Université Badji Mokhtar - ANNABA
Faculté de Technologie
Département d'Informatique

Série de Travaux Pratiques

Programmation Orientée Objet

Mini-Projet : Gestion d'une Supérette

Réalisé par : **Dr. Hariati**
Niveau : **Deuxième année Licence en Informatique**

Année Universitaire 2024/2025

Table des matières

Séances 1 & 2 : Classes, Méthodes, Attributs, Constructeurs et Encapsulation

Objectif

Comprendre et appliquer les concepts de base de la POO : classes, attributs, méthodes, constructeurs, encapsulation, ainsi que les méthodes et attributs statiques.

Connaissances requises

Notions de base en Java (syntaxe, variables, boucles, conditions).

Entrées et sorties

- **Entrées** : Identifiant unique, nom, prix, quantité d'un produit et catégorie.
- **Sorties** : Affichage des informations des produits créés.

Travail à faire

1. Créer une classe `Product` avec :
   - **Attributs privés (encapsulation)** :
     - `String id` (identifiant unique du produit).
     - `String name` (nom du produit).
     - `double price` (prix du produit).
     - `int quantity` (quantité disponible).
     - `String category` (catégorie du produit, par exemple "alimentaire", "électronique").
   - **Attribut statique** :
     - `static int totalProducts` (compteur du nombre total de produits créés).
   - **Constructeurs** :
     - `Product(String id, String name, double price, int quantity, String category)` Constructeur principal permettant d'initialiser tous les attributs.
       Dans ce constructeur, utiliser le mot-clé `this` pour distinguer les attributs de la classe des paramètres du constructeur.
       À l'intérieur du constructeur, incrémenter l'attribut statique `totalProducts` à chaque fois qu'un nouveau produit est créé.
   - **Méthodes publiques** :
     - `void displayInfo()` : Affiche les détails du produit (`id`, `name`, `price`, `quantity`, `category`).
     - `boolean isAvailable()` : Retourne `true` si la quantité est > 0, sinon `false`.
   - **Méthode statique** :
     - `static int getTotalProducts()` : Retourne le nombre total de produits créés, en accédant à l'attribut statique `totalProducts`.
   - **Getters et Setters** :
     - `String getId()`, `void setId(String id)`, etc.
2. Ajouter des validations dans les setters pour garantir l'intégrité des données :
   - Refuser un nom vide.
   - Refuser un prix ou une quantité inférieure à 0.
3. Implémenter une classe `Main` pour tester les fonctionnalités :

— Créer quelques objets `Product`, afficher leurs informations avec `displayInfo()`.
— Tester la méthode `isAvailable()`.
— Afficher le nombre total de produits créés avec `getTotalProducts()`.

Séance 3 : Héritage

Objectif

— Apprendre à créer une hiérarchie de classes en utilisant l'héritage.

Connaissances requises

— Maîtrise des classes et des méthodes.

Entrées et sorties

— **Entrées :** Informations sur un produit alimentaire et un produit électronique.

— **Sorties :** Affichage des informations spécifiques selon le type de produit.

Travail à faire

1. Créer une classe `FoodProduct` qui hérite de `Product` :
   — Attributs :
      — `String expirationDate` (date d'expiration).
   — Méthodes :
      — Constructeur : Utiliser `super()` pour initialiser les attributs hérités de la classe `Product`.
      — Redéfinir `displayInfo()` pour inclure la date d'expiration.
      — Utiliser `super.displayInfo()` pour afficher les informations générales du produit avant d'ajouter l'expiration.
2. Créer une classe `ElectronicProduct` qui hérite de `Product` :
   — Attributs :
      — `int warrantyPeriod` (durée de garantie).
   — Méthodes :
      — Constructeur : Utiliser `super()` pour initialiser les attributs hérités de la classe `Product`.
      — Redéfinir `displayInfo()` pour inclure la durée de garantie.
      — Utiliser `super.displayInfo()` pour afficher les informations générales du produit avant d'ajouter la garantie.

