        injections.put(application, inject);
        methodes.put(inject, method);
      }
    }
```

*Analyse XML* (for loop above)

*Création des instances* (the two put calls above)

# Assembler2 suite : introspection

```
private
 Object injection(String applicationName, String injectionName){
  private Object object=null;
  try{
// obtention de la classe Class/Application
    Class<?> appli = Class.forName(applicationName);
// obtention de la classe Class/A Injecter
    Class<?> inject = Class.forName(injectionName);

// recherche du mutateur depuis la classe Application
   Method m = appli.getDeclaredMethod("setService", ServiceI.class);

// création d'une instance de la classe Application,
    object = appli.newInstance();


// appel du mutateur avec une instance de la classe à injecter
   m.invoke(object, inject.newInstance());

    }catch(Exception e){}
   return object;
  }
```

# Assembler2 suite : newInstance

```
public Object newInstance(Class<?> application){
     return newInstance(application.getName());
  }


  private Object newInstance(String applicationName){
     String injectionName = injections.get(applicationName);
     String methodName = methodes.get(injectionName);
     return injection(applicationName, injectionName);
  }
```

Côté Client:

```
   Cours c = (Cours)assembler.newInstance(Cours.class);
```

# Introspection, suite possible, laissée en exercice

1. **Parmi toutes les interfaces implémentées par la classe à injecter,**
   **Exemple : class ListeEnCSV implements …, ServiceI, …**

2. **Quelles sont celles dont le nom est associé à la déclaration d'un attribut de la classe Cours ?**

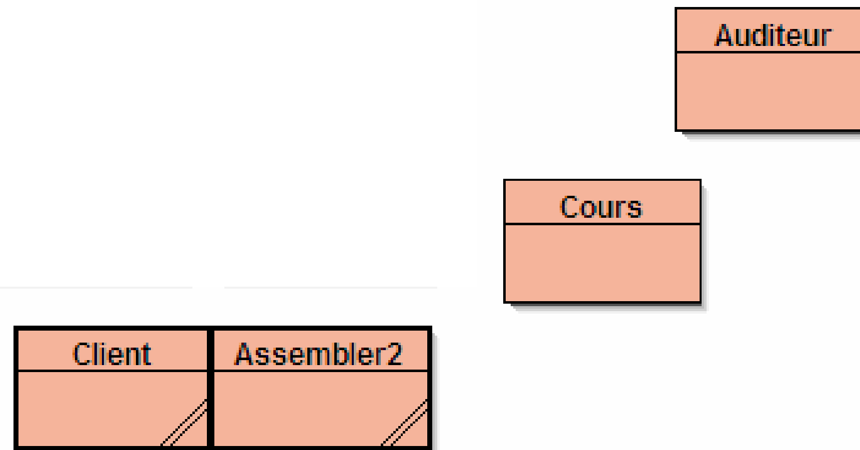
```
class Cours …
  …
  private ServiceI service;
```

3. **Existe-t-il le mutateur avec la signature habituelle ?**

```
 class Cours …
   …
    public void setService( ServiceI service){ …
```

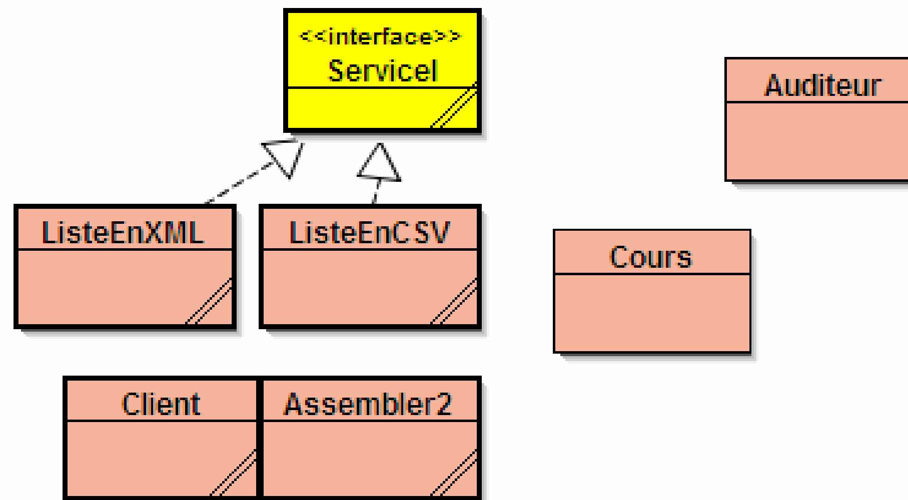
4. **Alors la création d'une instance de la classe Cours et l'appel du mutateur avec une instance de la classe à injecter deviennent possibles.**

# Point de vue: Le client s'adresse à l'assembleur (face A)




```
public class Client{

  public static void main(String[] args) throws Exception{

    // création d'une instance de l'assembleur/conteneur
    Assembler2 assembler = new Assembler2();

    // Obtention d'une nouvelle instance de la classe Cours,
    // configurée correctement
    Cours c = (Cours)assembler.newInstance(Cours.class);
   }
}
```

# Point de vue: Le client s'adresse à l'assembleur (face B)




```
public class Client{

  public static void main(String[] args) throws Exception{
    // Analyse du fichier XML, config.xml
    Assembler2 assembler = new Assembler2();

    // Par introspection : le mutateur est appelé
    // en fonction de la configuration
    Cours c = (Cours)assembler.newInstance(Cours.class);

   }
}
```

# <Injection> et généricité, autres syntaxes …

- **Injection et généricité ?**
  - **Pertinent ou non ?**

- **Injection et d'autres patrons**
  - **Patron  Factory Method**
  - **Fabrique par une méthode statique**
    - **Un classique des API Java**

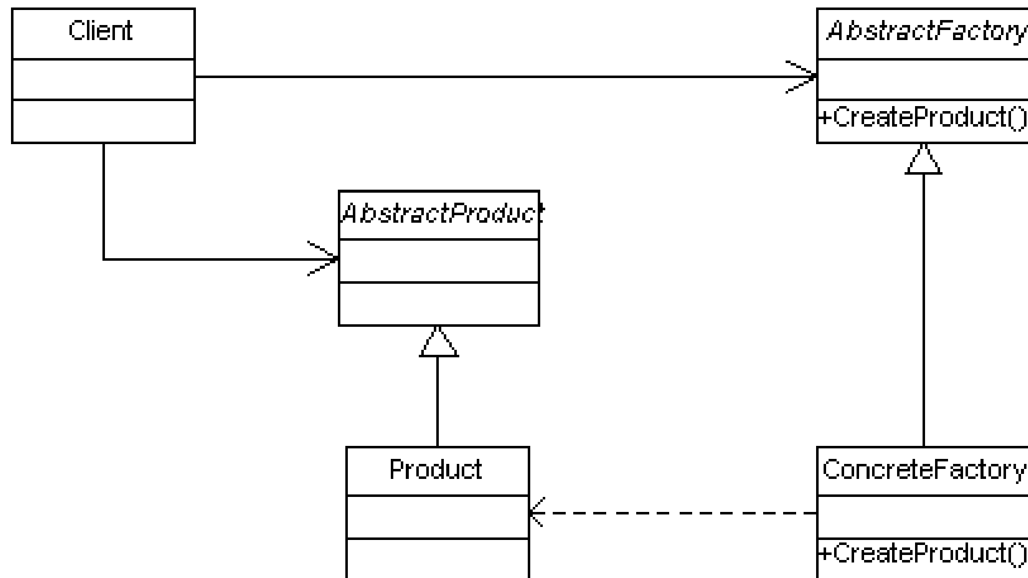
# Injection et généricité ?

```
public class CoursG<S extends ServiceI>{
  private S service;

  public void setService(S service){
    this.service = service;
    List<Auditeur> liste = service.lesInscrits();
  }

}

CoursG<ListeEnXML> c = new CoursG<ListeEnXML>();
c.setService(new ListeEnXML());
```

Discussion

Du typage uniquement, utile certes

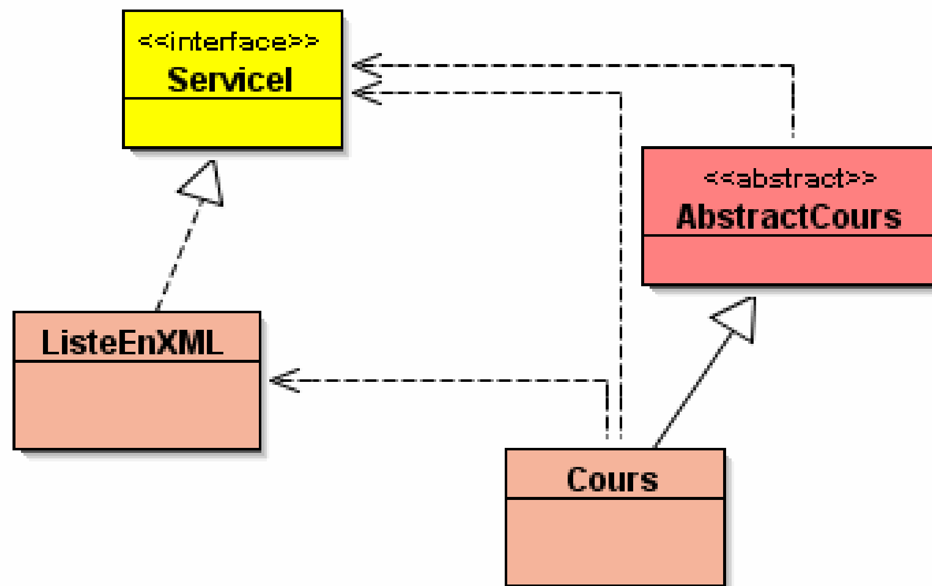
# Et le patron Factory Method ?




- **Définit une interface pour la création d'un objet, *mais* laisse aux sous classes le soin de décider quelle est la classe à instancier,**
  - comme si nous disposions d'un constructeur « dynamique »

- **Est-ce de l'injection de dépendance par héritage ?**
  - **Ou simplement de l'héritage …**

# En exemple, l'exemple




Auditeur

