RAPORT TP UML

Unified Modeling Language

Réalisé Par :

LACHHAB FATIMA EZZAHRAE

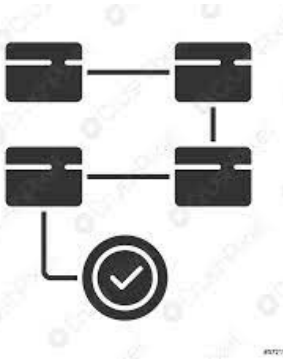
KIRANE OUMAIA

Sommaire:

Introduction générale :

L'Unified Modeling Language (UML) est un langage de modélisation graphique utilisé dans le domaine du génie logiciel pour représenter visuellement un système logiciel. Créé pour unifier les différentes méthodes de modélisation, UML offre un ensemble de notations standardisées permettant aux développeurs, aux architectes logiciels et aux parties prenantes de communiquer efficacement et de comprendre la structure, le comportement, et les interactions d'un système logiciel. Les diagrammes UML, tels que les diagrammes de classes, de séquence, d'activités et de cas d'utilisation, offrent une vue complète du système, facilitant ainsi la conception, la mise en œuvre et la maintenance des logiciels. En intégrant des concepts visuels intuitifs, UML favorise une communication claire et précise tout au long du cycle de vie du développement logiciel, améliorant ainsi la collaboration entre les membres de l'équipe et la compréhension des spécifications du système.

TP NO 1 : UML



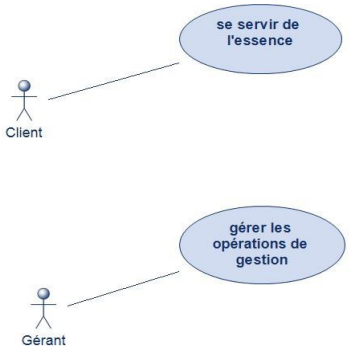
Exercice 1 : Recensement acteurs et cas d'utilisation simples, Relations entre acteurs.

Un diagramme de cas d'utilisation aide à visualiser les interactions entre les utilisateurs et le système, fournissant une base solide pour comprendre les exigences fonctionnelles d'un projet.

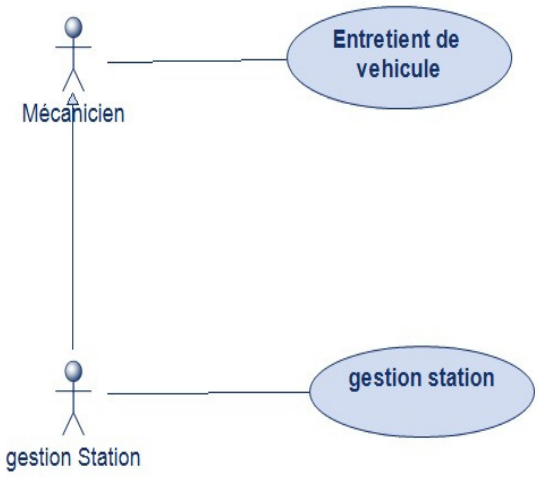
1- L'acteur du système est le client car pistolet et la gâchette sont seulement des outils pour prendre de l'essence.

2- Le gérant est un acteur du système.

La modélisation :

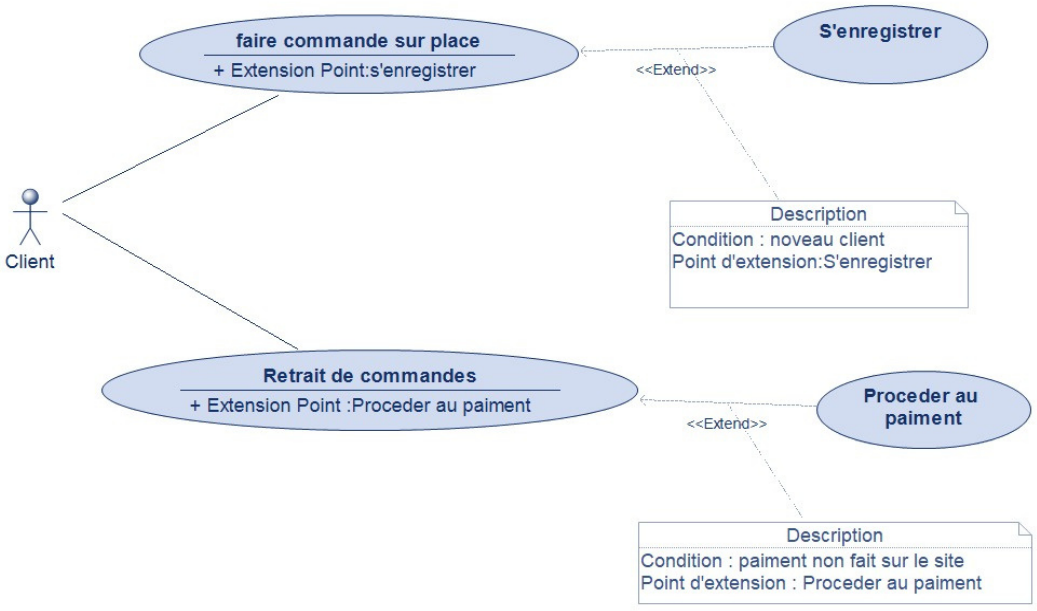


3- Puisque le chef d'atelier est aussi un mécanicien alors le gérant hérite de mécanicien



