Formation Java 8

TP n°2 : Les références des méthodes

SOMMAIRE

I- Prérequis :

- JDK 8.
- Eclipse.

II- Objectifs

✓ Comprendre comment utiliser une référence à une méthode statique.
✓ Comprendre comment utiliser une référence à une méthode d'une instance.
✓ Comprendre comment utiliser une référence à une méthode d'une instance arbitraire d'un type.
✓ Comprendre comment utiliser une référence à un constructeur.

III- Les Références des méthodes

Les références de méthodes permettent d'offrir une syntaxe simplifiée pour invoquer une méthode comme une expression lambda : elles offrent un raccourci syntaxique pour créer une expression lambda dont le but est d'invoquer une méthode ou un constructeur.

Une expression lambda correspond à une méthode anonyme dont le type est défini par une interface fonctionnelle. Les références de méthodes ont un rôle similaire mais au lieu de fournir une implémentation, une référence de méthode permet d'invoquer une méthode statique ou non ou un constructeur.

Les références de méthodes ou de constructeurs utilisent le nouvel opérateur ::.

Il existe quatre types de références de méthodes :

Type	Syntaxe	Exemple
Référence à une méthode statique	nomClasse::nomMethodeStatique	String::valueOf
Référence à une méthode sur une instance	objet::nomMethode	personne::toString
Référence à une méthode d'un objet arbitraire d'un type donné	nomClasse::nomMethode	Object::toString
Référence à un constructeur	nomClasse::new	Personne::new

Il y a deux possibilités pour invoquer une méthode d'une instance :

- préciser directement l'instance concernée dans la référence de méthodes
- préciser le type concernée dans la référence de méthodes : dans ce cas, l'instance sera passée en paramètre pour désigner celle qui sera invoquée

La syntaxe d'une référence de méthode est composée de trois éléments :

- un qualificateur qui précise le nom d'une classe ou d'une instance sur lequel la méthode sera invoquée
- l'opérateur ::
- un identifiant qui précise le nom de la méthode ou l'opérateur new pour désigner un constructeur

Le qualificateur peut être :

- un type pour les méthodes statiques et les constructeurs
- un type ou une expression pour les méthodes non statiques. L'expression doit préciser l'objet sur lequel la méthode est invoquée

Une référence de constructeur comprend :

- un qualificateur qui est un type dont il est possible de créer une instance : cela exclut les interfaces et les classes abstraites
- l'opérateur ::
- le mot clé new

Exemple :

```
package ma.formation.referencesmethodes;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Test1 {
	static List<Etudiant> etudiants = new ArrayList<Etudiant>();
	static Integer[] entiers = { 4, 3, 2, 5, 92, 100 };
	static {
		etudiants.add(new Etudiant("ALAMI"));
		etudiants.add(new Etudiant("Ziani"));
		etudiants.add(new Etudiant("Foukan"));
		etudiants.add(new Etudiant("Bali"));
	}

	public static void main(String[] args) {
		test1();

	}
```

```
	public static void test1() {
		etudiants.sort(Test1::comparrerParNom);

		Arrays.sort(entiers, Test1::comparrerEntiersDescendants);

		System.out.println(Arrays.deepToString(entiers));

		etudiants.forEach(System.out::println);
	}

	public static int comparrerParNom(Etudiant e1, Etudiant e2) {
		return e1.getNom().compareTo(e2.getNom());

	}

	public static int comparrerEntiersDescendants(int a, int b) {
		return b - a;

	}

}

class Etudiant {
	private String nom;

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

	public String getNom() {
		return nom;
	}

	public Etudiant(String nom) {
		super();
		this.nom = nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

}
```

Le tableau ci-dessous donne quelques exemples de références de méthodes et leurs expressions lambda équivalentes :

Type	Référence de méthode	Expression lambda
Référence à une méthode statique	System.out::println Math::pow	x -> System.out.println(x) (x, y) -> Math.pow(x, y)
Référence à une méthode sur une instance	monObject::maMethode	x -> monObject.maMethode(x)
Référence à une méthode d'un objet arbitraire d'un type donné	String::compareToIgnoreCase	(x, y) -> x.compareToIgnoreCase(y)
Référence à un constructeur	MaClasse::new	() -> new MaClasse();

Dans le cas d'une référence à une méthode statique ou d'une référence à une méthode non statique sur une instance, les paramètres définis dans l'interface fonctionnelle sont simplement passés en paramètres de la méthode invoquée.

