Chapitre 4
Les expressions lambda

1. Introduction

Les expressions lambda ont été longuement attendues dans l'écosystème Java. Elles sont apparues avec Java 8. Java a ainsi rattrapé son retard sur certains langages concurrents comme le C#. Mais que sont les expressions lambda ? Ce sont tout simplement des fonctions que l'on peut passer en paramètre d'une autre fonction. Cela permet de simplifier le code dans certaines situations là où le développement objet offre des solutions verbeuses à base de classes anonymes notamment.

2. Fonctionnement

2.1 Les interfaces fonctionnelles

Le fonctionnement des expressions lambda s'appuie sur le fonctionnement des interfaces. En effet, un paramètre de méthode attendant une expression lambda est tout simplement un paramètre du type d'une interface. **Cette interface** doit cependant respecter une règle importante : elle **ne doit définir la signature que d'une seule méthode abstraite**.

Dans ce cas de figure, l'interface devient une interface fonctionnelle. Il est toujours possible d'avoir des méthodes complémentaires si elles ne sont pas abstraites (méthodes par défaut, méthodes statiques).

Pour exemple, voici le code simplifié de l'interface `java.util.Comparator<T>`. Son rôle est de proposer un mécanisme de comparaison d'objets d'un même type :

```
@FunctionalInterface
public interface Comparator<T> {
   int compare(T o1, T o2);
  ...
}
```

Une interface fonctionnelle est une interface comme les autres dans sa structure. La différence est la présence de l'annotation `@FunctionalInterface` qui est d'ailleurs optionnelle, mais qui permet au compilateur de faire les vérifications de cohérence nécessaire. Tant que vous n'avez pas exactement une et une seule méthode abstraite dans l'interface, le compilateur indiquera le message suivant :

Java propose un ensemble d'interfaces fonctionnelles répondant à la grande majorité des situations. Il ne sera donc pas forcément fréquent d'en créer soi-même. Elles sont disponibles dans le package `java.util.function`. Une section dédiée permettra d'explorer le contenu de ce package.

Un exemple concret d'utilisation des interfaces fonctionnelles est la méthode `sort` de l'interface `List<T>`. Cette méthode permet de trier les éléments de la liste selon un critère de comparaison fourni en paramètre de la méthode sous la forme d'une implémentation de l'interface `Comparator<T>` qui, je le rappelle, permet de comparer deux objets du même type.

Voici la signature de la méthode `sort` :

```
void java.util.List.sort(Comparator<? super String> c)
```

La méthode `sort` attend un paramètre de type `Comparator<T>`. Comme c'est une interface fonctionnelle, il est possible de passer en paramètre :

– Une instance de classe implémentant cette interface :

```
List<String> liste = new ArrayList<String>(
                    List.of("bonjour", "tout", "le", "monde"));

liste.sort(new Comparator<String>() {

           @Override
           public int compare(String s1, String s2) {
               return s1.compareTo(s2);
           }
       });
```

Pour plus d'informations sur les classes anonymes, veuillez vous référer à la section éponyme dans le chapitre Programmation objet.

– Une expression lambda (ou méthode anonyme) respectant la signature de l'unique méthode abstraite de cette interface :

```
liste.sort((s1,s2)->s1.compareTo(s2));
```

Pour plus d'informations sur les règles syntaxiques, veuillez vous référer à la section suivante, Les méthodes anonymes.

– Une référence vers une variable du type de l'interface :

```
Comparator<String> comparateur = (s1, s2)->s1.compareTo(s2);
liste.sort(comparateur);
```

– Une référence vers une méthode existante respectant la signature de l'unique méthode abstraite de cette interface :

```
private static int compare(String s1, String s2)
{
   return s1.compareTo(s2);
}
```

Pour passer en paramètre une référence vers cette méthode, l'écriture est la suivante :

```
liste.sort(MaClasse::compare);
```

Pour plus d'informations sur les règles syntaxiques, veuillez vous référer à la section Les références de méthodes un peu plus loin dans ce chapitre.

Ces premiers exemples montrent qu'une expression lambda offre une syntaxe beaucoup plus concise qu'une classe anonyme. La référence de méthode permet en plus de factoriser le traitement dans une méthode réutilisable.

