

CENTRALE
L Y O N

Programmation Orientée Aspect - POA

BTD/MAI/POA

1

CENTRALE
L Y O N

Plan

- Principes
- AspectJ
- JAC

Bertrand DAVID : Méthodologies avancées 'informatisation

BTD/MAI/POA

2

CENTRALE L Y O N	<h2>Aspect-oriented programming (AOP)</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Aspect-oriented programming, or AOP, complements object-oriented programming by allowing the developer to dynamically modify the static object-oriented model to create a system that can grow to meet new requirements, allowing an application to adopt new characteristics as it develops. ■ *AOP provides a solution for abstracting cross-cutting code that spans object hierarchies without functional relevance to the code it spans. Instead of embedding cross-cutting code in classes, AOP allows you to abstract the cross-cutting code into a separate module (known as an aspect) and then apply the code dynamically where it is needed. You achieve dynamic application of the cross-cutting code by defining specific places (known as pointcuts) in your object model where cross-cutting code should be applied. At runtime or compile time, depending on your AOP framework, cross-cutting code is injected at the specified point
BTD/MAI/POA	3

CENTRALE L Y O N	<h2>Historique</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Les bases : <ul style="list-style-type: none"> → Méta-programmation → Réflexivité ■ 1996 : <ul style="list-style-type: none"> → Gregor Kiczales pose les bases d'AspectJ à Xerox PARC, Palo Alto ■ 1998 : <ul style="list-style-type: none"> → Première version d'AspectJ ■ 1998 ► 2005 <ul style="list-style-type: none"> → AspectJ → JAC (Java Aspect Components) → JBoss AOP → AspectWerkz → Spring AOP → www.aosd.net ...
BTD/MAI/POA	4

CENTRALE L Y O N	<h2>Qu'est-ce que la Programmation Orientée Aspect (POA) ?</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Les développeurs d'applications ont généralement plusieurs préoccupations, qui peuvent être divisées en deux catégories :<ul style="list-style-type: none">→ Les préoccupations fonctionnelles (cœur de métier de l'application)→ Les préoccupations techniques (non fonctionnelles) liées à l'environnement d'exécution.■ Le principe de séparation cherche à rendre indépendantes ces préoccupations afin d'améliorer la modularité des applications.<ul style="list-style-type: none">→ La POO a permis d'atteindre un certain degré d'indépendance, sans totalement casser les liens entre préoccupations
BTD/MAI/POA	5

CENTRALE L Y O N	<h2>POA</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ La POA est un nouveau paradigme de programmation qui cherche à améliorer la séparation des préoccupations en modularisant les éléments transversaux des applications■ La POA ne remet pas en cause les autres paradigmes de programmation (approche procédurale ou objet), mais les étend en offrant des mécanismes complémentaires pour mieux modulariser les différentes préoccupations d'une application
BTD/MAI/POA	6

CENTRALE L Y O N	<h2>AOP</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ La définition originale de l'AOP est la suivante : <i>"la programmation par aspect est une technique novatrice permettant de mettre en facteur certaines responsabilités dont la réalisation est à priori dispersée à travers un système, fût-il orienté objet."</i> (ouf!)
BTD/MAI/POA	7

CENTRALE L Y O N	<h2>POA - origines</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ La métaprogrammation■ La réflexivité■ Les protocoles à méta-objets ■ Au-delà de la POO<ul style="list-style-type: none">→ Modularité→ Réutilisation→ Sûreté→ Extensibilité ■ Limites de la POO<ul style="list-style-type: none">→ Prise en compte de fonctionnalités transversales→ Dispersion du code
BTD/MAI/POA	8

CENTRALE L Y O N	<h2>POA + POO</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Solution aux deux challenges :<ul style="list-style-type: none">→ Fonctionnalités transversales→ Dispersion du code■ POA sur POO<ul style="list-style-type: none">→ Les CLASSES constituent le socle de l'application. Il s'agit des données et des traitements, qui correspondent au cœur de la problématique de l'application et qui répondent aux besoins premiers de celle-ci.→ Les ASPECTS intègrent aux classes des éléments (classes, méthodes, données) supplémentaires qui correspondent à des fonctionnalités transversales ou à des fonctionnalités dont l'utilisation est dispersée.
BTD/MAI/POA	9

CENTRALE L Y O N	<h2>Démarche en deux temps</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Dans un premier temps<ul style="list-style-type: none">→ Opérer un processus de décomposition pour identifier les classes et les aspects en fonction des besoins identifiés.→ Il n'y a pas de règle universelle pour dire qu'une fonctionnalité est du ressort d'une classe plutôt que d'un aspect. Une fonctionnalité peut, selon les contextes applicatifs, être implémentée tantôt sous forme de classe, tantôt sous forme d'aspect.<ul style="list-style-type: none">• C'est le cas, par exemple, des contraintes de temps d'exécution, qui sont des aspects dans les applications de gestion et des classes dans les applications de supervision de processus industriels.■ Dans un second temps<ul style="list-style-type: none">→ La conception d'une application orientée aspect s'accompagne d'un processus de recomposition. Il s'agit de mettre en commun les classes et les aspects pour obtenir une application opérationnelle. Ce processus est réalisé à l'aide des notions de POA de <u>coupe</u> et de <u>point de jonction</u>. Ces notions permettent de désigner les emplacements du programme autour desquels vont être greffés les <u>aspects</u>, et donc les fonctionnalités qu'ils implémentent.
BTD/MAI/POA	10

CENTRALE L Y O N	<h2>La notion d'aspect</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ La programmation objet induit un découpage en fonction des données qui seront encapsulées dans les classes. Certaines fonctionnalités s'accrochent mal de ce découpage, et les instructions correspondant à leur utilisation se retrouvent dispersées dans l'ensemble de l'application.■ Tout changement dans l'utilisation de ces fonctionnalités implique dès lors de devoir consulter et modifier un grand nombre de fichiers. On dit en ce cas que leur code est éparpillé ou dispersé (<i>scattered</i> en anglais)■ Exemples : La gestion des traces
BTD/MAI/POA	11

CENTRALE L Y O N	<h2>Analyse du phénomène de dispersion</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ La dispersion du code est inéluctable : une classe fournit, via ses méthodes, un ou plusieurs services. Il est aisé de rassembler tous les services dans une classe, mais rien dans l'approche objet ne permet de rassembler les utilisations de ce service.■ La dispersion d'une fonctionnalité dans une application est un frein à son développement, sa maintenance et à son évolution.■ Le code devient embrouillé (<i>tangled</i> en anglais).
BTD/MAI/POA	12

