Qualité du Code

INTRODUCTION

Qualité logicielle : définition

ANSI : « ensemble des attributs et caractéristiques d'un produit ou d'un service qui portent sur sa capacité à satisfaire des besoins donnés »

- Difficile à définir, du fait du grand nombre d'indicateurs utilisables
- On peut d'abord distinguer
  - la **qualité du produit ou service** : analyse du **résultat**
  - la **qualité des process** : analyse de la **production** et des **moyens de productions**

Non Qualité

- L'absence d'analyse de qualité finit par produire son effet
- L' Expérience des dysfonctionnements permet de déterminer les **indicateurs de qualité**

INTRODUCTION

- Assurance Qualité (AQ)

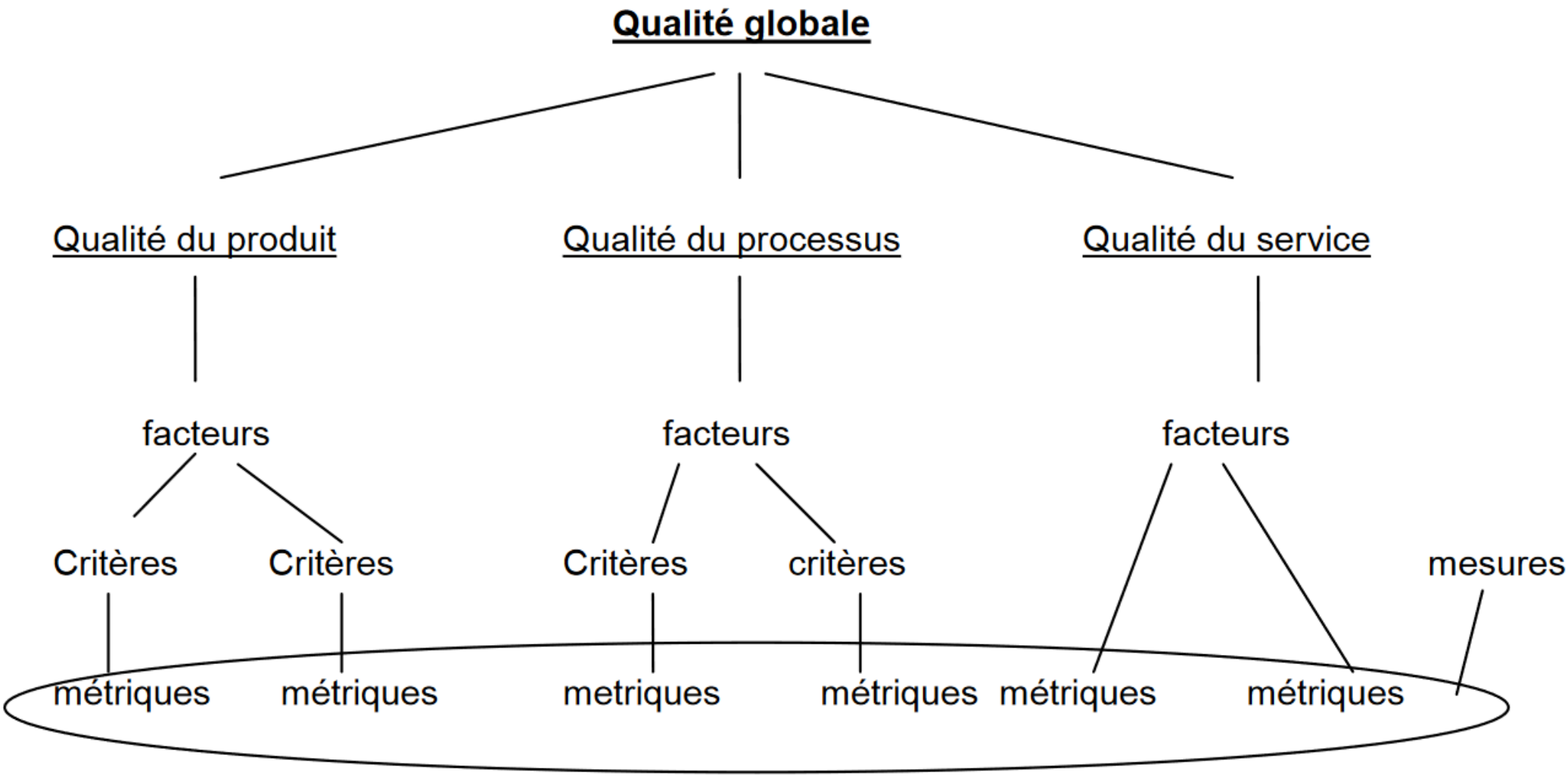
  Ensemble des mesures, procédures, méthodes utilisées dans le cadre du processus de développement du logiciel afin d'obtenir le niveau de qualité souhaitée.

