UBMA – Faculty of Technology – Computer Science Department **May 2025**

Course Instructor : Dr. M. Hariati

Model Solution for the Object-Oriented Programming Exam

Exercice 1 (6 points)

Code Java de la solution

```java
public abstract class Product {

    // (1) Private attributes (encapsulation)
    private String id;
    private String name;
    private double price;

    // (4) Static attribute to count the number of created products
    private static int totalProducts = 0;

    // (2) Constructor initializing all attributes and handling counting
    public Product(String id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
        totalProducts++;
    }

    // (3) Abstract method to be implemented by subclasses
    public abstract String specificInfo();

    // (4) Static method returning total number of created products
    public static int getTotalProducts() {
        return totalProducts;
    }
}
```

Barème de correction (sur 6 points)

Élément attendu	Points
(1) Déclaration des attributs `id`, `name`, `price` en `private` (encapsulation)	1.5 points
(2) Constructeur avec 3 paramètres initialisant correctement les attributs	1.5 points
(3) Déclaration correcte de la méthode abstraite `specificInfo()`	1 point
(4) Déclaration de l'attribut statique `totalProducts`	0.5 point
(4) Incrémentation du compteur dans le constructeur	0.5 point
(4) Méthode `getTotalProducts()` retournant le total des produits créés	1 point

Exercice 2 (8 points)

Code Java de la solution

```
1
2         // 1. Create three objects: FoodProduct, ElectronicProduct, Service
3         FoodProduct apple = new FoodProduct("F001", "Apple", 2.5, "2025-06-01");
4         ElectronicProduct laptop = new ElectronicProduct("E001", "Laptop",
              1200.0, 24);
5         Service delivery = new Service("S001", "Home Delivery", 15.0);
6
7         // 2. Create an ArrayList and add all items
8         ArrayList<Object> items = new ArrayList<>();
9         items.add(apple);
10        items.add(laptop);
11        items.add(delivery);
12
13        // Note: The type Object here can be replaced by Taxable,
14        // and then the casting in the first loop is no longer necessary.
15
16        // 3. Display taxes for all items (cast directly to Taxable)
17        System.out.println("=== Taxes ===");
18        for (Object item : items) {
19            ((Taxable) item).displayTax();
20        }
21
22        // 4. Display discounts for all items (cast directly to Discountable)
23        System.out.println("\n=== Discounts ===");
24        for (Object item : items) {
25            ((Discountable) item).displayDiscount();
26        }
27
28        // 5. Display specific info only for Product-type elements (still need
              instanceof)
29        System.out.println("\n=== Specific Information (Products only) ===");
30        for (Object item : items) {
31            if (item instanceof Product) {
32                ((Product) item).specificInfo();
33            }
34        }
```

Barème de correction (sur 8 points)

- Création correcte des trois objets (FoodProduct, ElectronicProduct, Service) : 1.5 points
- Création de la liste et ajout des objets : 1.5 points
- Affichage des taxes pour tous les éléments (avec cast correct) : 1.5 points
- Affichage des réductions pour tous les éléments (avec cast correct) : 1.5 points
- Affichage des informations spécifiques uniquement pour les objets de type Product : 2 points

Exercice 3 (6 points)

```java
// (1) HashSet collection to avoid duplicates
HashSet<Product> specialDelivery = new HashSet<>();

// (2) Method to add a product with messages
public void addProduct(Product p) {
    if (specialDelivery.add(p)) {
        System.out.println("Product added successfully: " + p);
    } else {
        System.out.println("Error: Product already exists!");
    }
}

// (3) Override equals() to compare all attributes
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || !(obj instanceof Product)) return false;
    Product other = (Product) obj;
    return id != null && id.equals(other.id)
        && name != null && name.equals(other.name)
        && price == other.price;
}

// (4) hashCode based only on the unique identifier
@Override
public int hashCode() {
    return uniqueId;
}

// To ensure that the hashCode() method works correctly, these changes must be made in the Product class

private int uniqueId; // We add this line in the declaration of the Product class

public Product(String id, String name, double price) {
    this.id = id;
    this.name = name;
    this.price = price;
    totalProducts++;
    this.uniqueId = totalProducts; // We add this line in the constructor of the Product class
}

// (5) toString for readable display
@Override
public String toString() {
    return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";
}
```

Barème (8 points) :

- (1) Déclaration de la collection `HashSet` : 0.5 point
- (2) Méthode `addProduct()` avec messages : 1.5 points
- (3) Redéfinition de `equals()` (tous les attributs) : 2 points
- (4) Redéfinition de `hashCode()` avec identifiant unique, ceci un inclut le code de la méthode `hashCode()`, la declaration de `uniqueId` ainsi que son affectation dans le constructeur : 1 point
- (5) `toString()` pour affichage lisible : 1 point