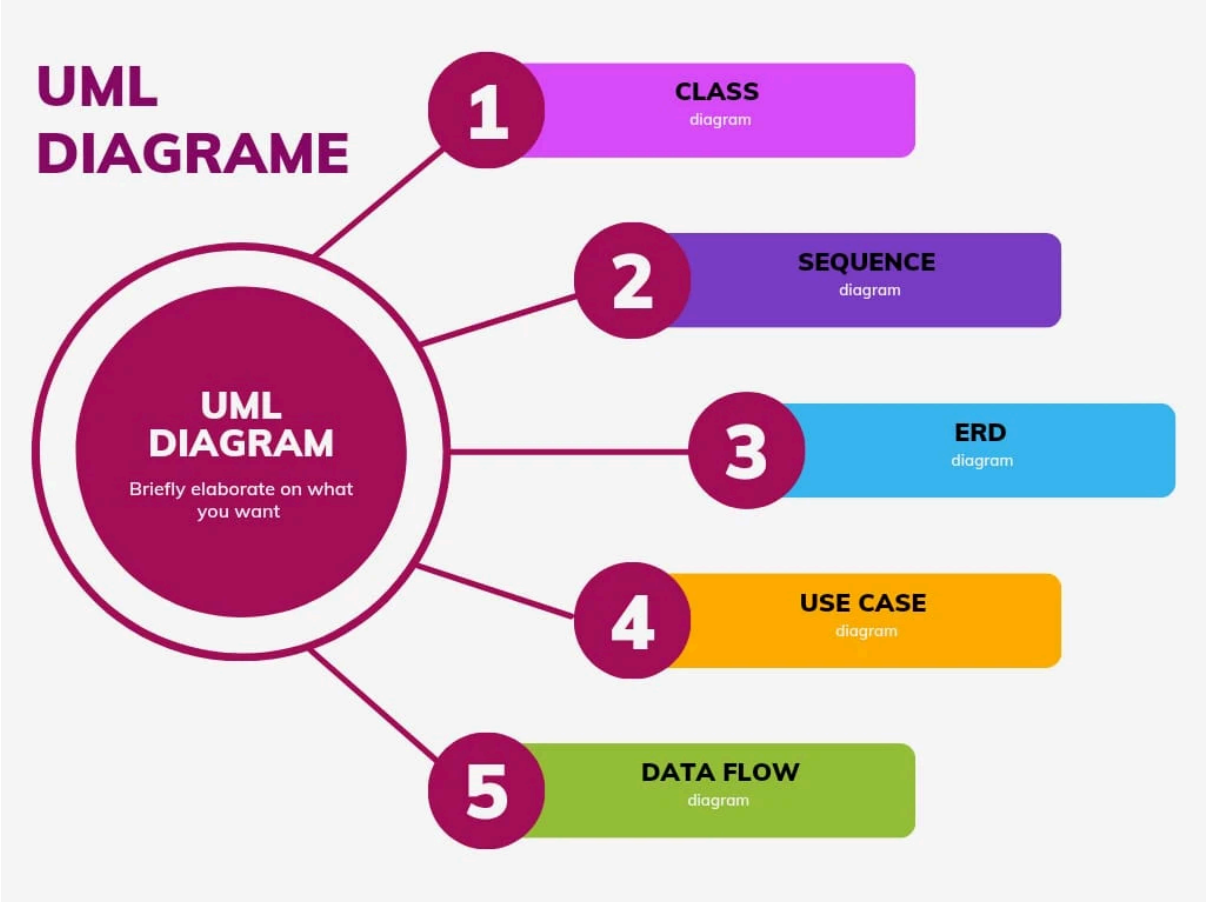
Kit d’exercices sur UML

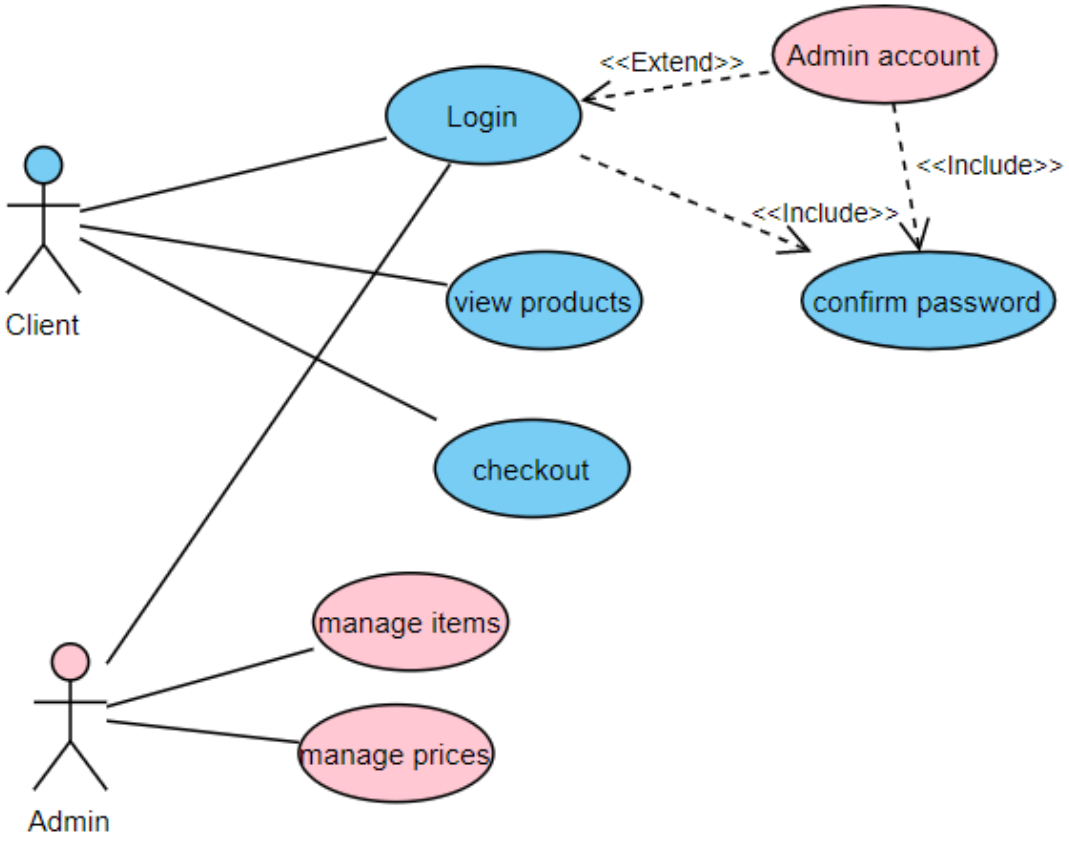


Chers étudiants,

Bienvenue à cette série d'exercices sur la modélisation UML (Unified Modeling Language). Dans ces exercices, nous allons explorer deux types de diagrammes fondamentaux : les diagrammes de cas d'utilisation (UML Use Case Diagrams) et les diagrammes de classes (UML Class Diagrams). Ces outils vous permettront de mieux comprendre et concevoir les systèmes logiciels de manière structurée et claire. Pour ces exercices, nous utiliserons le logiciel UMLet, un outil pratique et intuitif pour créer des diagrammes UML. Assurez-vous de bien suivre les instructions fournies et de modéliser chaque aspect avec précision.

Bonne modélisation !

LE DIAGRAMME DE CAS D'UTILISATION (USE CASE)



1. Système de Gestion de Bibliothèque

Contexte :

Vous êtes un analyste junior chez **BiblioSoft Solutions**, une entreprise spécialisée dans le développement de logiciels pour les bibliothèques. Votre mission est de modéliser un nouveau système de gestion pour la bibliothèque municipale. Ce système doit permettre aux utilisateurs de rechercher des livres, emprunter des livres, retourner des livres, s'inscrire à la bibliothèque, et recevoir des notifications sur les nouveaux livres et les rappels de retour.

Les utilisateurs du système incluent les membres de la bibliothèque (utilisateurs) et le personnel de la bibliothèque (bibliothécaires). Vous devez créer un diagramme de cas d'utilisation pour représenter ces interactions.

Tâches à réaliser :

1. **Créer un diagramme de cas d'utilisation pour le système de gestion de bibliothèque en utilisant UMLet.**

Étapes à suivre :

1. **Ouvrir UMLet et créer un nouveau diagramme de cas d'utilisation.**
2. **Définir les acteurs principaux :**
    - **Utilisateur** : Un membre de la bibliothèque qui peut rechercher, emprunter, et retourner des livres, s'inscrire à la bibliothèque et recevoir des notifications.
