

Questions de cours (10 points)

Vector et ArrayList (1 points)

Pourquoi l'utilisation de la classe Vector est plus lente que celle de la classe ArrayList ?

Bien que permettant la création de liste dynamiques, les classes sont différentes. La classe ArrayList n'est pas synchronisée au contraire de la classe Vector. La pose des verrous à chaque utilisation d'un élément de type vector ralentit son exécution.

Processus (3 points)

Dans la programmation par processus, le code suivant, permet à plusieurs processus de stocker un objet dans un tableau *tampon*. Que se passe-t-il si un processus appelle la fonction *depose* alors qu'un processus est déjà entré dedans ?

Si un processus est déjà entré et qu'il n'est pas en attente (wait) alors l'autre processus est automatiquement mis en attente à l'entrée de la fonction. Il entrera lorsque le 1er sera mis en attente par un wait ou lorsque ce 1er sortira de la fonction.

Que se passe-t-il si un processus entre dans la fonction *depose* et que le nombre d'objets actuellement dans le tableau (*nbObjetsCourant*) est au maximum (*taille-1*) ? **(3 points)** :

Si le nombre d'objets a atteint le seuil, le processus sera mis en attente. Il ne sera 'réveillé' que par un signal notify émis par un autre processus. Le processus vérifiera la condition pour « re-renter » ou non en attente

```
public synchronized void depose(Object obj) {
    while (nbObjetsCourants == (taille-1)) {
        try { wait(); }
        catch (InterruptedException e) {}
    }
    try { tampon[nbObjetsCourants] = obj; }
    catch (Exception e) {System.out.println(e);}
    nbObjetsCourants++;
    produits++;
    notify();
}
```

ramasse miettes (4 points)

Quel est le principe du garbage collector (ramasse miettes) ?

Il surveille périodiquement la mémoire virtuelle pour vérifier si chaque bloc est bien lié à au moins une référence. Sinon, la mémoire est libérée.

Comment accélérer l'exécution du code suivant : (Random est une classe dont la méthode *nextInt(borne)* retourne un entier au hasard pris entre 0 et *taille-1*)

Il faut déplacer la création de l'objet hasard avant la boucle pour gagner ainsi le temps de création de 9999 objets...

```
int taille=10000 ;
int[] tab = new int[taille]
for(int i=0 ; i<taille; i++)
```

```
{  
    Random hasard = new Random() ;  
    tab[i] = hasard.nextInt(taille) ;  
}
```

Réseau (2 points)

En Java, quels sont les moyens possibles pour faire communiquer deux applications entre elles ?

Exercices (10 points)

Exercice 1 : Création d'Objets (2 points)

Créer une classe représentant un *Hotel*

Un objet *Hotel* est composé d'un nom, de deux coordonnées GPS réelles (*gpsx*, *gpsy*), d'un nom et d'un coût réel / nuit.

Le constructeur par défaut de *Hotel* initialise les coordonnées à 0 et le nom à "".

Vous écrirez également un constructeur de *Hotel* prend en paramètre son nom et ses coordonnées.

Surcharger la méthode **toString** pour retourner une représentation d'un objet *Hotel* par son nom, ses coordonnées et son coût.

```
Class Hotel{
    String nom;
    Double gpsx;
    Double gpsy;
    Double cout;
    Hotel(){nom = « » ; gpsx = gpsy = 0; }
    Hotel(String nom, Double gpsx, Double gpsy ){this.nom = nom ; this.gpsx = gpsx; this.gpsy = gpsy;}
    public String toString(){return "Hotel " + nom + " (" +gpsx+ " , "+gpsy+ " ) ";}
}
```

Exercice 2 : Héritage (2 points)

Créer une classe *HotelZoneS* qui étend *Hotel*. Un objet *HotelZoneS* possède en plus une variable *cat* de type *Catégorie*. Permet de décrire des hotel placé en zone géographique non recommandée pour cause sanitaire.

Le type *Catégorie* est une énumération à définir, contenant les constantes *OK*, *ATEST*, *QUARANTAINE*, *FERME*. Le constructeur par défaut de *HotelZoneS* initialise la catégorie 'cat' à *OK*.

Un constructeur de *HotelZoneS* prenant en paramètre le nom, les coordonnées et sa catégorie doit être défini. Il doit utiliser un constructeur de la classe mère.

Redéfinir la méthode **toString()** pour *HotelZoneS*, pour retourner une représentation par son nom, ses coordonnées, son numéro d'instance, et sa catégorie. (2 points)

```
enumeration Categorie {OK, ATEST, QUARANTAINE, FERME;}
class HotelZoneS extends Hotel{
    Catégorie cat;
    HotelZoneS(){cat = Catégorie.OK; }
    HotelZoneS(String nom, Double gpsx, Double gpsy, Catégorie cat ){super(nom, gpsx, gpsy); this.cat = cat;}
    public String toString(){return super.toString() + " categorie " + cat;}
}
```

Exercice 3 : Comparaison et tri (8 points)

Afin de pouvoir trier les collections d'objets de type *Hotel* et d'objets de type *HotelZoneS*, ces classes doivent implémenter l'interface **Comparable**.

Réécrivez l'entête de ces classes, et **écrivez** la fonction **int compareTo(Object o)** nécessaire dans ces deux classes pour que le tri de *Hotel* se fasse par nom en ordre alphabétique, et que le tri de *HotelZoneS* se fasse par catégorie, puis par nom. (4 points)

Aucune méthode de tri n'est demandée, la méthode `Collections.sort(liste)` de Java effectue ce tri :

```
ArrayList< Hotel > lesPlagesTriees = Collections.sort(lesHotels);
```

```
ArrayList< HotelZoneS > lesSurveilléesTriees = Collections.sort(lesHotelsZoneS);
```

On suppose *lesHotels* : une liste d'Hotels, et *lesHotelsZoneS* une liste d'Hotels en zone S.

Par lambda-expressions :

- **Ecrivez** les lignes qui calculent le prix moyen de tous les hôtels de la liste *lesHotels*. (2 points)

- **Ecrivez** les lignes qui permettent de supprimer de la liste *lesHotelsZoneS* (une liste de *HotelsZoneS*) tous les hotels qui ne sont pas OK (2 points)

```
Class Hotel implements Comparable<Hotel>{.....
public int compareTo(Hotel autre){return nom.compareTo(autre.nom);}
}
```

```
Class HotelZoneS implements Comparable< HotelZoneS >{.....
public int compareTo(Hotel autre){
    int comp = cat.compareTo(autre.cat);
    if (comp==0) comp = nom.compareTo(autre.nom);
    return comp;
}
```

```
Stream<Hotel> flux = lesHotels.stream();
OptionalDouble prixMoyen = flux.mapToInt(h->h.getCoût()).average();
prixMoyen.ifPresent(prix-> System.out.printf("prix moyen = %.2f\n", prix));
```

```
lesHotelsZoneS.removeIf(h->(h.cat != Categorie.OK));
```