  - Rédaction d'un Manuel d'Assurance Qualité : catalogue listant les procédures générales sur le périmètre global de **l'organisation**

- Plan de Qualité Logicielle (PQL)

  - Document spécifiant la démarche qualité relative à la mise en œuvre et au résultats attendus d'un **projet informatique**

INTRODUCTION



INTRODUCTION

- Facteurs de qualité logicielle : norme ISO/CEI 9126
  - **Capacité fonctionnelle** : réponse aux **besoins fonctionnels** exprimés
    - pertinence, exactitude, ergonomie, sécurité / intégrité des données et conformité.
  - **Fiabilité** : Capacité à maintenir un **niveau de service**
    - disponibilité, tolérance aux pannes, conditions de remise en service, maturité
  - **Facilité d'utilisation** : effort nécessaire pour utiliser le système
    - compréhension, apprentissage, exploitation (UX) ou encore pouvoir d'attraction.
  - **Efficacité / Efficience / Rendement**
    - rapport entre la propension à **satisfaire un besoin** et les **ressources consommées** pour ce faire.
  - **Maintenabilité** : capacité à **corriger ou faire évoluer** le système
    - lisibilité, flexibilité, stabilité et testabilité de la solution.
  - **Portabilité** : capacité à transférer le système sur une autre plateforme
    - facilité d'adaptation et d'installation, coexistance, interchangeabilité.

INTRODUCTION

- Ex : Calcul d'un indice de qualité
  - Liste de critères pondérés pour le facteur maintenabilité, pour une phase du processus de production

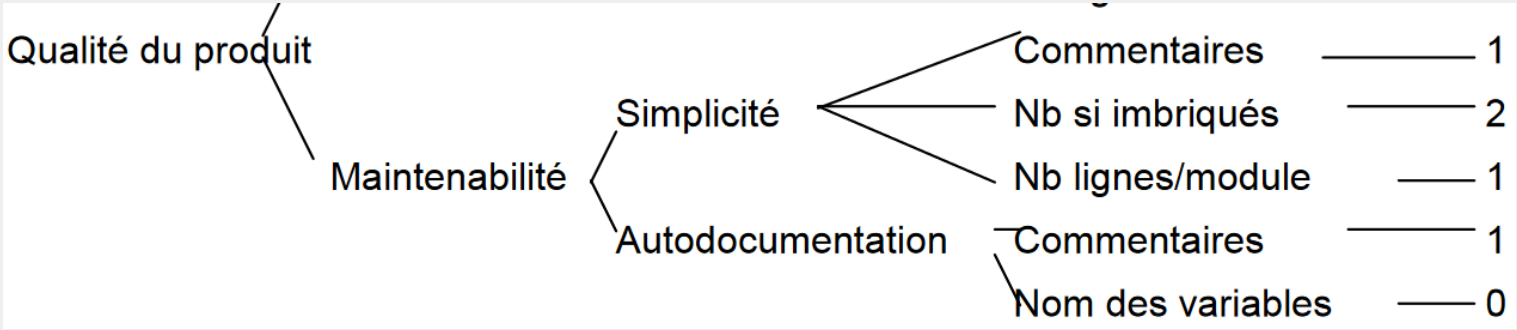
facteur	critère	phase	coefficient
maintenabilité	simplicité	4	0,7
maintenabilité	autodocumentation	4	0,3

  - Liste de métriques pondérées pour un critère et une phase

critère	métrique	phase	coefficient
autodocumentation	commentaires	4	0,5
autodocumentation	Nom des variables	4	0,4

INTRODUCTION

- Ex : Calcul d'un indice de qualité
  - Mesures



  - Calculs :