CENTRALE
L Y O N

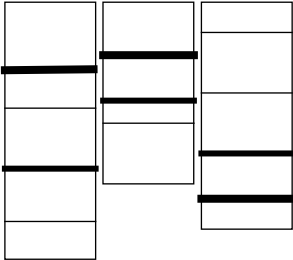
Bertrand DAVID : Méthodologies avancées 'informatisation

BTD/MAI/POA

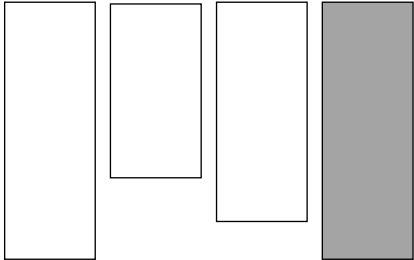
Définition

- **Aspect** : Entité logicielle qui capture une fonctionnalité transversale à une application
- **G. Kiczales** : *AOP is about capturing a crosscutting structure* (Le rôle de la POA est de capturer une structure transversale).

Sans aspect



Avec aspect



BTD/MAI/POA

13

CENTRALE
L Y O N

Bertrand DAVID : Méthodologies avancées 'informatisation

BTD/MAI/POA

Intégration de fonctionnalités transversales

- Deux éléments entrent dans l'écriture d'un aspect :
 - La *coupe*
 - Le code *advice*
- Le code d'un aspect dépend de la fonctionnalité à implémenter. Au lieu de coder directement, on fait appelle à une API dédiée. Le code de l'aspect « se contente » de faire des appels à l'API technique.
- Dans ce cas, l'aspect est un mécanisme d'intégration entre une fonctionnalité transversale, implémentée à l'aide d'une API dédiée et le code de l'application.
- Par le biais de cette bonne pratique de POA, l'aspect n'implémente plus directement la fonctionnalité transversale mais s'appuie pour cela sur une API dédiée.

BTD/MAI/POA

14

CENTRALE
L Y O N

Bertrand DAVID : Méthodologies avancées 'informatisation

Services non fonctionnels et aspects

- Il existe une séparation communément admise dans les applications entre le parties métier et non fonctionnelle.
- La partie métier, appelée également fonctionnelle, est le code qui correspond au comportement concret de l'application.
- La partie non fonctionnelle concerne tous les services supplémentaires fournis par l'application, principalement techniques (sécurité, contrôle d'accès, ...)

BTD/MAI/POA 15

CENTRALE
L Y O N

Bertrand DAVID : Méthodologies avancées 'informatisation

Inversion de dépendances

- La POA introduit un lien entre le code des classes et celui des aspects
- En général, toute application fait appelle explicite aux fonctions ou méthodes d'une API technique. Lorsque cette API change il faut répercuter ces modifications à l'application.
- Le changement apporté par la POA est le suivant :

```

graph TD
    subgraph Left
        App1[Application] -- Utilise --> ST1([Service technique])
    end
    subgraph Right
        App2[Application] -- Utilise --> AS[Aspect]
        AS -- Utilise --> ST2([Service technique])
    end
    
```

- Un aspect utilise un service technique pour l'intégrer à une application. La dépendance est donc inversée, l'application ne dépend plus du service technique, mais c'est l'aspect qui dépend de l'application

BTD/MAI/POA 16

CENTRALE L Y O N	<h2>Inversion</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Le principal avantage de l'inversion réside dans l'allègement de la tâche du développeur applicatif.■ En POA, le développeur applicatif ne se préoccupe pas des services techniques. C'est le développeur d'aspects qui, en plus de l'implémentation concrète du service, propose une façon d'intégrer ce service aux applications métier.
BTD/MAI/POA	17

CENTRALE L Y O N	<h2>Aspects et Frameworks</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Les frameworks partagent avec la POA l'inversion de dépendance entre code des applications et celui des bibliothèques.■ Un framework est un ensemble de classes fournissant un cadre de base réutilisable pour la construction d'applications :<ul style="list-style-type: none">→ Application réutilisable semi-complète, qui peut être spécialisée pour produire des applications personnalisées.
BTD/MAI/POA	18

CENTRALE L Y O N	<h2>Le tissage d'aspects</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Une application POA est composée d'un ensemble de classes et d'un ou plusieurs aspects. Une <u>opération automatique</u> est nécessaire pour obtenir une application opérationnelle, intégrant des classes et des aspects.■ Cette opération est désignée sous terme de tissage (<i>weaving</i> en anglais) et le programme qui la réalise est un tisseur d'aspects (<i>aspect weaver</i>). L'application obtenue à l'issue du tissage est dite tissée.■ Définition Tisseur d'aspects (<i>aspect weaver</i>) est un programme qui réalise une opération d'intégration entre un ensemble de classes et un ensemble d'aspects.■ L'opération de tissage peut être effectuée à la compilation ou à l'exécution.
BTD/MAI/POA	19

CENTRALE L Y O N	<h2>Le tissage à la compilation</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Le tisseur est un programme qui, avant l'exécution, prend en entrée un ensemble de classes et un ensemble d'aspects et fournit en sortie une application augmentée des aspects. Un tel tisseur est donc très proche du compilateur.■ Les classes fournies en entrée du tisseur peuvent se présenter soit sous forme de code source, soit sous forme de code intermédiaire.■ Avec le tissage à la compilation, les aspects sont pris en compte lors de la compilation, leurs effets sont présents dans le code final, mais il n'y a aucun moyen de les distinguer du reste du code.■ Ce tissage est dit <u>statique</u>.
BTD/MAI/POA	20

CENTRALE L Y O N	<h2>Le tissage à l'exécution</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Le tisseur est un programme qui permet d'exécuter à la fois l'application et les aspects qui lui ont été ajoutés. Les aspects ont une existence propre lors de l'exécution.■ On travaille au niveau du code intermédiaire, préalablement compilé.■ L'avantage est que les aspects peuvent être ajoutés, enlevés ou modifiés à chaud, pendant que l'application s'exécute.<ul style="list-style-type: none">→ Prise en compte de fortes contraintes de disponibilité.■ Un tel mode de tissage est dit <u>dynamique</u>.
BTD/MAI/POA	21

