**UE Conception Orientée Objet**

# Devoir Surveillé

3h

Copie annotée des diapositives de cours et fiches « design pattern » diffusées autorisées.
Dictionnaires de langue autorisés.
Autres documents interdits.

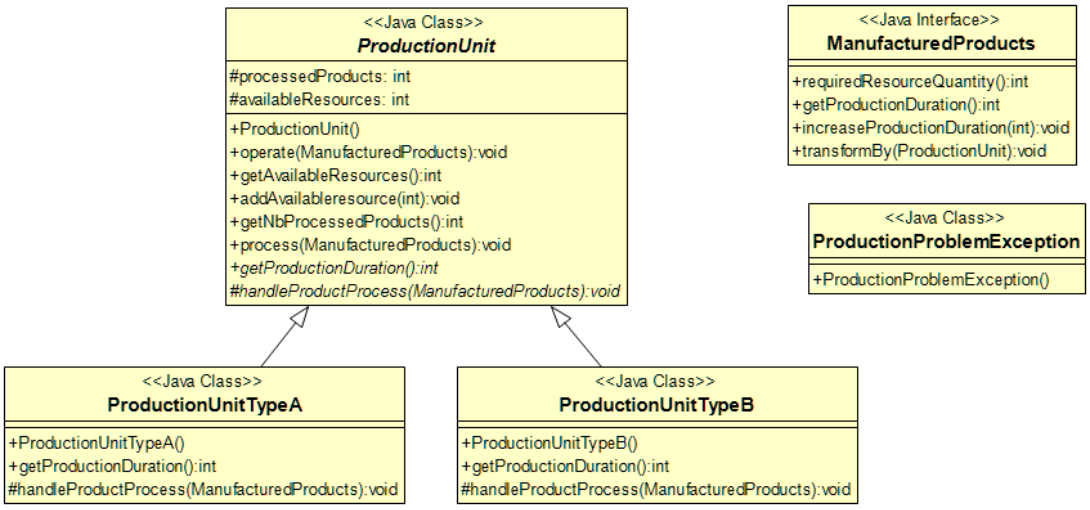
Répondre aux exercices 1 et 2 sur **copie jaune** et à l'exercice 3 sur **copie verte**.

*Ne donnez la javadoc ou les tests des classes à écrire que lorsqu'ils sont explicitement demandés.*

*Les exercices sont indépendants. L'exercice 3 est plus long à traiter que les deux autres.*

## Exercice 1 : Tests (répondre sur copie **jaune**)

Le contexte général de l'exercice est celui de produits qui sont manufacturés (fabriqués) (*manufactured products*) par actions successives de plusieurs unités de production (*production unit*). Pour ce faire, les unités de production utilisent des ressources (*resource*).
On ne considère qu'une version très simplifiée d'un tel problème, suffisante pour les besoins de l'exercice (par exemple on ne s'intéresse pas aux types des ressources ni à comment le produit est transformé par l'unité de production).
Ainsi la modélisation proposée se limite aux classes qui apparaissent dans le diagramme suivant :




Le code de `ManufacturedProduct` est le suivant :

```
package exo1.main;

public interface ManufacturedProduct {
  /** @return the amount of resource to be used by a production unit to process this product
   *  (As a simplification it is assumed to be independent from the production unit.)
   */
  public int requiredResourceQuantity();
  /** @return the total time spent (for now) by all the productions unit that processed this product
   */
  public int getProductionDuration();
  /** increase the production duration for this product
   * @param duration the duration to add to the production duration
   */
  public void increaseProductionDuration(int duration);
  /** handle how the unit operates on the product
   * @param unit the production unit that works on this product
   */
  public void transformBy(ProductionUnit unit);
}
```

Pour `ProductionUnit`, assez naturellement, on a :

- `getAvailableResources()` fournit le nombre de resources disponibles pour cette unité (afin qu'elle puisse traiter les produits) ;
- `addAvailableResources()` permet d'ajouter des nouvelles ressources disponibles pour cette unité ;
- `getNbProcessedProducts()` fournit le nombre de produits qui ont pu être traités (via sa méthode `process`) par cette unité ;
- `getProductionDuration()` fournit le temps mis par cette unité pour traiter un produit.

On s'intéresse plus particulièrement aux méthodes `operate()` et `process()` de `ProductionUnit`. Le cahier des charges stipule :

1. L'exécution de la méthode `process(p)` lève une exception `ProductionProblemException` si les resources disponibles pour l'unité de production sont insuffisantes pour le produit `p`.
2. Si les ressources sont en quantité suffisante, à l'issue de l'exécution de la méthode `process(p)`
   - le nombre de ressources disponibles pour l'unité de production est diminué du montant des ressources requises par le produit `p`.
   - la durée de production du produit `p` est augmentée du temps de production de l'unité.
3. Chaque appel à la méthode `operate()` déclenche l'exécution de la méthode `process(p)`.
4. A l'issue de l'exécution de la méthode `operate(p)`, si la méthode `process(p)` n'a pas déclenché d'exception, le nombre de produits traités par l'unité de production est augmenté de 1.

