Évolution de Java
JDK 1.1, Java SE 5 et Java SE 8

Barry Alpha Volte--Vieira Philippe
Université de Paris

M2S2 2022

I Naissance du projet Java

Java est un langage orienté objet qui fut pensé par rapport au C++, par et pour une équipe de Sun Microsystems travaillant sur système embarqué au début des années 1990. Le C++ avait été publié en 1985, et étendait le C qui, lui, avait été publié au début des années 1970.

Les créateurs de Java, Patrick Naughton, James Gosling et Mike Sheridan, avaient donc plusieurs objectifs, notamment pour corriger les problèmes que leur posaient le C++ :

— intégrer un gestionnaire de ressources automatique (*garbage collector*),
— permettre une large portabilité,
— faciliter le multi-threading,
— avoir de bonnes performances et une exécution sécurisée.

Moins de 5 ans après, en mai 1995, le langage fut publié et présenté publiquement avec son nom définitif pour la première fois (version Beta).

Pour permettre la portabilité, les programmes Java sont lancés sur la machine virtuelle Java (JVM). Ainsi le développement du langage Java s'est accompagné du développement de cette machine, avec beaucoup d'améliorations jusqu'à la création d'autres machines. Ce sujet est très riche et a déjà été traité dans d'autres exposés, nous n'en parlerons donc pas ou peu.

II Versions

Beaucoup de notations différentes sont utilisées pour les différentes versions de Java suivant les politiques menées par le groupe maintenant le langage. Comme nous ne parlerons que de l'édition standard nous nous permettons de les simplifier au besoin : nous pourrons noter Java 1 au lieu de Java 1.1, Java 5 au lieu de J2SE 5.0 etc.

Notez que chaque version modifie la précédente, principalement en ajoutant des fonctionnalités. Nous ne comparons donc pas tout à fait différents langages mais différentes étapes de l'évolution d'un langage.

II.1 Java 1

Java 1.0 contient déjà les principales fonctionnalités du langage comme les classes abstraites, les interfaces, les exceptions, et la synchronisation de méthodes avec `synchronized`[9]. On y trouve aussi les principales librairies `java.lang`, `java.util` et `java.io` (bien moins fournies qu'aujourd'hui).

Java 1.1, publié en 1997, introduit entre autres les classes internes et anonymes, permettant un code plus concis[8]. La sérialisation et la réflexion sont aussi ajoutées, elles permettent respectivement d'encoder des objets (puis les sauvegarder par exemple) et d'obtenir un objet `Class` donnant l'accès aux signatures des méthodes de la classe (pour parcourir le nom des méthodes, les types demandés, renvoyés etc.).

II.2 Java 5

Java 5 (ou Java 1.5) est l'une des versions de java la plus importantes en terme de nombre de fonctionnalités ajoutées et changements du langage. Elle a été publiée en 2004, et introduit de nouvelles notations de versions.

Les types énumérés

Lors de l'écriture de nos programmes informatiques, on a parfois besoin de représenter des variables qui ont des valeurs qui doivent rester dans un ensemble fini et prédéfini. C'est le cas par exemple des valeurs et des couleurs des cartes dans un jeu de cartes, ou des quatre points cardinaux.

La solution avant l'introduction de la structure `enum` était d'utiliser les types existants de java. Le *int enum pattern* par exemple consistait à définir des constantes entières. Cette solution présente beaucoup de défauts dont la fragilité du code et l'absence de sécurité du typage. Les `enum` apportent une meilleure solution pour définir des types énumérés.

La Généricité

L'une des justifications de l'introduction de la Généricité est liée aux collections. Dans les versions antérieures à java 5, il était possible de mettre des objets de différents types dans une même collection. Pour récupérer un élément, il fallait donc faire des *casts explicites*, avec le risque d'avoir des exceptions (`ClassCastException`) lors de l'exécution. Avec les types génériques nous informons au compilateur quels sont les types acceptés par la collection ce qui permet de détecter la plupart de ces erreurs dès la compilation.

