

Prog avancé q2

🕒 Date de création	@15 février 2024 13:37
--------------------	------------------------

UML → unified modeling Language

▼ Rappel

orienté objet → organisation d'un logiciel sous la forme d'une collection d'objets indépendants incorporant structure de données et comportement

Objet

- Entité discrète et distinguable (noah diff de jean), concrète ou abstraite
- Identifiant unique (instancié → crée un objet à partir d'une classe)
- Deux objets sont donc distincts, même s'ils ont des valeurs d'attributs identiques

Classification : regroupement des objets ayant la même structure de données (attributs) et même comportement (opérations)

Classe : abstraction décrivant un ensemble d'objets potentiellement infini

Instance d'une classe : objet créé à partir d'une classe

Héritage : partage de propriétés entre classes sur la base d'une relation hiérarchique

- Super-classe (classe mère)
- Sous-classe (classe fille)

Polymorphisme : possibilité de comportements différents d'une même opération dans différentes classes

opération : action exécutée par un objet ou transformation subie par un objet

Spécification initiale : collaboration entre les analystes métier et les utilisateurs pour la genèse de l'application

Analyse : Étude et reformulation des besoins : collaboration avec le client pour comprendre le problème

- modèle d'analyse : abstraction concise et précise de l'objectif du système à développer
- Modèle de domaine : description des objets du monde réel manipulés par le système
- Modèle de l'application : parties du système visible par l'utilisateur

Conception des classes : concentration sur les structures de données et algorithmes de chaque classe

Abstraction : concentration sur le comportement avant d'implémenter

Encapsulation : Masquage de l'information

Regroupement des données et du comportement

Partage

Mise en évidence de la nature intrinsèque des objets

Les bases de L'UML

- Esperanto (Ariba)
- Dessins
- Histoire

But et polyvalence de L'UML

- Pas juste le plan

- Quoi mais aussi comment !
- Outils de communication avec des techniciens ou autres

UML et POO sont "IN LOVE"

- Concepts POO sont en UML visuel
- Conception facilitée
- Compréhension
- Communication de système complexe

Plus qu'un langage, une méthodologie

- Holistique
- Structure et comportement

Fondement de L'UML :

- Modélisation basée sur les objets
- Abstraction et encapsulation

Quand l'utiliser ?

- Dès qu'on a l'idée jusqu'à la réalisation

C'est un langage universel

Qui est adaptable

Les 2 grandes catégories de diagramme

Complexité

- Diagramme de comportement
- Diagramme de structure
- Très vite compliqué si mal réfléchi
- Beaucoup de possibilités

- Diagramme de Classe : représente les classes et leurs relations, soulignant la structure et la conception du système
- Diagramme D'objet : montre des instances de classes a un moment donné, illustrant la situation réelle de l'application
- Diagramme de paquet : groupe les elements associes tels que les classes dans des paquet facilitant l'organisation du modèle
- Diagramme de composant
- Diagramme de déploiement
- Diagramme de séquence : va montrer la temporalité (quelle action quelle moment)
- Système : Constitué de différente partie qui vont essayer de réaliser un but bien défini
- Modèle : Le modèle d'un système représente le point de vue du développeur pendant la phase d'analyse du projet et à ce titre, il n'existe pas vraiment de modèle unique
- Structurel ou comportemental
 - Les diagrammes de structure sont statiques, ils permettent de définir l'architecture du système, c'est en quelque sorte le plan d'un logiciel ou d'un système
 - Les diagrammes de comportement sont dynamiques. Ils décrivent le comportement du système et son fonctionnement dans le temps.

SI → système d'information

Analyse processus métier (qu'est ce que la personne doit faire pour que ca fonctionne) → Architecture fonctionnelle (verbaliser les actions que la personne devra faire) → architecture applicative (a quoi doit ressembler mon application) → architecture technique (étape 1 mais plus poussé)

Cycle vie d'un projet

- Analyse des besoin et faisabilité
- Spécificité ou conception générale
- Conception détaillé
- Codage (implémentation ou programmation)
- Test unitaires
- Intégration
- **MENTAL BREAKDOWN**
- Qualification (ou recette)
- Documentation
- **MENTAL BREAKDOWN**
- Mise en production
- Maintenance

Réalisation d'un diagramme

- Compréhension du besoin client
- Limiter l'afflux d'informations
- Dialogue entre tous
- Fonctionnnalités

C'est un diagramme comportemental qui représente toute les fo
du point de vue utilisateur