Exercice 2 : Relations entre cas d'utilisation

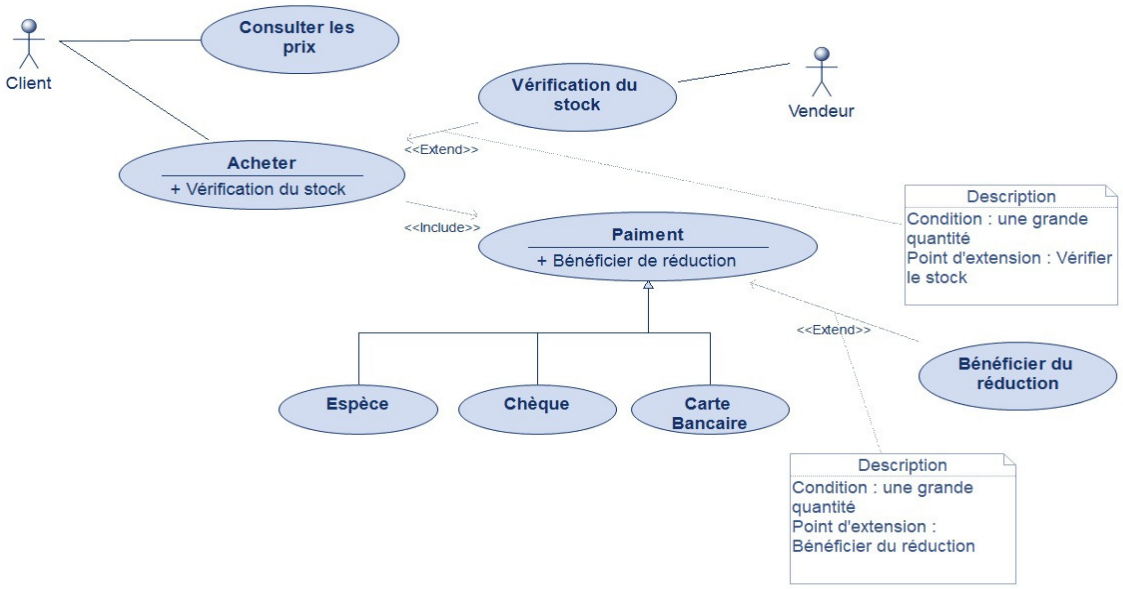
Modélisation des fonctionnalités proposées par la borne automatique.



Exercice 3 : Relations entre cas d'utilisation

Diagramme de Cas d'Utilisation pour le Processus de Vente dans un Magasin :

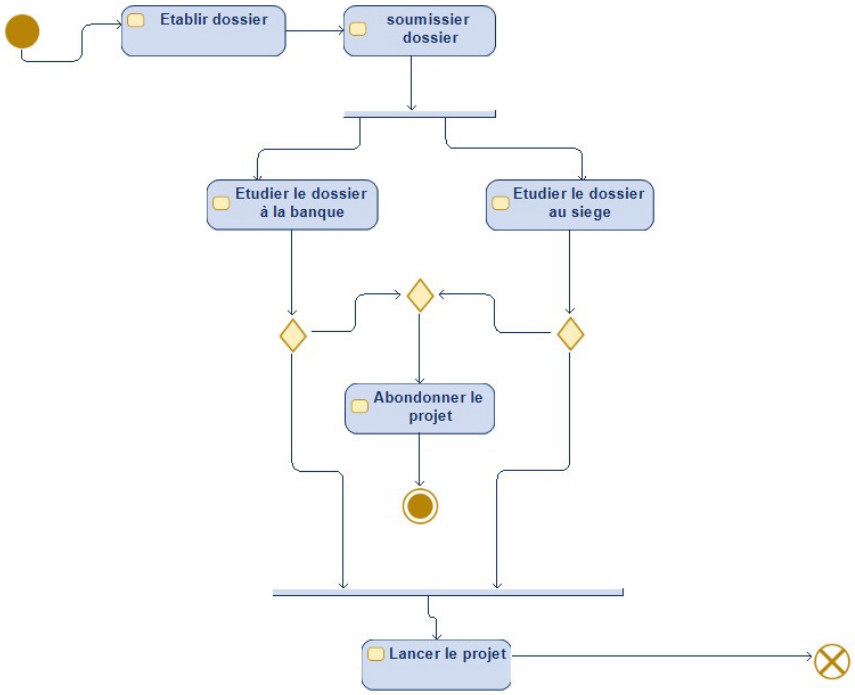
1. **Acteurs :**
   -Client
   -Vendeur
   - Caissier



- Ce diagramme de cas d'utilisation représente les différentes interactions possibles entre les acteurs (Client, Vendeur, Caissier) et le système de vente du magasin.
- Il met en évidence les principales étapes du processus de vente, y compris la consultation des prix, l'achat de quantités importantes, le paiement, et l'application de réductions.

EXERCICE 4 : DIAGRAMME D'ACTIVITÉS

LE DIAGRAMME D'ACTIVITÉ DES CAS D'UTILISATION PERMET DE MODÉLISER LE FLUX D'ACTIVITÉS ASSOCIÉ AU PROCESSUS DE LANCEMENT D'UN PROJET D'AMÉNAGEMENT.



TP2 : UML

Diagramme de classe :

Objectif : Le diagramme de classe montre la structure statique d'un système en identifiant les classes du système, leurs attributs, méthodes et les relations entre ces classes.

