

Seule une feuille recto-verso d'aide mémoire manuscrite est autorisée.

NOM :

PRÉNOM :

**Préambule.**

Malgré les apparences, les questions qui vont suivre sont indépendantes. Il est toutefois conseillé de lire le sujet dans son ensemble avant de commencer, pour avoir une bonne compréhension de l'application autour de laquelle le sujet s'articule.

**Mise en situation.**

Jeune diplômé recruté dans une SSII, vous êtes envoyés dans une petite entreprise pour améliorer les performances de son réseau informatique interne. Votre tâche consiste à étudier les différentes configurations réseau possibles pour pouvoir choisir celle qui sera adaptée aux besoins et moyens de l'entreprise. Vous décidez d'implémenter un simulateur en Java, qui va vous permettre de tester rapidement les configurations de votre choix. Pour vous concentrer sur la conception du simulateur et la façon dont ensuite vous allez l'utiliser, l'entreprise vous demande de recruter un stagiaire qui se chargera de l'implémentation.

★ Exercice 1.

Vous recevez un candidat et vous lui avez préparé un questionnaire (ci-dessous) évaluant ses connaissances en programmation objet et en Java.

Correcte ?

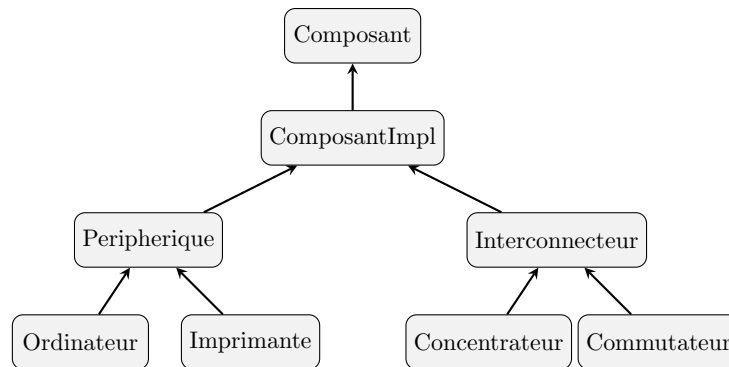
- ☐ Une classe doit au moins contenir une variable pour être définie.
- ☐ Une classe est une réalisation d'une instance.
- ☐ Dans une méthode, il est possible d'accéder à l'instance courante à l'aide du mot-clé **this**.
- ☐ Le mot-clé **private** permet de garantir qu'une variable ne pourra être accédée que l'instance à laquelle elle appartient.
- ☐ Toute classe implémentant une interface doit définir les méthodes déclarées dans cette interface.
- ☐ Il est possible de déclarer des variables dans une interface si elles sont déclarées **static**.
- ☐ Lorsque qu'une classe définit plusieurs constructeurs, ceux-ci seront exécutés dans l'ordre de définition.
- ☐ Il est possible d'appeler un constructeur à partir d'un autre constructeur de la même classe.
- ☐ Il n'est pas possible de définir un constructeur sans paramètre car celui-ci est toujours défini automatiquement par le compilateur.
- ☐ Une classe mère ne peut pas définir de membres privés.
- ☐ Une classe mère a accès à tous les membres de ses sous-classes.
- ☐ Une classe ne peut étendre qu'une seule classe.
- ☐ En Java, toutes les classes implémentent l'interface **Object**.
- ☐ Les variables statiques sont définies dans des méthodes statiques.
- ☐ Il est possible d'accéder à une variable statique définie dans une classe sans créer d'instance de cette classe.
- ☐ Les variables statiques ne peuvent être utilisées que dans des méthodes statiques.
- ☐ Les méthodes statiques peuvent appeler des méthodes non statiques sans préciser sur quelle instance elles s'appliquent.
- ☐ On ne peut pas hériter de classes abstraites.
- ☐ Une classe abstraite doit définir des méthodes abstraites.
- ☐ Une méthode abstraite peut être appelée dans la classe qui la définit.
- ☐ Une sous-classe qui étend une classe abstraite peut également être définie comme abstraite.
- ☐ Un objet avec une référence **final** ne peut pas être modifié.
- ☐ Une classe ne peut pas hériter d'une classe finale.
- ☐ Une classe déclarée **final** ne peut pas être instanciée plus d'une fois.
- ☐ le mot clé **try** permet de lancer une exception.
- ☐ Les exceptions sont des objets.
- ☐ Un bloc **try** doit être suivi d'au moins un bloc **catch** ou/et d'un bloc **finally**.
- ☐ Après un ou plusieurs blocs **try/catch**, un bloc **finally** est obligatoire.
- ☐ Les méthodes de la classe **String** ne modifient jamais le contenu du receveur.
- ☐ L'instruction **String s = "1"+"1"** affecte la valeur "2" à la variable **s**.

