Architecture et Design pattern clean code

- AOP.
- Architecture MVC.
- Architecture MVP.
- Architecture MVVM.
- Architecture en couches.
- Architecture Hexagonale / Ports et Adapters.
- Architecture Clean

Architecture et Design pattern clean code

AOP

- **AOP (Aspect Oriented Programming)** : AOP est une approche de programmation qui permet aux développeurs de modulariser les préoccupations transversales, c'est-à-dire les fonctionnalités qui affectent plusieurs parties d'une application.
- Les exemples courants de préoccupations transversales sont la journalisation (logging), la gestion des transactions, la sécurité, etc.
- Dans une application non-AOP, le code pour gérer ces préoccupations serait souvent répété à plusieurs endroits, ce qui rend le code plus difficile à maintenir et à évolutif.

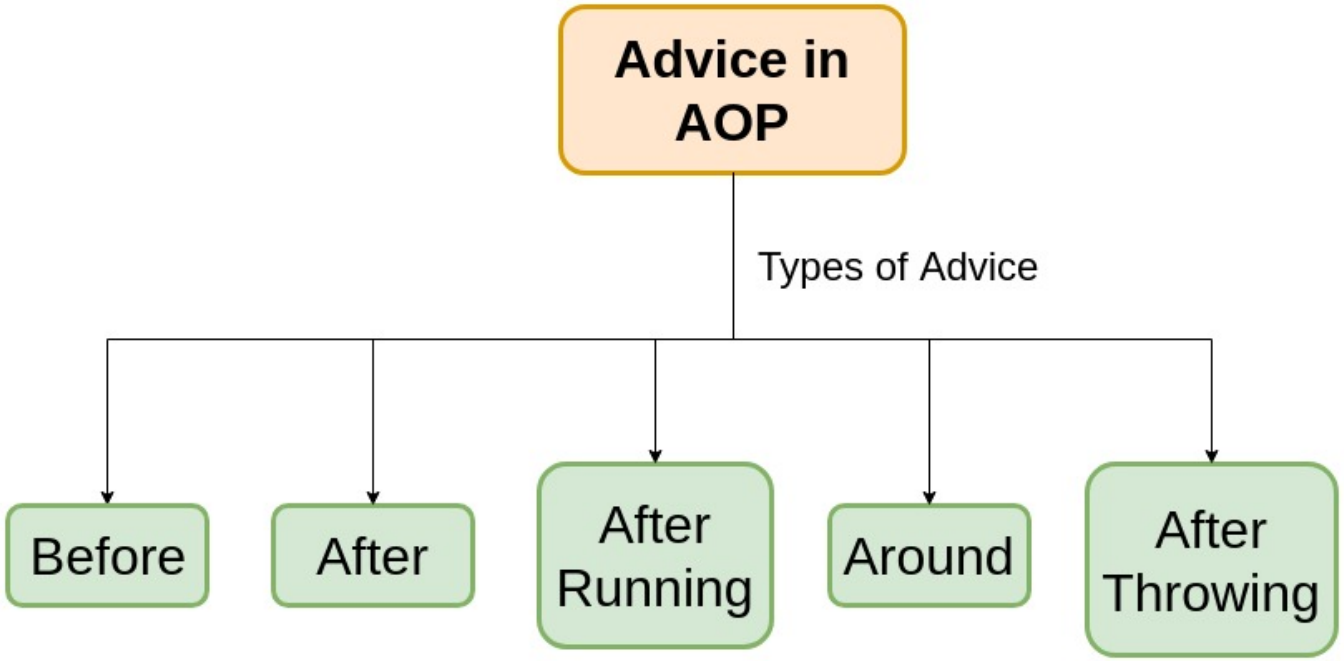
Architecture et Design pattern clean code

AOP

- En AOP, ces préoccupations transversales sont encapsulées dans ce qu'on appelle un "aspect".
- Les "aspects" sont ensuite "tissés" dans votre code à des points définis, appelés "points de jointure" (join points), à l'aide d'un "conseil" (advice) qui définit quand et comment l'aspect doit être appliqué.
- Cela permet de séparer la logique des préoccupations transversales de la logique métier, rendant le code plus propre, plus modulaire et plus facile à maintenir.
- La Programmation Orientée Aspect est souvent utilisée en conjonction avec la Programmation Orientée Objet (POO) pour aider à résoudre les problèmes qui sont difficiles à exprimer efficacement en POO seule.
  Les langages de programmation et les cadres tels que Java (avec Spring AOP et AspectJ).

