🐉 Projet Java — RPG Console (version étendue, 3 monstres)

🎯 Pitch

Développez un RPG textuel en Java. Créez un héros (`Guerrier`, `Mage`, ou `Archer`) et affrontez des monstres (`Gobelin`, `Troll`, `Dragon`) dans des combats au tour par tour. Gagnez de l'**XP** et de l'**or**, achetez des **objets** (potions, trinkets, armes) à la **boutique**, **équipez**-vous, et reposez-vous à la **taverne**.

L'interface est 100% **console** (menus numériques).

🎓 Objectifs pédagogiques (M320)

- **Encapsulation** : attributs `private`, getters/setters.
- **Héritage / classes abstraites** : `Personnage` → `Heros` / `Monstre` → classes concrètes.
- **Polymorphisme** : `competenceSpeciale()`, `Objet.utiliser(...)`.
- **Séparation des responsabilités** : métier vs. I/O (ConsoleIO) vs. orchestration (Main).
- **Collections** : gestion d'inventaire, offres boutique.
- **Validation** : contrôles d'indices/entrées pour ne pas faire planter le programme.

🧱 Architecture (vue d'ensemble)

- `ConsoleIO` : lecture d'entiers/chaînes.
- `Personnage` *(abstraite)* : PV, ATK, DEF, attaque de base **±20%**.
  - `Heros` *(abstraite)* : XP/level, **or**, inventaire, **slot d'arme**.
    - `Guerrier`, `Mage`, `Archer` : compétences spéciales distinctes.
  - `Monstre` *(abstraite)* : butin XP + or.
    - `Gobelin` (facile), `Troll` (intermédiaire), `Dragon` (difficile).
- `Objet` *(abstraite)* : `utiliser(Heros)`.

- `Potion` (soin), `Trinket` (bonus permanent), `Arme` (équipable).

- `Boutique` : achat d'objets (clonage propre via getters).
- `Taverne` : repos payant (PV → max).
- `Main` : menu principal, **pré-combat** (gérer inventaire, puis combattre), combat.

🧩 Règles métier

- **Attaque de base** : dégâts = ATK * (80%..120%) – DEF, min 1.
- **Compétences** :
  - Guerrier → `Coup Puissant` (ATK×1.5, DEF adverse ÷2),
  - Mage → `Boule de Feu` (coût mana, ignore DEF),
  - Archer → `Tir Précis` (50% critique 2×, sinon ~1.25×).
- **Monstres** :
  - Gobelin → `Coup Bas` (petits dégâts fixes),
  - **Troll** → `Massue Écrasante` (18 dégâts – 70% de DEF),
  - Dragon → `Souffle de Feu` (28 dégâts – 70% de DEF).
- **Gains** : Victoire = +XP + or (selon le monstre).
- **Inventaire** : potions (soin), trinkets (bonus permanents), armes (bonus ATK).
- **Armes** : une seule équipée (l'ancienne revient dans l'inventaire).
- **Taverne** : PV restaurés pour un coût fixe.
- **Pré-combat** : permet de s'équiper/utiliser avant d'affronter.

▶️ Menu cible

```
=== RPG Simplifié ===
[1] Voir le héros
[2] Préparer un combat (gérer inventaire) puis combattre
[3] Voir l'inventaire / utiliser ou équiper
[4] Boutique (acheter objets)
```

```
[5] Taverne (se reposer)
[0] Quitter
```

Pendant le combat :

`1) Attaquer` · `2) <Compétence spéciale>` · `3) Utiliser un objet` · `4) Fuir`

🛠️ Contraintes techniques

- 1 **classe = 1 fichier** (nom de fichier = nom de classe).
- Attributs **private**, `@Override` sur redéfinitions.
- I/O **uniquement** dans `Main` / `ConsoleIO`.
- Évitez le `static` sauf là où nécessaire.
- Validez les indices et entrées pour éviter les crashs.

🗺️ Progression par jalons (détaillée, avec DoD)

> DoD (Definition of Done) = critères de validation à cocher avant de passer au jalon suivant.
> Rappel M320 : 1 classe = 1 fichier, attributs private, @Override sur redéfinitions, aucune I/O dans le métier (seulement dans Main/ConsoleIO).

---

1) Console & squelette

But : créer le socle d'I/O et un menu minimal.

