ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Object Oriented Conception

Professor:

- HARIK Ismail

UNIFIED MODELING LANGUAGE™

Class: 2025/2026

Introduction to UML & Basic Diagrams (Foundation of UML)

- UML Overview.
- Basic Diagrams. (Use Case, Sequence, Class) Diagrams

Activity & Transition Diagrams

- Activity Diagram, State Transition Diagram.
- *Case Study and Practical Application covering all prev Diagrams*

Advanced Structure Diagrams

- Package Diagram.
- Component Diagram.
- Deployment Diagram.

Communication & Interaction Diagrams

- Communication Diagram.
- Interaction Overview Diagram.

Design Patterns

- Design Patterns Overview.
- GOF Classification.
- Examples of Design Patterns: Singleton DAO, DTO.

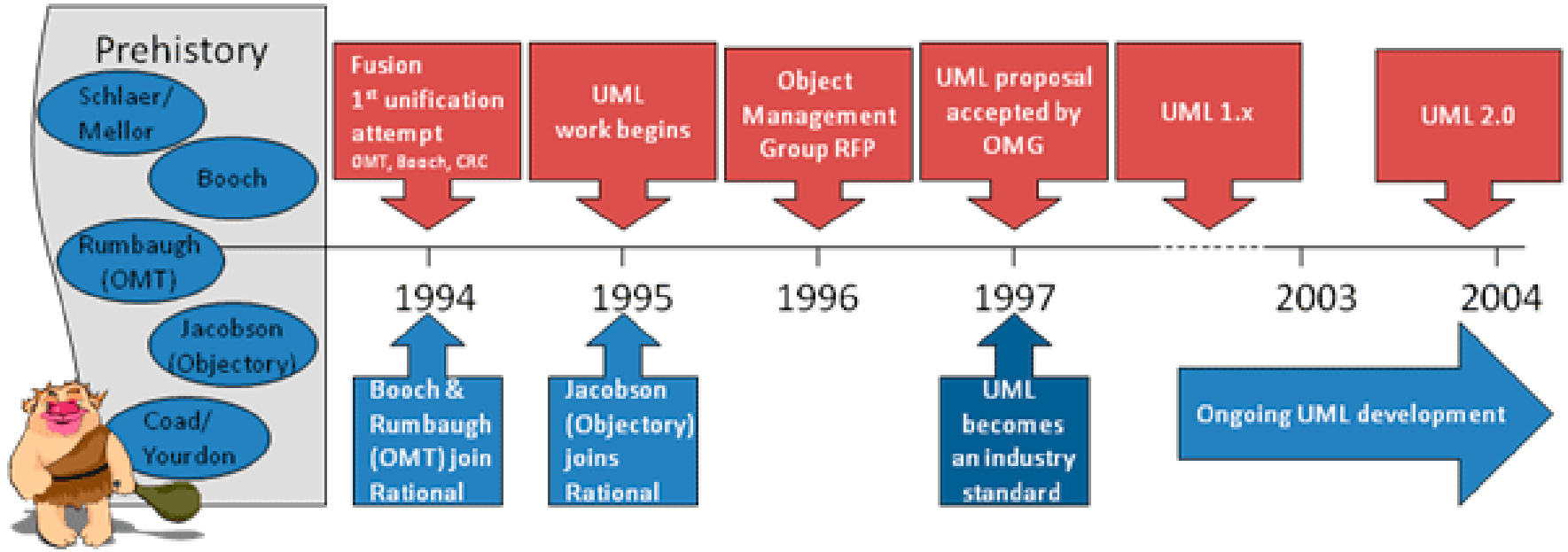
Qu'est-ce que l'UML ? L'UML (Unified Modeling Language) est un langage visuel standardisé pour spécifier, construire et documenter des systèmes logiciels.
Aide à visualiser, concevoir et communiquer les structures et les comportements des systèmes.
Principalement utilisé dans les systèmes orientés objet, mais applicable à d'autres paradigmes.

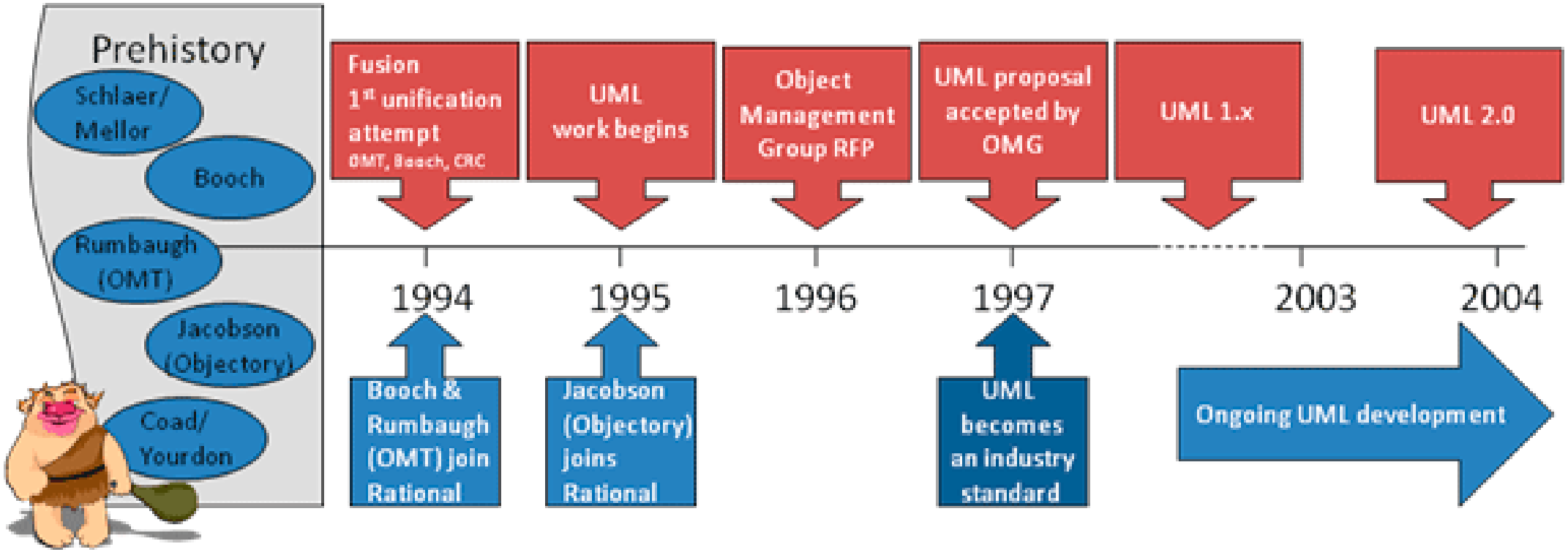
Noter Bien:

- L'UML est un **langage**, pas une méthodologie.
- Il fournit un ensemble de diagrammes pour modéliser différents aspects d'un système.

History of UML:

- **Origines :** Développé au milieu des années 1990 par Grady Booch, Ivar Jacobson et James Rumbaugh.
- **Unification :** Combinaison des langages de modélisation antérieurs (Booch, OMT, OOSE).
- **Standardisation :** Géré par l'Object Management Group (OMG) depuis 1997.
- **Dernière version :** UML 2.0 (couverte dans la troisième édition du livre de Fowler).




Prehistory
Schlaer/ Mellor
Booch
Rumbaugh (OMT)
Jacobson (Objectory)
Coad/ Yourdon
Fusion 1st unification attempt
OMT, Booch, CRC
UML work begins
Object Management Group RFP
UML proposal accepted by OMG
UML 1.x
UML 2.0
1994
1995
1996
1997
2003
2004
Booch & Rumbaugh (OMT) join Rational
Jacobson (Objectory) joins Rational
UML becomes an industry standard
Ongoing UML development

Why Use UML ?

- **Clarté visuelle :** Simplifie les systèmes complexes grâce à des diagrammes.
- **Documentation :** Sert de plan directeur pour le développement logiciel.
- **Conception et analyse :** Aide à identifier les exigences et à concevoir des systèmes.

"UML is a tool for communication, not a design methodology. Its primary value is in helping people understand and discuss software designs. It's not about creating perfect diagrams, it's about creating shared understanding"

– **Martin Fowler**, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*

Types of UML Diagrams:

UML 2.x définit 14 types de diagrammes officiels, classés en **diagrammes structurels** et **diagrammes comportementaux**.

1. Structural Diagrams:

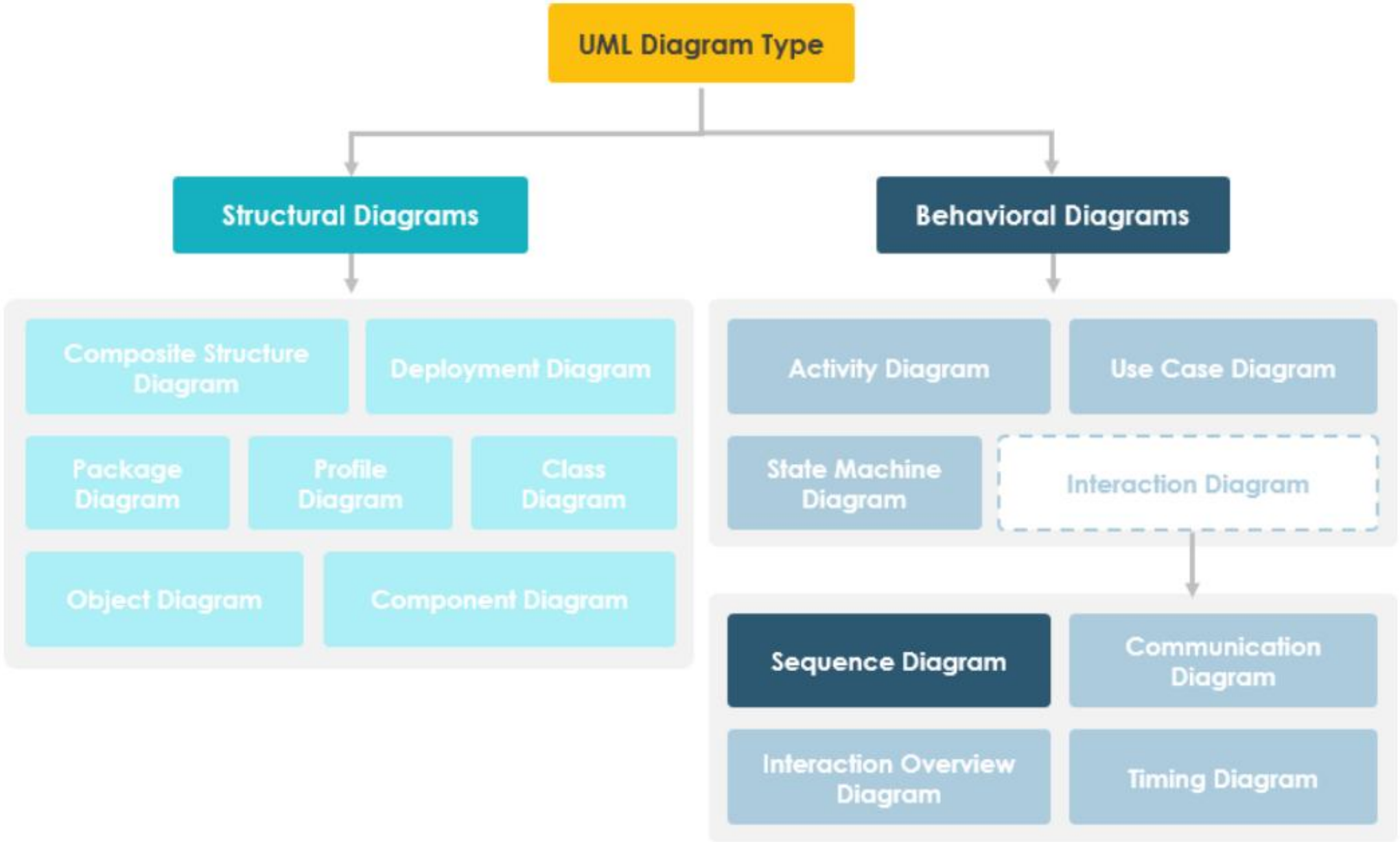
Représentent *la structure statique* d'un système, en se concentrant sur les composants et leurs relations.

- Diagramme de classes.
- Diagramme de composants.
- Diagramme de déploiement.
- Diagramme de packages.
- Diagramme d'objets.
- Diagramme de structure composite.
- Diagramme de profil.

2. Behavioral Diagrams:

Représentent *le comportement dynamique* d'un système, en se concentrant sur les interactions et les changements d'état.

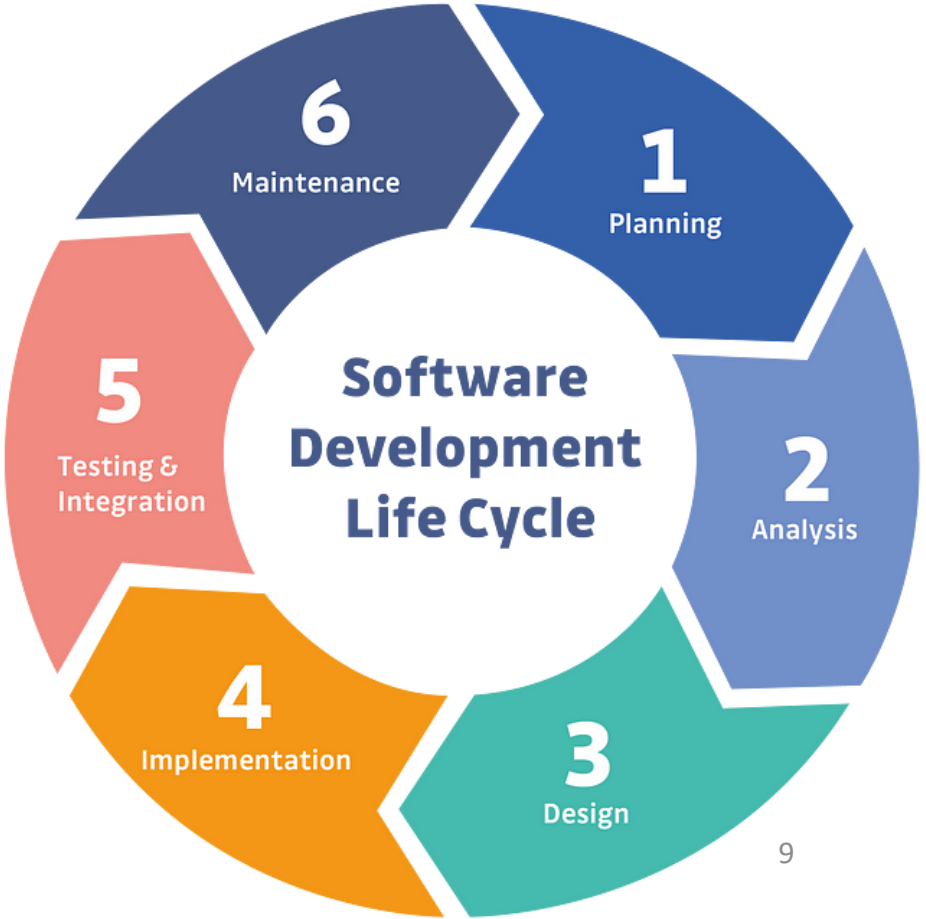
- Diagramme de cas d'utilization.
- Diagramme de sequence.
- Diagramme d'activités.
- Diagramme d'état-transition (State Machine)
- Diagramme de communication (Collaboration)
- Diagramme global d'interaction.
- Diagramme de timing.


UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

UML in Software Development Lifecycle (SDLC)?

L'UML est utilisé dans toutes les phases du cycle de vie du développement logiciel (SDLC). Il assure la cohérence et la clarté tout au long du processus de développement:

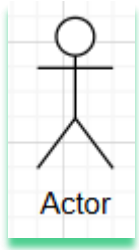
1. **Requirement Analysis:** Use Case Diagrams, Activity Diagram.
2. **System Design:** Class Diagrams, Sequence Diagrams.
3. **Implementation:** Diagrammes de composants, Diagrammes de déploiement.
4. **Testing:** Diagrammes d'activités, Diagrammes d'état-transition.
5. **Maintenance:** Tous les diagrammes pour comprendre et mettre à jour le système.

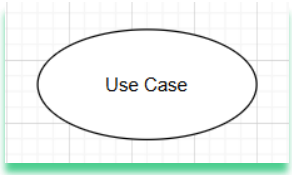


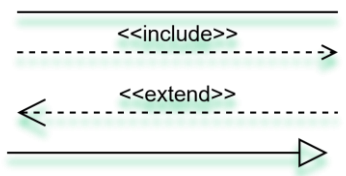
Definition: Un diagramme de cas d'utilisation capture les exigences fonctionnelles d'un système. Il montre les acteurs, les cas d'utilisation et leurs relations, fournissant une vue de haut niveau de ce que le système est censé faire.
– **Alistair Cockburn**, *Writing Effective Use Cases*

Elements:

- **Acteur :** Représente un utilisateur ou un système externe interagissant avec le système.
- **Use Case:** Represents a specific functionality or action.
- **Relationships:** Association, Include, Extend, generalization.

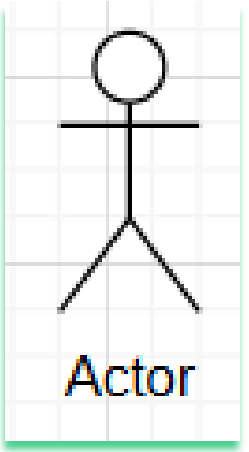






Acteur (Actor):

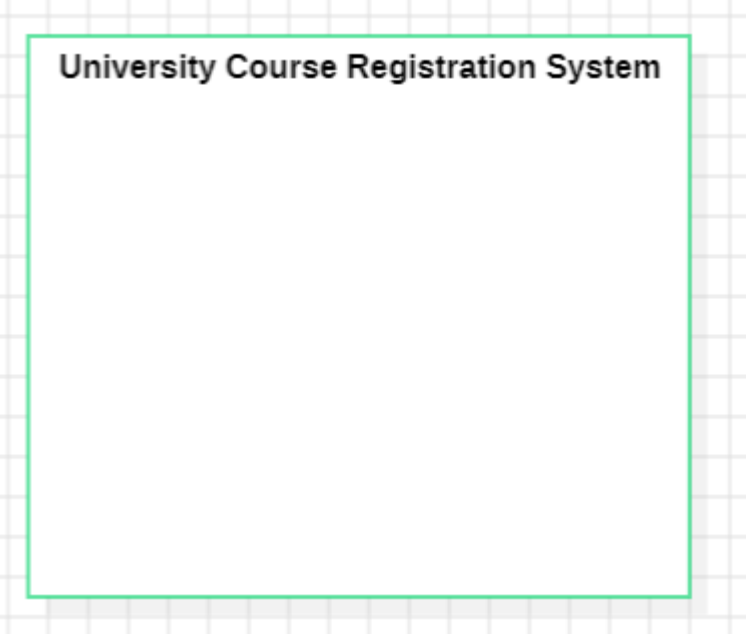
- Représente un utilisateur ou un système externe interagissant avec le système.
- Peut être un **humain** (Client, Bibliothécaire) ou un **système externe** (Passerelle de paiement).
- Représente des **rôles**, pas des individus spécifiques.
- **Exemples d'acteurs :** Étudiant, Enseignant, Administrateur, Parent.



