Formation

Notre expertise est votre avenir

TRAVAUX PRATIQUES

Nouveautés Java 8

SOMMAIRE

TP1 - Les expressions lambda sans paramètre ni retour

Vous allez créer ici votre première utilisation d'expressions lambda (EL).

Vous pourrez, dans la suite des travaux pratiques faire autant de package qu'il y a de TP.

TP1 : package tp1.

Dans ce package :

Créez une interface nommée ImachineAlaver, possédant une méthode nommée void demarrer().

Créez une classe nommée Lavomatic, avec un main().

Dans le main(), écrivez une expression lambda sans paramètre, dont le contenu sera l'affichage à la console de « Elle démarre ».

Affectez cette expression lambda à une déclaration de référence à l'interface iMachineAlaver.

Comme les signatures de l'unique méthode ImachineAlaver et de l'expression lambda sont égales, cela ne pose pas de problèmes.

Appelez demarrer() via l'interface.

Testez.

TP2 - Les expressions lambda avec paramètre mais sans retour

Copiez / collez le package TP1 dans un autre, nommé TP2, si vous voulez garder la trace de chaque TP, sinon modifiez le premier.

Reprendre l'interface demarrer() de ImachineAlaver, et ajoutez une string en paramètre.

Faites-en sorte de pouvoir afficher dans la console le paramètre passé à l'appel de l'expression lambda.

TP3 - Les expressions lambda avec paramètre et retour

Copiez / collez le package TP2 dans un autre, nommé TP3, si vous voulez garder la trace de chaque TP, sinon modifiez le précédent.

Reprendre l'interface demarrer() de ImachineAlaver, et déclarez maintenant une string en type de retour.

Dans le corps de l'expression lambda, remplacez l'affichage console par un « return » du paramètre passé concaténé à une string constante, pour que cela reste compatible avec la méthode d'interface.

Vous récupérerez la string lors de l'appel de l'expression lambda, et l'afficherez à la console.

RESUME : La signature de l'expression lambda doit correspondre à l'interface à laquelle on l'affecte. Cette interface NE DOIT AVOIR QU'UNE METHODE.

TP4 - Simplification d'appel de l'expression lambda

Copiez / collez le package précédent dans ce nouveau nommé TP4, si vous voulez garder la trace de chaque TP, sinon modifiez le précédent.

Reprendre l'appel de l'expression lambda, qui devrait ressembler à cela :

```
ImachineAlaver iMachineAlaver = (String type)->{return "départ "+type;};
```

Vous allez maintenant retirer :

- Les parenthèses du paramètre
- Le type du paramètre
- Les accolades du code
- Et même le return, quoi devient implicite.

Testez, cela devrait continuer à fonctionner.

TP5 - Annotation @FunctionalInterface

Copiez / collez le package précédent dans ce nouveau nommé TP5, si vous voulez garder la trace de chaque TP, sinon modifiez le précédent.

Essayez d'ajouter une nouvelle méthode à l'interface ImachineAlaver.

Vous avez une erreur sur cette déclaration.

Faites : précéder l'interface ImachineAlaver de l'annotation @FunctionalInterface.

Vous avez de nouveau une erreur, mais maintenant, sur l'interface elle-même.

Conclusion : dans une interface fonctionnelle, vous n'avez droit qu'à une méthode déclarée, SAUF :

Si vous implémentez d'autres méthodes de l'interface, et que vous les préfixiez de default.

TP6 - Utilisation d'interfaces fonctionnelles livrées avec la JDK8

Créez un nouveau package nommé TP6.

Nous allons ici mettre en œuvre des interfaces fonctionnelles livrées avec la JDK.

Exemple 1 :

Nous allons d'abord utiliser des threads sans lambda ni interface fonctionnelle, puis nous ajouterons.

Créer une classe nommée Dummy, avec un main().

