Introduction au Génie Logiciel

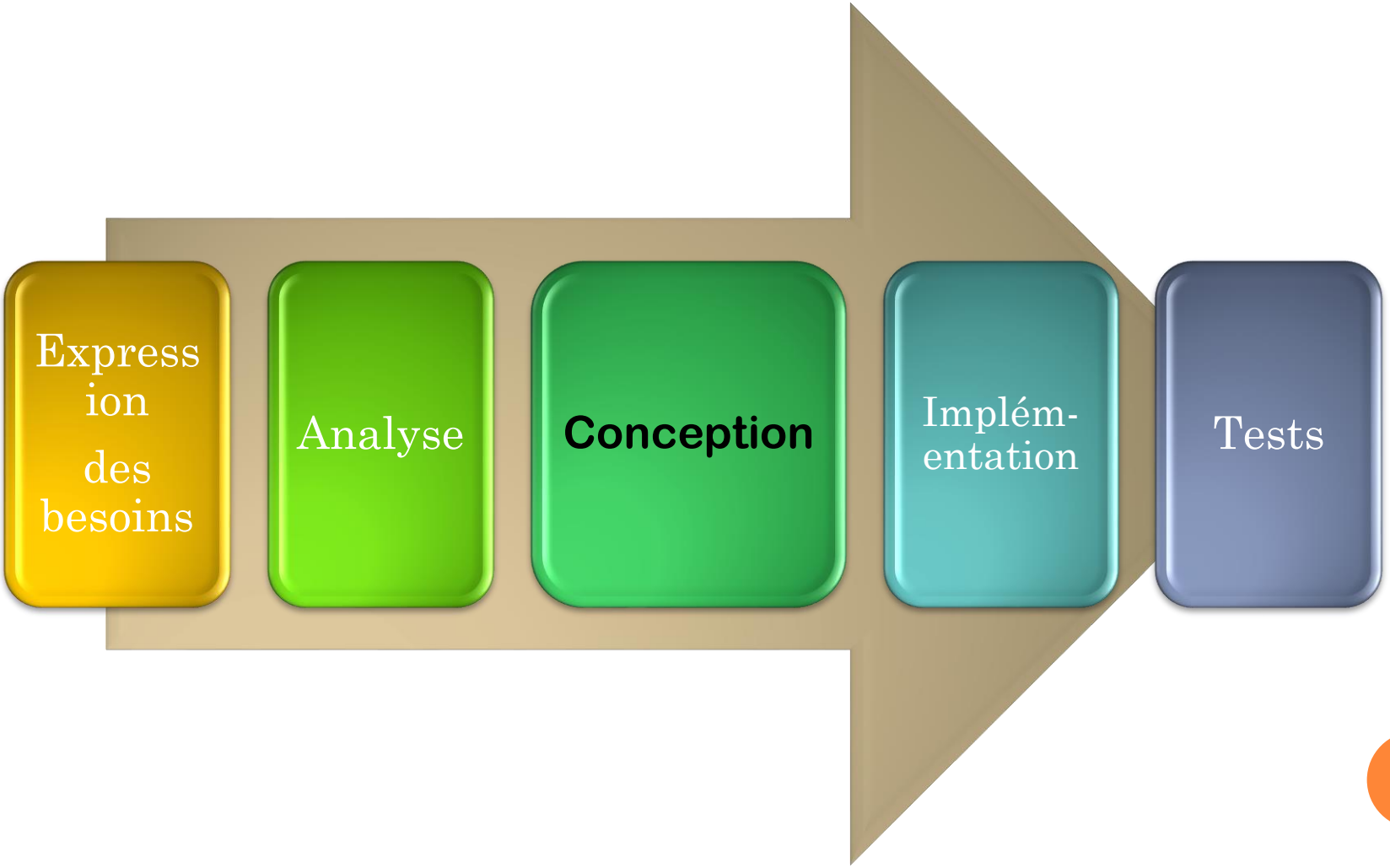
La phase de Conception

Section 1,2,3

1ère année second cycle
Janvier 2022

AMAR BENSABER Djamel

Cycle de Vie


Expression des besoins
Analyse
Conception
Implém-entation
Tests

Objectifs du Cours

Présenter l'activité de conception

Faire le lien entre la conception haut niveau et la conception bas niveau

Présenter les classes, les interfaces et les composants de conception

Découvrir les principes d'une bonne conception

Plan du Cours

Introduction

Définition

- Le modèle d'analyse définit les fonctionnalités (le ***quoi)*** du système à développer.
- La conception s'intéresse à ***comment*** *ces fonctionnalités seront* implémentées.
- La conception se base sur le ***modèle de besoins et le modèle d'analyse.***
- Les solutions proposées par la conception repose sur le ***domaine métier*** et le ***domaine technique***

Domaine métier et domaine technique

- Le domaine métier concerne les actions relatives au ***métier du logiciel.***
- Le domaine technique inclut les ***actions*** techniques à utiliser par la conception (base de données, techniques de persistance,…etc.)
- Par exemple, un logiciel de facturation. La validation de facturation fait partie du domaine métier. La bibliothèque permettant l'enregistrement des données dans une BDD fait partie du domaine technique.