Un acteur a un **nom unique** et une **description optionnelle**, comme *Passager : Une personne dans le train*.

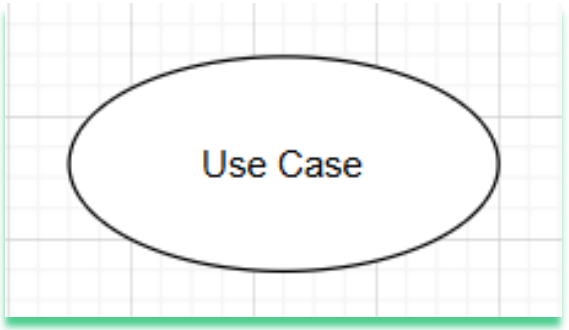
Limite du système:

La limite du système délimite clairement les frontières du système, indiquant quels composants sont internes au système et quels sont les acteurs ou entités externes interagissant avec le système



Cas d'utilisation:

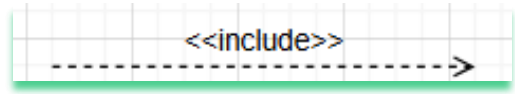
- Un cas d'utilisation représente une fonctionnalité ou une action spécifique que le système effectue pour atteindre un objectif pour un acteur.
- Décrit **ce que** le système fait, pas **comment** il le fait.
- **Exemples de cas d'utilisation :** Planifier les cours, S'inscrire à un cours, Gérer le contenu des cours.



Relationships in Use Case Diagrams:

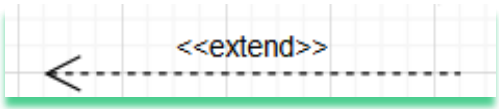
- Les relations définissent comment les acteurs et les cas d'utilisation interagissent ou dépendent les uns des autres.
- ***Association***: Links an actor to a use case (Student —→ Enroll in Course).

- ***Include***: Signifie que le cas d'utilisation inclus est **obligatoire** et se produit toujours dans le cadre du cas d'utilisation de base.



Order Food —(include)→ Pay Bill, Enroll in Course —(Include)→ Validate Student Details

- **Extend***:* le cas d'utilisation étendu est **optionnel** et ne se produit que sous certaines conditions**.**

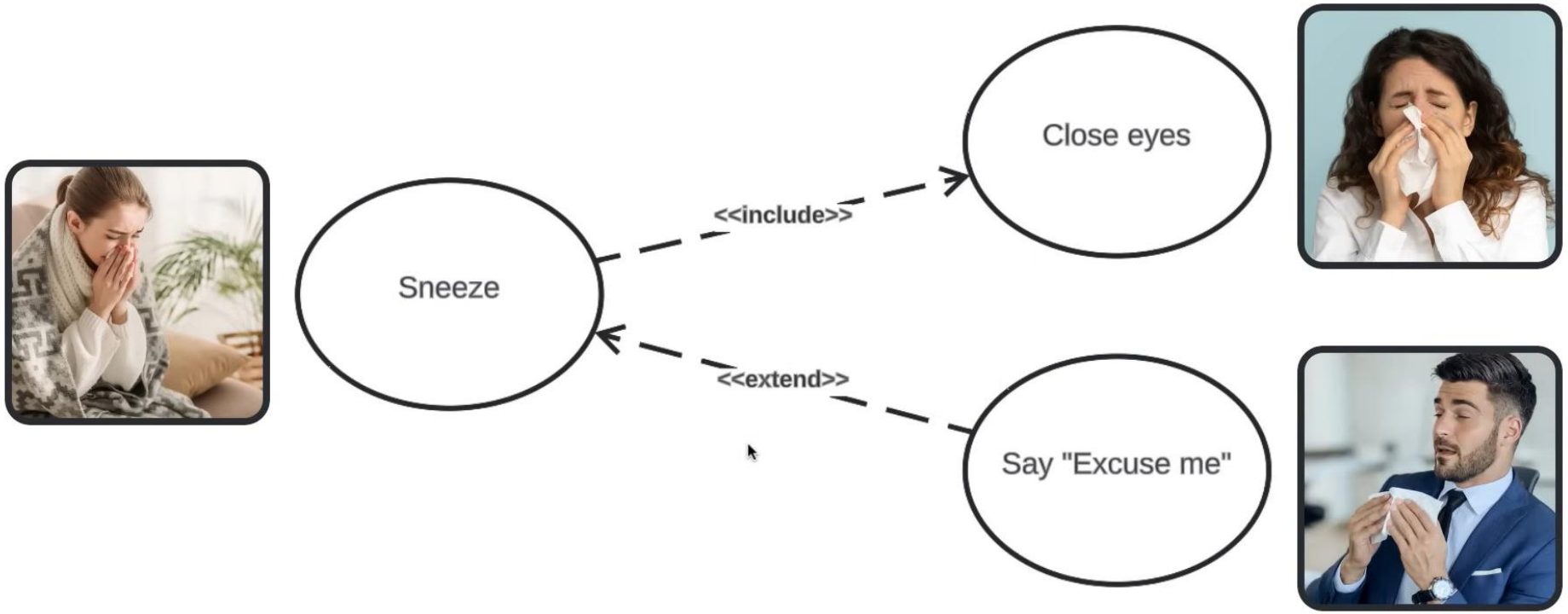


Order Food can be *extended* by Order Dessert, You **may** order dessert, but only if you want to. It's not mandatory.

Relationships in Use Case Diagrams:

- ***Generalization:*** Représente une relation d'**héritage** entre des acteurs ou des cas d'utilisation.
  - **Student**, **Professor**, and **Admin** sont des acteurs spécialisés qui héritent de l'acteur généralisé **User**.
  - **Pay with Credit Card** and **Pay with PayPal** sont des cas d'utilisation spécialisés qui héritent du cas d'utilisation généralisé **Make Payment.**

Basic Example



Exercise 1: Conception d'un diagramme de cas d'utilisation pour un système de gestion de bibliothèque:

Dans ce système, les étudiants ont la possibilité de réserver des livres ou d'annuler leurs réservations. Ils reçoivent des notifications de confirmation ou d'annulation de leurs demandes. Par la suite les responsables peuvent valider ou annuler la demande. Si un étudiant ne retourne pas un livre dans les délais impartis, le système de gestion de notification envoie une alerte au responsable de la bibliothèque. Par la suite, le responsable contacte l'étudiant pour lui rappeler de retourner le livre

❖ Acteurs:

- **Étudiant**: Un type spécialisé d'utilisateur.
- **Personnel:** Un type spécialisé d'utilisateur.
- **Notification System**: Le système de gestion des notifications.

❖ Cas d'utilisation:

✓ Utilisateur:

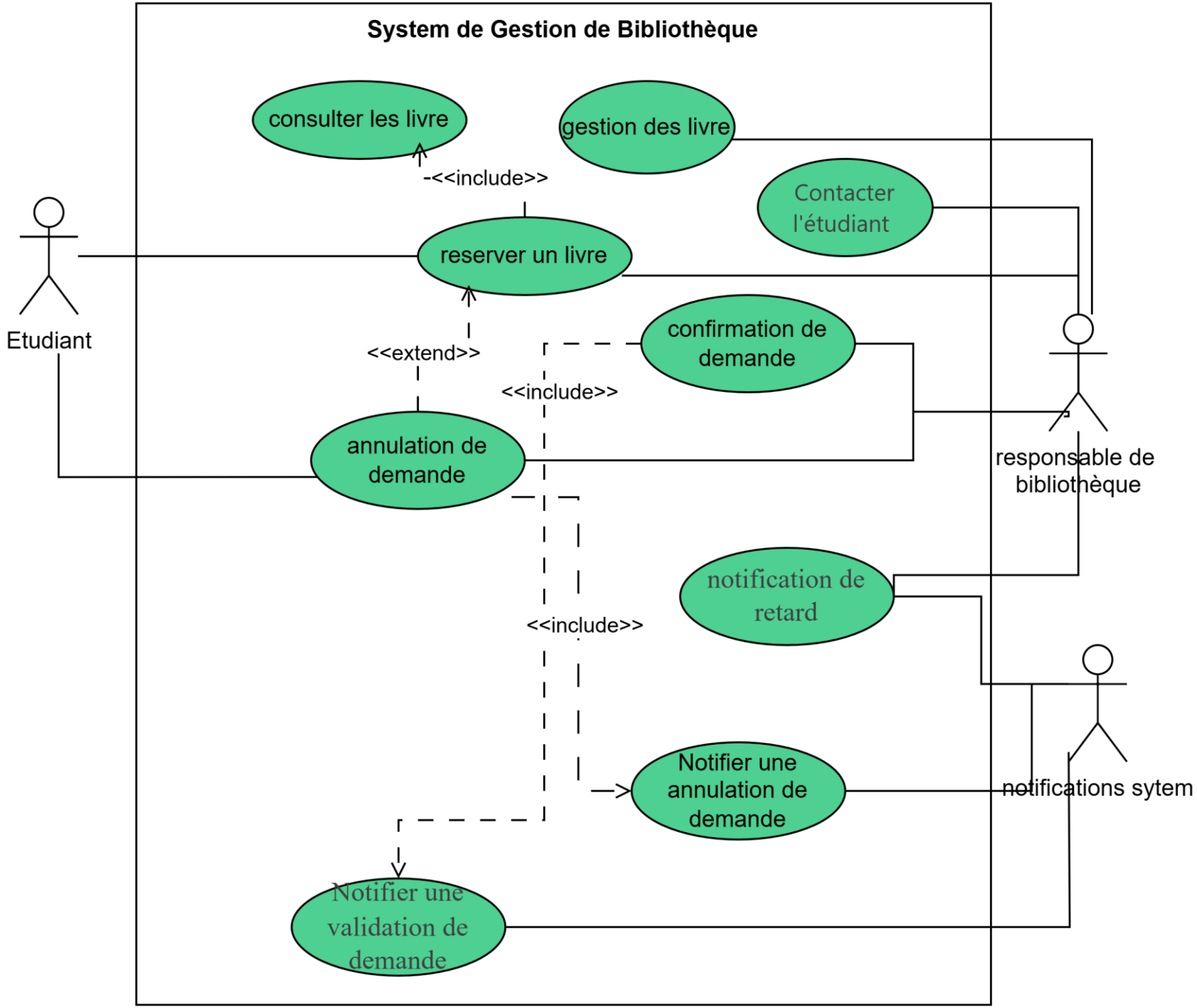
- Consulter les livres.
- validation d'identifiant et mot de passe
- Réserver un livre.
- Annuler la demande.

✓ Personnel:

- Gestion des livres
- contacter l'étudiant
- annulation de demande.
- confirmation de demande.

✓ Notification System:

- Envoyer une notification de retard.
- Notifier une annulation de demande.
- Notifier une validation de demande.


System de Gestion de Bibliothèque
consulter les livre
gestion des livre
-<<include>>
Contacter l'étudiant
reserver un livre
Etudiant
<<extend>>
confirmation de demande
<<include>>
annulation de demande
responsable de bibliothèque
notification de retard
<<include>>
Notifier une annulation de demande
notifications sytem
Notifier une validation de demande

Exercise 2: Système de Location de Véhicules

Vous devez modéliser le processus de location d'un véhicule dans une agence de location sous forme de cas d'utilisation. Le processus commence lorsque le client recherche un véhicule disponible en fonction de critères spécifiques, tels que le type de véhicule , la marque, le nbr de KMs. Le système affiche les véhicules correspondants et le client en sélectionne un tout en fournissant ses informations personnelles. Ensuite, le système vérifie la disponibilité du véhicule et valide les informations du client. Si cette validation est réussie, il calcule le coût total de la location et propose des options de paiement. Le client choisit un mode de paiement et effectue la transaction. Si le paiement est accepté, le système confirme la réservation et génère un contrat de location. L'agent de location valide la disponibilité du véhicule, finalise le contrat et remet les clés au client une fois le contrat signé. En cas d'échec de la validation ou du paiement, le système affiche un message d'erreur. De plus, l'agent de location peut annuler une réservation existante à la demande du client ou en cas de problème.

❖ Actors:

Client, Agent de Location, Système de Paiement.

❖ Cas d'utilisations:

✓ **Client**:

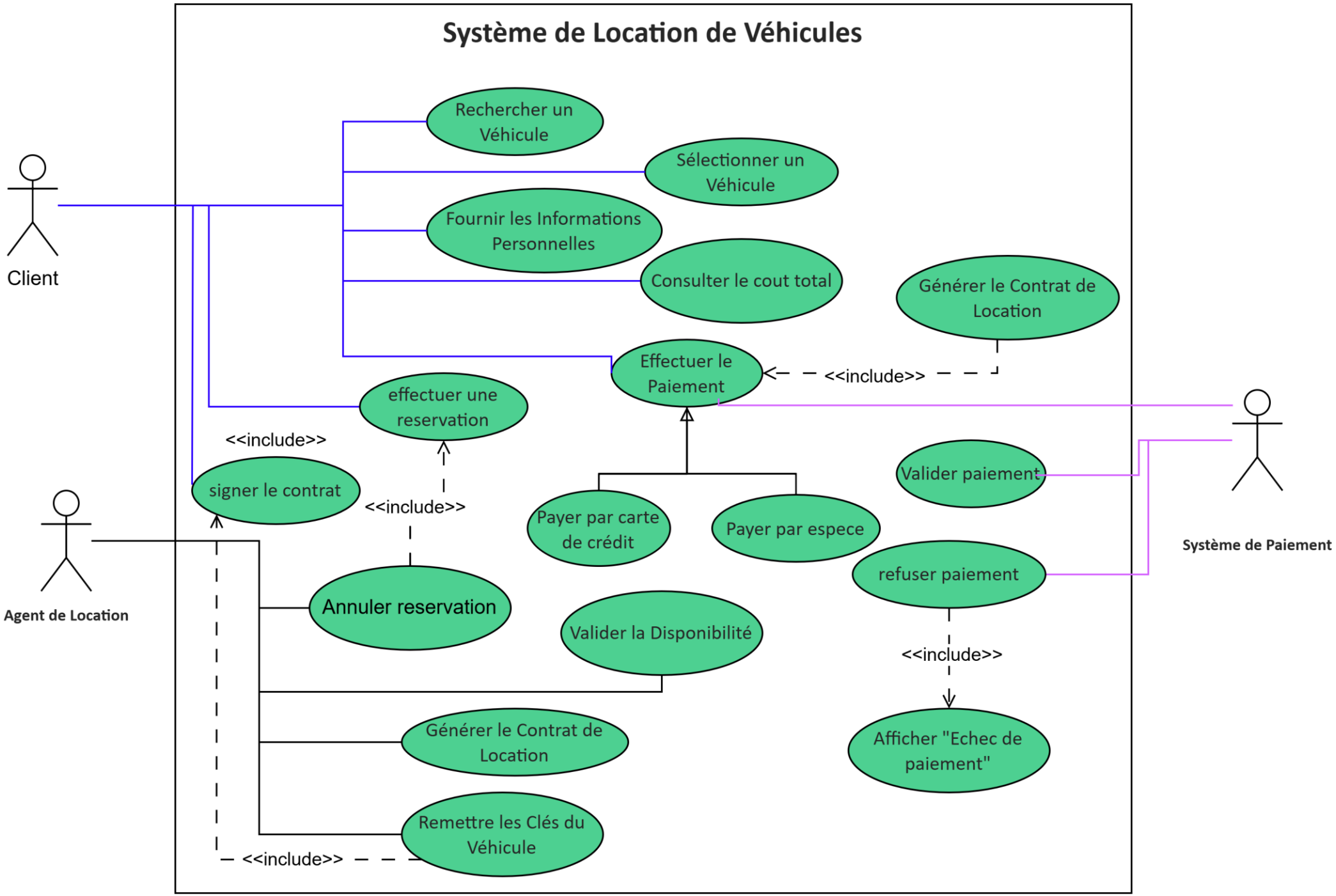
- Rechercher un Véhicule.
- Sélectionner un Véhicule.
- Fournir les Informations Personnelles.
- Effectuer le Paiement.
- Signer le Contrat.
- Effectuer une Réservation.

✓ **Agent de Location**:

- Valider la Disponibilité.
- Générer le Contrat de Location.
- Remettre les Clés du Véhicule.
- Annuler une Réservation.

✓ **Système de Paiement :**

- Valider le Paiement.
- Refuser le Paiement.


Système de Location de Véhicules
Client
Agent de Location
Système de Paiement
Rechercher un Véhicule
Sélectionner un Véhicule
Fournir les Informations Personnelles
Consulter le cout total
Générer le Contrat de Location
Effectuer le Paiement
<<include>>
effectuer une reservation
<<include>>
signer le contrat
<<include>>
Payer par carte de crédit
Payer par espece
Valider paiement
refuser paiement
<<include>>
Afficher "Echec de paiement"
Annuler reservation
Valider la Disponibilité
Générer le Contrat de Location
Remettre les Clés du Véhicule
<<include>>

Commun Mistakes to avoid

- Overloading the Diagram with Too Many Use Cases.
- Confusing Actors with Use Cases.
- Misusing <<include>> and <<extend>> Relationships.
- Ignoring the System Boundary
- Missing Key Use Cases or Actors
- Overusing Generalization
- Including non-functional requirements

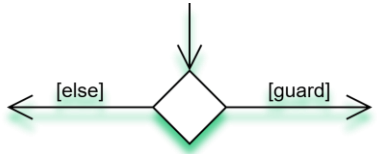
Definition: Les diagrammes d'activité sont utiles pour décrire des algorithmes complexes, des processus métier et les flux de travail (workflows).
Il représente les enchaînements d'actions, les décisions, et les flux parallèles.

Elements:

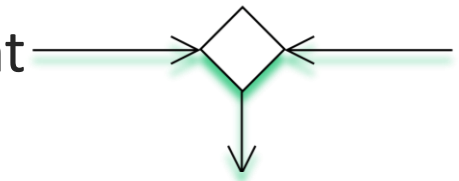
- **Nœud Initial (Début):** Represents the beginning of the workflow (solid circle).

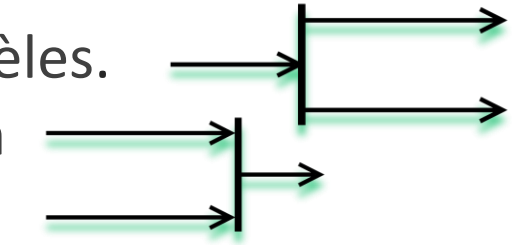
- **Activité / Action :** Represents a task or step in the process.

- **Décision:** est un point où le flux se divise en plusieurs chemins en fonction d'une condition.



