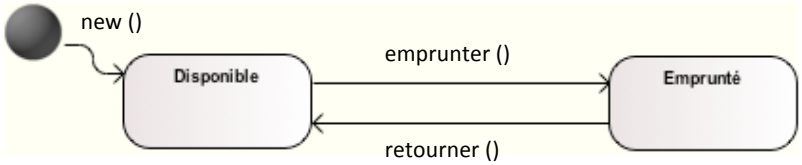
Objectif du TP : valider le code Java d'une classe à partir de son diagramme d'états-transitions UML. Il s'agit de concevoir des classes de tests automatiques JUnit à partir de diagrammes d'états-transitions.

Indications de mise en œuvre : il vous est proposé de suivre la démarche suivante :

1. Tous les états définis dans le diagramme d'états-transitions sont codés dans un type énuméré (*enum* en java).
2. Le diagramme d'états-transitions est codé par une seule classe de test incluant :
    a. Une méthode privée *EtatClasseTestée getEtat (ClasseTestée o)* permettant de retourner l'état courant d'un objet de la classe testée.
    b. Une méthode de test de l'état initial.
    c. Une méthode de test par transition.

Exercice 1 – Tests de la classe *Livre*

On gère l'emprunt de livres. La classe *Livre* et son diagramme d'états-transitions sont donnés comme suit :



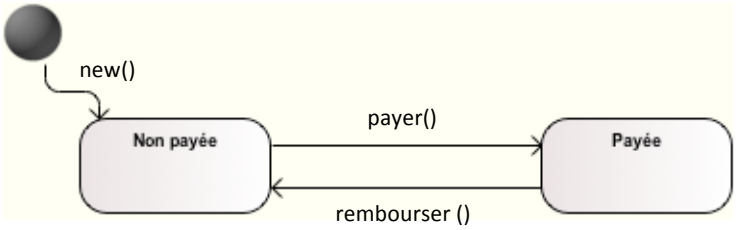
Travail demandé

Valider le code Java (fourni) de la classe *Livre* avec le diagramme d'états-transitions ci-dessus.

1. Préparation du projet Eclipse de test
    a. Créer un nouveau projet Java.
    b. Importer dans le projet les classes *Livre* et *TestLivreAvecDE* et l'énumération *EtatsLivre*.
    c. Analyser ce code Java, notamment celui de la classe de test et de son énumération.
2. Test de l'état initial (test fourni)
    a. Exécuter les tests. Constat ? Quels sont les tests qui ne réussissent pas ?
    b. Vérifier le code Java de la classe Livre et expliquer pourquoi le test de l'état initial ne passe pas.
    c. Corriger le code Java de la classe pour que le test de l'état initial réussisse.
    d. Réexécuter les tests ? Le test de l'état initial réussit !
3. Test de la transition em*prunter ()* (test fourni)
    a. Expliquer pourquoi le test de cette transition ne réussit pas ?
    b. Corriger le code Java de la classe *Livre* pour que le test de la transition réussisse.
    c. Réexécuter les tests ? Le test réussit !
4. Ecriture d'une méthode de test de la transition *retourner ()*
    a. La classe de test *TestDELivre* couvre-t-elle entièrement le diagramme d'états-transitions de la classe *Livre* ? Quelles sont les transitions non couvertes.
    b. Ecrire une méthode de test de la transition *retourner ()*.
    c. Exécuter le test. Constat ?
    d. Corriger le code Java fourni pour que le test réussisse.
    e. Exécuter le test. Constat ? Maintenant tous les tests passent ?! Classe validée, bravo !!!

Exercice 2 – Tests de la classe *Commande*

On gère le paiement et le remboursement des commandes. On met à votre disposition les modèles UML de la classe *Commande* ci-dessous et son code Java (à récupérer sur le *Commun*).