Ici la méthode sans lambda :

Dans cette classe, instanciez la classe Thread à laquelle vous allez passer une instance de la classe Runnable, pour laquelle vous allez implémenter void run(), et afficher dans cette méthode un message à la console. Lancez le thread par start().

Vous pouvez remarquer, dans la documentation, que Runnable est une interface fonctionnelle.

Transformez maintenant ce qui est passé en constructeur de thread en expression lambda compatible avec Runnable, et simplifiez le code comme au TP précédent

Exemple 2 :

Vous allez trier une liste, sans et avec l'utilisation de lambda.

La classe Dummy créée, créez une nouvelle classe nommée Machine.

Dans cette classe, qui sera un Java Bean, déclarer trois propriétés privées, nommées annee de type entier, nom de type string. Ajoutez constructeurs avec paramètre, accesseurs et toString().

Dans le main() de la classe Dummy, créez une liste de machines, en utilisant les classes List<> et Arrays.asList(…).

Vous allez trier cette liste, en utilisant la méthode sort de la classe Collections, en passant en paramètre de la méthode sort() la liste à trier, et une instance de la classe Comparator<>, pour laquelle vous implémenterez la méthode int compare (Machine o1, Machine o2). Dans cette implémentation, vous retournerez par exemple : o1.getAnnee()-o2.getAnnee() afin de trier par ordre ascendant ; inversez pour l'ordre descendant.

Une fois que vous avez cette liste triée qui fonctionne, appliquez une expression lambda afin de rendre ce code plus simple et plus succinct. Vous remplacerez donc le new Comparator, qui est en fait une interface fonctionnelle dans Java 8, n'ayant qu'une méthode.

Cela veut dire qu'à la place d'une classe anonyme, on peut passer une expression lambda, avec la restriction du fait que l'interface ne doit avoir qu'une méthode abstraite.

TP7 - Utilisation des références de méthodes

Créez un nouveau package nommé TP7, et une classe nommée Dummy oui autre.

Nous allons ici découvrir les références de méthode en Java8.

Exemple 1

Vous allez d'abord, avec les classes List<> et Arrays.asList() créer un tableau d'entiers.

Puis vous allez l'énumérer avec forEach, méthode de la liste d'entiers.

Vous utiliserez une expression lambda pour afficher à la console l'ensemble des entiers du tableau.

Vous pouvez remarquer que forEach attend en paramètre une interface fonctionnelle Consumer qui possède une méthode, accept, qui, elle, est compatible avec System.out.Println que vous avez utilisé.

Vous devriez avoir une ligne ressemblant à celle-ci :

```
listeentier.forEach((e->System.out.println(e)));
```

Comme le premier paramètre de forEach (Interface fonctionnelle de type Consumer) est compatible avec l'expression lambda passée, alors on peut remplacer l'expression lambda par une référence de méthode, utilisant le signe ::.

Remplacez donc votre expression lambda avec la référence de méthode, en référençant avec :: la méthode println de System.out.

Testez.

Exemple 2

Dans le fichier contenant la classe Dummy, déclarez une interface fonctionnelle nommée par exemple ImethodReference. Déclarez, dans cette interface, une méthode nommée affiche, sans paramètre et renvoyant void.

Dans la classe Dummy, déclarez une méthode static nommée par exemple maMethode, dont la signature est égale la méthode affiche() de l'interface fonctionnelle. Dans cette méthode, affichez à la console un message quelconque.

Dans le main() maintenant, vous allez déclarer une référence sur l'interface ImethodReference, et stocker dans cette référence la référence de méthode maMethode, en utilisant le signe ::

Toujours dans le main(), appelez maintenant la méthode affiche() en utilisant l'objet référence interface.

Il est équivalent de passer par une référence de méthode ou par une expression lambda.

Vous pouvez donc remplacer votre référence de méthode, utilisant le signe ::, par une expression lambda comme ()->System.out.println(« message »).

Exemple 3

Vous allez reprendre le code que vous venez d'écrire, et utiliser une référence de méthode, non pas sur une méthode static, mais sur une méthode d'instance de classe.