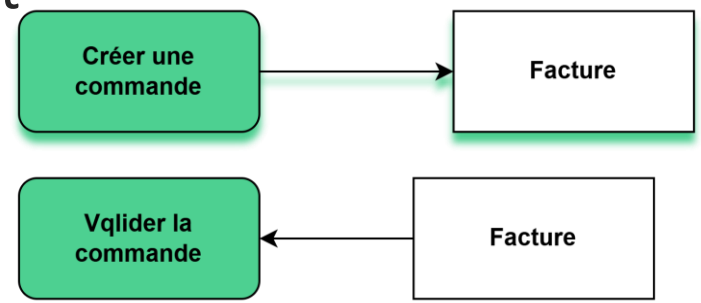
1.Nœud de Fusion (Merge Node): est un point où plusieurs flux entrants sont combinés en un seul flux sortant



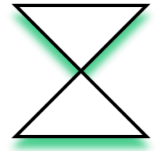


- **Nœud de Division (Fork Node):** Divise un flux en plusieurs branches parallèles.
- **Nœud de Jointure (Join Node):** Rejoint plusieurs branches parallèles en un seul flux.
- **Nœud Final:** Represents the end of the workflow.
- **Flux de Contrôle (Control Flow):** Représente la séquence des actions et la direction du flux.
- **Object Node (Nœud d'Objet):** Représente un objet utilisé ou produit dans l'activité.

  - **Créer une commande** Cette action déclenche automatiquement la **création d'une facture.**
  - **Valider la commande** : Pour valider la commande, la **facture** doit être disponible.



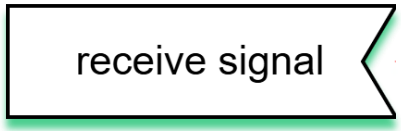
8. **Événement Temporel:** Représente un délai ou un déclencheur temporel.

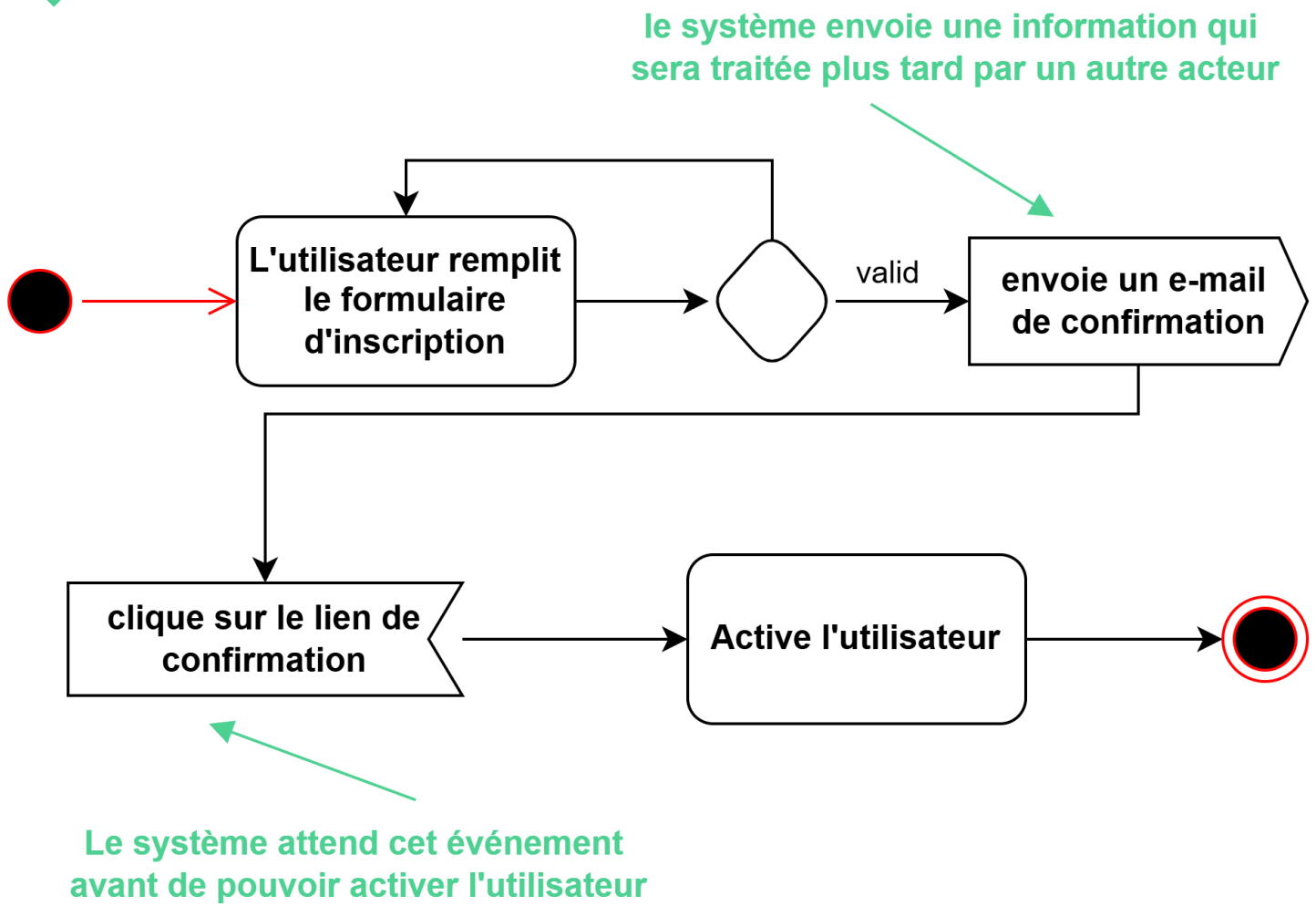


9. **Signals :** un mécanisme de communication asynchrone utilisé pour envoyer une information d'un objet à un autre sans attendre de réponse immédiate. Il représente un événement qui peut déclencher une action dans un autre élément du système.

- **Action d'Envoi de Signal:** Envoie un signal à une autre activité ou processus, ex Envoyer un e-mail de confirmation.

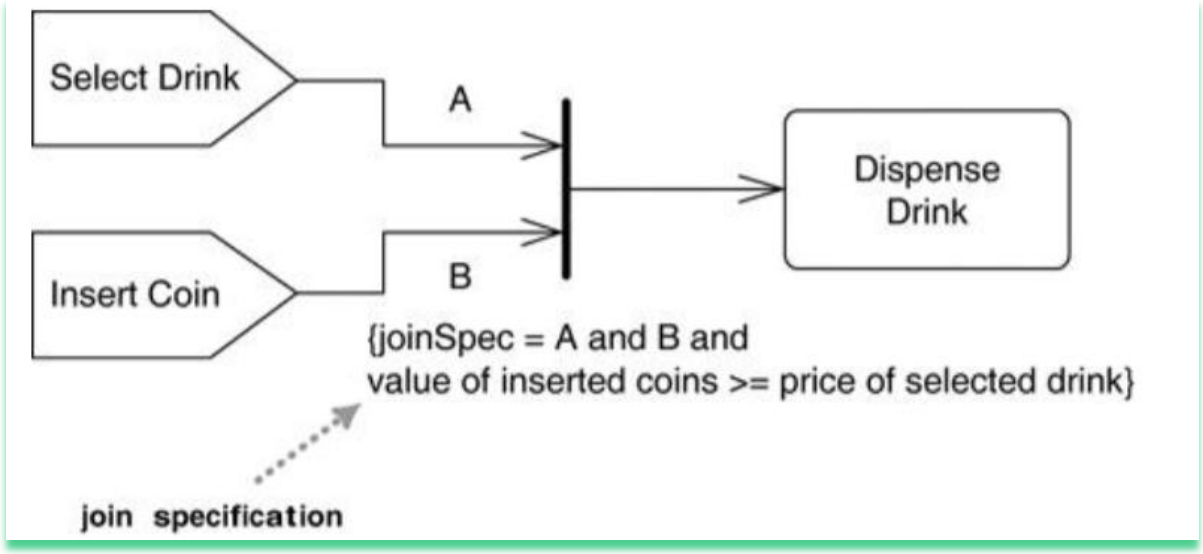
- **Action d'Acceptation d'Événement:** Attend qu'un événement se produise avant de continuer, ex: Attendre la confirmation de paiement.



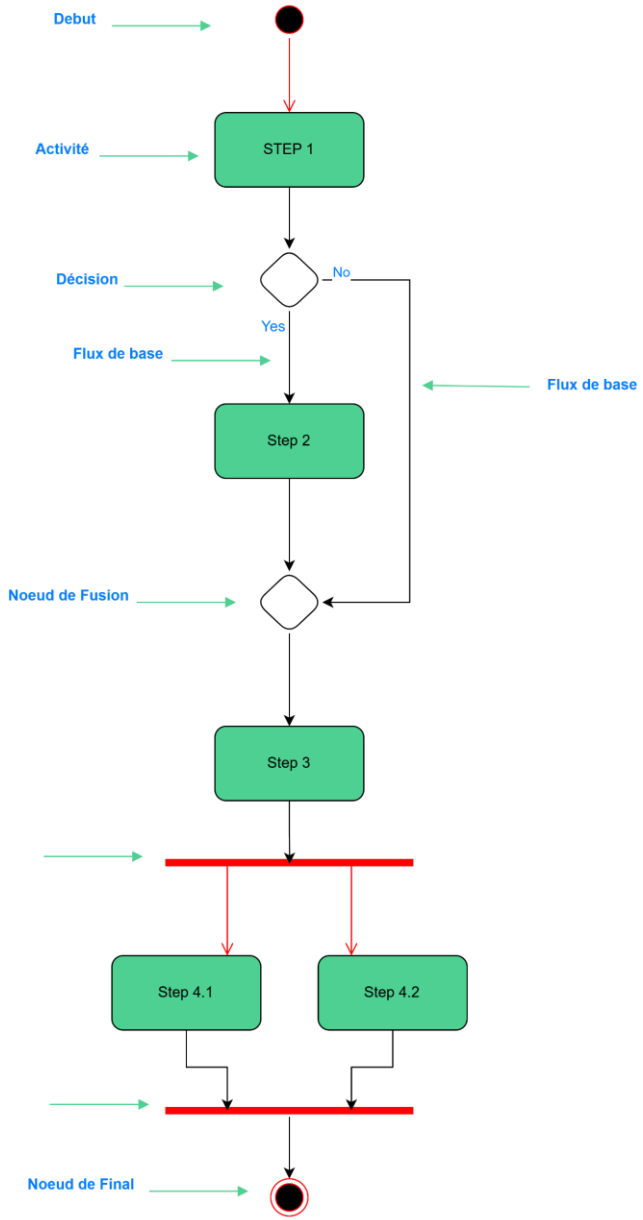

le système envoie une information qui sera traitée plus tard par un autre acteur
L'utilisateur remplit le formulaire d'inscription
valid
envoie un e-mail de confirmation
clique sur le lien de confirmation
Active l'utilisateur
Le système attend cet événement avant de pouvoir activer l'utilisateur

10. Spécification de Jonction (Join Specifications):

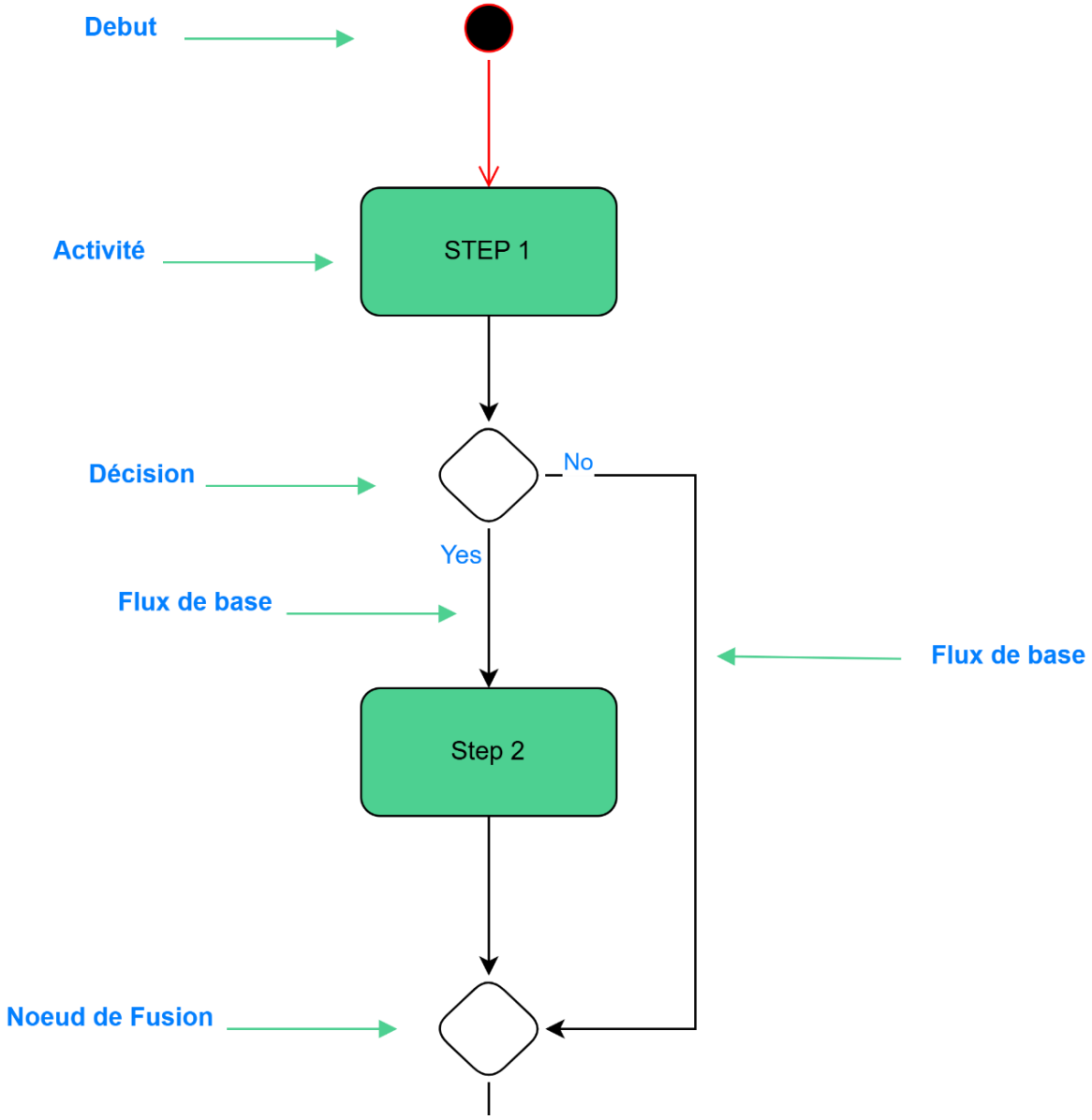
- **Synchronisation** : Garantit que tous les flux entrants requis sont terminés avant de continuer.
- **Logique Conditionnelle** : Permet de définir des conditions complexes pour fusionner les flux.



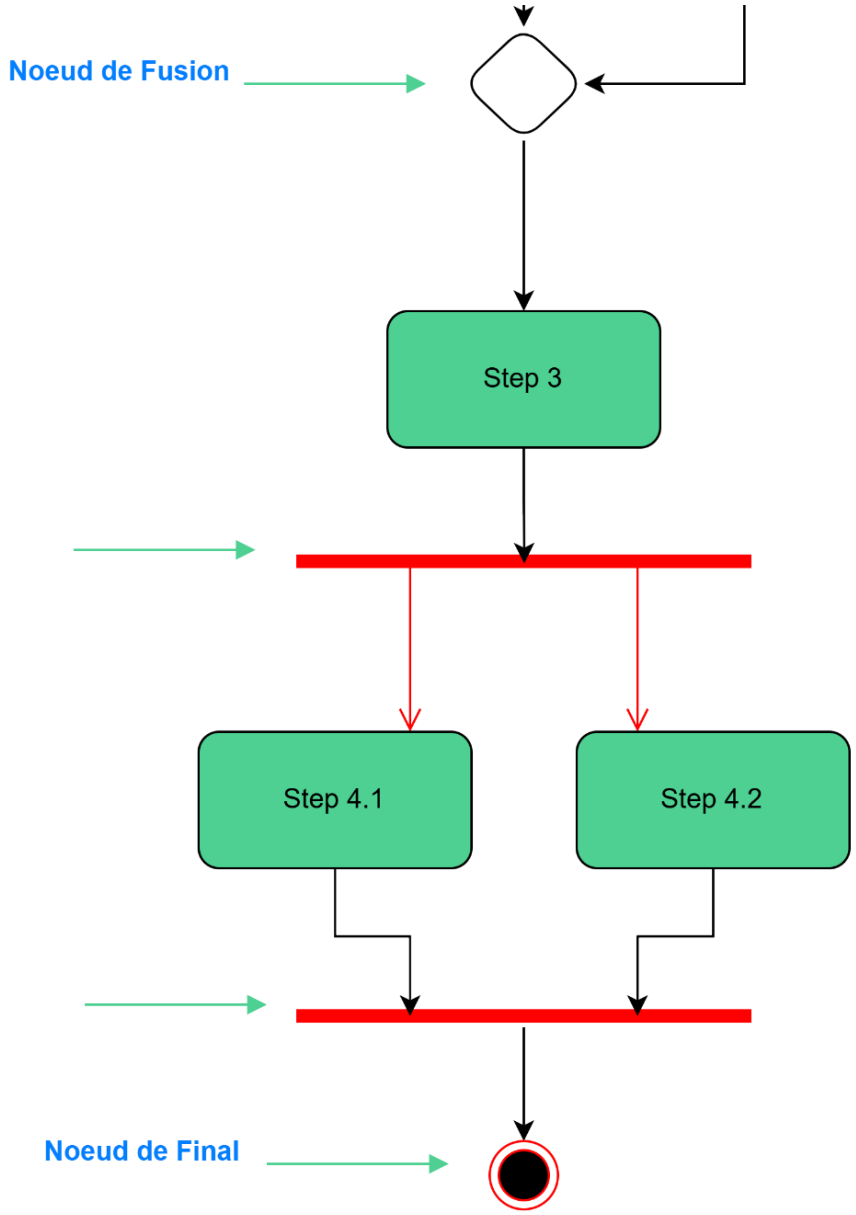
Example


Debut
Activité
STEP 1
Décision
No
Yes
Flux de base
Flux de base
Step 2
Noeud de Fusion
Step 3
Step 4.1
Step 4.2
Noeud de Final

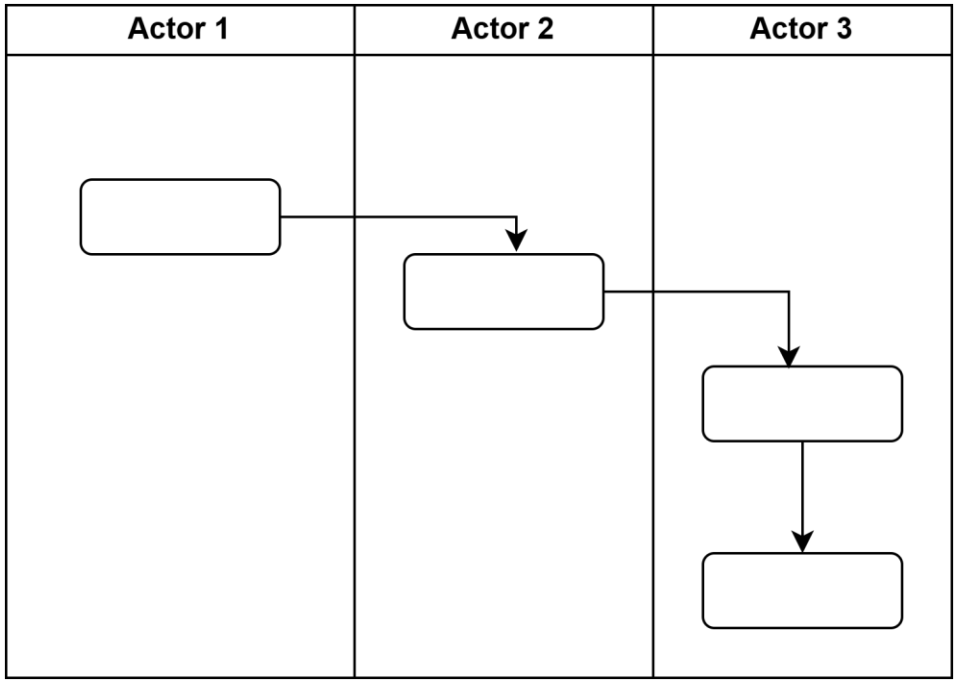
Example


Debut
Activité
STEP 1
Décision
No
Yes
Flux de base
Flux de base
Step 2
Noeud de Fusion