CENTRALE L Y O N	<h2>Exemples d'aspects</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Il n'existe pas encore de consensus sur les étapes pour identifier des aspects.■ Le fait de décider si une fonctionnalité est du ressort du métier, et donc implémentable sous forme de classe, ou des aspects reste de l'ordre du choix conceptuel.<ul style="list-style-type: none">→ Contrainte temps réel<ul style="list-style-type: none">• Aspect pour une application de gestion• Classes métier pour une application de contrôle de trafic■ Manipulation de collections<ul style="list-style-type: none">→ Aspect pour exprimer la persistance■ IHM<ul style="list-style-type: none">→ Framework STRUTS et modèle de conception MVC■ Design patterns<ul style="list-style-type: none">→ Mise en œuvre de certains patterns à l'aide d'aspects.
BTD/MAI/POA	22

CENTRALE L Y O N	<h2>Les points de jonction</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Un aspect est une structure transversale à une application. ■ Le <u>point de jonction</u> (<i>join point</i>) est un point dans l'exécution d'un programme autour duquel un ou plusieurs aspects peuvent être ajoutés <ul style="list-style-type: none"> → Cette notion de point de jonction peut être comparée à celle de point d'arrêt d'un programme. ■ Dans le cadre de la POA, un point de jonction désigne un endroit du programme où l'on souhaite ajouter un aspect.
BTD/MAI/POA	23

CENTRALE L Y O N	<h2>Les différents types de points de jonction</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ La notion de point de jonction révèle un caractère éminemment dynamique, car il s'agit d'événements qui surviennent lors de l'exécution du programme, il est donc nécessaire de s'appuyer sur la structure des programmes. ■ Les points de jonction s'appuient sur des méthodes (début de méthode, activation d'une méthodes, ...). ■ Les éléments suivants peuvent servir d'appui de points de jonctions : <ul style="list-style-type: none"> → Méthodes (appel, début, fin) → Constructeurs → Classes et interfaces (plus discutable) → Exceptions → Attributs → Blocs de code (mais granularité très/trop fin) → Instructions (mais granularité très/trop fin) ■ La tâche du programmeur d'aspect consiste à sélectionner les points de jonctions pertinents pour son aspect.
BTD/MAI/POA	24

CENTRALE L Y O N	<h2>Les coupes</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ La <u>coupe</u> (<i>crosscut</i>) désigne un ensemble de points de jonction. ■ Une coupe est définie à l'intérieur d'un aspect. Dans les cas simples, une seule coupe suffit pour définir la structure transversale d'un aspect. Dans les cas plus complexes, un aspect est associé à plusieurs coupes. ■ Les notions de coupes et de points de jonction sont liées par leur définition. Pourtant, leur nature est très différente : <u>Une coupe est un élément de code défini dans un aspect, alors qu'un point de jonction est un point dans l'exécution d'un programme.</u> ■ Si une coupe désigne un ensemble de points de jonction, un point de jonction donné peut appartenir à plusieurs coupes d'un même aspect ou d'aspects différents. Dans ce cas, il est nécessaire de définir l'ordre d'application des différentes coupes et des différents aspects autour des points de jonction.
BTD/MAI/POA	25

CENTRALE L Y O N	<h2>Exemples de coupes</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Coupe de modification de données : <ul style="list-style-type: none"> → Désigne toutes les instructions d'écriture sur un ensemble d'attributs (sauvegarde de manière persistante des données, mise à jour des vues associées aux données). ■ Coupe d'appel de méthode : <ul style="list-style-type: none"> → Désigne toutes les instruction d'appel pour un ensemble de méthodes (pour reconstituer le diagramme de séquence d'appels de méthodes). ■ Coupe d'exécution de méthode : <ul style="list-style-type: none"> → Désigne toutes les exécutions d'un ensemble de méthodes (pour calculer les temps d'exécution respectifs d'un ensemble de méthodes).
BTD/MAI/POA	26

CENTRALE L Y O N	<h2>Les codes advice</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Une coupe définit où un aspect doit être greffé dans l'application. Elle désigne pour cela un ensemble de points de jonction.■ Le code advice définit ce que l'aspect greffe dans l'application (les instructions ajoutées par l'aspect).■ Définition :<ul style="list-style-type: none">→ Code advice est le bloc de code définissant le comportement d'un aspect.■ Un aspect comporte un ou plusieurs codes advice. Chacun définit un comportement particulier pour son aspect. Le code advice joue en quelque sorte le même rôle qu'une méthode, à la différence que les codes advice sont associés à une coupe, et donc à des points de jonction et ont un type.■ Un code advice définit l'intégration d'une fonctionnalité a priori transversale.
BTD/MAI/POA	27

CENTRALE L Y O N	<h2>Exemples de codes advice</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Code advice de sauvegarde de données :<ul style="list-style-type: none">→ Permet de sauvegarder la valeur d'un attribut dans une base de données ou un fichier. Est utilisé en conjonction avec une coupe de modification de données, permet d'implémenter un aspect de persistance de données.■ Code advice de synchronisation :<ul style="list-style-type: none">→ Garantit que, en présence d'exécution multithread, l'accès à un ensemble de méthodes reste cohérent. Il s'agit, par exemple, de bloquer l'accès à une méthode lorsque cette dernière est en cours d'exécution.
BTD/MAI/POA	28

CENTRALE L Y O N	<h2>Les différents types de code advice</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Il existe trois types principaux de code advice<ul style="list-style-type: none">→ Before : le code est exécuté avant les points de jonction→ After : après→ Around : avant et après
BTD/MAI/POA	29

CENTRALE L Y O N	<h2>Le mécanisme d'introduction</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Le mécanisme d'introduction permet d'étendre le comportement d'une application en lui ajoutant des éléments, essentiellement des attributs ou des méthodes.■ Ces nouveaux éléments sont introduits, c'est-à-dire ajoutés à l'application.■ L'introduction est un mécanisme purement additif, qui ajoute de nouveaux éléments à une classe.
BTD/MAI/POA	30

CENTRALE L Y O N	<h2>Exemples d'introductions</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Introduction d'interface d'accès distant :<ul style="list-style-type: none">→ Ajoute une interface d'accès distant à une classe et rend accessible les instances de cette classe via un middleware (par exemple Java RMI ou des services Web). ■ Introduction d'un attribut de type date :<ul style="list-style-type: none">→ Ajoute un attribut de type date afin que la date de création de toutes les instances de cette classe puisse être enregistrée.
BTD/MAI/POA	31

CENTRALE L Y O N	<h2>La composition d'aspects</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Conflits entre aspects :<ul style="list-style-type: none">→ Incompatibilité→ Dépendance→ Redondance ■ L'ordonnement des aspects■ L'ordre de tissage des aspects
BTD/MAI/POA	32