```
public abstract class AbstractCours {
  private ServiceI service;

  public AbstractCours(){
    this.service = creerService();
    List<Auditeur> l = service.lesInscrits();
  }

  public abstract ServiceI creerService();

}
```

```
public class Cours extends AbstractCours{

  public ServiceI creerService(){
    return new ListeEnXML();
  }
```

- **Constat : la classe Cours devient dépendante de ListeEnXML**
  - **Mais l'Injection de dépendance (constructeur, mutateur) paraît plus simple …**

# ServiceFactory, par une méthode de classe

```
public class ServiceFactory{
   private static ServiceI instance = new ListeEnXML(); //ou config…

   public static ServiceI getInstance(){ return instance;}

   public static void setInstance(ServiceI service){
     instance = service;
    }
}
```

```
Cours(){
  List<Auditeur> l = ServiceFactory.getInstance().lesInscrits();
```

**Revenons à l'injection classique constructeur ou mutateur**

# Injection Constructeur ou Mutateur ?

- **Constructeur**
  - **Naturel**

  - **Si beaucoup de paramètres ?**
  - **Un héritage de classes avec les constructeurs des super classes**
    - **avec eux aussi beaucoup de paramètres ?**
      - **Comment rester lisible ?**
  - **Absence de mutateur -> en lecture seule (*immutable*)**

- **Mutateur**
  - **Naturel**

  - **Affectation d'un attribut -> écriture**
  - **Oubli possible ?** *( ne dure pas… nullPointerException …)*
  - **Automatisation possible par un outil … cf. beans**

# Configuration ou API ?

- **Avec Configurations en XML,… ?**
  - **Langage pour non informaticien,** *soi-disant*
    - **Cependant quelles sont les personnes qui les utilisent ?**
      - **Serait-ce uniquement des informaticiens ?**
  - **Chaque assembleur/conteneur de configuration a sa DTD …**

- **A l'aide d'API Java … ?**
  - **Méthodes pour que cette configuration soit faite par programme**
  - **Naturel pour un programmeur Java**

- **Après discussion, en conclusion**
  - **Proposer les 2 !**

# La suite : Retour sur le Container ou Assembler

- **Une interrogation légitime**
  – **Il devrait bien exister des « conteneurs » tout prêts …**

1. **L'article de Martin Fowler, encore…**

2. **Un Conteneur minimal fait « Maison » en Novembre 2016**
   - Des exemples, une démonstration en direct

   - **Le conteneur Maison**
     - que nous nommerons **femtoContainer** *(femto pour rester humbles)*

     - **Effectue l'injection de dépendance selon un fichier de configuration**
       – **Injection par mutateur**
       – **Séparation effective de la configuration de l'utilisation**

     - **Contient les instances créées**
       – **Accès aux instances via le conteneur**

# Des conteneurs tout prêts

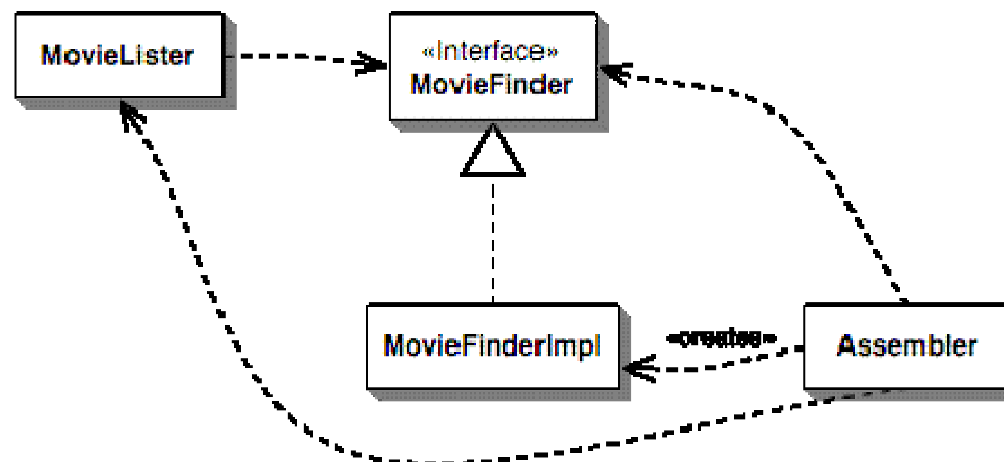
- **Spring**
  - **Injection par mutateur**
- **picoContainer**
  - **Injection par constructeur**
- **XWork**
- **HiveMind**
- **Zend**
- **…**
- https://de.wikipedia.org/wiki/Liste_von_Dependency_Injection_Frameworks

- **Par annotations …**
  - **Google Guice, EJB3**
  - **@Inject**
  - **Annotation qu'est-ce ?**

# L'exemple de Martin Fowler, l'article référent

- **Dont la lecture semble indispensable**
  - **http://www.martinfowler.com/articles/injection.html**
  - **Inversion of Control Containers and the Dependency Injection pattern**

- **Thème :**
  - **Obtenir la liste des films d'un certain réalisateur …**
    - **À la recherche de films dans une liste sous plusieurs formats**
      - **Texte, CSV, XML, BDD …**



# MovieLister, MovieFinder, Assembler

```
public class Movies {
        private MovieFinder finder;

        public Movies() {
        }

        public void setFinder(MovieFinder finder) {
                this.finder = finder;
        }

        public List<Movie> moviesDirectedBy(director: String) {
        List<Movie> allMovies = finder.findAll();
        for (m in allMovies) {
                if (!movie.getDirector().equals(director))
                                allMovies.remove(m);
                }
        return allMovies;
   }
}

public class SemiColonDelimitedMovieFinder implements MovieFinder {
        private String filename;

        public void setFilename(String filename) {
                this.filename = filename;
        }
}
```

**Injection ici du déjà vu !**

**Cours/Movies**

- **Assembler as Container, page suivante**

# Spring : XML et container

```
<beans>
    <bean id="MovieLister" class="spring.MovieLister">
        <property name="finder">
            <ref local="MovieFinder"/>
        </property>
    </bean>
    <bean id="MovieFinder" class="spring.ColonMovieFinder">
        <property name="filename">
            <value>movies1.txt</value>
        </property>
    </bean>
</beans>
```

```
public void testWithSpring() throws Exception {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("spring.xml");
    MovieLister lister = (MovieLister) ctx.getBean("MovieLister");
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
```

- **Une configuration XML,**
  - **Séparation de la configuration de l'utilisation,**
- **Suivie d'un usage du conteneur Spring**
- **Lisible ?**
  - **Property comme attribut, setFileName, et setFinder sont appelés**
  - **Injection par mutateur**

# femtoContainer Cnam/NFP121…Nov. 2016

- **Un framework « Maison »** *femto comme modeste*

- **Obtention d'instance (de bean)**
  - **bean : constructeur sans paramètre + mutateur**

- **Injection par appel des mutateurs**

- **Configuration à l'aide d'un fichier de *properties***

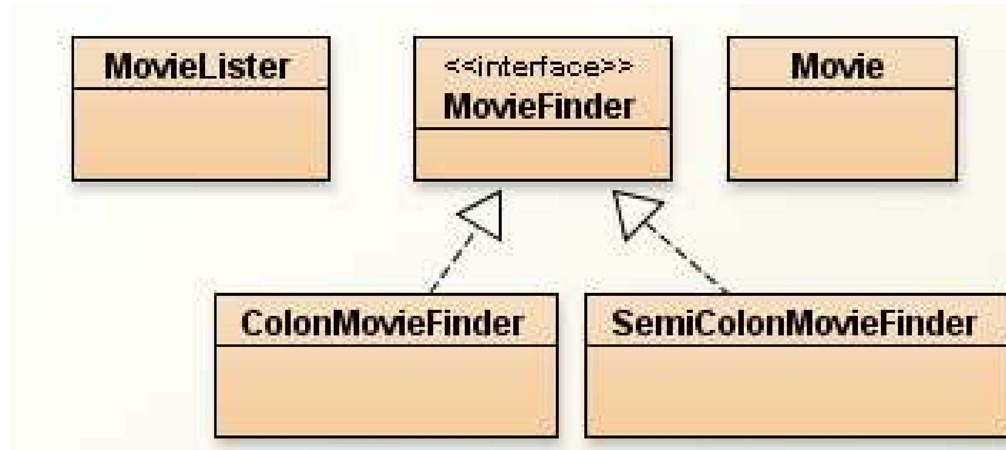
- ***Utilisation en syntaxe apparentée Spring***
  - ***Bien plus modeste…que Spring … alors ce sera femtoContainer***

# Utilisation, cf. article de M. Fowler

```
public void testWithSpring() throws Exception {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("spring.xml");
    MovieLister lister = (MovieLister) ctx.getBean("MovieLister");
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
```