▷ Question 1. (4 pts / -0.25 par réponse fausse) Corrigez ce questionnaire en indiquant quelles sont les affirmations correctes.

## ★ Exercice 2.

Le réseau est constitué de périphériques (ordinateurs, imprimantes), et d'appareils d'interconnexion (commutateurs, concentrateurs). Les concentrateurs transmettent les messages à tous les composants réseau qui leur sont connectés, tandis que les commutateurs n'envoient le message qu'au bon destinataire. Un périphérique n'est connecté qu'à un seul concentrateur ou commutateur. Pour les besoins du simulateur, les classes vont implémenter une méthode `transmettre(...)` qui simule l'envoi d'un message quelconque à un composant. Cette méthode compte et retourne le nombre de messages qui seraient envoyés dans le réseau en situation réelle. Les appareils d'interconnexion ont un coût (deux concentrateurs coûtent le même prix, deux commutateurs coûtent le même prix). Chaque composant aura également un numéro entier servant à l'identifier (deux composants de même type et de même numéro seront considérés comme égaux – un concentrateur et une imprimante sont donc toujours différents même si ils ont le même numéro, deux concentrateurs sont égaux si et seulement si ils ont le même numéro –).

Vous choisissez la hiérarchie de classes/interfaces illustrée ci-dessous pour votre application.



▷ **Question 2.** (3 pts) Expliquez au stagiaire recruté comment l'implémenter correctement :

- quelles sont les interfaces ?
- quelles sont les classes abstraites ?
- où la méthode `transmettre(Composant origine, Composant destination)` doit être définie ?
- où doit elle être implémentée ?
- où sont définis les attributs de classe nécessaires à l'implémentation – `prix` (de type `int`), `numero` (de type `int`), `composants` (de type `List<Composant>`), `connecteA` (de type `Interconnecteur`) ?

## ★ Exercice 3.

Le stagiaire vient vous voir au début de son travail en vous expliquant que son code ne compile pas et qu'il n'en voit pas la cause.

Constructeurs dans la classe `ComposantImpl`

```

1 public ComposantImpl(int n){
2     numero = n;
3 }
  
```

Constructeurs dans la classe `Interconnecteur`

```

1 // on passe en paramètres la valeur du numéro, et la liste des composants connectés à
2 // l'appareil d'interconnexion
3 public Interconnecteur(int n, List<Composant> l) {
4     this(n);
5     composants = l;
6 }
7
8 // on passe en paramètres la valeur du numéro, la liste est initialisée vide
9 // et elle sera remplie ultérieurement par un appel à la methode connecter(...).
10 public Interconnecteur(int n) {
11     composants = new ArrayList<Composant>();
12     numero = n;
13 }
  
```

Constructeurs dans la classe `Commutateur`

```

1 public Commutateur(int n, List<Composant> l) {
2     cout = 25;
3     this(n);
4     composants = l;
5 }
6
7 public Commutateur(int n) {
8     cout = 25;
9     numero = n;
10 }
  
```

```

1 // on passe en paramètres le numéro du périphérique, ainsi que l'appareil d'interconnexion
2 // auquel il est connecté
3 public Peripherique(int n, Interconnecteur i) {
4     numero = n;
5     connecteA = i;
6 }

```

▷ **Question 3.** (2 pts) Après avoir examiné ses constructeurs (ci-dessus), expliquez et corrigez ses erreurs. (Il y a 4 constructeurs qui ne compilent pas).

★ **Exercice 4.**

Le stagiaire rajoute le code suivant pour tester l'égalité de deux composants.

```

1 // Dans la classe Ordinateur
2 public boolean equals(Ordinateur other) {
3     return this.numero == other.numero;
4 }

```

```

1 // Dans la classe Imprimante
2 public boolean equals(Imprimante other) {
3     return this.numero == other.numero;
4 }

```

```

1 // Dans la classe Commutateur
2 public boolean equals(Commutateur other) {
3     return this.numero == other.numero;
4 }

```

```

1 // Dans la classe Concentrateur
2 public boolean equals(Concentrateur other) {
3     return this.numero == other.numero;
4 }

```

▷ **Question 4.** (2 pts)

- Expliquez pourquoi c'est une mauvaise implémentation en exhibant un exemple de code qui devrait logiquement renvoyer **true** mais renvoie **false**.
- Indiquez les corrections à apporter.

