

**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 ?

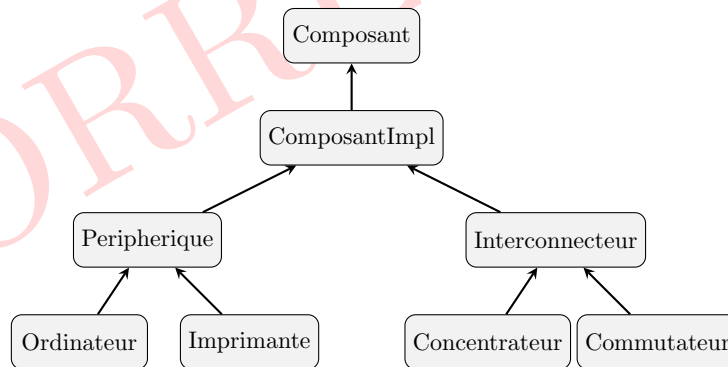
- ☐ Une classe doit au moins contenir une variable pour être définie.
- ☐ Une classe est une réalisation d'une instance.
- ☒ Dans une méthode, il est possible d'accéder à l'instance courante à l'aide du mot-clé **this**.
- ☐ Le mot-clé **private** permet de garantir qu'une variable ne pourra être accédée 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`) ?

### Réponse

**Composant** est une interface où la méthode `transmettre` sera définie (donc uniquement le profil de la méthode).

**ComposantImpl** est une classe abstraite qui contient la définition de l'attribut `numero`.

**Péripherique** est une classe abstraite qui contient la définition de l'attribut `connecteA` et qui peut contenir le code factorisé de la méthode `transmettre`.

**Interconnecteur** est une classe abstraite qui contient la définition de l'attribut `prix` et de l'attribut `composants`.

**Ordinateur** et **Imprimante** sont des classes concrètes avec l'implémentation de la méthode `transmettre` (ou du moins la partie non factorisée dans **Péripherique**).

**Concentrateur** et **Commutateur** et **Commutateur** sont des classes concrètes avec l'implémentation de la méthode `transmettre`.

### Fin réponse

## ★ 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 = 1;
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 = 1;
5 }
6
7 public Commutateur(int n) {
8     cout = 25;
9     numero = n;
10 }

```

Constructeurs dans la classe Peripherique

```

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).

### Réponse

Il n'y a pas d'erreur dans le constructeur `ComposantImpl(int n)`.

Il y a une erreur dans le constructeur `Interconnecteur(int n)`. Aucun appel explicite au super constructeur de la classe `ComposantImpl` n'est fait. Le compilateur va donc ajouter un appel implicite au constructeur par défaut (sans paramètre). Or, un tel constructeur n'existe pas dans la classe `ComposantImpl`. Il faut donc soit ajouter un constructeur sans paramètre dans la classe `ComposantImpl`, soit faire appel de manière explicite au super constructeur avec un paramètre (`ComposantImpl(int n)`) en rajoutant l'instruction `super(n)` entre les lignes 10 et 11.

Il y a une erreur dans le constructeur `Commutateur(int n, List<Composant> l)`. Celui-ci fait appel au constructeur frère par l'instruction `this(n)` se trouvant à la ligne 3. Cependant cet appel doit être la première instruction du constructeur. Autrement dit, il faut bouger cette instruction entre les lignes 1 et 2. Il y a une erreur dans le constructeur `Commutateur(int n)`. Il s'agit de la même erreur que pour le constructeur `Interconnecteur(int n)`, il faut soit ajouter un constructeur sans paramètre dans la classe `Interconnecteur`, soit faire un appel explicite au super constructeur avec un paramètre en ajoutant l'instruction `super(n)` entre les lignes 7 et 8.

Il y a une erreur dans le constructeur `Peripherique(int n, Interconnecteur i)`. Il faut soit ajouter un constructeur sans paramètre dans la classe `ComposantImpl`, soit faire un appel explicite au super constructeur avec paramètre.

### Fin réponse

#### ★ Exercice 4.

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

Dans la classe `Ordinateur`

```

1 public boolean equals(Ordinateur other) {
2     return this.numero == other.numero;
3 }

```

Dans la classe `Imprimante`

```

1 public boolean equals(Imprimante other) {
2     return this.numero == other.numero;
3 }

```

Dans la classe `Commutateur`

```

1 public boolean equals(Commutateur other) {
2     return this.numero == other.numero;
3 }

```

Dans la classe `Concentrateur`

```

1 public boolean equals(Concentrateur other) {
2     return this.numero == other.numero;
3 }

```

▷ **Question 4.** (2 pts)

- (a) Expliquez pourquoi c'est une mauvaise implémentation en exhibant un exemple de code qui devrait logiquement renvoyer `true` mais renvoie `false`.

