Héritage Java: L'art de réutiliser le code!

Au Menu du Jour:

L'Héritage: C'est quoi et comment ça marche (extends)?

Surcharge vs Redéfinition: Deux techniques à maîtriser!

Super: L'appel à la classe maman!

Polymorphisme: Un mot barbare pour un concept génial!

Personnage, Joueur, Monstre: Un exemple concret pour tout comprendre

L'Héritage: La base de la POO

- L'héritage, c'est comme avoir les gènes de tes parents: tu récupères des trucs!
- Une classe (fille) hérite des propriétés et méthodes d'une autre (mère/parent)
- Mot-clé: `extends`

Héritage: Exemple en Code

- public abstract class Personnage {...}
- public class Joueur extends Personnage {...}
- public class Monstre extends Personnage {...}
- Joueur et Monstre héritent de Personnage

Surcharge vs Redéfinition: Attention à ne pas mélanger!

- Surcharge (overloading): Mêmes noms, paramètres différents
  - Comme avoir plusieurs fonctions 'addition' qui acceptent des nombres différents
- Redéfinition (overriding): Changer le comportement d'une méthode héritée
  - Comme réécrire une recette de cuisine pour l'améliorer

Redéfinition en Code

- @Override //Pour être sûr de pas se tromper!
- public String toJSON() { return gson.toJson(this); }
- Joueur et Monstre adaptent la méthode `toJSON()` de Personnage

Super: Un coup de fil à Maman!

- super() : Appelle le constructeur de la classe mère
- Toujours en PREMIER dans le constructeur de la classe fille!
- super.methode() : Appelle une méthode de la classe mère

Exemple Super en Code

- public Joueur(String nom, ..., int y, int niveau, ...) {
- super(nom, force, dexterite, armure, vitesse, viemax, viecourante, mindamage, maxdamage, x, y);
- this.niveau = niveau;
- // ...
- }

Polymorphisme: Tous différents, tous égaux!

- Polymorphisme = Pouvoir traiter des objets de types différents de la même façon
  - Comme dire à tous les animaux de 'manger', même si un chat mange des croquettes et un chien des os

Polymorphisme: Exemple en Code

- Personnage[] tab = new Personnage[2];
- tab[0] = new Joueur(...);
- tab[1] = new Monstre(...);
- for (Personnage p : tab) {
- System.out.println(p.getNom());
- System.out.println(p.toJSON()); // Appelle la bonne version!
- }

À toi de jouer! - Crée ton PNJ

- Crée une classe `PNJ` (Personnage Non Joueur) qui hérite de `Personnage`
- Ajoute un attribut `role` (String) avec getter/setter
- Implémente un constructeur appelant `super`
- Surcharge `toJSON()`

À toi de jouer! - Redéfini le Monstre

- Dans `Monstre`, redéfinis `toJSON()` pour ajouter un champ `"type":"Monstre"`
- Utilise `@Override` et `super.toJSON()`

À toi de jouer! - Polymorphisme en Action!

- Instanciez deux `Joueur` et un `Monstre`
- Stockez-les dans une `List<Personnage>`
- Parcourez la liste et appelez `toJSON()` sur chaque élément

À toi de jouer! - Super Constructeur!

- Dans `Joueur`, ajoute un nouveau constructeur (avec moins de paramètres)
- Utilise `this(...)` et `super(...)` pour déléguer

À toi de jouer! - Méthode de Caractérisation

- Écris une méthode statique `public static void afficherCaracteristiques(Personnage p) { ... }`
- Elle doit afficher `p.getNom()`, `p.getForce()`, `p.getVieCourante()`
- Teste avec un `Joueur` et un `Monstre`

Besoin d'aide? Ressources Utiles

- Héritage en Java (Oracle) : https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- Override vs Overload : https://www.geeksforgeeks.org/overloading-vs-overriding-in-java/
- Polymorphisme Java : https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html