Nouveautés de Java 9

version sortie en octobre 2017

Bruno Mermet

Septembre 2019

Plan

- Jigsaw
- Java Shell
- Méthodes privées dans les interfaces
- «Fabriques» de collections non modifiables
- Nouvelles fonctionnalités liées au flux
  - nouvelles sources
  - nouveaux traitements
- Nouveau ramasse-miette par défaut
- Optimisation des boucles d'attente active
- Modification de la javadoc
  - modules
  - champ de recherche

Jigsaw : Introduction

But

Regrouper les packages java (standards et tiers) en modules pour :

- limiter la taille de la machine virtuelle en ne chargeant que les modules nécessaires ;
- accroître la sécurité en restreignant les visibilités inter-modules

Notion de module

Un module est caractérisé par :

- Un nom (par convention *URLinversée.nomPackage*)
- Des dépendances vis à vis d'autres modules
- Une liste de packages *exportés*
- Un fichier `module-info.java` / `module-info.class`
- Un fichier jar

Jigsaw : Quelques avantages

- Seuls les packages exportés par un module sont visibles des autres modules
  - limite les risques de conflits entre packages de même nom ;
  - plus de sécurité
- Détection préalable possible de classes non trouvées
- Optimisation possible du chargement des classes (seuls les modules requis sont chargés)

Jigsaw : structure d'un fichier `module-info.java` :

```
module nomModule {
  requires autreModule;
  requires encoreUnAutreModule;
  exports nomPackage;
  exports unAutrePackage;
}
```

Jigsaw - exemple : Arborescence

```
|- Repertoire
|    |- SourceModule
|    |      |- mermet
|    |      |      |- horloge
|    |      |      |      |- Heure.java
|    |      |      |      |- Mode.java
|    |      |      |      |- concret
|    |      |      |      |      |- Heure1.java
|    |      |      |      |      |- Heure2.java
|    |      |- module-info.java
|    |- SourceUtilisation
|    |      |- Utilisation
|    |      |      |- Main.java
|    |      |- modul-info.java
|    |- mods
|    |    |- horloge
|    |    |- utilisation
```

Jigsaw - exemple : Fichiers Mode.java et Horloge.java

```java
// Fichier Mode.java
package mermet.horloge;
public enum Mode {EFFICACITE_TEMPS, EFFICACITE_MEMOIRE}

// Fichier Horloge.java
package mermet.horloge;
import mermet.horloge.concret.Heure1;
import mermet.horloge.concret.Heure2;
public interface Heure {
    int getH();
    void setH(int h);
    int getM();
    void setM(int m);
    default String toStringDefault() {
        return String.format("%2d:%2d", getH(), getM());
    }
    static Heure getHeure(int h, int m, Mode mode) {
        if (mode == Mode.EFFICACITE_TEMPS) {return new Heure1(h,m);}
        else {return new Heure2(h,m);}
    }
}
```

Jigsaw - exemple : Fichier Heure1.java

```
package mermet.horloge.concret;
import mermet.horloge.Heure;
public class Heure1 implements Heure {
    private int h;
    private int m;

    public Heure1(int mh, int mm) {
        h = mh; m = mm;
    }
    @Override
    public int getH() {return h;}
    @Override
    public void setH(int h) {this.h = h;}
    @Override
    public int getM() {return m;}
    @Override
    public void setM(int m) {this.m = m;}
    @Override
    public String toString() {
        return toStringDefault();
    }
}
```

Jigsaw - exemple : Fichier Heure2.java

```
package mermet.horloge.concret;
import mermet.horloge.Heure;
public class Heure2 implements Heure {
  private short minutes;
  public Heure2(int h, int m) {
    minutes = (short) (60*(h%24) + (m%60));
  }
  @Override
  public int getH() {return minutes/60;}
  @Override
  public void setH(int h) {minutes = (short) (getM() + h*60);}
  @Override
  public int getM() {return minutes % 60;}
  @Override
  public void setM(int m) {minutes = (short) (getH()*60+getM());}
  @Override
  public String toString() {
    return toStringDefault();
  }
}
```

Jigsaw - exemple : Fichier Main.java

```java
package utilisation;
import mermet.horloge.Heure;
import mermet.horloge.Mode;
//la ligne suivante génère une erreur : type non accessible
//import mermet.horloge.concret.Heure2;

public class Main {
  public static void main(String[] args) {
    Heure h1 = Heure.getHeure(12, 34, Mode.EFFICACITE_MEMOIRE);
    System.out.println(h1);
    System.out.println(h1.getClass());
    Heure h2 = Heure.getHeure(12, 34, Mode.EFFICACITE_TEMPS);
    System.out.println(h2);
    System.out.println(h2.getClass());
    //la ligne suivante nécessite le troisième import
    //Heure h3 = new Heure2(12,34);
  }
}
```

Jigsaw - exemple : Fichiers `module-info.java`

Dans répertoire SourceModule

```
module horloge {
  exports mermet.horloge;
}
```

Dans répertoire SourceUtilisation

```
module utilisation {
  requires horloge;
}
```

Jigsaw - exemple : compilation du module `horloge`

(depuis le répertoire `Repertoire/SourceModule`)

Commande

```
javac -d ../mods/horloge
      module-info.java
      edu/mermet/horloge/*.java
      edu/mermet/horloge/classes/*.java
```

Résultat : Contenu du répertoire mods/horloge

```
mods
 |- horloge
 |     |- mermet
 |     |      |- horloge
 |     |      |      |- Heure.class
 |     |      |      |- Mode.class
 |     |      |      |- concret
 |     |      |      |      |- Heure1.class
 |     |      |      |      |- Heure2.class
 |     |- module-info.class
```

