Les nouveautés du JDK 1.8

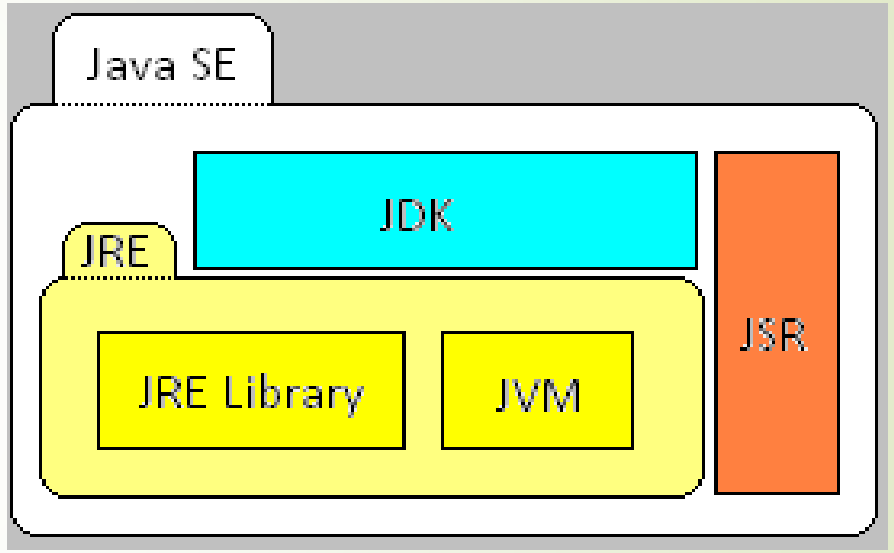
Cédric PINTO – IR3

Plan

- Introduction
  - Java SE
  - Version JDK
- Les nouveautés
- Conclusion

Introduction

Java SE (Standard Edition)



Version JDK

- 1.0 (1995)
- 1.1 (1997)
- 1.2 (1999)
- 1.3 (2001)
- 1.4 (2002)
- 1.5 (2004)
- 1.6 (2006)
- 1.7 (2011)
- 1.8 (2014)

Les principales nouveautés

- Lambdas
- Traits
- java.util. function
- java.util. Stream
- java.time

Les lambdas : pourquoi ?

- Ajout de programmation fonctionnelle
- Effacer le côté verbeux de JAVA
- Pression des langages « alternatifs » ? (Groovy, Scala, C#,…)

Les lambdas

```java
public class InnerClass {
    public Thread uneThread() {
        return new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Ma Thread");
            }
        });
    }
}
```

```java
public class InnerClass {
    public Thread uneThreadPlusMieux() {
        return new Thread(() -> System.out.println("Ma lambda"));
    }
}
```

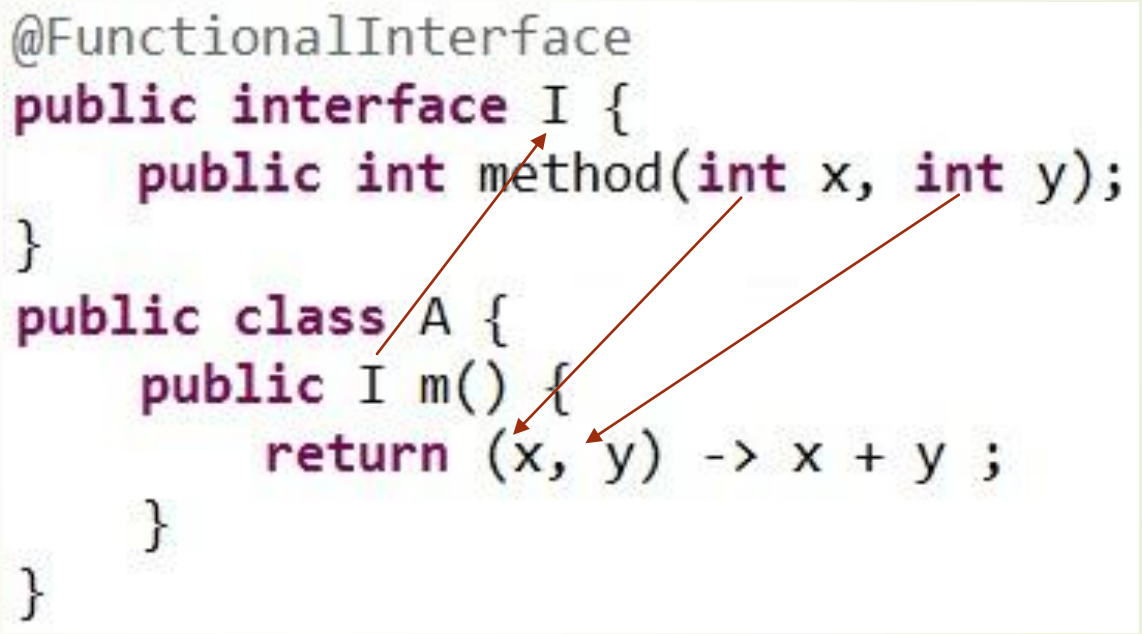
Les lambdas: Syntaxe

```
() -> System.out.println("Une ligne de code")

() -> {
    System.out.println("Une lambda");
    System.out.println("sur plusieurs lignes")
}

x -> x + 1

(x, y) -> x + y ;
```

```
(int x,int  y) -> x + y
```