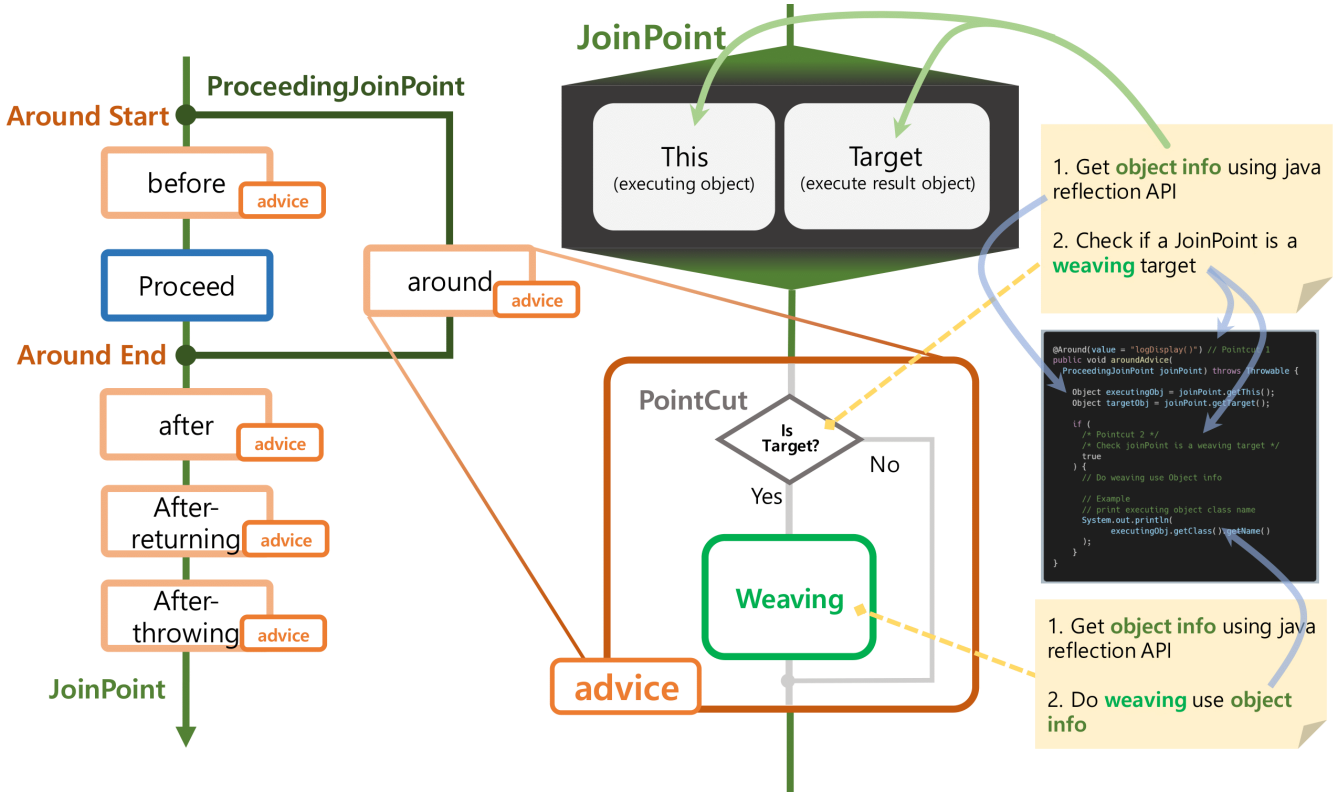
Architecture et Design pattern clean code