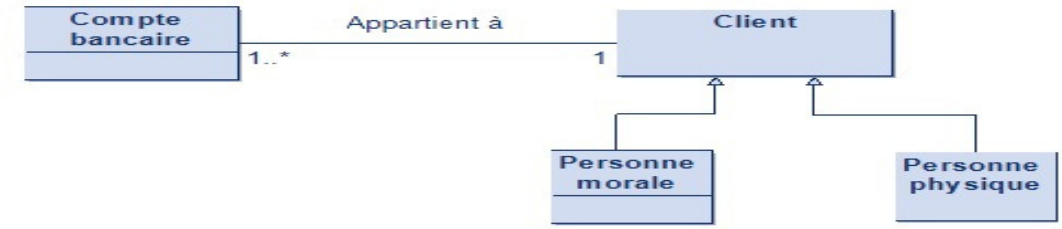
Diagramme d'Objet :

Objectif : Le diagramme d'objet représente des instances spécifiques de classes et montre comment ces objets interagissent à un moment donné.

Exercice 1 : Relation entre classes

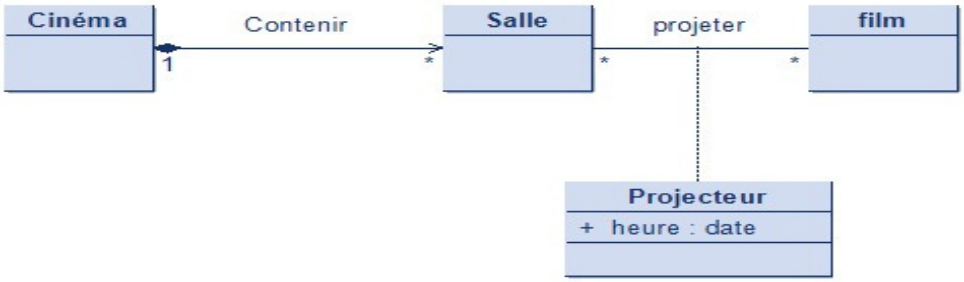
simples, Relations entre acteurs.

1- Un compte bancaire peut appartenir à un seul client qui peut être une personne physique ou morale.

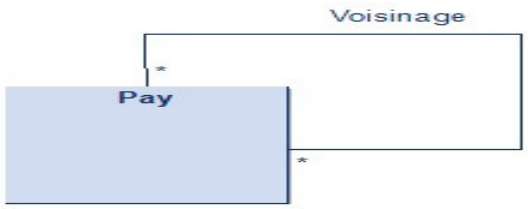


2- Un répertoire contient des fichiers.

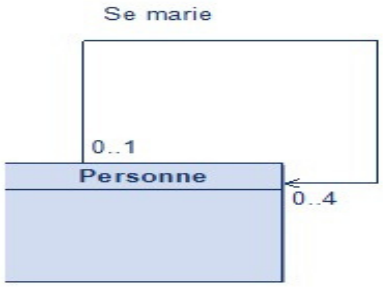
3- Un cinéma est composé de plusieurs salles où on projette des films à une heure déterminée.



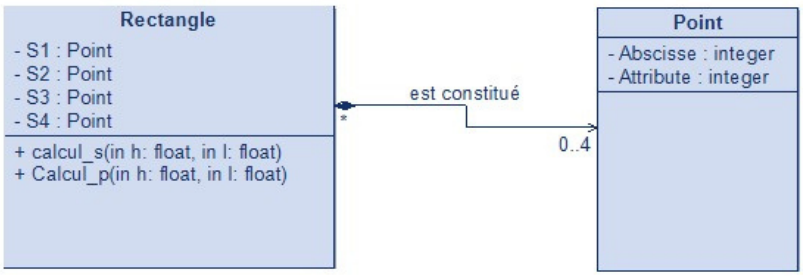
4- Un pays est voisin de plusieurs pays.



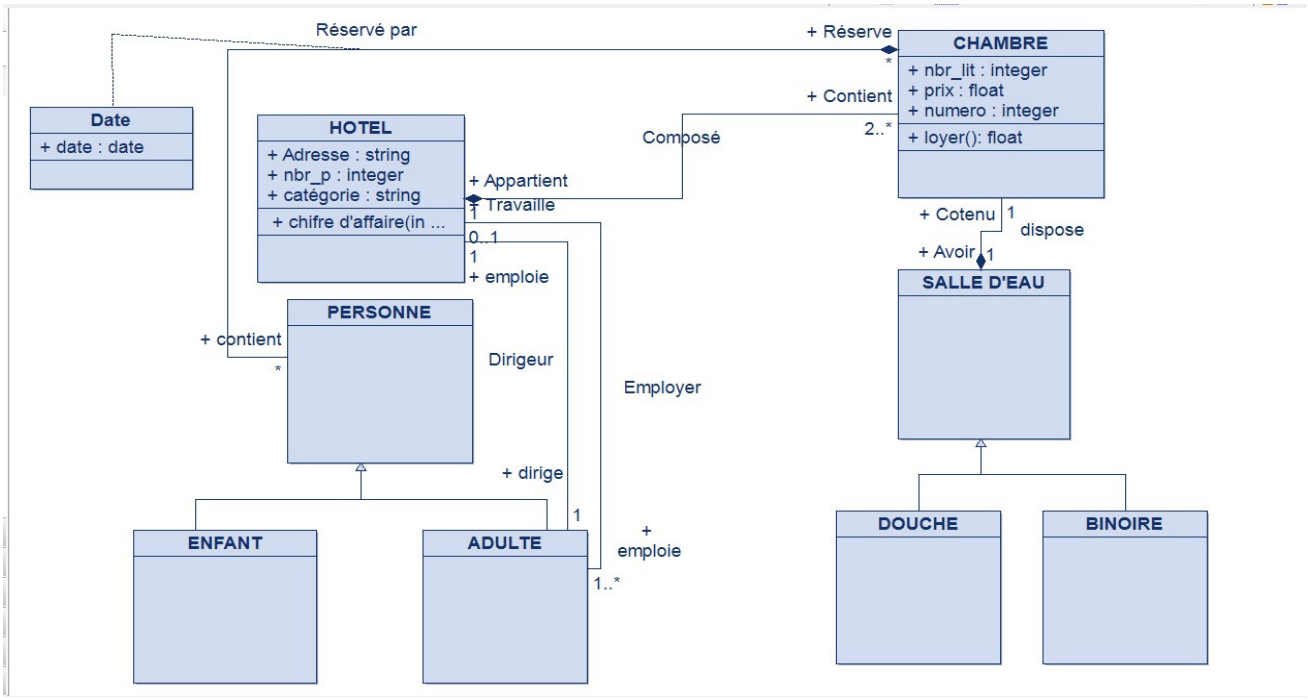
5- Un homme peut se marier à quatre femmes et une femme peut se marier à un seul homme. Un homme ou une femme sont des personnes