Example

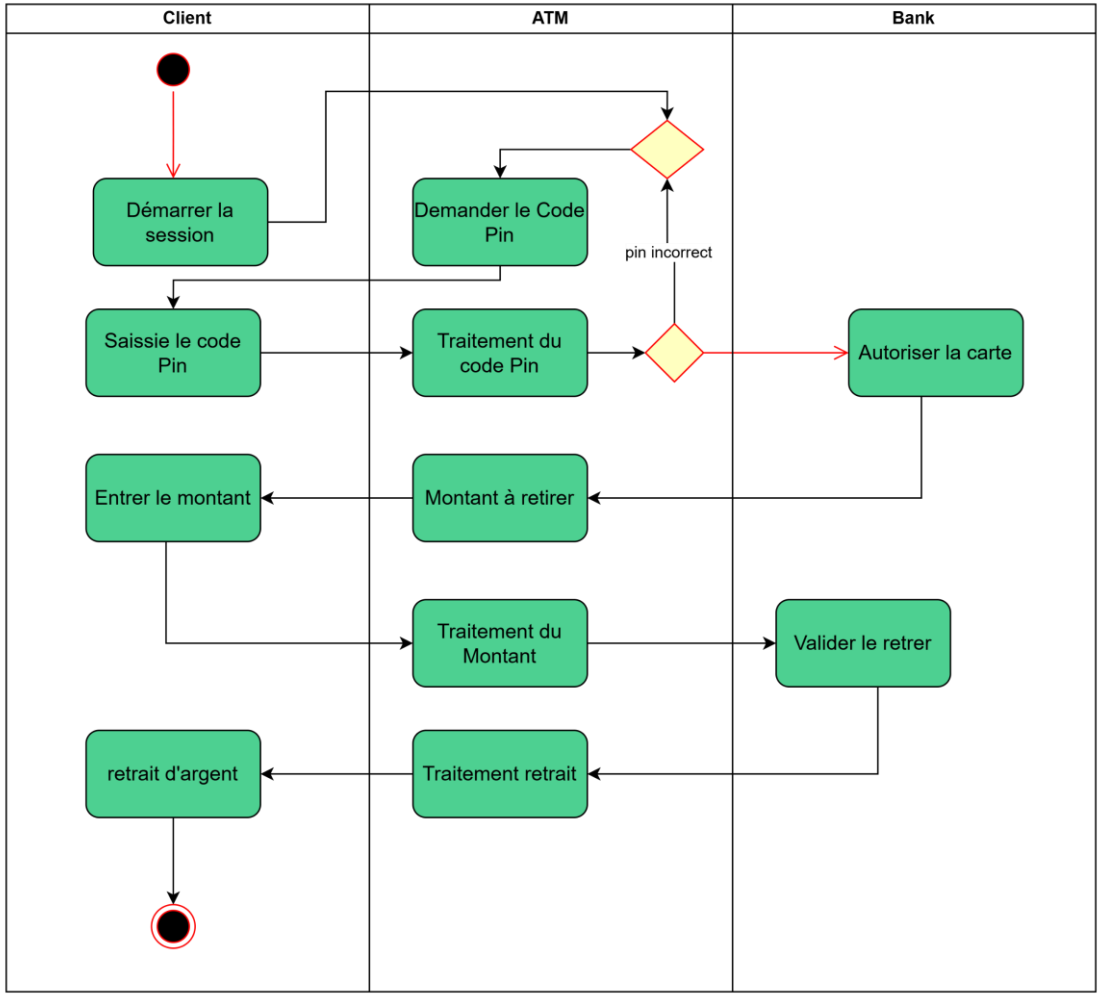


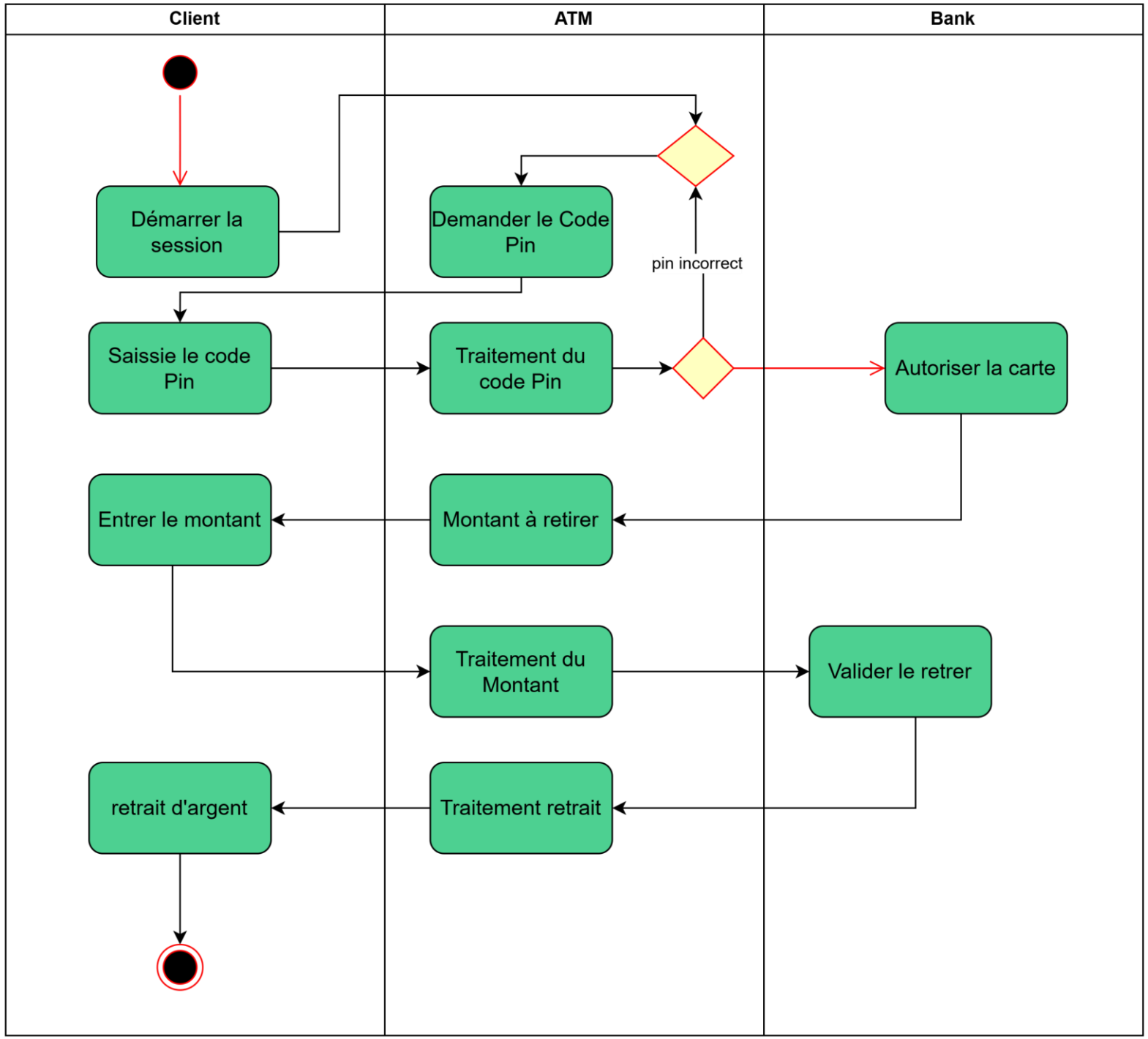
1.Couloirs (Swimlanes): Des zones verticales ou horizontales qui divisent le diagramme pour représenter les responsabilités des différents acteurs.
Exemple **:** "Client", "Système de Commande", "Service de Livraison".



Étapes pour Dessiner un Diagramme d'Activité:

1. Ajouter des Swimlanes (Couloirs)
2. Identifier les Acteurs Impliqués.
3. Déterminer les Étapes d'Action à Partir du Cas d'Utilisation.
4. Trouver un Flux entre les Activités


Client
ATM
Bank
Démarrer la session
Demander le Code Pin
pin incorrect
Saissie le code Pin
Traitement du code Pin
Autoriser la carte
Entrer le montant
Montant à retirer
Traitement du Montant
Valider le retrer
retrait d'argent
Traitement retrait


Client
ATM
Bank
Démarrer la session
Demander le Code Pin
pin incorrect
Saissie le code Pin
Traitement du code Pin
Autoriser la carte
Entrer le montant
Montant à retirer
Traitement du Montant
Valider le retrer
retrait d'argent
Traitement retrait

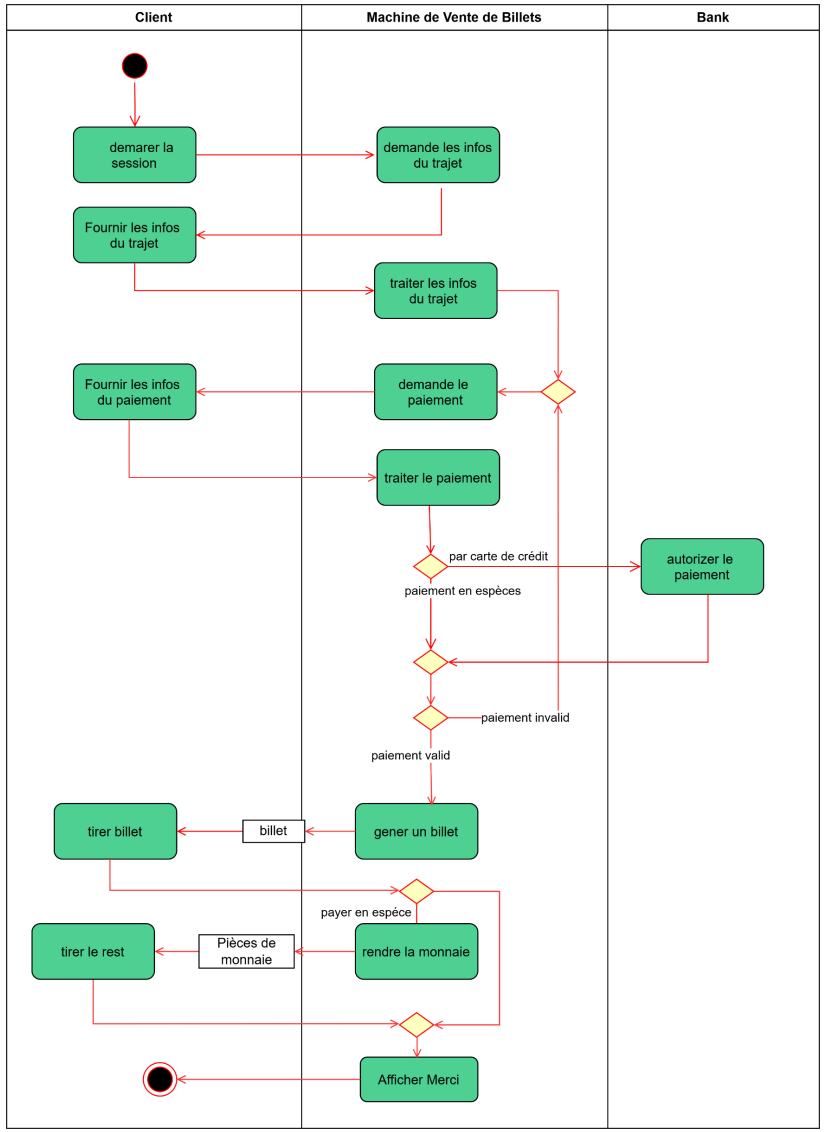
Exercice 1: Machine de Vente de Billets:

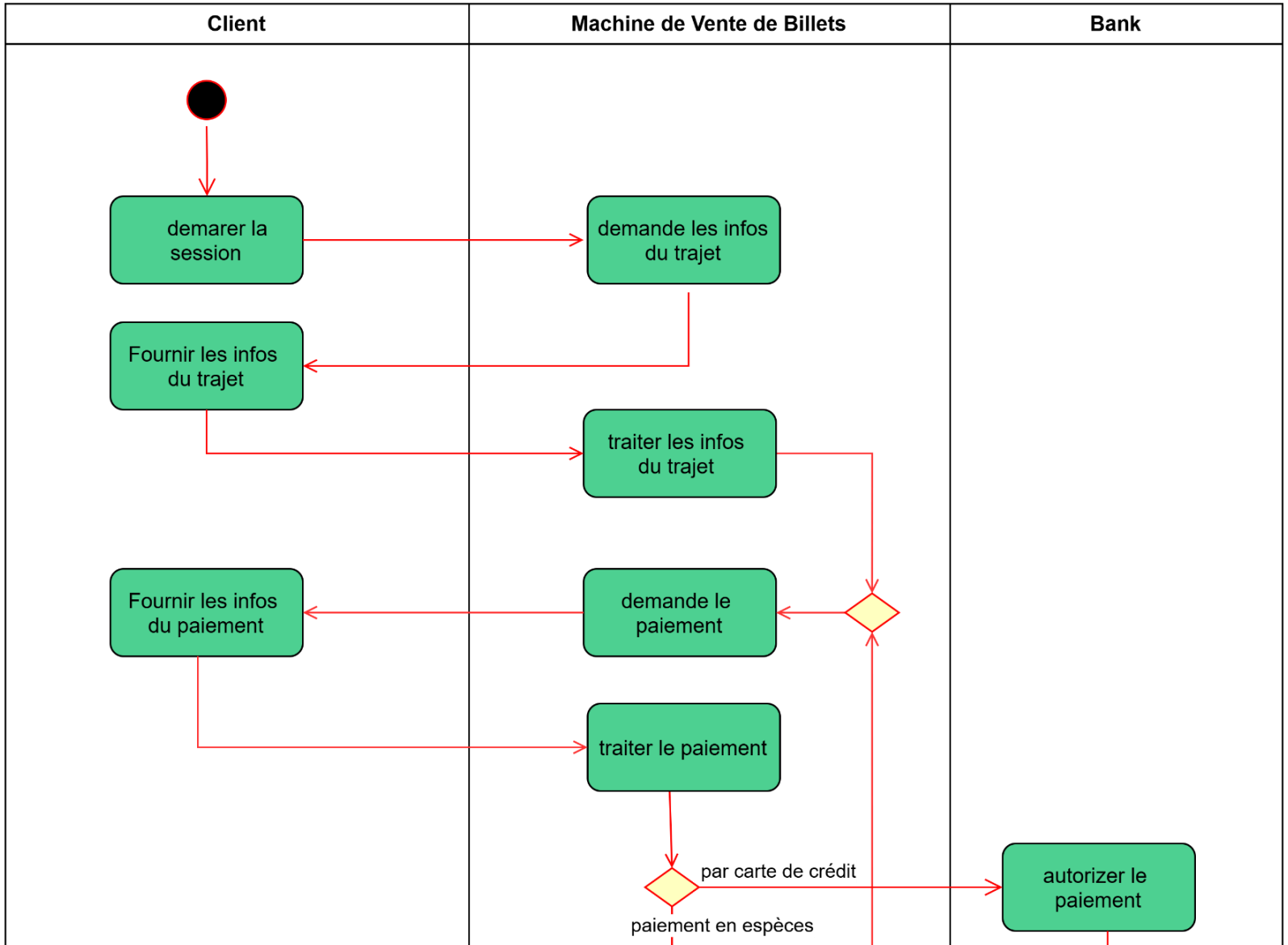
La machine de vente de billets demande au client des informations sur son trajet.
En fonction des informations fournies, la machine calcule le montant à payer et propose différentes options de paiement :

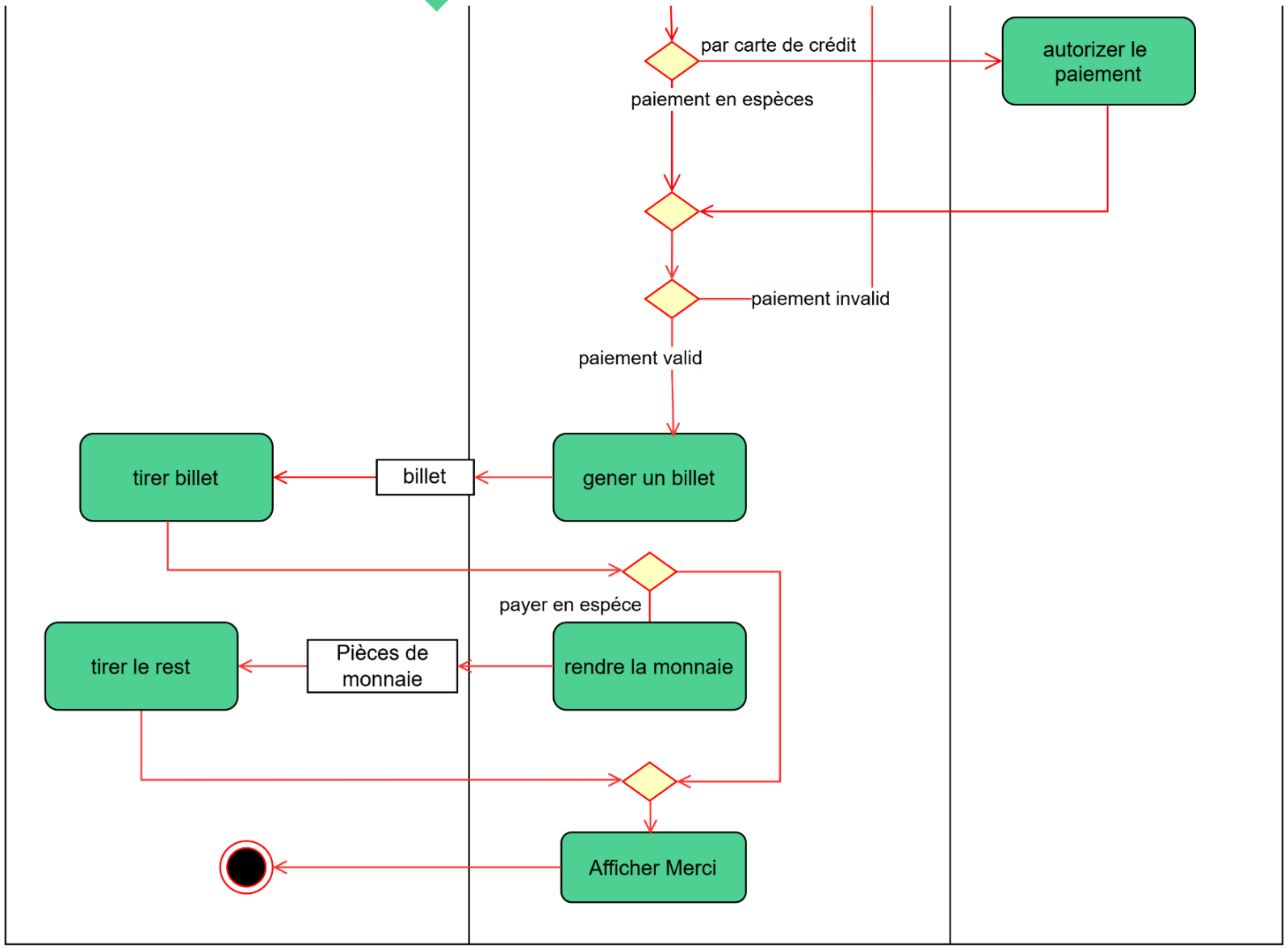
- **Paiement en espèces.**
- **Paiement par carte de crédit ou de débit.**

Si le client choisit de payer par carte, un acteur externe, la **Banque**, intervient pour autoriser le paiement. Une fois les informations de paiement validées, le système :

- Génère le billet.
- Rend la monnaie si le paiement a été effectué en espèces.
- Affiche un message de remerciement : "Merci".

