R3.04 : Qualité de développement
TP1 : Conception et rétro-conception

2025/2026

1 Introduction

Ce TP a pour objectif de vous familiariser avec la création de diagrammes UML (diagrammes de classes et de séquence) en utilisant PlantUML. Vous allez également pratiquer la rétro-conception à partir de code Java.

2 Objectifs

— Maîtriser la création de diagrammes de classes et de séquence avec PlantUML.
— Appliquer la rétro-conception à partir de code Java.
— Analyser les relations entre classes et les interactions dynamiques.

3 Exercice 1 : Rétro-conception du diagramme de classes

Un extrait du jeu de Nim est fourni. L'objectif est de reconstruire un modèle UML à partir du code source.

3.1 Étapes

1. Téléchargez le code Java du jeu de Nim depuis Moodle.
2. Analysez le code pour identifier les classes, attributs, méthodes et relations.
3. Créez un fichier `.puml` pour le diagramme de classes.
4. Améliorez le diagramme :
   — Ajoutez les dépendances avec les stéréotypes vus en cours.
   — Masquez les attributs redondants (remplacés par des associations).

4 Exercice 2 : Extension du jeu

Ajoutez les éléments suivants au code puis reportez-les dans le diagramme de classes :

- Introduire une interface `PlayerStrategy` pour formaliser la stratégie de jeu (y ajouter la méthode : `takeTurn(int heapSize)`). `HumanPlayer` et `ComputerPlayer` implémenteraient cette interface indirectement via la classe `Player` (faire en sorte que cette dernière implémente l'interface. Y retirer du coup la méthode `takeTurn(...)`. Objectif : l'interface permet ici de standardiser les comportements et de faciliter l'ajout de nouvelles stratégies.‘
- Ajouter une classe `SmartComputerPlayer` qui implémente une stratégie gagnante (basée sur un `move` calculé à partir de la taille de tas modulo 4, par exemple) :

```
public int takeTurn(int heapSize) {
  // Strategie optimale pour un seul tas
  int move = heapSize % 4;
  if (move == 0) {
   // Si le tas est un multiple de 4, on prend
   // un nombre aleatoire entre 1 et 3
   move = 1 + random.nextInt(3);
  }
  System.out.println(name + " removes " + move + "
      objects (smart move).");
  return move;
}
```

- Ajouter une classe `ScoreBoard` pour suivre les scores des joueurs sur plusieurs parties.

5 Exercice 3 : Rétro-conception du diagramme de séquence

Dessinez les diagrammes de séquence pour :

- Séquence d'un tour de jeu : Montrer l'interaction entre NimGame, Player, et Scanner pour un tour complet (demande de coup, validation, mise à jour du tas, changement de joueur).
- Séquence de fin de partie : Illustrer le cas où un joueur prend le dernier objet et perd la partie. L'idée est de modéliser les conditions et les messages de fin de jeu.
- Séquence avec stratégie intelligente : Montrer comment SmartComputerPlayer calcule son coup en utilisant une méthode interne (ex : calculateWinningMove(int heapSize)). L'objectif est de montrer comment on peut décomposer les appels de méthodes complexes.

5.1 Exemple de syntaxe

Voici un exemple de syntaxe pour un diagramme de séquence :

```
@startuml
actor User
participant "NimGame" as game
participant "HumanPlayer" as human
participant "ComputerPlayer" as computer
participant "Scanner" as scanner

User -> game: play()
game -> human: takeTurn(heapSize)
human -> scanner: nextInt()
scanner --> human: move
human --> game: move
game -> game: heapSize -= move
alt heapSize == 0
    game -> User: "You lose!"
else
    game -> computer: takeTurn(heapSize)
    computer --> game: move
    game -> game: heapSize -= move
end
@enduml
```

6 Rendu du TP

6.1 Format

- Une archive `.zip` contenant :
  - Les fichiers `.puml` pour chaque diagramme.
  - Un fichier PDF généré à partir des diagrammes.
  - Les sources modifiées.
- Nom de l'archive : `TP1_Nom_Prenom.zip`.

6.2 Remarques

- Les diagrammes automatiques sans amélioration personnelle ne seront pas notés.
- Utilisez des commentaires dans vos fichiers `.puml` pour expliquer vos choix de modélisation.