

Examen Final

Architecture et Développement Logiciels

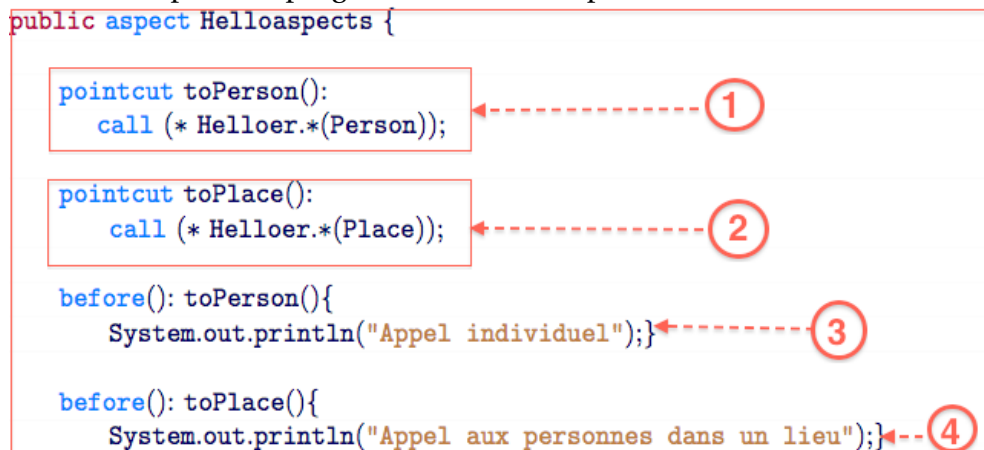
Remarques :

- Les documents ne sont pas autorisés ainsi que les appareils électroniques (PC, Tablette, téléphone, etc).
-

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'exemple d'un programme écrit 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");  
    }  
}
```



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

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 reçu;
    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 Ecrire 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?

Exercice 2 :

On veut créer un **Aéroport** ainsi que des objets de type **Avion**. Voici le code des classes **Aéroport** 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 créer 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 ?

Bon courage et bonne continuation.