Produits de la Conception

Composants (sous-systèmes)

Classes

Interfaces

Diagrammes de déploiement

Conception et analyse

Elément	Analyse	Conception
Sous-systèmes		Composants définissant la composition du système
Classes	La classe d'analyse représentent des concepts métier. Niveau bas de détail (signature de méthodes, types, …etc.)	Les classes de conception sont plus détaillées et précises. Elles préparent le terrain aux développeur pour l'implémentation. En plus des concepts métier, elles contiennent les classes du domaine technique.

Conception et analyse

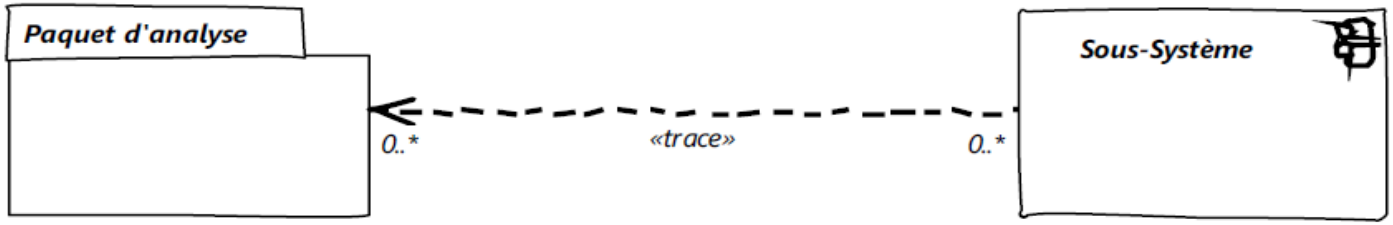
Elément	Analyse	Conception
Interface		Représente une « façade » du système indépendante de l'implémentation
Diagrammes de composants et déploiement		Structure et architecture du système

La trace d'analyse

- Un élément de l'analyse peut être relatif à ***0,1 ou plusieurs éléments de*** conception.
- Cette relation est modélisée en UML avec une dépendance avec le stéréotype « ***trace ».***

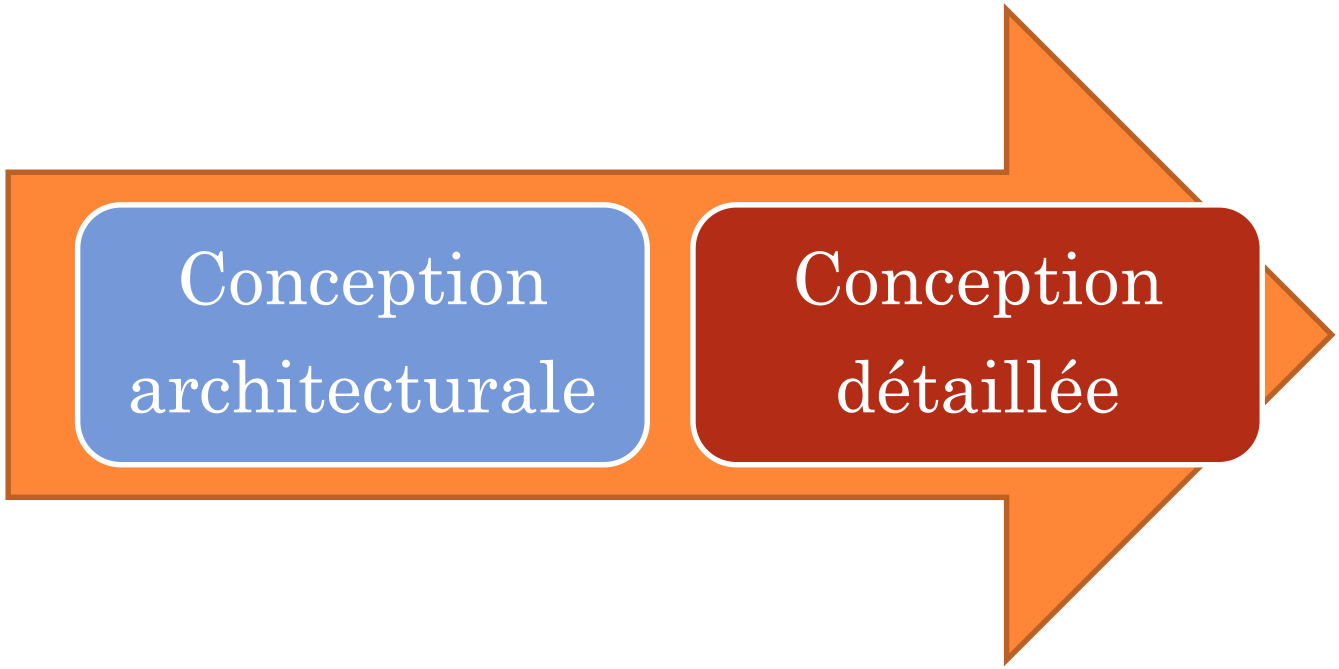
Trace d'analyse

Un paquet d'analyse peut se traduire en un ou plusieurs sous-systèmes (ou 0)



Processus

- La conception passe par deux étapes : la conception architecturale et la conception détaillée.