Retirez donc static de la déclaration maMethode, et changez l'affectation de la méthode static maMethode par l'appel à une méthode issue d'un new de Dummy, mais toujours en utilisant le signe ::.

Testez.

TP8 - Utilisation des références de constructeur

Créez un nouveau package nommé TP8, et une classe nommée Dummy oui autre.

Nous allons ici découvrir les références de méthode en Java 8.

Exemple 1

Vous allez ici déclarer une interface fonctionnelle signée de la même façon que le constructeur de la classe String Java, qui attend un tableau de char pour renvoyer une string. Vous nommerez cette interface ConstructInterface par exemple.

Une fois ceci fait, dans le main(), déclarez un tableau de char nommé par exemple chars chargé de caractères quelconques.

Puis déclarez une variable string qui recevra le résultat du new string (tableau préalablement déclaré).

Déclarez une référence sur l'interface ConstructInterface, nommée ci par exemple, et affectez-lui une expression lambda ayant une signature égale à ConstructInterface, comme :

(chars1)->{return new String(chars1);};

Une fois ceci fait affichez à la console, la string retourné en n'utilisant plus que la méthode de l'interface fonctionnelle déclarée ci.

Testez.

Exemple 2

Reprenez ce code, et allégez-le largement afin de remplacer l'expression lambda par une référence de méthode au constructeur string.

Vous utiliserez : String ::new

Testez

TP9 - Les variables au sein des expressions lambda

Créez un nouveau package nommé TP9, et une classe nommée Dummy oui autre.

Nous allons ici voir les limites de l'utilisation des variables au sein d'expressions lambda.

Déclarez une liste d'entiers chargée de quelques entiers en utilisant List<> et Arrays.asList(..) ;

Déclarez ensuite une variable nommée par exemple maVariable, de type entier.

Faites ensuite un forEach sur la liste des variables, en fournissant une expression lambda dans le forEach permettant d'afficher à la console les différentes valeurs numériques.

Testez.

Dans l'expression lambda, ajoutez maintenant une instruction qui modifie l'entier maVariable déclaré précédemment.

Vous ne pouvez pas modifier des données situées en dehors du scope de la fonction lambda.

Tentez de modifier la variable maVariable avant le forEach.

Vous ne pouvez pas. En fait, les données manipulées dans une expression lambda doivent soit être static, soit ne jamais avoir été modifiées avant.

Essayez maintenant de re-déclarer cette variable au sein de la fonction lambda. Vous ne pouvez pas non plus.

Déplacez maintenant la déclaration de variable après le forEach.

Cela fonctionne maintenant. Le compilateur ne contrôle que les déclarations situées avant l'expression lambda.

TP10 - L'utilisation de « this » dans une expression lambda

Créez un nouveau package nommé TP10, et une classe nommée Dummy oui autre.

Nous allons ici découvrir l'utilisation de this dans une expression lambda, comparée au this dans une classe anonyme.

Exercice 1

Créez une interface fonctionnelle nommée ImethodReference contenant une méthode, avec une signature du type void affiche().

Dans la classe Dummy, déclarez une variable de type entier nommée valeur, chargée à 1.

Créez ensuite une méthode nommée maMethode(), dans laquelle vous allez déclarer une référence vers l'interface fonctionnelle ImethodReference. Vous chargerez dans cette référence l'implémentation de l'interface et ce, via une classe anonyme. Vous déclarerez, dans cette classe, une variable du même nom que la variable de Dummy, nommée valeur mais valant 2. Dans la méthode affiche, affichez à la console la valeur de this.valeur. Toujours dans la méthode maMethode(), appelez ensuite affiche sur la référence d'interface déclarée dans maMethode.

Dans le main(), faites un new Dummy() sur lequel vous appellerez maMethode().

Testez et regardez la valeur affichée de valeur.

Le this correspond ici à l'objet implémenté en tant que classe anonyme.