    - **Bibliothécaire** : Un membre du personnel de la bibliothèque avec des responsabilités supplémentaires, comme la gestion des livres et la vérification des retours.
3. **Ajouter les cas d'utilisation suivants :**

- **Rechercher des livres** : L'utilisateur peut rechercher des livres dans le catalogue de la bibliothèque.
- **Emprunter des livres** : L'utilisateur peut emprunter des livres disponibles.
- **Retourner des livres** : L'utilisateur peut retourner les livres empruntés.
- **S'inscrire à la bibliothèque** : Une personne peut s'inscrire pour devenir membre de la bibliothèque.
- **Recevoir des notifications** : L'utilisateur peut recevoir des notifications sur les nouveaux livres et les rappels de retour.

4. **Relier les acteurs aux cas d'utilisation appropriés :**
   - L'utilisateur doit être lié à tous les cas d'utilisation mentionnés.
   - Le bibliothécaire doit également être lié à ces cas d'utilisation, en plus de pouvoir gérer le système.
5. **Ajouter une relation de généralisation entre "Utilisateur" et "Bibliothécaire" :**
   - Le bibliothécaire est un type d'utilisateur avec des responsabilités supplémentaires.

Conseils pour la réalisation :

Lorsque vous créez votre diagramme de cas d'utilisation, assurez-vous de bien comprendre les rôles et les responsabilités de chaque acteur.

Utilisez des noms clairs et descriptifs pour chaque cas d'utilisation afin de rendre le diagramme facile à comprendre. Pensez à la manière dont chaque action interagit avec le système global et représentez ces interactions avec précision. N'oubliez pas que la relation de généralisation montre que le bibliothécaire peut effectuer toutes les actions d'un utilisateur standard, ainsi que des tâches supplémentaires liées à la gestion de la bibliothèque. Prenez le temps de bien organiser votre diagramme pour qu'il soit propre et lisible.