- Représenter les fonctions ou les services fournis par le sys
- Chaque cas d'utilisation est une séquence d'actions que le s
- un résultat utile à un acteur
- Habituellement décrits par un verbe et un nom (Gérer command

1ère étape diagramme de cas

- définir périmètre du système
- ressortir les ACTEURS
- Acteur = type stéréotypé qui représente un rôle joué par une personne ou une chose qui interagit avec le système

- Une même personne physique = plusieurs rôles

Descriptif d'un acteur

- Représentent les utilisateurs ou les entités externes qui interagissent avec le système
- Peuvent être des utilisateurs, entité externe, autre système, entité organisationnelles
- Acteur principal : personne qui utilise fonctions principales du système
- Acteur secondaire : ce sont les personnes qui effectuent des tâches dites secondaires : tâches de maintenance, tâches administratives, etc.
- Matériel externe : il s'agit des dispositifs matériels qui doivent être utilisés. Ils font partie intégrante du domaine de l'application

Acteur est représenté par un stickman avec un nom

/!\ Acteur pas genre Eric et Ramzy, enfin ça peut mais c'est pas ça qu'on veut dire par là chef

UML

- Frontière
- Nom
- Ca(S)
- Acteur(s)
- Liaison(s)

0..1 → zéro ou une fois

1 → une et une seule fois

* → 0 à plusieurs fois

1..* → 1 a plusieurs fois

M .. N → de M a N fois

Include → c'est inclut dans les toutes différents actions possible et donc si celles ci n'existe plus, elle non plus. Elle serra obligatoirement activer si une de ces actions sont effectuer (truc par lequel on va obligatoirement passer)

L'extension (capilaire pour Mathias le bientôt chauve)

- Enrichir un cas
- est optionnel
- application lors de chaque scénario

permet de développer plus une option en gros mais c'est pas obligé d'y passer

Héritage → flèche vide de l'enfant vers le parent

Identifier les acteur → qui utilisera l'app

Definir les cas d'utilisation

Etablir les relation (inclusion, extension, ...)

Vérifier et valider

Simplicité → gardez le diagramme simple et compréhensible

Focaliser sur l'utilisateur → Pensez du point de vue utilisateur

Les cas d'utilisation servent à :

1. **Définir ce que le système doit faire** en établissant clairement les fonctionnalités attendues par les utilisateurs.
2. **Confirmer que le système remplisse ces fonctions** une fois qu'il est prêt à être livré.
3. **Identifier les limites du système**, c'est-à-dire ce qu'il fait et ne fait pas.

4. **Aider à rédiger la documentation du système**, qui guide les utilisateurs et les développeurs.

5. **Développer des tests** pour s'assurer que le système fonctionne comme prévu.

- Reste proche du langage naturel → bon pour aider le client
 - Ajout d'un lexique (tableau est un plus)
-
- Importance des diagrammes de classe
 - Rôle dans le développement logiciel
 - Visualisation des concepts de POO
 - Utilité des diagrammes de classe
 - Communication entre développeurs et parties prenantes (client, ...)
 - Planification et conception

Diagramme de classe

- Modélisation de la structure du code
- Illustration de la POO
- Relation entre les classes

En UML, un objet est toujours dynamique (Un livre va se fermer tout seul, ça serra sa méthode, elle serra certes activer par qqn mais c'est SA méthode)

Une classe c'est :

- Nom (au singulier)
- Attribut
- Méthode

le + signifie que c'est public

Le - signifie que c'est privé

Le # signifie que c'est protégé (meilleur)

propriété ordered → lorsqu'une variable peut contenir plusieurs valeurs qui seront ordonnées

Unique → Lorsqu'une variable peut contenir plusieurs valeur faire en sorte qu'elles soit toutes différentes

No Unique → l'inverse

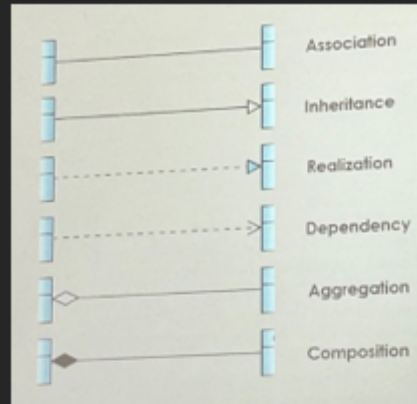
OCL → Le fait de rajouter des contraintes/caractéristique sur notre diagramme

L'agrégation : une flèche entre 2 cadre (Prof → université)

Losange vide → Ca se compose de ... mais ca ne disparaîtra pas si l'autre n'est plus là

Losange plein → leur vie sont lié

Les liaisons entre les classes



Association : C'est quand deux classes sont liées d'une manière quelconque. Par exemple, une classe "Étudiant" peut être liée à une classe "École" parce que les étudiants vont à l'école.

