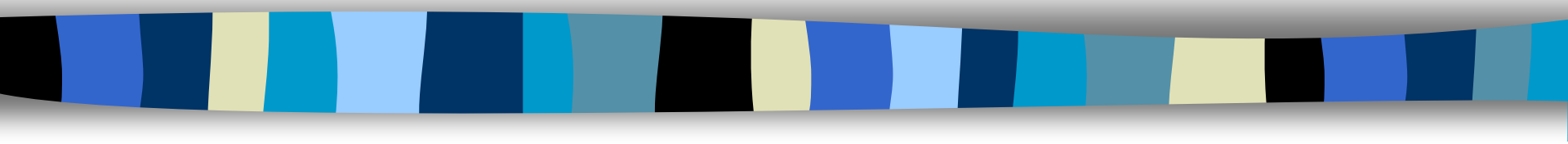
Introduction à la métaprogrammation



M1 Informatique – MIF16 – 2013-2014
Lionel Médini
Université Claude Bernard Lyon 1

Plan

- Introduction
- Métaprogrammation par annotations
- Programmation Orientée Aspects
- Conclusion

Introduction

- La réutilisation comme principe de conception
  - S’appuyer sur du code existant
  - Mettre en œuvre des patterns éprouvés
  - « Sortir » les tâches répétitives du corps d’un programme
    - Abstraction
    - Isolation
    - Standardisation

Introduction

- Programmation par configuration
  - Abstraction
    - Identifier les données finales (paramètres du programme)
    - Identifier les traitements répétitifs liés à ces données

Introduction

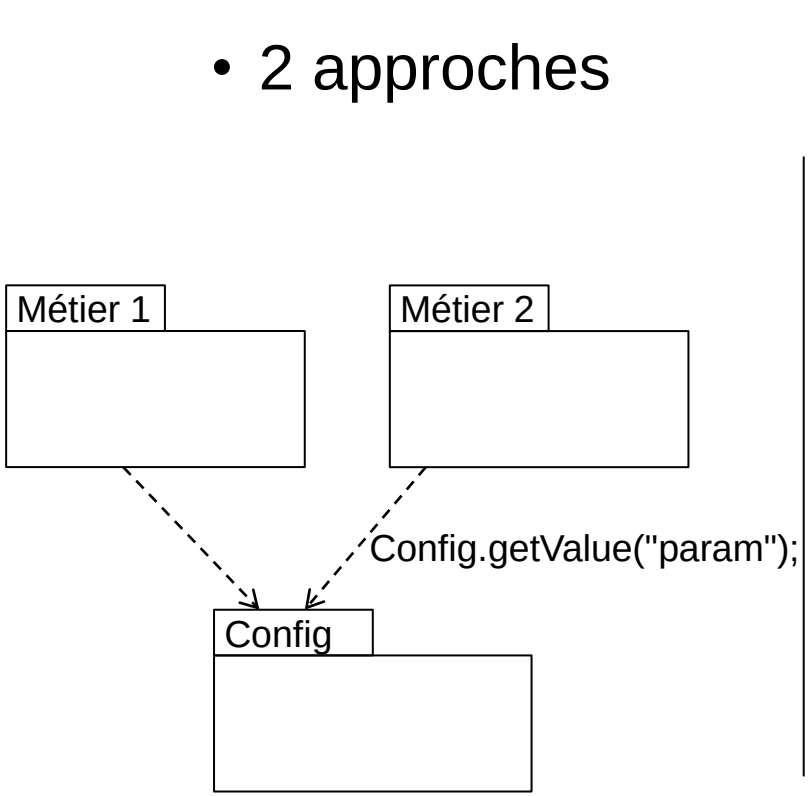
- Programmation par configuration
  - Isolation
    - « Séparer » ces données et ce code
    - Exemple
      - Placer données et code dans un package spécifique
      - Les rendre disponibles à l'aide de méthodes statiques