AOP



Architecture et Design pattern clean code

AOP



Architecture et Design pattern clean code

Architecture MVC (Model-View-Controller)

Cette architecture divise une application en trois composants interconnectés. Le 'Model' gère les données, la logique et les règles de l'application. La 'View' peut être une sortie textuelle, une interface graphique, etc. Le 'Controller' accepte les entrées et les convertit en commandes pour le 'Model' ou la 'View'

Architecture et Design pattern clean code

Architecture MVP (Model-View-Presenter)

C'est une dérivation de l'architecture MVC, utilisée surtout pour des applications qui nécessitent une grande quantité de travail d'interface utilisateur. Dans cette architecture, le Presenter agit comme un médiateur entre le Model et la View.

Architecture et Design pattern clean code

Architecture MVVM (Model-View-ViewModel)

C'est une autre dérivation de l'architecture MVC, et elle est souvent utilisée pour concevoir des applications qui tirent parti de l'architecture de programmation d'événements, comme de nombreuses applications de Microsoft.

Architecture et Design pattern clean code

Architecture en couches

Cette architecture sépare le code de l'application en couches distinctes, chacune ayant une responsabilité spécifique. Cela peut aider à organiser le code et à le rendre plus lisible et plus maintenable

Architecture et Design pattern clean code

Architecture Hexagonale / Ports et Adapters

- L'architecture hexagonale, aussi connue sous le nom de Ports and Adapters, a été proposée par Alistair Cockburn.
- Elle vise à créer des applications qui sont indépendantes des détails spécifiques de leur infrastructure (comme les bases de données, le réseau, ou les frameworks web).
- L'idée principale derrière l'architecture hexagonale est que le code de l'application (le domaine ou la logique métier) est au centre de la conception, et que toute l'infrastructure, l'interface utilisateur, et les autres détails se trouvent à la périphérie