Exercice 2

Reprendre le code et remplacez la classe anonyme par une expression lambda, laissez la déclaration de la variable valeur.

Exécutez et comparez. Le this correspond à l'objet englobant, et non plus à l'instance de la classe anonyme.

Voici une très importante différence entre classe anonyme et expression lambda.

TP11 - L'API Stream

Dans les TP qui vont suivre, vous allez travailler sur les mêmes données et les filtrer.

Il s'agit de données des vainqueurs des derniers Tours de France.

On donne ici le code de création des données, il faudra ensuite répondre aux questions en utilisant les Streams.

Une classe nommée Gagnants comporte les propriétés privées suivantes :

annee, nationalité, nom, equipe, kilometres, temps, joursenjaune.

Le temps est donné en Duration, formulée ainsi : PT89H40M27S, signifiant Period Of Time (PT), puis les heures, minutes, secondes.

Il existe autant d'accesseurs que de propriétés privées, ainsi qu'un constructeur permettant d'alimenter chaque instance de Gagnant.

La propriété nommée gagnantsTDF comporte la liste des gagnants, sous forme de List<Gagnants>.

Comprenez ce code, copiez-collez-le dans Eclipse.

Rendez-vous après le code.

```
package tp11streams;
import java.time.Duration;
import java.util.*;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Collectors.*;
public class Gagnants {
    private int annnee;
    private String nationalite;
    private String nom;
    private String equipe;
    private int kilometres;
    private Duration temps;
    private int joursenjaune;
    public Gagnants(int annee, String nationalite, String nom, String equipe, int kilometres,
    Duration temps,
                int joursenjaune) {
        this.annnee = annee;
        this.nationalite = nationalite;
        this.nom = nom;
        this.equipe = equipe;
        this.kilometres = kilometres;
        this.temps = temps;
        this.joursenjaune = joursenjaune;
    }
    public static final List<Gagnants> gagnantsTDF = Arrays.asList(
                new Gagnants(2006, "Spain", "Óscar Pereiro", "Caisse d'Epargne–Illes
    Balears", 3657,Duration.parse("PT89H40M27S"), 8),
```

```
            new Gagnants(2007, "Spain", "Alberto Contador", "Discovery Channel",
3570, Duration.parse("PT91H00M26S"),4),
            new Gagnants(2008, "Spain", "Carlos Sastre", "Team CSC", 3559,
Duration.parse("PT87H52M52S"), 5),
            new Gagnants(2009, "Spain", "Alberto Contador", "Astana", 3459,
Duration.parse("PT85H48M35S"), 7),
            new Gagnants(2010, "Luxembourg", "Andy Schleck", "Team Saxo Bank",
3642, Duration.parse("PT91H59M27S"), 12),
            new Gagnants(2011, "Australia", "Cadel Evans", "BMC Racing Team",
3430, Duration.parse("PT86H12M22S"), 2),
            new Gagnants(2012, "Great Britain", "Bradley Wiggins", "Team Sky", 3496,
Duration.parse("PT87H34M47S"), 14),
            new Gagnants(2013, "Great Britain", "Chris Froome", "Team Sky", 3404,
Duration.parse("PT83H56M20S"), 14),
            new Gagnants(2014, "Italy", "Vincenzo Nibali", "Astana", 3661,
Duration.parse("PT89H59M06S"), 19),
            new Gagnants(2015, "Great Britain", "Chris Froome", "Team Sky", 3360,
Duration.parse("PT84H46M14S"), 16),
            new Gagnants(2016, "Great Britain", "Chris Froome", "Team Sky", 3529,
Duration.parse("PT89H04M48S"), 14));

public static void main(String args[]) {

}
public double getVitesseMoyenne() {
        return (getKilometres() / (getTemps().getSeconds() / 3600));
}
public int getAnnee() {
        return annnee;
}
public void setAnnee(int year) {
        this.annnee = year;
}
public String getNationalite() {
        return nationalite;
}
public void setNationalite(String nationality) {
        this.nationalite = nationality;
}
public String getNom() {
        return nom;
}
public void setNom(String name) {
        this.nom = name;
}
public String getEquipe() {
        return equipe;
}
public void setEquipe(String team) {
        this.equipe = team;
}
public int getKilometres() {
        return kilometres;
}
public void setKilometres(int lengthKm) {
        this.kilometres = lengthKm;
```

```
	}
	public Duration getTemps() {
		return temps;
	}
	public void setTemps(Duration winningTime) {
		this.temps = winningTime;
	}
	public int getJoursenjaune() {
		return joursenjaune;
	}
	public void setJoursenjaune(int daysInYellow) {
		this.joursenjaune = daysInYellow;

	@Override
	public String toString() {
		return nom;
	}
}
```

