# AP4B - Examen final du 24 Juin 2024

## Durée 2 heures – aucun document autorisé

**Exercice 1 (2pts).** Considérons le schéma de conception (*design pattern*) Singleton.

1) Quel est l’intérêt de ce modèle dans une application ?

2) Compléter la définition de classe ci-dessous pour que le modèle fonctionne correctement, même lorsqu’il est utilisé via des threads concurrents. Compléter à l’endroit des pointillés (…).

**class Singleton {**

 **... Singleton unique = null;**

 **... Singleton() {} // supprime le constr. public**

 **... Singleton getInstance() {**

 **if (unique == null)**

 **unique = new Singleton();**

 **return unique;**

 **}**

**}**

3) Donner un exemple d’utilisation et d’appel dans un programme.

**Exercice 2 (3pts).** Considérons maintenant le schéma de conception Adapter définit suivant le modèle de classe suivant :



1) Quel est l’intérêt de ce modèle dans une application ?

2) Donner un exemple d’application de ce pattern en écrivant un code Java pour transformer une classe Cercle en une classe Point. Préciser bien les 3 classes écrites en Java avec méthodes, constructeurs et attributs. Pour l’interface Point on prendra les 2 méthodes getX() et getY(). Pour une classe Cercle, on prendra getX() et getY() et getR() avec ses attributs.

**Exercice 3 (3pts).** **Gestion de carrière professionnelle.**

Un logiciel de gestion de carrière doit mémoriser des informations pertinentes sur une carrière professionnelle. Modéliser par un diagramme de classe les deux situations ci-dessous, sachant que le modèle doit être adapté au besoin.

a) Une personne peut travailler pour plusieurs entreprises. Pour une entreprise, elle exerce une unique fonction (ou métier) à laquelle sont rattachées une rémunération horaire et une durée totale de travail.

b) Une personne peut travailler pour plusieurs entreprises et exercer en même temps plusieurs fonctions dans l’entreprise auxquelles sont rattachées une rémunération horaire et une durée totale de travail.

**Exercice 4 (8pts).** On désire réaliser un Système d’Information (SI) permettant de suivre l’activité d’une compagnie aérienne et obtenir toutes sortes d’informations utiles. Dans ce problème, on s’intéresse principalement à la phase d’analyse préalable qui débouche sur l’élaboration d’un Modèle de classes UML.

Cette compagnie dispose d’avions sur lesquels volent un commandant et des membres d’équipage. Un commandant est caractérisé par son nom, son nombre d’heures total de vol depuis le début de l’année ; il vole toujours sur le même type d’avion (exemple : toujours AIRBUS). L’équipage de l’avion est constitué de différents membres avec différentes fonctions (hôtesse, steward, ...) donc différents salaires. On prend comme hypothèse qu’un membre d’équipage est dirigé toujours par le même commandant donc vole également toujours sur le même type d’avion. Le nombre d’heures de vol effectuées par un membre d’équipage fait l’objet de primes aussi désire-t-on conserver pour chaque mois les heures qu’il a effectuées. On désire également connaître le nombre d’heures total depuis le début de l’année. Enfin on voudrait conserver l’âge des commandants et des membres d’équipage afin de décider des contrôles médicaux et des mises à la retraite ; on conserve également leur numéro de sécurité sociale.

Un avion est caractérisé par un numéro d’avion (code interne à la compagnie) et appartient à un certain type d’avion (par exemple : AIRBUS, BOEING, CARAVELLE, DC 10, ...), suivant son type il a un certain nombre de places disponibles en première et deuxième classe ; on connaît ses dates de mise en service et de dernière révision ; de plus on conserve son nombre d’heures de vol total et son nombre d’heures de vol depuis la dernière révision afin de décider d’une révision ou de sa mise à la retraite.

Les avions effectuent des liaisons entre des villes de départ et des villes d’arrivée en effectuant éventuellement des escales (exemple : PARIS, NEW-YORK avec escale à LONDRES) ; une escale est caractérisée par un numéro, le nom de la ville ainsi que le pays où elle se trouve ; pour chaque vol (ex : AIR FRANCE 503) on connaît la liaison à laquelle il correspond, son heure de départ, son heure d’arrivée, la durée du vol, le jour du vol dans la semaine ; évidemment un vol a une ou plusieurs escales, chaque ville escale peut recevoir différents vols. Plusieurs vols peuvent réaliser une même liaison (ex : AIR FRANCE 503 et BA202 font PARIS-NEW YORK), par contre un vol de numéro donné correspond à une seule liaison.