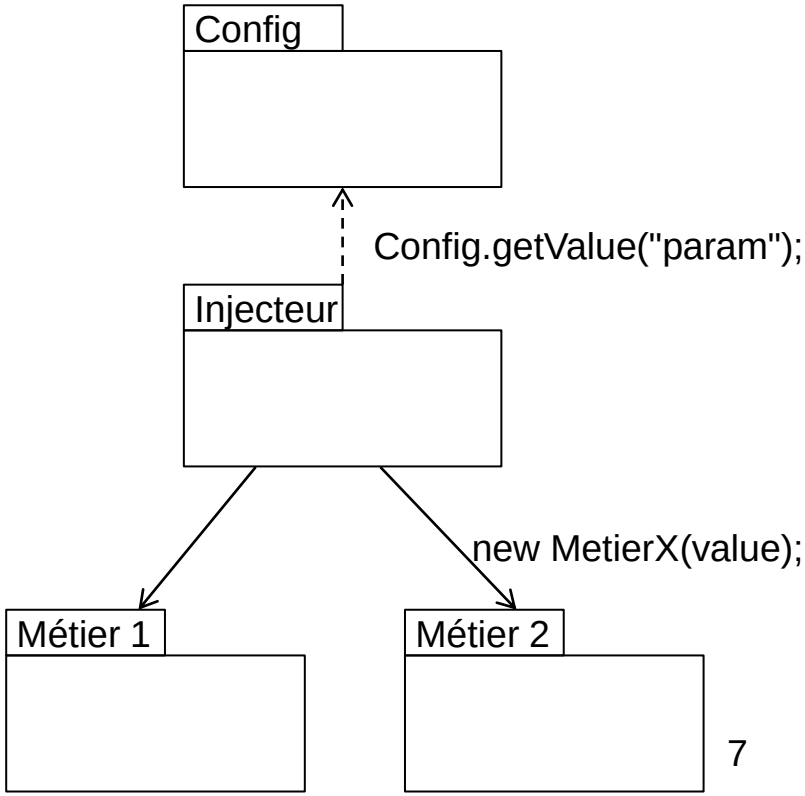
Introduction

- Programmation par configuration
  - Standardisation
    - Homogénéiser la collecte des données
      - Fichier de configuration
      - Arguments passés au programme
      - Lecture sur l'entrée standard
    - Créer une classe dont la responsabilité sera de les mettre à disposition
      - En fonction des traitements où elles sont utilisées
      - De manière homogène

Introduction

- Programmation par configuration
  - Standardisation
    - 2 approches





Introduction

- Programmation par configuration
  - Fonctionnement courant
    - La classe de configuration lit et stocke des valeurs dans un fichier XML
    - Ces valeurs sont injectées dans un programme qui instancie des classes
    - Ces classes sont ensuite liées aux paramètres présents dans le fichier
  - Citez un exemple…

Introduction

- Programmation par configuration
  - Remarques
    - Dans le deuxième cas, le mécanisme d'injection *inverse la dépendance* entre le métier et la configuration
    - Il s'appuie sur une description explicite
      - des dépendances entre le métier et la configuration
      - des valeurs de configuration

    ➔ Programmation déclarative

Introduction

- Programmation par configuration
  - Remarques
    - Dans les 2 cas, le package Configuration est réutilisable
      - Mais est-il bien nécessaire ?