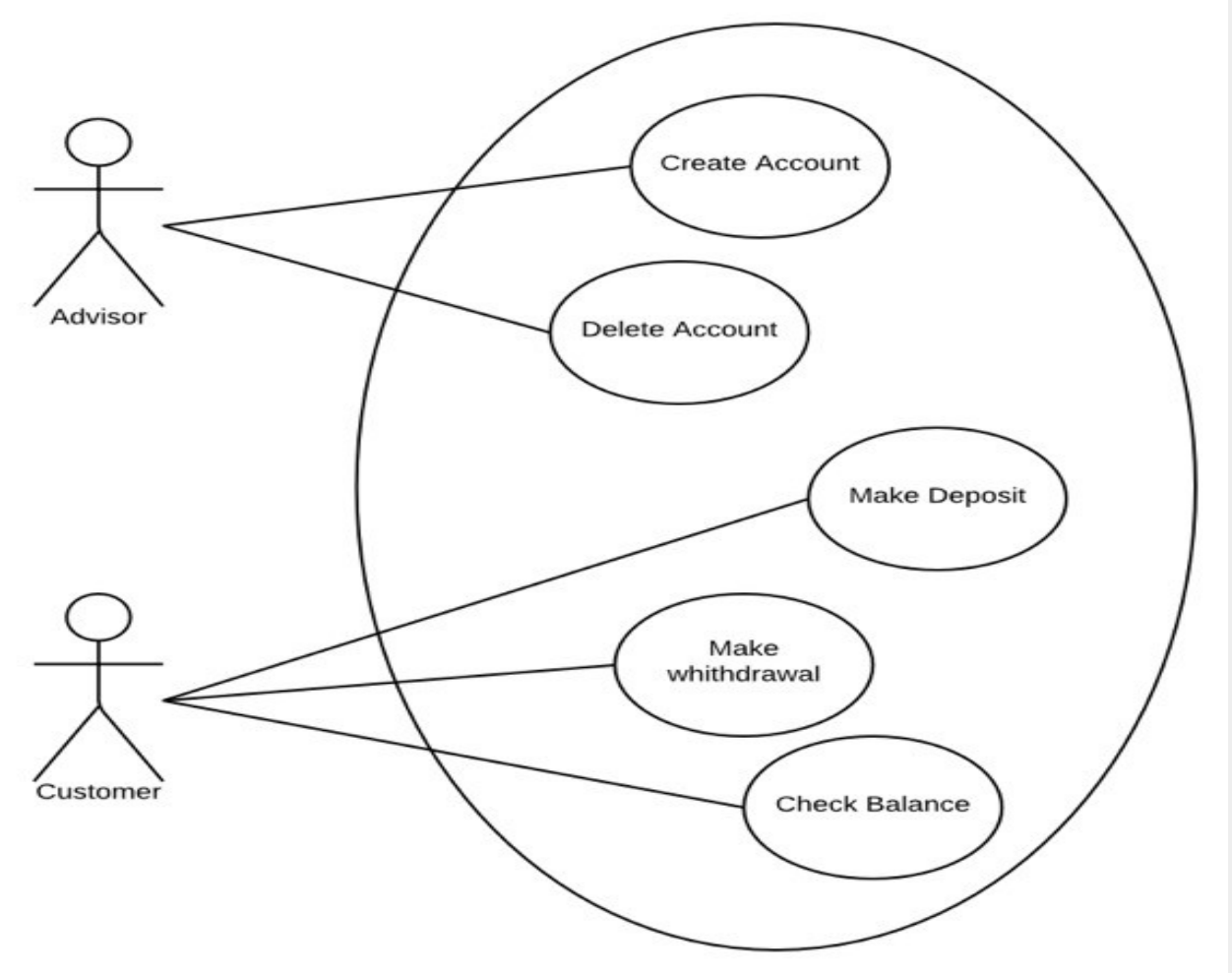
```
# sommes pour les critères
AD = 1 * 0,5 + 0 * 0,4
S = …
 # sommes pour les facteurs
M = 0,7 * S + 0,3 * AD
# Indice Qualité
IQ = M * … + ...
```

CONCEPTS OBJET

- UML : « Unified Modeling Language »
  - Langage de **modélisation graphique standardisé** : 14 types de diagrammes en 2.5
  - Répartis sur 3 vues :
    - **Fonctionnelle :** acteurs et besoins
      - diagrammes de cas d'utilisation
    - **Statique :** les structures de la solution et leur relations de dépendance
      - diagrammes de classe
      - diagrammes d'objet
    - **Dynamique :** comportement de ces structures dans le temps
      - diagrammes de séquence

CONCEPTS OBJET

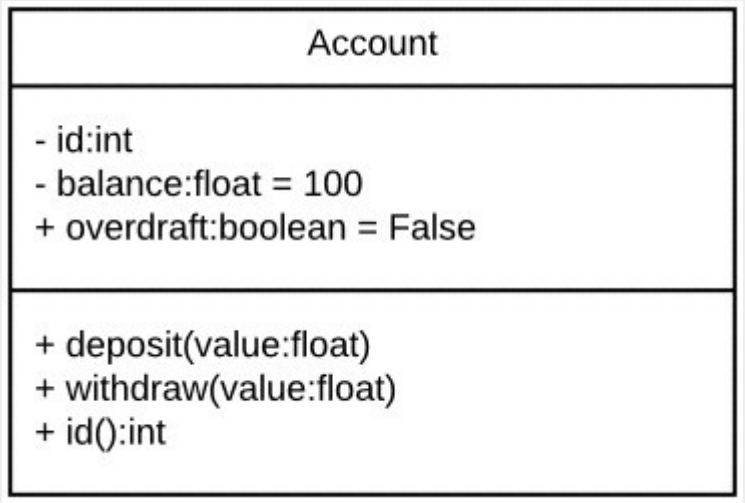
- UML : Cas d’utilisations



CONCEPTS OBJET

- UML : Classes et Objets

Classe (définition d'un type)