```
// Avec femtoContainer
public void testWithOurContainer() throws Exception{
  ApplicationContext ctx = Factory.createApplicationContext();
  MovieLister lister = (MovieLister) ctx.getBean("MovieLister");
  List<Movie> movies = lister.moviesDirectedBy("Sergio Leone");
  assertEquals("Once Upon a Time in the West",movies.get(0).getTitle());
 }

//le nom du fichier est défini dans la fabrique
```

# Les classes en présence (M. Fowler)




```
public void testWithOurContainer() throws Exception{
  ApplicationContext ctx = Factory.createApplicationContext();
  MovieLister lister = (MovieLister) ctx.getBean("MovieLister");
  List<Movie> movies = lister.moviesDirectedBy("Sergio Leone");
  assertEquals("Once Upon a Time in the West",movies.get(0).getTitle());
}
```

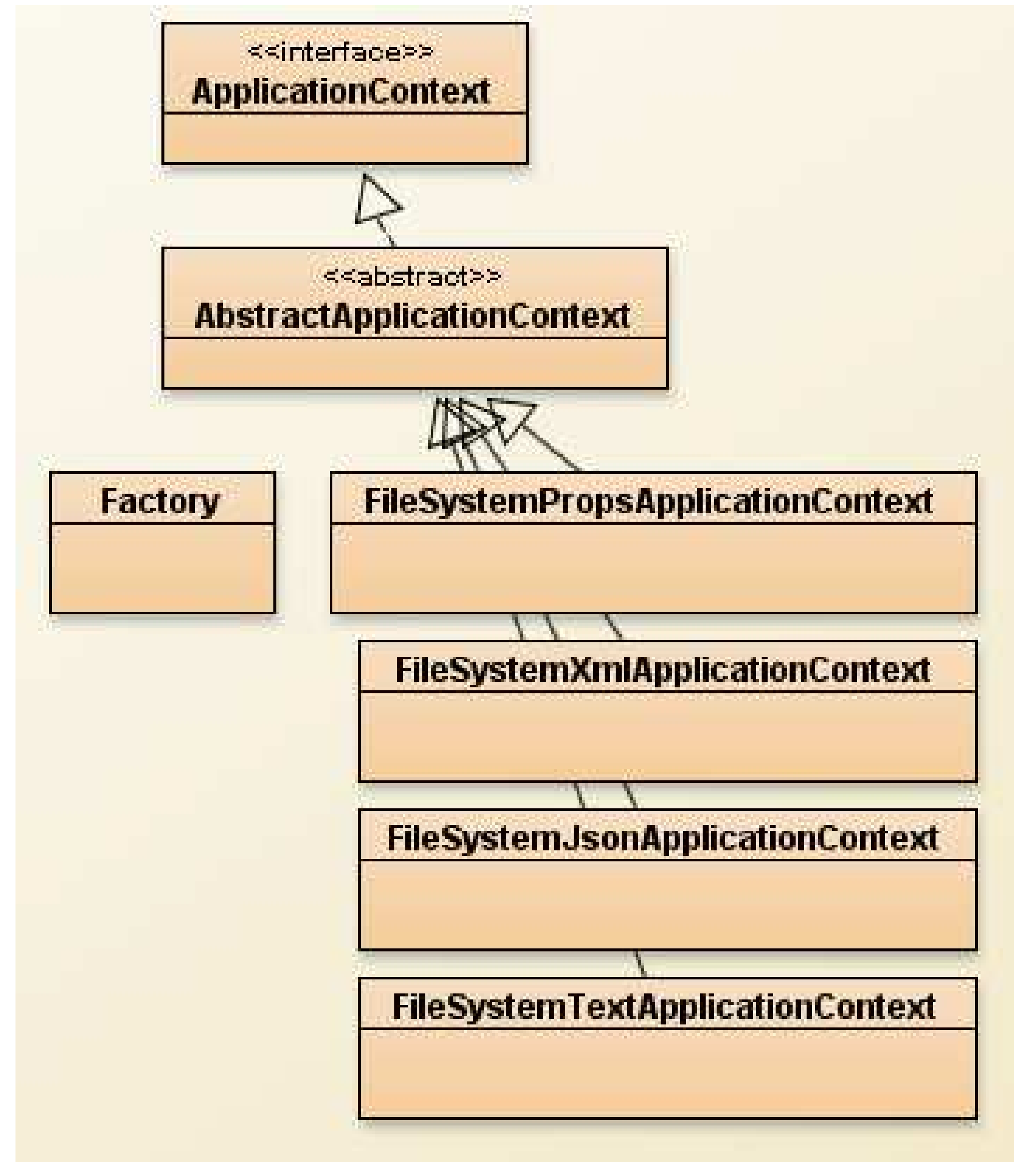
# Configuration Spring/femtoContainer

```
<beans>
    <bean id="MovieLister" class="spring.MovieLister">
        <property name="finder">
            <ref local="MovieFinder"/>
        </property>
    </bean>
    <bean id="MovieFinder" class="spring.ColonMovieFinder">
        <property name="filename">
            <value>movies1.txt</value>
        </property>
    </bean>
</beans>
```

```
# MovieLister est l'identifiant du bean
bean.id.1=MovieLister
# à quelle classe java ce bean est-il associé ?
MovieLister.class=spring.MovieLister
# Quelle propriété est à affecter, ici une seule
MovieLister.property.1=finder
# Le mutateur n'a qu'un paramètre
MovieLister.property.1.param.1=MovieFinder

bean.id.2=MovieFinder
MovieFinder.class=spring.ColonMovieFinder
MovieFinder.property.1=filename
MovieFinder.property.1.param.1=movies1.txt
```

# Un framework « Maison » pour un TP à la maison

- **Deux essentielles**
  - AbstractApplicationContext
  - Factory
- **Configuration**
  - Plusieurs formats



# AbstractApplicationContext

```java
public interface ApplicationContext extends Iterable<String>{

  /** Obtention d'une instance d'un bean géré par le conteneur.
   *  Il n'existe qu'une seule instance avec cet id, c'est un singleton.
   * @param id l'identifiant unique du bean
   * @return l'instance associée ou null si cet identifiant est inconnu
   */
  public Object getBean(String id);

  /** Obtention du type du bean à partir de son identifiant.
   * param id l'identifiant unique du bean
   * @return le type du bean ou null
   */
  public Class<?> getType(String id);

  /** Obtention d'un itérateur sur les beans déjà créés.
   * L'opération de retrait, méthode remove, est sans effet.
   * @return un itérateur des identifiants du conteneur
   */
  public java.util.Iterator<String> iterator();
}
```

- **Conteneur de Beans**
  - **Obtention d'un bean à partir de son identifiant**
  - **Obtention de la classe constatée + un itérateur sur la table**

# Factory pour la sélection du fichier de configuration

```
public class Factory{

public static ApplicationContext createApplicationContext(){
    // à remplacer par votre choix format et de fichier de configuration

    //String filename = "question1/config.props"; // le nom de votre fichier
    String filename = "question1/README.TXT"; // le nom de votre fichier
    InputStream inputStream = Factory.class.getClassLoader().getResourceAsStream(filename);
  return new FileSystemPropsApplicationContext(inputStream);

// Le choix du format
// return new FileSystemXmlApplicationContext(inputStream);
// return new FileSystemJsonApplicationContext(inputStream);
// return new FileSystemTexteApplicationContext(inputStream);

}
```

- **Choix du format du fichier de configuration**
  - **Serait-ce une forme d'injection de dépendances ?, légère discussion …**
  - **Ici un fichier de properties**
    - **Question1/README.TXT  par commodité, icône en haut à droite avec Bluej**

# femtoContainer properties un choix

- **Fichier de configuration, Properties clef=valeur**

```
# bean.id.N un numéro unique suite croissante +1
# MovieLister est l'identifiant du bean
bean.id.1=MovieLister

# à quelle classe java ce bean est-il associé ?
MovieLister.class=spring.MovieLister

# Quelle propriété est à affecter, ici une seule
MovieLister.property.1=finder

# Le mutateur n'a qu'un paramètre
MovieLister.property.1.param.1=MovieFinder
```

# 4 exemples avec femto

- **La classe Cours**
  - Configuration du service ListeEnXML ou ListeEnCSV
  - Utilisation ou affichage
- **Une IHM**
  - Configuration du layout et du listener
  - Utilisation en un click
- **Le patron Command**
  - Configuration des commandes concrètes dans le fichier de configuration
  - Utilisation des commandes
- **Le patron MVC**
  - Configuration depuis le fichier de configuration
    - Du modèle et de sa vue,
    - De la vue et du contrôleur
    - Du contrôleur et de son modèle
  - Utilisation en simulant un click

# Un exemple simple …

- **La classe Cours**
  - **Mutateur setService**
  - **Création de l'instance + appel du mutateur dans le conteneur**

```java
public class Cours{
  private ServiceI service;
  private List<Auditeur> liste;

  public void setService(ServiceI service){
    this.service = service;
    this.liste = service.lesInscrits();
  }

  public ServiceI getService(){
    return service;
  }
```

# La configuration et le test

**bean.id.22=cours**
**cours.class=question1.Cours**
**cours.property.1=service**
**cours.property.1.param.1=service**

**bean.id.23=service**
**service.class=question1.ListeEnCSV**