À créer

- `ConsoleIO`

```
public final class ConsoleIO {
    public static int lireInt(String prompt) { /* println + parse, -1 si invalide */ }
    public static String lireStr(String prompt) { /* println + readLine */ }
}
```

- `Main` (menu vide)
  Afficher :

```
[1] Voir le héros
[0] Quitter
```

Test manuel

- Lancer, saisir `1`, `abc`, `0` → pas d'exception, retour propre.

DoD

- [ ] `ConsoleIO` lit proprement int/str (retourne `1` si invalide).
- [ ] Menu s'affiche, boucle principale fonctionne.

---

2) `Personnage` (abstraite)

But : base commune pour héros/monstres (stats, attaque ±20%).

À créer

- `Personnage`
  Attributs `private` : `nom`, `niveau=1`, `pointsDeVieMax`, `pointsDeVie`, `attaque`, `defense`.
  Constructeur : borne `pvMax≥1`, `atk≥0`, `def≥0`.
  Méthodes :

```
public boolean estVivant()
public void soigner(int pv)
public int attaquer(Personnage cible)   // ATK * (80..120%) - DEF, min 1
protected void subirDegats(int d)       // borne à 0
public abstract String competenceSpecialeNom();
public abstract int competenceSpeciale(Personnage cible);
// + getters protégés / setters protégés utiles aux sous-classes
```

  Astuce testabilité (facultatif) :

```
protected int rollPourcentDegats() { return ThreadLocalRandom.current().nextInt(80,121); }
```

Test manuel

- Créer temporairement deux sous-classes de test (dans `Main`) et appeler `attaquer` plusieurs fois : les dégâts doivent varier (±20%), minimum 1, PV ne passent jamais < 0.

DoD

- [ ] Attaque utilise bien la fourchette **80–120%**.
- [ ] Dégâts finaux = `max(1, bruts - DEF)`.
- [ ] Invariants PV respectés.

---

3) `Heros` (abstraite)

But : progression (XP/niveau), économie (or), inventaire, boosts, slot d'arme.

À créer

- `Heros extends Personnage`
  Attributs `private` : `experience=0`, `or=50`, `List<Objet> inventaire`, `Arme armeEquipee=null`.
  Méthodes :

```
// Inventaire
public void ajouterObjet(Objet o)
public boolean utiliserObjetIndex(int index, Heros cible) // retire et o.utili
ser(cible)
public List<Objet> getInventaire()

// XP / Niveau
public void gagnerXP(int xp) // 100 ⇒ niveau +1
// level up: +10 PVmax, PV=PVmax, +2 ATK, +1 DEF

// Or
public int getOr()
public void ajouterOr(int montant)      // ≥0
public boolean depenserOr(int montant)  // false si insuffisant

// Boosts permanents
public void boosterPvMax(int bonus)
public void boosterAttaque(int bonus)
public void boosterDefense(int bonus)

// Arme
public Arme getArmeEquipee()
```

```
public void equiperArme(Arme nouvelle)  // retire bonus ancienne (+ ret
our inventaire), applique bonus nouvelle
```

Test manuel

- `gagnerXP(100)` → niveau +1, PV=PVmax, +ATK/+DEF.
- `depenserOr(99999)` renvoie `false` sans changer l'or.
- Ajouter un objet fictif dans l'inventaire puis `utiliserObjetIndex(0, hero)` l'applique et retire l'objet.

DoD

- [ ] Montée de niveau appliquée exactement comme spécifié.
- [ ] Économie fiable.
- [ ] Inventaire modifiable, utilisation appelle `utiliser(Heros)`.

---

4) Héros concrets

But : 3 classes jouables avec compétences distinctes.

À créer

- `Guerrier extends Heros`
  Stats : `pvMax=90, atk=14, def=8`.
  Compétence : **Coup Puissant** → `degats = round(ATK * 1.5) - (DEF/2)` (borne min 1).
- `Mage extends Heros`
  Stats : `pvMax=70, atk=12, def=5`.
  Attributs : `mana=50`, `manaMax=50`, `regenMana(int)`.
  Compétence : **Boule de Feu** (coût 15 mana) → `degats = 20 + round(ATK * 0.8)` **ignore DEF**.
  Si mana insuffisante → retourne `0`.
- `Archer extends Heros`
  Stats : `pvMax=80, atk=13, def=6`.
  Compétence : **Tir Précis** (50% critique 2×, sinon ~1.25× `(ATK-DEF)` ; min 1).

Test manuel

- Vérifier que chaque compétence inflige des dégâts cohérents (et échec pour Mage si mana < 15).
- Exécuter plusieurs fois l'archer pour voir des coups critiques.

