Classes Java: Les Fondations de Tes Applications!

Ta Boîte à Outils:

Constructeurs: Des Usines à Objets!

Visibilité: Le Secret des Espions (private, public, protected)

Encapsulation: Le Coffre-Fort de Tes Données!

Membres Statiques: Partage les Ressources!

Constantes: Les Valeurs Immuables!

La Classe Inventaire: Notre Terrain de Jeu Concret!

Constructeurs: Crée Tes Objets à la Demande

- Constructeur: Une méthode spéciale qui donne vie à tes objets
  - Il a le même nom que la classe et ne renvoie rien
- Son rôle: Initialiser les attributs (donner des valeurs de départ)
- Surcharge (overloading): Plusieurs constructeurs, chacun avec sa propre recette

Constructeurs: Exemple en Code

- public class Inventaire implements Serializable {
- private final List<Item> objets;
- private int or;
- // Constructeur par défaut (sans arguments)
- public Inventaire() {
- this.objets = new ArrayList<>(); this.or = 100;
- }
- //Surcharge: Constructeur avec un montant d'or au départ
- public Inventaire(int orInitial) { this(); this.or = orInitial; }
- }

Visibilité: Sécurité Avant Tout!

- public: Tout le monde peut voir et utiliser!
- private: Seule la classe a le droit d'y toucher!
- protected: Les classes du même package et les enfants peuvent accéder
- Le but: Protéger les données et éviter les catastrophes!

Visibilité: Exemple Concret

- private int or; //L'or du joueur, on ne veut pas que n'importe qui puisse le voler!
- public boolean equiperObjet(...) { ... } //Tout le monde peut essayer d'équiper un objet!
- protected void helper() { ... } //Méthode utile pour les classes filles, mais pas pour les autres

Encapsulation: Protège Tes Précieuses Données!

- Encapsulation: Cacher les attributs et contrôler comment on y accède
  - C'est comme protéger tes codes secrets avec des coffres forts!
- Getters: Pour lire la valeur d'un attribut
- Setters: Pour modifier la valeur d'un attribut (avec des règles de sécurité!)

Encapsulation: Exemple de Code

- public int getOr() { //Getter: Je peux lire l'or
- return or;
- }
- public void setOr(int or) { //Setter: Je peux modifier l'or, mais...
- if (or >= 0 && or <= OR_MAX) { //...seulement si c'est positif et pas trop grand
- this.or = or; //C'est bon, je modifie!
- }
- }

Membres Statiques: Partage le Pouvoir!

- static: Appartient à la classe (pas à un objet en particulier)
- Variable statique: Une seule valeur pour tous les objets de la classe
- Méthode statique: S'utilise sans avoir besoin de créer un objet

Membres Statiques: Exemple en Code

- public class Inventaire {
- private static final int OR_MAX = 9999; //L'or max, une constante partagée par tous les inventaires
- private static int nombreInventairesCrees = 0; //Compte le nombre d'inventaires créés
- public Inventaire() { nombreInventairesCrees++; } //A chaque nouvel inventaire, on incrémente le compteur
- public static int getNombreInventairesCrees() { return nombreInventairesCrees; } //Accéder au compteur sans créer d'objet
- }

Constantes: Valeurs Pour l'Éternité!

- final: Une valeur qu'on ne peut JAMAIS changer
  - Comme le nom de la classe, le prix d'un item légendaire...
- On les écrit en majuscules (OR_MAX, PI, etc.) pour les repérer facilement

Constantes: Code d'Exemple

- private static final long serialVersionUID = 1L; //Un identifiant unique pour la sauvegarde
- private static final int OR_MAX = 9999; //La quantité maximale d'or qu'on peut stocker
- Ces valeurs ne changeront JAMAIS pendant la partie!

Illustrations et usages de la classe Inventaire

- private, public, protected, static, final: Tu connais leur pouvoir!
- Maintenant, utilise ces mots magiques à bon escient dans tes classes!

Exemples de code

- Inventaire inv1 = new Inventaire(); //Création d'un inventaire (il a 100 or par défaut)
- Inventaire inv2 = new Inventaire(500); //Création d'un inventaire avec 500 or au départ
- System.out.println(Inventaire.getNombreInventairesCrees()); //Affiche le nombre d'inventaires créés
- inv1.ajouterOr(200); //Ajoute 200 or à l'inventaire
- inv1.setOr(1200); //Modifie le montant d'or (si c'est autorisé par les règles)
- //boolean eq = inv1.equiperObjet(joueur, epee); //Utilisation d'une méthode pour équiper un objet

À toi de jouer! - Inventaire Sur Mesure

- Ajoute un constructeur à la classe Inventaire qui prend: un montant d'or de départ et une liste d'objets
- Utilise `this()` pour réutiliser le constructeur de base: Sois fainéant, mais malin!

À toi de jouer! - Visibilité VIP

- Passe en revue chaque attribut de la classe Inventaire
- Choisis le niveau de visibilité le plus approprié (private, public ou protected)
- Justifie tes choix: Pourquoi cet attribut est-il privé, public ou protected?

À toi de jouer! - Encapsulation Extrême

- Crée un setter pour l'attribut or: `public void setOr(int or) { ... }`
- Empêche les valeurs négatives: Pas de dettes!
- Empêche de dépasser la limite OR_MAX: La banque n'aime pas trop ça...

À toi de jouer! - Compteur de Butin

- Crée une variable static private int totalObjetsRamasses: Pour suivre le nombre d'items attrapés!
- Incrémente ce compteur à chaque fois qu'un objet est ajouté à l'inventaire
- Crée une méthode public static int getTotalObjetsRamasses() pour lire la valeur du compteur (sans créer d'objet!)

À toi de jouer! - Le Nombre d'Emplacements

- Crée une constante public static final int SLOT_MAX = 6
- Elle représente le nombre maximum d'emplacements dans l'équipement
- Dans une nouvelle classe Equipement, utilise cette constante pour déclarer un tableau d'Item: `Item[] slots = new Item[SLOT_MAX];`

Besoin d'aide? L'Oracle est là!

- Constructeurs Java (Oracle) : https://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html
- Modificateurs d’accès : https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html
- Encapsulation en Java : https://www.geeksforgeeks.org/encapsulation-in-java/
- Statics en Java : https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html