Héritage : C'est quand une classe obtient des choses d'une autre classe. Par exemple, une classe "Chien" peut obtenir des caractéristiques d'une classe "Animal".

Réalisation : C'est quand une classe fait ce que dit une autre classe. Par exemple, une classe "Voiture" peut faire ce que dit une interface "Conductible".

Dépendance : C'est quand une classe a besoin d'une autre classe pour faire quelque chose. Par exemple, une classe "Voiture" peut avoir besoin d'une classe "Moteur" pour fonctionner.

Agrégation : C'est quand une classe est composée de plusieurs parties. Par exemple, une classe "Équipe" peut avoir plusieurs "Joueurs".

- A: Composé | B: Composant



Une forme spéciale d'association qui représente une relation « tout parti »

Composition: C'est quand une classe possède des parties qui ne peuvent pas exister sans elle. Par exemple, une classe "Ordinateur" possède des parties comme le processeur et la mémoire, qui sont nécessaires pour qu'il fonctionne.

- Caractéristiques: Relation d'inclusion, durée de vie, encapsulation, réutilisabilité, flexibilité

et technologies

Critère	Classe Abstraite	Classe Concrète
Instanciation	Ne peut pas être instanciée directement.	Peut être instanciée pour créer des objets.
Utilisation	Sert de modèle ou de base pour d'autres classes.	Utilisée pour créer des objets spécifiques dans le système.
Méthodes	Peut contenir des méthodes abstraites (sans implémentation) et des méthodes avec implémentation.	Doit fournir une implémentation pour toutes ses méthodes, y compris celles héritées.
Objectif	Définir une interface commune et un comportement partagé pour les classes dérivées.	Implémenter cette interface commune en fournissant des comportements spécifiques.
Héritage	Peut être héritée par d'autres classes abstraites ou concrètes.	Peut hériter d'une classe abstraite mais doit implémenter toutes les méthodes abstraites.
Exemple d'utilisation	Une classe Véhicule avec une méthode abstraite démarrer().	Une classe Voiture qui implémente la méthode démarrer() de Véhicule.



Réalisé par Erwin Desmet
06-03-24 ● ● ● ●
136

Classe abstraite : idéale pour définir un cadre général que plusieurs classes dérivées peuvent suivre.

Classe concrète : Cruciale pour la création d'objets réels dans le système.

Polymorphisme

- Classes (souvent abstraite) = ensemble d'objets différents
 - Instance de sous-classes différentes
 - Appel méthode même nom
- Interface commune
- implémentation multiple
- utilisation dynamique

Stéréotype = spécialiser un concept

Constitution = mot clé entre guillemets

Diagramme séquence

Un diagramme de séquence montre comment différentes parties d'un système fonctionnent dans une "séquence" pour accomplir quelque chose

Flèche pointillée = flèche de retour

Souvent sur plusieurs diagramme

Représente les interactions entre un groupe d'objet

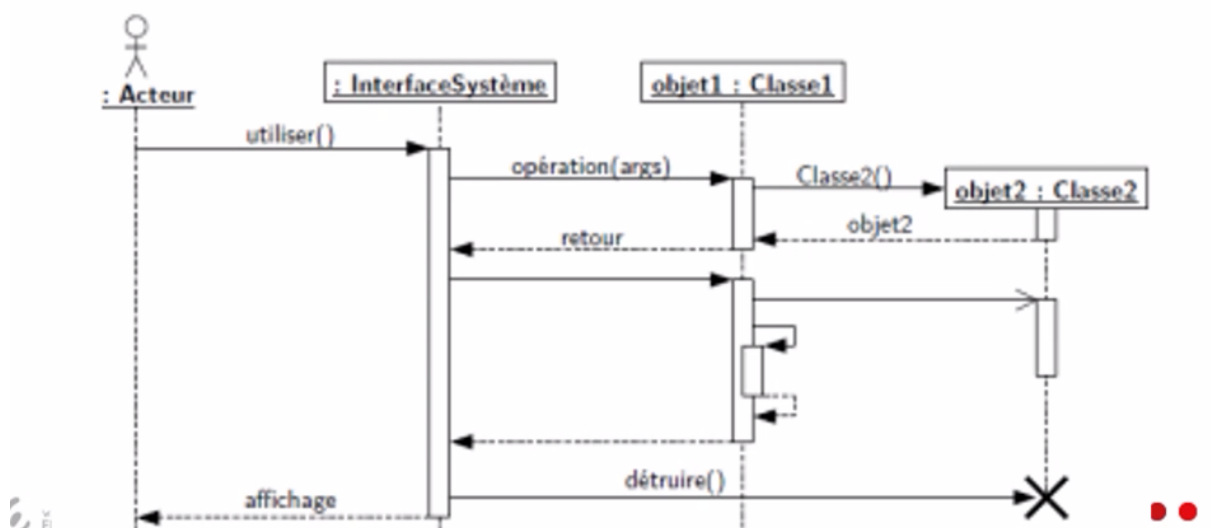
pour interagir entre eux, les objets s'envoient des messages. Lors de la réception d'un message, un objet devient actif et exécute la méthode de même nom. Un envoi de message est donc un appel de méthode



