Comprendre l'API des collections

1 Généricité et sous-typage

Exercice 1 : Généricité et sous-typage : cas des classes
Intéressons nous au lien entre généricité et sous-typage dans le cas d'une interface ou classe.
1.1. Y a-t-il sous-typage entre List<String> et ArrayList<String> ? Si oui, dans quel sens ?
1.2. Y a-t-il sous-typage entre ArrayList<Object> et ArrayList<String> ? Si oui, dans quel sens ?
1.3. Que penser du programme suivant ?

```java
import java.util.ArrayList;

public class GenericiteSoustypageExempleClasse {

    public static void main(String[] args) {
        ArrayList<String> ls = new ArrayList<String>();
        ArrayList<Object> lo = ls;
        lo.add("texte");
        lo.add(15.5);
        System.out.println("premier élément  : " + lo.get(0));
        System.out.println("deuxième élément : " + lo.get(1));
    }

}
```

Est-ce qu'il compile ? Que donne l'exécution ? Que peut-on en conclure ?

Exercice 2 : Généricité et sous-typage : cas des tableaux
Intéressons sous au lien entre généricité et sous-typage dans le cas des tableaux.
2.1. Y a-t-il sous-typage entre un tableau d'objets `Object[]` et un tableau de chaînes `String[]` ? Si oui, dans quel sens ?
2.2. Est-ce que le programme suivant compile ? Que donne l'exécution ?

```java
import java.util.ArrayList;

public class GenericiteSoustypageExempleTableau {

    public static void main(String[] args) {
        String[] ts = new String[2];
        Object[] to = ts;
        to[0] = "texte";
        to[1] = 15.5;
        System.out.println("premier élément  : " + to[0]);
        System.out.println("deuxième élément : " + to[1]);
    }

}
```

Exercice 3 : Afficher une liste
On souhaite écrire une méthode qui permettra d'afficher une liste, quelque soit le type de ses éléments. Voici le programme qui est proposé.

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class AfficherListe {

    /** Afficher les éléments de liste, un par ligne... */
    public static void afficher(List<Object> liste) {
        for (Object o : liste) {
            System.out.println("  - " + o);
        }
    }

    public static void main(String[] args) {
        List<Object> lo = new ArrayList<>();
        Collections.addAll(lo, "un", "deux", 3);
        afficher(lo);