Attributs

privés

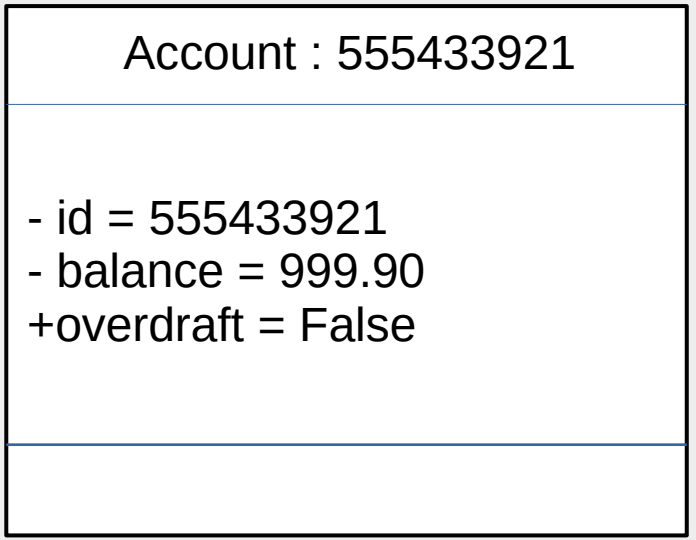
publiques

Méthodes

privés

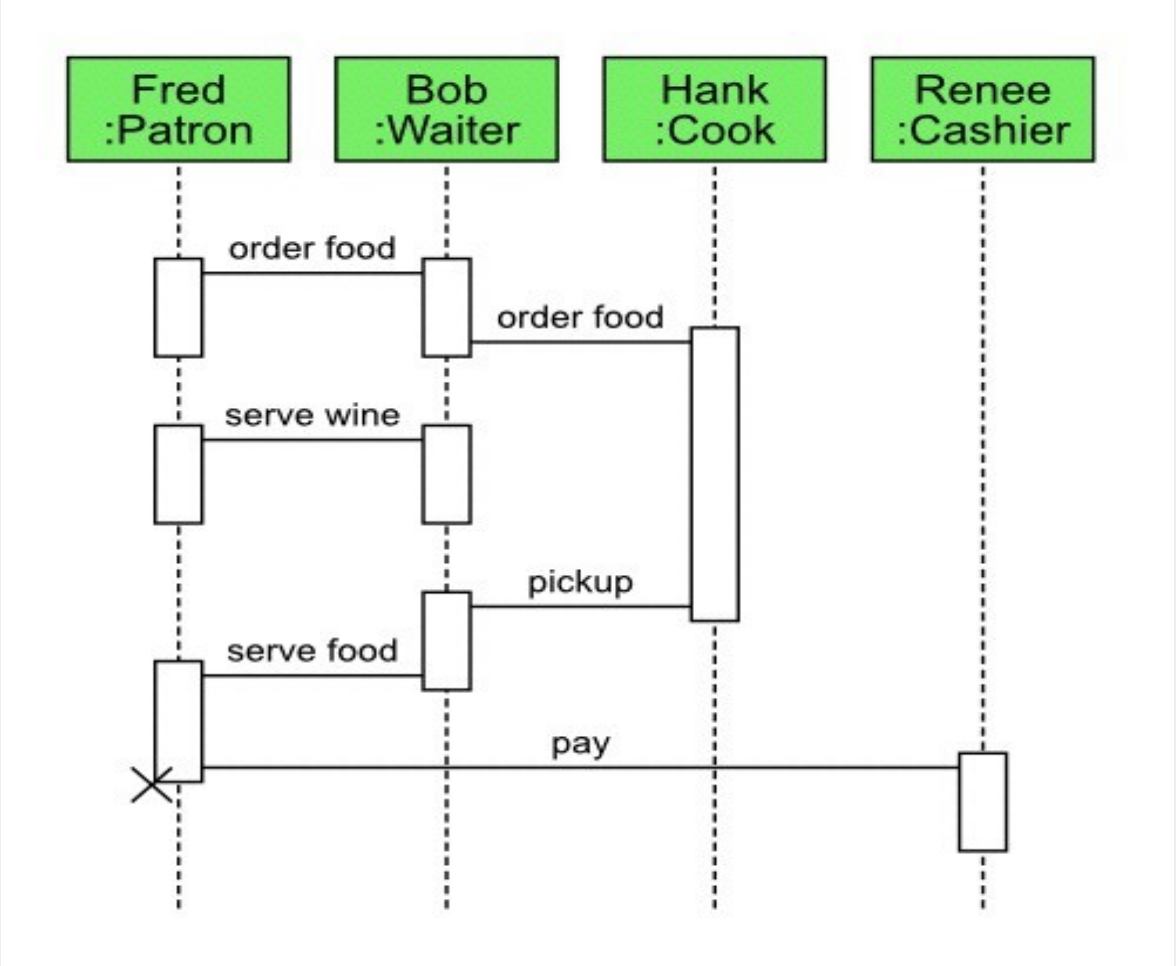
publiques

Objet ou Instance (donnée du type)