Diagramme de cas d'utilisation



Diagramme de classes



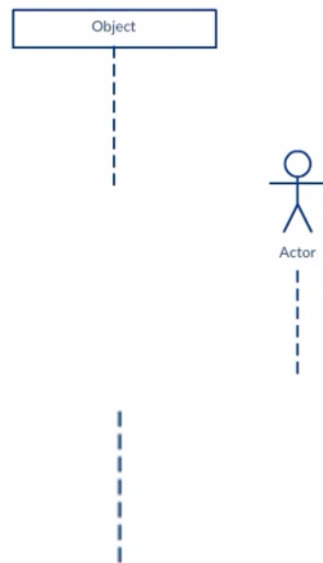
Celui en dessous = Diagramme séquence

- Est chronologique
- De haut en bas
- Chaque objet a sa propre ligne de vie

- Objet

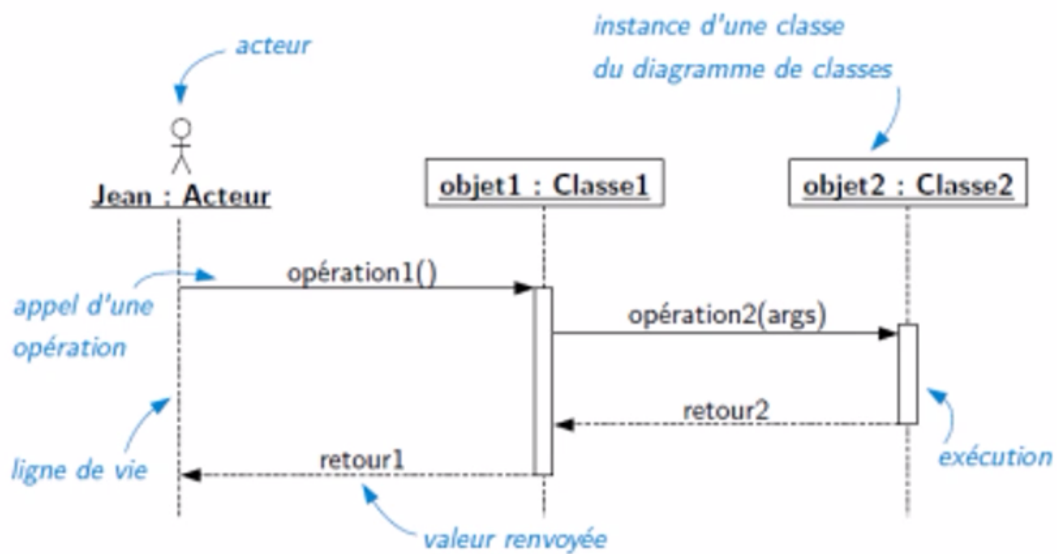
- Acteurs

- Lignes de vie



Les messages

- Décrit l'interaction entre les objets
- Représenté par des flèches
- Doit être fait dans l'ordre
- Type de messages
 - Message d'appel (demande (ligne pleine))
 - Message de retour (retour (ligne pointillé))
 - Message personnel
 - Message récursif (s'appelle en boucle)
 - Message créateur de "cycle" (création d'un objet grâce a un autre objet)
 - Message destructeur de "cycle" (détruire un objet en gros)
 - A pas confondre avec les messages hot que m'envoie Tom
- Durée d'un message (représente le passage du temps lors d'un appel)
- Des notes (important pour bien comprendre **utiliser Visual Paradigm**)