Exercice 1

Créer une liste des noms des gagnants ayant parcouru moins de 3500km.

Vous utiliserez : stream, filter, les références de méthode avec ::, et enfin collect.

Exercice 2

Même exercice, mais avec ceux ayant parcouru 3500km et plus.

Exercice 3

Même exercice que l'exercice 1, mais en ne présentant que les deux premiers. Vous utiliserez en plus limit(2).

Exercice 4

Listez les noms distincts de tous les gagnants, quel que soit le nombre de km parcourus. Vous utiliserez distinct.

Exercice 5

Donnez le nombre de gagnants différents.

Vous utiliserez count()

Exercice 6

Vous listerez les noms de tous les gagnants, en sautant les deux premiers.

Vous utiliserez skip(2)

Exercice 7

Vous afficherez l'année suivie du nom du gagnant.

Vous utiliserez en tout stream, map et collect.

Exercice 8

Vous afficherez le nombre de caractères du nom de chaque gagnant sans filtre

Vous utiliserez deux fois map.

Exercice 9

Vous récupérerez les gagnants ayant « Wiggins » dans leur nom.

Vous utiliserez stream, filter contenant une expression lambda, contains, findAny, et vous récupérerez le résultat dans un type Optional<Gagnants>.

Exercice 10

Vous identifierez s'il y a eu un gagnant l'année 2014.

Vous utiliserez stream, map, une référence de méthode sur gerAnnee(), filter, findFirst(). Vous récupérerez le résultat dans un type Optional<Integer>, et testerez si le résultat est vide avec isPresent() sur Optional.

Exercice 11

Vous récupérerez le total des kilomètres effectués par l'ensemble des gagnants.

Vous utiliserez stream, map, reduce(0,Integer ::sum) ; vous récupérerez le résultat dans un Integer.

Exercice 12

Vous récupérez la plus courte distance effectuée par tous les gagnants, quelle que soit l'année.

Vous utiliserez stream, map, reduce(Integer ::min), vous récupérerez dans un Optional<Integer>

Exercice 13

Même question que l'exercice précédent, mais en extrayant la plus longue distance. Vous utiliserez Integer::min

Exercice 14

Vous récupérerez le gagnant qui a eu la vitesse moyenne la plus rapide. Vous avez pour cela une méthode toute prête, getVitesseMoyenne(). Vous utiliserez pour cela :

stream, min(Comparator.comparingDoucle(Gagnants ::getVitesseMoyenne)).
Vous récupérerez dans un Optional<Gagnants>

Exercice 15

Vous ferez la même chose, mais avec un code plus resserré, et en utilisant :
Stream, mapToDouble(Gagnants ::getVitesseMoyenne).min()

Exercice 16

Vous grouperez les gagnants par Nom ; en fait seul Froome aura plusieurs entrées.
L'objectif est d'avoir ce genre de sortie :

...,Chris Froome=[Chris Froome, Chris Froome,Chris Froome],...

Vous utiliserez stream,collect(groupingBy(Gagnants ::getNom)).