Réponse

Avec la définition actuelle, le scénario suivant poserait problème :

```
1 Commutateur c = new Commutateur(27);
2 Composant cp = new Ordinateur(1, c);
3 Ordinateur cp2 = new Ordinateur(1, c); // même 'id' et relié au même commutateur que cp
```

L'expression `cp.equals(cp2)` serait vraie alors que l'expression `cp2.equals(cp)` serait fausse. En effet, lors de l'évaluation de cette dernière expression, ce serait la méthode `equals` telle qu'elle est définie dans la classe `Object` qui sera exécutée et non pas celle définie dans la classe `Ordinateur`. Cette erreur survient car lors de la compilation, une liste des profils des méthodes candidates à la liaison dynamique est construite. Comme cette liste est construite à la compilation, elle repose sur les types statiques des variables. Ainsi, à la compilation, il sera décidé que la méthode à appeler aura devrait avoir comme profil soit `boolean equals(Composant c)` soit `boolean equals(Object o)`. Comme il n'existe pas de méthode avec le premier profil à l'exécution, c'est une méthode avec le second profil qui est cherchée. Il s'agira de la méthode définie dans la classe `Object` dont le comportement par défaut est de comparer les références des objets.

Pour corriger, la méthode `equals` devrait redéfinir la méthode telle que définie par défaut dans la classe `Object`. Le profil de cette méthode est `public void equals(Object o)`. L'erreur porte donc sur le type du paramètre. Il faut qu'il soit de type `Object` et non du type de la classe où la méthode `equals` est définie.

Fin réponse

- (b) Indiquez les corrections à apporter.

Réponse

Il faut réécrire les constructeurs de la manière suivante : il faut modifier le type du paramètre qui doit être `Object` ; effectuer un test sur le type dynamique du paramètre (à l'aide d'`instanceof`) ; puis transtyper (`cast`) le paramètre avant d'accéder aux variables d'instance. On peut également s'assurer que le paramètre n'est pas `null`. Cela donne donc des constructeurs qui ont la forme suivante :

```
1 // dans Ordinateur
2 public boolean equals(Object other) {
3     if (other == null && ! other instanceof Ordinateur) {
4         return false;
5     } else {
6         Ordinateur o = (Ordinateur) other;
7         return this.id == o.id
8     }
9 }
```

Fin réponse

★ **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`.

Dans la classe `Concentrateur`

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

```

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);
12         Ordinateur o2 = o1;
13         o2.numero = num2;
14
15         c1.connector(o1);
16         c1.connector(o2);
17
18         //source est mise à null pour le premier envoi
19         System.out.println(o1.transmettre(null,o2));
20     }
21
22     public void test2() {
23         // les trois ordinateurs sont connectés chacun à un concentrateur,
24         // les trois concentrateurs sont reliés entre eux
25
26         Interconnecteur c1 = new Concentrateur(1);
27         Interconnecteur c2 = new Concentrateur(2);
28         Interconnecteur c3 = new Concentrateur(3);
29
30         Ordinateur o1 = new Ordinateur(1, c1);
31         c1.connector(o1);
32
33         Ordinateur o2 = new Ordinateur(2, c2);
34         c2.connector(o2);
35
36         Ordinateur o3 = new Ordinateur(3, c3);
37         c3.connector(o3)
38
39         c1.connector(c2);
40         c2.connector(c3);
41         c3.connector(c1);
42
43         //source est mise à null pour le premier envoi
44         System.out.println(o1.transmettre(null, o3));
45     }