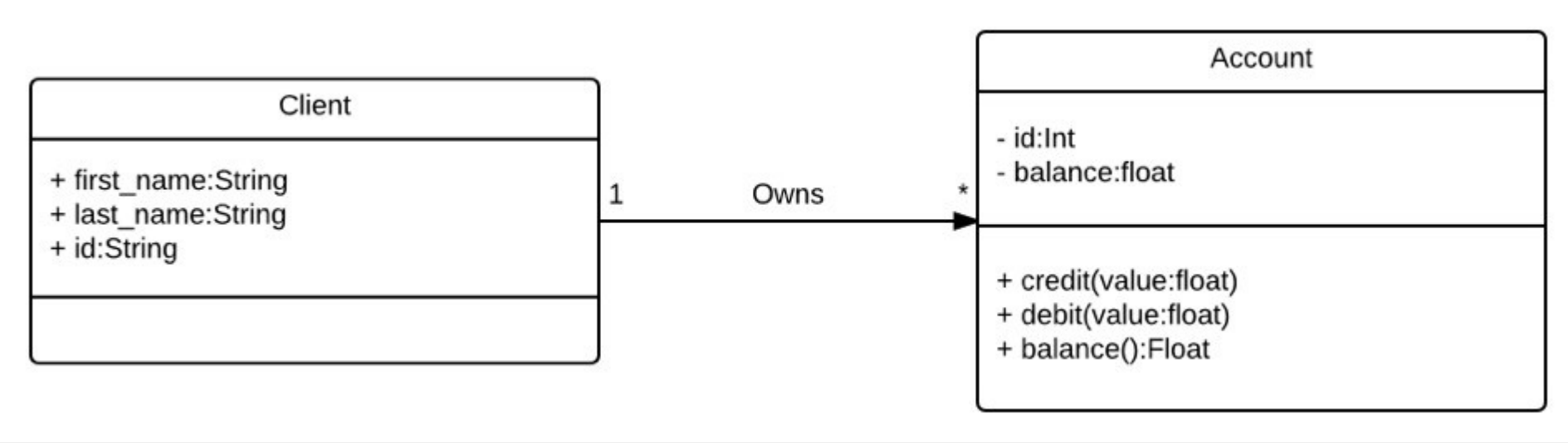
CONCEPTS OBJET

- UML : diagramme de séquence



CONCEPTS OBJET

- UML : Association entre classes

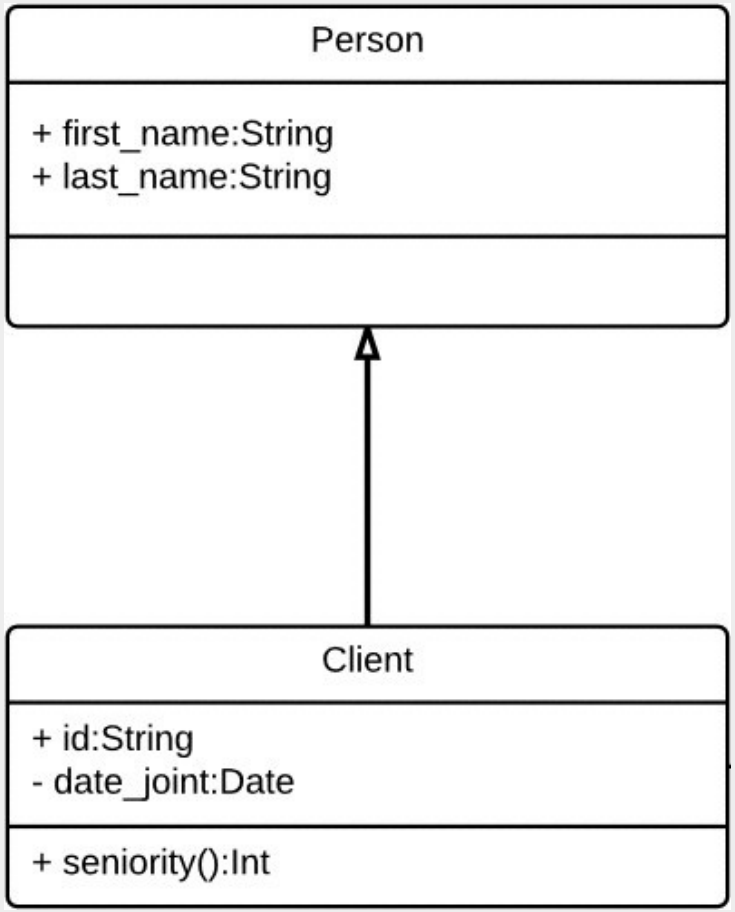


- Un client peut posséder n comptes : **« * »** ou **« min..max »**
- Un compte n'est possédé que par 1 client
- **l'aggrégation** est une association de type **« aggrégat <=> composant »** 
- **La composition** est une aggrégation forte **<=> le composant est essentiel et n'existe pas en dehors de l'aggrégat** 