La généricité permet aussi de définir des méthodes génériques et donc d'implémenter des algorithmes génériques.

Les annotations

Les annotations sont des métadonnées que l'on attache à une classe, interface, méthode ou même à un attribut d'une classe. Elles ont été introduites pour donner des informations supplémentaires au compilateur ou à la JVM. Une annotation, intégrée, souvent utilisé est `@override` qui indique au compilateur qu'on est en train de redéfinir une méthode d'une classe parente (avec la même signature). Si jamais la signature de la méthode ne correspond pas à une méthode d'une classe parente on aura une erreur à la compilation, aidant la maintenance du code.

En plus des annotations intégrées il est possible de définir ses propres annotations, ce qui a permis la mise en place de librairies travaillant sur ces annotations (ce qui peut rappeler les extensions de langage vues plus tôt dans un autre exposé).

La boucle for-each

Il s'agit d'une extension de la boucle for traditionnelle. L'usage des indices dans une boucle peut souvent être source d'erreurs. La boucle *for-each* fournit une syntaxe simple pour parcourir les éléments d'un tableau ou d'une collection.

```
1 List<String> strings = Arrays.asList("prog", "comp", "2022");
2 for(String s: strings){
3   System.out.println(s);
4 }
```

Varargs

varargs permet d'écrire des méthodes à nombre variables d'argument. C'est une solution plus jolie que l'usage de tableaux pour contenir les valeurs à passer à la méthode ou à la surcharge de méthode qui peut conduire à une grosse quantité de code difficile à maintenir.

```java
// methode à nombre variable d'argument
// retourne la somme des entiers passés en paramètre
int sum(int... args){
  return Arrays.stream(args).reduce(0, Integer::sum);
}
```

Static import

`import static` permet d'accéder directement à des membres `static` d'une classe sans avoir à les qualifier par le nom de la classe.

```java
import static java.util.Arrays.asList;
// pas besoin de mettre Arrays.asList
List<String> strings = asList("prog", "comp", "2022");
```

Autoboxing et unboxing

Permet de faire le cast implicite des types primitifs vers les types englobants correspondants (boxing) et aussi l'opération inverse (unboxing). Ainsi on peut passer à une méthode qui attend comme paramètre un type primitif `int` son type englobant correspondant `Integer` et vice versa.

II.3 Java 7 et 8

Java 7 et 8 furent publiées en 2011 et 2014. Java 8 est la première version LTS de Java, et Java 7 la première version publiée après le rachat de Sun Microsystems par Oracle 5 ans plus tôt. Java 8 fut longtemps la version de Java la plus répandue, et reste encore aujourd'hui très utilisée.

Java 7

Le mot-clé `switch` permettant une disjonction de cas sur les nombres, fonctionne aussi depuis Java 7 sur les chaînes de caractères. Cette version apporte aussi de l'inférence de type générique lors de leur création et une gestion plus concise des exceptions.

Lambdas

Avant l'arrivée des clôtures, si nous voulions utiliser des fonctions génériques pour, par exemple, trier un tableau, nous devions utiliser un objet d'une classe comportant la fonction de comparaison. Depuis Java 8, les expressions lambdas permettent une concision encore plus importante.

```java
Comparator<Integer> mycomp = new Comparator<>(){
@Override
int compare(Integer o1, Integer o2)
{ return (o1-o2)%5; }
}
// devient
Comparator<Integer> mycomp = (o1, o2) -> (o1-o2)%5;
```

De plus une nouvelle notation a fait son arrivée permettant d'imiter les variables de premier ordre, comme si on passait des fonctions en paramètres ou dans un objet. En pratique, les signatures des méthodes prennent toujours des objets en paramètre, on peut alors considérer les lambdas comme étant du sucre syntaxique.

```
mylist.forEach(System.out::println);
// Applique la méthode println de System.out sur les éléments de mylist.
// Équivalent à
Consumer<String> strprint = System.out::println;
mylist.forEach(strprint);
```

L'arrivée des expressions lambdas dans Java a lieu 3 ans après leur arrivée dans C++ (C++11), mais cette notion théorisée dans les années 1930 par A. Church avait déjà été implémentée en Lisp plus d'un demi-siècle avant ces deux langages, au début des années 1960.