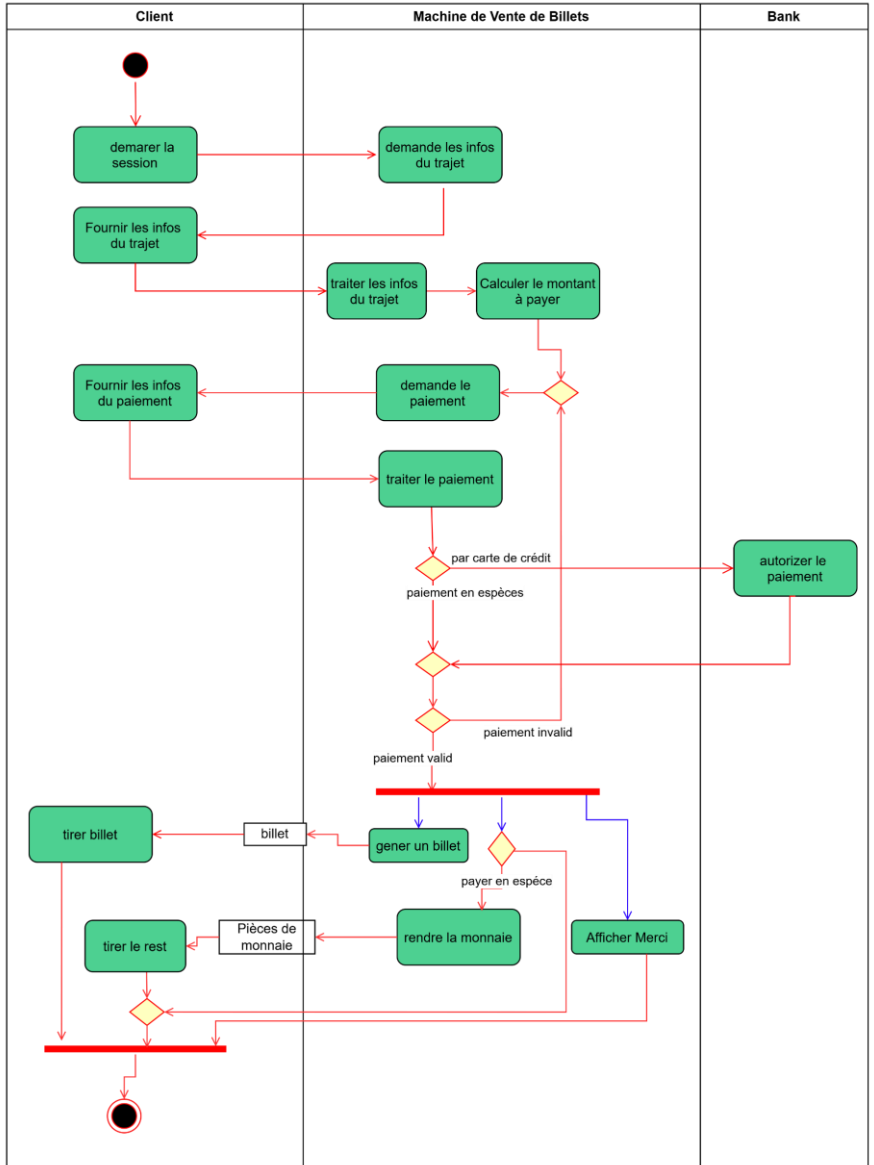

Client
Machine de Vente de Billets
Bank
demarer la session
demande les infos du trajet
Fournir les infos du trajet
traiter les infos du trajet
Fournir les infos du paiement
demande le paiement
traiter le paiement
par carte de crédit
paiement en espèces
autorizer le paiement
paiement invalid
paiement valid
tirer billet
billet
gener un billet
payer en espéce
tirer le rest
Pièces de monnaie
rendre la monnaie
Afficher Merci

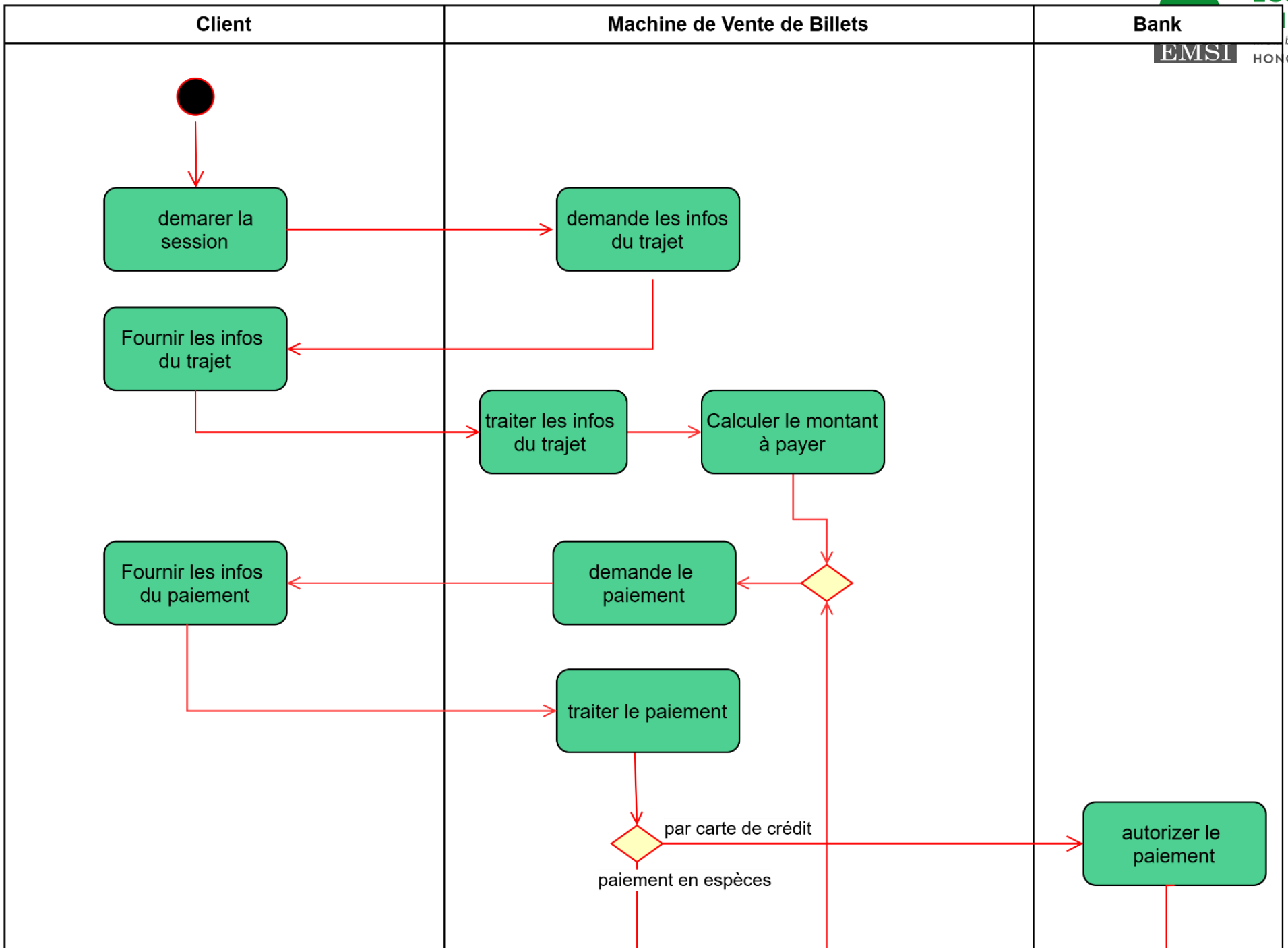

Client
Machine de Vente de Billets
Bank
demarer la session
demande les infos du trajet
Fournir les infos du trajet
traiter les infos du trajet
Fournir les infos du paiement
demande le paiement
traiter le paiement
par carte de crédit
paiement en espèces
autorizer le paiement

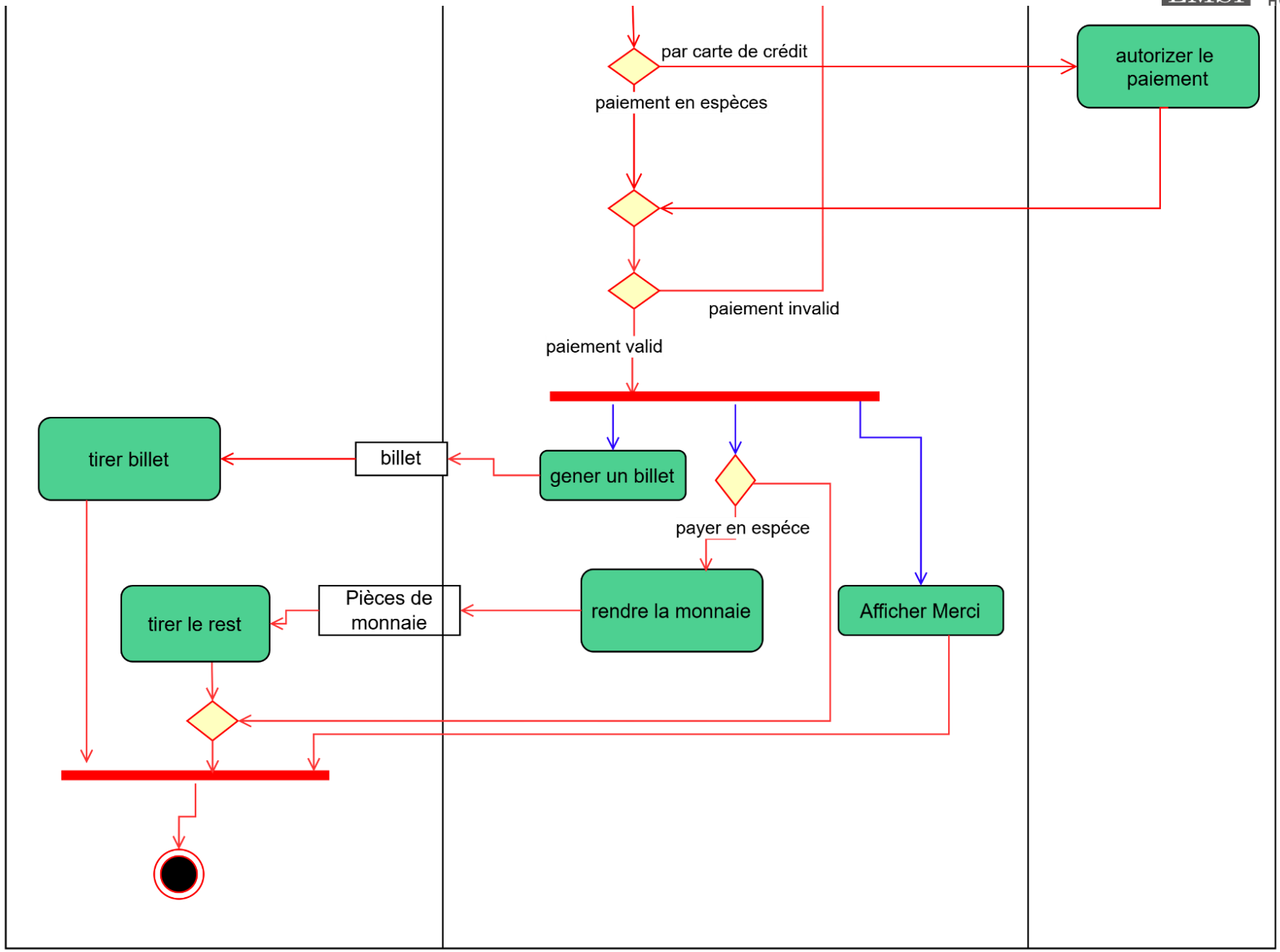

par carte de crédit
autorizer le paiement
paiement en espèces
paiement invalid
paiement valid
tirer billet
billet
gener un billet
payer en espéce
tirer le rest
Pièces de monnaie
rendre la monnaie
Afficher Merci

Modifiez le diagramme d'activité pour que les actions suivantes s'exécutent en parallèle.

1. **Générer le billet.**
2. **Rendre la monnaie (si le paiement est en espèces).**
3. **Afficher "Merci".**


Client
Machine de Vente de Billets
Bank
demarer la session
demande les infos du trajet
Fournir les infos du trajet
traiter les infos du trajet
Calculer le montant à payer
Fournir les infos du paiement
demande le paiement
traiter le paiement
par carte de crédit
paiement en espèces
autorizer le paiement
paiement invalid
paiement valid
tirer billet
billet
gener un billet
payer en espéce
tirer le rest
Pièces de monnaie
rendre la monnaie
Afficher Merci

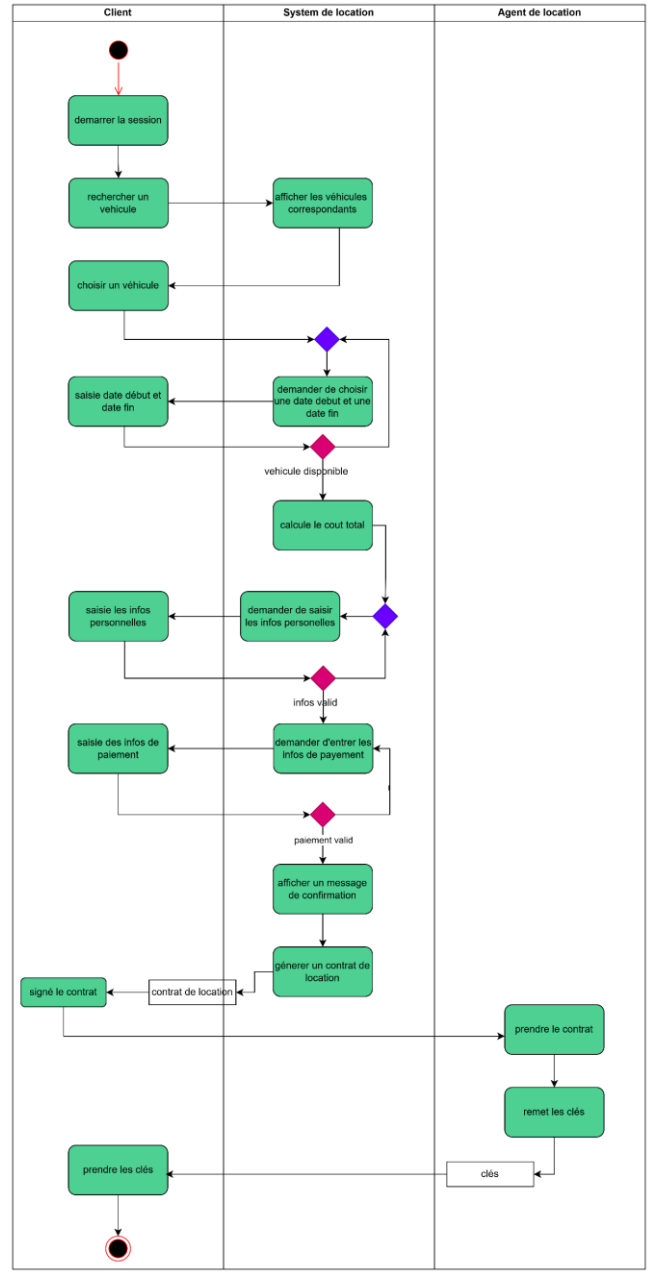

Client
Machine de Vente de Billets
Bank
demarer la session
demande les infos du trajet
Fournir les infos du trajet
traiter les infos du trajet
Calculer le montant à payer
Fournir les infos du paiement
demande le paiement
traiter le paiement
par carte de crédit
paiement en espèces
autorizer le paiement

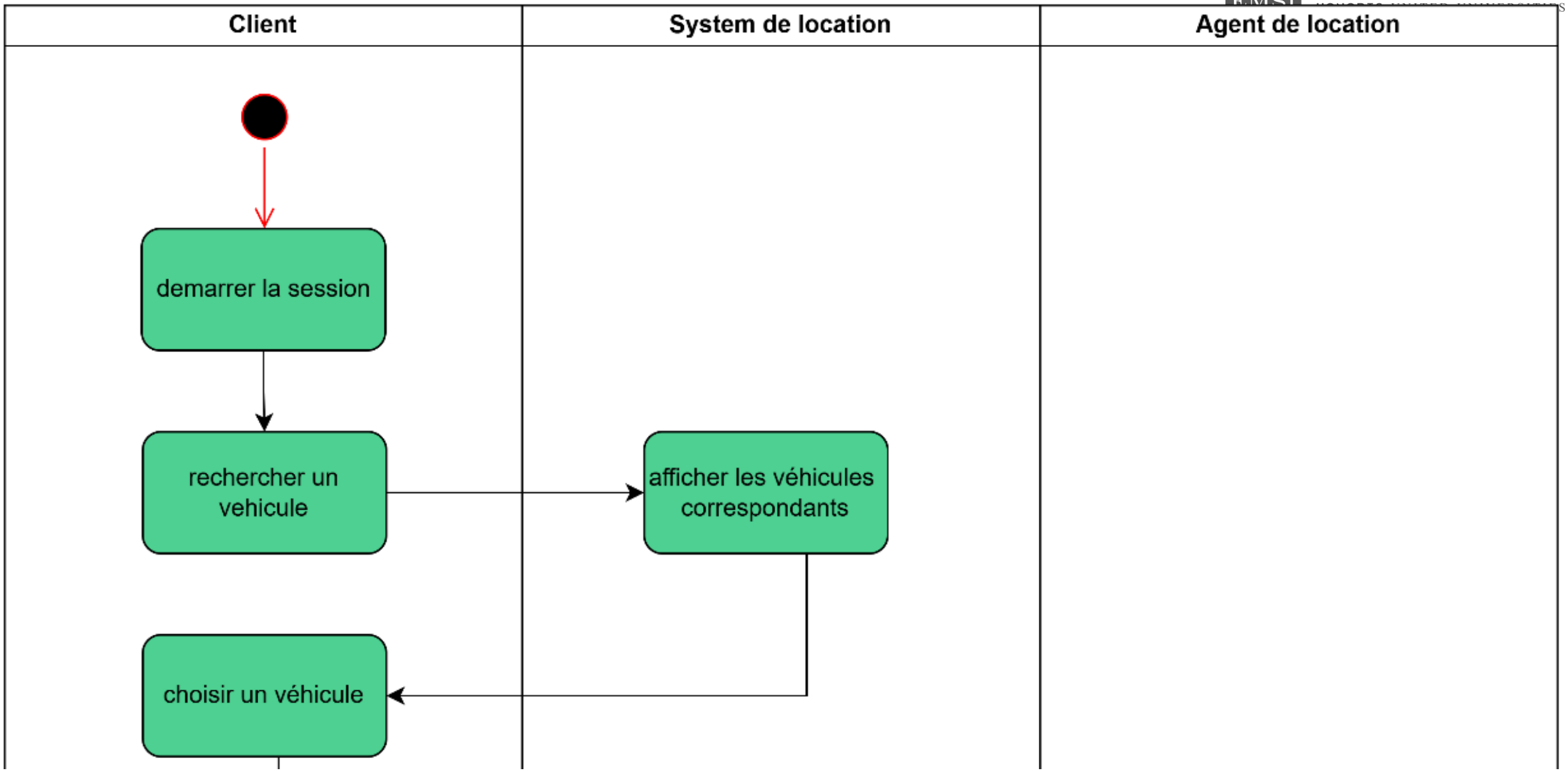

par carte de crédit
autorizer le paiement
paiement en espèces
paiement invalid
paiement valid
tirer billet
billet
gener un billet
payer en espéce
tirer le rest
Pièces de monnaie
rendre la monnaie
Afficher Merci