Séance 4 : Polymorphisme

Objectif

Comprendre et implémenter les deux types de polymorphisme en Java :

1. Polymorphisme à la compilation (surcharge de méthodes).
2. Polymorphisme à l'exécution (redéfinition de méthodes via héritage).

Connaissances requises

— Classes, constructeurs, encapsulation.
— Héritage, surcharge et redéfinition des méthodes.

Détails des classes à manipuler

1. Classe de base : `Product`

Attributs :
— `String id` : Identifiant unique du produit.
— `String name` : Nom du produit.
— `double price` : Prix du produit.
— `int quantity` : Quantité disponible.
Méthodes :

1. **Constructeur** : Permet d'initialiser tous les attributs.
2. **Méthodes publiques :**
   — `void displayInfo()` : Affiche les informations générales du produit (`id`, `name`, `price`, `quantity`).
   — `boolean isAvailable()` : Retourne `true` si la quantité est supérieure à 0, sinon `false`.
3. **Méthodes surchargées pour démontrer le polymorphisme à la compilation :**
   — `void displayInfo(String currency)` : Affiche les informations avec le prix converti dans une devise donnée (exemple : Euro).
   — `void displayInfo(boolean detailed)` : Si `detailed` est `true`, affiche toutes les informations (`id`, `name`, `price`, `quantity`), sinon affiche uniquement le nom et le prix.

2. Sous-classe : `FoodProduct`

Hérite de : `Product`
Attributs supplémentaires :
— `String expirationDate` : Date d'expiration du produit (par exemple, "2025-03-15").
Méthodes :
— **Constructeur** : Initialise les attributs de `Product` et l'attribut spécifique `expirationDate`.
— **Méthode redéfinie :**
   — `void displayInfo()` : Affiche les informations du produit en incluant la date d'expiration (démonstration du polymorphisme à l'exécution).

3. Sous-classe : `ElectronicProduct`

Hérite de : `Product`
Attributs supplémentaires :
— `int warrantyPeriod` : Durée de garantie en mois (par exemple, 24 mois).
Méthodes :
— **Constructeur** : Initialise les attributs de `Product` et l'attribut spécifique `warrantyPeriod`.
— **Méthode redéfinie :**
   — `void displayInfo()` : Affiche les informations du produit en incluant la durée de garantie (démonstration du polymorphisme à l'exécution).

Travail à faire

1. Polymorphisme à la compilation (surcharge)

- Créer des objets de type `Product`.
- Tester les différentes versions de la méthode surchargée `displayInfo()` :
  - `displayInfo()` : Affiche les informations générales.
  - `displayInfo("Euro")` : Affiche le prix converti en euros.
  - `displayInfo(true)` : Affiche les détails supplémentaires si demandé.

2. Polymorphisme à l'exécution (redéfinition)

- Créer une liste de type `Product` contenant des instances de `FoodProduct` et `ElectronicProduct`.
- Parcourir la liste et appeler la méthode `displayInfo()` pour chaque objet.
- Observer que la version redéfinie de la méthode dans les sous-classes est appelée en fonction du type réel des objets.

Séance 5 : Classes Abstraites

Objectif

- Comprendre et utiliser les classes abstraites en Java pour structurer une hiérarchie de classes de manière efficace.
- Apprendre à définir et implémenter des méthodes abstraites afin d'assurer une spécialisation dans les sous-classes.

Connaissances requises

- Héritage et hiérarchie des classes.
- Polymorphisme et redéfinition des méthodes.

Entrées et Sorties

- **Entrées :** Données d'un produit spécifique (exemple : nom, prix, quantité, date d'expiration ou garantie).

- **Sorties :** Affichage des détails des produits, y compris leurs caractéristiques spécifiques.

Travail à faire

1. Transformer Product en classe abstraite

La classe `Product` représente un produit générique, mais elle n'a pas vocation à être instanciée directement. En la rendant abstraite, on impose aux sous-classes d'implémenter leurs propres comportements spécifiques.