2.2 Les méthodes anonymes

Il existe différentes syntaxes d'écriture des méthodes anonymes en fonction de différents critères :

– Le nombre de paramètres,
– Le type de méthode : fonction ou procédure,
– Le nombre d'instructions à écrire dans la méthode anonyme.

2.2.1 Syntaxe générale

Une expression lambda a la forme générale suivante :

```
paramètre -> corps
```

Une expression lambda doit déclarer les paramètres avant de déclarer le corps de la méthode. Ces deux blocs sont forcément séparés par une flèche `->` opérateur *arrow*).

2.2.2 Déclaration des paramètres

S'il n'y a aucun paramètre, la déclaration s'écrit de la manière suivante :

```
() -> corps
```

La paire de parenthèses indique l'absence de paramètre.

S'il y a un ou plusieurs paramètres, la déclaration peut s'écrire de la manière suivante :

```
(TypeParametre nomParametre, ...) -> corps
```

Cela se fait donc exactement comme pour une méthode classique.

Les méthodes anonymes offrent cependant quelques facilités :

– Il n'est pas obligatoire de définir le type des paramètres. Le compilateur met en œuvre l'inférence de type pour deviner le type des paramètres. Il est donc possible de simplifier l'écriture précédente en ôtant les types :

```
(nomParametre, ...) -> corps
```

– S'il y a un unique paramètre, il est même possible d'ôter les parenthèses pour la déclaration des paramètres :

```
nomParametre -> corps
```

Dans ce cas là, il n'est pas possible de définir explicitement le type du paramètre.

2.2.3 Déclaration du corps

Après la déclaration des paramètres, il est nécessaire de déclarer le corps de la méthode anonyme. Pour cela, il est possible d'utiliser la même organisation qu'une méthode classique en utilisant les accolades (`{}`) comme délimiteurs :

```
... -> {
   instruction;
   instruction;
   [return ...;]
}
```

Tout comme pour les paramètres, les méthodes anonymes offrent quelques facilités pour l'écriture du corps de la méthode.

– Si la méthode ne contient qu'une seule instruction (ce qui est souvent le cas dans une méthode anonyme), alors il est possible d'enlever les accolades et le point-virgule !

```
... -> l'unique instruction sans point-virgule
```

– Si la méthode ne contient qu'une seule instruction dont le résultat doit être retourné, il n'est pas utile d'utiliser le mot-clé `return` :

```
... -> l'unique instruction retournant une valeur sans return
et point-virgule
```

2.2.4 Utilisation des variables "externes"

Une expression lambda peut accéder à des variables "externes", c'est-à-dire des variables déclarées en dehors de l'expression lambda. Il faut simplement que la variable soit déclarée `final` ou qu'elle soit effectivement `final`. Une variable est effectivement `final` si elle n'est pas déclarée `final` mais qu'elle est utilisée comme si elle était `final` dans l'expression lambda (c'est-à-dire qu'elle n'est utilisée qu'en lecture).

2.3 Les références de méthodes

La seconde manière d'exploiter les interfaces fonctionnelles est d'utiliser les références de méthodes. Il est possible d'utiliser en lieu et place d'une expression lambda une référence vers une méthode dont la signature respecte la signature de l'unique méthode abstraite de l'interface fonctionnelle. La manière de référencer peut évoluer en fonction de différentes situations décrites dans les sections suivantes.

2.3.1 Méthode d'instance

Si la méthode à référencer est une méthode d'instance, la syntaxe à utiliser est la suivante :

```
nomDeLaVariable::nomDeLaMethode
```

Si la méthode à référencer est sur l'instance en cours de manipulation, la syntaxe devient la suivante :

```
this::nomDeLaMathode
```

Les deux-points (`::`) permettent de séparer le nom de la méthode de son "propriétaire". Ils permettent aussi d'indiquer au compilateur que ce n'est pas un appel de méthode que l'on souhaite réaliser, mais simplement une référence. Notez bien aussi qu'aucun paramètre n'est fourni, les parenthèses sont absentes.