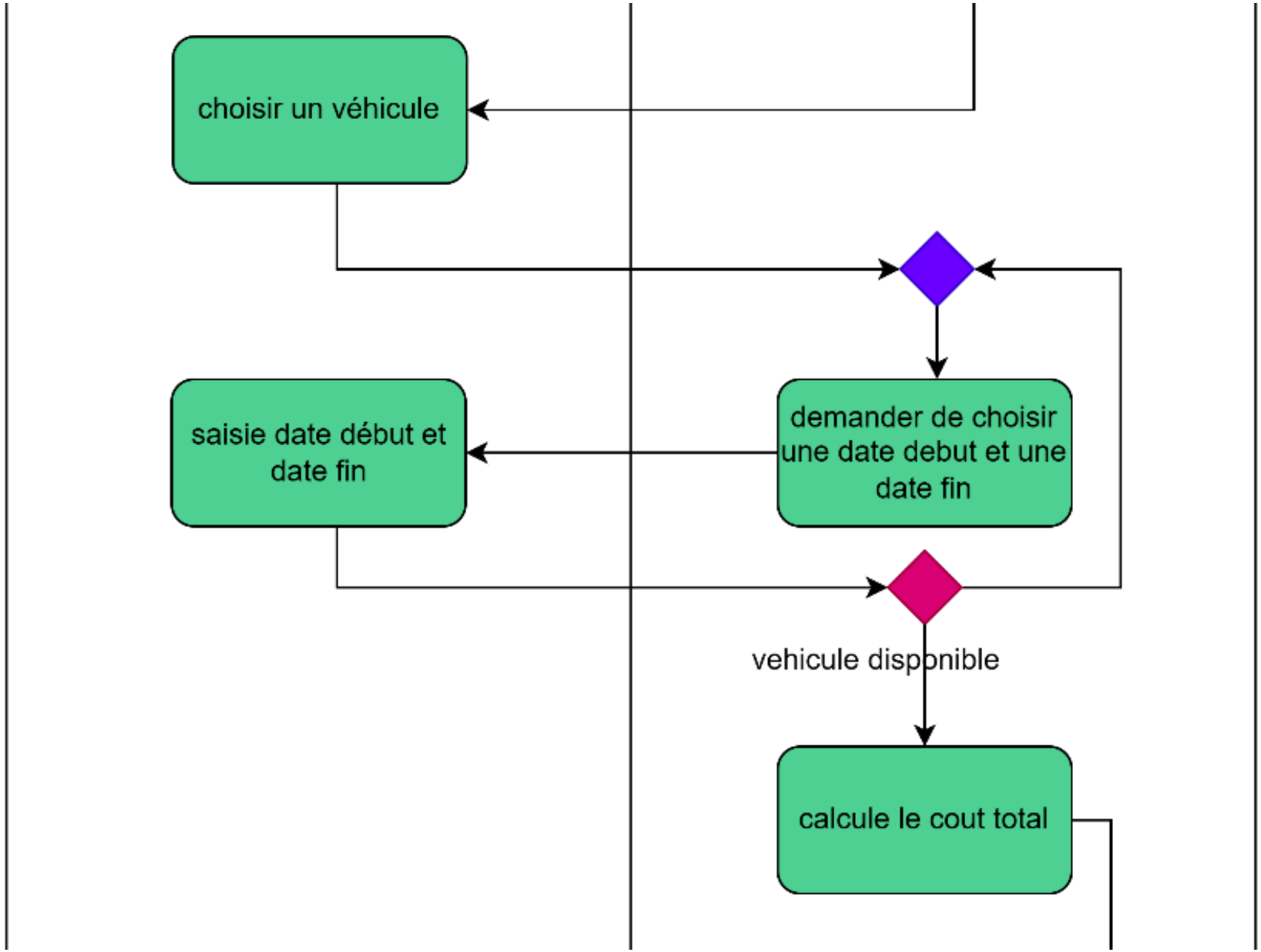
Exercise 3: Système de Location de Véhicules

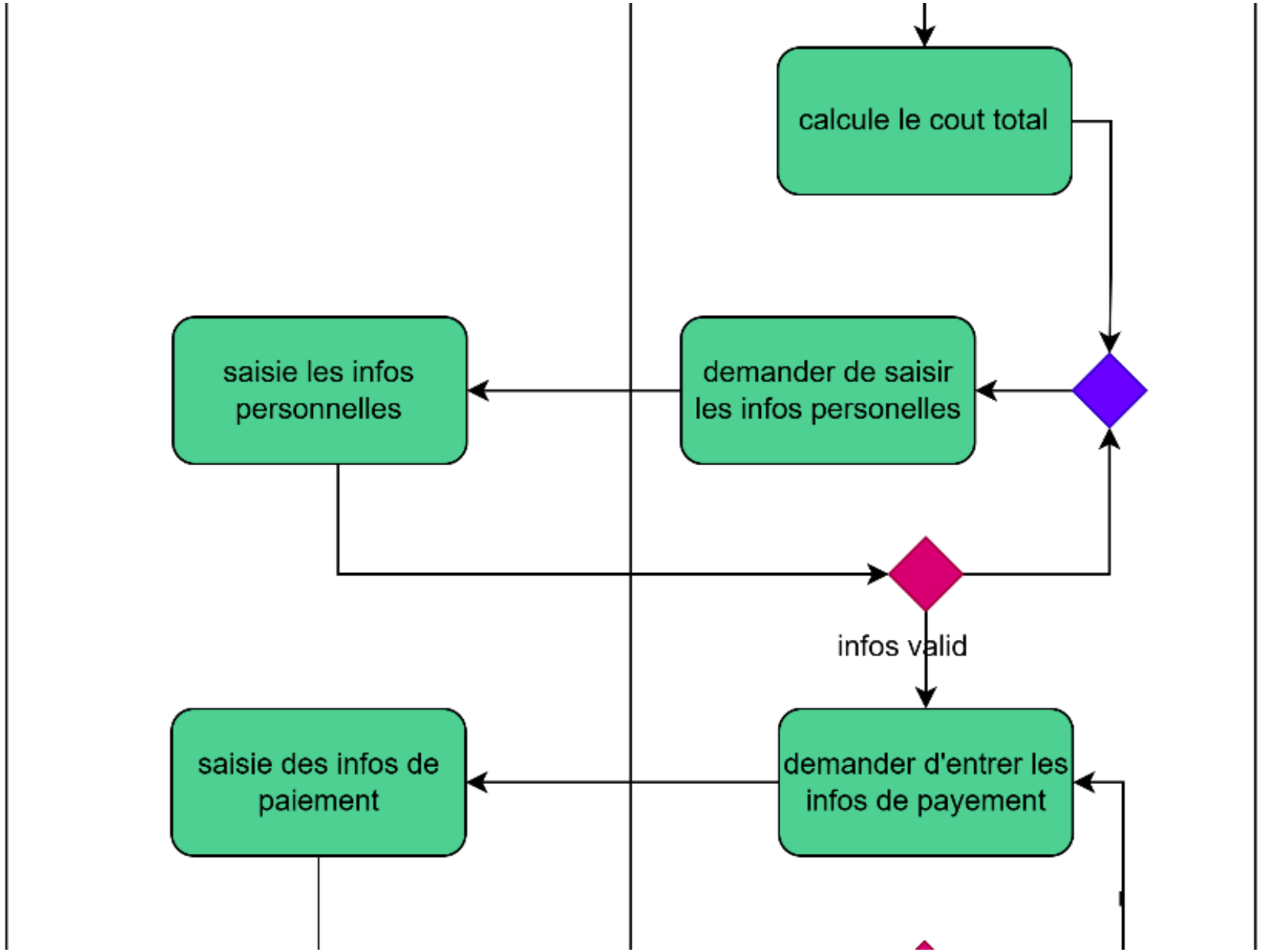
Vous devez modéliser le processus de location d'un véhicule dans une agence de location sous forme de diagramme d'activités UML. Le processus commence lorsque le client recherche un véhicule disponible en fonction de critères spécifiques, tels que le type de véhicule, la marque et le nombre de kilomètres. Le système traite les informations fournies et affiche les véhicules correspondants. Le client choisit un véhicule et saisit les dates de début et de fin de location.

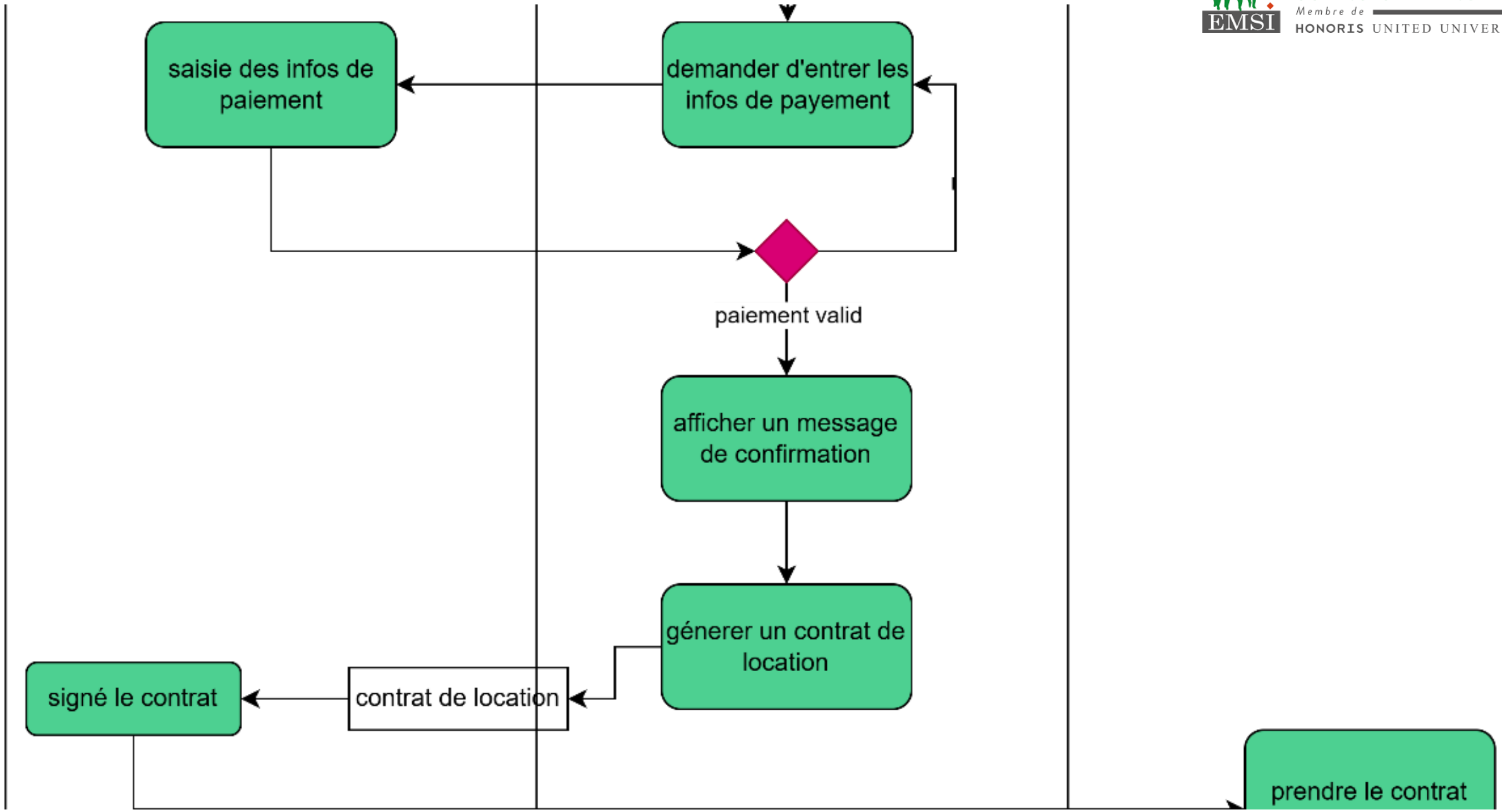
Ensuite, le système calcule le coût total de la location en fonction des dates choisies. Le client saisit ensuite ses informations personnelles et ses informations de paiement. Le système vérifie les informations et, si la validation est réussie, affiche un message de confirmation. Le client signe alors le contrat de location généré par le système et la demande à l'agent de location. Par la suite L'agent de location remet les clés du véhicule au client.

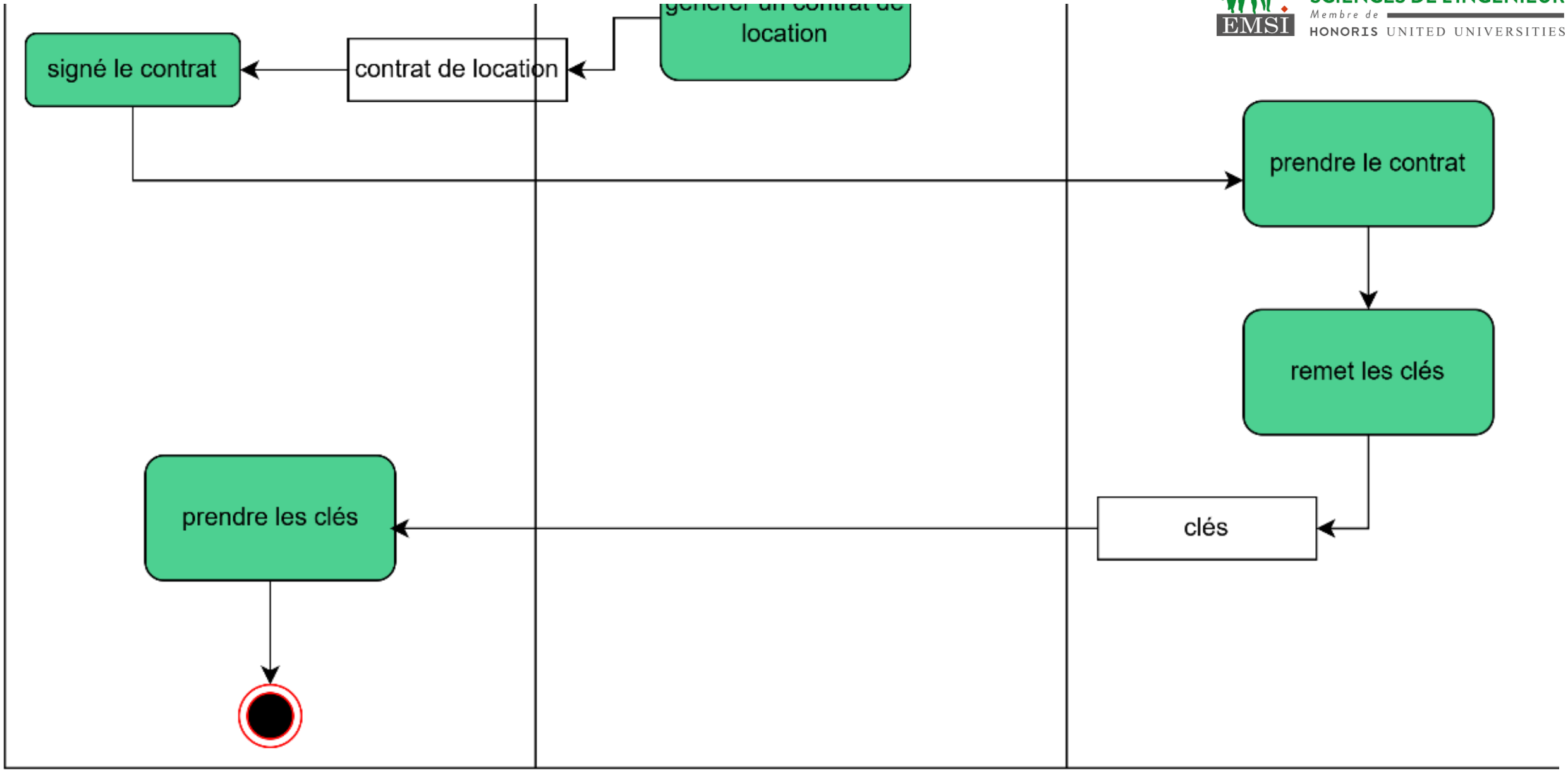

Client
System de location
Agent de location
demarrer la session
rechercher un vehicule
afficher les véhicules correspondants
choisir un véhicule
saisie date début et date fin
demander de choisir une date debut et une date fin
vehicule disponible
calcule le cout total
saisie les infos personnelles
demander de saisir les infos personelles
infos valid
saisie des infos de paiement
demander d'entrer les infos de payement
paiement valid
afficher un message de confirmation
génerer un contrat de location
signé le contrat
contrat de location
prendre le contrat
remet les clés
prendre les clés
clés


Client
System de location
Agent de location
demarrer la session
rechercher un vehicule
afficher les véhicules correspondants
choisir un véhicule


choisir un véhicule
saisie date début et date fin
demander de choisir une date debut et une date fin
vehicule disponible
calcule le cout total


calcule le cout total
saisie les infos personnelles
demander de saisir les infos personelles
infos valid
saisie des infos de paiement
demander d'entrer les infos de payement

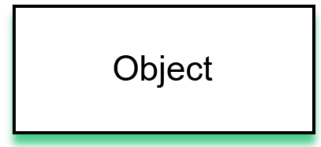

saisie des infos de paiement
demander d'entrer les infos de payement
paiement valid
afficher un message de confirmation
génerer un contrat de location
contrat de location
signé le contrat
prendre le contrat


location
signé le contrat
contrat de location
prendre le contrat
remet les clés
prendre les clés
clés

Definition: est utilisé pour modéliser les interactions entre les objets ou les composants d'un système dans un ordre chronologique. *Il montre comment les objets communiquent entre eux pour réaliser une fonctionnalité spécifique*, en mettant l'accent sur l'ordre chronologique des messages échangés.