CENTRALE L Y O N	<h2>Cas concret AspectJ</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Programmation avec AspectJ ■ AspectJ ajoute des nouveaux mots-clés à Java. ■ AspectJ fait le tissage statique – à la compilation.
BTD/MAI/POA	33

CENTRALE L Y O N	<h2>Classe Customer</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> Package aop.aspectj; /** * Dans l'application Gestion de commandes, * cette classe represente l'activite d'un client * qui achete 1 DVD et 2 CD, * puis demande le calcul du montant de la commande. */ **/ public class Customer { /** * L'activite du client. */ public void run() { Order maCommande = new Order(); maCommande.addItem("DVD",1); maCommande.addItem("CD",2); double montant = maCommande.computeAmount(); System.out.println("Montant de la commande : "+montant+" euros"); } Public static void main (String[] args) { New Custoler().run(); } } </pre> <ul style="list-style-type: none"> ■ La classe <i>Customer</i> est le point d'entrée dans l'application. ■ La méthode <i>main</i> crée un objet de la classe <i>Customer</i> et appelle la méthode <i>run</i>. ■ Celle-ci crée une commande (objet <i>myOrder</i>), appelle deux fois la méthode <i>addItem</i> en fournissant une référence et une quantité puis calcule le montant de la commande et l'affiche.
BTD/MAI/POA	34

CENTRALE L Y O N	<h1>Classe Order</h1>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.aspectj; import java.util.HashMap; import java.util.Iterator; import java.util.Map; /** * Dans l'application Gestion de commandes, cette classe * représente les commandes des clients. * Les articles commandés sont ajoutés à une map. */ public class Order { /** Les articles commandés. */ private Map articles = new HashMap(); /** * Ajout d'un article a la commande. * * @param reference la reference de l'article. * @param quantite la quantite commande. */ public void addItem(String reference, int quantite) { articles.put(reference,new Integer(quantite)); System.out.println(quantite+" article(s) "+reference+" ajoute(s) a la commande"); } } </pre> <ul style="list-style-type: none"> ■ Les commandes sont gérées par la classe <i>Order</i> ■ La classe <i>Order</i> stocke les articles et les quantités commandées dans une table de hachage (attribut <i>items</i>). Cette table est indexée par la référence de l'article. <pre> /** * Calcul du montant de la commande. * * @return le montant de la commande. */ public double computeAmount() { double montant = 0.0; for (Iterator iter = articles.entrySet().iterator(); iter.hasNext();) { Map.Entry entry = (Map.Entry) iter.next(); String article = (String) entry.getKey(); Integer quantite = (Integer) entry.getValue(); double prix = Catalog.getPrice(article); montant += prix*quantite.intValue(); } return montant; } </pre> <ul style="list-style-type: none"> ■ La méthode <i>addItem</i> ajoute un article à la commande et affiche un message à l'écran pour signaler cette opération. ■ La méthode <i>computeAmount</i> passe en revue l'ensemble des articles détermine le prix de chacun et retourne le montant total de la commande.
BTD/MAI/POA	35

CENTRALE L Y O N	<h1>Classe Catalog</h1>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.aspectj; import java.util.HashMap; import java.util.Map; /** * Dans l'application Gestion de commandes, cette classe * représente le catalogue d'articles disponibles. */ public class Catalog { /** * Les tarifs. * Map avec comme clé la reference de l'article et comme * valeur le prix de l'article. */ private static Map tarif = new HashMap(); /** Les tarifs des articles du catalogue. */ static { tarif.put("CD", new Double(15.0)); tarif.put("DVD", new Double(20.0)); } /** * Calcul du prix d'un article. * * @param reference la reference de l'article. * @return le prix de l'article. */ public static double getPrice(String reference) { Double prix = (Double) tarif.get(reference); return prix.doubleValue(); } } </pre> <ul style="list-style-type: none"> ■ Le prix de chaque article est déterminé à l'aide de la méthode <i>Catalog.getPrice</i> ■ La classe <i>Catalog</i> stocke les prix de chaque article dans la table de <i>hachage priceList</i>. ■ Cette table est indexée par les références des articles. ■ Les valeurs de cette table sont les prix des articles. ■ Le bloc de code <i>static</i> initialise la table <i>priceList</i> avec deux articles : un CD à 15 € et un DVD à 20 € ■ La méthode <i>getPrice</i> retourne le prix d'un article dont la référence est passée en paramètre.
BTD/MAI/POA	36

CENTRALE L Y O N	<h2>Aspect TraceAspect</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.aspectj; import java.util.Date; import java.util.Map; /** * Dans l'application Gestion de commandes, * cet aspect trace les appels a la methode addItem. */ public aspect TraceAspect { /** * Cette coupe est tres detaillee puisque seuls les * appels a la methode * dont la signature est citee sera tracee. */ pointcut toBeTraced(): call(public void aop.aspectj.Order.addItem(String,int)); /** * Code advice qui affiche les messages de trace. */ void around(): toBeTraced() { System.out.println("-> Avant appel addItem"); proceed(); System.out.println("-> Apres appel addItem"); } } </pre>
BTD/MAI/POA	<ul style="list-style-type: none"> ■ AspectJ étend la syntaxe Java avec de nouveaux mots-clés. ■ <i>Aspect</i>, ici TraceAspect, entité similaire à une classe ■ Les coupes ■ Le mot-clé <i>pointcut</i> définit une coupe, ici <i>toBeTraced</i>. ■ Chaque coupe est associée à une expression, qui définit son ensemble de points de jonction. ■ Les points de jonction sont typés, et chaque type est associé à un mot-clé. ■ Une expression de coupe peut comporter des points de jonctions de types différents. ■ Ce n'est pas le cas dans cet exemple simple de la coupe <i>toBeTraced</i>, où seul le type <i>call</i> est utilisé. ■ Les points de jonction de type <i>call</i> désignent les points où une méthode est appelée. La méthode à prendre en compte est fournie, via son profil, entre parenthèses. ■ Ici on constate que le profil est très précis, il ne désigne qu'une seule méthode parmi toutes celles existant dans l'application. ■ Il est possible d'utiliser des <i>wildcards</i> pour désigner en une seule expression plusieurs méthodes. ■ La coupe <i>toBeTraced</i> est associée à la méthode <i>addItem</i>. Cela ne signifie pas qu'il n'y a qu'un seul point de jonction pour cette coupe. Tous les points où la méthode est appelée sont concernés. ■ Ici, il y a en deux, puisque la méthode <i>addItem</i> est appelée deux fois dans la méthode <i>run</i> de la classe <i>Customer</i>.
	37

