Université Abou Beckr Belkaid
Master 2 Génie Logiciel
Année Universitaire 2016/2017

Correction d'Examen Final

—

Architecture et Développement Logiciels

—

Partie A

1 Donner la définition de la programmation par aspect ? Donnez un de ses avantages.
2 Quelle est le rôle d'AspectJ ?
3 Voici l'example d'un programme ecrit en AspectJ :

```
public aspect Helloaspects {

   pointcut toPerson():
      call (* Helloer.*(Person));

   pointcut toPlace():
       call (* Helloer.*(Place));

   before(): toPerson(){
       System.out.println("Appel individuel");}

   before(): toPlace(){
       System.out.println("Appel aux personnes dans un lieu");}
```

(1) (2) (3) (4)

- Expliquer en détails les points de coupures 1 et 2.
- Que font les advices 3 et 4.

Solution :

1 La programmation par aspect est un paradigme de programmation qui s'intègre dans les langages de programmation existantes pour les complétés. Un de ses avantages : elle permet la séparation entre la partie logique métier et les besoins transversaux (sécurité, authentification..)
2 AspectJ est le plugin à ajouter au langage Java afin d'intégrer la programmation orientée aspect.
3 :

- Le point de coupure 1 : représente tous les appels des méthodes de la classe **Helloer** qui prend un paramètre de type **Person** et rend n'importe quelle type. Ce point de coupure s'appelle **ToPerson()**.
- Le point de coupure 2 : il s'appelle **toPlace()**. Il représente tous les appels des méthodes de la classe **Helloer** qui prend un paramètre de type **Place** et rend n'importe quelle type.
- L'advice 3 : permet d'afficher le message "Appel Individuel" avant le point de coupure **ToPerson()**.
- L'advice 4 : permet d'afficher le message "Appel aux personnes dans un lieu" avant le point de coupure **ToPlace()**.

Partie B

Exercice 1 :

Voici l'exemple d'un programme qui contient trois classes :

```
public class Courrier {
  private Long id;
  private String destinataire;
  private String adresse;
}

public class Suivi extends Courrier {
  private boolean arrivé
}

public class Recommande extends Suivi {
  private Date envoi;
  private Date recu;
  private boolean signature;
}
```

Questions :

1 Compléter les classes données pour les transformer en classe Entité Bean.
2 Après transformation en classes entités beans, quelles sont les tables créées dans la base de données relationnelle selon la stratégie **Join Strategy**.
3 Ecriver le code de la classe qui permet de manipuler les instances de la classe Courrier (trouver, ajouter, supprimer).
4 Donner les différentes étapes nécessaires pour que cette application soit opérationnelle sur un serveur d'application ?

Solution :

1 Pour transformer les classes en Entité Bean, il faut :
  – ajouter l'annotation **@entity**.
  – implémenter l'interface **Sérializable**.
  – ajouter un constructeur sans arguments.
  – ajouter des getters et setters.
2 Dans la base de données une table est crée par classe avec des relations de jointure entre les classes. la table qui correspond à la classe mère contient les champs associés a ses attribues et les tables des sous classes contiennent que leurs propres champs, et ils accèdent aux champs de la classe mère via des clés étrangères.
3 La classe qui permet de manipuler la classe courrier doit déclarer un entité manager et utiliser les méthodes associées.
4 Pour que cette application soit opérationnelle sur un serveur d'application, il faut :
  – Associer un serveur, exemple Jboss (ou équivalent), au projet et cela à sa création et lui spécifier le chemin vers ce serveur.
  – Déployer le projet fini sur le serveur.

Exercice 2 :

On veut créer un **Aeroport** ainsi que des objets de type **Avion**. Voici le code des classes **Aeroport** et **Avion**, ainsi que de la classe **test** de l'ensemble.

```
public class Aeroport
{
      public Aeroport()
      {
            piste_libre=true;
      }
}

class Avion extends Thread
{
      String nom;
      Aeroport a;

      public Avion(String s)
      {
            nom=s;
      }

      public void run()
      {
            a=new Aeroport();
            System.out.println("Je suis avion "+nom+" sur aeroport "+a);
      }
}
```

```
class testaeroport
{
      public static void main(String[] args)
      {
            Avion v1 = new Avion("Avion 1");
            Avion v2 = new Avion("Avion 2");
            Avion v3 = new Avion("Avion 3");
            Avion v4 = new Avion("Avion 4");

            v1.start();
            v2.start();
            v3.start();
            v4.start();  } }
```

Questions :

1 Que fait la méthode **start** lorsqu'elle est appelée sur les avions ?
2 On souhaite que les clients (les objets de type **Avion**) ne puisse pas créer plus d'un **Aeroport**, afin qu'ils se situent tous dans un même **Aeroport**. On doit empêcher la possibilité qu'à un **Avion** puisse créé un **Aeroport** s'il en existe déjà un. Pour cela, il faut utiliser le pattern **Singleton**.
   – donner la définition du pattern **Singleton**.
   – Donner la nouvelle conception (diagramme UML) en utilisant le pattern **singleton**.
   – quelles sont les changements à faire dans le code des classes déjà données ?

Solution :

1 La méthode **start** lance la méthode *run()* de la classe **Avion**, elle crée un aéroport et affiche le message correspondant.
2 Le pattern Singleton est un pattern très utilisée il est de type création, il permet d'avoir une seul instance d'une classe. Pour cela, il faut ajouter un contrôle sur le nombre d'instances que peut retourner une classe.
3 Dans la nouvelle conception il suffit de définir la classe **Aéroport** comme étant un Singleton en remplaçant la construteur public par private, ajouter un attribut statique a cette classe pour stoker l'instance unique de la classe, et il faut ajouter la méthode statique de contrôle de création de l'instance unique.
4 Pour utiliser le pattern Singleton dans la classe **Aéroport**, il faut :
   – déclarer le constructeur de la classe en **private**.
   – Définir un attribut statique dans la classe **Aéroport** qui va contenir l'instance unique. Elle est null avant la première création.
   – Ajouter à la classe **Aéroport** une méthode statique qui permet de contrôler la création d'un objet **Aéroport**, si l'instance existe elle la retourne sinon elle appelle le constructeur pour la créée.
   – dans la méthode *run()* de la classe **Avion** on remplace l'appel du constructeur par l'appel de la méthode statique qu'on vient de définir dans la classe **Aéroport**.

Bon courage et bonne continuation.