Dans le cas d'une référence à une méthode non statique sur une classe, le premier paramètre est l'instance sur laquelle la méthode sera invoquée avec les autres paramètres qui lui sont passés.

La méthode ou le constructeur qui sera invoqué est déterminé par le compilateur selon le contexte d'utilisation. Une méthode ou un constructeur peut avoir plusieurs surcharges. Le compilateur s'appuie sur l'interface fonctionnelle utilisée dans le contexte : la surcharge invoquée sera celle dont les paramètres correspondent à ceux définis dans l'unique méthode abstraite de l'interface fonctionnelle.

a. La référence à une méthode statique

La référence à une méthode statique permet d'invoquer une méthode statique d'une classe.

La syntaxe d'utilisation est : nom_de_la_classe::nom_de_la_methode_statique

L'exemple ci-dessous invoque la méthode statique de trois manières : la version historique en utilisant une classe anonyme interne et les deux possibilités offertes par Java 8 c'est à dire une expression lambda et une référence de méthode. Les trois invocations sont rigoureusement identiques, seule la quantité de code nécessaire pour les réaliser change.

```
package ma.formation.referencesmethodes;

public class Test2 {

        public static void main(String[] args) {
```

```
            // classe anonyme interne
            new Thread(new Runnable() {
              @Override
              public void run() {
                executer();
              }
            }).start();

            // expression lambda
            new Thread(() -> executer()).start();

            // référence de méthode statique
            new Thread(Test2::executer).start();
          }
          static void executer() {
            System.out.println("execution de mon traitement par
"+Thread.currentThread().getName());
          }
}
```

```
<terminated> Test2 [Java Application] C:\Java\jdk1.8.0_121\bin\javaw.exe (20 déc. 2021 à 15:54:52)
execution de mon traitement par Thread-0
execution de mon traitement par Thread-1
execution de mon traitement par Thread-2
```

❖ L'exemple ci-dessous définit une interface fonctionnelle qui est utilisée en paramètre d'une méthode.

```
package ma.formation.referencesmethodes;

import java.util.Arrays;
import java.util.List;

public class Test3 {

        public static void main(String[] args) {
                List<String> pays = Arrays.asList("Maroc", "Egypt", "France", "Canada");
                afficher(pays, (fmt, arg) -> String.format(fmt, arg));
        }

        public static void afficher(List<String> liste, MonFormateur formateur) {
                int i = 0;
                for (String element : liste) {
                        i++;
                        System.out.print(formateur.formater("%3d %s%n", i, element));
                }
```

```
        }

        @FunctionalInterface
        interface MonFormateur {
                String formater(String format, Object... arguments);
        }
}
```

```
<terminated> Test3 (1) [Java Application] C:\Java\jdk1.8.0_121\bin\javaw.exe (20 déc. 2021 à 16:05:21)
1 Maroc
2 Egypt
3 France
4 Canada
```

A la place de l'expression lambda, il est possible d'utiliser directement une référence de méthodes static sur format() de la classe String.

```
package ma.formation.referencesmethodes;

import java.util.Arrays;
import java.util.List;

public class Test4 {
        public static void main(String[] args) {
                List<String> pays = Arrays.asList("Maroc", "Egypt", "France", "Canada");
                afficher(pays, String::format);
        }

        public static void afficher(List<String> liste, MonFormateur formateur) {
                int i = 0;
                for (String element : liste) {
                        i++;
                        System.out.print(formateur.formater("%3d %s%n", i, element));
                }
        }

        @FunctionalInterface
        interface MonFormateur {
                String formater(String format, Object... arguments);
        }
}
```