1. Donner un diagramme des cas d’utilisation du SI.
2. Donner un diagramme de séquence pour un scénario de réservation de vol départ/arrivée.
3. Donner un modèle de classe UML détaillé du SI à réaliser.

**Exercice 5 (4pts). Trieur dichotomique en Java.**

Voici un algorithme de tri en ordre croissant d'une tranche de tableau comprise entre les éléments d'indices debut et fin :

trier(debut, fin) {
  si (fin - debut < 2) { // on a fini, pas d'appel récursif
    si (t[debut] > t[fin])
      echanger(t[i], t[j])
  }
  sinon {
    milieu = (i + j) / 2
    trier(debut, milieu)
    trier(milieu + 1, fin)
    triFusion(milieu) // tri fusion des 2 moitiés de la tranche du tableau
  }
}

On remarque que les 2 tris qui sont effectués avant la fusion sont indépendants l'un de l'autre et il est donc facile de les faire exécuter en parallèle par 2 threads. La structure du programme étant donnée dans les deux pages suivantes, le but de l'exercice est de la compléter aux endroits indiqués par des pointillés "**..."** .

/\*\* Tri d'un tableau d'entiers multi-thread. \*/

**public** **class** Trieur **extends** **...** {

 **private** **int**[] t; // tableau à trier

 **private** **int** debut, fin; // tranche de ce tableau qu'il faut trier

 **private** Trieur parent; // tread Trieur qui a lancé ce (this) Trieur

 **private** **int** nbNotify = 0; // Nombre de notifys envoyés à ce (this) Trieur

 **public** Trieur(**int**[] t) {

 **this**(**null**, **...**, **...**, **...**);

 }

 **private** Trieur(Trieur parent, **int**[] t, **int** debut, **int** fin) {

 **this**.parent = parent;

 **this**.t = t;

 **this**.debut = debut;

 **this**.fin = fin;

 start();

 }

 **public** **...** **void** notifier() {

 **this**. **...**

 **this**. **...**

 }

 **public** **void** run() {

 **if** (fin - debut < 2) {

 **...**

 }

 **else** {

 **int** milieu = debut + (fin - debut) / 2;

 Trieur trieur1 = **...**

 Trieur trieur2 = **...**

 // attend les 2 threads

 **synchronized**(**this**) {

 **try** {

 // Tant que 2 notifications n'ont pas été reçues (1 par

 // trieur "fils"), on attend.

 **while** (**...**) {

 **...**

 }

 }

 **catch**(InterruptedException e) {}

 }

 triFusion(**...**);

 }

 **if** (parent != **...**) {

 parent. **...**

 }

 }

 /\*\* Echanger t[i] et t[j] \*/

 **private** **void** echanger(**int** i, **int** j) {

 **int** valeur = t[i];

 t[i] = t[j];

 t[j] = valeur;

 }

 /\*\*

 \* Fusionne 2 tranches déjà triées du tableau t.

 \* - 1ère tranche : de debut à milieu = (debut + fin) / 2

 \* - 2ème tranche : de milieu + 1 à fin

 \*/

 **private** **void** triFusion(**int** debut, **int** fin) {

 // tableau où va aller la fusion

 **int**[] tFusion = **new** **int**[fin - debut + 1];

 **int** milieu = (debut + fin) / 2;

 // Indices des éléments à comparer

 **int** i1 = debut,

 i2 = milieu + 1;

 // indice de la prochaine case du tableau tFusion à remplir

 **int** iFusion = 0;

 **while** (i1 <= milieu && i2 <= fin) {

 **if** (t[i1] < t[i2]) {

 tFusion[iFusion++] = t[i1++];

 }

 **else** {

 tFusion[iFusion++] = t[i2++];

 }

 }

 **if** (i1 > milieu) {

 // la 1ère tranche est épuisée

 **for** (**int** i = i2; i <= fin; ) {

 tFusion[iFusion++] = t[i++];

 }

 }

 **else** {

 // la 2ème tranche est épuisée

 **for** (**int** i = i1; i <= milieu; ) {

 tFusion[iFusion++] = t[i++];

 }

 }

 // Copie tFusion dans t

 **for** (**int** i = 0, j = debut; i <= fin - debut; ) {

 t[j++] = tFusion[i++];

 }

 }

 **public** **static** **void** main(String[] args) {

 **int**[] t = {5, 8, 3, 2, 7, 10, 1, 6, 9, 4};

 Trieur trieur = **new** Trieur(t);

 **try** {

 trieur.join();

 }

 **catch**(InterruptedException e) {}

 **for** (**int** i = 0; i <t.length; i++) {

 System.***out***.print(t[i] + " ; ");

 }

 System.***out***.println();

 }

}