Les lambdas: Types inférés

```
@FunctionalInterface
public interface I {
    public int method(int x, int y);
}
public class A {
    public I m() {
        return (x, y) -> x + y ;
    }
}
```



Les lambdas: Sémantique

```java
@FunctionalInterface
public interface I {
    public int method(int x, int y);
}
public class A {
    public I m(int x, int y) {
        return () -> x + y ;
    }
}
```

```java
public class A {
    private int x;
    public I m() {
        return () -> this.x;
    }
}
```

```java
public class A {
    public I m2() {
        return new I() {
            private int x;
            @Override
            public int method() {
                return this.x;
            }
        };
    }
}
```

Les lambdas: Références de méthodes

```java
@FunctionalInterface
public interface I {
    public int method();
}
```

```java
public class B {
    public int m() {
        return 0;
    }
}
```

```java
public class C {
    public static int m() {
        return 0;
    }
}
```

```java
public class A {
    public I m() {
        B b = new B();
        return () -> b.m();
    }
}
```

```java
public class A {
    public I m() {
        B b = new B();
        return b::m;
    }
}
```

```java
public class A {
    public I m() {
        return () -> C.m();
    }
}
```

```java
public class A {
    public I m() {
        return C::m;
    }
}
```

Lambdas & Collection

Horizon des solutions :

- Ajout de méthodes statiques dans Collections
- Ajout d'interfaces spécifiques aux lambdas (Set2, List2, …)

Solution retenue :

- Ajout de méthodes dans les interfaces (traits)

Traits

```
@FunctionalInterface
public interface Iterable<T> {
    Iterator<T> iterator();

    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
}
```

```
for (Integer x : list) {
    System.out.println(x);
}
```

```
list.forEach(x -> System.out.println(x));
```

java.util.stream

- Opérations fonctionnelles
- Séquentiel / Parallèle (stream.sequential() / stream.parallel())
- Intermédiaire / Terminale

⚠️ Ne pas paralléliser quand ce n'est pas nécessaire !

java.util.stream

```java
public class Article {
    private double prix;
    private String name;
    private Double promo;
}
```

```java
double sum = 0;
for (Article article : list) {
    if(article.getPromo() != null)
        sum += article.getPromo();
}
System.out.println(sum);
```

```java
System.out.println(list.stream().filter(x -> x.getPromo() != null)
.mapToDouble(x -> x.getPromo())
.sum());
```

java.util.function

- Function → Prend un T en paramètre, renvoi un R.
- Predicate → Prend un T en paramètre, renvoi un boolean
- Consumer → Prend un T en paramètre, renvoi void
- Supplier → Prend pas de paramètre, renvoi un T
- BinaryOperation → Prend deux T en paramètre, renvoi un T

java.time

- Deux conceptions du temps :
  - Temps machine (entier augmentant depuis le 1er janvier 1970)
  - Temps humain. (plusieurs champs, jour – mois – année,…)

- 4 principes architecturaux :
  - Immuable et Thread Safe
  - Chaînable
  - Clarté
  - Extensibilité

java.time.Instant : Temps Machine

- Instant.EPOCH → 1970-01-01T00:00:00Z
- Instant.MIN → -1000000000-01-01T00:00:00Z
- Instant.MAX → +1000000000-12-31T23:59:59.999999999Z
- Instant.now(); → 2014-01-15T20:40:27.093Z
- D'autres méthodes (isAfter(),isBefore(),parse(),…)

java.time.Duration : Temps Machine

- Duration.ZERO //représente 0.
- .plusDays(long x) // ajoute x jours à la durée (plusHours, plusNanos,…)
- .minusDays(long x) // retire x jours à la durée (minusHours, minusNanos,…)

java.time : Temps Humain

- LocalDate.MIN // -999999999-01-01
- LocalDate.MAX // +999999999-12-31
- LocalDate.now() // 2014-01-15

- LocalTime.now() // 22:09:46.553
- LocalTime.NOON // 12:00
- LocalTime.MIDNIGHT // 00:00

- LocalDateTime.now() // 2014-01-15T22:10:49.852

- D'autres méthodes (minusX, plusX,parse, getMonth,…)

Conclusion

- Pourquoi j'ai choisi ce langage pour mon exposé ?

Sources

- http://www.infoq.com/fr/news/2013/09/everything-about-java-8
- http://ttux.net/post/java-8-new-features-release-performance-code/
- http://thecodersbreakfast.net/index.php?post/2012/05/30/Java-8-et-les-Lambda
- http://blog.soat.fr/2012/12/java-8-jsr-335-33-impacts-sur-lapi-collection-et-utilisation-de-la-jdk-8-early-access/
- http://blog.soat.fr/2013/06/java-8-whats-new-23-date-and-time/

Fin

- Throw new FinDexposéException(‘’Avez-vous des questions ?’’);