Autres apports

Java 8 apporte d'autres changements, comme la systématisation du lancement de JavaFX, une refonte de l'API des dates et calendrier, et diverses améliorations de performances.

II.4 Après java 8

L'ajout de nouvelle fonctionnalités ne s'est pas arrêté à la version 8 de java. Aujourd'hui nous sommes à la version 18 et la liste des fonctionnalités ajoutées est longue :

- nouveaux GC,
- message d'erreur plus précis pour `NullPointerException` à l'exécution,
- inférence de type pour les variables locales avec `var`,
- `sealed class`, `record` ...

Java 8 et java 11 restent les versions les plus utilisées en production comme le montre ce rapport de Snyk faites en 2021 [10].

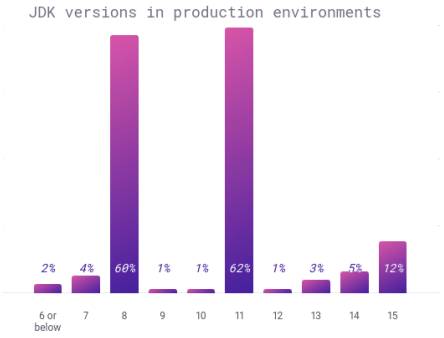

FIGURE 1 – Utilisation des JDK

III Orientations et communauté

A partir de la version 1.4 de java l'évolution de la plateforme est guidée par la JSR (Java Spécification Request). Il s'agit d'un document ouvert qui contient les fonctionnalités à ajouter ou à modifier, proposées et révisées, par les membres de la JCP (Java Community Process).

Mais depuis que Oracle est maître de la plateforme beaucoup de choses ont changé. La cadence de sortie de nouvelles versions est plus importante. Depuis java 9 une version à support standard sort tous les 6 mois, et une à support long terme tous les 3 ans. Le changement de licence Oracle est aussi devenu un sujet brûlant dans la communauté et a soulevé des inquiétudes au-près de celle-ci.

Les derniers ajouts, notamment avec les `sealed class` et les améliorations du `switch` montrent une certaine influence d'autres langages comme Scala et une volonté d'implémenter du *pattern matching*. Java a notamment nommé *pattern matching* le cast automatique dans des conditionnelles `if (s instanceof T)`, qui est bien un filtrage par motif mais bien plus pauvre que pour d'autres langages comme Scala ou OCaml permettant d'utiliser plusieurs motifs.

Bibliographie

[1] Documentation java 5. https://www.oracle.com/technical-resources/articles/java/java-5-features3.html.

[2] Documentation java 7. https://docs.oracle.com/javase/7/docs/technotes/guides/language/.

[3] Java performance. https://en.wikipedia.org/wiki/Java_performance.

[4] Java (programming language). https://en.wikipedia.org/wiki/Java_(programming_language).

[5] Java (software platform). https://en.wikipedia.org/wiki/Java_(software_platform)#History.

[6] Java version hystory. https://en.wikipedia.org/wiki/Java_version_history.

[7] Les expressions lambdas. https://www.jmdoudoux.fr/java/dej/chap-lambdas.htm.

[8] Addison-Wesley. The java programming language. https://www.cs.cornell.edu/andru/javaspec/1.1Update.html, 1997.

[9] Sun Microsystems. Java Language Specification. https://www.cs.cornell.edu/andru/javaspec/1.doc.html, août 1996. HTML généré par Doug Kramer.

[10] Snyk. JVM Ecosystem Report 2021. https://snyk.io/jvm-ecosystem-report-2021/?utm_campaign=JVM-SC-2021&utm_medium=Report-Link#developers-use-se-11.