- L'exemple ci-dessous utilise une référence à la méthode statique compare() de la classe Integer pour trier un tableau en invoquant la méthode sort() de la classe Arrays. Cette méthode attend en paramètre le tableau à trier et une interface fonctionnelle de type Comparator.

```
package ma.formation.referencesmethodes;

import java.util.Arrays;

public class Test5 {

	public static void main(String[] args) {
		Integer[] valeurs = { 10, 4, 2, 7, 5, 8, 1, 9, 3, 6 };
		Arrays.sort(valeurs, Integer::compare);
		System.out.println(Arrays.deepToString(valeurs));
	}
}
```

Attention, il n'est pas possible d'appliquer cette fonctionnalité si le tableau est un tableau d'entiers de type primitif.

La signature de la méthode Compare() de la classe Integer est compatible avec l'interface fonctionnelle Comparator. Il est donc possible d'utiliser une référence statique sur cette méthode en paramètre de la méthode sort() de la classe Arrays.

De la même façon, il est possible d'utiliser une référence de méthode sur une méthode statique définie dans une classe de l'application tant que sa signature respecte celle de l'interface fonctionnelle.

```
package ma.formation.referencesmethodes;

import java.util.Arrays;

public class Test6 {
	public static void main(String[] args) {
		Integer[] valeurs = { 10, 4, 2, 7, 5, 8, 1, 9, 3, 6 };
		Arrays.sort(valeurs, Test6::comparerEntierAscendant);
		System.out.println(Arrays.deepToString(valeurs));
	}

	public static int comparerEntierAscendant(int a, int b) {
		return a - b;
	}
}
```

b. La référence à une méthode d'une instance

La référence d'une méthode d'instance permet d'offrir un raccourci syntaxique pour invoquer une méthode d'un objet.

La syntaxe est de la forme : instance::nom_de_la_methode

Où :

- instance est l'objet sur lequel la méthode est invoquée
- nom_de_la_methode est le nom de la méthode à invoquer

Dans l'exemple ci-dessous, la classe ComparaisonPersonne propose deux méthodes pour comparer deux objets de type Personne selon deux critères. La classe ci-dessous utilise une référence à une méthode d'une instance de type ComparaisonPersonne en paramètre de la méthode sort() de la classe Arrays pour trier un tableau de type Personne.

```
package ma.formation.referencesmethodes;

import java.util.Arrays;

public class Test7 {
	public static void main(String[] args) {
		Personne[] personnes = { new Personne("ALAMI", "AHMED"), new Personne("JILALI",
"SAMIR"), new Personne("Kaoutari", "Nada") };
		ComparaisonPersonne comparaisonPersonne = new ComparaisonPersonne();

		Arrays.sort(personnes, comparaisonPersonne::comparerParNom);
		System.out.println(Arrays.deepToString(personnes));

		Arrays.sort(personnes, comparaisonPersonne::comparerParPrenom);
		System.out.println(Arrays.deepToString(personnes));
	}
}

class ComparaisonPersonne {

	public int comparerParNom(Personne e1, Personne e2) {
		return e1.getNom().compareTo(e2.getNom());
	}

	public int comparerParPrenom(Personne e1, Personne e2) {
		return e1.getPrenom().compareTo(e2.getPrenom());
	}
}

class Personne {
```

```
	private String nom;
	private String prenom;

	public Personne(String nom, String prenom) {
		super();
		this.nom = nom;
		this.prenom = prenom;
	}

	@Override
	public String toString() {
		return this.nom + " " + this.prenom;
	}

	public String getNom() {
		return nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

}
```

```
<terminated> Test7 [Java Application] C:\Java\jdk1.8.0_121\bin\javaw.exe (20 déc. 2021 à 17:50:42)
[ALAMI AHMED, JILALI SAMIR, Kaoutari Nada]
[ALAMI AHMED, Kaoutari Nada, JILALI SAMIR]
```