```

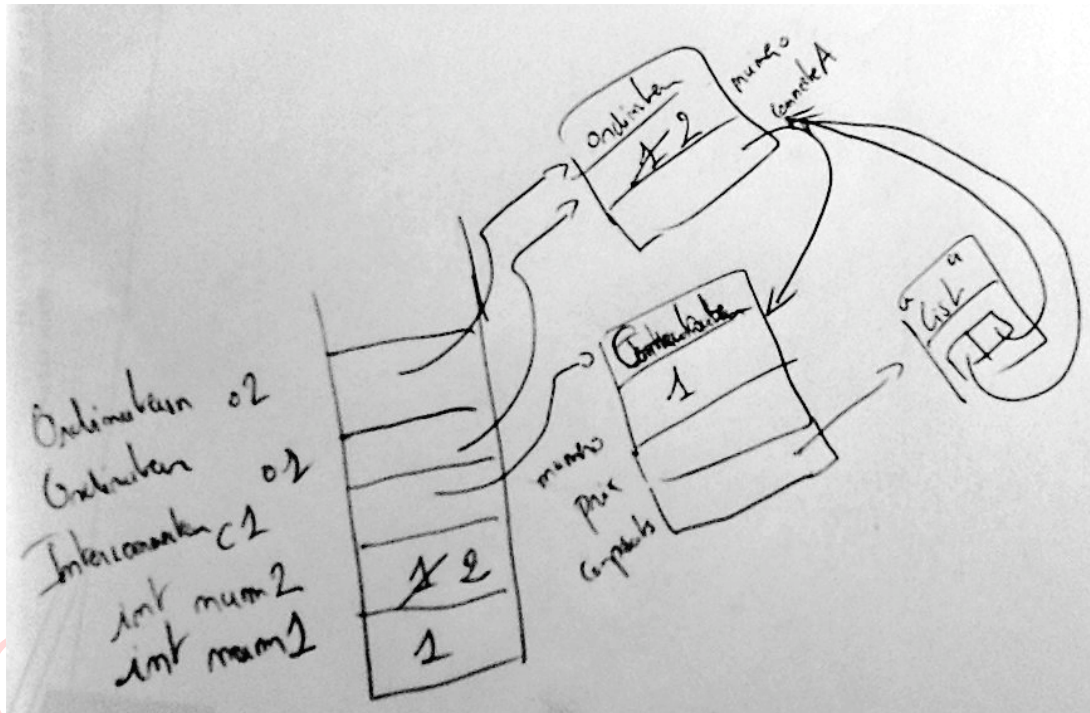
Néanmoins le premier test du stagiaire renvoie la valeur 0 alors qu'il s'attendait à la valeur 2, et le deuxième test semble ne rien renvoyer.

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

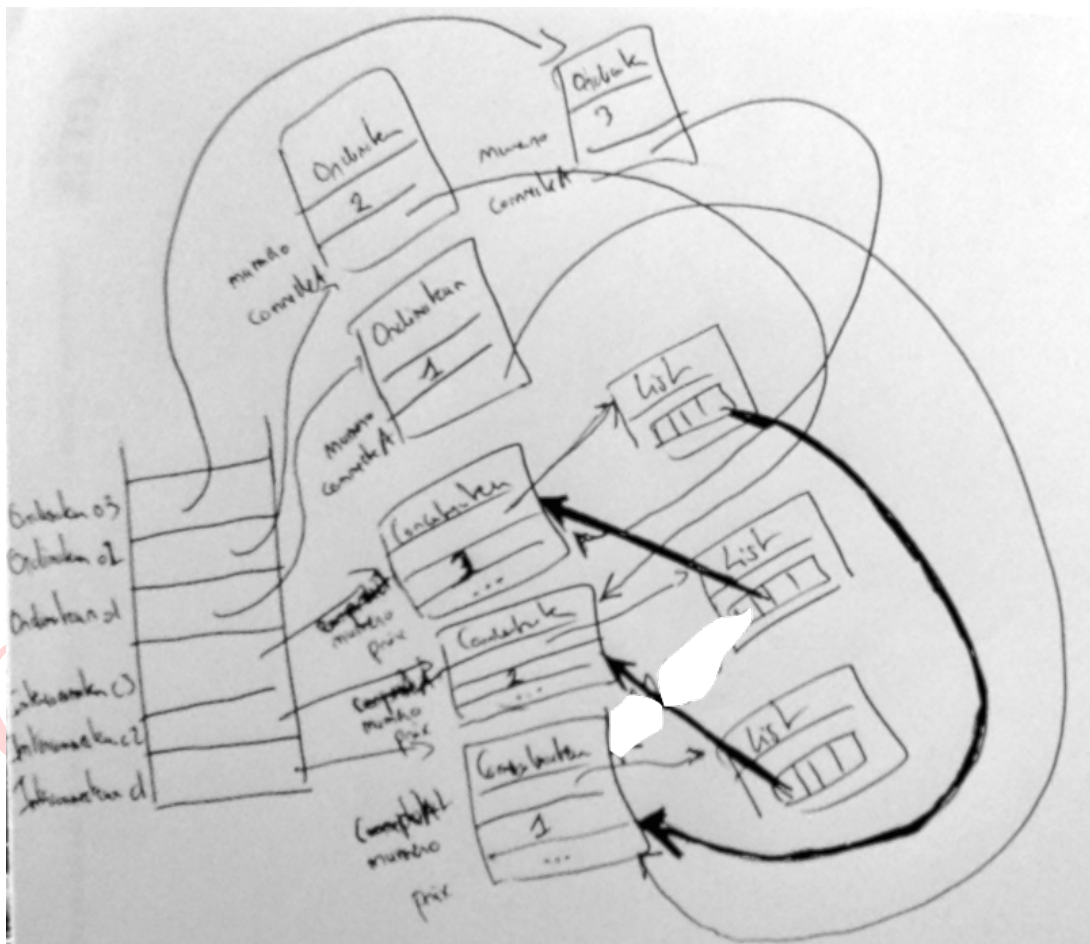
### Réponse

Dans le premier test (`test1()`), ce sont deux références vers le même ordinateur qui sont connectées au concentrateur. Lors de l'appel à transmettre, le message part donc d'un ordinateur qui est le même qui celui de destination.