    ➔ Autre approche
      - Stocker les valeurs de configuration au plus près du métier
      - Nécessite que les valeurs de configuration ne s'appliquent qu'à des éléments particuliers du métier

      ➔ « Annoter » les éléments avec leur configuration

Introduction

- Approche plus générale
  - Abstraire et isoler les traitements répétitifs
  - Annoter le code métier de manière
    - déclarative
    - à lier ce code aux traitements
  - Définir un mécanisme qui réalise la « jonction » entre code annoté et traitements

Objectifs de ce cours

- Présenter différents outils d'aide à la conception et au développement
  - Les annotations en Java
    - Automatisation de traitements sur le code (métaprogrammation)
  - La Programmation Orientée Aspects
    - Séparation des préoccupations

Plan

- Introduction
- Métaprogrammation par annotations
- Programmation Orientée Aspects
- Conclusion

Métaprogrammation par annotations

- Position du problème
  - Générer automatiquement certains éléments récurrents des programmes
    - Documentation : Javadoc, XDoclet
    - Fonctions transverses : persistance des données, transactions, sécurité (AOP)
    - Gestion du cycle de vie d'objets complexes (EJB)
    - Rajouter des métadonnées pour la compilation
    - Traitements spécifiques
    - Injection de données issues de fichiers de configuration
    - …

Métaprogrammation par annotations

- Avantages de la génération automatique
  - Gain de temps
  - Pas d'erreur de programmation
  - Pas de maintenance de « side files » qui doivent être synchonisés avec le code source (l'information est directement stockée dans les fichiers sources)
- Inconvénients
  - Traitements non modifiables
  - Nécessite de comprendre / accepter le comportement automatique

Métaprogrammation par annotations

- Principe
  - Programmation déclarative
    - le concepteur décrit ce qu'il veut obtenir
    - Un outil interprète ces annotations et le réalise
  - Pour cela, il faut pouvoir intervenir
    - Sur les fichiers sources
    - Sur les fichiers compilés (.class)
    - Lors de l'exécution (réflection)

Métaprogrammation par annotations

- Fonctionnement
  - Des annotations dans le code
    - Avec éventuellement des paramètres
  - Des processeurs qui réalisent des tâches spécifiques à ces tags
    - Chaque *AnnotationProcessor* comporte une méthode *process()* qui lui indique quoi faire de l'élément annoté
  - Un mécanisme de liaison entre code et annotations

Métaprogrammation par annotations

- Exemples
  - Les annotations standard
    - Permettent d'interagir avec le compilateur
    - Intégrées au langage Java depuis Java 2 SE 5.0

      @Deprecated

      @Override

      @SuppressWarnings (String_ou_tab warnings)
  - Les annotations spécifiques aux applications
    - Intégrées aux JARs

      @Inject

Métaprogrammation par annotations

- Utilisation dans le code
  - Nom de l'annotation préfixé par '@'
  - Devant l'élément concerné
    - Types d'éléments affectés : package, class interface, enum, annotation, constructeur, méthode, paramètre, champ d'une classe, variable locale
    - Plusieurs annotations différentes sont autorisées sur un même élément (mais jamais deux fois la même)
    - Avec des paramètres « simples » (types primitifs, String, Class, Annotation, enum…)

Métaprogrammation par annotations

- Utilisation dans le code
  - Exemple avec une annotation simple (« marqueur »)
    ```
    @MonAnnotation
    public class MaClasse { /* ... */ }
    ```
  - Exemple avec des attributs
    ```
    @MonAnnotation (att1 = 12)
    public void MaMethode() { /* ... */ }
    ```
  - Exemple de syntaxe simplifiée
    ```
    @MonAnnotation (“toto”)
    public void MaMethode() { /* ... */ }
    ```

Métaprogrammation par annotations

- Déclaration d'une annotation
  - Mot-clé « interface » préfixé par '@'

```
public @interface MonAnnotation {
    ...
}
```

Métaprogrammation par annotations

- Attributs d'une annotation
  - Méthodes avec type de retour et nom

```
public @interface MonAnnotation {
/** Message décrivant la tâche à
effectuer.*/
   int att1();
   String value();
}
```

Métaprogrammation par annotations

- Remarques
  - toute annotation hérite implicitement de java.lang.annotation.Annotation (Java SE 5)
  - la portée des attributs d'une annotation est implicite et toujours public
  - la méthode standard String value(); permet d'omettre le nom de l'attribut à l'appel de l'annotation (syntaxe simplifiée)