Processus

- Activité assurée par les architectes.
- La conception architecturale concerne une ***vision descriptive*** des éléments de la conception : sous-systèmes (composants), classes et interfaces.
- La conception détaillée a pour but de ***donner les détails précis*** des éléments produits dans la conception architecturale et préparer au mieux l'implémentation.

Caractéristiques d'une bonne conception

Répond avec précision aux besoins

- La conception doit permettre de fournir une solution technique répondant aux attentes du client. Ne pas faire plus que ce qu'attend le client.

Séparation des fonctionnalités SoC (Separation of Concerns)

- Une bonne conception engendre un haut niveau de modularité où chaque module a des fonctionnalités précises et indépendantes. De tels systèmes sont plus simples et plus faciles à maintenir.

Simplicité

- Quand plusieurs solutions s'offrent pour la résolution du même problème, choisir la plus simple. Plus une conception est simple, plus il est facile de la comprendre et de la maintenir.

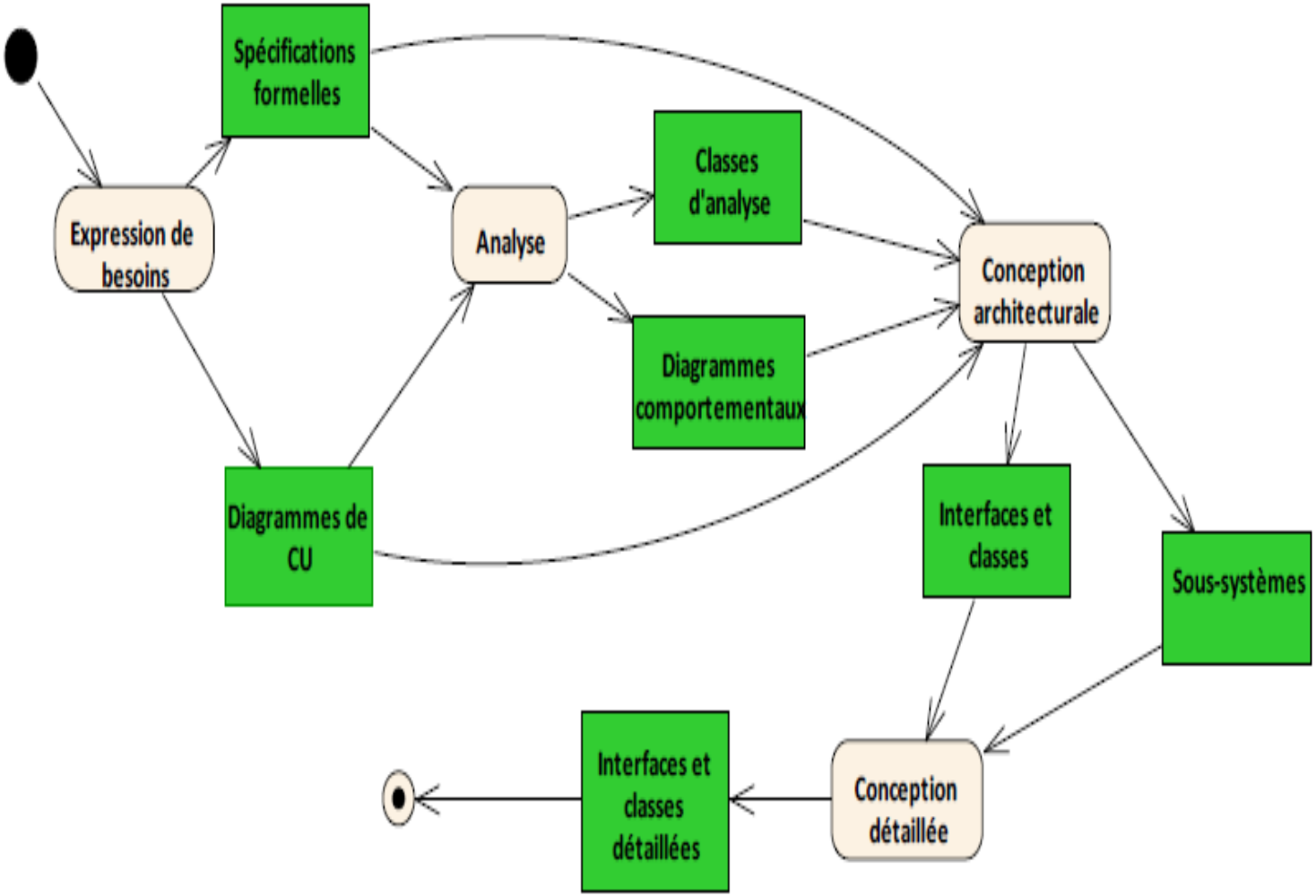
Facilité de maintenance

- Une bonne conception permet d'isoler rapidement les erreurs et des les réparer efficacement

Les Classes de Conception

- La classe est l'élément principal de la *conception orientée objet.*
- La conception architecturale produit des classes et des interfaces sans trop aller dans le détail.
- La conception détaillée finalise le contenu des classes et des interfaces (***classes détaillées*)**.
- Une classe détaillée est le commencement de l'opération ***d'implémentation (codage)***