Fragment combinés : permettant de modéliser des décisions, des boucles, parallélismes (alt, loop)

<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> HEH^{be} Sciences et technologies </div> </div>	
Opérateur	Type de fragment
Break	Le fragment break permet de modéliser une sortie prématurée d'une boucle ou d'un autre fragment combiné si une certaine condition est remplie. Il fonctionne comme l'instruction "break" dans de nombreux langages de programmation.
Critical	Le fragment critical est utilisé pour indiquer une section critique de l'interaction où les messages sont traités de manière atomique. Cela garantit qu'aucune autre interaction ne peut intervenir pendant l'exécution de cette séquence.
Neg (Négatif)	Le fragment neg est utilisé pour représenter des scénarios qui ne devraient pas se produire. Cela peut être utile pour tester ou valider qu'un certain ensemble d'interactions est incorrect ou indésirable.
Assert	Le fragment assert est utilisé pour indiquer que les interactions dans le fragment doivent absolument se produire telles qu'elles sont spécifiées. Cela peut être utilisé pour des vérifications ou des affirmations strictes sur le comportement du système.
strict	Le fragment strict indique que les interactions à l'intérieur doivent se produire exactement dans l'ordre spécifié, sans aucune déviation ou parallélisme.

Mettre le Rôle et la classe et nom dans l'entête

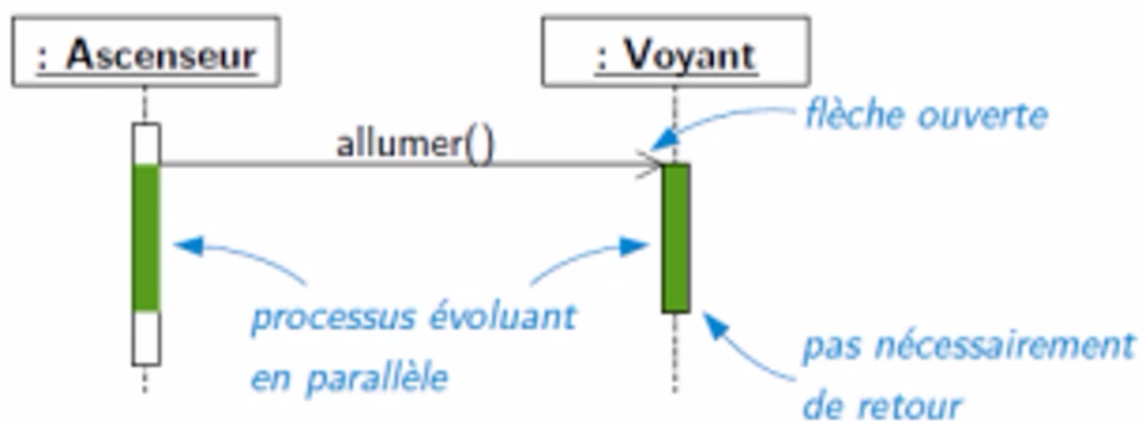
Message numéroté à partir de 1 (et faire 1.1 ; 1.2 ; ...

- Différents types d'envois de messages



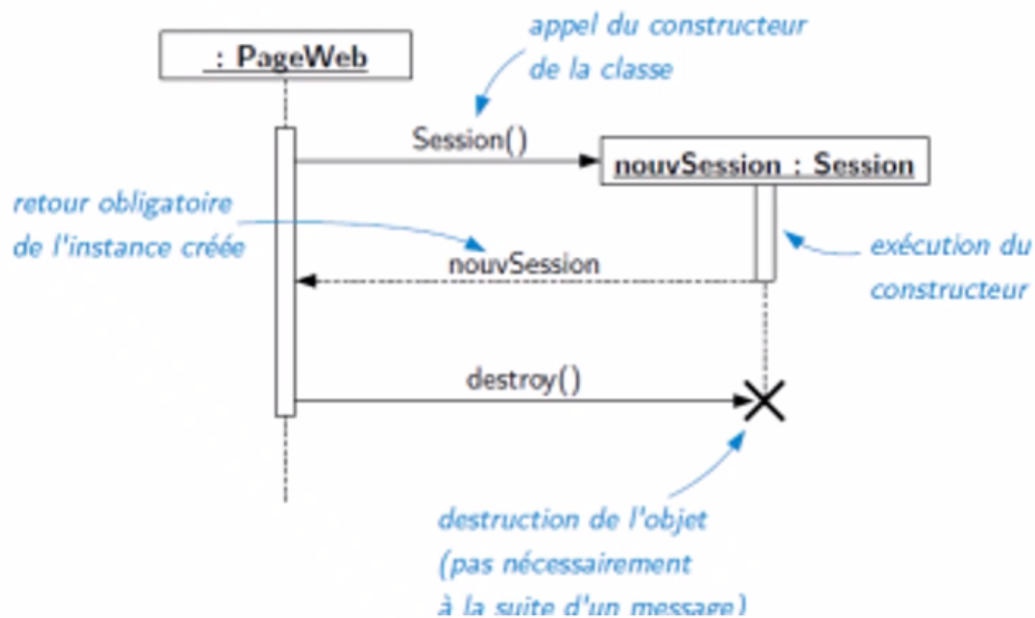
- Synchrone = le plus régulier, attente que l'activation de la méthode invoquée soit finie chez le destinataire
- Asynchrone = On attend pas la fin → Fct parallèle