Métaprogrammation par annotations

- Les méta-annotations standard
  - Qualifient les annotations non standard
    - @Documented
    - @Inherit
    - @Retention (duree_de_vie)
    - @Retention (RetentionPolicy.SOURCE)
    - @Retention (RetentionPolicy.CLASS)
    - @Retention (RetentionPolicy.RUNTIME)

Métaprogrammation par annotations

- Les méta-annotations standard
  - @Target (type_d_element_ou_tab)
    - @Target (ElementType.ANNOTATION_TYPE)
    - @Target (ElementType.CONSTRUCTOR)
    - @Target (ElementType.FIELD)
    - @Target (ElementType.LOCAL_VARIABLE)
    - @Target (ElementType.METHOD)
    - @Target (ElementType.PACKAGE)
    - @Target (ElementType.PARAMETER)
    - @Target (ElementType.TYPE)

Métaprogrammation par annotations

- Traitement des annotations
  - Annotations standard
    - Un processeur fourni avec la JVM
  - Annotations non standard
    - Nécessitent un outil ad hoc :
      - Inclus dans le framework de l'application
        - » Jboss, PicoContainer...
      - Externe
        - » Javadoc, XDoclet, APT...

Métaprogrammation par annotations

- Traitement des annotations non standard
  - Réalisation d'un AnnotationProcessor
    - dérive de la classe abstraite javax.annotations.AnnotationProcessor
    - Instancié par une AnnotationProcessorFactory
    - Relié aux annotations par la méthode getProcessorFor() de la factory

Métaprogrammation par annotations

- Traitement des annotations non standard
  - Réalisation d'un AnnotationProcessor
    - Implémente la méthode
      public boolean process(Set<? extends TypeElement> arg0, RoundEnvironment arg1);
      - Utilisation de l'API Reflection pour déterminer les éléments annotés
      - Détaille les traitements à effectuer pour un "round" donné (programmation impérative)
      - Renvoie un booléen indiquant si un traitement a été effectué

Métaprogrammation par annotations

- Utilisation des annotations dans le code
  - Utilisation de l'introspection dans le code d'un processeur
    - Permet d'accéder aux annotations dont la rétention est RUNTIME
    - Depuis Java SE 5
    - Interface java.lang.AnnotatedElement (implémentée par AccessibleObject, Class, Constructor, Field, Method et Package)
    - Méthodes getAnnotation(), getAnnotations(), getDeclaredAnnotations(), isAnnotationPresent()

Métaprogrammation par annotations

- Traitement des annotations non standard
  - L'outil Annotation Processing Tool (APT)
    - Relie chacune des annotations figurant dans le code à l'AnnotationProcessor la concernant
    - Syntaxe
      - Proche de celle de javac (ligne de commande, options)
    - Évolution
      - Package externe dans Java SE 5
      - Comportement embarqué par le compilateur depuis Java SE 6