6- Un rectangle est constitué de quatre sommets qui sont des points, et il est possible de calculer sa surface et son périmètre. Un point est caractérisé par les coordonnées (abscisse et ordonnée qui sont des entiers). Les attributs sont privés et les méthodes sont publiques. La visibilité de l'association est privée.



Exercice 2 : Diagramme de classes complet.



1. **Classes principales :**

- **Hôtel :** Représente l'hôtel avec des attributs tels que "adresse", "nombre de pièces" et "catégorie".

-**Chambre :** Caractérise chaque chambre avec des attributs comme "nombre de lits", "prix", et "numéro". La chambre est associée à une "Salle d'eau" (sous-classe de Chambre) qui peut être soit "Douche" soit "Baignoire".

- **Personne :** Classe générique pour représenter les personnes, avec des sous-classes "Adulte" et "Enfant". Contient les attributs communs "nom" et "prénom".

2. **Relations :**

- **Occupation :** Associe une personne à une chambre à une date donnée.

-**Emploi :** Lie un employé à l'hôtel en tant que "Directeur" ou autre type d'employé.

3. **Règles spécifiques :**

-**Restrictions d'âge :** La classe "Personne" est subdivisée en "Adulte" et "Enfant" pour respecter les règles de non-emploi des enfants.

- **Calcul du loyer :** La classe "Chambre" permet de calculer le loyer quotidien en fonction du prix de la chambre.

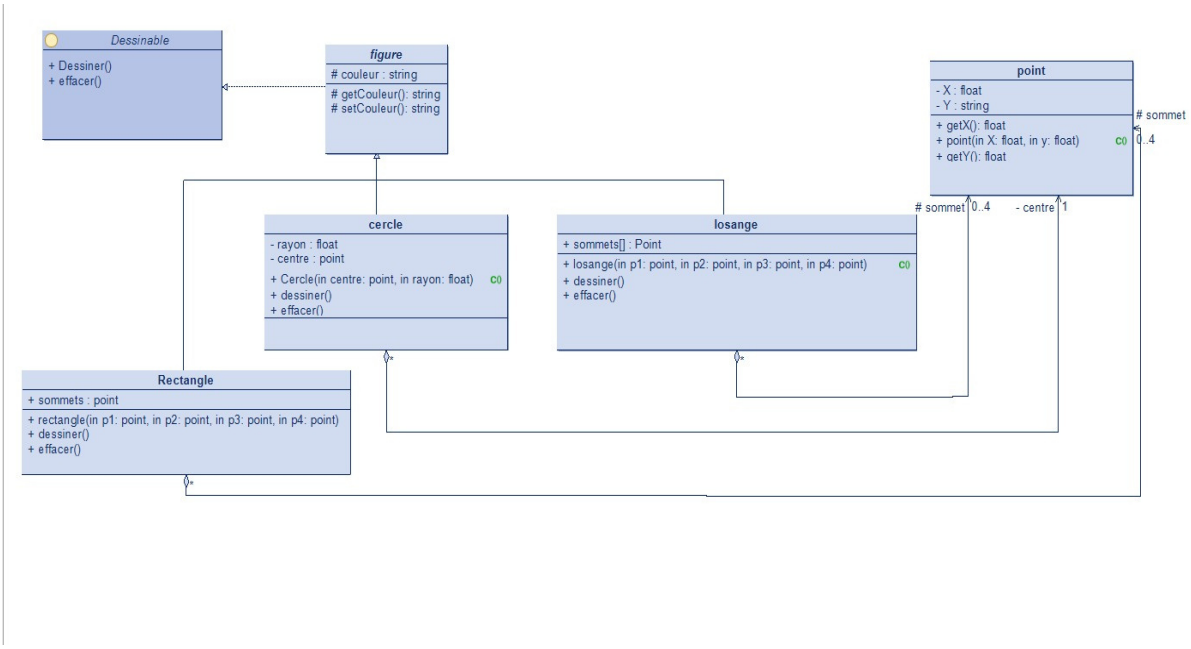
4. **Chiffre d'affaires :**

- La possibilité de calculer le chiffre d'affaires de l'hôtel entre deux dates est implicite dans les relations entre les classes "Hôtel", "Chambre", et "Occupation".