Bonne chance !

2. Système de Gestion de Restaurant

Contexte :

Vous travaillez comme analyste junior chez **GourmetSoft Solutions**, une entreprise spécialisée dans le développement de logiciels pour la restauration. Votre mission est de modéliser un nouveau système de gestion pour un restaurant. Ce système doit permettre aux clients de réserver une table, commander des plats, payer l'addition, et recevoir des notifications sur les offres spéciales.

Les utilisateurs du système incluent les clients et le personnel du restaurant (serveurs et chefs). Vous devez créer un diagramme de cas d'utilisation pour représenter ces interactions.

Tâches à réaliser :

1. **Créer un diagramme de cas d'utilisation pour le système de gestion de restaurant en utilisant UMLet.**

Étapes à suivre :

1. **Ouvrir UMLet et créer un nouveau diagramme de cas d'utilisation.**
2. **Définir les acteurs principaux :**
    - **Client** : Un client du restaurant qui peut réserver une table, commander des plats, payer l'addition et recevoir des notifications.
    - **Serveur** : Un membre du personnel du restaurant qui prend les commandes, sert les plats, et gère les paiements.
    - **Chef** : Un membre du personnel du restaurant qui prépare les plats commandés.
3. **Ajouter les cas d'utilisation suivants :**
    - **Réserver une table** : Le client peut réserver une table dans le restaurant.

- **Commander des plats** : Le client peut commander des plats disponibles au menu.
- **Payer l'addition** : Le client peut payer l'addition après avoir consommé les plats.
- **Recevoir des notifications** : Le client peut recevoir des notifications sur les offres spéciales et les nouveaux plats.

4. **Relier les acteurs aux cas d'utilisation appropriés :**
   - Le client doit être lié à tous les cas d'utilisation mentionnés.
   - Le serveur doit être lié aux cas d'utilisation "Commander des plats" et "Payer l'addition".
   - Le chef doit être lié au cas d'utilisation "Commander des plats".

3.Système de Gestion de d'Hôpital

Contexte :

Vous travaillez comme analyste junior chez **MediSoft Solutions**, une entreprise spécialisée dans le développement de logiciels pour les hôpitaux. Votre mission est de modéliser un nouveau système de gestion pour un hôpital. Ce système doit permettre aux patients de prendre des rendez-vous, consulter leurs dossiers médicaux, et recevoir des notifications sur leurs rendez-vous. Le personnel médical (médecins et infirmières) doit pouvoir gérer les rendez-vous et consulter les dossiers médicaux des patients.

Tâches à réaliser :

1. **Créer un diagramme de cas d'utilisation pour le système de gestion d'hôpital en utilisant UMLet.**

Étapes à suivre :

1. **Ouvrir UMLet et créer un nouveau diagramme de cas d'utilisation.**
2. **Définir les acteurs principaux :**
    - **Patient** : Un patient de l'hôpital qui peut prendre des rendez-vous, consulter ses dossiers médicaux et recevoir des notifications.
    - **Médecin** : Un membre du personnel médical qui peut gérer les rendez-vous et consulter les dossiers médicaux des patients.
    - **Infirmière** : Un membre du personnel médical qui peut gérer les rendez-vous et consulter les dossiers médicaux des patients.
3. **Ajouter les cas d'utilisation suivants :**
    - **Prendre un rendez-vous** : Le patient peut prendre un rendez-vous médical.

- **Consulter les dossiers médicaux** : Le patient peut consulter ses dossiers médicaux.
- **Gérer les rendez-vous** : Le médecin et l'infirmière peuvent gérer les rendez-vous des patients.
- **Consulter les dossiers médicaux des patients** : Le médecin et l'infirmière peuvent consulter les dossiers médicaux des patients.
- **Recevoir des notifications** : Le patient peut recevoir des notifications sur ses rendez-vous.

4. **Relier les acteurs aux cas d'utilisation appropriés :**
   - Le patient doit être lié aux cas d'utilisation "Prendre un rendez-vous", "Consulter les dossiers médicaux", et "Recevoir des notifications".
   - Le médecin et l'infirmière doivent être liés aux cas d'utilisation "Gérer les rendez-vous" et "Consulter les dossiers médicaux des patients".

4. Système de Gestion de Projet

Contexte :

Vous êtes analyste chez **ProjectPlus Inc.**, une entreprise spécialisée dans les outils de gestion de projet pour les grandes organisations. Votre mission est de modéliser un nouveau système de gestion de projet. Ce système doit permettre aux utilisateurs de créer des projets, ajouter des tâches, assigner des tâches aux membres de l'équipe, suivre l'avancement des tâches, et générer des rapports de projet.

Les utilisateurs du système incluent les chefs de projet, les membres de l'équipe et les administrateurs système. Vous devez créer un diagramme de cas d'utilisation pour représenter ces interactions.