Classe de conception



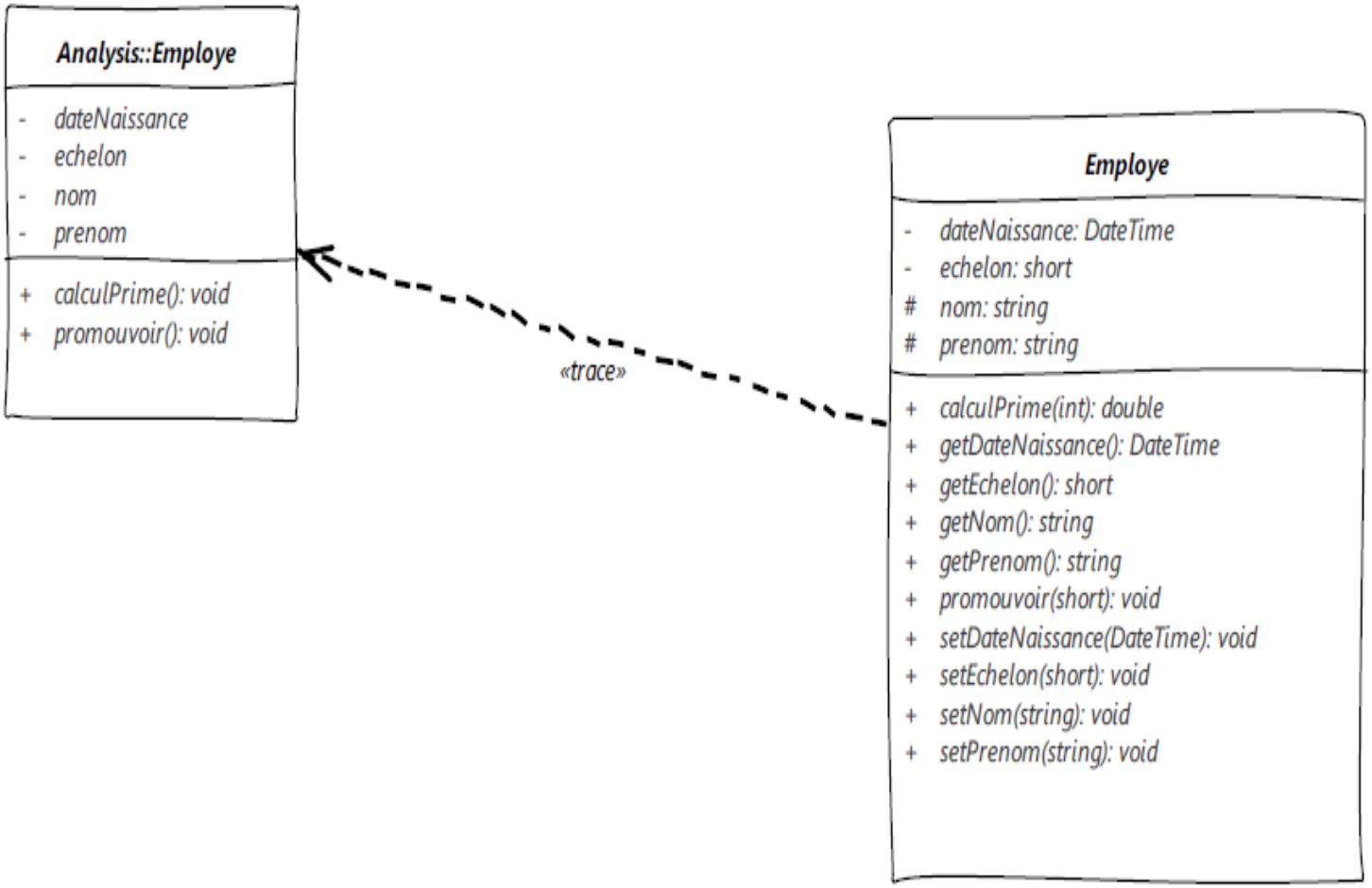
PROVENANCE DES CLASSES DE CONCEPTION

- Les classes de conception peuvent provenir de deux domaines : le ***domaine métier*** et le ***domaine technique.***
- Les classes du domaine métier sont une évolution des ***classes d'analyse.***
- Les classes du domaine technique font partie des ***bibliothèques*** ou ***frameworks*** faisant partie de la ***solution technique*** *(bibliothèques,* GUI, …), par exemple (Spring, Hibernate, Nhibernate, …)