Elements:

1.Object : Représente un objet ou un composant participant à l'interaction



- Ex: Client, SystèmeRéservation, Paiement.

2. **Activation (Boîte d'Activation):** Représente la période pendant laquelle un objet exécute une action.

Elements:

1. **Messages:** Représente une communication entre deux objets.

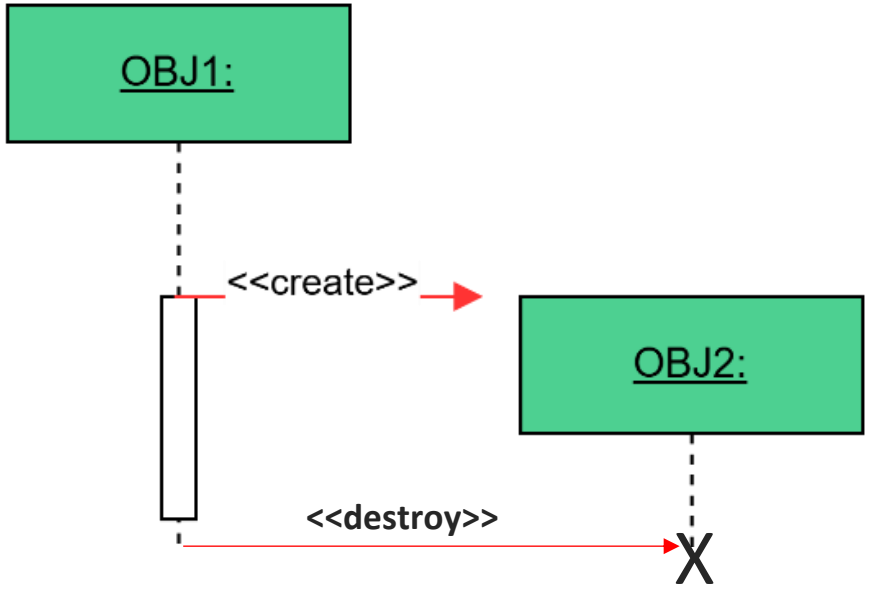
- **Message synchrone:** l'expéditeur attend la fin du traitement avant de continuer.

- **Message asynchrone** : Un message envoyé sans attendre de réponse immédiate. L'expéditeur continue son exécution.

- **Message de retour:** Réponse d'un message synchrone, indiquant la fin du traitement (pointe ouverte ou bien triangulaire).

- **Message de réflexion:** une interaction qu'un objet **a avec lui-même**.

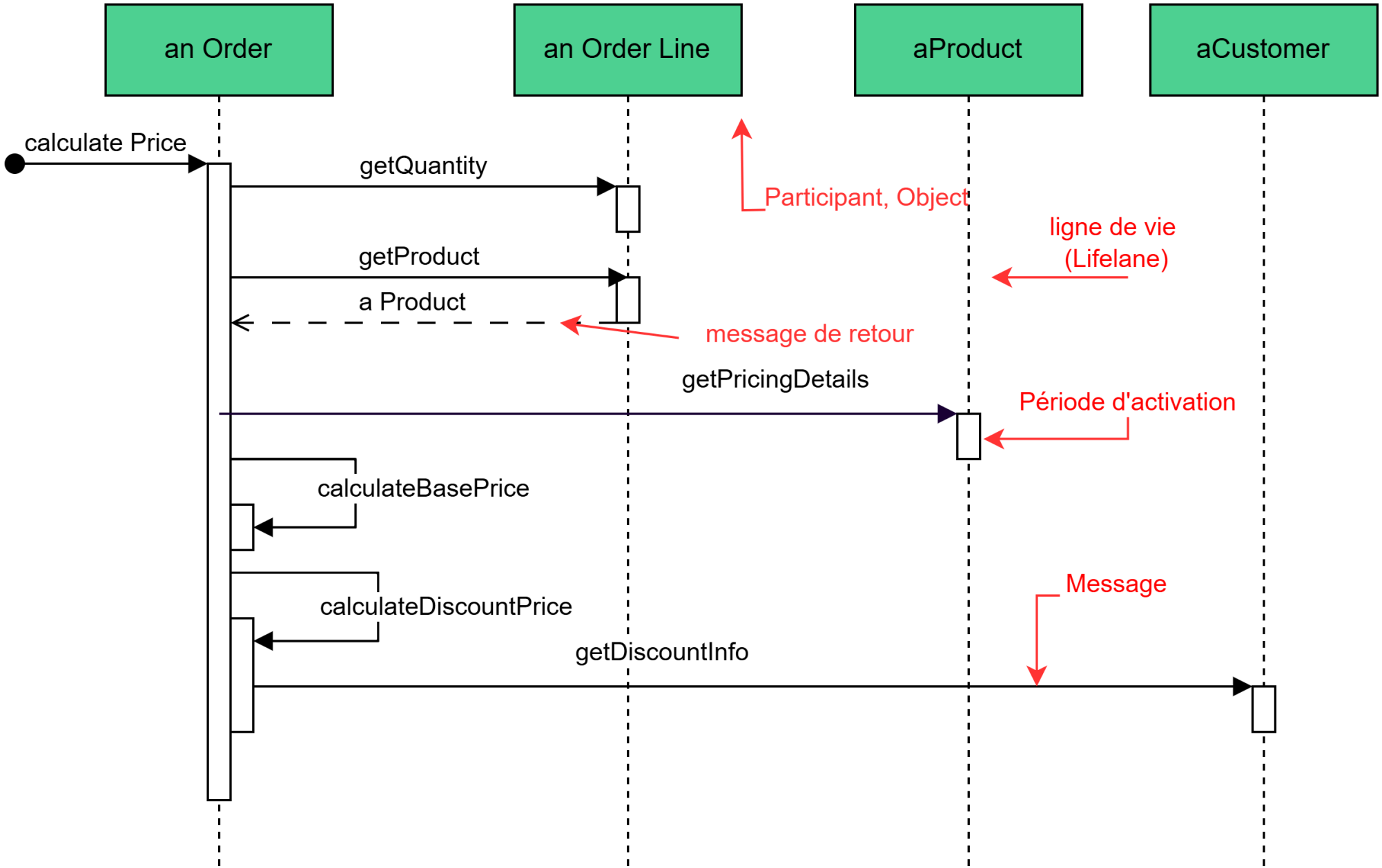
Càd qu'un objet appelle une de ses propres méthodes ou effectue une action interne.



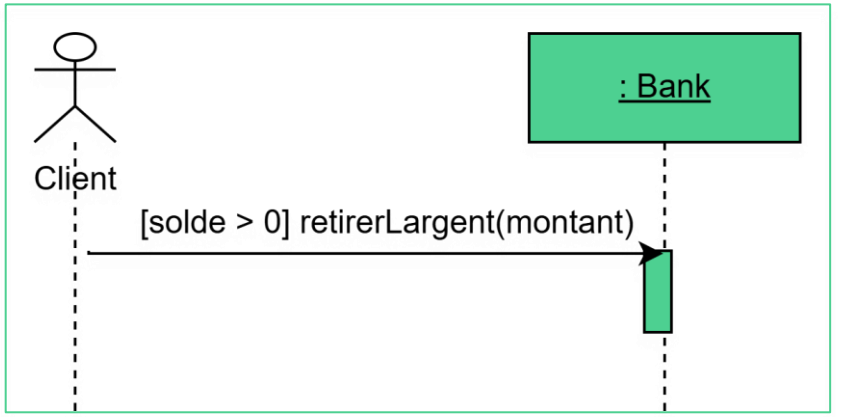
Elements:

- **Message de création:** Un message utilisé pour **instancier un nouvel objet** pendant l'exécution du scénario.
- **Message de destruction**: Un message indiquant la **fin de vie d'un objet**, souvent en raison d'une suppression explicite.

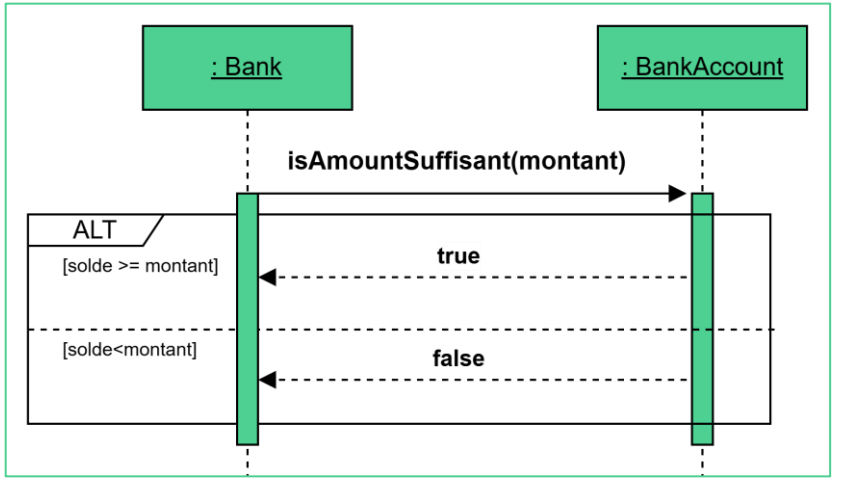
<<Create>> , <<Destroy>> sont deux stéréotypes de messages.


an Order
an Order Line
aProduct
aCustomer
calculate Price
getQuantity
Participant, Object
ligne de vie
(Lifelane)
getProduct
a Product
message de retour
getPricingDetails
Période d'activation
calculateBasePrice
calculateDiscountPrice
Message
getDiscountInfo

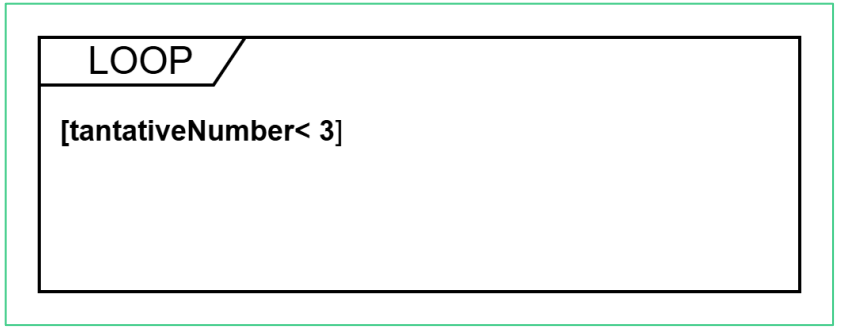
- **Guards:** Un **guard** est une condition booléenne qui détermine si un message peut être envoyé ou non.



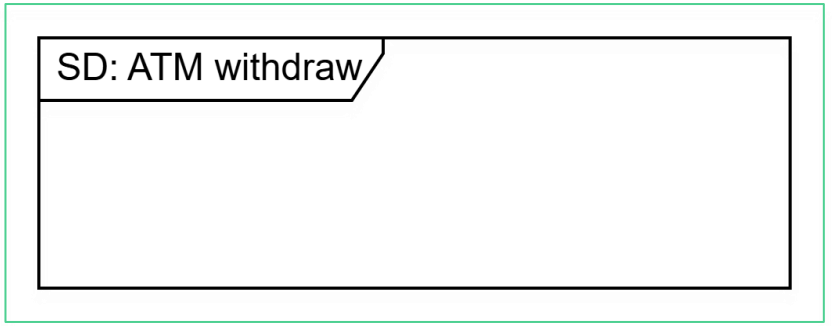
- **Les Alternatives (Bloc ALT):** Une **alternative** permet de modéliser des choix entre plusieurs scénarios en fonction de conditions.



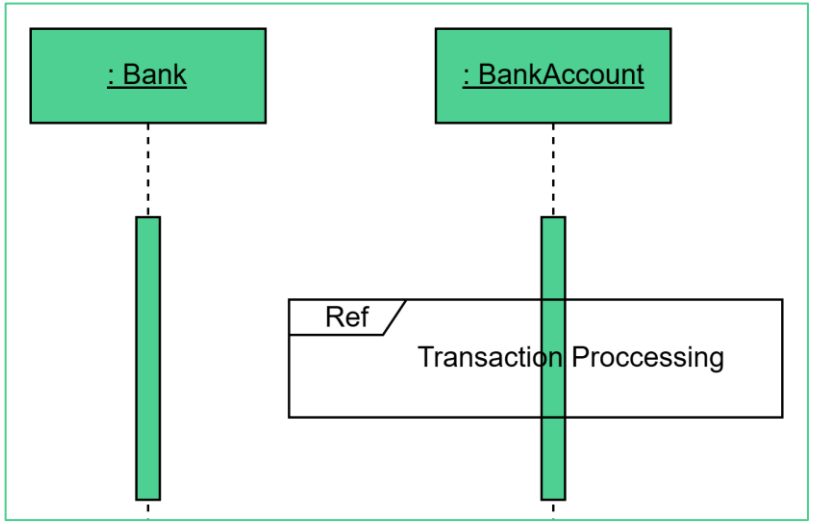
- **Les Loops (Bloc LOOP):** Une **loop** permet de modéliser des répétitions d'un ensemble de messages.



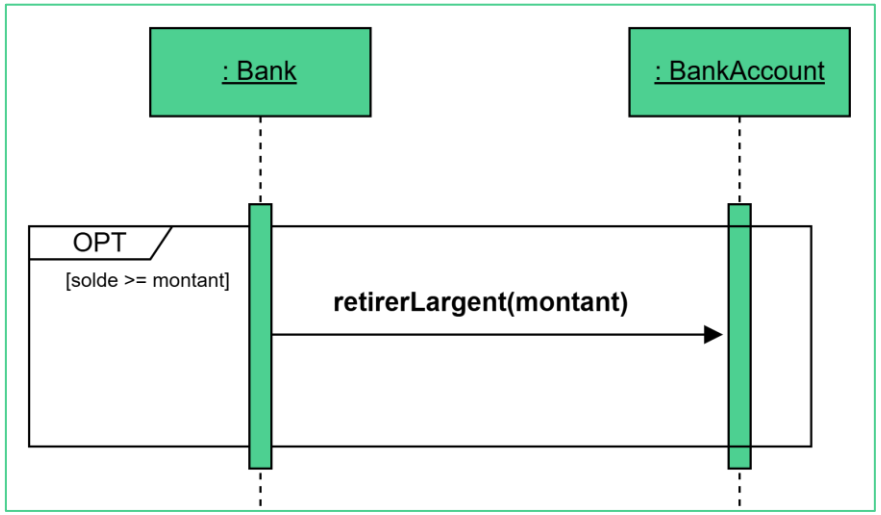
- **Sequence Diagram:** L'ensemble du diagramme de séquence est entouré par un cadre (Frame).



- **Reference:** permet de faire référence à un autre diagramme de séquence ou à un fragment d'interaction existant. Cela évite de dupliquer des interactions complexes.



- **Optional:** utilisé pour représenter une interaction qui ne se produit que si une condition spécifique est remplie, equivalent de **if**.



Deux approches principales pour représenter les interactions:

✓ **Diagram de séquence système (vue global, boîte noire):**

- Cette approche vise à donner une **vue d'ensemble** du système en montrant les interactions entre les **acteurs** (utilisateurs, systèmes externes) et le **système** dans son ensemble.
- Elle se concentre sur les **cas d'utilisation** et les **flux principaux** sans entrer dans les détails techniques (comme les contrôleurs, les entités)…

✓ **Diagram de séquence (vue détaillée, boîte blanche):**

- Cette approche vise à **détailler les interactions** entre les **objets internes** du système (comme les contrôleurs, les entités, les interfaces utilisateur).
- Elle se concentre sur la **logique métier** et les **flux techniques**.

Résumer sur les Deux approches :

Aspect	Vue globale (système)	Vue détaillée (objets)
Niveau de détail	Haut niveau, abstrait.	Bas niveau, technique.
Objets représentés	Acteurs et système (boîte noire).	Objets internes (controleurs, entités).
Public cible	Parties prenantes non techniques.	Développeurs, architectes.
Objectif	Comprendre les cas d'utilisation.	Comprendre la logique interne.
Exemple	Utilisateur -> Système : saisirDonnées()	ObjetA -> ObjetB : appelMéthode()

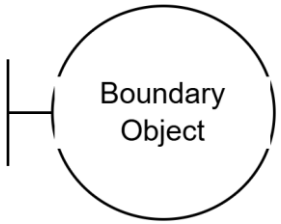
Exercise 1:

Un candidat suit l'état d'une demande de licence et le système affiche les informations relatives à la licence.

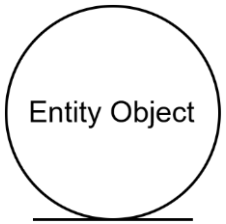
La procédure est la suivante:

1) Le candidat demande à suivre le statut d'une demande de licence.
2) Le système affiche le formulaire de connexion
3) Le candidat soumet les informations de connexion
4) Le système valide le candidat
5) Le système affiche le formulaire de saisie du numéro de suivi
6) Le candidat soumet le numéro de suivi
7) Le système récupère les informations relatives à la licence
8) Le système affiche les informations relatives à la licence

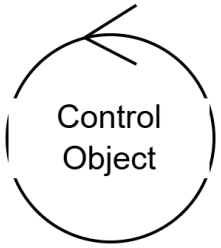
Stéréotypes: ajoutent une couche sémantique supplémentaire pour clarifier la fonction ou la responsabilité d'un objet.



- **<<boundary>>:** Utilisé pour désigner un objet qui fait partie de l'interface du système, est responsable de **gérer les interactions avec l'extérieur du système**.



- **<<entity>>:** utilisé pour désigner des **objets qui représentent des entités métiers** ou **des données persistantes** dans un système. sont responsables de la **gestion des données** et de l'**interaction avec la base de données** .



- **<<control>>:** Utilisé pour désigner un objet qui contient la logique de contrôle ou les règles métier.

Exercise 2:

Vous devez modéliser le processus de connexion d'un utilisateur à une application en utilisant une architecture **MVC (Modèle-Vue-Contrôleur)**. Le processus commence lorsque l'utilisateur saisit son identifiant (email) et son mot de passe dans l'interface de connexion **.**

Le système vérifie d'abord que l'email respecte un format valide (par exemple, xxx@gmail.com) et que le mot de passe n'est pas vide a partir de la vue.

Ensuite, le Contrôleur intervient pour valider les informations de connexion en récupérant les données de l'utilisateur depuis la base de données via le Modèle User, puis compare le mot de passe saisi avec celui stocké. Si les informations sont valides, l'accès est autorisé et l'utilisateur est redirigé vers la page d'accueil, sinon, un message d'erreur est affiché.

Modélisez ce processus en identifiant clairement les composants Vue, Contrôleur et Modèle, et en détaillant les interactions entre eux.