La modélisation est donc détaillée et tient compte des différents aspects du système hôtelier, facilitant la compréhension et la communication des concepts clés.

Exercice 3 : Correspondance entre code Java et diagramme de classes.

- La correspondance entre le code Java et le diagramme de classes est fluide et intuitive. Chaque classe dans le diagramme de classes a été représentée de manière cohérente en Java. Les attributs et les relations entre les classes ont été traduits de manière claire et directe. Les classes Java reflètent fidèlement la structure du diagramme, ce qui facilite la compréhension et la mise en œuvre du modèle. La hiérarchie des classes, les associations, et les concepts de l'orienté objet sont bien préservés dans le code Java, rendant la transition du modèle conceptuel vers l'implémentation pratique et sans ambigüité.



Exercice 4 : Génération du code Java à partir du diagramme de classes.

La génération du code Java à partir du diagramme de classes a été réalisée de manière automatique et efficace. Chaque classe, attribut, méthode et relation dans le diagramme a été converti en code Java correspondant. L'outil de génération de code a su interpréter les concepts du modèle de manière précise, produisant un code Java propre et conforme aux normes. Cette approche automatisée facilite la cohérence entre le modèle conceptuel et l'implémentation, réduisant ainsi les risques d'erreurs humaines. En résultat, le processus de développement est accéléré, offrant une transition fluide de la conception à l'application concrète.

Pour la classe Rectangle :

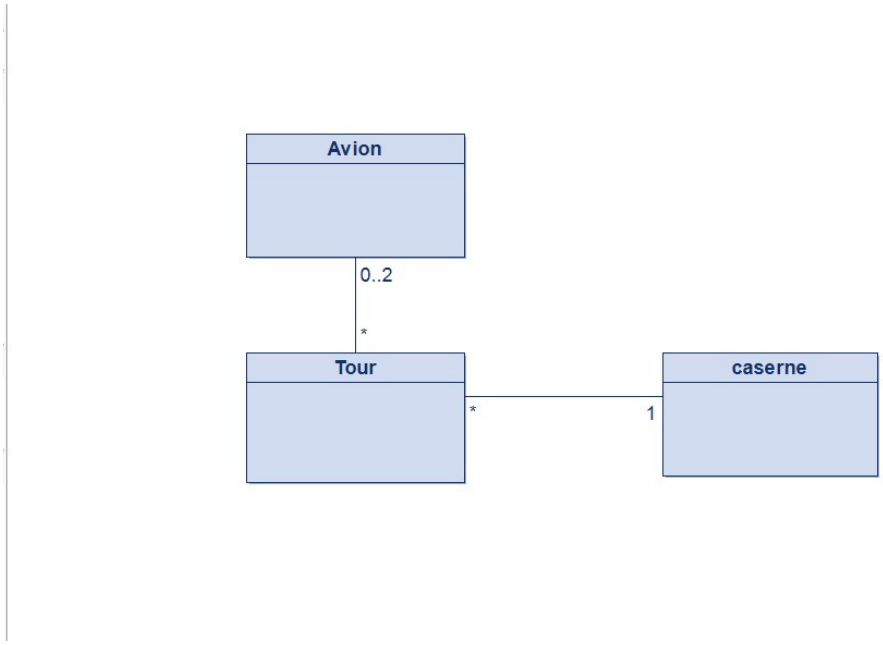
```java
import java.util.ArrayList;
import java.util.List;
import com.modeliosoft.modelio.javadesigner.annotations.objid;

@objid ("8098f25b-13d8-4358-8da0-5f1bfc076a35")
public class Rectangle {
   @objid ("91306408-dcd5-4bbd-906d-2f47a6b0147c")
   private Point S1;

   @objid ("19fbbfdb-26e0-4cc6-a509-fc47e004da30")
   private Point S2;

   @objid ("88254b8c-95e0-4ebe-9d07-ece41663b362")
   private Point S3;

   @objid ("f447e448-21dd-4d6e-83a1-f5a99696df7d")
   private Point S4;

   @objid ("0ff6c5ba-11e9-4c19-b841-05ee9b02a07f")
   public List<Point>  = new ArrayList<Point> ();

   @objid ("ccd46d3c-4dbd-4569-abaf-716ba3d6bf9b")
   public void calcul_s(final float h, final float l) {
   }

   @objid ("876921f4-c25e-4baf-901d-10239f020665")
   public void Calcul_p(final float h, final float l) {
   }
```

Pour la classe point :

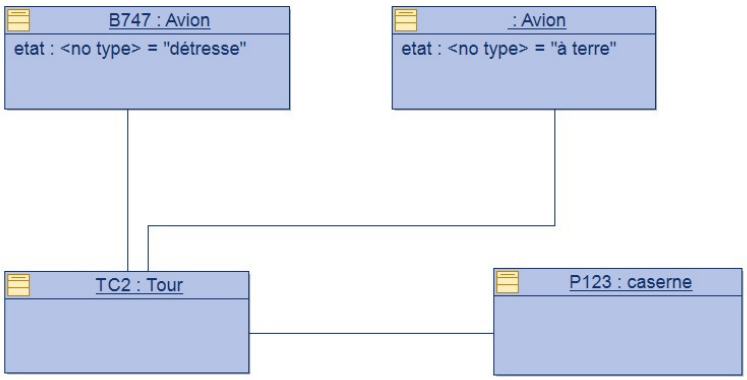
```
import com.modeliosoft.modelio.javadesigner.annotations.objid;

@objid ("513d11d6-a3e2-489f-acda-87912f0c3126")
public class Point {
    @objid ("579b9f7a-68db-4496-b388-c23d3b4014db")
    private int Abscisse;

    @objid ("ab1f66f2-6efd-489c-9696-c56ba5a338fd")
    private int Attribute;

}
```

Exercice 5 : Transformation de diagramme de classes en diagramme d'objets.