CONCEPTS OBJET

UML : Héritage entre classes

- Classe mère :
  - concept + abstrait mais instanciable …
  - … ou complètement **abstrait => non instantiable**

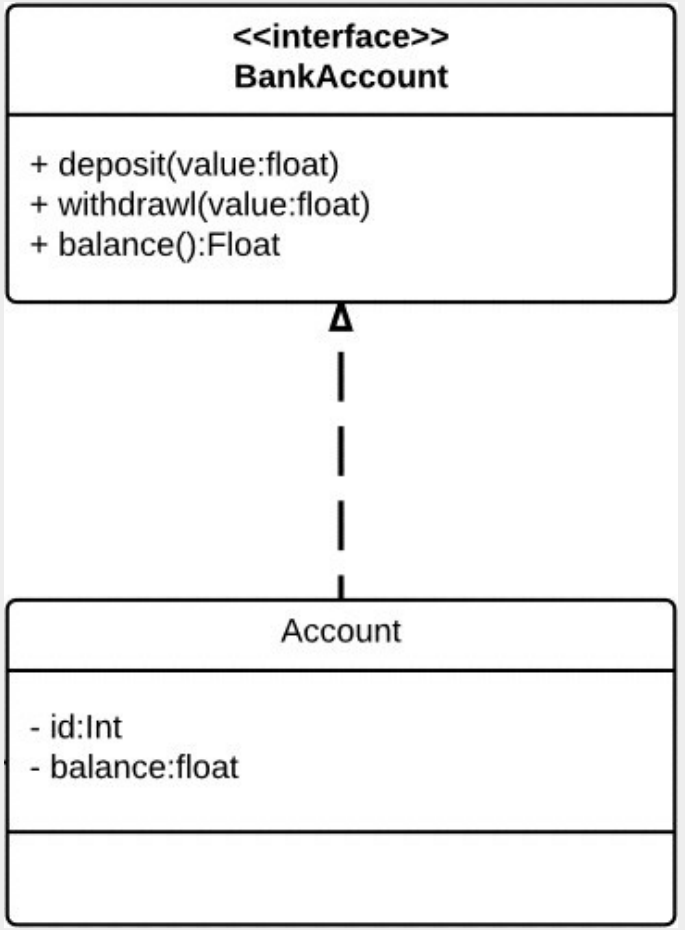


- Classe fille :
  - **accède** aux attributs / méthodes **publiques** de la classe mère
  - accède aux attributs / méthodes **protégés** //
  - **spécifie** ses propres attributs / méthodes
  - et peut **surcharger** les méthodes parentes existantes

UML : Interfaces

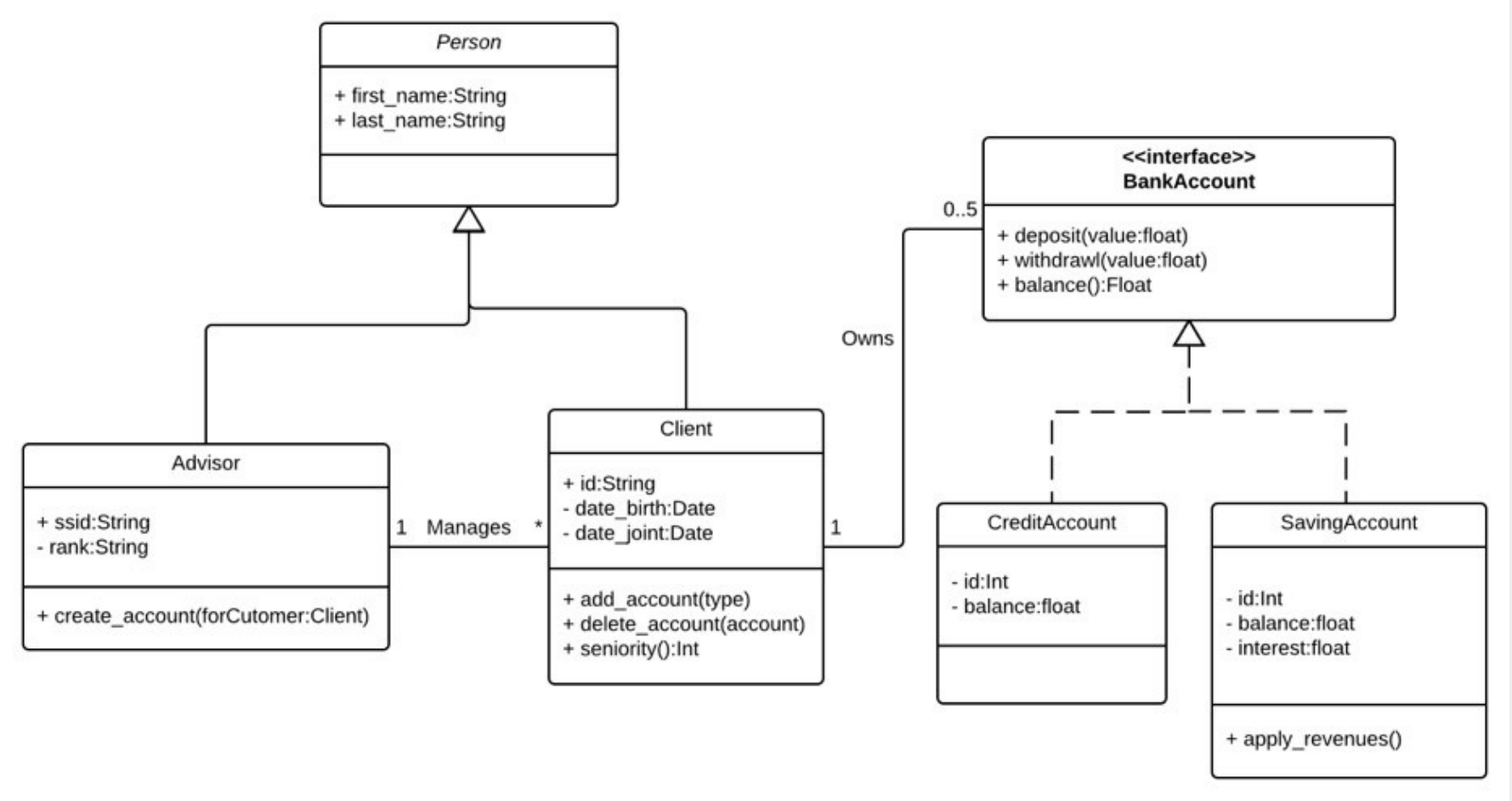
- Définit un ensemble de **signatures** de méthodes
  - propriétés : publique / privé, statique, finale
  - nom
  - paramères et valeurs de retour