CENTRALE L Y O N	<h2>Les codes advice</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ L'aspect TraceAspect comporte un seul code advice. ■ Les codes advice peuvent être de trois types : <ul style="list-style-type: none"> → before → after → around ■ Dans le cas <i>around</i>, <i>proceed</i> déclenche l'exécution du point de jonction. ■ <i>Proceed</i> délimite donc une partie avant et une partie après le code advice ■ Les coupes désignent certains emplacements stratégiques dans l'application ■ Les <i>wildcards</i> fournissent des mécanismes exprimant la genericité <ul style="list-style-type: none"> → Les noms de méthode et de classe : * → Les profils de méthode : * → Les sous-types : *
BTD/MAI/POA	38

CENTRALE L Y O N	<h2>Le mécanisme d'introspection de point de jonction</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Introspection : inspection de l'intérieur pour obtenir des informations sur le code ayant déclenché le point de jonction. ■ En aspectJ l'introspection s'effectue avec le mot-clé <code>thisJoinPoint</code>. De la même façon que <code>this</code> est une référence à l'objet en cours, <code>thisJoinPoint</code> est une référence à un objet décrivant le point de jonction en cours. Cet objet appartient à la classe prédéfinie <code>org.aspectj. Lang.JoinPoint</code> <ul style="list-style-type: none"> → Principales méthodes : <ul style="list-style-type: none"> • <code>Object[]getArgs()</code> Retourne les arguments du point de jonction • <code>Signature getSignature ()</code> Retourne la signature du point de jonction • <code>String getKind ()</code> Retourne le type de point de jonction • <code>SourceLocation getSourceLocation ()</code> Retourne la localisation du point de jonction dans le code source • <code>Object getTarget ()</code> Retourne l'objet cible du point de jonction • <code>Object getThis ()</code> Retourne l'objet source du point de jonction
BTD/MAI/POA	39

CENTRALE L Y O N	<h2>Aspect AddDate</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.aspectj; import java.util.Date; /** * Dans l'application Gestion de commandes, * cet aspect introduit une date pour chaque * commande. */ public aspect AddDate { /** Attribut date introduit dans la classe * Order. */ private Date Order.date; /** 2 méthodes getDate() et setDate() * introduites dans Order. */ public Date Order.getDate() { return date; } public void Order.setDate(Date date) { this.date = date; } /** Un constructeur introduit dans Order. */ public Order.new (Date date) { this.date = date; } /** * Après la construction d'une nouvelle * instance de Order, * on initialise le l'attribut introduit date avec * la date courante. */ after() : initialization(Order.new (..)) { Order myOrder = (Order) thisJoinPoint.getTarget(); myOrder.date = new Date(); } /** * Coupe et code advice "artificiel" pour * tester l'ajout de l'attribut date. * A la fin de la méthode main, * on crée et une instance de Order et on * affiche la valeur du champ date. */ after() : execution(void Customer.main(..)) { Order myOrder = new Order(); System.out.println("Date de creation de la commande "+myOrder.date); } </pre>
BTD/MAI/POA	40

CENTRALE L Y O N	<h2>Aspect TraceAspect2</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.aspectj; import java.util.Date; import java.util.Map; /** * Dans l'application Gestion de commandes, * cet aspect trace les appels a toutes les * methodes de la classe Order. */ public aspect TraceAspect2 /** * Cette coupe capture les appels a toutes les * methodes de la classe Order. */ pointcut toBeTraced(): call(* aop.aspectj.Order.*(..)); /** * Code advice qui affiche les messages de * trace. */ Object around(): toBeTraced() { System.out.println("-> Avant appel"); Object ret = proceed(); System.out.println("-> Apres appel"); return ret; } </pre>
BTD/MAI/POA	<p>■ Exemple d'utilisation de wildcards</p> <p style="text-align: right;">41</p>

CENTRALE L Y O N	<h2>Aspect TraceAspect3</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.aspectj; import java.util.Date; import java.util.Map; /** * Dans l'application Gestion de commandes, * cet aspect trace les appels a toutes les methodes de la * classe Order et * affiche des informations sur chaque point de jonction * rencontre. */ public aspect TraceAspect3 { /** * Cette coupe capture les appels a toutes les * methodes de la classe Order. */ pointcut toBeTraced(): call(* aop.aspectj.Order.*(..)); /** * Code advice qui affiche les messages de trace et * les informations sur l'introspection des points de * jonction. */ Object around(): toBeTraced() { String methodName = thisJoinPoint.getSignature().getName(); Object[] args = thisJoinPoint.getArgs(); Object caller = thisJoinPoint.getThis(); Object callee = thisJoinPoint.getTarget(); System.out.println("-> Debut methode "+methodName); System.out.print("-> "+args.length+" parametre(s) "); for (int i = 0; i < args.length; i++) { System.out.print(args[i]+" "); } System.out.println(); System.out.println("-> "+caller+" vers "+callee); Object ret = proceed(); System.out.println("<- Fin methode "+methodName); return ret; } } </pre>
BTD/MAI/POA	<p>■ Pour illustrer l'introspection nous voulons afficher</p> <ul style="list-style-type: none"> → Le nom de la méthode dont l'appel est intercepté → Les paramètres de l'appel → La référence de l'objet courant, autrement dit l'objet appelant → La référence de l'objet cible, autrement dit l'objet appelé <p style="text-align: right;">42</p>

CENTRALE L Y O N	<h2>Environnement Java Aspect Components (JAC)</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Alors que AspectJ est un langage, avec JAC, l'écriture d'un programme orienté aspect se fait en Java pur.■ Une API dédiée permet la définition d'aspects, de coupes et de code advice.■ Une deuxième différence entre AspectJ et JAC concerne le tissage d'aspects :<ul style="list-style-type: none">→ Avec AspectJ le tissage intervient au moment de la compilation,→ JAC effectue le tissage lors de l'exécution.■ Ceci introduit plus de souplesse dans la gestion, car les aspects peuvent être ajoutés ou retirés dynamiquement.■ JAC est un logiciel Open Source sous licence GNU LGPL
BTD/MAI/POA	43

CENTRALE L Y O N	<h2>Premier exemple de JAC</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ On prend le même exemple que précédemment (Gestion de commande) et on lui applique le même aspect de gestion de traces.■ Cet aspect sera écrit avec JAC. Il s'agit d'inspecter le comportement de l'application afin de connaître les méthodes appelées et l'ordre de ces appels.
BTD/MAI/POA	44