La classe de l'instance peut utiliser des generic tant que les types restent compatibles avec le contexte d'utilisation.

```
package ma.formation.referencesmethodes;

import java.util.Arrays;

public class Test8 {
	public static void main(String[] args) {
		Personne[] personnes = { new Personne("ALAMI", "AHMED"), new Personne("JILALI",
```

```
"SAMIR"),
                                    new Personne("Kaoutari", "Nada") };
                ComparaisonPersonne2  comparaisonPersonne = new ComparaisonPersonne2();

                Arrays.sort(personnes, comparaisonPersonne::comparerParNom);
                System.out.println(Arrays.deepToString(personnes));

                Arrays.sort(personnes, comparaisonPersonne::comparerParPrenom);
                System.out.println(Arrays.deepToString(personnes));
        }
}

class ComparaisonPersonne2<T extends Personne> {

        public int comparerParNom(T p1, T p2) {
                return p1.getNom().compareTo(p2.getNom());
        }

        public int comparerParPrenom(T p1, T p2) {
                return p1.getPrenom().compareTo(p2.getPrenom());
        }
}
```

Le résultat est le même.

Le qualificateur de l'instance peut être le mot clé this pour désigner l'instance courante ou super pour désigner une référence sur la classe mère.

Ainsi this::equals est équivalent à l'expression lambda x -> this.equals(x).

c. La référence à une méthode d'une instance arbitraire d'un type

Une référence à une méthode d'instance sur un objet arbitraire d'un type permet d'invoquer une méthode d'une instance qui est précisée dans le premier paramètre fourni.

La syntaxe est de la forme : nom_de_la_classe::nom_de_la_methode_d_instance

Où :

- nom_de_la_classe est le type de l'instance
- nom_de_la_methode_d_instance est le nom de la méthode à invoquer

Ce type de référence de méthodes ne précise pas explicitement le récepteur sur lequel la méthode sera invoquée. C'est le premier paramètre de la méthode de l'interface fonctionnelle qui correspond à ce récepteur.

```java
package ma.formation.referencesmethodes;

import java.util.Arrays;

public class Test9 {
	public static void main(String[] args) {
	   String[] pays = {"Maroc", "Egypt", "france", "Belgique", "Oman"};
	   Arrays.sort(pays, String::compareToIgnoreCase);
	   System.out.println(Arrays.deepToString(pays));
	  }
}
```

```
<terminated> Test9 [Java Application] C:\Java\jdk1.8.0_121\bin\javaw.exe (20 déc. 2021 à 18:20:57)
[Belgique, Egypt, france, Maroc, Oman]
```

L'exemple ci-dessous est le même exemple dans lequel l'utilisation d'une référence de méthode est remplacée par une expression lambda.

```java
package ma.formation.referencesmethodes;

import java.util.Arrays;

public class Test10 {
	public static void main(String[] args) {
	   String[] pays = {"Maroc", "Egypt", "france", "Belgique", "Oman"};
	   Arrays.sort(pays, (s1,s2)->s1.compareToIgnoreCase(s2));
	   System.out.println(Arrays.deepToString(pays));
	  }
}
```

Le résultat est le même.

L'expression lambda correspondante à la référence de méthode de l'exemple ci-dessus attend deux paramètres de type String. Le premier sera l'instance sur laquelle la méthode est invoquée. Le second est passé en paramètre de la méthode.

Le corps de l'expression invoque la méthode compareToIgnoreCase() sur l'instance du premier paramètre en lui passant en paramètre le second.

L'exemple ci-dessous est le même exemple dans lequel l'utilisation d'une référence de méthode est remplacée par une classe anonyme interne.

```java
package ma.formation.referencesmethodes;
```

```
import java.util.Arrays;
import java.util.Comparator;

public class Test11 {
	public static void main(String[] args) {
	   String[] pays = {"Maroc", "Egypt", "france", "Belgique", "Oman"};
	   Arrays.sort(pays, new Comparator<String>() {

	        @Override
	        public int compare(String s1, String s2) {
	                return s1.compareToIgnoreCase(s2);
	        };
	        });
	   System.out.println(Arrays.deepToString(pays));
	  }
}
```

d. La référence à un constructeur

Il est possible de faire référence à un constructeur.