Caractéristiques d'une classe de conception

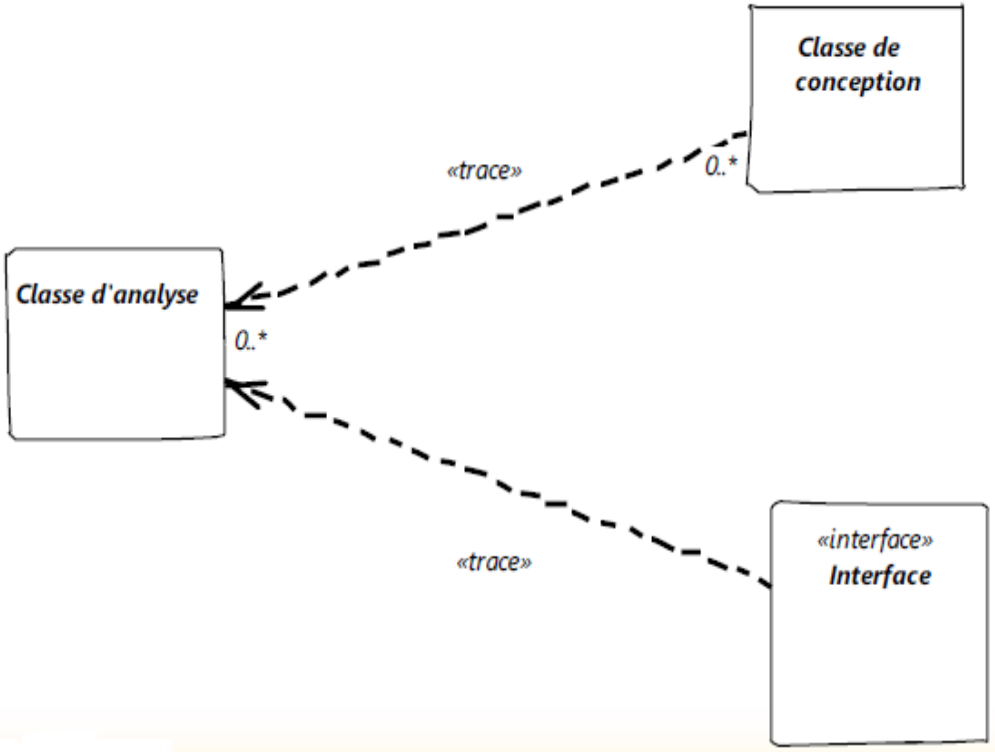
- Une classe d'analyse définit un ***aperçu*** de la classe sans s'occuper des détails techniques.
- La classe de conception (domaine métier) est une ***évolution*** de la classe d'analyse en tenant en compte tous les détails (visibilité, paramètres, ..).
- La classe de conception contient la ***liste complète*** des attributs et des opérations que devrait avoir la classe.

Analyse vers Conception : Exemple



Trace d'Analyse

Une classe d'analyse peut se traduire en plusieurs interfaces et classes



Interfaces et Composants

SECTION 3

Introduction

- La conception passe par ***diviser un système en sous-systèmes*** représentés par des ***composants.***

- L'interaction et la médiation entre ces sous-systèmes se fait en utilisant les interfaces.

QU'EST-CE QU'UNE INTERFACE ?

- Une interface est ***un ensemble d'opérations liées sémantiquement*** qui permettent de séparer la spécification de l'implémentation
- Une interface ***ne contient pas d'implémentation***. Les classes se chargent d'effectuer cette implémentation. Ce lien s'appelle ***réalisation.***
- ***L***es interfaces sont supportées par Java et C#. En C++ le concept d'interface équivaut à une classe abstraite.
- Par convention, les interfaces commencent par I et utilisent un objectif (par exemple, IComparable)