CENTRALE L Y O N	<h2>Premier aspect de trace</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.jac; import org.objectweb.jac.core.AspectComponent; /** * Dans l'application Gestion de commandes, * ce composant d'aspect trace les executions de la methode addItem. */ public class TraceAspect extends AspectComponent { public TraceAspect() { /** * Cette coupe est tres detaillee puisque seules les executions * de la methode dont la signature est citee sera tracee. */ pointcut(".*", "aop.jac.Order", "addItem.*", "aop.jac.TraceWrapper", null, false); } } </pre>
BTDA/MAI/POA	<ul style="list-style-type: none"> ■ Il s'agit de tracer les exécutions de la méthode addItem de la classe Order ■ Le code de l'aspect comporte deux classes : <ul style="list-style-type: none"> → TraceAspect pour définir la coupe → TraceWrapper pour définir le code advice qui, dans la terminologie JAC, s'appelle un wrapper ■ Aucun mot-clé nouveau n'est introduit. ■ La classe TraceAspect qui définit une coupe hérite de la classe AspectComponent, qui est fournie par JAC. ■ Les aspects JAC sont parfois désignés sous le terme de composant d'aspect ■ La classe TraceAspect comporte un constructeur qui appelle la méthode pointcut. Cette méthode est héritée de la classe AspectComponent. Elle permet de déclarer une nouvelle coupe. ■ Six paramètres sont nécessaires : <ul style="list-style-type: none"> → 3 premiers concernent la définition de la coupe <ul style="list-style-type: none"> • Quels objets • Quelles classes • Quelles méthodes → 4° fournit le wrapper associé → 5° et 6° indiquer les propriétés de la coupe <ul style="list-style-type: none"> • Gestionnaire d'exception • Mode d'instanciation
	45

CENTRALE L Y O N	<h2>Les wrappers</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.jac; import org.aopalliance.intercept.ConstructorInvocation; import org.aopalliance.intercept.MethodInvocation; import org.objectweb.jac.core.AspectComponent; import org.objectweb.jac.core.Wrapper; /** * Dans l'application Gestion de commandes, * ce wrapper affiche les messages de trace. */ public class TraceWrapper extends Wrapper { public TraceWrapper(AspectComponent ac) { super(ac); } /** * Interception des executions de constructeurs. */ public Object construct(ConstructorInvocation ci) throws Throwable { return proceed(ci); } /** * Interception des executions de methodes. */ public Object invoke(MethodInvocation mi) throws Throwable { System.out.println("-> Avant addItem"); Object ret = proceed(mi); System.out.println("-> Apres addItem"); return ret; } } </pre>
BTDA/MAI/POA	<ul style="list-style-type: none"> ■ Wrapper à la place de code advice ■ Le code d'un wrapper, comme un code advice, s'exécute avant ou après un point de jonction. ■ Avec JAC les wrappers sont définis dans une classe différente de celle de l'aspect. Cela permet de réutiliser les wrappers indépendamment des aspects et vice-versa ■ Le nom de la classe implémentant un wrapper est fourni au moment de la définition du pointcut. Il s'agit ici de la classe TraceWrapper ci-contre. ■ Les wrappers JAC sont des classes qui héritent de la classe Wrapper. Un wrapper JAC permet d'exécuter des instructions avant et après les points de jonction d'une coupe. ■ C'est systématiquement un wrapper de type around. Il n'y a pas de type before ou after avec JAC. Ceci n'est pas limitation, car il suffit de définir un wrapper around sans partie before ou after. ■ L'interface d'un wrapper JAC est conforme à l'API AOP Alliance. ■ Un wrapper doit définir un constructeur prenant en paramètre une instance de la classe AspectComponent. Ce constructeur peut contenir n'importe quelle instruction, mais il faut au minimum qu'il appelle le constructeur hérité. Ce constructeur est appelé par le framework JAC lorsqu'il instancie un wrapper. ■ Les points de jonction pris en compte par un wrapper sont soit des exécutions de méthode, soit des exécutions de constructeurs. ■ Les méthodes invoke et construct permettent de définir pour ces deux types de point de jonction du code avant et après. ■ Les méthodes invoke et construct sont invoquées par le framework JAC juste avant l'exécution du point de jonction appartenant à une coupe. Leur signature est imposée par JAC : elle retourne un Object et lève éventuellement une exception Throwable. En Java, Throwable est la classe ancêtre de toutes les exceptions. ■ La méthode invoke prend en paramètre une instance de MethodInvocation, et la méthode construct une instance de ConstructorInvocation. Ces deux paramètres permettent d'inspecter le point de jonction. ■ La méthode proceed délimite les parties avant et après d'un wrapper. ■ Dans la méthode invoke, le paramètre mi de type MethodInvocation doit être transmis lors de l'appel proceed. ■ Il va de même pour le paramètre ci de type ConstructorInvocation dans la méthode construct. ■ La valeur de retour de proceed correspond à la valeur retournée par le point de jonction. Elle doit être retournée par le wrapper.
	46

CENTRALE L Y O N	<h2>Configuration d'aspect</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ La configuration d'aspect est un concept essentiel de JAC ■ Chaque aspect JAC est associé à un fichier de configuration. Ce fichier fournit des valeurs des propriétés. <ul style="list-style-type: none"> → Par exemple, pour un aspect de transaction, il s'agit d'indiquer quelles méthodes de l'application doivent être rendues transactionnelles, de même pour persistance. → Les fichiers de configuration d'aspects de JAC sont des fichiers texte (extension .acc). → Chaque ligne correspond à l'appel d'une méthode publique de l'aspect qui lui est associé avec des paramètres associés
BTD/MAI/POA	47

CENTRALE L Y O N	<h2>Coupe configurable</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.jac; import org.objectweb.jac.core.AspectComponent; /** * Dans l'application Gestion de commandes, * ce composant d'aspect permet de définir des coupes * et * de tracer les executions de methodes appartenant a * ces coupes. */ public class TraceAspect2 extends AspectComponent { /** * L'appel de la methode trace dans le fichier de * configuration de * l'aspect definira une coupe a partir des trois * expressions fournies. * Cette coupe sera associee au wrapper * TraceWrapper2. */ public void trace(String objectPE, String classPE, String methodPE) { pointcut(objectPE, classPE, methodPE, "aop.jac.TraceWrapper2", null,false); } } </pre> <ul style="list-style-type: none"> ■ Au lieu de coder en dur la définition de la coupe dans son constructeur, on peut définir une méthode trace. ■ La méthode trace accepte trois paramètres objectPE, classPE, methoPE avec le suffixe PE pour pointcut une expression. ■ Trace appelle avec ces 3 paramètres la méthode pointcut, héritée de AspectComponent. ■ Elle définit donc une coupe en précisant que traceaspect2 est le wrapper associé. ■ Pour obtenir un comportement identique à celui de TracsAspect, le fichier de configuration traceaspect2.acc contient: <ul style="list-style-type: none"> → trace ".*" "aop.jac.Order" ".*" "addItem(java.lang.String,int):void « ■ En fournissant dans le fichier de configuration la définition de la coupe, TraceAsept2 peut être tissée à n'importe quelle application. De plus la méthode trace peut être utilisée plusieurs fois dans le fichier de configuration, ce qui permet de définir plusieurs coupes.
BTD/MAI/POA	48