DoD

- [ ] 3 classes opérationnelles, `@Override` corrects.
- [ ] Comportements conformes aux règles ci-dessus.

---

5) `Monstre` (abstraite) + `Gobelin`

But : support butins XP/or + 1er monstre.

À créer

- `Monstre extends Personnage`
  Attributs `private` : `xpButin`, `orButin`. Getters.
- `Gobelin extends Monstre`
  Stats : `pvMax=50, atk=9, def=3, xp=40, or=20`.
  Compétence : **Coup Bas → 6 dégâts fixes**.

Test manuel

- Appeler `competenceSpeciale` sur un héros : PV -6.
- Lire `getXpButin()`, `getOrButin()`.

DoD

- [ ] Monstre abstrait prêt, Gobelin OK.

---

6) Combat simple

But : boucle de combat tour par tour (sans objets pour l'instant).

À coder dans `Main`

- `monstreAleatoire()` → renvoie au début un `Gobelin`.
- `combat(Heros h, Monstre m)` :
  - Tour du joueur : afficher `h` / `m`, menu : `1 Attaquer`, `2 <Compétence>`, `3 Fuir`.
  - Tour du monstre : attaque de base.
  - Fin : victoire/défaite → retourner au menu principal.

Test manuel

- Gagner/perdre un combat.
- Tester la fuite (ex. 40% de réussite).

DoD

- [ ] Pas de crash, combat se termine proprement.
- [ ] Dégâts visibles, cohérents.

---

7) Objets & inventaire (Potions)

But : consommer une potion en combat.

À créer

- `Objet` (abstraite)

```
public abstract class Objet {
    private final String nom;
    public Objet(String nom) { this.nom = nom; }
    public String getNom() { return nom; }
    public abstract void utiliser(Heros cible);
    @Override public String toString() { return getClass().getSimpleNam
e()+"("+nom+")"; }
}
```

- `Potion extends Objet`
  Attribut : `soin (≥1)`. `utiliser(Heros)` → soigne et affiche PV récupérés.

À modifier

- `Main.combat` : ajouter l'option `Utiliser un objet` qui ouvre l'inventaire et applique l'objet choisi.

Test manuel

- Blesser le héros, utiliser une potion → PV augmentent (borne PVmax).

DoD

- [ ] Inventaire consultable en combat.
- [ ] Potions consommées et retirées de l'inventaire.

---

8) Boutique

But : acheter contre or (objets clonés).

À créer

- `Boutique`
  - Catalogue interne `Offre(objet, prix)` : potions de différentes tailles, trinkets, armes (à venir).
  - `visiter(Heros h)` : afficher, choisir, **payer** (`depenserOr`), **cloner** l'objet (via getters) et **ajouter à l'inventaire**.
  - Clonage :

```
Potion  → new Potion(p.getNom(), p.getSoin())
Trinket → new Trinket(t.getNom(), t.getStat(), t.getBonus())
Arme    → new Arme(a.getNom(), a.getAtkBonus())
```

Test manuel

- Achat avec or insuffisant → message, rien ne change.
- Achat réussi → or baisse, objet reçu.

DoD

- [ ] Catalogue affiché, paiements stricts.
- [ ] Objets reçus sont **nouvelles instances** (pas partagés).

---

9) Trinkets (bonus permanents)

But : objets consommés qui augmentent définitivement des stats.

À créer

- `Trinket extends Objet`
  Enum `Stat { PV_MAX, ATTAQUE, DEFENSE }`, champ `bonus≥1`.
  `utiliser(Heros)` : applique bonus permanent ; si PV_MAX, remettre PV=PVmax.

Test manuel

- Appliquer successivement +ATK, +DEF, +PV_MAX → vérifier les stats.

DoD

- [ ] Bonus **persistants** (même après d'autres combats/achats).

---

10) Armes (équipement, slot unique)

But : une arme équipée à la fois ; remplacer remet l'ancienne dans l'inventaire.

À créer

- `Arme extends Objet`
  Champ `atkBonus≥1`. `utiliser(Heros)` = `equiperArme(this)`.

À vérifier dans `Heros`

- `equiperArme` :
  - si déjà équipée → **retirer** son bonus ATK et **remettre** l'arme dans l'inventaire ;
  - équiper la nouvelle et **ajouter** son bonus à ATK.

Test manuel

- Équiper Épée(+3) → ATK augmente.
- Équiper Arc(+4) → ATK = base+4 et Épée revient dans l'inventaire.