Tâches à réaliser :

1. **Créer un diagramme de cas d'utilisation pour le système de gestion de projet en utilisant UMLet.**

Étapes à suivre :

1. **Ouvrir UMLet et créer un nouveau diagramme de cas d'utilisation.**
2. **Définir les acteurs principaux :**
    - **Chef de Projet** : Un utilisateur qui peut créer des projets, ajouter et assigner des tâches, suivre l'avancement des tâches, et générer des rapports.
    - **Membre de l'Équipe** : Un utilisateur qui peut consulter les tâches assignées, mettre à jour l'état des tâches, et consulter les rapports de projet.
    - **Administrateur Système** : Un utilisateur qui peut gérer les comptes utilisateurs et configurer les paramètres du système.
3. **Ajouter les cas d'utilisation suivants :**

- **Créer un projet** : Le chef de projet peut créer un nouveau projet.
- **Ajouter des tâches** : Le chef de projet peut ajouter des tâches à un projet.
- **Assigner des tâches** : Le chef de projet peut assigner des tâches aux membres de l'équipe.
- **Suivre l'avancement des tâches** : Le chef de projet et les membres de l'équipe peuvent suivre l'avancement des tâches.
- **Générer des rapports** : Le chef de projet et les membres de l'équipe peuvent générer des rapports de projet.
- **Gérer les comptes utilisateurs** : L'administrateur système peut gérer les comptes utilisateurs.
- **Configurer les paramètres du système** : L'administrateur système peut configurer les paramètres du système.

4. **Relier les acteurs aux cas d'utilisation appropriés :**
   - Le chef de projet doit être lié à tous les cas d'utilisation excepté ceux de l'administrateur système.
   - Le membre de l'équipe doit être lié aux cas d'utilisation "Suivre l'avancement des tâches" et "Générer des rapports".
   - L'administrateur système doit être lié aux cas d'utilisation "Gérer les comptes utilisateurs" et "Configurer les paramètres du système".

5. Système de Commerce Électronique

Contexte :

Vous travaillez comme analyste chez **E-Shop Solutions**, une entreprise spécialisée dans les plateformes de commerce électronique. Votre mission est de modéliser un nouveau système de gestion pour une plateforme de commerce électronique. Ce système doit permettre aux utilisateurs de créer un compte, naviguer dans les produits, ajouter des produits au panier, passer des commandes, effectuer des paiements, et suivre les commandes. Les administrateurs doivent pouvoir gérer les produits, les utilisateurs et les commandes.

Tâches à réaliser :

1. **Créer un diagramme de cas d'utilisation pour le système de commerce électronique en utilisant UMLet.**

Étapes à suivre :

1. **Ouvrir UMLet et créer un nouveau diagramme de cas d'utilisation.**
2. **Définir les acteurs principaux :**
    - **Client** : Un utilisateur qui peut créer un compte, naviguer dans les produits, ajouter des produits au panier, passer des commandes, effectuer des paiements, et suivre les commandes.
    - **Administrateur** : Un utilisateur qui peut gérer les produits, les utilisateurs et les commandes.
3. **Ajouter les cas d'utilisation suivants :**
    - **Créer un compte** : Le client peut créer un nouveau compte utilisateur.
    - **Naviguer dans les produits** : Le client peut parcourir les produits disponibles sur le site.

- **Ajouter des produits au panier** : Le client peut ajouter des produits à son panier d'achat.
- **Passer une commande** : Le client peut passer une commande pour les produits dans son panier.
- **Effectuer un paiement** : Le client peut payer pour les produits commandés.
- **Suivre les commandes** : Le client peut suivre l'état de ses commandes.
- **Gérer les produits** : L'administrateur peut ajouter, modifier ou supprimer des produits.
- **Gérer les utilisateurs** : L'administrateur peut gérer les comptes utilisateurs.
- **Gérer les commandes** : L'administrateur peut gérer les commandes passées par les clients.

4. **Relier les acteurs aux cas d'utilisation appropriés :**
   - Le client doit être lié à tous les cas d'utilisation sauf ceux de l'administrateur.
   - L'administrateur doit être lié aux cas d'utilisation "Gérer les produits", "Gérer les utilisateurs", et "Gérer les commandes".

6. Système de Réservation de Vols

Contexte :

Vous travaillez comme analyste chez **AeroSoft Systems**, une entreprise spécialisée dans les systèmes de réservation de vols pour les compagnies aériennes. Votre mission est de modéliser un nouveau système de réservation de vols. Ce système doit permettre aux utilisateurs de rechercher des vols, réserver des billets, annuler des réservations, consulter les détails des vols, et recevoir des notifications. Les agents de voyage doivent pouvoir gérer les réservations et les utilisateurs.