- Toute classe utilisant une interface doit **implémenter** toutes les méthodes définies dans cette dernière



CONCEPTS OBJET

- UML : exemple complet



Génie logiciel

- Principes de codage : SOLID
  - **S** : Principe de résponsabilité unique « Single Responsibility Principle (SRP) »
    - Une classe ne doit avoir qu'une seule et **unique fonction** (implémentée sur plusieurs méthodes)
  - **O** : Principe d'ouverture / fermeture « Open / Closed Principle »
    - Les entités doivent être **ouvertes à l'extension** et **fermées à la modification**
    - éviter de sérier les traitements dans des **cascades de if ou case** dépendant du type de donnée
    - **préférer les interfaces et types hérités**
  - **L** : Principe de substitution de Barbara Liskov : « Liskov Substitution Principle (LSB)»
    - On doit pouvoir **remplacer des objets par leur instances héritées** sans altérer le bon fonctionnement du programme
    - même signatures, nb de paramètres, valeur de retour, et exceptions (à un héritage près)
  - **I** : Principe de ségrégation de l'interface : « Interface Segregation Principle (ISP) »
    - Aucun client ne devrait être forcé d'implémenter des méthodes / fonctions qu'il n'utilise pas
    - Il vaut mieux faire plusieurs petites interfaces qu'une seule grande
  - **D** : Principe d'inversion de dépendance : « Dependency Inversion Principle (DIP) »
    - Une classe doit dépendre de son abstraction, pas de son implémentation
    - on évite de passer des objets en paramètre lorsqu'une interface est disponible

Principes de codage : Autres Principes

- **KISS** ; Keep It Simple Stupid !
  - créer des flux utilisateurs directs
  - ne pas jargonner
  - montrer au plus tôt et clairement les résultats

- **DRY** ; Don’t Repeat Yourself !
  - Tout élément d’information (classe, méthode documentation, test …) ne doit se trouver qu’à un endroit
  - Permet de diminuer la **dette technique**

- **YAGNI** : You Aren’t Gonna Need It ! (Extrem Programming)
  - ne créer une fonctionnalité / classe / module … que lorsque sa nécessité apparaît clairement

QUALITE DU CODE

- Critères généraux de qualité d'un code
  - **Capacité fonctionnelle** : (Functionality)
    - tests d'intégrations / fonctionnels / recette
  - **Fiabilité** : (Reliability)
  - **Facilité d'utilisation** : (Usability)
    - cohérence des éléments de code (responsabilité)
    - documentation
  - **Efficacité / Efficience / Rendement :** (Efficiency)
  - **Maintenabilité** : (Maintainability)
    - lisibilité / respect des conventions
    - cohérence, stabilité, non répétition
  - **Portabilité** : (Portability)

QUALITE DU CODE

- Conventions de codage
  - Ensembles de règles plus ou moins arbitraires relatives à un langage, une organistion
  - **Nomenclatures** de variables : PascalCase, camelCase, snake_case, kebab-case, ALL_CAPS
  - Règles d'**écritures** : saut de lignes, indentations, espaces, marge à droite
  - Respects de Standards :
    - documentation, commentaires, TODOs
    - SOLID ?
    - Design Patterns
  - Règles empiriques : pas plus de
    - 80 caractères par ligne
    - 25 lignes par méthode