Métaprogrammation par annotations

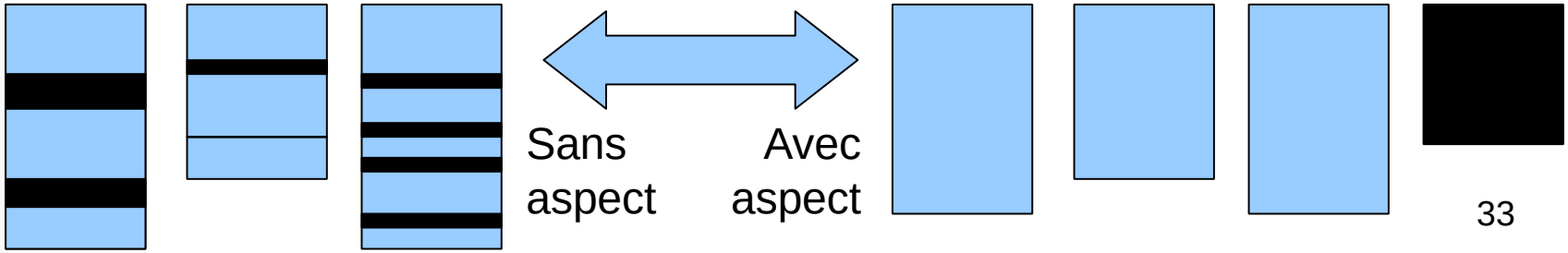
- Traitement des annotations non standard
  - L'outil Annotation Processing Tool (APT)
    1. Détermine les annotations présentes dans le code source
    2. Recherche les AnnotationProcessorFactories que vous avez écrites
       1. Demande aux factories les annotations qu'elles traitent
       2. Demande aux factories qui traitent des annotations présentes dans le code de fournir un AnnotationProcessor
    3. Exécute les AnnotationProcessor correspondants
    4. Si ces processeurs ont généré de nouveaux fichiers sources, APT reboucle jusqu'à ce qu'il n'y ait plus de nouveaux fichiers générés

Plan

- Introduction
- Métaprogrammation par annotations
- Programmation Orientée Aspects
- Conclusion

Position du problème

- Gérer les fonctionnalités transverses d'une application
  (accès aux données, transactions sécurité)
  - En regroupant (vs. dispersant) le code lié à ces fonctionnalités
  - En les séparant de la logique métier
  - Avantages : productivité, maintenance, réutilisabilité



Limites du paradigme objet

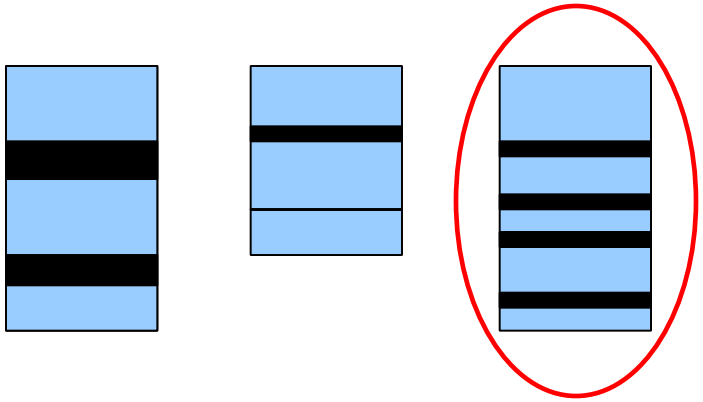
- Données encapsulées dans les classes
  - Traitements similaires sur des données différentes
- Différentes considérations (aspects) de l'application représentés au même niveau d'abstraction
- « Pollution » du modèle métier par les fonctionnalités transverses
- Même avec des patrons appropriés (façade, observateur), il reste beaucoup de code dans les classes

Le paradigme aspect

- Pas contradictoire, mais complémentaire au paradigme objet
- En POA, une application comporte des classes et des aspects
  - Une classe est un élément du domaine à modéliser
  - Un aspect est une fonctionnalité à mettre en oeuvre dans l'application
- Chaque aspect permet d'obtenir une « vision » différente de l'application
  - Métier, données, sécurité...

Concepts de base / glossaire

- Tangled code
  - Code embrouillé, code spaghetti



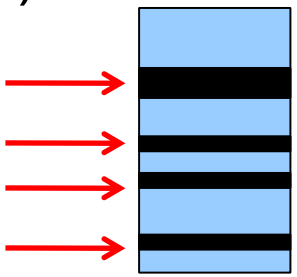
Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Crosscutting concerns
  - Aspects de la programmation qui concernent plusieurs classes, et qui donc transcendent le modèle objet (synchronisation, logging, persistance…)
  - Mélange, au sein d'un même programme, de sous-programmes distincts couvrant des aspects techniques séparés (Wikipédia)