2. Ajouter une méthode abstraite `void specificInfo();`

- La méthode `specificInfo()` est une méthode abstraite définie dans `Product`.
- **Utilité :** Elle oblige chaque type de produit à fournir des détails spécifiques qui ne sont pas communs à tous les produits.

— **Pourquoi ?** Un produit alimentaire a une date d'expiration, tandis qu'un produit électronique a une garantie. La méthode `specificInfo()` permet d'afficher ces informations sans impacter la structure générale.

3. Implémenter `specificInfo()` dans FoodProduct et ElectronicProduct

— Dans `FoodProduct`, `specificInfo()` affichera la date d'expiration du produit.
— Dans `ElectronicProduct`, `specificInfo()` affichera la durée de garantie.

Tester avec une liste de produits

1. Créer des objets de types `FoodProduct` et `ElectronicProduct`, mais les déclarer avec le type de référence `Produit`, puis les ajouter mélangés dans une même liste.
2. Parcourir la liste et appeler `specificInfo()` sur chaque élément sans se soucier de son type exact. Grâce au polymorphisme et au upcasting, la bonne version de `specificInfo()` sera appelée dynamiquement en fonction du type réel de l'objet.

Séance 6 : Interfaces

Objectif

Apprendre à utiliser et implémenter des interfaces.

Connaissances requises

— Héritage.
— Polymorphisme et redéfinition des méthodes.

Entrées et sorties

— **Entrées** : Informations sur un produit et un service (prix, description, etc.).
— **Sorties** : Affichage des prix avant et après remise du mois de Ramadan.

Travail à faire

Créer une interface Discountable

Définir une méthode `void displayDiscount()`, qui affiche le nom du produit ou du service avec son prix de base et son prix après application de la remise.

Implémenter Discountable dans Product

— Ajouter une méthode `displayDiscount()` qui affiche le prix du produit avant et après application d'une remise de 10%.
— Tester cette méthode sur des objets `FoodProduct` et `ElectronicProduct`.

Créer une nouvelle classe Service, qui implémente également l'interface Discountable

Attributs :
— `String description` (exemple : "Livraison à domicile").
— `double basePrice` (coût de base du service).

Méthodes :

— Implémenter `displayDiscount()` avec un taux de remise de 5% sur `basePrice`.
— Ajouter une méthode `void displayInfo()` qui affiche la description du service ainsi que son prix avant remise.

Tester avec des produits et des services

1. Créer des produits et des services en utilisant le type de référence **Discountable** (upcasting).
2. Créer une liste contenant un mélange de services et de produits déjà créés.
3. Parcourir la liste et afficher directement la remise pour chaque produit ou service en utilisant `displayDiscount()` (grâce au upcasting).
4. Afficher les autres informations des produits et services en utilisant `displayInfo()` ; un downcasting est nécessaire pour accéder à cette méthode.

Séance 7 : Gestion des exceptions

Objectif

— Apprendre à gérer les exceptions.

Connaissances requises

— Notions de base sur les exceptions.

Travail à faire

1. Ajouter une validation pour vérifier que le prix et la quantité sont positifs.
2. Lancer une exception personnalisée si ce n'est pas le cas.
3. Tester avec des cas valides et invalides.

Séance 8 : Interface graphique (Swing)

Objectif

— Concevoir une interface utilisateur simple pour gérer les produits.

Travail à faire

1. Créer une fenêtre Swing avec :
   — Champs pour entrer les informations du produit.
   — Boutons pour ajouter et afficher les produits.
2. Afficher les produits dans une zone de texte.

Séance 9 : Connexion à une base de données MySQL

Objectif

— Sauvegarder et récupérer des produits dans une base de données.

Travail à faire

1. Créer une base de données `superette` avec une table `products`.
2. Ajouter des méthodes pour :
   — Insérer un produit dans la base.
   — Récupérer et afficher tous les produits.