Refactoring to Java Streams & Lambdas

Durée 1 jour(s) (REFACTORING-TO-STREAMS)

Moderniser son "vieux" code Java pour le rendre plus facile à comprendre et maintenir

Formation officielle

Exclusivité Zenika

Description

Java a fêté ses 25 ans en 2020. Le code écrit en 1995 s'exécute toujours aujourd'hui, sans nécessiter une nouvelle phase de compilation. C'est l'une des raisons principales du succès de Java en entreprise.

Au fil des ans, de nombreuses améliorations du langage ont rendu son utilisation plus facile. Les classes membres sont apparues en Java 1.1. Le framework Collections en Java 1.2. Java 5 a amélioré les garanties apportées par le typage statique avec les paramètres génériques.

Mais la différence la plus notable pour les développeurs Java a été introduite en Java 8 avec les Streams et les Lambdas. Il est maintenant possible d'écrire du code Java dans un style déclaratif plutôt qu'impératif. Cela permet de se concentrer davantage sur l'intention (le "quoi") plutôt que la mécanique du programme (le "comment").

Depuis Java 8 le langage Java a continué d'évoluer en permanence : records, classes scellées, pattern matching, inférence de type et bien d'autres. Toutes ces fonctionnalités rendent plus facile l'écriture de code Java.

Malheureusement le code de nombreux projets est toujours écrit en se limitant aux fonctionnalités de Java 6. Il est temps de changer cela !

Selon Martin Fowler "le refactoring est une discipline qui permet de changer la structure d'un programme sans altérer son comportement

Pratique et évaluation des acquis

Cette formation est fortement orientée pratique. Chaque chapitre se termine par un exercice permettant de refactorer une partie du code d'un ERP de 300 000 lignes.

Formation disponible en Présentiel / Distanciel

La formation à distance se déroule de préférence sur des jours consécutifs (contactez nous si besoin de décomposer en demies journées)

En inter-entreprises, l'outil de visio-conférence privilégié est Microsoft Teams

En intra-entreprises, on privilégie Zoom mais Microsoft Teams est également proposé

Objectifs

- Examiner différentes méthodes de refactoring avec un focus sur les Streams
- Retracer les nouveautés du langage Java

Public

- Développeur / Développeuse Java

Prérequis

- Expérience en programmation Java

Répartition

50% Théorie, 50% Pratique

Evaluations des acquis

L'évaluation des acquis de la formation se fera en séance au travers d'ateliers, d'exercices et/ou de travaux pratiques. Dans le cas d'une formation officielle éditeur, veuillez nous consulter afin que nous vous fassions part des modalités d'évaluation.

A l'issue de la formation, vous sera transmis une évaluation à chaud de l'action de formation qui vous permettra de nous faire part de vos retours quant à votre expérience apprenant avec Zenika.

Ressources pédagogiques

Les ressources pédagogiques proviennent de productions des équipes Zenika et/ou de la documentation éditeur dans le cas d'une formation "Officielle". Les documents sont en français ou en anglais.

RQTH et ma formation Zenika

Si vous êtes sujet à un handicap, prenez contact avec nos équipes pour que nous puissions définir ensemble comment nous pourrons aménager la session afin que vous puissiez vivre une expérience en formation inchangée.

Programme

Introduction

Refactoring

Inspecter son code avec IntelliJ IDEA

Les nouveautés du langage Java

Les méthodes d'interfaces par défaut

- Exercice 1: Remplacer avec List.sort

Les méthodes d'interfaces statique

- Comparator.comparing
- Interfaces fonctionnelles

Lambdas

- Converting an Anonymous Type to Lambda Syntax
- Statement vs Expression Lambda
- Exercice: Remplacer les types anonymes par des lambda

Les références de méthode

- Exercice: Remplacer les lambda par des références de méthode

Iterable et Map forEach()

- Exercice: Remplacer loop par forEach()

removeIf()

- Exercice: Remplacer loop par removeIf()

Map Compound Methods

- Exercice: Remplacer avec Compound Map Methods

Streams

Stream.all/any/noneMatch()

- Exercice: Remplacer avec all/any/noneMatch

Stream.map() et collect()

- Exercice: Remplacer avec Map.collect()

Collectors.toCollection()

- Exercice: Remplacer avec map() et Collectors.toCollection()

Stream.filter()

- Exercice: Remplacer avec map(), filter(), collect()

Collectors.toMap()

- Exercice: Remplacer avec stream(), collect(), Collectors.toMap()

Stream.reduce()

- Exercice: Remplacer avec stream(), map(), reduce()

Stream.flatMap()

- Exercice: Remplacer avec flatMap()

Optional, findFirst(), findAny()

- Exercice: Remplacer avec findFirst() ou findAny()

groupingBy(), mapping()

- Exercice: Remplacer avec collect(), groupingBy() et mapping()

Checked Exceptions

- Handling checked exceptions with sneaky throw
- Exercice: Handling checked exceptions with ThrowingFunction

Conclusion