Architecture et Design pattern clean code

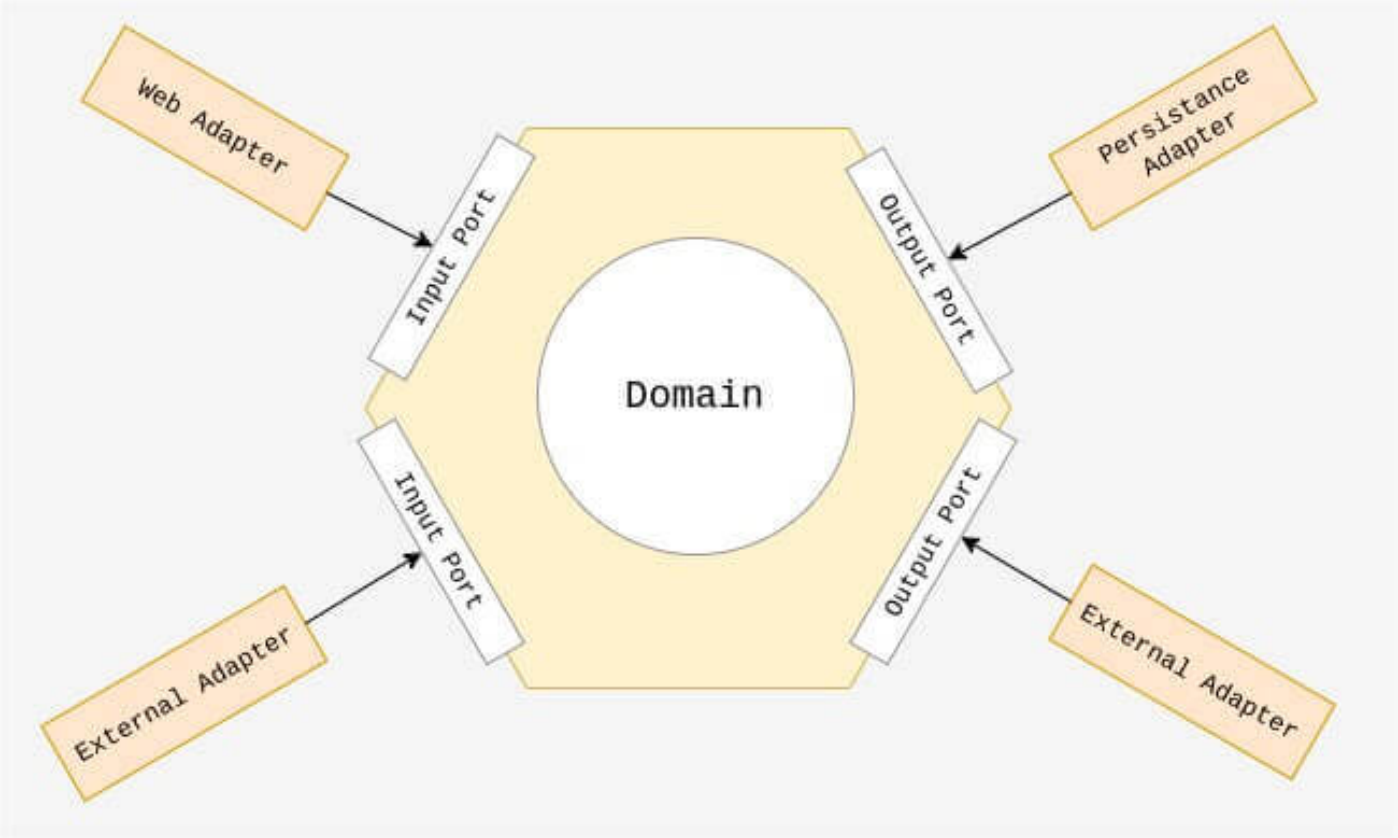
Architecture Hexagonale / Ports et Adapters

Les concepts clés de l'architecture hexagonale :

- **Le Domaine** : C'est le cœur de l'application. Il contient toute la logique métier et n'est pas dépendant de couches extérieures.
- **Les Ports** : Ils sont les interfaces qui définissent les comportements que notre application devrait avoir. On distingue généralement les ports primaires (Driven Ports), qui définissent les fonctionnalités que notre application offre, et les ports secondaires (Driving Ports), qui définissent les fonctionnalités nécessaires à notre application pour interagir avec l'extérieur.
- **Les Adapters** : Ce sont des implémentations spécifiques qui s'interfacent avec l'extérieur. Par exemple, l'implémentation d'un adaptateur pourrait se connecter à une base de données spécifique, envoyer des emails, etc. Il y a deux types d'adaptateurs : ceux qui adaptent le monde extérieur à nos ports (driven adapters) et ceux qui adaptent nos ports au monde extérieur (driving adapters).
- **Les événements de domaine** : Ils sont utilisés pour séparer davantage le cœur du domaine de l'infrastructure, et représentent des actions ou des événements importants qui se produisent dans le domaine.

Architecture et Design pattern clean code

Architecture Hexagonale / Ports et Adapters



Architecture et Design pattern clean code

Architecture Hexagonale / Ports et Adapters

Exemple:
Supposons que nous ayons une fonctionnalité pour emprunter un livre.

- Le Domaine: Il contiendra toutes les règles métier. Par exemple, une règle pourrait être qu'un livre ne peut être emprunté que s'il est actuellement disponible dans la bibliothèque. Une autre pourrait être qu'un utilisateur ne peut emprunter un livre que s'il n'a pas dépassé la limite d'emprunt.
- Les Ports : Nous définirions un port pour chaque fonctionnalité de l'application. Dans ce cas, nous pourrions avoir un port 'BookBorrowingPort' avec une méthode 'borrowBook' qui prend un identifiant de livre et un identifiant d'utilisateur.
- Les Adapters : Nous aurions besoin de deux types d'adaptateurs pour le port 'BookBorrowingPort'. Le premier serait un adaptateur qui appelle la méthode 'borrowBook' lorsque l'utilisateur interagit avec l'interface utilisateur. Le second adaptateur serait un adaptateur qui implémente 'BookBorrowingPort' et qui interagit avec la base de données pour mettre à jour l'état d'un livre lorsqu'il est emprunté