Interface fournie et interface requise

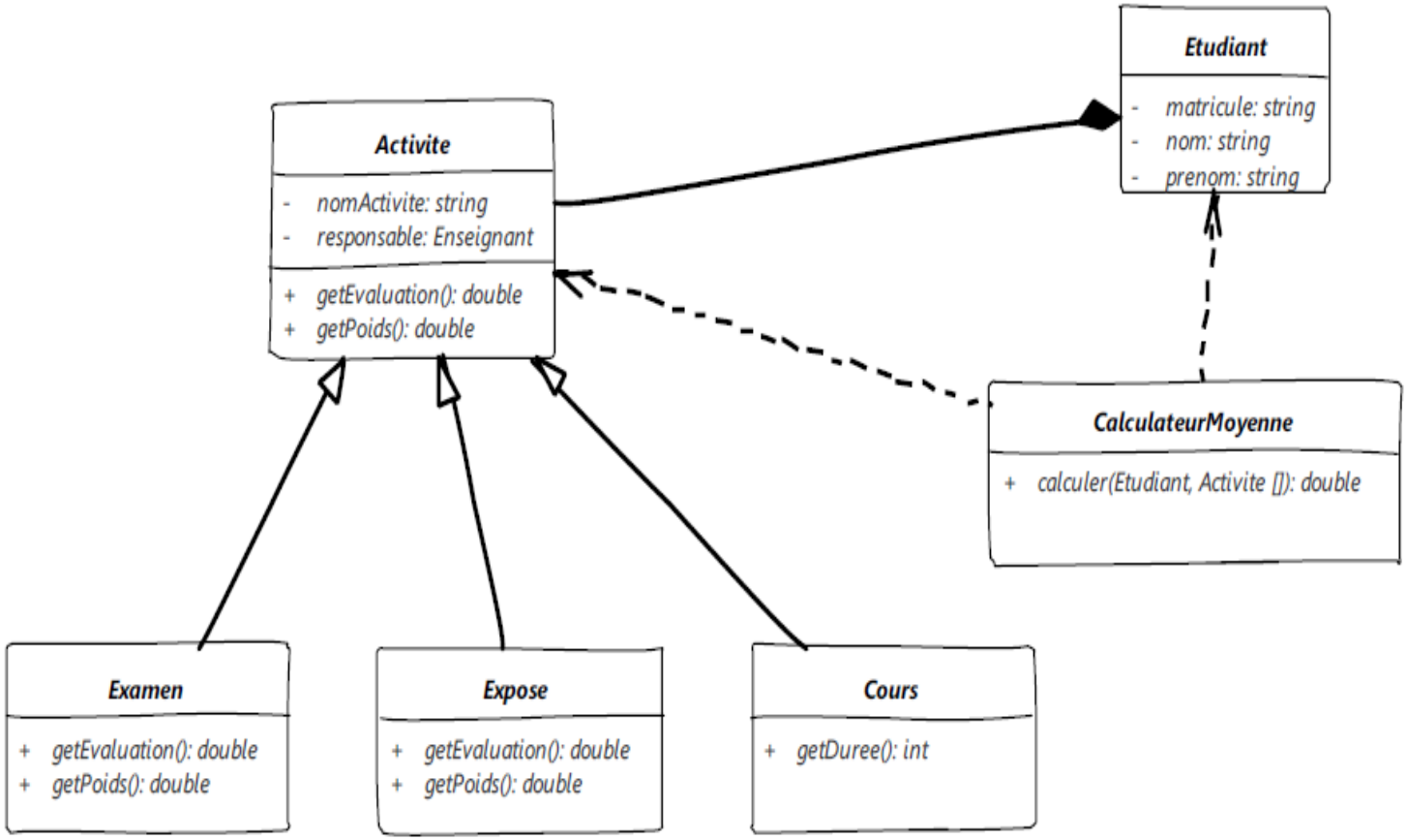
- Une interface fournie représente les fonctionnalités ***assurées par une*** classe ou par un sous-système.
- Une interface requise représente les fonctionnalités ***nécessaires à une*** classe.
- L'assemblage de deux composants sur la base d'une interface fournie / requise est appelé ***contrat.***

Exemple

- Imaginons une gestion de scolarité où l'élève suit un certain nombre d'activités. Certaines de ces activités sont évaluables comme les examens et les exposés et d'autres ne le sont pas comme les cours.