1- le diagramme de classes.



2- Le diagramme d'objet.


B747 : Avion
etat : <no type> = "détresse"
: Avion
etat : <no type> = "à terre"
TC2 : Tour
P123 : caserne

TP3 : UML

Diagramme de Séquence :

Objectif : Le diagramme de séquence modélise la séquence d'interactions entre différents objets ou composants au fil du temps. Il se concentre sur la manière dont les messages sont échangés entre ces objets pour accomplir une fonction spécifique.

Utilisation : Il est utilisé pour visualiser et comprendre les interactions entre les objets d'un système, montrant l'ordre chronologique des messages échangés.

Diagramme de Communication (ou de Collaboration) :

Objectif : Le diagramme de communication met l'accent sur la manière dont les objets coopèrent pour réaliser une tâche ou un scénario donné. Il met en évidence les connexions entre les objets et comment ils interagissent.

Utilisation : Il est utilisé pour montrer la structure statique et les connexions dynamiques entre les objets, mettant en lumière les collaborations nécessaires pour atteindre un objectif particulier.

~~Diagramme d'État de Transition :~~

Objectif : Le diagramme d'état de transition modélise les différents états d'un objet ou d'un système et les transitions
entre ces états en réponse à des événements spécifiques.

Utilisation : Il est utilisé pour représenter le comportement d'un objet ou d'un système dans le temps, en mettant en évidence les changements d'état et les événements déclencheurs.

Chacun de ces diagrammes offre une perspective unique sur le système et

est utilisé dans des contextes spécifiques du processus de modélisation

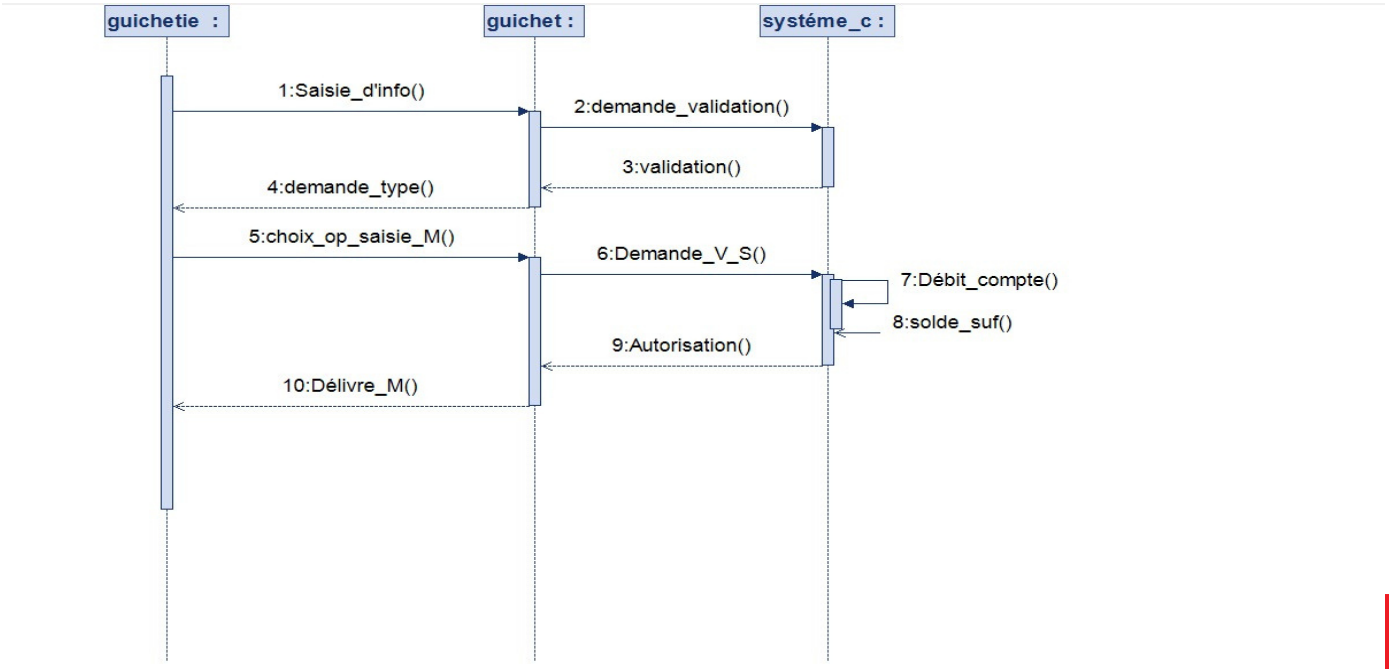
pour capturer différentes dimensions de la dynamique et du comportement

d'un système logiciel.