Jigsaw - exemple : compilation du module `utilisation`

(depuis le répertoire `Repertoire/SourceUtilisation`)

Commande

```
javac -d ../mods/utilisation/
      module-info.java
      utilisation/Main.java
      --module-path ../mods/
```

Résultat : Contenu du répertoire mods/utilisation

```
mods
 |- utilisation
 |     |-utilisation
 |     |      |- Main.class
 |     |- module-info.class
```

Jigsaw - exemple : compilation - détail des options utilisées

- `-d` : répertoire où seront placés les fichiers `.class`
- `--module-path` : répertoire(s) où trouver les modules requis
- *fichiers à compiler* : mettre les fichiers `.java` **et** le fichier `module-info.java`

Jigsaw - exemple : exécution

(depuis le répertoire `Repertoire`)

```
java --module-path mods utilisation.Main
```

Jigsaw - conclusion à partir de l'exemple

Les classes du package `mermet.horloge.concret` sont publiques, donc entièrement accessibles depuis tous les packages du module `horloge`. Mais comme le package en question n'est pas exporté, ces classes sont inaccessibles depuis les autres modules.

Jigsaw et Eclipse (photon)

- Créer un projet pour le module `horloge`
  - Préciser le nom du module (`horloge`) dans la deuxième boîte de dialogue
  - Remplir le fichier `module-info.java` ouvert automatiquement
  - Développer les classes du module
- Créer un projet pour le module `utilisation`
  - Dans la boîte de dialogue précisant les répertoires du projet :
    - cliquer sur l'onglet `Projects`
    - cliquer sur l'item `modulepath`
    - cliquer sur bouton `Add...`
    - sélectionner le premier projet
  - Définir le nom du module
  - Compléter le fichier `module-info.java` ouvert automatiquement
  - Développer les classes du module

Java shell

Interpréteur REPL (Read-Eval Process Loop) `jshell` :

- Complétion automatique
- Point-virgule facultatif
- Imports par défaut configurables
- Historique
- ...

Méthodes privées dans les interfaces

Il est maintenant possible de mettre des méthodes statiques privées dans les interfaces. Ceci permet de faire figurer dans les interfaces des méthodes utilitaires n'ayant pas à être accessible hors de l'interface.

Fabriques de collections non modifiables

Les différentes interfaces de collection fournissent des méthodes statiques permettant de créer des collections non modifiables :

- `static <E> List<E> of(...)` dans l'interface `List`
- `static <E> Set<E> of(...)` dans l'interface `Set`
- `static <K, V> Map<K,V> of(...)` dans l'interface `Map`

Remarque : ces méthodes existent pour un nombre fixe de paramètres (de 1 à 10) et pour un nombre variable de paramètres.

Flux : nouvelles sources de données (1)

- Classe Scanner
  - `Stream<String> tokens()`
  - `Stream<MatchResult> findAll(String motif)`
  - `Stream<MatchResult> findAll(Pattern motif)`
- Classe Matcher (objets renvoyés par la méthode `matcher()` de la classe `Pattern`)
  - `Stream<MatchResult> results()`

N.B. : un `MatchResult` est un objet représentant le texte trouvé, sa position de début et sa position de fin.

Flux : nouvelles sources de données (2)

- Classe `Optional`

La méthode `stream()` renvoie un flux à 1 ou 0 éléments suivant que l'objet de type `Optional` contient un élément ou pas.

- Interface `Stream`
  - La méthode statique `Stream<T> ofNullable(T donnée)` renvoie un flux à 0 ou 1 élément, suivant que `donnée` est à `null` ou pas
  - La méthode statique `Stream<T> iterate(T graine, Predicate<? super T> continuer, UnaryOperator<T> suivant)` génère un flux à partir de l'opérateur `suivant` tant que `continuer` est vrai

Flux : nouveaux traitements

- `default Stream<T> takeWhile(Predicate<? super T> predicat)` : ne conserve que les premiers éléments d'un flux, tant qu'ils vérifient le prédicat `predicat`
- `default Stream<T> takeWhile(Predicate<? super T> predicat)` : supprime les premiers éléments d'un flux, tant qu'ils vérifient le prédicat `predicat`

Nouveau ramasse-miette par défaut

le ramasse-miettes `G1` existait déjà dans la plate-forme Java. C'est un ramasse-miettes qui peut fonctionner en multi-thread. Avec Java 9, il devient le ramasse-miettes par défaut.

Optimisation des boucles d'attente active

- Avant :

```
class EventHandler {
  volatile boolean evenementRecu;
  void agirApresEvt() {
while (!evenementRecu) {
  evenementRecu = evaluerEvt();
}
agir();
  }
  void agir() {...}
}
```

- Après :

```
class EventHandler {
  volatile boolean evenementRecu;
  void agirApresEvt() {
while (!evenementRecu) {
  evenementRecu = evaluerEvt();
  java.lang.Thread.onSpinWait();
}
agir();
  }
  void agir() {...}
}
```

- Intérêt Une machine virtuelle peut éventuellement en tirer partie pour optimiser la gestion du CPU

Modification de la javadoc

- Intégration de la notion de module

Dans la version *avec frame*, le cadre en haut à gauche contient maintenant au première niveau la liste des modules. Il faut sélectionner un module pour avoir la liste des packages qu'il renferme.

- Intégration d'un champ de recherche

En haut à droite du cadre de droite, un champ de recherche permet de rechercher un mot dans toute la javadoc.