1ÈRE SOLUTION

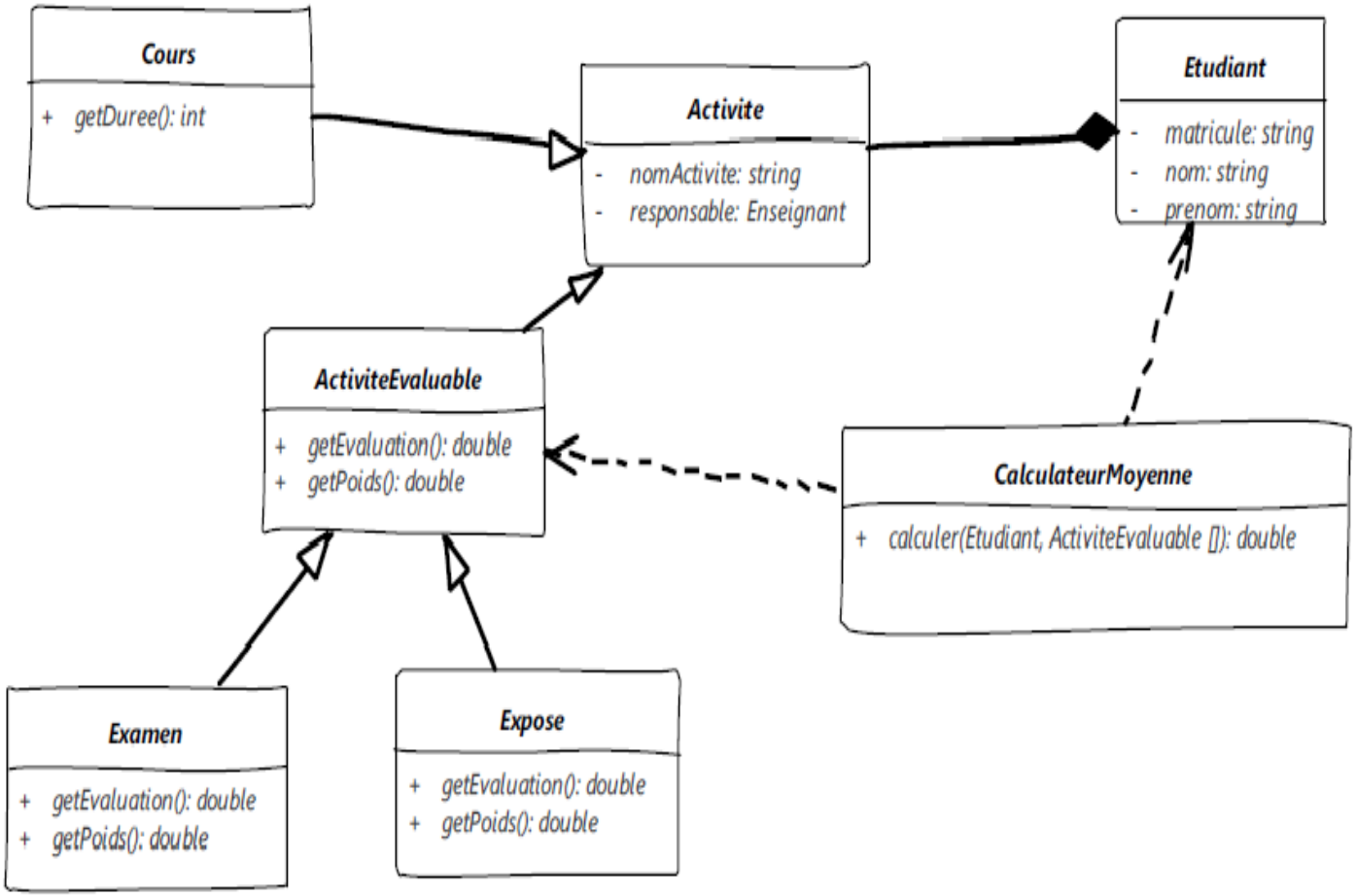
1ère solution



1ÈRE SOLUTION

- Conceptuellement, la 1ère solution n'est pas bonne.
- Les classes descendantes doivent redéfinir le mécanisme d'évaluation, chose qui n'est pas applicable sur les cours.

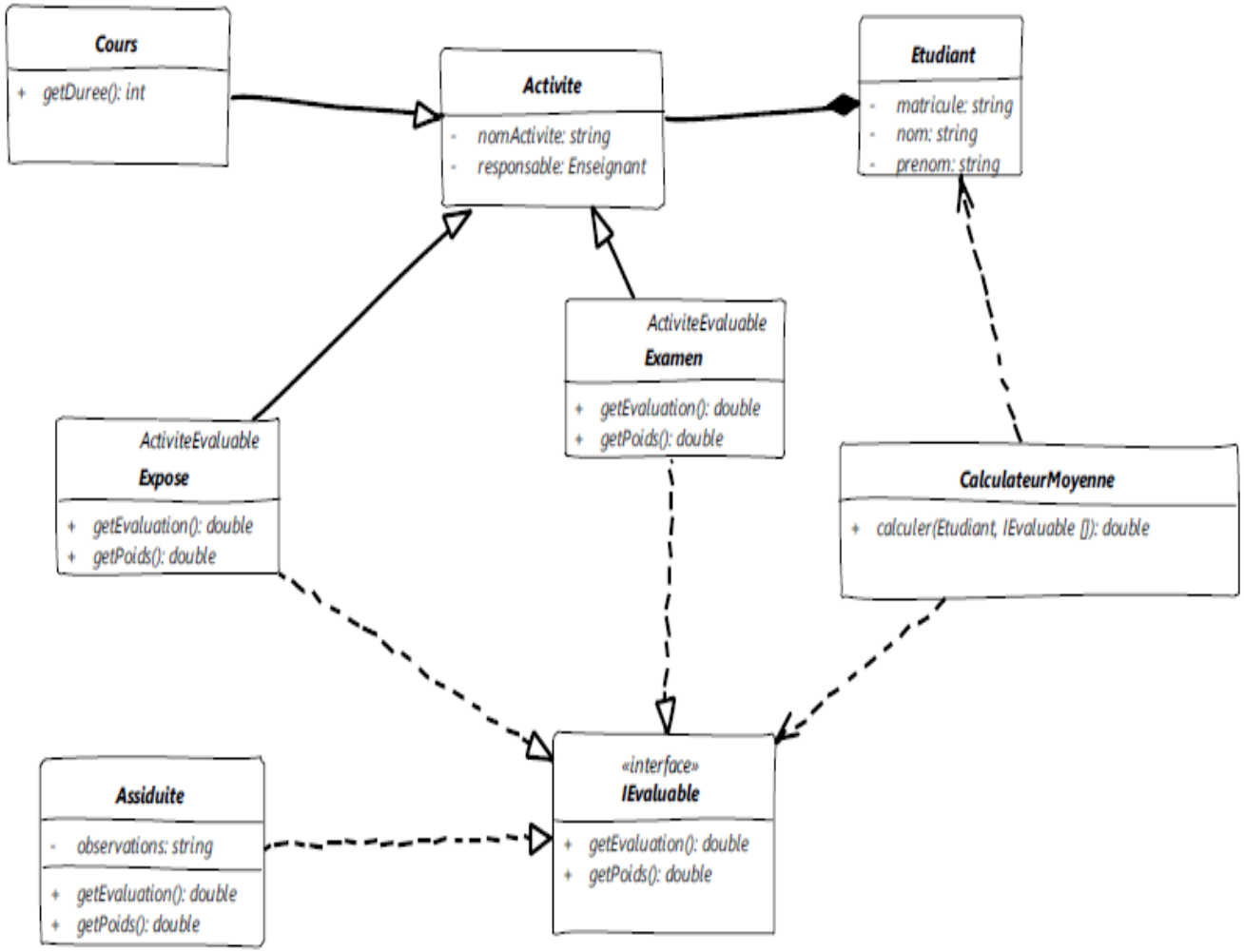
2ème solution



2ème solution

- La 2ème solution est meilleure que la 1ère. Supposons qu'on veuille intégrer l'assiduité de l'étudiant.
- Problème 1 : l'assiduité n'est pas une activité pédagogique sémantiquement.
- Problème 2 : Si la classe « Assiduité » hérite d'activité pédagogique, elle héritera de toutes les propriétés (par exemple « Responsable ») ce qui n'aura pas de sens.