Tâches à réaliser :

1. **Créer un diagramme de cas d'utilisation pour le système de réservation de vols en utilisant UMLet.**

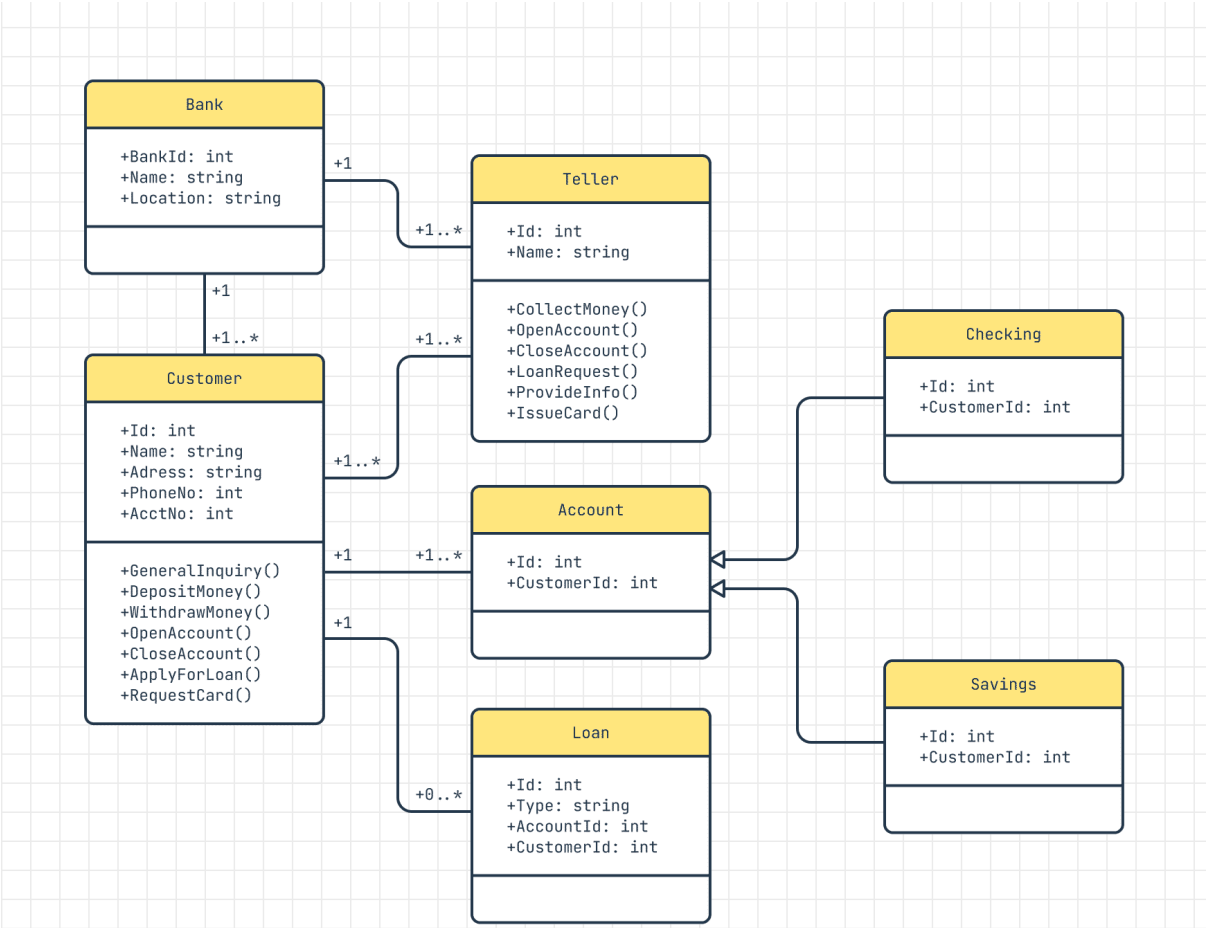
Étapes à suivre :

1. **Ouvrir UMLet et créer un nouveau diagramme de cas d'utilisation.**
2. **Définir les acteurs principaux :**
    - **Client** : Un utilisateur qui peut rechercher des vols, réserver des billets, annuler des réservations, consulter les détails des vols, et recevoir des notifications.
    - **Agent de Voyage** : Un utilisateur qui peut gérer les réservations et les utilisateurs.
3. **Ajouter les cas d'utilisation suivants :**
    - **Rechercher des vols** : Le client peut rechercher des vols disponibles.
    - **Réserver des billets** : Le client peut réserver des billets de vol.
    - **Annuler des réservations** : Le client peut annuler ses réservations.