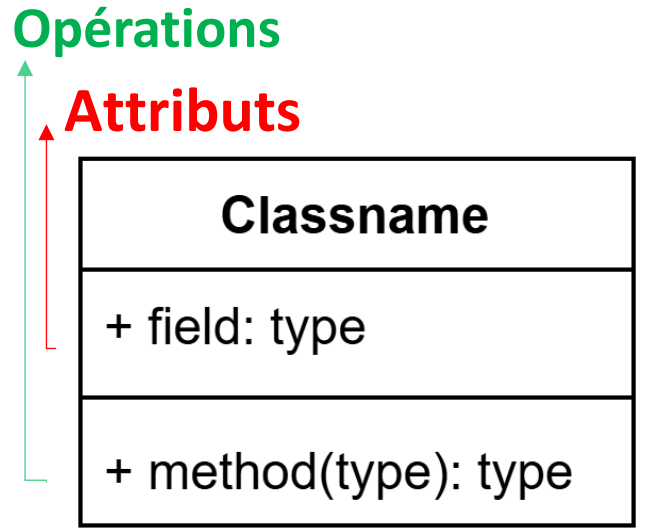
Exercise 3:

VOIR LE PDF

Definition: Un diagramme de classes est une représentation statique de la structure d'un système, Il décrit les classes, leurs attributs, méthodes, et les relations entre elles.

➢ **Classe:** une classe est un modèle qui sert à créer des objets. Définit Par:

- **Les attributs** : ce sont les données ou caractéristiques de l'objet.
- **Les méthodes** : ce sont les actions ou comportements que l'objet peut réaliser.



Visibilité des Attributs et Méthodes:

+ Public (accessible par toutes les classes).

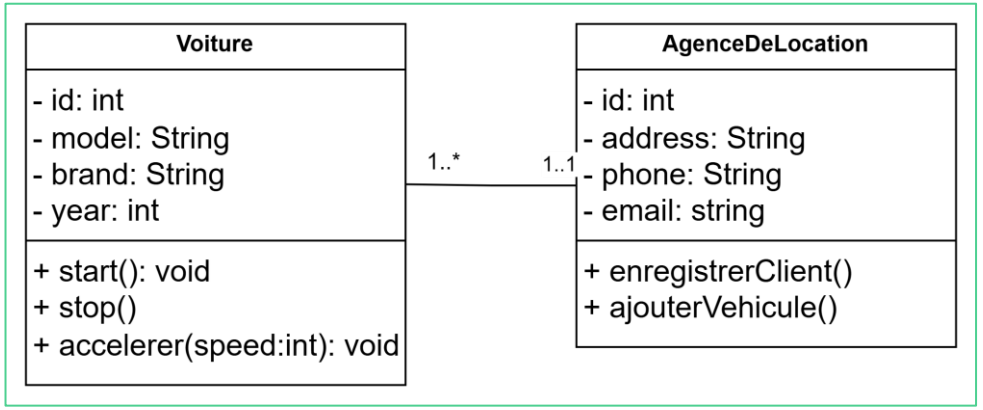
- Privé (accessible uniquement dans la classe).

 # Protégé (accessible dans la classe et ses sous-classes).

~ (package/default visibility)

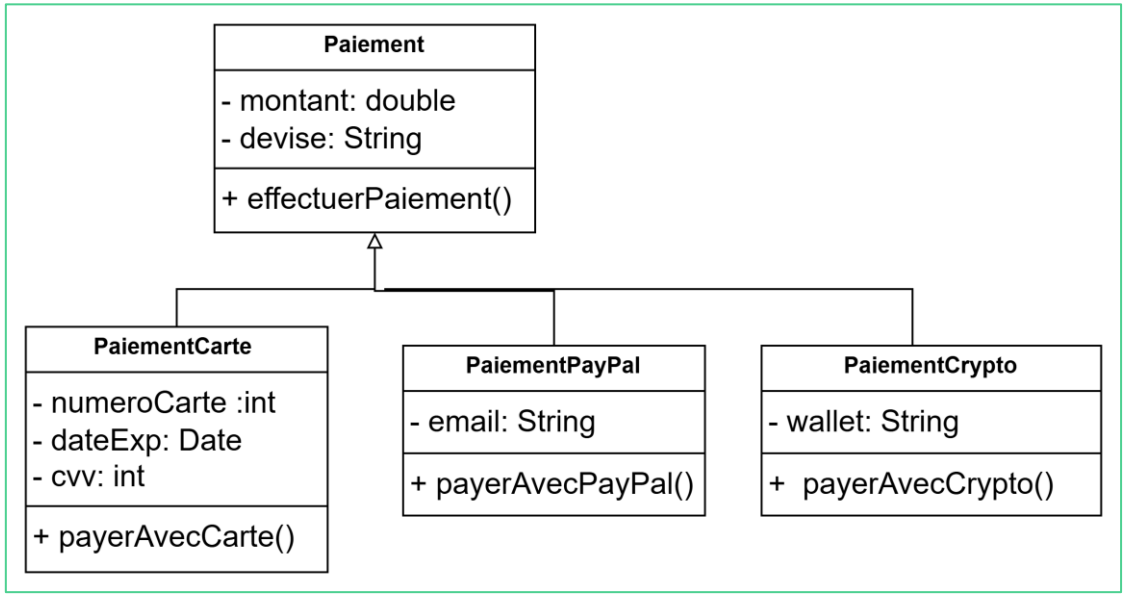
➢ Relations entre les Classes:

- **Association** : représente un lien bidirectionnelle entre deux classes.



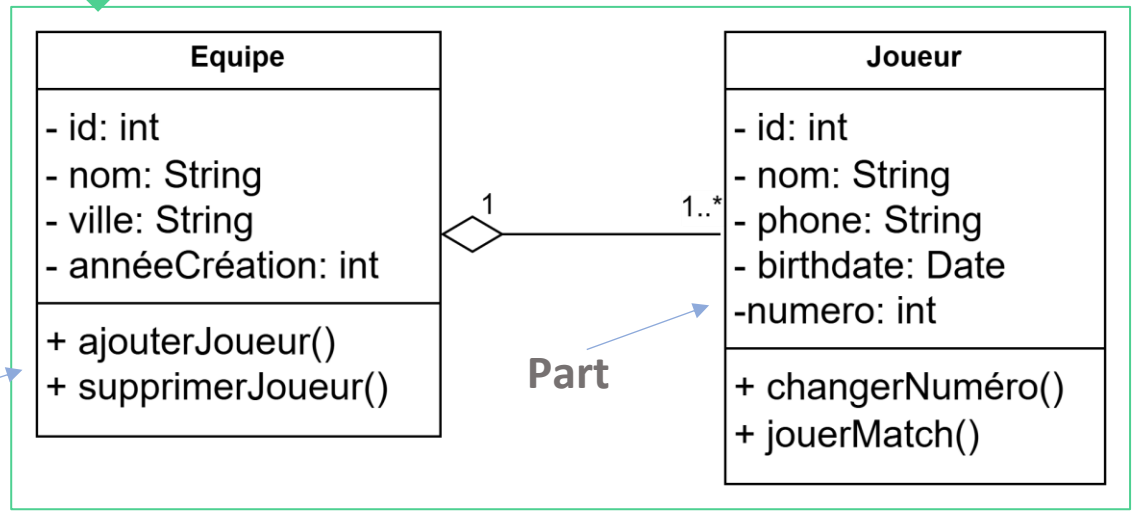
- **Héritage (Généralisation):** L'héritage représente une relation "est-un" entre une classe parente (générale) et une classe enfant (spécifique).

La classe enfant hérite des attributs et méthodes de la classe parente



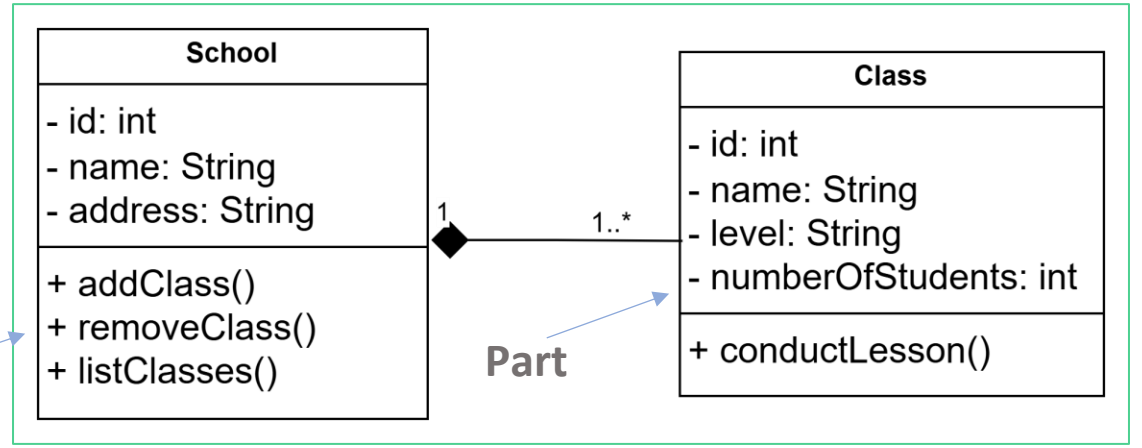
- **Agrégation** : L'agrégation représente une relation "part-of" où une classe contient une ou plusieurs instances d'une autre classe, mais ces instances peuvent exister indépendamment.

Couplage : Faible.



- **Composition:** La composition représente une relation "part-of" ***forte*** où une classe est composée d'une ou plusieurs instances d'une autre classe, et ces instances *ne peuvent pas exister indépendamment*.

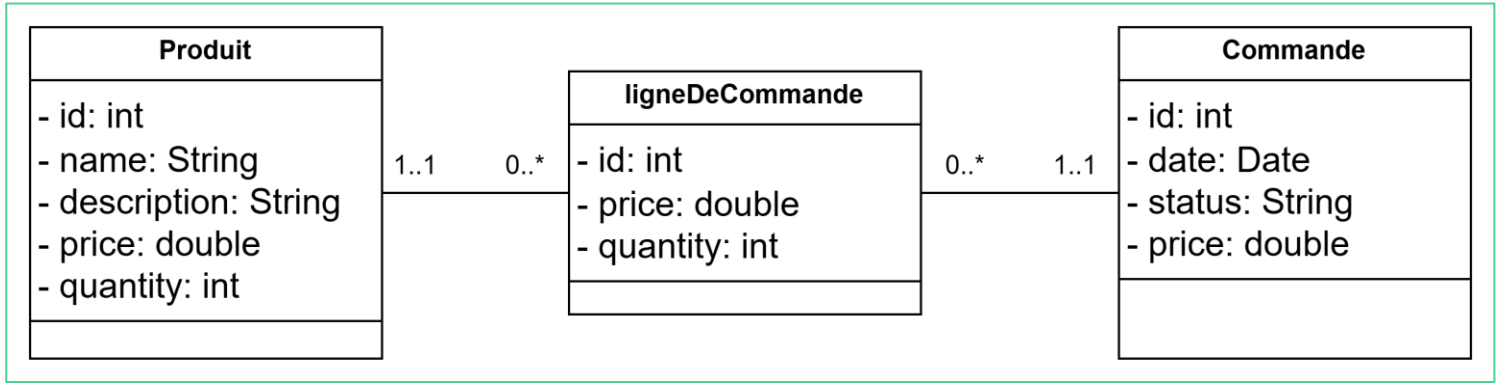
Couplage : Fort.

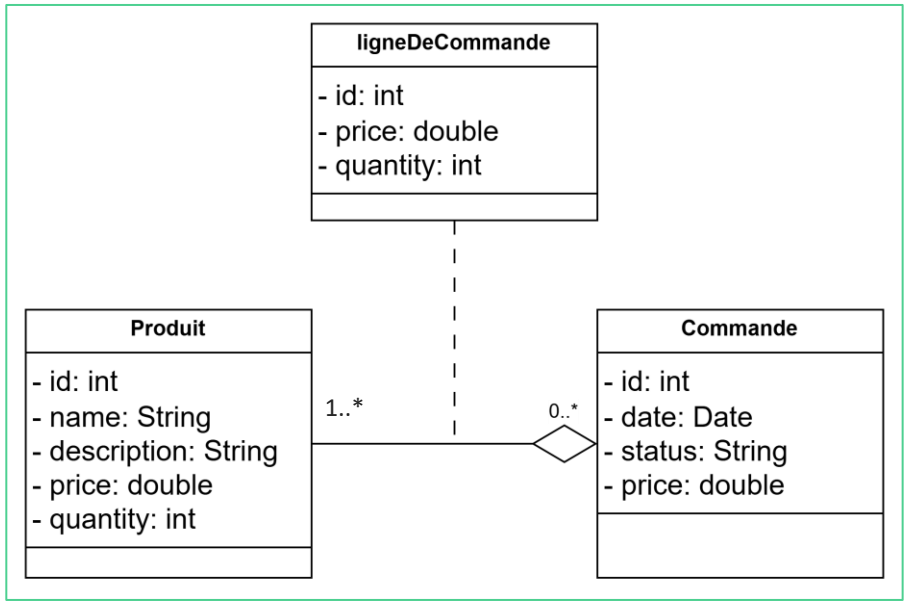


Association: Une **relation d'association** en UML représente un **lien entre deux classes** indiquant qu'elles interagissent entre elles.

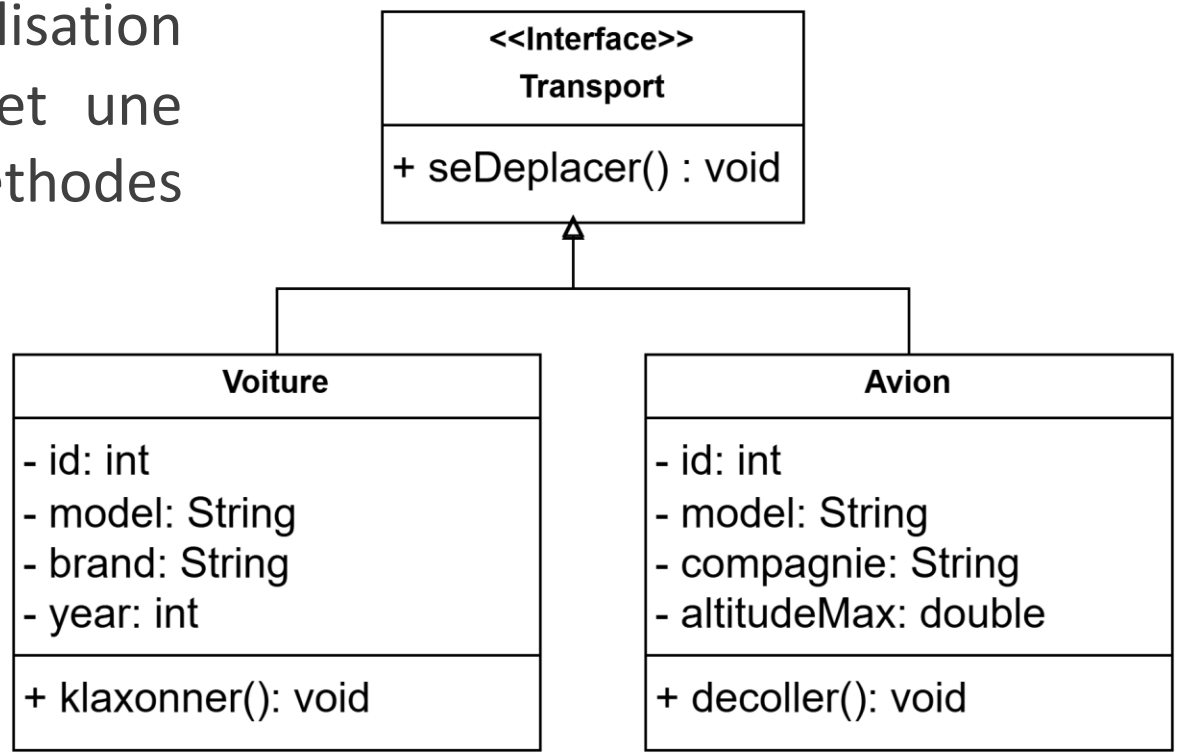
Elle est **utilisée** lorsque l'on a une relation **"Many-to-Many"**, c'est-à-dire qu'un objet de la première classe peut être lié à plusieurs objets de la seconde classe et inversement.

❖ Deux méthodes pour représenter les classes d'associations.

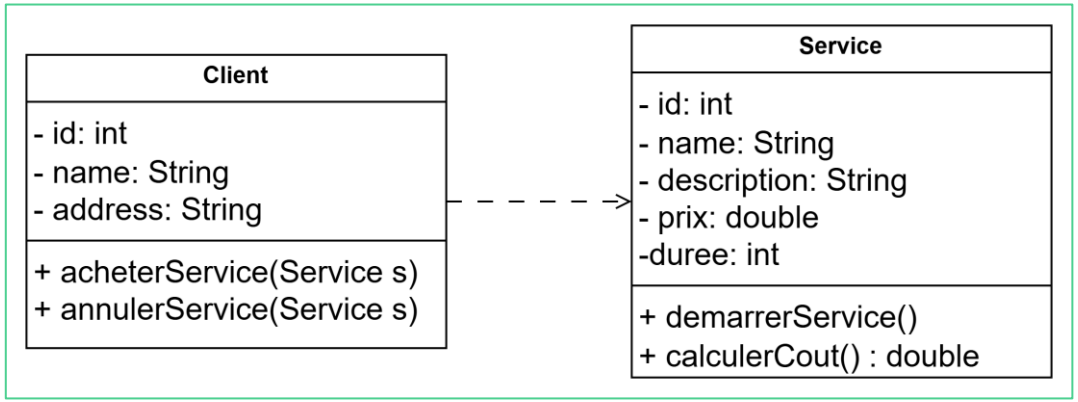




- **Réalisation (Implémentation):** La réalisation représente la relation entre une classe et une interface, La classe implémente les méthodes définies dans l'interface.



- **Dépendance (Dependency)**: Une dépendance indique qu'une classe utilise une autre classe de manière temporaire ou indirecte.
- Cela signifie qu'un changement dans une classe peut affecter l'autre classe, mais il n'y a pas de lien structurel fort entre elles.
- **Quand l'utiliser ?** Lorsqu'une classe utilise une autre classe comme:
  - Un paramètre de méthode.
  - Une variable locale dans une méthode.
  - Un retour de méthode.



Système de Gestion de Commandes en Ligne

Une entreprise souhaite développer un **système de gestion de commandes en ligne** permettant aux utilisateurs de parcourir des produits, de passer des commandes et de gérer leurs comptes.

- **Gestion des Produits:** Un Produit est défini par un id, un nom, une description, un prix et une quantité disponible en stock. Chaque produit appartient à **une seule Catégorie**, mais une catégorie peut contenir plusieurs produits.
- **Gestion des Utilisateurs:** Il existe deux types d'utilisateurs *Administrateur, Client*. Ayant tous Les deux un id, un nom, un email et un mot de passe. En plus pour le client une adresse et un téléphone.
- **Gestion des Commandes:** Une Commande est caractérisée par un id, une date, un statut et un prix total et peut contenir plusieurs Produits, chaque client peut passer plusieurs commandes.