TD2 : Rétro-conception de diagrammes de séquence

R.3.04 : Qualité de développement

2025/2026

Ce TD a pour objectif de mettre en pratique un processus d'analyse de code Java et de création de diagrammes de séquence qui documentent certaines interactions entre objets dans le code.

1 Exercice 1 : Rétro-conception d'un diagramme de séquence

Analyser le code Java fourni ci-dessous et sur Moodle pour comprendre le processus d'emprunt de livre.

```java
// User.java
public class User {
 private String name;
 public User(String name) {
  this.name = name;
 }
 public String getName() {
  return name;
 }
 public void borrowBook(Book book, Library library) {
  if (library.isBookAvailable(book)) {
   library.lendBook(book);
   System.out.println(name + " successfully borrowed
       the book: " + book.getTitle());
  } else {
   System.out.println("Sorry, " + name + ". The book
       is not available.");
  }
 }
}

// Book.java
```

```java
public class Book {
 private String title;
 public Book(String title) {
  this.title = title;
 }
 public String getTitle() {
  return title;
 }
}

// Library.java
import java.util.HashMap;
public class Library {
 private Map<Book, Integer> inventory = new HashMap
     <>();
 public void addBook(Book book, int quantity) {
  inventory.put(book, inventory.getOrDefault(book, 0)
      + quantity);
 }
 public boolean isBookAvailable(Book book) {
  return inventory.getOrDefault(book, 0) > 0;
 }
 public void lendBook(Book book) {
  if (isBookAvailable(book)) {
   inventory.put(book, inventory.get(book) - 1);
  }
 }
}
// Main.java
public class Main {
 public static void main(String[] args) {
  Library library = new Library();
  Book book1 = new Book("The Great Gatsby");
  Book book2 = new Book("1984");
  library.addBook(book1, 3);
  library.addBook(book2, 1);
  User user = new User("Alice");
  // Borrowing books
  user.borrowBook(book1, library);
  user.borrowBook(book2, library);
  user.borrowBook(book2, library);
 }
}
```

Créer un diagramme de séquence en utilisant PlantUML pour représenter le processus d'emprunt de livre. Prenez comme point de départ de l'interaction la

méthode `main()`.

2 Exercice 2 : Extension de l'application

- Gestion des réservations : ajouter une classe `Reservation` encapsulant un objet `User`, un objet `Book` et une date de réservation. Mettre à jour la classe `Library` pour permettre la réservation de livres en cas d'indisponibilité.
- Gestion d'un historique d'emprunts et des retards :
  1. Ajouter une classe `Loan` avec des attributs représentant un `User`, un `Book`, une date d'emprunt et une date de retour prévu (de type `LocalDate`), et `returned` de type `boolean` pour indiquer que le livre est retourné. Munir cette classes de méthodes pour le retour d'un livre (mettre à jour l'attribut booléen), et pour savoir si un emprunt est en retard (par rapport à la date actuelle).
  2. Mettre à jour la classe `User` pour sauvegarder un historique d'emprunts (`loanHistory` : une liste d'objets `Loan`). Mettre à jour la méthode `borrowBook` et ajouter d'autres méthodes, le cas échéant.
  3. Mettre à jour la classe `Library` pour stocker une liste d'emprunts actifs (`activeLoans`) et permettre le retour de livres afin d'actualiser la liste d'emprunts actifs et déclencher la notification du prochain utilisateur, qui a réservé le livre. La notification consiste simplement en l'affichage d'un message sur la console et la suppression de la réservation.
- Gestion des utilisateurs premium (utilisateurs pouvant emprunter un grand nombre de livres) : mettre à jour la classe `User` avec un attribut `isPremium` de type boolean. Mettre à jour les méthdoes nécessaires. Ajouter par exemple une méthode `canBorrowMoreBooks()` qui permet de savoir si un utilisateur peut emprunter d'autres livres ou non.

Implémenter les nouvelles classes et méthodes en Java. Écrire un code minimaliste en respectant les bonnes pratiques de codage : nommage pertinent des classes, méthodes, ... faire en sorte que les tailles des méthodes et classes ne soient pas grandes, ...

3 Diagrammes de séquence

Un utilisateur premium essaie d'emprunter un livre non disponible, le réserve, puis est notifié quand le livre est retourné par un autre utilisateur.

Créer le diagramme de séquence avec PlantUML pour le scénario décrit (avec deux participants de type acteur et des objets de l'application).

Créer un autre diagramme qui modélise l'interaction liée à la gestion des retards. Quel est le scénario d'utilisation idéal pour vous ? Ajouter le code nécessaire. Prévoir un code simple à écrire. Ne pas utiliser de solutions avec des entrées/sorties complexes : envoi de emails, ...

4 Questions pour approfondir

1. Comment gérer les pénalités de retard ? Imaginez une solution simple en termes de code et modéliser un scénario d'utilisation de cette *feature*, sous la forme d'un diagramme de séquence.

2. Comment étendre l'application pour gérer les retours de livres défectueux (les retirer de l'inventaire, les envoyer en réparation, notifier l'utilisateur) ? Proposer un diagramme de séquence qui prend cela en compte.