**Q 1 .** Illustrez (et expliquez) par l'exemple les 3 étapes (sans le code) d'un cycle TDD qui s'appliqueraient pour le développement de la méthode `process`.

**Q 2 .** Donnez le code des **classes de tests** (sauf leur entête et ses `import`) et leurs méthodes permettant de vérifier que les implémentations des méthodes `process()` et `operate()` vérifient bien les différents points de cahier des charges mentionnés pour chacune des classes `ProductionUnitTypeA` et `ProductionUnitTypeB`. Le cas échéant, la mise en place des tests de ces méthodes pour de nouvelles sous-classes de `ProductionUnit` doit être facile et demander peu de nouveau code.

**Q 3 .** Pourquoi couvrir 100% du code de la classe `ProductionUnit` par les tests ne garantit pas pour autant l'absence de bugs ?

**Exercice 2 : Le Donjon revisité** (répondre sur copie **jaune**)
Cet exercice revient sur le projet Donjon réalisé au cours du semestre. En particulier, nous nous intéressons à la classe `Fight` suivante qui a été développée par l'un d'entre vous :

```
package exo3;

import java.util.List;

public class Fight implements Action {

  /**
   * @see Action.execute
   */
  public void execute(Player player) {
    List<Monsters> monsters = player.getRoom().getMonsters();
    int i;
    for(i = 0; i < monsters.size(); i++)
      if (monsters.get(i).isAlive())
        System.out.println("Monstre "+(i)+" :"+monsters.get(i).toString());
    int index = ScannerInt.readInt(i+1);
    if ( (index >= 0) && (index < i) ) {
      Monsters monstre = monsters.get(index);
      if (monstre.isAlive()) {
        while (player.isAlive() && monstre.isAlive()) {
          player.attaque(monstre);
          if (monstre.isAlive())
            monstre.attaque(player);
        }
        if (player.isAlive()) {
          player.addGold(monstre.getGold());
          System.out.println("Bravo, vous gagnez "+ monstre.getGold()+" pièces d'or");
        }
      } else
        System.out.println("Ce monstre dégouline de sang à terre !");
    }
  }

  public boolean isPossible(Player player) {
    return (!(player.getRoom().allMonstersAreDied()));
  }

  public String toString() {
    return ("Fight : combat un des monstres de la pièce");
  }


}
```

**Q 1 .** Citez au moins un anti-patron dont souffre la classe `Fight`.

**Q 2 .** Quelle(s) proposition(s) feriez-vous au développeur de cette classe pour corriger l'un de ces anti-patrons ? Après avoir décrit les modifications à apporter, proposez une nouvelle version de la classe qui corrige cet anti-patron.

**Q 3 .** Quelle est la complexité de McCabe pour la méthode `execute` (telle qu'elle est codée ci-dessus dans le sujet) ? Expliquez comment vous obtenez ce résultat.

**Exercice 3 : Les voyages forment la jeunesse** (répondre sur copie **verte**)
Des agences de voyage proposent dans leur catalogue diverses activités touristiques[1]. Il est possible d'obtenir la description textuelle d'une activité touristique par la méthode `description()` et son coût par la méthode `cost()`. On distingue plusieurs catégories d'activités touristiques :

- des activités ponctuelles (*simple tourism activity*), telles que la *visite d'un musée* (*museum visit*), la *visite d'un site historique* (*historical site visit*) ou une *descente en rafting* (*rafting ride*), etc.

  Ces activités n'ont pas de caractéristiques particulières.

- des *journées touristiques* (*tourism day*) qui correspondent à des activités qui durent une journée et qui ont lieu à une date précise, précisée à la construction (de type `java.util.Date`).

  Il existe plusieurs sortes de journées touristiques, par exemple :

  - des *journées au programme bien rempli* (*busy day*) avec une activité ponctuelle le matin et une autre l'après-midi. Ces deux activités sont communiquées à la construction.

    Le coût de la journée est la somme du coût des deux activités et sa description est obtenue à partir de leurs descriptions.
  - des *excursions touristiques* (*excursion day*) qui consiste en la *visite d'un site touristique*.

    L'activité correspondant à la visite et l'heure de départ pour cette visite (de type `java.util.Date`) sont communiquées à la construction.

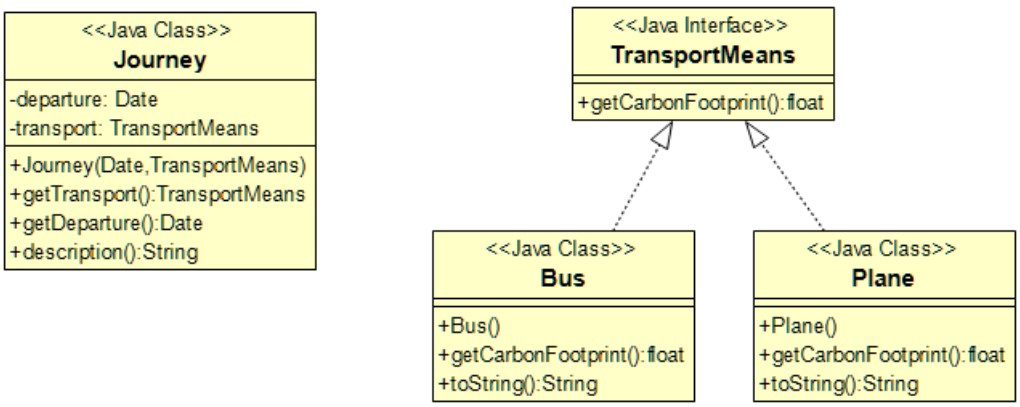
    Son coût est celui de l'activité augmentée de 15 euros de forfait (pour le trajet). La description correspond à la description de la visite et à l'annonce de l'heure de départ.
  - des journées de temps libre (*leisure day*) dans une ville. La ville est passée à la construction (`String`) et citée dans la description. Le coût d'une telle activité est 0.
  - etc.