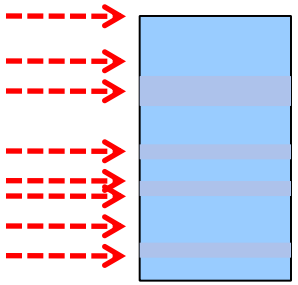
Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Crosscut (coupe)
  - Ensemble ou sous-ensemble des points de jonction liés à un aspect dans une application
  - Permet de définir la structure transversale d'un aspect



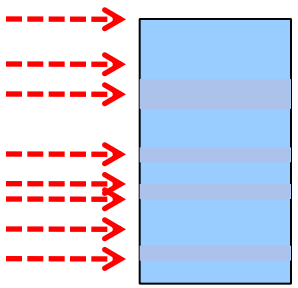
Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Joinpoint (point de jonction)
  - Endroit du code où il est autorisé d'ajouter un aspect (avant, autour de, à la place ou après l'appel d'une fonction)
  - Dans 80% des cas, liés aux méthodes
  - Parfois liés aux classes, interfaces, attributs, exceptions...



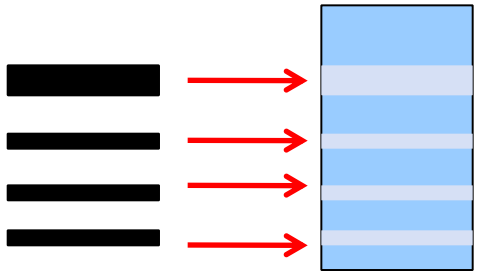
Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.html

Concepts de base / glossaire

- Pointcut (point de coupe)
  - Endroit du code métier où est inséré le code de l'aspect
  - Décrit un ensemble de joinpoints
  - Types
    - Méthodes : appel et exécution
    - Constructeurs : appel et exécution
    - Champs : getters et setters
    - Exception handlers : exécution
    - Initialisation statique d'une classe



Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Pointcut (point de coupe)
  - Syntaxe
    - Fait appel à des primitives du langage spécifiques
      - Type de pointcut
      - Instance de la classe appelante
      - Instance de la classe appelée
      - Arguments de la méthode
      - Handler d'exception
      - ...

Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Pointcut (point de coupe)
  - Exemples de syntaxe (AspectJ)
    - Simple

```
pointcut greeting() : execution(void myexample.Hello.sayHello ());
```

    - Plus complexe

```
pointcut sysout(Object message) : args(message) &&
                                  call (public void print*(*))
                                  && target(java.io.PrintStream)
                                  && !within(BankAspectSysout) ;
```

Sources :

http://blogexpertease.alti.com/index.php?2010/03/26/158-la-programmation-orientee-aspect-aop-avec-aspectj

http://www.tomjanofsky.com/aspectj_tutorial.html

Concepts de base / glossaire

- Advice (greffon)
  - Fragment de code qui sera activé à un certain point de coupe du programme
  - Doit correspondre à une définition de point de coupe
  - 4 types
    - before
    - after returning
    - around
    - after throwing

Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Advice (greffon)
  - Exemples (AspectJ)
    - Simple

```
pointcut greeting() : execution(void myexample.Hello.sayHello ());

after() returning() : greeting() {
      System.out.println(" World!");
    }
```

Source :
http://blogexpertease.alti.com/index.php?2010/03/26/158-la-programmation-orientee-aspect-aop-avec-aspectj

Concepts de base / glossaire

- Advice (greffon)
  - Exemples (AspectJ)
    - Plus complexe

```
pointcut sysout(Object message) : args(message) && call (public void
print*(*)) && target(java.io.PrintStream) && !within(BankAspectSysout) ;

before(Object message) : sysout(message) {
    System.out.println("Wrote " + message.toString()  + " to sysout at:" +
thisJoinPoint.getSignature());
    System.out.print(thisJoinPointStaticPart.getSourceLocation() + ":");
}
```