2.3.2 Méthode de classe

Si la méthode à référencer est une méthode de classe (autrement dit statique), la syntaxe diffère quelque peu :

```
NomDeLaClasse::nomDeLaMethode
```

2.3.3 Constructeur

Si la méthode à référencer est un constructeur, la syntaxe est la suivante :

```
NomDeLaClasse::new
```

Pour un tableau, il faudra utiliser la syntaxe suivante :

```
NomDeLaClasse[]::new
```

Pour une classe générique, il faudra utiliser la syntaxe suivante :

```
NomDeLaClasse<UnType>::new
```

2.4 L'API java.util.function

2.4.1 Présentation de l'API

L'API `java.util.function` contient 43 interfaces fonctionnelles et donc autant de signatures de méthodes abstraites. La plupart de ces méthodes sont génériques et s'adaptent donc en fonction du contexte dans lequel vous souhaitez les utiliser. La javadoc est disponible à l'adresse suivante : https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/function/package-summary.html

Ces 43 interfaces respectent une convention de nommage. Le suffixe permet de déterminer le rôle général de l'interface :

– `XxxConsumer` : les interfaces `XxxConsumer` permettent de déterminer la signature d'une procédure. En fonction de l'interface, la procédure accepte un ou deux paramètres. Bien sûr, elle ne retourne rien. Ce type d'interface permet d'effectuer une action. La méthode abstraite de ces interfaces s'appelle toujours `accept`.

Chapitre 2
Créer son premier projet avec NetBeans

1. Introduction

Le développement de projets informatiques répond à tout type de besoin. De la création d'un outil personnel automatisant une tâche quotidienne répétitive au développement de logiciels complexes utilisés par des centaines d'employés, l'environnement de développement intégré NetBeans peut être utilisé pour réaliser ces différents projets. Par conséquent, afin de sélectionner le projet le plus adapté au cahier des charges, il est nécessaire de découvrir les possibilités offertes par NetBeans.

Ce chapitre a pour objet de présenter les différents types de projets mis à disposition des développeurs, créer un projet Java simple, définir les bases de la programmation orientée objet à travers ce projet, présenter les vues de développement de NetBeans, développer une classe, rendre le projet exécutable, compiler le projet, l'exécuter, le documenter et proposer un exercice corrigé.

2. Les types de projets Java

Dans le monde professionnel, les projets Java sont régulièrement utilisés pour développer des applications de gestion web ou de bureau et pour réaliser des traitements manipulant un nombre important de données.

Le framework Java le plus récent régulièrement utilisé pour développer des applications de bureau est JavaFX . Sa mise en service a été réalisée pour remplacer les anciennes bibliothèques Swing et Abstract Window Toolkit (AWT). Les applications web actuelles sont fréquemment développées en deux parties distinctes : la partie interface utilisateur, en Angular, React ou Vue.js, et la partie serveur, en Java avec des micro-services Rest et les différentes bibliothèques Spring. Les traitements gérant un nombre conséquent de données sont ordinairement des projets Java utilisant la bibliothèque Springbatch.

NetBeans offre un choix important de projets ; dans un premier temps, leurs caractéristiques principales seront présentées.

Remarque

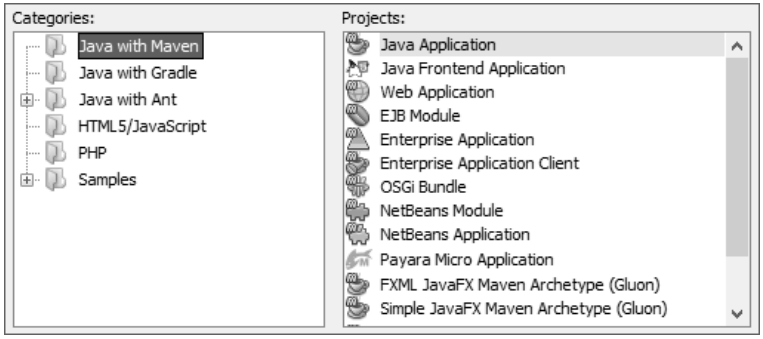
Note sur les projets NetBeans

NetBeans permet de créer un très grand panel de projets et contrairement à d'autres IDE, NetBeans force le développeur à utiliser un standard pour sa gestion de configuration.

Parmi les standards disponibles en Java, les trois majeurs sont Ant, Maven et Gradle (dans leur ordre historique d'apparition). L'utilisation de l'un de ces gestionnaires de configuration est une bonne approche, cela rend le projet indépendant de l'IDE utilisé et facilite ainsi la reprise de code par des gens habitués à ces standards.