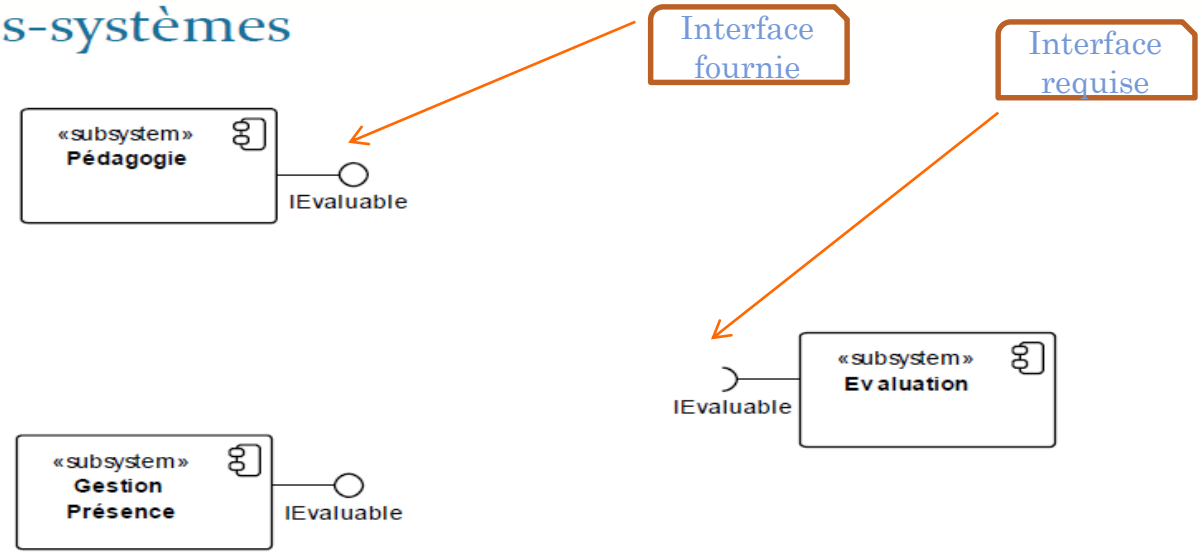
Solution 3


Cours
+ getDuree(): int
Activite
- nomActivite: string
- responsable: Enseignant
Etudiant
- matricule: string
- nom: string
- prenom: string
ActiviteEvaluable
Examen
+ getEvaluation(): double
+ getPoids(): double
ActiviteEvaluable
Expose
+ getEvaluation(): double
+ getPoids(): double
CalculateurMoyenne
+ calculer(Etudiant, IEvaluable []): double
«interface»
IEvaluable
+ getEvaluation(): double
+ getPoids(): double
Assiduite
- observations: string
+ getEvaluation(): double
+ getPoids(): double

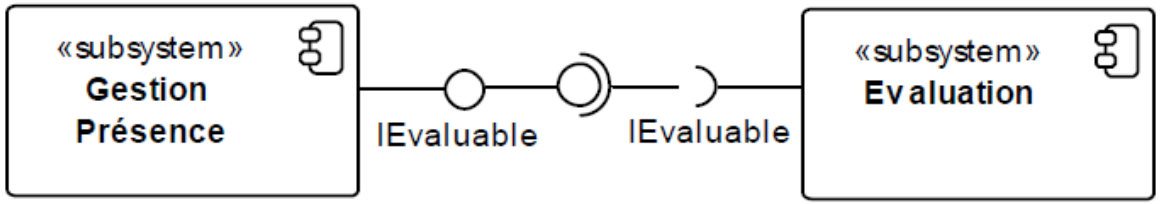
3ème solution

- La 3ème solution résout les problèmes précédents en ajoutant un très grand degré d'abstraction.
- En plus elle facilite la conception du système global, le sous-système d'évaluation a pour interface requise « IEvaluable » et d'autres sous systèmes comme la gestion des examens, ont cette interface comme fournie.

Sous-systèmes



Assemblage



Trouver les Interfaces

- Examiner chaque association, vérifier si une association doit être figée ou flexible.
- Examiner les opérations ayant un lien sémantique et voir s'il est bénéfique de les factoriser en une interface.
- Trouver les opérations qui se trouvent dans plusieurs classes et dont l'héritage ne convient pas pour les faire lier.
- Trouver les classes qui jouent le même rôle dans le système. Ce rôle peut être factorisé en une interface.
- Réfléchir en terme d'extensibilité, plus on utilise les interfaces, plus on est extensible.
- Identifier les interconnexions entre les composants et les sous-systèmes.