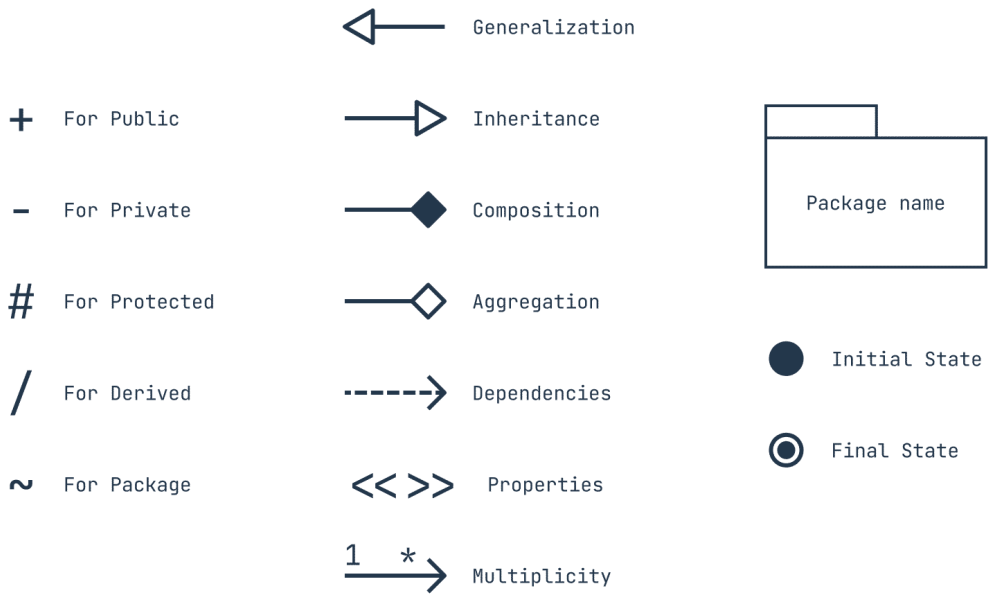
- **Consulter les détails des vols** : Le client peut consulter les détails des vols réservés.
- **Recevoir des notifications** : Le client peut recevoir des notifications sur ses vols.
- **Gérer les réservations** : L'agent de voyage peut gérer les réservations des clients.
- **Gérer les utilisateurs** : L'agent de voyage peut gérer les comptes des utilisateurs.

4. **Relier les acteurs aux cas d'utilisation appropriés :**
   - Le client doit être lié à tous les cas d'utilisation sauf ceux de l'agent de voyage.
   - L'agent de voyage doit être lié aux cas d'utilisation "Gérer les réservations" et "Gérer les utilisateurs".

LE DIAGRAMME DE CLASSE (CLASS DIAGRAM)



Class Diagram for a Banking System


Generalization
+ For Public
Inheritance
Package name
- For Private
Composition
For Protected
Aggregation
Initial State
/ For Derived
Dependencies
~ For Package
<< >> Properties
Final State
1 *
Multiplicity

1. Système de Bibliothèque

Contexte :

Vous travaillez pour **LibraryTech**, une entreprise spécialisée dans les systèmes de gestion de bibliothèques. Votre mission est de modéliser un système de gestion de bibliothèque. Ce système doit permettre de gérer les livres, les membres et les prêts de livres.

Tâches à réaliser :

1. **Créer un diagramme de classes pour le système de gestion de bibliothèque en utilisant UMLet.**

Étapes à suivre :

1. **Définir les classes principales :**
    - **Book** : Représente un livre dans la bibliothèque.
    - **Member** : Représente un membre de la bibliothèque.
    - **Loan** : Représente un prêt de livre à un membre.
2. **Ajouter les Attributes et Methods pour chaque classe :**
    - **Book** :
        - Attributes : title, author, ISBN, yearOfPublication, condition.
        - Methods : borrow(), return().
    - **Member** :
        - Attributes : name, membershipNumber, address, telephone.
        - Methods : registerMember(), borrowBook(), returnBook().
    - **Loan** :
        - Attributes : dateBorrowed, dateReturned.
        - Methods : recordBorrow(), recordReturn().
3. **Définir les relations entre les classes :**
    - Un **Book** peut être associé à plusieurs **Loan**.
    - Un **Member** peut être associé à plusieurs **Loan**.

- Un **Loan** est associé à un seul **Book** et à un seul **Member**.

Conseils pour la réalisation :

Lorsque vous créez votre diagramme de classes, assurez-vous de bien définir les Attributes et les Methods de chaque classe. Utilisez des noms clairs et descriptifs pour les classes, les Attributes et les Methods afin de rendre le diagramme facile à comprendre. Pensez à la manière dont chaque classe interagit avec les autres et représentez ces relations avec précision. Prenez le temps de bien organiser votre diagramme pour qu'il soit propre et lisible.

Bonne chance !

2. Système de Gestion de Restaurant

Contexte :

Vous travaillez pour **RestaurantPro**, une entreprise spécialisée dans les systèmes de gestion de restaurants. Votre mission est de modéliser un système de gestion pour un restaurant. Ce système doit permettre de gérer les tables, les commandes, les plats et les serveurs.

Tâches à réaliser :

1. **Créer un diagramme de classes pour le système de gestion de restaurant en utilisant UMLet.**

Étapes à suivre :

1. **Définir les classes principales :**
    - **Table** : Représente une table dans le restaurant.
    - **Order** : Représente une commande passée par un client.
    - **Dish** : Représente un plat du menu.
    - **Waiter** : Représente un serveur du restaurant.
2. **Ajouter les Attributes et Methods pour chaque classe :**
    - **Table** :
        - Attributes : numberTable, numberPlaces, isOccupied.
        - Methods : occupyTable(), freeTable().
    - **Order** :
        - Attributes : orderNumber, orderDate, status.
        - Methods : addDish(), deleteDish(), validateOrder().
    - **Dish** :
        - Attributes : name, description, price.
        - Methods : modifyDish(), deleteDish().
    - **Waiter** :
        - Attributes : name, identifier.
        - Methods : takeOrder(), serveOrder().

3. **Définir les relations entre les classes :**
    - Une **Table** peut être associée à plusieurs **Order**.
    - Une **Order** peut inclure plusieurs **Dish**.
    - Un **Waiter** peut gérer plusieurs **Order** .
    - Une **Order** est associée à une seule **Table** et à un seul **Waiter**.