L'installation de l'IDE terminée, à partir de l'écran d'accueil de NetBeans, effectuez la manipulation suivant.

- Cliquez sur le menu **File - New Project** : l'écran ci-dessous vous permet de sélectionner le type de projet Maven que vous souhaitez créer.



La première catégorie, *Java with Maven,* est composée de quinze types de projets différents. Maven est un outil open source d'Apache permettant de faciliter la gestion d'un projet. Il permet par exemple d'automatiser des tâches de compilation, de déploiement ou bien de gérer les dépendances entre les bibliothèques utilisées par le projet.

Type	Description
Java Application	Permet la création de programmes basiques en java. L'archive finale de ce projet est un JAR (*Java Archive*).
Java Frontend Application	Permet la création d'applications front exécutables par exemple depuis un navigateur. L'application est composée de deux parties, la partie cliente (front-end) et la partie serveur (back-end) ; les deux sont articulées par l'intermédiaire du pattern MVVM. L'archive finale de ce projet est un JAR (*Java Archive*).
Web Application	Permet la création d'applications web exécutables depuis un serveur d'applications, par exemple Apache Tomcat. L'archive finale de ce projet est un WAR (*Web application Archive*).
EJB Module (*Enterprise java-Beans*)	Permet la création de composants Java dédiés principalement à la gestion des transactions, de la sécurité. Ils sont exécutés exclusivement sur le serveur.

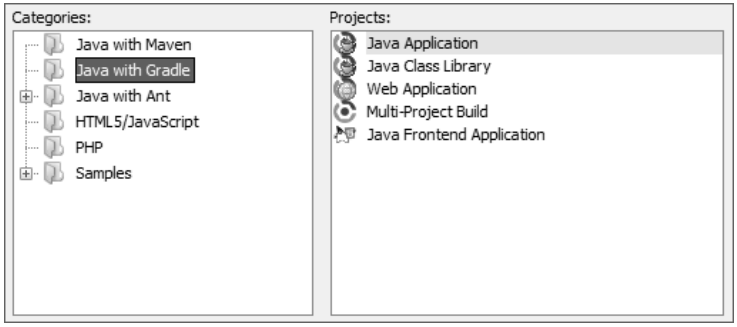
Type	Description
Enterprise application	Permet la création d'applications agrégeant des applications web et de modules EJB. L'archive finale de ce projet est un EAR *(Enterprise application Archive)*.
Enterprise application Client	Permet la création d'applications web et des services web de type *Representational state transfer* (Rest). L'archive finale de ce projet est un WAR *(Web application Archive)*.
OSGi Bundle (*Open Services Gateway initiative*)	*Framework* utilisé pour la programmation modulaire en Java. Il permet une meilleure gestion des versions et dépendances entre les composants installés dans la machine virtuelle Java (JVM).
NetBeans Module	Permet la création d'un composant d'une application basée sur la plateforme NetBeans.
NetBeans Application	Permet la création d'une application de bureau basée sur la plateforme NetBeans. Cela permet de réutiliser toute l'infrastructure applicative prévue par NetBeans (gestion des menus, des vues, des outils, des enregistrements et des notifications) sans avoir à le développer à nouveau.
Paraya Micro Application	Permet la création de plateformes utilisées pour le déploiement de micro-services Java.
FXML JavaFX Maven Archetype (Gluon)	Permet la création d'applications de bureau FXML Java. Ce type de projet permet de développer des écrans grâce à des fichiers de type fxml et de gérer leur comportement dans des classes Java.

Type	Description
Simple JavaFX MAven Archetype (Gluon))	Permet la création de projets utilisant le *framework* JavaFX. Il donne accès aux développeurs à une bibliothèque d'interface utilisateur utilisée pour créer des applications de bureau ou des applications exécutées sur une JVM et lancées dans un navigateur web.
POM Project	Permet la création d'un package contenant un fichier POM parent pouvant être ensuite utilisé pour assembler plusieurs modules.
Project from Archetype	Permet la création d'un projet Maven simple à partir d'un archétype précis, sélectionné à la création du projet.
Project with Existing POM	Permet l'ouverture d'un projet Maven à partir d'un fichier POM existant.

Tableau. Description des projets disponibles

- Cliquez sur le menu **Java with Gradle** ; l'écran ci-dessous vous permet de sélectionner le type de projet Gradle que vous souhaitez créer.