★ **Exercice 5.**

Maintenant que tout est supposé fonctionner, le stagiaire écrit du code pour tester la méthode **transmettre**. Le code (considéré correct) de cette méthode est donné pour les classes **Concentrateur** et **Peripherique**.

```

1 // Dans la classe Concentrateur
2 public int transmettre(Composant source, Composant destination) {
3     if (this.equals(destination)) {
4         //ne pas transmettre plus loin si le Concentrateur est la destination du message
5         return 0;
6     }
7     int res = 0;
8     for (int i = 0; i < composants.size(); i++) {
9         //transmettre à tous les Composants connectés, sauf celui qui a envoyé le message
10        if (!composants.get(i).equals(source)) {
11            res += 1 + composants.get(i).transmettre(this, destination);
12        }
13    }
14    return res;
15 }

```

```

1 // Dans la classe Peripherique
2 public int transmettre(Composant source, Composant destination) {
3     if (this.equals(destination)) {
4         //ne pas transmettre si le Peripherique est la destination du message
5         return 0;
6     }
7     if (this.connecteA.equals(source)) {
8         //ne pas retransmettre si le message vient de l'Interconnecteur
9         return 0;
10    }
11    //transmettre sinon
12    return 1 + this.connecteA.transmettre(this, destination);
13 }

```

```

1 // Dans la classe Test
2 public void test1(){
3     // o1 et o2 sont reliés ensembles par un Concentrateur
4
5     int num1 = 1;
6     int num2 = num1;
7     num2++;
8
9     Interconnecteur c1 = new Concentrateur(num1);
10
11    Ordinateur o1 = new Ordinateur(num1,c1);

```

```

11 Ordinateur o2 = o1;
12 o2.numero = num2;
13
14 c1.connecter(o1);
15 c1.connecter(o2);
16
17 //source est mise à null pour le premier envoi
18 System.out.println(o1.transmettre(null,o2));
19 }
20
21 public void test2() {
22     // les trois ordinateurs sont connectés chacun à un concentrateur,
23     // les trois concentrateurs sont reliés entre eux
24
25     Interconnecteur c1 = new Concentrateur(1);
26     Interconnecteur c2 = new Concentrateur(2);
27     Interconnecteur c3 = new Concentrateur(3);
28
29     Ordinateur o1 = new Ordinateur(1, c1);
30     c1.connecter(o1);
31
32     Ordinateur o2 = new Ordinateur(2, c2);
33     c2.connecter(o2);
34
35     Ordinateur o3 = new Ordinateur(3, c3);
36     c3.connecter(o3)
37
38     c1.connecter(c2);
39     c2.connecter(c3);
40     c3.connecter(c1);
41
42     //source est mise à null pour le premier envoi
43     System.out.println(o1.transmettre(null, o3));
44 }

```

Néanmoins le premier test du stagiaire renvoie la valeur 0 alors qu'il s'attendait à la valeur 2, et le deuxième test semble ne rien renvoyer.

▷ **Question 5.** (4 pts) A l'aide de schéma mémoire (un pour chaque test), expliquez ces comportements.

### ★ Exercice 6.

Vous souhaitez pouvoir savoir par quels composants réseau un message est passé, le code suivant est rajouté au début des méthodes `transmettre` par le stagiaire :

```

1  if(source == null) {
2      System.out.println("source initiale, le message va être envoyé");
3  } else if (source instanceof Ordinateur) {
4      System.out.println("message reçu depuis l'ordinateur " + source.numero + ".");
5  } else if (source instanceof Imprimante) {
6      System.out.println("message reçu depuis l'imprimante " + source.numero + ".");
7  } else if (source instanceof Concentrateur) {
8      System.out.println("message reçu depuis le concentrateur " + source.numero + ".");
9  } else if (source instanceof Commutateur) {
10     System.out.println("message reçu depuis le commutateur " + source.numero + ".");
11 }

```

▷ **Question 6.** (2 pts) Proposez une implémentation plus efficace qui prend en compte la hiérarchie des classes. Deux modifications majeures sont à apporter.

★ **Exercice 7.** Pour éviter le problème détecté dans la méthode `test2()` de la question 5, on souhaite rajouter un paramètre `nbAppel` (de type `int`) dans la méthode `transmettre` qui est incrémenté à chaque appel récursif. Si ce paramètre excède un seuil donné (10 par exemple), une exception `TropDeMessagesException` (classe supposée donnée) est levée. On veut ensuite que l'appel initial affiche un message d'erreur contenant le numéro et le type de la source initiale, chaque appel récursif devra donc renvoyer l'erreur à la méthode appelante jusqu'au premier appel (celui avec `nbAppel == 0`), et ce dernier affichera le message.

▷ **Question 7.**

- (a) (2 pts) Mettez à jour la méthode `transmettre` de la classe `Concentrateur`.
- (b) (1 pt) Mettez à jour la dernière ligne de la méthode `test2()`.

▷ **Question 8.** (bonus +2 pts) Proposez une solution permettant d'avoir accès aux composants par lesquels est passée la transmission qui a déclenché l'exception et pour les afficher après le message d'erreur.

### ★ Exercice 8.

▷ **Question 9.** (bonus +3 pts)

Proposez une implémentation de la méthode `transmettre` de la classe `Commutateur` (sans les ajouts des questions 6 et 7). Rajoutez des méthodes si besoin et commentez votre code.