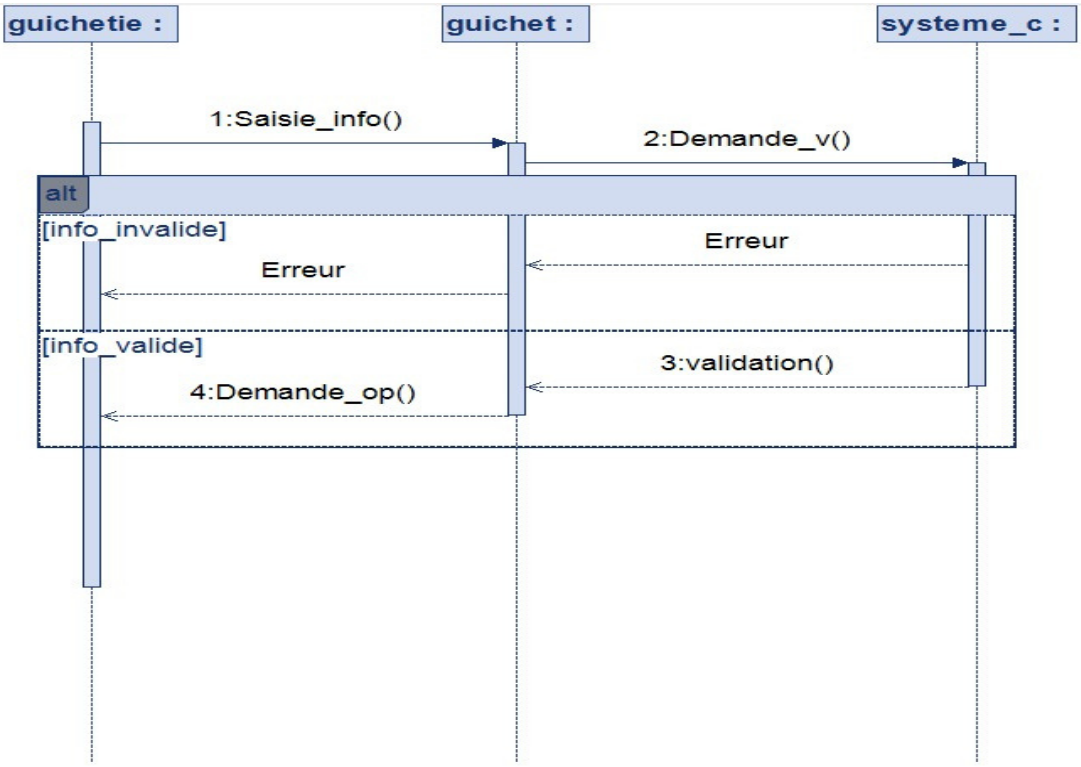
Exercice 1 :

Diagramme de séquence.

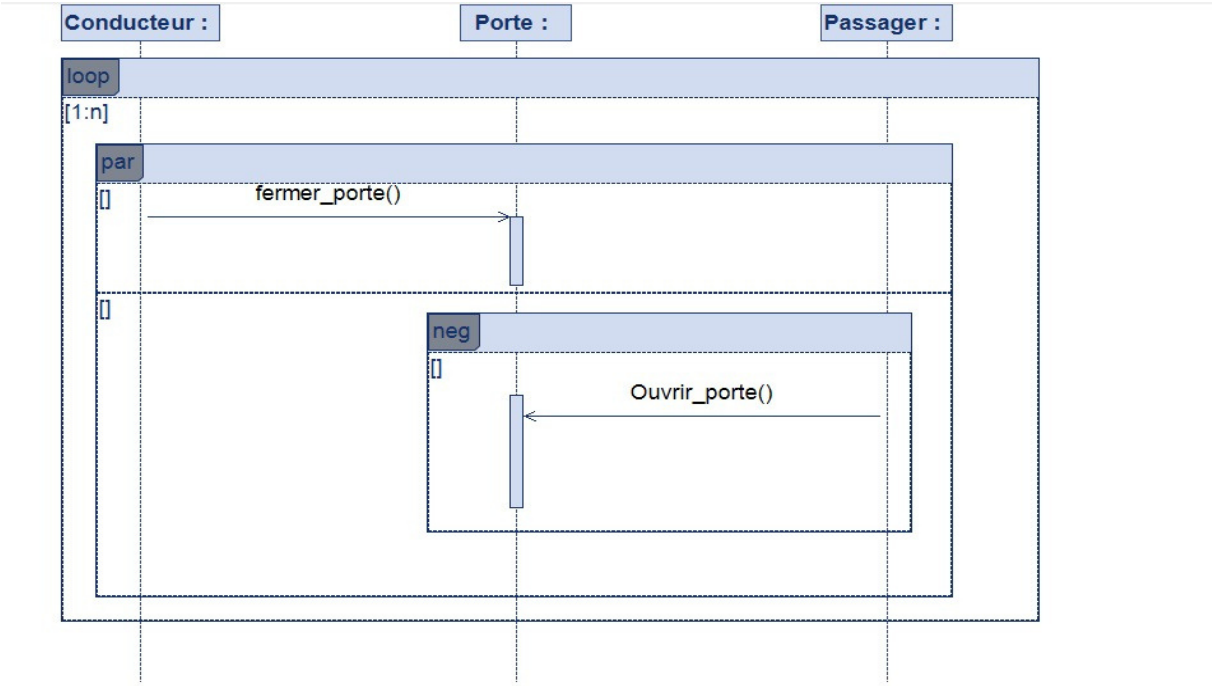
1-Le diagramme de séquence dans le cas normale :



2- Le diagramme dans le cas où les informations d’authentification sont erronées.