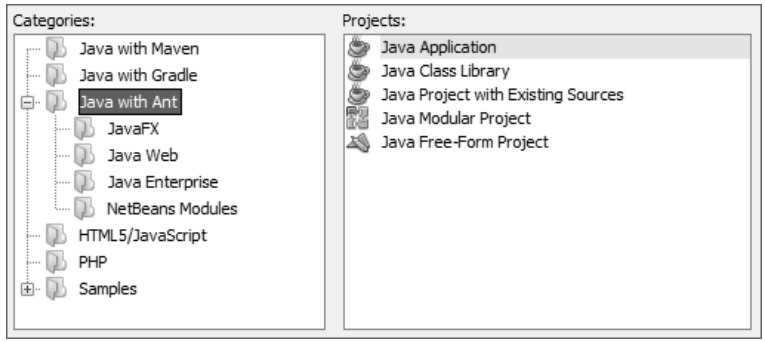
La seconde catégorie liste les projets Java avec ***Gradle***. *Gradle* est un outil de construction d'applications Java, Groovy, Scala et Java EE. Cette alternative à Maven ou Ant intègre ses deux concurrents en mixant les avantages de chacun. Il peut être intégré à une infrastructure existante afin de pallier des problèmes, par exemple de *build*, posés par Maven ou Ant.



Type	Description
Java Application	Permet la création de programmes basiques en Java avec Gradle. Sélectionner ce projet permet la création automatique d'une main java Main utilisée lors de l'exécution du projet.
Java Class Library	Permet la création d'un projet Gradle contenant une librairie Java.
Web Application	Permet la création d'un projet web Java EE avec Gradle.
Multi-Project Build	Permet la création d'un projet autorisant la construction de plusieurs sous-projets. Ainsi, chacun de ces modules peut être développé séparément puis construit grâce à un projet unique.
Java Frontend Application	Permet la création d'applications en Java, HTML, CSS et JavaScript, exécutées depuis un navigateur web.

- Cliquez sur le menu **Java with Ant** et déroulez les sous-menus ; l'écran ci-après vous permet de sélectionner le type de projet Ant que vous souhaitez créer.

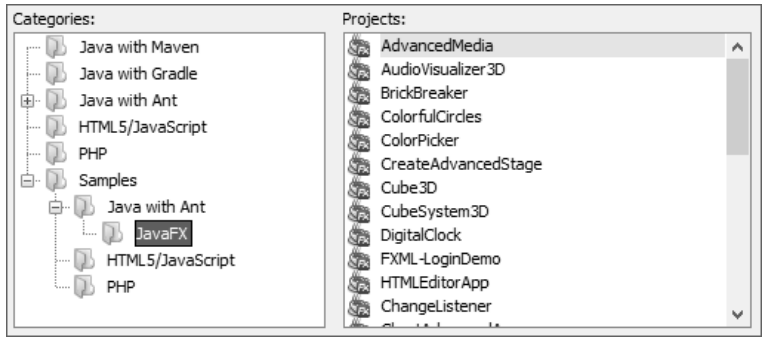
La troisième catégorie liste les projets Java avec *Another Neat Tool* (Ant). Ant est un outil de construction d'applications comme Maven ou Gradle. Il permet l'automatisation de tâches comme la compilation, la génération de documents, la création d'archives. Plus ancien que Maven, ses fonctionnalités sont moins nombreuses.



Les types de projets Java disponibles dans cette catégorie sont décrits dans les paragraphes précédents, il n'est pas nécessaire de les détailler. L'essentiel à retenir sur Ant est que, de la même manière que Maven ou Gradle, il permet la création d'applications Java basiques ou web.

Comme le montre l'image ci-dessus, l'utilisation de NetBeans pour des développements de projet HTML5/JavaScript et PHP est possible. Ces parties ne seront pas développées dans la suite de ce livre, consacré à Java avec NetBeans.

La dernière catégorie, Samples, donne accès à des exemples de projets JavaFX. Ils sont une base intéressante pour découvrir les différentes possibilités de création.



Les différents types de projets Java ont été introduits. La suite du chapitre a pour objet de présenter la création d'un projet Java basique, dont les modalités de développement et d'exécution sont les plus simples à aborder pour prendre en main NetBeans.