R3.04. Qualité de développement
TD3
Modélisation par machines à états UML

**Exercice 1 :**

**Scénario : Système de Feux de Circulation**

Vous devez concevoir un **diagramme de machine à états** pour un **système de feux de circulation** à un carrefour fréquenté. Le feu de circulation possède trois états principaux : **Rouge**, **Vert**, et **Orange**, et effectue des transitions entre ces états en fonction d'un minuteur et de certaines conditions.

**Description du problème :**

Le système de feux fonctionne comme suit :

1. Le système commence dans l'état **Rouge**.
2. Après **60 secondes** en état **Rouge**, le feu passe à **Vert**.
3. L'état **Vert** dure **45 secondes**, après quoi le système passe à **Orange**.
4. L'état **Orange** dure **5 secondes**, puis le système revient à **Rouge**.
5. En cas d'urgence :
    - Si le mode urgence est activé (ex. : pour une ambulance), le système passe immédiatement à **Clignotant Orange**.
    - En état **Clignotant Orange**, le système reste dans cet état jusqu'à la désactivation du mode urgence.
6. Une fois le mode urgence désactivé, le système retourne à l'état **Rouge** et reprend le cycle normal.

**Tâches :**

1. **Identifier les États** :
    - Listez tous les états possibles du système de feux de circulation.
2. **Identifier les Transitions** :
    - Définissez les événements ou conditions déclenchant les transitions entre les états (ex. : minuteur, activation/désactivation du mode urgence).
3. **Créer le Diagramme UML** :
    - Réalisez un **diagramme de machine à états UML** qui inclut :
        - Les **états** (**Rouge**, **Vert**, **Orange**, **Clignotant Orange**).
        - Les **transitions** avec des étiquettes appropriées (ex. : `timer == 60s`, `emergencyActivated`).
        - L'**état initial** et tout état final (si applicable).
4. **Considérer les Cas Limites** :
    - Que se passe-t-il si le minuteur tombe en panne ? Comment le système peut-il gérer des transitions imprévues ?

**Extensions :**

- **Ajout des piétons** : Intégrez les états des feux pour piétons (**Marcher**, **Ne pas marcher**) synchronisés avec les feux de circulation.
- **Gestion des Pannes** : Ajoutez un état **Panne** auquel le système bascule en cas de défaillance.

**Exercice 2 :**

Récupérer le code de la classe VendingMachine (Distributeur Automatique) sur Moodle.

Vous êtes chargé d'analyser et de modéliser une **machine à états UML** à partir d'un code Java existant qui simule le fonctionnement d'un distributeur automatique de boissons. Votre objectif est de comprendre le comportement du système et de représenter ses états, transitions, et actions sous forme de diagramme UML.

## Tâches :

1. **Analysez les États** :
    - Identifiez les états possibles de la machine d'après le code.
    - Listez les activités associées à chaque état (ex. : attendre, distribuer un article, accepter de l'argent).
2. **Identifiez les Transitions** :
    - Repérez les événements qui entraînent des transitions entre états (ex. : insertion d'argent, distribution d'un article, mise hors service).
3. **Modélisez la Machine à États** :
    - Créez un **diagramme UML de machine à états** qui illustre :
        - Les **états** : IDLE, WAITING_FOR_MONEY, DISPENSING_ITEM, OUT_OF_ORDER.
        - Les **transitions** déclenchées par des événements (ex. : `insertMoney`, `dispenseItem`, `setOutOfOrder`, `resetMachine`).
        - L'état **initial** (IDLE) et les autres transitions possibles.
4. **Ajoutez des Garde-Fous** :
    - Ajoutez des conditions aux transitions lorsque cela est nécessaire (ex. : le montant inséré est suffisant pour passer à DISPENSING_ITEM).
5. **Utilisez un éditeur UML** :
    - Dessinez le diagramme avec un outil comme Visual Paradigm ou PlantUML.

## Étapes Supplémentaires :

**I. Modéliser les cas limites suivants :**

1. **Annulation de la transaction** :
    - Permettre à l'utilisateur d'annuler une transaction lorsqu'il est en état `WAITING_FOR_MONEY`.
    - Transition vers `IDLE` avec remboursement du solde inséré.
    - **Exemple de transition UML** : `cancelTransaction()`.

2. **Panne pendant une transaction** :
    - Si la machine tombe en panne (`setOutOfOrder`) pendant qu'elle attend de l'argent ou distribue un article, l'argent inséré doit être restitué.
    - Ajoutez un mécanisme pour gérer la restitution dans l'état `OUT_OF_ORDER`.

**II. Ajoutez une gestion des stocks dans le code et dans le diagramme UML :**

1. **Nouveau paramètre** : `int stock`.
    - Si `stock == 0`, la machine ne peut plus distribuer d'article.
2. **Transition vers un nouvel état** :
    - Ajoutez un état `OUT_OF_STOCK`.
    - Transition depuis `IDLE` ou `WAITING_FOR_MONEY` si `stock == 0`.
    - Une méthode `refillStock()` permet de réinitialiser le stock et de retourner à `IDLE`.

**III. Ajoutez des éléments détaillés pour mieux représenter les comportements :**

1. **Transitions internes** :
    - Indiquez les actions internes dans les états, comme `do` pour l'état `DISPENSING_ITEM` (`do: Reduce stock by 1`).
2. **Événements conditionnels** :
    - Marquez clairement les gardes (conditions sur les transitions) :
        - `balance >= itemPrice`.
        - `stock > 0`.
    - Utilisez des notations UML pour les transitions conditionnelles (garde entre crochets : `[condition]`).
        - Exemple : `WAITING_FOR_MONEY --> DISPENSING_ITEM : [balance >= itemPrice && stock > 0]`.

**IV. Ajoutez des fonctionnalités supplémentaires au distributeur, comme :**

1. **Choix d'articles** :
    - Ajoutez un état `SELECTING_ITEM` où l'utilisateur choisit un produit spécifique.
    - La transition vers `WAITING_FOR_MONEY` se fait après la sélection de l'article.
2. **Paiement par carte** :
    - Ajoutez un événement `payWithCard()`.
    - Permettez à la machine de sauter l'état `WAITING_FOR_MONEY` et de passer directement à `DISPENSING_ITEM`.

**V. Implémentation et validation :**

1. **Mettre à jour le code Java** :

    Implémentez les nouvelles fonctionnalités (gestion des stocks, paiement par carte, etc.).
2. **Vérifier la cohérence UML <-> Code** :

Assurez-vous que chaque état et transition du diagramme UML correspond à une logique dans le code.

3. **Tester les scénarios** :

   Écrire des cas de test (JUnit) pour valider le fonctionnement correct des transitions.

**VI. Intégration avec une Interface Utilisateur**

Simulez une interface utilisateur textuelle ou graphique pour interagir avec le distributeur.

1. Ajouter un menu interactif permettant :
   - D’insérer de l’argent.
   - De sélectionner un article.
   - D'annuler une transaction.
2. Analysez comment les interactions avec l’interface affectent les états de la machine.