Source :
http://www.tomjanofsky.com/aspectj_tutorial.html

Concepts de base / glossaire

- Déclaration inter-types
  (= mécanisme d'introduction)
  - Ajout dans le code de l'aspect de nouveaux éléments structuraux au code de l'application
  - Extension du comportement d'une classe

```
protected String Point.name;
public int Point.compareTo(Object o) { … }
```

  - Modification des relations d'héritage

```
declare parents: Point implements Comparable;
```

Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Concepts de base / glossaire

- Weaver (tisseur d'aspects)
  - Infrastructure mise en place pour greffer le code des aspects dans le code des classes
  - Selon les tisseurs cette greffe peut avoir lieu
    - directement sur le code source (avant la compilation)
    - durant la compilation
    - après la compilation sur le code compilé (avant l'exécution)
    - pendant l'exécution (at runtime)

➔ Quel est l'intérêt de faire du tissage à l'exécution ?

Sources :

http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm

Code d'un aspect avec AspectJ

- Pointcuts et advices
- Déclarations inter-types
- Autres éléments Java (attributs, méthodes)

```
public aspect World {
        pointcut greeting() : execution(void myexample.Hello.sayHello
());
        after() returning() : greeting() {
                System.out.println(" World!");
    }
}
```

Source :
http://blogexpertease.alti.com/index.php?2010/03/26/158-la-programmation-orientee-aspect-aop-avec-aspectj

Outils orientés aspects

- Disponibles dans de nombreux langages
  - Java, C++, PHP, Python, CommonLisp, Ruby...
- Outils Java (tisseurs)
  - AspectJ
    - De loin le plus connu
    - Tisseur d'aspect à la compilation
    - Génère du bytecode
    - Utilisation de fichiers XML ou d'annotations Java
    - Plugin Eclipse : AJDT
- Frameworks Java : Spring, JBoss, Equinox

POA et méthodes de conception

- Comment utiliser la POA avec de l'OO ?
  - Bien isoler les responsabilité des objets
  - Ajouter des comportements à l'aide d'aspects
    - Décorateur, proxy…
  - Utiliser et améliorer les patterns du GoF
    - Importances des rôles dans les patterns
    - Exemple : pattern Observateur
      - Pattern disséminé dans le code
      - Notification = crosscut

Source :
http://hannemann.pbwiki.com/f/OOPSLA2002.pdf

Plan

- Introduction
- Métaprogrammation par annotations
- Programmation Orientée Aspects
- Conclusion

Conclusion

- Séparer certains traitements du code permet de
  - Abstraire / réutiliser les traitements répétitifs
  - Faciliter la configuration d'un logiciel
  - Séparer les préoccupations métier des services annexes
  - Passer facilement à une architecture à base de composants
- En revanche, cela ne dispense pas de
  - Réfléchir à la structure générale de votre application
  - Dérouler une méthode de conception rationnelle

Références

- Métaprogrammation par annotations
  - Javadoc Java SE (annotations)
  - Java EE (processeurs)
  - http://java.sun.com/javase/6/docs/technotes/guides/apt/GettingStarted.html
  - http://adiguba.developpez.com/tutoriels/java/tiger/annotations/
  - http://yourmitra.wordpress.com/2008/02/15/java-custom-annotations-runtime-and-compile-build-time-processing/

Références

- Programmation Orientée Aspects
  - http://hannemann.pbwiki.com/Design+Patterns
  - http://hannemann.pbwiki.com/f/OOPSLA2002.pdf
  - http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect
  - http://www.dotnetguru.org/articles/dossiers/aop/quid/AOP15.htm
  - http://www.eclipse.org/aspectj/index.php
  - http://blogexpertease.alti.com/index.php?2010/03/26/158-la-programmation-orientee-aspect-aop-avec-aspectj
  - http://www.tomjanofsky.com/aspectj_tutorial.html