Architecture et Design pattern clean code

Architecture Hexagonale / Ports et Adapters

Exercice Application de gestion de tâches:

- Créez une application simple de gestion de tâches. Cette application doit permettre à un utilisateur de créer des tâches, de les marquer comme terminées, et de les lister.

1. Définir le domaine : Commencez par définir le modèle de votre domaine. Vous pouvez avoir une classe Task qui contient des informations sur une tâche, comme un identifiant, un nom, et un statut pour savoir si la tâche est terminée ou non.
2. Créer les ports : Définissez ensuite les ports de votre application. Vous pourriez avoir un TaskPort avec des méthodes pour créer une tâche, marquer une tâche comme terminée, et obtenir la liste des tâches.

Architecture et Design pattern clean code

Architecture Hexagonale / Ports et Adapters

3. Implémenter les adaptateurs : Créez ensuite les adaptateurs pour vos ports. Vous aurez besoin d'un adaptateur qui implémente TaskPort pour gérer la logique métier de votre application, comme vérifier que le nom d'une tâche n'est pas vide lors de sa création, et un autre adaptateur pour gérer l'interaction avec l'utilisateur, comme prendre les entrées de l'utilisateur et les passer à la méthode appropriée de TaskPort.
4. Testez votre application : Enfin, créez des tests unitaires pour votre application. Grâce à l'architecture hexagonale, vous devriez être en mesure de tester la logique métier de votre application indépendamment de l'interface utilisateur ou de la base de données

Architecture et Design pattern clean code

Architecture Clean

- L'architecture clean, aussi connue sous le nom de "Clean Architecture", est une philosophie de conception de logiciels popularisée par Robert C. Martin (également connu sous le nom d'"Uncle Bob").
- Elle propose une organisation du code qui vise à séparer clairement les préoccupations et à rendre le système facile à comprendre, à développer, à tester et à maintenir.
- L'architecture clean est organisée en cercles concentriques, représentant différents niveaux de l'application :

1. **Entités** : Au centre se trouvent les entités, qui représentent les objets de votre domaine métier. Ces objets n'ont pas besoin de connaître rien d'autre en dehors d'eux-mêmes et contiennent la logique métier qui est propre à l'entreprise.
2. **Cas d'utilisation** : Autour des entités, nous avons les cas d'utilisation (ou règles métier). Ils représentent les actions spécifiques que votre système peut effectuer, comme "Créer un utilisateur" ou "Passer une commande". Les cas d'utilisation orchestrent les entités pour effectuer une action spécifique.

Architecture et Design pattern clean code

Architecture Clean

3. **Adaptateurs d'interface** : Le cercle suivant est celui des adaptateurs d'interface. Ce sont des classes qui convertissent les données d'une forme que les cas d'utilisation et les entités peuvent comprendre en une forme qui est utile pour des choses comme la base de données, le web, l'interface utilisateur, etc.
4. **Frameworks et pilotes** : Enfin, à l'extérieur, se trouvent les frameworks et les pilotes. C'est là que se trouvent les détails spécifiques à la technologie, comme les bases de données, les serveurs web, etc.

- L'idée clé de l'architecture clean est la règle de dépendance : une règle stipule que chaque cercle ne peut dépendre que de ceux qui sont plus proches du centre. Cela signifie que le code de votre application (les cas d'utilisation et les entités) ne dépend pas de l'infrastructure ou des détails spécifiques à la technologie. Cela permet de changer ces détails sans affecter le code de votre application, ce qui rend le système plus flexible et plus facile à maintenir.

Architecture et Design pattern clean code

Architecture Clean