CENTRALE L Y O N	<h2>Descripteur d'application</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ # ----- ■ # - Application: Gestion de commandes - ■ # - Aspect: Trace de la methode addItem - ■ # - Descripteur d'application - ■ # ----- ■ # L'identifiant de l'application ■ applicationName: Gestion de commandes ■ # Classe contenant la methode main de lancement de l'application ■ launchingClass: aop.jac.Customer ■ # Le(es) composant(s) d'aspect tisse(s) ■ # Ici seul le composant d'aspect traceid ■ # dont le fichier de configuration est traceaspect.acc ■ # est tisse ■ aspects: traceid traceaspect2.acc true ■ # La definition du(des) composant(s) d'aspect ■ jac.acs: traceid aop.jac.TraceAspect <ul style="list-style-type: none"> ■ Il s'agit d'un fichier décrivant une application orientée aspect avec JAC. ■ Chaque ligne commence par le nom d'une propriété : valeur de la propriété <ul style="list-style-type: none"> → ApplicationName Gestion de commandes → launchingClass aop.jac.Customer → Aspect: traceid traceaspect2.acc true <ul style="list-style-type: none"> • Avec trois valeurs: identificateur, fichier de configuration et valeur true ou false pour indiquer si l'aspect est initialement tissé ou non. → Jac.acs: traceid aop.jac.TraceAspect2 <ul style="list-style-type: none"> • Identificateur et classe Java d'implémentation
BTD/MAI/POA	49

CENTRALE L Y O N	<h2>Compilation</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Cinq catégories de fichiers : <ul style="list-style-type: none"> → Fichiers Java de l'application (Customer.Java, Order.Java, Catalog.Java) → Fichiers Java d'aspect (TraceAspect2.java) → Fichiers Java des wrappers (Tracewrapper.java) → Fichiers de configuration (traceaspect2.jac) → Fichier descripteur d'application (customer.jac) → Les fichiers Java sont compilables avec n'importe quel compilateur Java, avec la bibliothèque jac.jar → Le résultat de la compilation est un ensemble de fichiers .class
BTD/MAI/POA	50

CENTRALE L Y O N	<h2>Exécution</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ L'exécution d'une application orientée aspect avec JAC se fait en demandant au framework de charger un fichier .jac■ Le lancement du framework s'effectue à l'aide de la bibliothèque jac.jar
BTD/MAI/POA	51

CENTRALE L Y O N	<h2>Les coupes</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ La définition d'une coupe passe par l'appel de la méthode pointcut de la classe AspectComponent■ Trois catégories de paramètres sont associées à la méthode pointcut:<ul style="list-style-type: none">→ Expressions de coupe : définissent les points de jonction faisant partie de la coupe. La méthode pointcut accepte 3 ou 4 paramètres de types chaîne de caractères pour définir les expressions de coupe.→ Wrapper associé à une coupe : fournit le code, écrit dans une sous-classe de la classe Wrapper (avec 2 à 3 paramètres pour la méthode pointcut).→ Gestionnaire d'expression : Les exceptions peuvent générer des exceptions. Le gestionnaire est une méthode de la classe implémentant le wrapper. La méthode pointcut accepte un paramètre de type chaîne de caractères pour désigner ce gestionnaire. La valeur du paramètre correspond au nom de la méthode ou à null.
BTD/MAI/POA	52

CENTRALE L Y O N	<h2>Les expressions de coupe</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Trois expressions de coupe dans les cas standards et une pour des points de jonction de programmes répartis. ■ Les 3 méthodes concernent les classes, les objets et les méthodes (expression de c/o/m), elles peuvent être vues comme des filtres. <ul style="list-style-type: none"> → Expression de classe : filtres sur les noms de classes → Expression d'objet : filtres sur les noms d'objets → Expression de méthode : filtres sur les signatures de méthodes <ul style="list-style-type: none"> • addItem(java.lang.String, int):void → Expressions régulières GNU regexp (mécanismes des expressions régulières GNU) <ul style="list-style-type: none"> • Opérateur de désignation . • Opérateurs de répétitions +, * • Opérateur ensemble [] • Opérateur d'exclusion ^ • Intervalle de valeurs - • Opérateurs ALL, , &&, ! Respectivement *, OR, AND, NOT → Les opérateurs de type de méthode permettent de caractériser le comportement de méthodes vis-à-vis des attributs de leur classe <ul style="list-style-type: none"> • ACCESSORS : lecture d'au moins un attribut de la classe • MODIFIERS : écriture d'au moins un attribut de la classe • GETTERS (liste) : méthodes retournant des valeurs • SETTERS (liste) : méthodes effectuant des écritures • ADDERS (liste) : méthodes ajoutant des éléments à la collection • REOVERS (liste) méthodes retirant des éléments
BTD/MAI/POA	53

CENTRALE L Y O N	<h2>Le wrapper associé à une coupe</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ La méthode pointcut permet de définir des coupes. ■ Ses premiers paramètres concernent les expressions de coupe : classe, objet et méthode ■ Pour indiquer le wrapper associé à la coupe, trois cas se présentent : <ul style="list-style-type: none"> → Deux paramètres : une chaîne de caractères wrapperClassName et un booléen one2one respectivement le nom de la classe implémentant le wrapper (sous-classe de org.objectweb.jac.core Wrapper) et pour one2one : <ul style="list-style-type: none"> • Si false, le wrapper est le singleton, tout le monde partage le même wrapper • Si true, chaque point de jonction a sa propre instance de wrapper → Trois paramètres : les deux précédents et un tableau d'objets initParameters. Ce tableau est utilisé lors de l'instanciation pour transmettre des valeurs initiales au constructeur du wrapper. → Le développeur fournit directement une instance de la classe implémentant le wrapper.
BTD/MAI/POA	54