```
public void testFemtoCours() throws Exception{
  ApplicationContext ctx = Factory.createApplicationContext();
  Cours nfp121 = (Cours) ctx.getBean("cours");
  String name = nfp121.getService().getClass().getSimpleName();
  assertEquals("est-ce le bon service ?", "ListeEnCSV", name);
}
```

# Un exemple aussi simple ...

- **La classe IHM**
  - **3 mutateurs**
  - **Création de l'instance + appels des mutateurs**

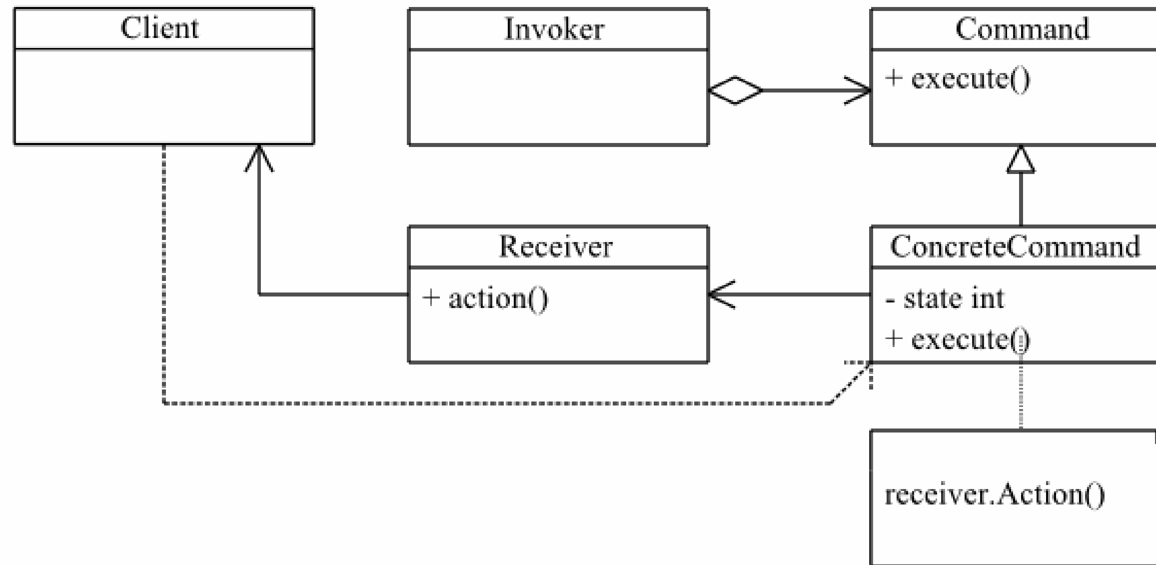
```
public class IHM{
  private JFrame jFrame;
  private JButton jButton;

  public IHM(){this.jButton = new JButton(" click ! ");}
  public void setJFrame(JFrame jFrame){this.jFrame = jFrame;}
  public void setLayout(LayoutManager layout){this.jFrame.setLayout(layout);}
  public void setListener(ActionListener listener){
    this.jButton.addActionListener(listener);
  }

  public void draw(){
    jFrame.add(jButton);
    jFrame.pack();
    jFrame.setVisible(true);
```

# La configuration et le test

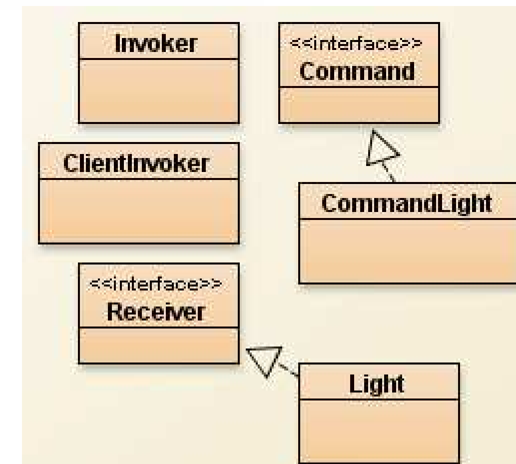
```
bean.id.15=ihm
ihm.class=question1.IHM
ihm.property.1=jFrame
ihm.property.1.param.1=jframe
ihm.property.2=layout
ihm.property.2.param.1=layout
ihm.property.3=listener
ihm.property.3.param.1=action

bean.id.16=layout
layout.class=java.awt.FlowLayout

bean.id.17=action
action.class=question1.ActionAuClick

bean.id.18=jframe
jframe.class=javax.swing.JFrame
jframe.property.1=title
jframe.property.1.param.1=femtoContainer
```

```
public void testIHM()throws Exception{
      ApplicationContext ctx = Factory.createApplicationContext();
      IHM ihm = (IHM) ctx.getBean("ihm");
      ihm.draw();}
```

# Un exemple : le pattern Command




- Configuration de l'*Invoker,*
  - de ses commandes concrètes
  - De ses effecteurs concrets



# Utilisation de femtoContainer + configuration

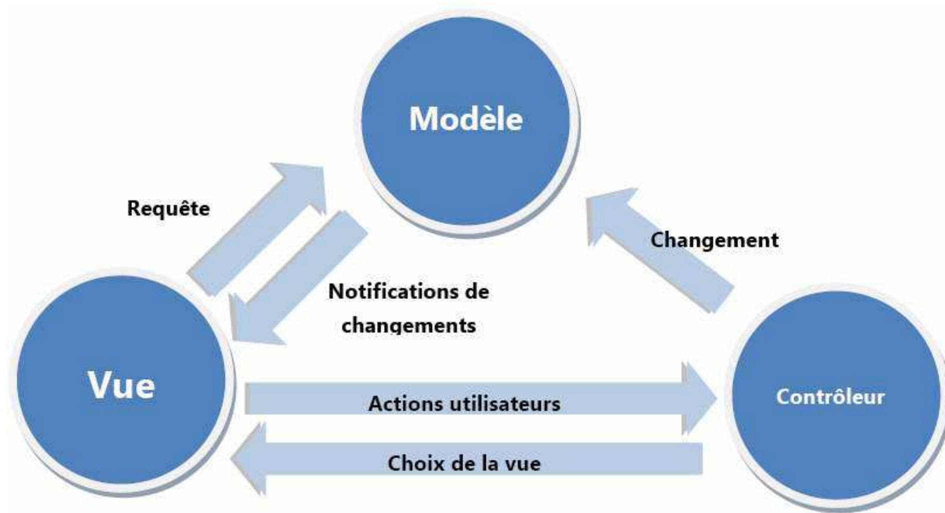
**bean.id.10=invoker**
**invoker.class=question1.Invoker**
**invoker.property.1=cmd**
**invoker.property.1.param.1=command**

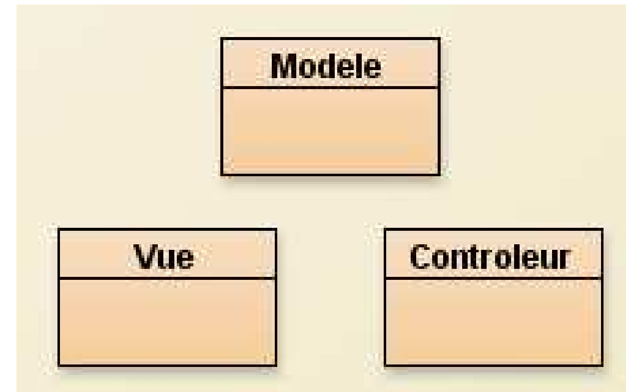
**bean.id.11=command**
**command.class=question1.CommandLight**
**command.property.1=light**
**command.property.1.param.1=receiver**

**bean.id.12=receiver**
**receiver.class=question1.Light**

```
public void testCommandPattern()throws Exception{
  ApplicationContext ctx = Factory.createApplicationContext();
  Invoker invoker = (Invoker) ctx.getBean("invoker");
  Receiver receiver = (Receiver) ctx.getBean("receiver");
  assertEquals("Off", receiver.getState());
  invoker.on();
  assertEquals("On", receiver.getState());
  }
```

# Un autre exemple MVC




- **Assemblage M V C par le fichier de configuration**
- **Obtention de la vue, puis un click**

# M V C

```java
public class Modele{
  private Vue vue;
  public void setVue(Vue v){
    this.vue = v;
  }
  public void change(){
    vue.update();
  }
}
```

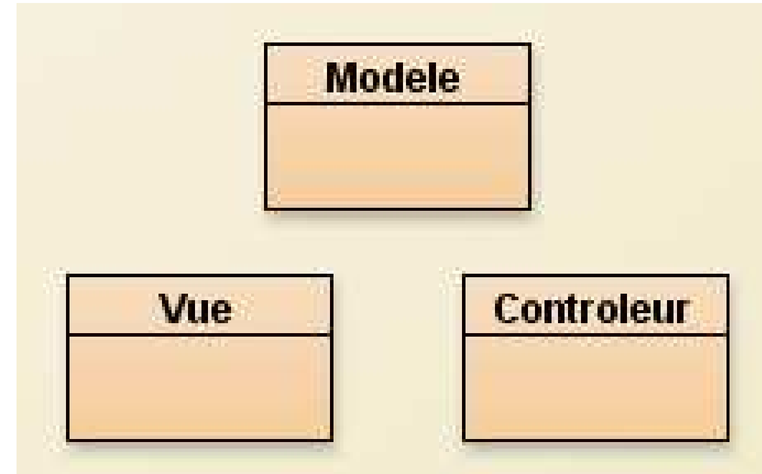
```java
public class Vue{
  private Controleur controleur;
  public int compte;

  public void update(){
    compte++;
  }
  public void click(){
    controleur.action();
  }
  public void setControleur(Controleur c){
    this.controleur = c;
  }
}
```