- des *voyages organisés* (*tour package*) qui regroupent un trajet aller, plusieurs journées touristiques et un trajet retour.

  On construit un voyage à partir des données sur les trajets aller et retour (les trajets sont présentés ci-dessous). Les journées qui composent ce voyage sont ajoutées ensuite[2].

  La description d'un voyage est obtenue en agrégeant les descriptions de ses trajets et de chacune des journées qui le composent et son coût par l'addition des coûts de ses journées (pas de coût lié aux trajets).

**Trajets.** Certaines activités touristiques (comme les voyages organisés par exemple) nécessitent des trajets (*journey*). Un trajet est caractérisé par l'information sur le moment du départ (de type `Date`) et le moyen de transport (*transport means*) utilisé, le bus ou l'avion par exemple. Ces informations sont communiquées à la construction du trajet. Le diagramme suivant présente la modélisation de ces données :




[1] De nombreuses simplifications seront faites dans cet exercice. On s'intéresse par exemple assez peu aux lieux où se déroulent les activités et on ne s'intéressera pas à la vérification des contraintes de cohérence des dates des différentes activités ou trajets.

[2] C'est par exemple ici que l'on ne fait aucune vérification de cohérence.

**Q 1 .** Sous la forme de diagrammes UML détaillés faites une proposition de conception pour représenter les activités touristiques (On doit y retrouver les éléments *en italique* du texte précédent).
Dans votre diagramme, vous ferez apparaître clairement les attributs, constructeurs et méthodes (et leurs signatures), ainsi que les surcharges.
**Faites un diagramme clair et lisible. N'hésitez pas à prendre votre copie au format paysage, ou mieux à faire votre diagramme sur les deux pages intérieures de votre copie.**

**Q 2 .** Donnez le code java des classes, et de leur(s) super-type(s), correspondant aux :

- *visite d'un site historique* (*historical site visit*) ;
- *excursions touristiques* (*excursion day*) ;
- *voyages organisés* (*tour package*).

**Q 3 .** Des options ou variantes sont possibles pour les journées touristiques.
Ainsi pour chacune des journées existantes il est possible d'y ajouter l'option *repas inclus* (*meals included*). Cela ne modifie pas le prix de la journée, mais la mention `"Meals included."` est ajoutée à la description de la journée.
Il est également possible d'ajouter une *activité nocturne* (*night tourism activity*) à une journée existante. Les activités nocturnes sont une catégorie particulière d'activités ponctuelles. L'activité nocturne ajoutée est fournie à la construction et sa description vient compléter la description de la journée et son coût s'ajoute à celui de la journée.

**Q 3.1.** Donnez un diagramme UML détaillé proposant une solution pour prendre en compte ces deux options en faisant le lien avec les types de *journées touristiques* (inutile de reprendre les autres types dans le diagramme !).

**Q 3.2.** Donnez le code java correspondant à la gestion de la variante « repas inclus », ainsi que de ses super types s'ils n'ont pas déjà été donnés dans les question précédentes.

**Q 4 .** Une *agence de voyage* (*travel agency*) possède un catalogue d'activités de tourisme.
Le choix du moyen de transport utilisé pour les voyages organisés sont à la charge de l'agence de voyage, selon ses propres critères.
On trouve différents types d'agence de voyages.

- les agences low cost (*low cost agency*) ne proposent que des trajets en bus ;
- les agences qui offrent des voyages transatlantiques (*transatlantic agency*) ne proposent que des trajets en avion.

Une agence de voyage offre la possibilité de construire un *voyage organisé* grâce à cette méthode :

```
/** Build a TourPackage from the departure and return dates for the journeys and the days that composed
 *    the tour.
 * The transport means of the tour journeys depends on this agency.
 *
 * @param forwardDate the departure date
 * @param backDate the return date
 * @param days the list of TourismDay in this package
 * @return the built tour package
 */
public TourPackage buildTourPackage(Date forwardDate, Date backDate, List<TourismDay> days) {
```

L'impression du catalogue correspond à l'impression de la description suivie du coût de chacune de ces activités. On se contentera ici d'une impression sur la sortie standard du catalogue.

**Q 4.1.** Donnez un diagramme UML détaillé modélisant les différentes agences citées ci-dessus.

**Q 4.2.** Donnez le code java pour la classe représentant les agences low cost et son (ou ses) super-type(s).