

# TD 2

## Pré-requis

- Java 17+
- Outil au choix: IntelliJ/VS Code + JDK. Optionnel: Maven si vous prenez l'exo sérialisation JSON avec Jackson.

## Objectifs pédagogiques

- Encapsulation, invariants, types énumérés
- Exceptions métier et validations
- Composition (Inventaire, Objet)
- Journalisation et petites I/O
- Méthodes d'instance vs de classe
- Égalité/hash et identité
- Petite boucle CLI pour "faire vivre" l'objet

## Partie A — Énoncés

### Exercice 1 — Classe Personnage minimale

Créez une classe Personnage avec:

- Attributs privés:
  - id (UUID, immuable)
  - nom (String, non vide)
  - pv (int, borné 0-100)
  - position x, y (int)
  - statut (Enum Statut: HORS\_LIGNE, EN\_LIGNE, EN\_COMBAT, KO)
  - journal (List<String>) pour tracer les actions
- Constructeur: Personnage(String nom, int pvInitial, int x, int y)
- Méthodes publiques:
  - seConnecter(), seDeconnecter()
  - entrerCombat(), sortirCombat()
  - prendreDegats(int n), soigner(int n)
  - deplacer(int dx, int dy)

#### Invariants/règles:

- $0 \leq pv \leq 100$  à tout moment.
- Si  $pv == 0$ , statut = KO; impossible de se déplacer.
- Actions interdites si HORS\_LIGNE (sauf seConnecter()).
- Impossible de seDeconnecter() depuis EN\_COMBAT.

Encapsulation: exposez uniquement des getters en lecture (id, nom, pv, position, statut), pas de setters publics. Ajoutez un toString() lisible (nom, pv, statut, position).

## Exercice 2 — Exceptions métier et validation

Créez une exception `ActionInterdite` extends `RuntimeException`.

Toutes les violations de règles jettent `ActionInterdite` avec un message clair.

Validez les paramètres (ex: dégâts/soins négatifs  $\Rightarrow$  clamp à 0 ou exception, à vous de choisir, justifiez).

## Exercice 3 — Journalisation

- Méthode privée `log(String msg)` qui préfixe chaque entrée par un timestamp ISO (yyyy-MM-dd HH:mm:ss).
- Loggez les actions significatives (connexion, déconnexion, dégâts, soin, déplacement, transitions d'état).
- Ajoutez `getJournal()` qui renvoie une copie non modifiable (`List.copyOf(...)`).

## Exercice 4 — Composition: Inventaire et Objet

- Classe `Objet { String nom; double poids; }` immuable.
- Classe `Inventaire { double capaciteMax; List<Objet> internes; }`
- `double poidsTotal()`
- `void ajouter(Objet o)` — interdit si `poidsTotal + poids > capaciteMax`  $\Rightarrow$  `ActionInterdite`
- `Objet retirer(String nom)` — retire le premier objet de ce nom ou exception

Ajoutez un champ inventaire dans `Personnage`; getter en lecture.

## Exercice 5 — Communication “admin réseau”

Méthode `envoyerMessage(Personnage cible, String contenu)`: Interdit si émetteur `HORS_LIGNE` ou cible `HORS_LIGNE`  $\Rightarrow$  `ActionInterdite` Loggez “Msg  $\rightarrow$  <cible>: <contenu>” côté émetteur

Méthode `simulatePing()`: renvoie une latence simulée en millisecondes (int entre 5 et 200), interdit si `HORS_LIGNE`.

## Exercice 6 — Sérialisation (au choix)

Option A (sans dépendance):

`String toJson()` — construisez une chaîne JSON simple via `StringBuilder` (attention aux guillemets, échappez au minimum nom/objets). `static Personnage fromJson(String json)` — parse minimaliste: pour simplifier, vous pouvez ignorer l'inventaire ou parser seulement nom/pv/position/statut. Option B (avec Jackson): Ajoutez `Jackson Databind`. Sérialisez/désérialisez `Personnage` (id lu/écrit en `String`). Marquez les champs non voulus si besoin.

## Exercice 7 — Identité, égalité, hash

`id` (UUID) défini au constructeur et jamais modifié. Implémentez `equals()` et `hashCode()` basés uniquement sur `id`. Démonstration: mettez des `Personnage` dans un `HashSet` et vérifiez l'unicité.

## Exercice 8 — Méthodes de classe (“usines”)

static Personnage nouveau(String nom) — valeurs par défaut: pv=100, x=0, y=0. static Personnage fromSnapshot(...) — construit depuis des valeurs primaires (ex: pour tests).

## Exercice 9 — Petite CLI

Programme Main lisant des lignes sur stdin: Commandes: connect, disconnect, fight, stopfight, move dx dy, hit n, heal n, ping, status, additem nom poids, rmitem nom, json, quit Gérez proprement les exceptions (afficher “Erreur: ...”).

Affichez l’état après chaque commande utile.

## Exercice 10 — Tests unitaires (JUnit 5)

Tests suggérés: PV bornés (ne pas dépasser 100; pas négatif). Transitions d’états interdites (ex: disconnect depuis EN\_COMBAT). Inventaire plein ⇒ exception. KO ⇒ déplacement impossible; premier soin réanime à min 1 PV (si vous adoptez cette règle). equals/hashCode par id.

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/cnam/utc503/td2-c?rev=1756721737>

Last update: **2025/09/01 12:15**