```java
public class Controleur{
  private Modele modele;
  public void setModele(Modele m){
    this.modele = m;
  }
  public void action(){
    modele.change();
  }
}
```

1: Vue.click() → Controleur.action()
2: Controleur.action() → Modele.change()
3: Modele.change() → Vue.update()

# Utilisation de femtoContainer + configuration

```
 bean.id.17=modele
modele.class=question1.Modele
modele.property.1=vue
modele.property.1.param.1=vue

bean.id.18=vue
vue.class=question1.Vue
vue.property.1=controleur
vue.property.1.param.1=controleur

bean.id.19=controleur
controleur.class=question1.Controleur
controleur.property.1=modele
controleur.property.1.param.1=modele
```




```
public void testMVC()throws Exception{
    ApplicationContext ctx = Factory.createApplicationContext();
    Vue vue = (Vue) ctx.getBean("vue");
     vue.click(); // un clic de l'utilisateur
     assertEquals(1, vue.compte);
     vue.click();
     assertEquals(2, vue.compte);
  }
```

# Démonstration

```
bean.id.17=modele
modele.class=question1.Modele
modele.property.1=vue
modele.property.1.param.1=vue

bean.id.18=vue
vue.class=question1.Vue
vue.property.1=controleur
vue.property.1.param.1=controleur

bean.id.19=controleur
controleur.class=question1.Controleur
controleur.property.1=modele
controleur.property.1.param.1=modele
```

*Le conteneur effectue en interne*

```
modele = new question1.Modele()
vue = new question1.Vue()
controleur=new question1.Controleur()

modele.setVue(vue)
vue.setControleur(controleur)
controleur.setModele(modele)
```

Un exemple de patron ? Lequel ?

Un exemple dans la salle ?

# Pause syntaxique : autres formes d'injection

- **Patron Template Method + Interface**

- **Syntaxe d'interface interne d'une classe**

- *Soit un moment de pause… enfin…*

# Autre forme d'injection

- **Un** *getter*
  - Redéfini dans une sous-classe, cf. le patron TemplateMethod

```
public abstract class AbstractA{
  private I i;

  public AbstractA(){
    this.i = getI();
  }
  protected abstract I getI();
}


public class A extends AbstractA{

  protected I getI(){
    return new B(); // ou depuis un fichier de configuration
  }
}
```

# Autre forme d'injection

- **Interface interne**
- **Exercice de style ?**

```
classe A{

      interface Inject{
         I get();
      }

   A(Inject inject){
   }
}
```

# Une autre : Une interface interne… 1/2

```
public class AAAAA {
  public interface Inject{
    I get();
  }

  private I i;
  public AAAAA(Inject inject){
    this.i = inject.get();
  }
…}
```

- à suivre : un assembleur/Conteneur

- Nb: AAAAA comme **Association Amicale des Amateurs d'Andouillette Authentique**
  Aucun lien avec notre propos

# Une interface interne… 2/2

- **L'assembleur/Conteneur**
  - **Un exemple possible**

```
AAAAA a = new AAAAA(new AAAAA.Inject(){
                                   public I get(){ return new B();}
                                 }
                                 );
```

- **Avantages/inconvénients**

- **Discussion**

  - Une autre syntaxe ? Non merci …

# Un résumé d'étape

- **Les 2 principales formes d'injection,** cf. Martin Fowler
  - **Injection de Constructeur**
  - **Injection de Mutateur**

**+**

  - **Injection d'interface**
  - Injection de redéfinition
    - Patron Template Method
  - Injection de Constructeur avec Interface interne

- **Avec une Configuration XML ou Texte**
  - **Le conteneur se charge de tout**

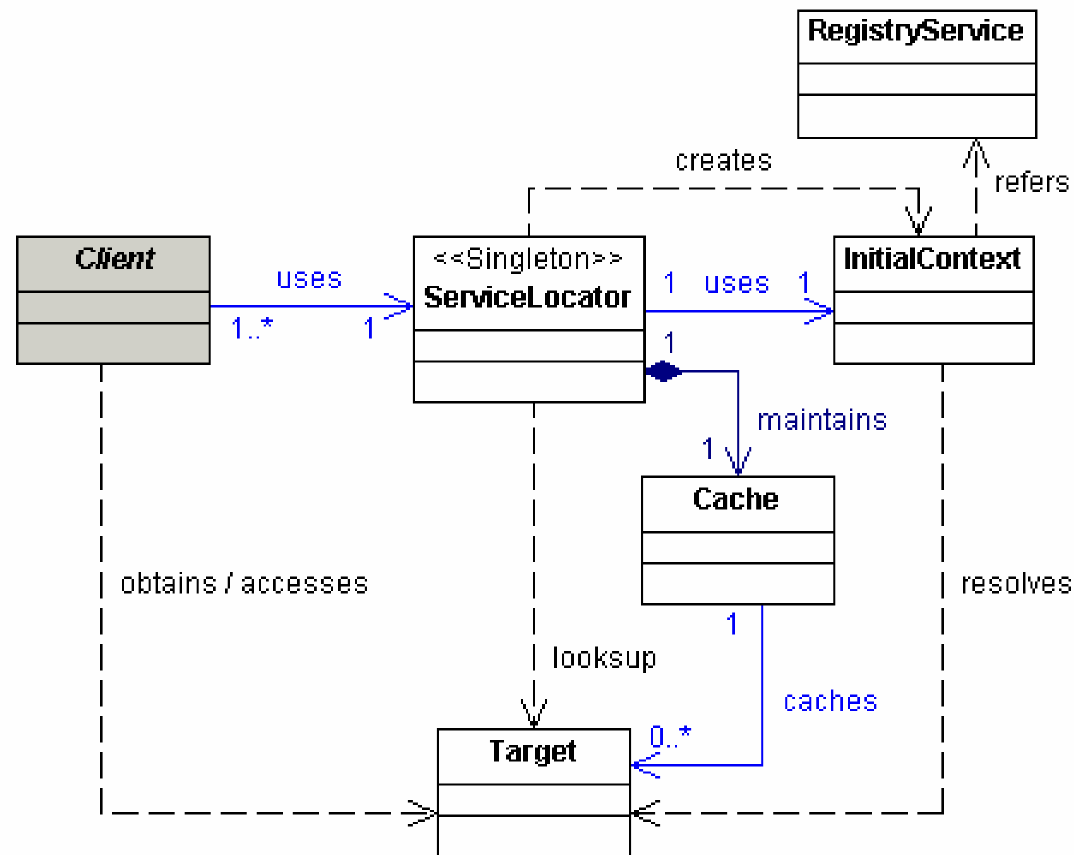
# Discussion (encore...)

- Inversion de contrôle et Injection de dépendance
- Inversion de contrôle sans Injection de dépendance
- Injection de dépendance sans Inversion de contrôle
- A lire
  - http://msdn.microsoft.com/en-us/library/cc304758.aspx
  - http://martinfowler.com/articles/injection.html

# Pause, suite : Autre forme de couplage faible

- **Le patron Service Locator**
  - **Localiser, sélectionner un service selon les besoins d'une application**
    - **accès de manière uniforme**
      - **Par exemple (DNS, LDAP, JNDI API)**
  - **Un Singleton en général …**
    - **Une table, un cache, un espace de nommage**

# Patron ServiceLocator




- **Source : [http://www.corej2eepatterns.com/Patterns2ndEd/ServiceLocator.htm](http://www.corej2eepatterns.com/Patterns2ndEd/ServiceLocator.htm)**

# Exemple : un cours, une liste d'auditeurs ...

- **Service Locator**
  - Nul ne sait où se trouve l'implémentation…

```
public class ConteneurServiceLocator{

  public static void assemble() throws Exception{
    Cours cours = new Cours();


    // obtention du service de nommage Cf. Singleton
    ServiceLocator locator = ServiceLocator.getInstance();

    // obtention du service du service ListeEnXml
    ServiceI service = (ServiceI)locator.lookup("ListeEnXML");

    // appel du mutateur
    cours.setService(service);
  }
}
```

# Service Locator, un exemple

- **À la recherche du .class**
  - **Usage de URLClassLoader(URL1, URL2, URL3, …)**
    - **Un classpath sur le web, …**
  - Les différentes URLs se trouvent dans un fichier de configuration XML …
    - Fichier nommé classpathWeb.xml

```
- <classpath>
    <url>http://jfod.cnam.fr/data/</url>
    <url>http://jfod.cnam.fr/data/archive.jar</url>
  </classpath>
```

  - **Un exemple de Conteneur : patron Singleton, lecture XML avec JDOM**

# ServiceLocator 1/2

```
public final class ServiceLocator{
  private Map<String,Class<?>> cache; // le cache
  private URLClassLoader classLoader; // chargeur de classes sur le web

  private static ServiceLocator instance;
  static{
    instance = new ServiceLocator(); // Cf. Singleton
  }

  public static ServiceLocator getInstance(){
    return instance;
  }

  public Object lookup(String name) throws Exception{
    Class<?> classe = cache.get(name);
    if(classe==null){ // il n'est pas dans le cache
      classe = Class.forName(name, true, classLoader); // par défaut
      cache.put(name,classe);
    }
    return classe.newInstance();
  }
```

# ServiceLocator 2/2, ici avec JDOM

