UNIVERSITÉ DE ROUEN
FACULTÉ DES SCIENCES

Master GIL, 2e année

Examen de POA
mardi 5 janvier 2016
Durée : 2h

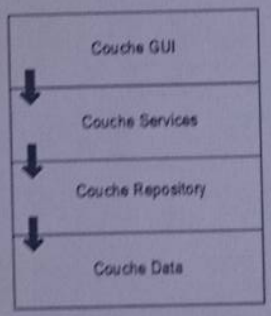
Calculatrices, portables et documents sont interdits à l'exception de l'aide-mémoire « AspectJ 5 Quick Reference ».
Il sera tenu compte du soin apporté à la présentation.

Exercice I

1. A quoi servent les deux méthodes dont les signatures sont fournies ci-après ?
   `T Class<T>.newInstance()`
   `T Constructor<T>.newInstance(Object...)`
   Quelles sont les deux principales différences existant entre ces deux méthodes ?
2. Donner un maximum d'informations concernant la classe `java.util.logging.Level` (les constantes et commodités qui y sont définies, à quoi servent les instances de cette classe et comment on les utilise).

Exercice II

Une bonne pratique de programmation, lorsqu'on développe une application dont l'architecture est structurée en couches, consiste à ne réaliser des appels depuis une couche qu'en direction de la couche suivante. Un exemple classique est illustré par la figure ci-dessous :



Ainsi, une méthode de la couche Services ne doit être appelée que depuis la couche Services ou la couche GUI, une méthode de la couche Repository ne doit être appelée que depuis la couche Repository ou la couche Services,...

Pour imposer cette bonne pratique lors d'un projet de développement Web, un chef de projet décide d'utiliser la POA. Il crée trois paquetages : web, services et repository qui devront être utilisés pour y stocker les types en rapport. Il observe que la gestion des données (la couche Data) se fait exclusivement à l'aide des outils fournis dans les paquetages java.sql et javax.persistence. Puis il vous demande de développer "le plus proprement possible" un

aspect qui garantisse une erreur à la compilation dès que la pratique recommandée n'est pas respectée.

Par exemple, une erreur surviendra à la compilation au niveau de la ligne 9, quand un développeur rentrera le code suivant :

```
1  package web;
2
3  import java.sql.DriverManager;
4  ... autres imports
5
6  public class ListInventoryController implements Controller {
7      public ModelAndView handleRequest(HttpServletRequest request,
8              HttpServletResponse response) throws Exception {
9          Connection conn = DriverManager.getConnection(
10                 "jdbc:hsqldb:file:ecommerce", "sa", "");
11         ... use connection ...
12     }
13 }
```

Vous devez fournir l'outil attendu par le chef de projet en précisant les aspects que vous développez. Par "proprement" on entend un système facile à maintenir (identificateurs explicites, choix de coupes pertinents,...) et à étendre (si une nouvelle couche apparait dans l'architecture, si un nouvel outil d'accès aux données est utilisé,...).

Exercice III

On considère l'extrait de code suivant :

```
1  public class BusinessType {
2      private Collection<Item> warehouse;
3      private Stock<Item> stock;
4
5      public void businessMethod(Item item) {
6          TransactionAttributeWithRollbackRules attributes =
7                  new TransactionAttributeWithRollbackRules();
8          List<Class<? extends Throwable>> list =
9                  new LinkedList<Class<? extends Throwable>>();
10         list.add(IOException.class);
11         list.add(BusinessException.class);
12         attributes.addRollbackFor(list);
13         PlatformTransactionManager tm = new JpaTransactionManager();
14         TransactionStatus ts = tm.getTransaction(attributes);
15         try {
16             warehouse.add(item);
17             stock.extract(item);
18             tm.commit(ts);
19         } catch(Throwable ex) {
20             if (attributes.rollbackOn(ex)) {
21                 tm.rollback(ts);
22             } else {
23                 tm.commit(ts);
24             }
25         }
26     }
27 }
```

La méthode `businessMethod` doit être exécutée de manière transactionnelle, c'est-à-dire totalement (la transaction se termine par un "commit") ou pas du tout (la transaction est annulée grâce à un "rollback"). En effet, cette méthode extrait un item d'un stock pour le placer dans une collection. Si la méthode est exécutée partiellement, le système se retrouve dans un état incohérent, un item étant présent à deux endroits à la fois.

La mise en place de la transaction suppose de définir des attributs. Il en existe plusieurs, nous nous limiterons à la définition de la liste des exceptions pour lesquelles la transaction sera annulée. On fait ensuite appel à un gestionnaire de transactions auquel on fournit les attributs que nous avons définis. Il nous renvoie la transaction nécessaire pour ensuite réaliser le commit ou le rollback souhaité.

La gestion de la transaction est évidemment une préoccupation transversale. On souhaite donc la définir dans un unique endroit : un aspect. Du coup le code métier est épuré. On aura juste à préciser, par le biais d'une annotation, que l'opération est transactionnelle. On obtient donc :

```
1  public class BusinessType {
2      private Collection<Item> warehouse;
3      private Stock<Item> stock;
4
5      @Transactional(rollbackFor = {IOException.class, BusinessException.class})
6      public void businessMethod(Item item) {
7          warehouse.add(item);
8          stock.extract(item);
9      }
10 }
```

1. On demande d'abord de coder le type d'annotation `Transactional`. On précise que ce type devra permettre d'annoter une méthode ou un type (classe ou interface). Dans ce dernier cas, toutes les méthodes du type seront transactionnelles.

   Par ailleurs, en plus du paramètre `rollbackFor`, il lui est associé un paramètre booléen `readOnly`, positionné à faux, par défaut.

2. Donner ensuite le code de l'aspect permettant effectivement de rendre transactionnelles les méthodes concernées par l'annotation.

   On précise qu'il va falloir récupérer les paramètres de l'annotation par introspection et que si un type et une méthode de ce type sont tous les deux annotés, les paramètres de l'annotation de la méthode seront prioritaires pour celle-ci.