        List<String> ls = new ArrayList<>();
        Collections.addAll(ls, "un", "deux", "trois");
        afficher(ls);
    }
}
```

3.1. Est-ce que ce programme compile ? Que donne alors son exécution ?
3.2. Modifier le programme pour qu'il fonctionne. On utilisera la généricité.
3.3. Est-ce que nommer le paramètre de généricité est nécessaire ? Modifier le programme pour utiliser le type joker.

2 Bornes d'un paramètre de généricité

Exercice 4 : Manipuler des collections
Intéressons nous à deux opérations sur des listes qui s'appuient sur un critère. Un critère fournit une méthode qui s'applique sur un élément et retourne vrai ou faux suivant que le critère est vérifié ou non. La première opération s'appelle `tous`. Elle retourne vrai si et seulement si tous les éléments d'une collection satisfont un critère. La deuxième opération s'appelle `filtrer`. Elle permet d'ajouter dans une liste résultat tous les éléments d'une première liste qui respectent un critère donné. Ces méthodes sont définies comme méthodes de classe dans la classe `Outils`.

Notons qu'il seraient possible de les généraliser pour qu'elles travaillent sur des collections plutôt que des listes.
Remarque : Sous eclipse ou autre IDE, des erreurs seront certainement signalées sur les classes de test. Il ne faut pas chercher à les corriger avant que ce soit demandé dans les exercices.

4.1. Compléter le code de la méthode `tous` de la classe `Outils`. On la testera en utilisant la classe de test `TestOutilsTousString`.
4.2. La classe `TestOutilsTousNumber` signale des erreurs.
4.2.1. Expliquer ces erreurs.
4.2.2. Corriger la méthode `tous` pour réussir les tests de `TestOutilsTousNumber`.
4.3. Compléter le code de la méthode `filtrer` de la classe `Outils`. On la testera en utilisant la classe de test `TestOutilsFiltrerString`.
4.4. La classe `TestOutilsFiltrerStringObject` signale des erreurs.
4.4.1. Expliquer ces erreurs.
4.4.2. Corriger la méthode `filtrer` pour réussir les tests de `TestOutilsFiltrerStringObject`.
4.5. Tester (et corriger) la méthode `filtrer` avec la classe `TestOutilsFiltrerNumber`.

3 API Collection

Exercice 5 : Comprendre `Collections.binarySearch`
On s'intéresse à la méthode `binarySearch` qui prend deux paramètres, une liste et l'élément cherché. On peut consulter sa documentation.
5.1. Quelles sont les contraintes sur la liste ?
5.2. S'agit-il de programmation défensive ou offensive ?
5.3. À quoi correspond RandomAccess ?
5.4. Expliquer le code de cette méthode `binarySearch`. Voir le source de la classe Collections.

Exercice 6 : Comprendre List et Collections.unmodifiableList
Intéressons nous à `Collections.unmodifiableList`.
6.1. *Exemple1*. Compléter le code de la méthode `consulter` de `ExempleUnmodifiableList`. Il s'agit de remplacer les `TODO()` et les `XXX*` par le code attendu. La méthode `exemple1()` doit alors s'exécuter sans erreur.

Si des erreurs sont signalées, il faut comprendre les messages qui sont affichés car ils devraient vous permettre de comprendre le problème et comment le corriger.

Est-ce que la méthode consulter se contente de consulter la liste sans la modifier ? Est-ce que mettre le premier paramètre en **`final`** résoudrait le problème ?
6.2. Dans la question précédente, on s'est rendu compte que l'on ne pouvait pas faire confiance à la méthode qui dit ne pas modifier la liste. Elle a accès à notre liste et donc à toutes ses opérations, y compris celles de modification.

Proposer une manière pour garantir que la méthode `consulter` ne modifiera pas la liste qu'elle reçoit. Cette solution ne doit pas induire un surcout trop important, en particulier si la liste contient un grand nombre d'éléments, surtout qu'en général, la méthode ne devrait pas modifier la liste (puisqu'elle s'y est engagée). On ne mettra pas en œuvre cette solution. Il faut juste donner le principe.
6.3. *Exemple2*. Dans la méthode `exemple2`, on veut utiliser `Collections.unmodifiableList` pour rendre une liste non modifiable et ainsi répondre à la question 6.2. Compléter le code de exemple2().

6.4. *Code source.* Comprenons le fonctionnement de la méthode `unmodifiableList`.
6.4.1. Consulter le code source de la classe Collections de l'API Java 8 dans le fichier `/mnt/n7fs/ens/tp_cregut/src/Collections.java` ou sur `http://cregut.perso.enseeiht.fr/ens/src/Collections.java` pour répondre aux questions qui suivent.
6.4.2. Quel est le patron de conception mis en œuvre ?
6.4.3. Pourquoi un test est-il fait sur `RandomAccess` ?
6.5. Est-ce que l'utilisation de `Collections.unmodifiableList` garantit que les objets de la liste ne seront pas modifiés ?
6.6. *Synthèse.* Répondre de manière concise aux questions suivantes :

1. Peut-on déclarer une liste d'entiers en faisant `List<int>` ? Pourquoi ?
2. Combien y a t-il de méthodes `remove` sur List ?
3. Est-ce que `remove` retourne une information ? Si oui, quelle est sa signification ?
4. Peut-on faire `remove("abc")` sur une liste d'entier (`List<Integer>`) ?
5. Dans le code de `unmodifiableList`, à quoi correspond `RandomAccess` ?

4 Itérateurs

Exercice 7 : range de Python en Java
On veut faire l'équivalent du `range` de Python. On se limite à la forme qui prend trois paramètres entiers : début, fin, pas avec un pas positif. Voici un exemple d'utilisation.

```
1 for n in range(2, 11, 3):
2     print(n)
```

Ce programme affiche 2, 5 et 8 : les entiers de 2 (début) inclus à 11 (fin) exclu de 3 en 3 (pas).
7.1. Qu'affichera le programme suivant ?

```
1 for n in range(5, 12, 4):
2     print(n)
```

7.2. En Java, on peut faire l'équivalent de range. Ainsi, si on définit une méthode de classe `range` sur une classe `Range` qui est dans le paquetage `exercices`, on peut écrire :

```
1 import static exercices.Range.range;
2 ...
3 for (int n : range(2, 11, 3)) {
4     System.out.println(n);
5 }
```

7.2.1. Sur quoi peut s'appliquer un `foreach` en Java ?
7.2.2. Écrire la classe Java `exercices.Range`.
7.2.3. Reproduire l'exemple du sujet.
7.2.4. Tester votre classe `exercices.Range` avec la classe `exercices.TestRange`.
7.2.5. La fonction range de Python peut prendre deux paramètres (dans ce cas le pas vaut 1) ou un seul (il correspond à la fin, le début vaut 0 et le pas vaut 1). Ajouter ces possibiités et les tester avec la classe `exercices.TestRangeParametresOptionnels`.