Travail demandé

A partir du diagramme d'états-transitions ci-dessus, écrire une classe Java de tests permettant de tester et, le cas échéant, corriger le code Java de la classe *Commande*. Suivre les la marche suivante :

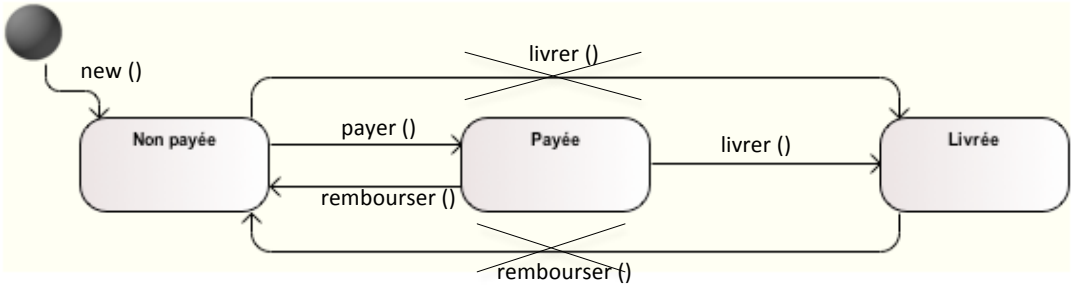
1. Préparations d'avant test
    a. Créer un nouveau projet Java.
    b. Importer dans le projet la classe *Commande.java* (à récupérer du *Commun*).
    c. Coder les états du diagramme d'états-transitions dans l'énumération *EtatsCommande.java*.
    d. Créer la classe de test *TestCommandeAvecDE*.
    e. Ecrire la méthode privée *EtatsCommande getEtat (Commande uneCommande)* de la classe *TestCommandeAvecDE* permettant de renvoyer l'état courant d'un objet *Commande*.
2. Test de l'état initial
    a. Ecrire une méthode de test de l'état initial (*testEtatInitial()*).
    b. Exécuter le test. Constat ? Expliquer pourquoi le test de l'état initial ne passe pas.
    c. Corriger l'erreur dans la classe *Commande.java* pour que le test de l'état initial réussisse.
    d. Réexécuter le test. Le test de l'état initial réussit !
3. Test de la transition *payer ()*
    a. Ecrire une méthode de test (*testTransitionPayer()*) de la transition *payer ()*.
    b. Exécuter le test. Constat ?
    c. Corriger le code Java fourni pour que le test réussisse.
    d. Exécuter le test.
4. Test de la transition *rembourser ()*
    a. Ecrire une méthode de test (*testTransitionRembourser()*) de la transition *rembourser ()*.
    b. Exécuter le test. Constat ?
    c. Corriger le code Java fourni pour que le test réussisse.
    d. Exécuter le test.

Exercice 3 – Tests de la classe *Commande* (suite)

On rajoute la livraison des commandes selon les deux règles de gestion suivantes :

- Une commande ne peut être livrée qu'après paiement.
- Une fois livrée, la commande ne peut plus être remboursée.

Les modèles UML de la classe *Commande* sont données comme suit :



Travail demandé

1. Compléter les classes UML et Java pour gérer la livraison des commandes.
2. Ajouter l'état « Livrée » à l'énumération *EtatsCommande* puis dans la méthode *getEtat (Commande uneCommande)* de la classe de test *TestCommandeAvecDE*.
3. Ecrire les méthodes de tests suivantes dans la classe *TestCommandeAvecDE* :
    a. Test de la transition *livrer ()*.
    b. Test de la transition non autorisée *livrer ()*.
    c. Test de la transition non autorisée *rembourser ()*.
4. Exécuter les tests et corriger le code de la classe *Commande* jusqu'à ce que tous les tests réussissent.

Exercice 4 (supplémentaire) – Tests de la classe *Absence*

On développe une application de gestion d'absences des élèves (numéro d'étudiant et nom) en cours (date, heure de début). Les élèves ne sont pas organisés en groupes (pas de gestion de groupes).

Les absences sont enregistrées par les enseignants pendant les cours et concerne un élève.

L'élève doit justifier son absence dans les deux semaines. Passé ce délai, l'absence est confirmée.

Le directeur des études étudie les justificatifs d'absence qui les accepte ou pas. Les absences sont confirmées si les justificatifs ne sont pas acceptés.

Un étudiant est considéré défaillant s'il a cumulé plus de 3 absences confirmées.

Travail demandé

1. Faire le diagramme de classes de l'application et les diagrammes d'états-transitions des absences et des élèves.
2. Coder l'application modélisée en Java (seulement les trois classes Elève, Cours et Absence).
3. Ecrire des classes de tests de *Absence* et *Elève* à partir de son diagramme d'états-transitions.
4. Exécuter les tests et corriger le code si nécessaire.