La syntaxe d'une référence à un constructeur est de la forme : nom_classe::new

Il est inutile de préciser la surcharge du constructeur qui sera invoquée : le compilateur la détermine selon le contexte. La surcharge utilisée sera celle dont les paramètres correspondent le mieux à ceux de la méthode de l'interface fonctionnelle.

Une référence à un constructeur peut être passée en paramètre ou assignée à une variable d'un type d'une interface fonctionnelle.

```
package ma.formation.referencesmethodes;

import java.util.function.Supplier;

public class Test12 {
	public static void main(String[] args) {
	   Supplier<Personne> supplier = Personne::new;
	   System.out.println(supplier.get());
	  }
}
```

Cet exemple est équivalent à celui ci-dessous qui utilise une expression lambda.

```
package ma.formation.referencesmethodes;
```

```
import java.util.function.Supplier;

public class Test13 {
	public static void main(String[] args) {
	   Supplier<Personne> supplier = () -> new Personne();
	   System.out.println(supplier.get());
	 }
}
```

Le tableau ci-dessous contient quelques exemples de références à un constructeur et leur équivalent sous la forme d'une expression lambda.

Référence à un constructeur	Expression lambda
Integer::new	(int valeur) -> new Integer(valeur) ou (String s) -> new Integer(s)
ArrayList<Personne>::new	() -> new ArrayList<Personne>()
String[]::new	(int size) -> new String[size]

Il est possible d'invoquer un constructeur possédant des paramètres : il faut pour cela que la méthode de l'interface fonctionnelle possède les paramètres qui correspondent à ceux du constructeur invoqué.

```
package ma.formation.referencesmethodes;

public class Test14 {
	@FunctionalInterface
	public interface PersonneSupplier {
		Personne creerInstance(String nom, String prenom);
	}
	public static void main(String[] args) {
		PersonneSupplier supplier = Personne::new;
		Personne personne = supplier.creerInstance("nom1", "prenom1");
		System.out.println(personne);
	}
}
```

Le constructeur qui sera invoqué dépend du contexte d'exécution : le compilateur va inférer les types des paramètres définis dans la méthode de l'interface fonctionnelle pour rechercher le constructeur possédant les mêmes types de paramètres.

Cet exemple est équivalent à celui ci-dessous qui utilise une expression lambda.

```
package ma.formation.referencesmethodes;

public class Test15 {

	@FunctionalInterface
	public interface PersonneSupplier {
		Personne creerInstance(String nom, String prenom);
	}

	public static void main(String[] args) {
		PersonneSupplier supplier = (nom,prenom) -> new Personne(nom,prenom);
		Personne personne = supplier.creerInstance("nom1", "prenom1");
		System.out.println(personne);
	}
}
```

Il est possible de préciser le ou les types si la classe est typée avec un generic.

```
package ma.formation.referencesmethodes;

import java.util.ArrayList;
import java.util.List;

public class Test16 {

	public static void main(String[] args) {
		MaFabrique<List<String>> fabrique = ArrayList<String>::new;
		System.out.println(fabrique.creerInstance());
	}
}

interface MaFabrique<T> {
	T creerInstance();
}
```

Il est possible d'utiliser une référence de constructeur pour un tableau.

```
package ma.formation.referencesmethodes;
public class Test17 {
	public static void main(String[] args) {
		MaFabrique2<Integer[]> fabrique = Integer[]::new;
		Integer[] entiers = fabrique.creerInstance(10);
		System.out.println("taille = " + entiers.length);
	}
}
interface MaFabrique2<T> {
	T creerInstance(int taille);
}
```

Fin du TP 2.