QUALITE DU CODE

- **Métriques triviales** de qualité du code
  - Nombre total de **lignes de codes**, de **classes**, de **méthodes**
    - nb de ligne total d'une source ou **nb de lignes exécutables**
  - Nb lignes / classe
  - Nb méthodes / classe
  - Ratio Nb lignes / Nb méthodes
  - Ratio Nb lignes / Nb classes
  - Ratio Nb lignes de commentaires / Nb lignes (densité de commentaires)

QUALITE DU CODE

- **Métriques standard** de qualité du code
  - Indice de **spécialisation** d'une classe
  - Indice d'**instabilité** d'une librairie
  - Coefficient d'**abstraction** d'une librairie
  - Distance de la **bonne conception** d'une librairie
  - **Complexité cyclomatique** d'une méthode
  - **Taux de couverture** d'une méthode par les tests

QUALITE DU CODE

- Indice de spécialisation d'une classe
  - Formule : $\frac{NORM \times DIT}{NOM}$ avec **NORM** : nb de méthodes surchargées,

    DIT : pronfondeur d'héritage, **NOM** : nombre de méthodes
  - Valeurs **> 1,5 => refactorisation** à envisager
  - Sert à distinguer des classes spécialisant trop ses méthodes pour légitimer un héritage
  - Priviliégier alors l'utilisation **d'interfaces**

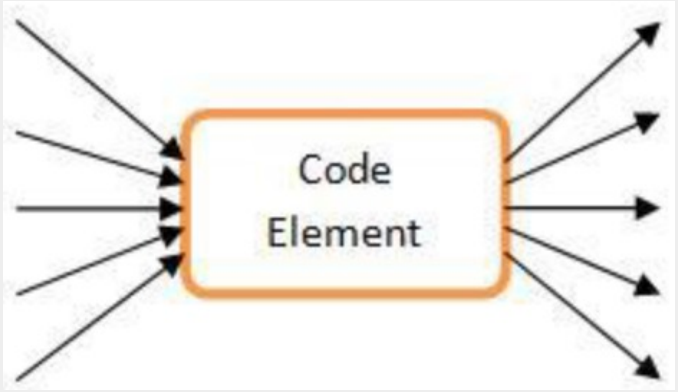
QUALITE DU CODE

- Indice d'instabilité d'une librairie
  - Formule : $\frac{Ce}{Ca+Ce}$ avec **Ca** : Couplage afférent, nb. références à la lib dans le code externe (imports)

    Ce : Couplage efférent, nb. références externes dans la lib (imports)

  - Ca peut mesurer la responsabilité d'une classe, Ce la compléxité
  - Valeur entre 0 et 1 : **~0** => librairie stable, Valeur **~1** => librairie instable

C. afférent



C. efférent

QUALITE DU CODE

- Coefficient d'abstraction d'une librairie
  - Formule : $\frac{I}{T}$ avec **I** : nb d'interfaces et de classes abstraites
    T : nb total de classes
  - Valeur entre 0 et 1 : pas immédiatement signifiant utilisé seul
- Distance à la bonne conception
  - Formule : | ABST + INST - 1 | avec **ABST** : coefficient d'abstraction
    INST : coefficient d'instabilité
  - Valeur entre 0 et 1 : **~0** => bon design, **> 0,5** => probablement à revoir
  - Une lib instable avec peu d'abstraction peut être bien conçue !

QUALITE DU CODE

- Complexité cyclomatique d'une méthode
  - Formule : Nb Branches + 1 avec **Branches** : structures de contrôle du langage (**if, case, while, for**)
  - Valeur > 0 **> 30** => à refactoriser
  - Le Nb de branches d'une méthode permet de déduire le **nombre de chemins possibles** dans celle ci
  - Cet ensemble de chemins possibles doit correspondre aux **cas d'utilisations** de la méthode
  - La **couverture de code** (cf infra) doit vérifier que tous les cas d'utilisations sont testés

QUALITE DU CODE

- Couverture**s** du code par les tests

ISTQB : « Degré, exprimé en pourcentage, selon lequel un **élément de couverture** spécifié a été exécuté lors d'une suite de test »

- Couverture par les méthodes : (Nb méthodes traversées au – 1x) / Nb méthodes du code
- // par les lignes : (Nb lignes traversées au – 1x) / Nb lignes du code **(le + utilisé)**
- // par les **branches :** (Nb branches traversées au – 1x) / Nb branches du code
- // par les chemins possibles : en pratique impossible à opérer
- Autres éléments liés aux **techniques de testing** (cf infra)
  - par les partittions d'équivalence
  - par les valeurs limites
  - par les tables de décisions
- Autres types de couvertures : couverture des risques, des personnes, des machines, des langues...