```
private ServiceLocator() {
   this.cache = new HashMap<String,Class<?>>();
   try{
     SAXBuilder sxb = new SAXBuilder();
     Document document = sxb.build(new File("classpathWeb.xml"));
     Element racine = document.getRootElement();
     URL[] urls = new URL[racine.getChildren("url").size()];

     int index = 0;
     for(Object e : racine.getChildren("url")){ // JDOM
       urls[index] = new URL(((Element)e).getValue().toString());
       index++;
     }

     classLoader = URLClassLoader.newInstance(urls);
   }catch(Exception e){
   }
 }
```

# Démonstration ?

- **Injection de dépendance**

**Ou**

- **ServiceLocator ?**

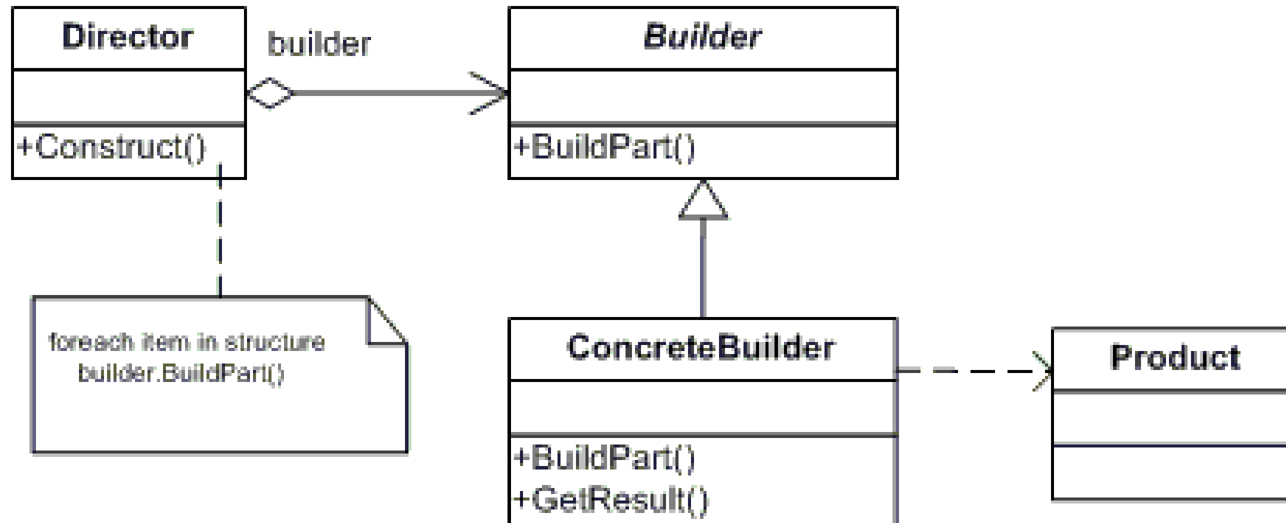
- **Et le patron Builder ? Un oubli  ? Non en annexe**

# Résumé

- **Injection de dépendance**
  - **Interfaces en java**
  - **Séparation de la configuration de l'utilisation**
  - **Usage d'un framework tout prêt**
    - **Spring**
  - **Le couplage faible devient implicite**

- **La suite en annexe**
  - **Le patron builder**
  - **Google guice**
  - **@Inject issue du paquetage javax.inject**

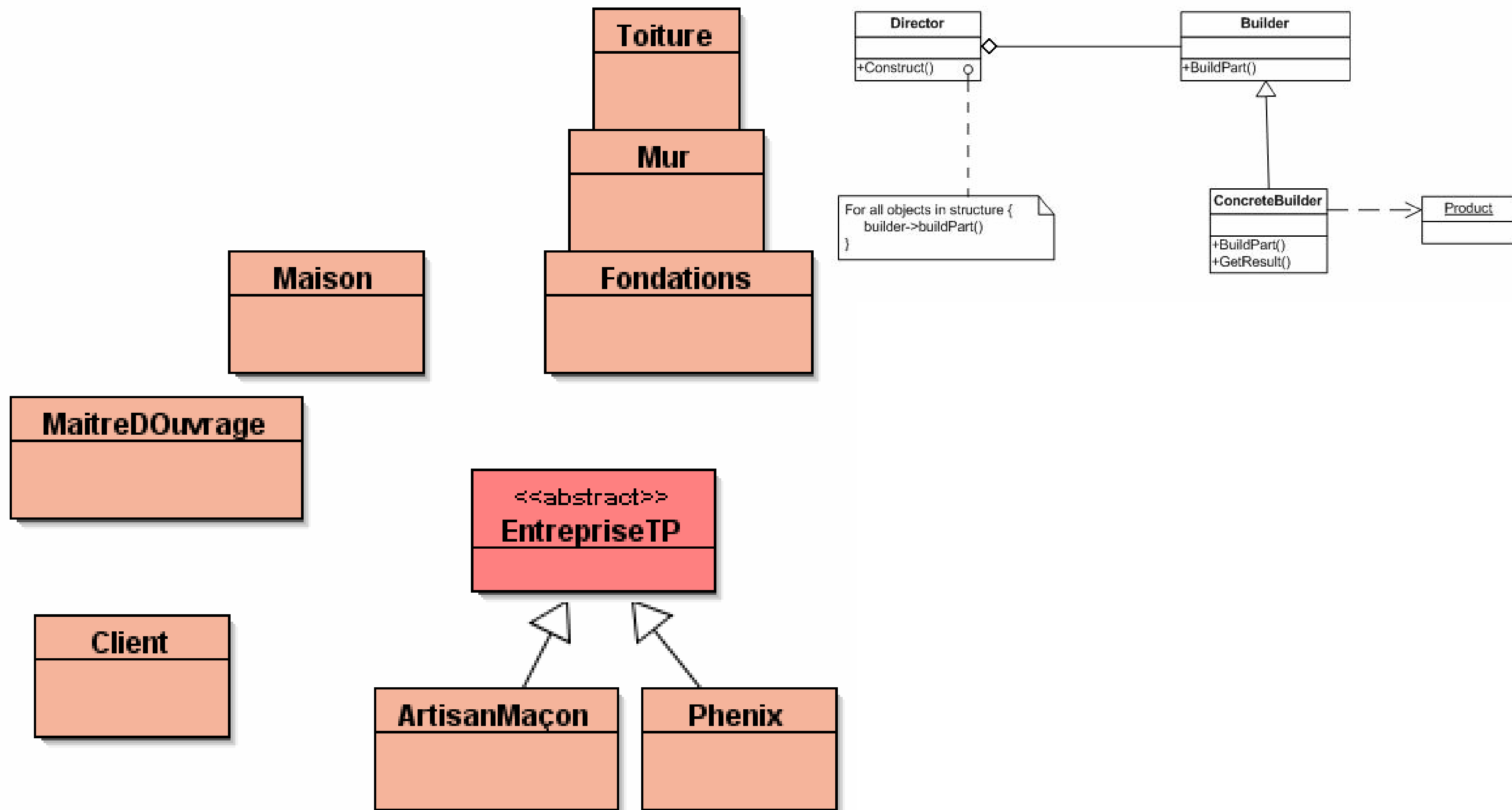
# Annexe : Le Patron Builder




- **Principe**
  - **Le Directeur utilise une Entreprise pour construire un Produit**
    - **Le directeur a la responsabilité de faire effectuer les travaux dans le bon ordre**
  - **L'entreprise est choisie « injectée » en fonction d'une configuration**

- **http://en.wikipedia.org/wiki/Builder_pattern#Java**
- **http://sourcemaking.com/design_patterns/builder/java/2**

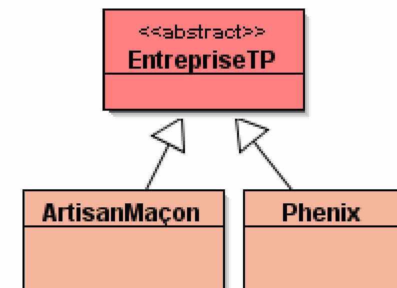
# En annexe : la construction d'une maison …




- **EntrepriseTP / Builder**
- **Maison / Product**
- **MaitreDOuvrage / Director**

# Entreprise de TP et un Artisan maçon

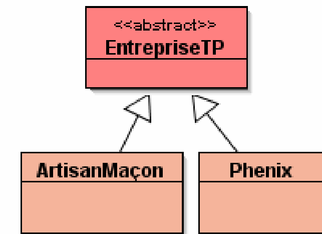
```
// src http://sourcemaking.com/design_patterns/builder/java/2

public abstract class EntrepriseTP{
  protected Maison maison;

  public Maison livraison(){
    return this.maison;
  }

  public Maison démarrerLeChantier(){
    return new Maison();
  }

  public abstract void construireLesFondations();
  public abstract void construireLesMurs();
  public abstract void construireLaToiture();
}
```




Classe abstraite pour la création d'une Maison,

**Démarrage du chantier et la livraison, après les constructions…**

# Un artisan maçon…construit

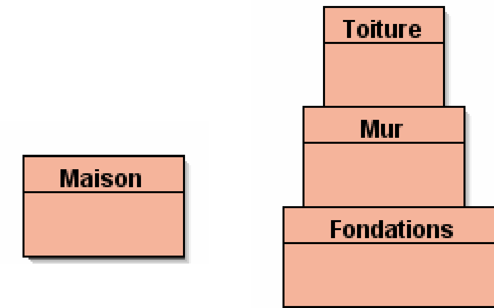



```
public class ArtisanMaçon extends EntrepriseTP{

  public void construireLesFondations(){
    //…

    Fondations fondations = new Fondations();
    maison.installerLesFondations(fondations);
  }

  public void construireLesMurs(){
     //…

     Mur[] murs = new Mur[4];
     maison.installerLesMurs(murs);
  }

  public void construireLaToiture(){
     //…

     Toiture toiture = new Toiture();
     maison.installerLaToiture(toiture);
  }
}
```

# Le MaitreDOuvrage choisit et organise …