DoD

- [ ] Jamais deux armes équipées.
- [ ] Pas de cumul fantôme de bonus.

---

11) Taverne (repos payant)

But : restaurer PV au max contre or.

À créer

- `Taverne` avec `visiter(Heros)` : afficher prix, demander confirmation, `depenserOr`, `soigner(999999)`.

Test manuel

- Pas assez d'or → message *"Pas assez d'or"*.
- Assez d'or → PV = PVmax.

DoD

- [ ] Débit d'or correct.
- [ ] PV restaurés.

---

12) Pré-combat (préparer puis combattre)

But : permettre d'utiliser/équiper **avant** de lancer le combat.

À modifier dans `Main`

- Nouveau flux `[2] Préparer un combat` :
  - Sous-menu : `[1] Gérer inventaire` / `[2] Commencer combat` / `[0] Retour`.
  - `[1]` réutilise la même méthode d'inventaire que pendant le combat.

Test manuel

- S'équiper d'une arme **avant** et constater l'ATK en combat.

DoD

- [ ] Préparation claire, pas de duplication de code.

---

13) Monstres supplémentaires

But : variété d'adversaires.

À créer

- `Troll extends Monstre`
  Stats : `pv=90, atk=14, def=6, xp=80, or=45`.
  Compétence **Massue Écrasante** : `18 - round(DEF * 0.7)` (min 1).
- `Dragon extends Monstre`
  Stats : `pv=140, atk=18, def=10, xp=150, or=100`.
  Compétence **Souffle de Feu** : `28 - round(DEF * 0.7)` (min 1).

À modifier

- `monstreAleatoire()` → tirer sur 3 types (ex. 50% Gobelin / 35% Troll / 15% Dragon).

Test manuel

- Lancer plusieurs combats → rencontrer les 3, vérifier dégâts spéciaux.

DoD

- [ ] Tirage aléatoire visible.
- [ ] Compétences spéciales conformes.

---

14) UX & robustesse

But : expériences propres, pas de crash.

À faire

- Partout où on lit un index : **vérifier bornes** (0..n-1).
- En cas d'entrée invalide : afficher "Choix invalide" et **rester dans le menu**.
- Dans "Voir le héros" : afficher **or**, **XP/100**, **arme équipée** (ou "Aucune"), **taille inventaire**, (si `Mage`) **mana**.

Test manuel

- Taper `abc`, `1`, `999` dans les menus.
- Naviguer longtemps : pas d'exception.

DoD

- [ ] Aucune saisie ne fait planter.
- [ ] Messages explicites, lisibles.

(Optionnel) 15) Équilibrage & extensions

- Ajuster l'ATK des monstres au niveau du héros.
- Ajouter une 2e arme par classe (épée lourde, arc long, bâton ancien).
- Équipe de 2 héros (tour par tour en duo).
- Sauvegarde/chargement simple (fichier texte).

DoD

- [ ] Nouvelles règles bien isolées, pas de régression du flux de base.

✅ Récap "mini checklist" par jalon

- **2** : Attaque ±20% OK, min 1, PV bornés.
- **3** : XP→level up + bonus, or fiable, inventaire/objets utilisables.
- **4** : 3 héros, comp' distinctes (mana pour Mage).
- **5** : Monstre/Butins OK, Gobelin spécial=6.
- **6** : Combat boucle complète, fuite possible.
- **7** : Potions soignent en combat.
- **8** : Boutique paie + clone → inventaire.
- **9** : Trinkets = bonus permanents.

- **10** : Armes = slot unique, swap propre.
- **11** : Taverne restaure PVmax, paie.
- **12** : Préparer puis combattre.
- **13** : Troll/Dragon + tirage 3 types.
- **14** : Aucun crash, UX claire.

✅ Checklist de rendu

- [ ] Le jeu compile et tourne en console.
- [ ] Héros jouable avec compétence spéciale.
- [ ] 3 monstres (Gobelin, Troll, Dragon) et butins.
- [ ] XP/or gagnés, niveau augmente.
- [ ] Boutique (potions/trinkets/armes) opérationnelle.
- [ ] Inventaire : utiliser (soins/bonus), **équiper** (arme).
- [ ] Taverne : repos payant, PV restaurés.
- [ ] Pré-combat : gestion avant d'engager.
- [ ] Code : encapsulation, `@Override`, I/O isolées.

Bon courage et amusez-vous bien à faire évoluer votre RPG ! 🎮🐉