Vous récupérerez la liste dans Map<String,List<Gagnants>>

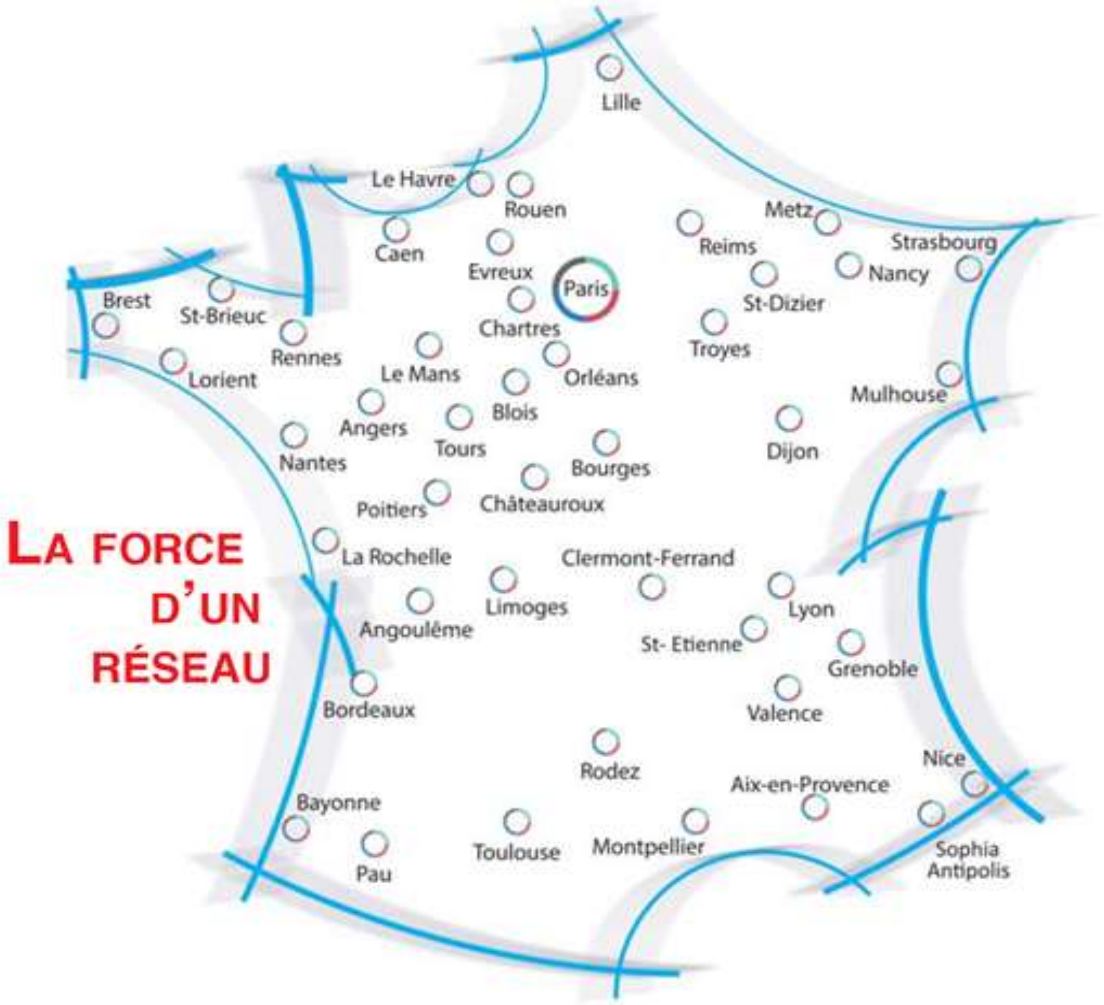
Exercice 17

Affichez les gagnants par nationalité.
Vous utiliserez :

stream, collect(groupingBy(Gagnants ::getNationalite,counting())

Formation

Notre expertise est votre avenir



Découvrez également l'ensemble des stages à votre disposition sur notre site

http://www.m2iformation.fr

N°Azur 0 810 007 689

PRIX D'UN APPEL LOCAL DEPUIS UN POSTE FIXE