3.Système de Gestion d'Hôpital

Contexte :

Vous travaillez pour **HealthCare IT Solutions**, une entreprise spécialisée dans les systèmes de gestion pour les hôpitaux. Votre mission est de modéliser un système de gestion d'hôpital. Ce système doit permettre de gérer les patients, les médecins, les rendez-vous et les traitements.

Tâches à réaliser :

1. **Créer un diagramme de classes pour le système de gestion d'hôpital en utilisant UMLet.**

Étapes à suivre :

1. **Définir les classes principales :**
   - **Patient** : Représente un patient de l'hôpital.
   - **Doctor** : Représente un médecin de l'hôpital.
   - **Appointment** : Représente un rendez-vous médical.
   - **Treatment** : Représente un traitement médical prescrit à un patient.
2. **Ajouter les Attributes et Methods pour chaque classe :**
   - **Patient** :
     - Attributes : name, patientNumber, address, telephone, dateOfBirth.
     - Methods : takeAnAppointment(), cancelAnAppointment().
   - **Doctor** :
     - Attributes : name, specialty, identifier.
     - Methods : consultPatient(), prescribeTreatment().
   - **Appointment** :
     - Attributes : appointmentDate, appointmentTime, status.

        - Methods : scheduleAnAppointment(), cancelAnAppointment().
    - **Treatment** :
        - Attributes : description, startDate, endDate.
        - Methods : startTreatment(), endTreatment().
3. **Définir les relations entre les classes :**
    - Un **Patient** peut avoir plusieurs **Appointment** .
    - Un **Doctor** peut avoir plusieurs **Appointment** .
    - Un **Appointment** est associé à un seul **Patient** et à un seul **Doctor** .
    - Un **Patient** peut avoir plusieurs **Treatment**.
    - Un **Treatment** est associé à un seul **Patient**.

4. Système de Gestion de l'École

Contexte :

Vous travaillez pour **EduManage**, une entreprise spécialisée dans les systèmes de gestion pour les institutions éducatives. Votre mission est de modéliser un système de gestion scolaire qui permet de gérer les étudiants, les enseignants, les cours et les inscriptions.

Tâches à réaliser :

1. **Créer un diagramme de classes pour le système de gestion scolaire en utilisant UMLet.**

Étapes à suivre :

1. **Définir les packages principaux :**
   - `School`
   - `Persons`
   - `Courses`
   - `Registration`
2. **Définir les classes principales et interfaces :**
   - `Persons` :
     - **Person** (Classe abstraite)
     - **Student** (Hérite de `Person`)
     - **Teacher** (Hérite de `Person`)
   - `Courses` :
     - **Course**
     - **Subject** (Interface)
   - `Registration` :
     - **Registration**
3. **Ajouter les Attributes et Methods pour chaque classe avec différentes visibilités et concepts avancés :**
   - **Person** (Classe abstraite) :

        - Attributes :
            - #name : String
            - #address : String
            - #telephone : String
        - Methods :
            - +getName() : String
            - +setName(name : String) : void
    - **Student** :
        - Attributes :
            - -studentNumber : String
            - -dateBirth : Date
        - Methods :
            - +registerCourse() : void
            - +unregisterCourse() : void
    - **Teacher** :
        - Attributes :
            - -speciality: String
            - -identifier : String
        - Methods :
            - +teachCourse() : void
            - +assessStudent() : void
    - **Course** :
        - Attributes :
            - +courseName : String
            - +courseCode : String
            - +description : String
        - Methods :
            - +addCourse() : void
            - +removeCourse() : void
    - **Registration** :
        - Attributes :
            - +registrationDate : Date
            - #status : String

        - Methods :
            - +validateRegistration() : void
            - +cancelRegistration() : void
    - **Subject** (Interface) :
        - Methods :
            - +getDescription() : String
4. **Définir les relations entre les classes :**
    - Un **Student** peut être inscrit à plusieurs **Course** (Association).
    - Un **Teacher** peut enseigner plusieurs **Course** (Association).
    - Un **Course** peut avoir plusieurs **Student** inscrits (Aggregation) et plusieurs **Teacher** (Aggregation).
    - Une **Registration** est associée à un seul **Student** et un seul **Course** (Composition).

5. Système de Gestion de Projet

Contexte :

Vous travaillez pour **ProjectHub**, une entreprise spécialisée dans les systèmes de gestion de projets. Votre mission est de modéliser un système de gestion de projet qui permet de gérer les projets, les tâches, les employés et les rapports de progression.