Dans le second test (**test2()**), il y a bien 3 ordinateurs distincts et 3 concentrateurs distincts également. Cependant, les concentrateurs sont reliés de manière circulaire. Ainsi lorsqu'un message est envoyé par l'ordinateur o1, o1 l'envoie au concentrateur c1. Puisque les concentrateurs envoient leur messages à toutes leurs connexions, c1 envoie le message à c2, qui va l'envoyer à c3 qui lui va le réenvoyer à c1 ... le message va donc boucler dans le réseau.



Fin réponse

## ★ 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.

Réponse

La solution repose sur l'exploitation du mécanisme de liaison dynamique. Il n'est pas nécessaire de tester le type du paramètre `source`. Il suffit de faire un appel de méthode sur l'objet référencé par la variable `source`. Cet appel sera résolu en utilisant le type dynamique (le type de l'objet référencé à l'exécution) et la "bonne" méthode sera ainsi appelée. Il suffit donc de remplacer le code proposé pour la méthode `transmettre` par :

```

1  if(source == null) {
2      System.out.println("source initiale, le message va être envoyé");
3  } else {
4      System.out.println(source.getMessage() + this.numero + ".");
5  }

```

Il faut alors ajouter une méthode `String getMessage()` dans l'interface `Composant` et l'implémenter dans chaque classe (`Ordinateur`, `Imprimante`, `Concentrateur`, `Commutateur`) selon le patron ci-dessous :

```

1  // dans la classe Ordinateur
2  public String getMessage() {
3      return "message reçu depuis l'ordinateur ";
4  }

```

Le nouveau code proposé n'a pas besoin d'être dupliqué dans toutes les méthodes `transmettre` de chaque classe, il peut être factorisé dans la classe `ComposantImpl` en tant que code de la méthode `transmettre`. Il faudra alors faire appel à cette méthode dans les autres méthodes `transmettre` qui seront redéfinies dans chaque sous-classe par un appel à `super.transmettre(...)`.

Fin réponse

★ **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`.

Réponse

```

1  // Dans la classe Concentrateur
2  public int transmettre(Composant source, Composant destination, int nbAppel) throws TropDeMessagesException {
3      if (nbAppel > 10) {
4          throw new TropDeMessagesException();
5      }
6      if (this.equals(destination)) {
7          //ne pas transmettre plus loin si le Concentrateur est la destination du message
8          return 0;
9      }
10     return 1 + destination.transmettre(source, destination, nbAppel);
11 }

```



```

9   }
10  int res = 0;
11  for (int i = 0; i < composants.size(); i++) {
12    //transmettre à tous les Composants connectés, sauf celui qui a envoyé le message
13    if (! composants.get(i).equals(source)) {
14      try {
15        res = 1 + composants.get(i).transmettre(this, destination, nbAppel + 1);
16      } catch (TropDeMessageException ex) {
17        if (nbAppel == 0) {
18          System.out.println("trop de ... pour le message de "+source.getClass()+" num = source.numero");
19        } else {
20          throw ex
21        }
22      }
23    }
24  }
25  return res;
26 }

```

Fin réponse

(b) (1 pt) Mettez à jour la dernière ligne de la méthode `test2()`.

Réponse

La dernière ligne de `test2()` est remplacée par :

```

1  try {
2    System.out.println(o1.transmettre(null, o3, 0));
3  } catch (TropDeMessagesException ex) {
4    // Bien que la définition de la méthode transmettre(...) indique que cette méthode
5    // est susceptible de lever une exception. Cette exception ne devrait jamais arriver jusqu'ici.
6  }

```

Fin réponse

▷ **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.

Réponse

Une solution consiste à ajouter une `List<Composant>` comme attribut dans la classe d'exception `TropDeMessageException`. Puis, à chaque fois qu'une exception de ce type est attrapée dans la méthode `transmettre`, d'ajouter le composant courant (`this`) à cette liste avant de relancer l'exception.

Fin réponse

### ★ 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.

Réponse

// TODO:

Fin réponse