Exercice 2 : Diagramme de séquence.



Fragment "loop" pour la fermeture continue des portes :

 Après que le conducteur a initié l'action de fermer les portes, on a utilisé un fragment "loop" pour indiquer que cette action se répète jusqu'à ce que certaines conditions soient remplies (par exemple, jusqu'à ce que toutes les portes soient fermées avec succès).

Fragment "par" pour modéliser le parallélisme :

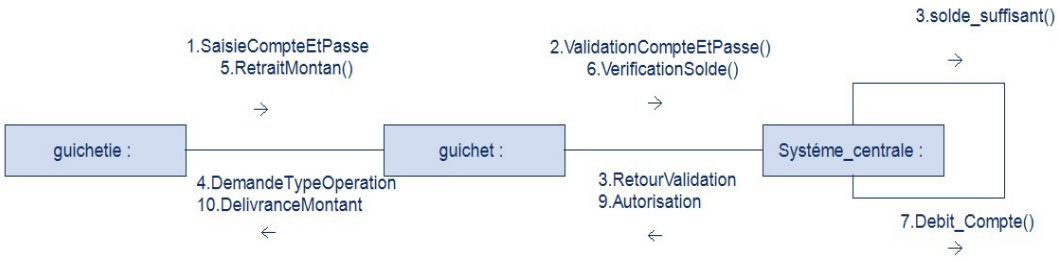
 Si des passagers tentent d'ouvrir différentes portes en même temps, on a utilisé un fragment "par" pour montrer ce parallélisme.

Fragment "neg" pour représenter l'interdiction d'ouverture des portes par les passagers :

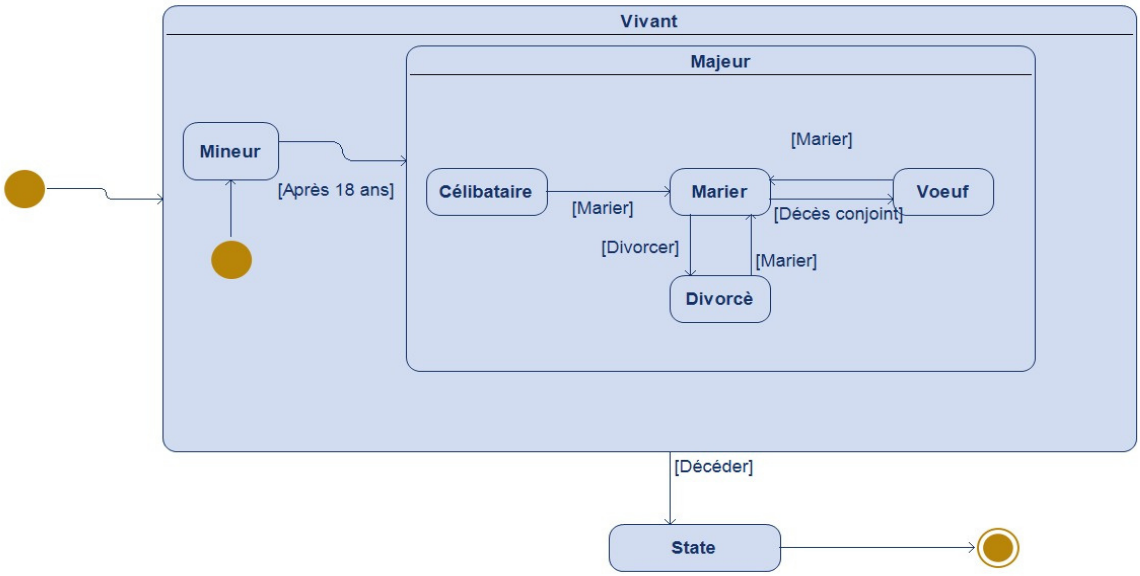
À l'intérieur du fragment "loop", on a utilisé un fragment "neg" pour indiquer que les passagers ne peuvent pas ouvrir les portes pendant que le conducteur essaie de les fermer.

Diagramme de communication.

Exercice 3 :



Exercice 4 : Diagramme d'états-tansitions.



en conclusion de ce travail pratique regroupant une diversité d'exercices sur différents diagrammes uml, nous pouvons constater l'importance et la puissance d'unified modeling language (uml) dans le domaine du génie logiciel. les diagrammes uml, tels que les diagrammes de classes, de séquence, d'activités et de cas d'utilisation, offrent des outils visuels essentiels pour la modélisation, la conception et la communication des systèmes logiciels.

À travers ces exercices, nous avons exploré comment représenter la structure statique et dynamique d'un système, les interactions entre ses composants, ainsi que les aspects fonctionnels et comportementaux. Chaque type de diagramme UML a ses propres avantages et applications spécifiques, permettant aux développeurs et aux architectes de choisir les représentations les plus adaptées à leurs besoins.

En adoptant UML, les équipes de développement peuvent améliorer la compréhension collective des exigences, favoriser la collaboration entre les membres de l'équipe, et faciliter la gestion et l'évolution des systèmes logiciels.

Ainsi, ce travail pratique a contribué à renforcer nos compétences dans l'utilisation de ces diagrammes, nous préparant ainsi à aborder de manière plus approfondie les défis du développement logiciel dans un environnement professionnel.