```
public class MaitreDOuvrage{

  private EntrepriseTP entreprise;

  public void choisirUneEntreprise(EntrepriseTP entreprise){
    this.entreprise = entreprise;
  }

  // le maître d'ouvrage a la responsabilité de la bonne
  // séquence des travaux
  public void construireMaison(){
    this.entreprise.démarrerLeChantier();
    this.entreprise.construireLesFondations();
    this.entreprise.construireLesMurs();
    this.entreprise.construireLaToiture();
  }

  public Maison livraison(){
    return entreprise.livraison();
  }
}
```

# Maison et le Client




```
public class Maison{
  private Toiture    toiture;
  private Mur[]      murs;
  private Fondations fondations;

  public Maison(){
  }

  public void installerLesFondations(Fondations fondations){
    this.fondations = fondations;
  }

  public void installerLesMurs(Mur[] murs){
    this.murs = murs;
  }

  public void installerLaToiture(Toiture toiture){
    this.toiture = toiture;
  }
}
```

*// La maison a été construite par un artisan via le maître d'ouvrage,*
*// l'entrepriseTP choisie réalise(installe) les fondations, murs et toiture*

# Le Client

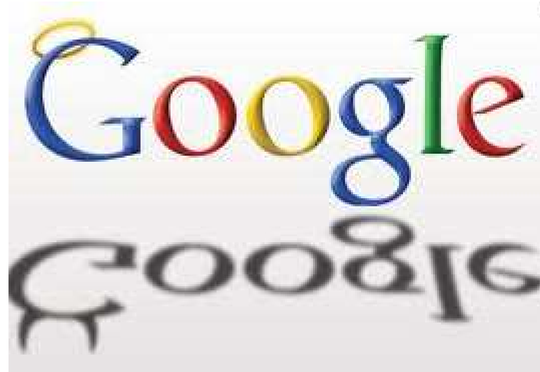
```
public static void main(String[] args){
   MaitreDOuvrage maitreDOuvrage = new MaitreDOuvrage();

// en fonction d'une configuration, choix de l'entreprise
   EntrepriseTP maitreDOeuvre = new ArtisanMaçon();
// ou bien maitreDOeuvre = new Phenix();
// en fonction d'un fichier de configuration



   maitreDOuvrage.choisirUneEntreprise(maitreDOeuvre);
   // sous-traitance par une entreprise de TP
   maitreDOuvrage.construireMaison();
   // démarrer le chantier,
   // construire les fondements, les murs et la toiture

   Maison maison = maitreDOuvrage.livraison();

 }
```

# Annexes Google Juice ++

- **Google juice**

- **Cf. JSR 330, EJB @Inject**

# Google est partout, avec nous ?



- **Injection avec google guice**

# Google Guice

- http://code.google.com/p/google-guice/
- http://google-guice.googlecode.com/files/Guice-Google-IO-2009.pdf
- http://google-guice.googlecode.com/svn/trunk/javadoc/packages.html

## Java on Guice

*Guice* (pronounced "juice") is an ultra-lightweight, next-generation dependency injection container for Java 5 and later.

- **Une mise en œuvre des exemples avec bluej, archives nécessaires**
  - +libs/**guice-3.0-no_aop.jar** (sans aop, compatible android, 471Ko)
  - +libs/**javax.inject.jar** (extrait de guice-3.0.jar)
  - Ou bien en ligne de commande

# Préambule : Annotations qu'es aquò* ?

- **Les annotations sont parmi nous**

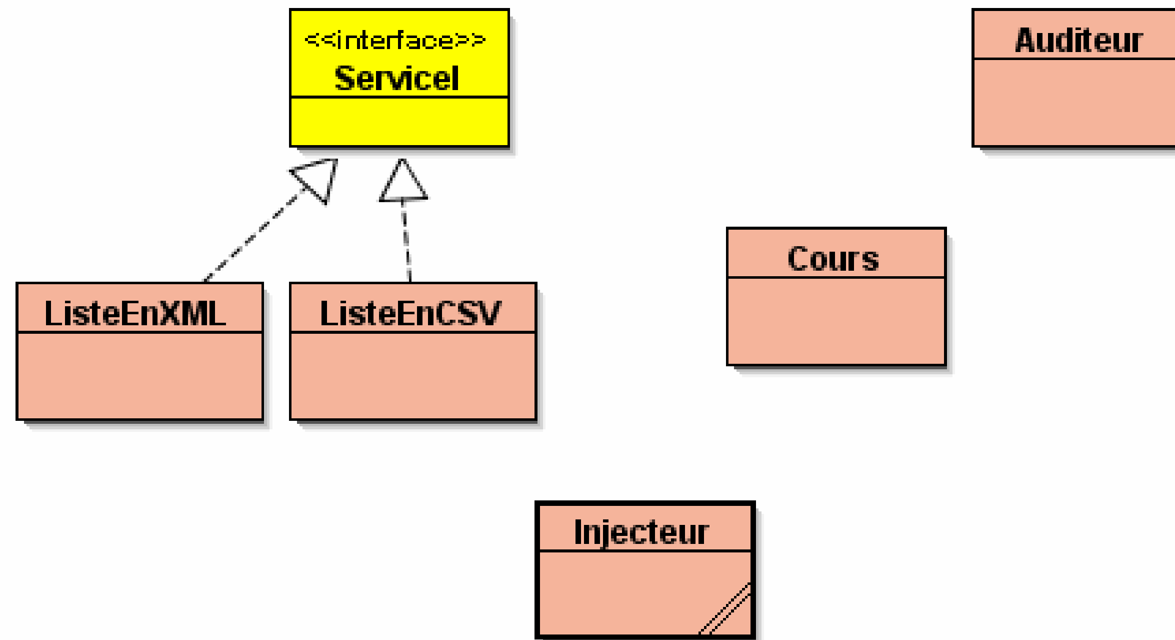
- **@Override**
  - **Destinée au compilateur**

- **@Runtime**
  - **Utilisée pendant l'exécution, et présente dans le .class**

- **@Discussion**
  - **Définie par l'utilisateur**
    - **À la compilation et/ou à l'exécution**

** « Qu'est-ce que c'est » en occitan*

# Notre exemple revu avec Google Juice



- **@Inject**
- **import com.google.inject.Inject;**

# @Inject, par un Mutateur, un constructeur …

```
public class Cours{

@Inject
  public void setService(ServiceI service){
    this.service = service;

  }
```

**@Inject** c'est simple concis … *que des avantages ?*

# Google Guice

- **Absence de fichier XML**

**alors**

- **La configuration est en java**
  - **AbstractModule par la redéfinition de la méthode configure**

# Assembleur, Injecteur, dans le code

- **import com.google.inject.Guice;**
- **import com.google.inject.Injector;**
- **import com.google.inject.AbstractModule;**

```
public class Injecteur{

public static void main(String[] args){
    Injector injector = Guice.createInjector(new Configuration());
    Cours c = injector.getInstance(Cours.class);
    …
  }

  private static class  Configuration extends AbstractModule{
    protected void configure(){

   //
      bind(ServiceI.class).to(ListeEnCSV.class);
    }
  }
```

# Mutateur, attribut, idem …

```
@Inject public Cours(ServiceI service){
    this.service = service;

}
```

**Ou**

```
private @Inject ServiceI service;
```

# Démonstration

- @Inject
- Configuration/utilisation
- À la recherche du couplage faible …

# Mais

- Simple mais …

  – Ici l'attribut `intitulé` a été configuré de l'extérieur…

```
public class Cours{

  @Inject private final String intitulé = null;

}
```

  – Par cette ligne dans la classe Injecteur

```
  bind(String.class).toInstance("NFP121");
```

**Toujours lisible, un outil ?**

# Annotation pour être plus lisible…enfin presque

- **Dans la classe Cours nous avons :**

```
@Inject Cours(@Service ServiceI service){
    this.service = service;
}
```

- **Dans l'Injecteur**
  - **bind(ServiceI.class).annotatedWith(Service.class).to(ListeEnCSV.class);**

- **L'annotation @Service décrite comme suit** (fichier Service.java)
    - Une annotation d'annotation