Tâches à réaliser :

1. **Créer un diagramme de classes pour le système de gestion de projet en utilisant UMLet.**

Étapes à suivre :

1. **Définir les packages principaux :**
    - `Project`
    - `Tasks`
    - `Employees`
    - `Reports`
2. **Définir les classes principales et interfaces :**
    - `Project`:
        - **Project**
    - `Tasks`:
        - **Task**
        - **ComplexTask** (Hérite de `Task`)
        - **SubTask** (Hérite de `Task`)
        - **ITask** (Interface)
    - `Employees`:
        - **Employee**

        - **Manager** (Hérite de `Employee`)
    - `Reports` :
        - **Report**
3. **Ajouter les Attributes et Methods pour chaque classe avec différentes visibilités et concepts avancés :**
    - **Project**:
        - Attributes :
            - +name : String
            - #budget : double
            - -projectCode : String
            - -startDate : Date
            - -EndDate : Date
        - Methods :
            - +addTask() : void
            - +removeTask() : void
    - **Task**(Implémente `ITask`) :
        - Attributes :
            - +TaskName : String
            - #description : String
            - -estimatedDuration : int
            - -status : String
        - Methods :
            - +assignEmployee() : void
            - #markComplete() : void
    - **ComplexTask**(Hérite de `Task`) :
        - Attributes :
            - +subtasks : List<Subtasks>
        - Methods :
            - +addSubTask() : void
            - +removeSubTask() : void
    - **Subtask** (Hérite de `Task`)
        - Methods :
            - +completedTask() : void

- **ITask** (Interface) :
  - Methods :
    - +getStatus() : String
- **Employee** :
  - Attributes :
    - +name: String
    - -identifier : String
    - #post : String
  - Methods :
    - +takeTask() : void
    - #submitReport() : void
- **Manager** (Hérite de `Employee`)
  - Methods :
    - +superviseProject() : void
    - +approveTask() : void
- **Report** :
  - Attributes :
    - +reportDate : Date
    - #content : String
    - -note : String
  - Methods :
    - +createReport() : void
    - #approveReport() : void

4. **Définir les relations entre les classes :**
   - Un **Project** peut contenir plusieurs **Tasks** (Aggregation).
   - Une **ComplexTask** peut contenir plusieurs **Subtask** (Composition).
   - Une **Task** peut être assignée à plusieurs **Employee** (Association).
   - Un **Employee** peut rédiger plusieurs **Report** (Aggregation).
   - Un **Report** est associé à une seule **Task** et un seul **Employee** (Composition).
   - Un **Manager** supervise plusieurs **Project** (Association).

6. Système de Gestion de la Vente en Ligne

Contexte :

Vous travaillez pour **E-Commerce Solutions**, une entreprise spécialisée dans les systèmes de gestion de la vente en ligne. Votre mission est de modéliser un système de gestion de la vente en ligne qui permet de gérer les clients, les produits, les commandes et les paiements.

Tâches à réaliser :

1. **Créer un diagramme de classes pour le système de gestion de la vente en ligne en utilisant UMLet.**

Étapes à suivre :

1. **Définir les packages principaux :**
   - `Customers`
   - `Products`
   - `Orders`
   - `Payments`
2. **Définir les classes principales et interfaces :**
   - `Customers` :
     - **Customer**
     - **Address** (Classe)
     - **Contact** (Interface)
   - `Products` :
     - **Product**
     - **Stock**
   - `Orders` :
     - **Order**
     - **OnlineOrdering** (Classe)

- `Payments` :
  - **Payment**
  - **PaymentMode** (Interface)

3. **Ajouter les Attributes et Methods pour chaque classe avec différentes visibilités et concepts avancés :**
   - **Customer** :
     - Attributes :
       - +name : String
       - #address : Address
       - -email : String
       - -phone : String
     - Methods :
       - +makeOrder() : void
       - +cancelOrder() : void
   - **Address** :
     - Attributes :
       - +street : String
       - +city : String
       - +postCode : String
     - Methods :
       - +getCompleteAddress() : String
   - **Product** :
     - Attributes :
       - +productName : String
       - #price : double
       - -stock : Stock
     - Methods :
       - +addProduct() : void
       - #updateProduct() :void

- **Stock** :
  - Attributes :
    - +quantityAvailable: int
    - +quantityReserved : int

    - Methods :
        - +addStock() : void
        - +removeStock() : void
- **Order** :
    - Attributes :
        - +ordernumber : String
        - #orderdate : Date
        - -status : String
        - -orderline : List<OnlineOrdering>
    - Methods :
        - +validateOrder() : void
        - +cancelOrder() : void
- **OnlineOrdering** :
    - Attributes :
        - +quantity : int
        - #unitPrice : double
    - Methods :
        - +calculateAmount() : double
- **Payment**:
    - Attributes :
        - +amount: double
        - #paymentDate : Date
        - -paymentMode : PaymentMode
    - Methods :
        - +makePayment() : void
        - +cancelPayment() : void
- **PaymentMode**(Interface) :
    - Methods :
        - +makeTransaction() : void
- **BankCard**(Implémente `PaymentMode`) :
    - Methods :
        - +makeTransaction() : void
- **PayPal** (Implémente `PaymentMode`) :

    - Methods :
        - +makeTransaction() : void

Relations :

- Un **Customer** peut passer plusieurs **Order** (Association).
- Une **Order** peut contenir plusieurs **OnlineOrdering** (Composition).
- Un **Product** est associé à un seul **Stock** (Composition).
- Un **Stock** est associé à un seul **Product** (Composition).
- Une **Order** est associée à un seul **Payment** (Composition).
- Un **Payment** utilise un seul **PaymentMode** (Association).