# Design patterns, Java

# Exercices généralités, lambdas

### Préambule

Copier le programme principal et la méthode auxiliaire fournis :

public static void main(String[] args) {

List<String> laListe = Arrays.asList("Un", "Deux", "Trois", "Quatre", "Cinq", "Six", "Sept", "Huit", "Neuf");

affiche(laListe);

Collections.sort(laListe, new Comparator() {

public int compare(Object s1, Object s2) {

return ((Comparable)s1).compareTo(s2);

}

});

affiche(laListe);

}

public static void affiche(List<String> laListe) {

for (String suivante : laListe) {

System.out.println(" " + suivante);

}

System.out.println(" ---------");

}

### Lambda

Dans cet exercice, on modifie le code existant et on en ajoute. Il ne faut utiliser que des lambdas expressions

1. Remplacer le paramètres de Collections.sort par une lambda expression

Collections.sort(laListe, (s1, s2) → s1.compareTo(s2));

1. Afficher ensuite le contenu de la liste trié par ordre alphabétique inverse (en utilisant une lambda expression bien sûr)

Collections.sort(laListe, (s1, s2) → s2.compareTo(s1));

1. Afficher ensuite le contenu de la liste trié par longueur de chaîne croissante (en utilisant une lambda)

Collections.sort(laListe, (s1, s2) → s1.length() - s2.length());

1. Afficher ensuire le contenu de la liste en mettant d’abord les chaînes possédant un ‘e’, les autres chaînes ensuite (en utilisant une lambda)

Collections.sort(laListe, (s1, s2) → s1.contains("e") || ! s2.contains("e") ? -1:1;

### Référence à une méthode

1. Dans le tri par ordre alphabétique, remplacer la lambda par une référence à une méthode existante de la classe String

Collections.sort(String ::compareTo);

1. Dans le tri par ordre alphabétique inverse, remplacer la lambda par une référence à une méthode qu’il faudra écrire

Collections.sort(Main::compareStrings);

// …

public static int compareStrings(String s1, String s2) {

return s2.compareTo(s1);

}

### Interface fonctionnelle

1. Créer l’interface fonctionnelle « MeilleureChaine » qui déclare la méthode « meilleure ». Cette méthode prend deux chaînes en paramètres et renvoie un booléen

public interface MeilleurChaine {

public boolean meilleure(String s1, String s2);

}

1. Créer une fonction statique « String getMeilleure(String inS1, String inS2, MeilleureChaine inSelecteur) ». Cette fonction renvoie la première chaîne si inSelecteur.meilleure(inS1, inS2) vaut true, la seconde sinon.

public static String getMeilleure(

String inS1, String inS2, MeilleureChaine inSelecteur) {

return inSelecteur.meilleure(inS1, inS2) ? inS1 : inS2;

}

1. Appeler cette fonction statique avec les chaines « Un », « Deux » et la lambda adéquate pour obtenir la première chaîne par ordre alphabétique

String m1 = getMeilleure(«Un», «Deux»,

(s1, s2) → s1.compareTo(s2));

1. Appeler cette fonction statique avec les mêmes chaines et la lambda adéquate pour obtenir la chaîne de plus grande longueur

String m1 = getMeilleure(«Un», «Deux»,

(s1, s2) → s1.length() > s2.length());

1. Appeler cette fonction statique avec les mêmes chaines et une lambda adéquate pour obtenir une chaîne contenant un « e » s’il y en a une, sinon le premier argument de l’appel.

String m1 = getMeilleure(«Un», «Deux»,

(s1, s2) → s1.contains("e") || ! s2.contains("e") ? s1, s2);