```
@Retention(RetentionPolicy.RUNTIME)
@Target( {ElementType.PARAMETER} )
@BindingAnnotation
public @interface Service{
}
```

# Ultra light weight framework, Les paquetages

| | |
|---|---|
| com.google.inject | *Google Guice* (pronounced "juice") is an ultra-lightweight dependency injection framework. |
| com.google.inject.assistedinject | Extension for combining factory interfaces with injection; this extension requires `guice-assistedinject-3.0.jar`. |
| com.google.inject.binder | Interfaces which make up `Binder`'s expression language. |
| com.google.inject.grapher | |
| com.google.inject.grapher.graphviz | |
| com.google.inject.jndi | JNDI integration; this extension requires `guice-jndi-3.0.jar`. |
| com.google.inject.matcher | Used for matching things. |
| com.google.inject.multibindings | Extension for binding multiple instances in a collection; this extension requires `guice-multibindings-3.0.jar`. |
| com.google.inject.name | Support for binding to string-based names. |
| com.google.inject.persist | Guice Persist: a lightweight persistence library for Guice; this extension requires `guice-persist-3.0.jar`. |
| com.google.inject.persist.finder | Dynamic Finder API for Guice Persist. |
| com.google.inject.persist.jpa | guice-persist's Java Persistence API (JPA) support. |
| com.google.inject.servlet | Servlet API scopes, bindings and registration; this extension requires `guice-servlet-3.0.jar`. |
| com.google.inject.spi | Guice service provider interface |
| com.google.inject.spring | Spring integration; this extension requires `guice-spring-3.0.jar`. |
| com.google.inject.throwingproviders | Extension for injecting objects that may throw at provision time; this extension requires `guice-throwingproviders-3.0.jar`. |
| com.google.inject.tools.jmx | JMX integration; this extension requires `guice-jmx-3.0.jar`. |
| com.google.inject.util | Helper methods for working with Guice. |

- **Ultra light weight** *heureusement* **…**

# Guice3.0 D'autres annotations

- **D'autres annotations existent et ne sont pas abordées ici**
- http://code.google.com/docreader/#p=google-guice&s=google-guice&t=UsersGuide

- **@ImplementedBy**
- **@providedBy**
- **@Singleton**
- **@Provides**
- **@CheckedProvides**
- **@RequestScoped**
- **@SessionScoped**

  - **Vers une meilleure productivité au détriment de la compréhension ?**
    - **Mais avons besoin de comprendre pour être plus productif ?**
    - **Usage d'un framework deviendrait-il obligatoire ?**

  - **Chaque API proposerait-elle ses propres annotations ?**
    - **Serions nous envahis par les annotations ?**

# Google Web Toolkit et Google-gin

- google-gin
  - GIN (GWT INjection) is Guice for Google Web Toolkit client-side code
    - Du java qui engendre du javascript pour les clients web

- Google Web Toolkit
  - http://code.google.com/intl/fr/webtoolkit/

# Paquetage javax.inject

- http://docs.oracle.com/javaee/6/api/javax/inject/package-summary.html
- http://docs.oracle.com/javaee/6/api/javax/inject/Inject.html

## Package javax.inject Description

This package specifies a means for obtaining objects in such a way as to maximize reusability, testability and maintainability compared to traditional approaches such as constructors, factories, and service locators (e.g., JNDI). This process, known as *dependency injection*, is beneficial to most nontrivial applications.

Examples:

```
public class Car {
  // Injectable constructor
  @Inject public Car(Engine engine) { ... }

  // Injectable field
  @Inject private Provider<Seat> seatProvider;

  // Injectable package-private method
  @Inject void install(Windshield windshield, Trunk trunk) { ... }
}
```

# Conclusion

- **A la recherche du couplage faible**
  - **Devient implicite avec l'usage de l'injection de dépendance**
  - **Séparer la configuration de l'implémentation**
    - **oriente naturellement vers de l'injection de dépendance donc un couplage faible**
  - **Configuration XML ou par une API ?**
    - **Une DTD par outil, par framework**
      - **Framework ancien ?, sans les annotations …**
    - **Une API**
      - **Plutôt simple à utiliser (Google Guice (mérite le détour)),**
        - » mais ne serait-ce pas un nouveau langage à base d'annotations,
        - » ou l'ajout d'annotation par le programmeur, ne nuit-il pas à la lisibilité …

# Autres Annexes

- **À lire**
  - [http://martinfowler.com/articles/injection.html](http://martinfowler.com/articles/injection.html)
  - [http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection](http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection)

# Injection et tests unitaires

- **http://code.google.com/p/jukito/**

- **Avec google guice + junit4**
- **http://code.google.com/p/guiceberry/**

- **http://code.google.com/p/atunit/**
- http://code.google.com/p/mockito/

- **http://code.google.com/p/guiceyfruit/**

- GuiceBerry, there is also AtUnit and GuiceyFruit

# HiveMind

## Welcome to HiveMind

**HiveMind** is an services and configuration microkernel. Its features are also referred to as Inversion of Control (IoC) Container or Lightweight Container. The adoption of HiveMind in an application ensures the use of certain design principles which improve encapsulation, modularization, testability and reusability.

- **Services**: HiveMind services are *POJOs* (Plain Old Java Objects) that can be easily accessed and combined. Each service ideally defines a Java interface it implements (this is not mandatory). HiveMind takes care of the life cycle of services. It instantiates and finalizes services and configures each service just as necessary. HiveMind lets services collaborate with each other via dependency injection, so that the service code itself is released from the task of looking up dependencies.
- **Configuration**: HiveMind allows you to provide complex configuration data to your services in a format *you* define. HiveMind will integrate the contributions of such data from multiple modules and convert it all into data objects for you. HiveMind configurations allow for powerful, data-driven solutions which combine seemlessly with the service architecture.

- http://hivemind.apache.org/

# EJB, JSR330 @Inject

```
public class Cart {
  @Inject OrderSystem ordering;
  @Inject CustomerNotification notifier;

  public void checkout(){
    ordering.placeOrder();
    notifier.sendNotification();
  }
}

public class OrderSystem { public void placeOrder(){ } }
```

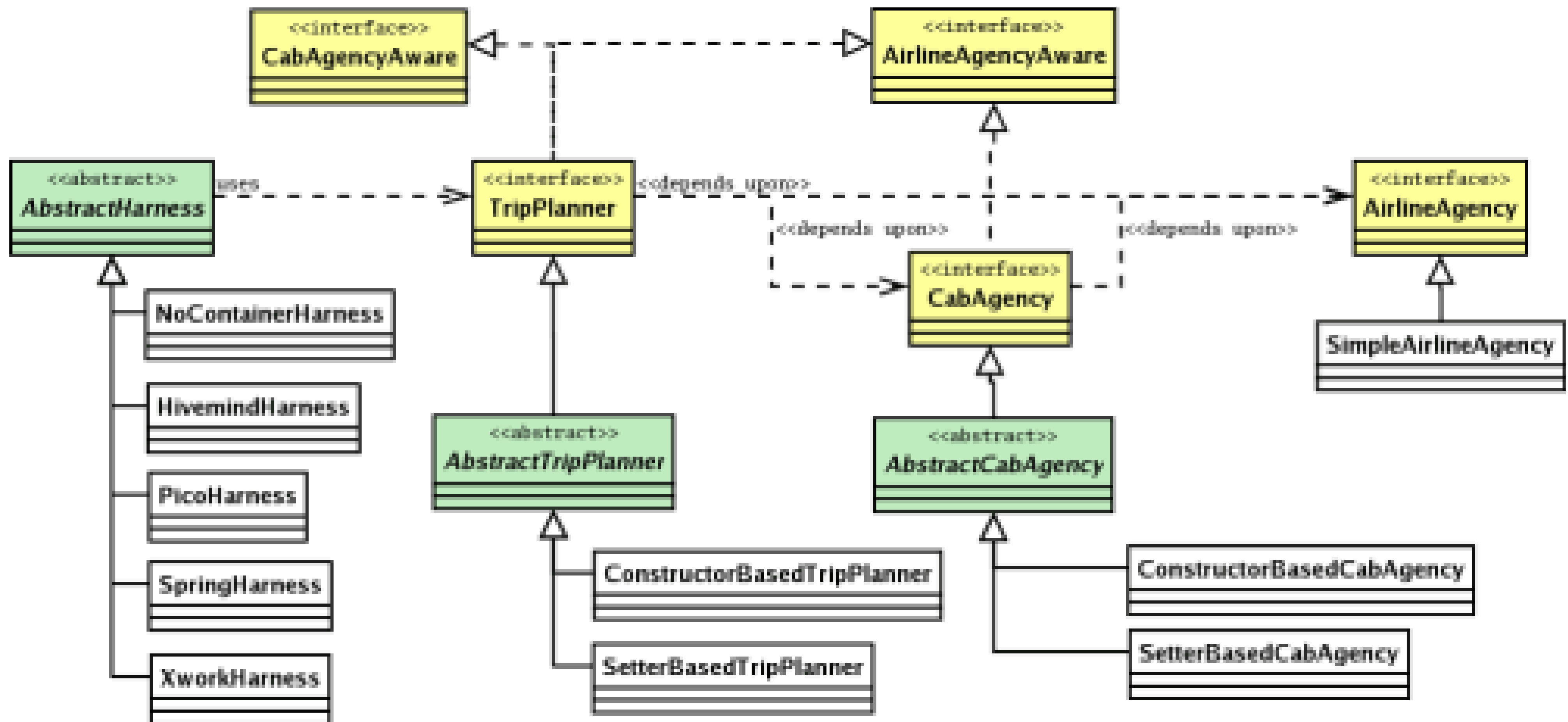
- http://www.adam-bien.com/roller/abien/entry/simplest_possible_pojo_injection_example
- http://kenai.com/projects/javaee-patterns/sources/hg/show/EJBAndCDI/src/java/com/abien/ejbandcdi/control?rev=238

# L'exemple de http://www.theserverside.com

- **Préambule lire cet article**
  - http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection

- **Application : planification d'un voyage**
  - Choisir la destination et la date d'arrivée souhaitée et l'heure
  - Appeler l'agence de voyages et obtenir une réservation de vol.
  - Appeler l'agence de taxis, demander un taxi pour un vol particulier en fonction de la résidence
    - l'agence de taxi pourrait avoir besoin de communiquer avec l'agence de voyages afin d'obtenir l'horaire de départ du vol, l'aéroport, et ainsi calculer la distance entre votre résidence et l'aéroport et calculer le temps approprié pour avoir à l'avion depuis votre résidence
  - Prendre le taxi avec les tickets, être sur votre chemin

# Un exemple au complet




- **Hairness as assembler**