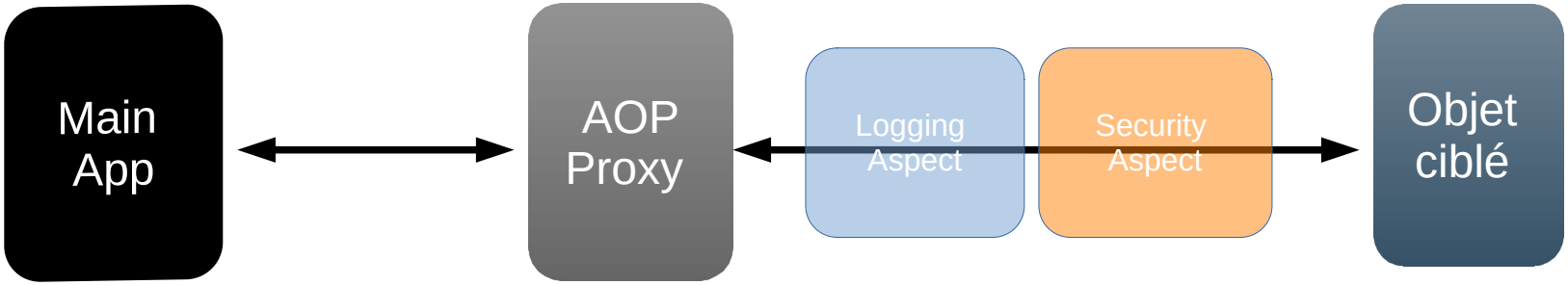
Aspect Use Case

Design pattern Proxy

- Aop solution



MainApp

```
//appel d'une méthode de l'objet cible
objetCible.faitUneAction() ;
```

ObjetCible

```
public void faitUneAction(){
        . . .
}
```

Les bénéfice de AOP

- Le code d'un aspect est défini dans une seule classe
- Votre code métier n'est pas pollué par du code technique
- Les aspects sont configurables , et s'appliquent de façon sélective.

AOP Uses cases

- Logging , sécurité, transactions
- Audit logging (qui , quoi , quand , où)
- Gestion des exceptions (envoi d'SMS , journalisation des erreurs)
- API management(statistiques , analytics, monitoring, "mouchard")

AOP Balance

Avantages

Modules réutilisables
Évite le code spaghetti
Evite le code éparpillé
S'applique sélectivement selon la configuration

Inconvénients

Trops d'aspects gênent l'execution de l'application

Minore les performance pendan tl'execution des aspects

Lexique AOP

- Aspect : module du code pour une fonction transverse
- Advice : quelle action à réaliser et quand elle doit s'executer
- Join Point : où insérer le code pendant l'execution du programme`
- Pointcut : un prédicat pour indiquer quand advice doit être appliqué

Advice

- Before advice : execution avant la méthode
- After finally advice : executer après la méthode
- After returning advice : executer après la méthode (si la méthode s'execute entièrement)
- After throwing advice : exécuter après la methode (en cas d'exception lancée durant l'execution de la méthode)
- Around advice: executer avant et après la méthode

Weaving (tissage)

- Tisser un aspect c'est connecter un aspect à un objet ciblepour créer "un objet advised"
- Il existe le tissage à la compilation, au chargement, à l'exécution (runtime est moins performant)

AOP Frameworks (java)

- Spring AOP : spring utilise aop pour la sécurité, les transactions , gérer le cache...etc automatiquement. Et spring tisse des aspects d'execution , utilise le pattern proxy pour écouter les communications et executer des aspects
- AspectJ : il s'agit du framework original sorti en 2001. Toutes les spécifications AOP sont implémentées par AspectJ. Il fournit un support complet pour joinPoints (methodes, constructeurs, champs), tissage d'aspect (compilation, chargement, post-compilation)

Spring AOP VS AspectJ

Avantages

Spring AOP Plus simple
Spring AOP utilise le DP Proxy
Dans Spring AOP, on peut migrer vers AspectJ en utilisant des annotations @Aspect

AspectJ supporte tous les join points
AspectJ fonctionne avec les POJO, pas seulement avec les spring beans
AspectJ est plus performant
AspectJ offre un support complet

Inconvénients

Spring AOP supporte seulement le join points sur les méthodes

Spring AOP applique uniquement les aspects aux beans crées dans le contexte spring d'une application

Spring AOP ralenti les performances pendnat l'execution des aspects (tissage pendant l'execution)

Le tissgae à la compilation impose des étapes supplémentaires

La syntaxes des pointcut avec AspectJ peut vite devenir complexe

Spring AOP vs AspectJ

- Spring AOP est une implémentation partielle de l'implémentation des spécifications AOP
- Il résoud les problèmes courants d'une application d'entreprise
- Débutez AOP via le module spring AOP
- Si vous avez des besoins complexes à traiter , évoluez vers AspectJ

Cf : www.spring.io
Livre AspectJ in Action (Raminvas Laddad)