Asynchrone :



Envoie de message :

Création et destruction



On a aussi les options (opt) : un if mais sans forcément de else

L'alternative (alt) c'est un if avec un else

la boucle c'est une loop avec un minimum et un maximum

l'assertion : peut-être le seul type d'envoi possible sans erreur par moment

il s'agit d'une notion proche des assertions dans les langages de programmation qui spécifient une condition sur l'état du système qui doit être vérifiée pour que cet état soit correct

▼ Diagramme d'activité

def : Visualise les flux de travail de données ou de contrôle

utilité : idéal pour modéliser les processus d'affaires et les flux de travail

exemple : processus d'approbation de congé

▼ Diagramme communication

Def : similaire au diagramme de séquence mais se concentre sur les relations entre objets

Utilité : idéal pour montrer les interactions dans un contexte moins séquentiel

exemple :

▼ Diagramme d'état

Def : montrer les états d'un objet à travers son existence

Utilité : Parfait pour modéliser les machine a états ou cycle de vie des entités

Exemple : états d'une commande en ligne (passée ,en préparation, expédié)

▼ Diagramme de composant

Def : représente les composant logiciels et leurs relations

Utilité : utilisé pour structurer et organiser le système

Exemple :

▼ Diagramme de déploiement

Déf : décrit l'architecture physique du système et déploiement des composants

Utilité : indispensable pour comprendre l'infrastructure requise et la distribution des composants

exemple :

▼ Diagramme de structure composite

Def : montre la structure interne des classes et leurs interactions

Utilité : approfondit les détails des collaborations entre parties d'un système

Exemple :

Bonnes pratiques :

- Utilisation de noms clair et descriptifs
- Cohérence dans vos notations
- Adaptation du grain selon l'audience

pwp page 264 a pour les 10 points importants

Rajouter les différents types de stéréotype

▼ Cas d'utilisation

Objectif: Comprendre les besoins du client pour rédiger le cahier des charges fonctionnel

Trois questions:

1. Définir les utilisations principales du système : à quoi sert-il ?
2. Définir l'environnement du système qui va l'utiliser ou interagir avec lui ?
3. Définir les limites du système où s'arrête sa responsabilité ?

Séquence d'étape

- Décrivant une interaction entre l'utilisateur et le système
- Permettant à l'utilisateur de réaliser un objectif

▼ Exemple

Systeme : Site de vente en ligne

Scénario : Commander

Le client s'authentifie dans le système puis choisit une adresse et un mode de livraison. Le système indique le montant total de sa commande au client. Le client donne ses informations de paiement.

Ce qui peut arriver : La transaction n'est pas autorisée; le système invite le client à changer de mode de paiement; le client modifie ses informations

La transaction est effectuée et le système en informe le client par e-mail.

- Ensemble de scénarios réalisant un objectif de l'utilisateur

- Fonctionnalités principales du système du point de vue extérieur

Acteur : Entité qui interagit avec le système

- Personne, chose, logiciel, extérieur au système décrit
- Représente un rôle (plusieurs rôles possibles pour une même entité)
- Identifié par le nom du rôle

Cas d'utilisation: Fonctionnalité visible de l'extérieur

- Action déclenchée par un acteur
- Identifié par une action (verbe à l'infinitif)

<https://www.youtube.com/watch?v=GC5BdRve38A>

▼ Diagramme de classe

<https://www.youtube.com/watch?v=QLPRsWCQ5BE>