CENTRALE L Y O N	<h1>Les wrappers</h1>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Don a deux catégories de classes : celles pour définir les aspects et celles pour définir les wrappers <ul style="list-style-type: none"> → Les wrappers héritent de la classe <code>org.objectweb.jac.core Wrapper</code> → Les wrappers sont par défaut de type <code>around</code>, les types <code>before</code> et <code>after</code> n'existent pas explicitement. → Les instances de wrapper sont créées par le framework JAC → La classe implémentant les wrappers doit fournir un constructeur de type <code>org.objectweb.jac.core.AspectComponent</code> → Une classe <code>MyWrapper</code> implémentant un wrapper : <pre>import org.objectweb.jac.core Wrapper import org.objectweb.jac.core.AspectComponent public class MyWrapper extends Wrapper { public MyWrapper (AspectComponent ac) {super (ac);} }</pre> → Un wrapper fournit des parties avant et après pour les points de jonction de type exécution de méthode et exécution de constructeur (<code>invoke</code> et <code>construct</code>) → Profil de la méthode <code>invoke</code> <pre>public Object invoke (MéthodInvocation mi) throws Throwable ;</pre> → Schéma général : <pre>public Object invoke (MéthodInvocation mi) throws Throwable { // code avant Object ret = proceed (mi) ; // code après Return ret ; }</pre>
BTD/MAI/POA	55

CENTRALE L Y O N	<h1>Les wrappers</h1>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> → Les constructeurs → La méthode <code>construct</code> : <pre>public Object construct (ConstructorInvocation ci) throws Throwable ;</pre> → Schéma général : <pre>public Object construct (ConstructorInvocation ci) throws Throwable { // code avant Object ret = proceed (mi) ; // code après Return ret ; }</pre> → Même si aucun code avant ou après n'est défini, la méthode <code>construct</code> est obligatoirement dans un wrapper
BTD/MAI/POA	56

CENTRALE L Y O N	<h2>Le mécanisme d'introspection de point de jonction</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none"> ■ Introspection : inspecter de l'intérieur le point de jonction courant (méthode ou constructeur) ■ Les informations sont accessibles via le paramètre de type <code>MéthodInvocation</code> de la méthode <code>invoke</code> et <code>ConstructorInvocation</code> de la méthode <code>construct</code> ■ Interface <code>JoinPoint</code> est le type de base pour toutes les interceptions : toute interception concerne un point de jonction. ■ On y retrouve : <ul style="list-style-type: none"> → La méthode <code>proceed</code> → La méthode <code>getThis</code> qui retourne l'objet applicatif dans lequel le point de jonction a été déclenché. → La méthode <code>getStaticPart</code> retourne l'élément du programme dans lequel le point de jonction a été déclenché (méthode ou constructeur) de type <code>java.lang.reflect.AccessibleObject</code> ■ Les deux interfaces <code>ConstructorInvocation</code> et <code>MethodInvocation</code> étendent <code>Invocation</code>. Ces deux méthodes fournissent une méthode pour retourner respectivement le constructeur et la méthode correspondant au point de jonction.
BTD/MAI/POA	57

CENTRALE L Y O N	<h2>Exemple d'introspection</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<pre> package aop.jac; import org.aopalliance.intercept.ConstructorInvocation; import org.aopalliance.intercept.MethodInvocation; import org.objectweb.jac.core.AspectComponent; import org.objectweb.jac.core.Wrapper; /** Dans l'application Gestion de commandes, * ce wrapper affiche les messages de trace. */ public class TraceWrapper2 extends Wrapper { public TraceWrapper2(AspectComponent ac) { super(ac); } /** Interception des executions de constructeurs. */ public Object construct(ConstructorInvocation ci) throws Throwable { return proceed(ci); } /** Interception des executions de methodes. */ public Object invoke(MethodInvocation mi) throws Throwable { System.out.println("> Avant "+mi.getMethod().getName()); Object ret = proceed(mi); System.out.println("> Apres "+mi.getMethod().getName()); return ret; } } </pre> <ul style="list-style-type: none"> ■ Avant chaque exécution d'une méthode de la coupe, ce wrapper affiche le nom de la méthode exécutée, les paramètres de la méthode et la référence de l'appelé. <p>Trace obtenue :</p> <pre> -> Début méthode addItem -> 2 paramètre(s) DVD 1 -> vers aop.jac.Order@6963d0 1 article(s) DVD ajouté (s) à la commande <- Fin méthode addItem -> Début méthode addItem -> 2 paramètre(s) CD 2 -> vers aop.jac.Order@6963d0 2 article(s) CD ajouté(s) à la commande <- Fin méthode addItem Montant de la commande : 50.0 euros </pre>
BTD/MAI/POA	58

CENTRALE L Y O N	<h2>La chaîne de wrappers</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Plusieurs wrappers peuvent être associés à un même point de jonction.■ On parle alors de chaîne de wrappers. ■ L'appel de la méthode <code>proceed</code> dans un wrapper provoque soit l'exécution du wrapper suivant dans la chaîne, soit l'exécution du point de jonction si la fin de la chaîne est atteinte.
BTD/MAI/POA	59

CENTRALE L Y O N	<h2>Le mécanisme de configuration</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Ce mécanisme permet de faire en sorte que les aspects JAC soient facilement réutilisables.■ Configuration des aspects<ul style="list-style-type: none">→ Fichier <code>.acc</code>■ Groupement permet d'accélérer l'écriture des fichiers de configuration lorsque certains attributs sont fréquemment répétés.■ Configuration d'application<ul style="list-style-type: none">→ Fichier <code>.jac</code>
BTD/MAI/POA	60

CENTRALE L Y O N	<h2>Le mécanisme d'introduction</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ Ce mécanisme permet d'étendre le comportement d'une application en lui ajoutant des éléments■ Le terme renvoie au fait que l'aspect ajoute des éléments du code à l'application.■ Deux catégories d'éléments peuvent être introduits :<ul style="list-style-type: none">→ Des méthodes→ Des gestionnaires d'exception
BTD/MAI/POA	61

CENTRALE L Y O N	<h2>Bibliothèque d'aspects de JAC</h2>
Bertrand DAVID : Méthodologies avancées 'informatisation	<ul style="list-style-type: none">■ JAC fournit en standard 19 aspects<ul style="list-style-type: none">→ IHM – Swing et HTML→ Persistance et transaction→ Répartition→ Seupervision→ Aspects généraux : CacheAC, IntegrityAC, SynchronisationAC
BTD/MAI/POA	62