Java 8

Java User Group – Zielona Góra

Paweł Żalejko

@pzalejko

rec Global

Agenda

- Java – short history
- What's New in JDK 8
  - JavaFX, Nashorn
  - Annotations
  - Default methods
  - Lambda expressions
  - Method References
  - Functional programming & Stream API
  - ...

Java - history

- 1.0 (January 23, 1996)
- 1.1 (February 19, 1997)
- 1.2 (December 8, 1998)
- 1.3 (May 8, 2000)
- 1.4 (February 6, 2002)
- 5 (September 30, 2004)
- 6 (December 11, 2006)
- 7 (July 28, 2011)
- **8 (March 18, 2014)**
- 9 (2016?)

Java 8 – Plan B

There's not a moment to lose!

Mark Reinhold's Blog

It's time for ... Plan B

2010/09/20 16:42:59 -07:00

In my previous entry I described two plausible plans for moving forward with JDK 7:

Plan A:	JDK 7 (as currently defined)	Mid 2012
Plan B:	JDK 7 (minus Lambda, Jigsaw, and part of Coin)	Mid 2011
	JDK 8 (Lambda, Jigsaw, the rest of Coin, ++)	Late 2012

Thanks to everyone who responded to that entry, both directly and indirectly. The voluminous feedback was strongly—though not universally—in favor of Plan B. As of today that is the plan of record for JDK 7 and JDK 8.

- http://openjdk.java.net/projects/jdk8/

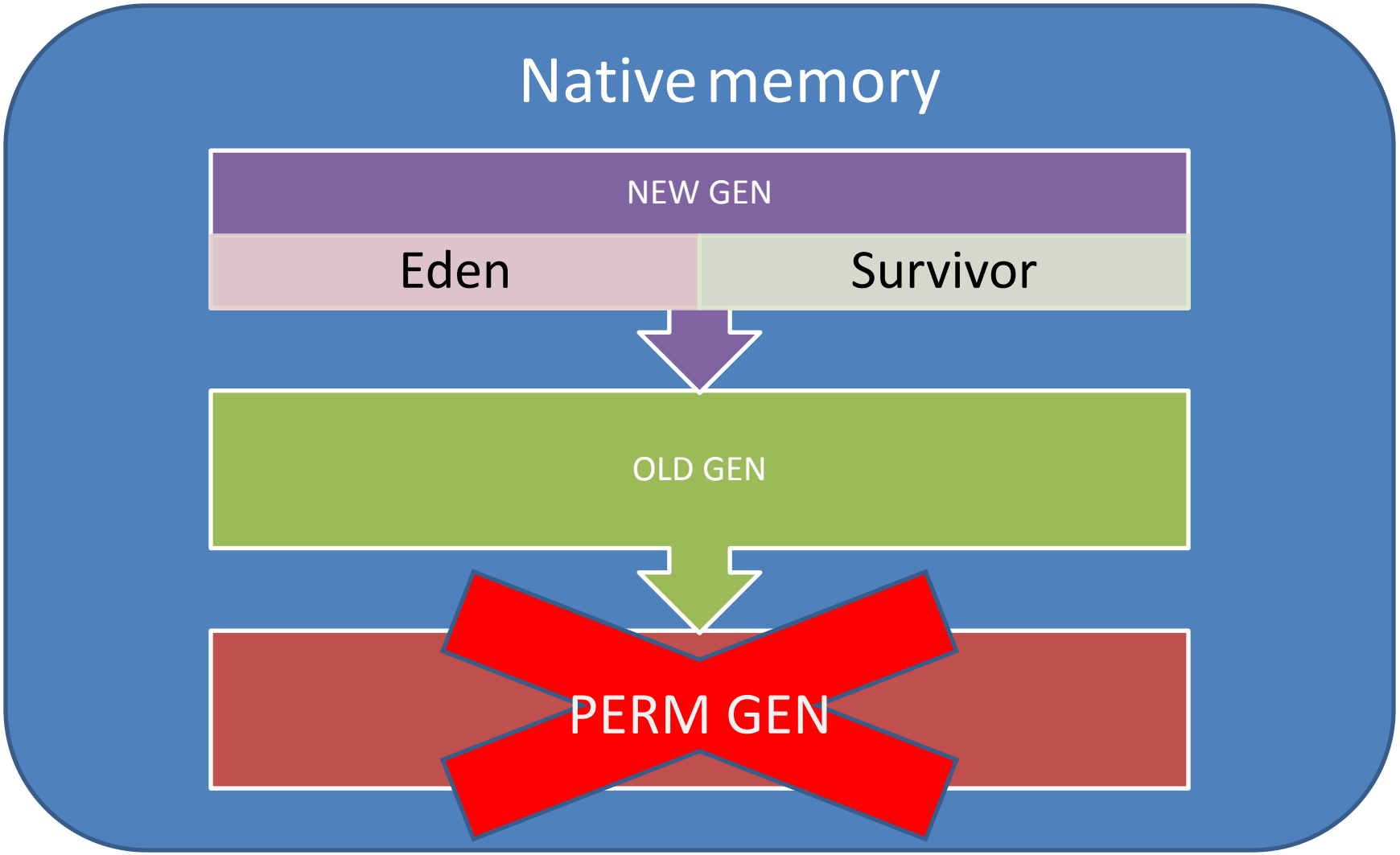
Java 8

Java 8 update 31

Java 8 update 40(March 2015)

http://openjdk.java.net/projects/jdk8u/releases/8u40.html

Metaspace



Compact Profiles

- Three profiles:
  - *Compact1*
  - *Compact2*
  - *Compact3*

- Compact profiles address API choices only
- The full SE API is a superset of the *compact3* profile

javac -profile <profile>

- **JEP 161: Compact Profiles:** http://openjdk.java.net/jeps/161

Compact Profiles

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact3

javax.xml.crypto.dsig.spec

Class XPathType

java.lang.Object

javax.xml.crypto.dsig.spec.XPathType

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.util

Interface Comparator<T>

JEP (JDK Enhancement Proposal)

java.util.Base64

(http://openjdk.java.net/jeps/135)

Nashorn

- JavaScript Engine for the JVM
- Can be used as:
  - a command line tool(jjs.exe)
  - an embedded interpreter in Java applications
- Java code can call JavaScript code, and vice versa
- Nashorn uses the invokedynamic JVM instruction

JavaFX

- HTML5 improvements via WebView (including the use of Nashorn)
- New controls, such as TreeTableView and DatePicker
- The new Modena GUI theme
- Support for Touch-Enabled Devices
  - Multi-touch support for tablets
  - Virtual Keyboard

- JavaFX 8 (update 40) finally includes simple Dialogs and Alerts!

Annoations(JSR 308)

- **Type Annotations**
  - *TYPE_USE*
  - *TYPE_PARAMETER* (e.g. MyClass<T>)

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Target;

@Target({ ElementType.TYPE_USE, ElementType.TYPE_PARAMETER })
public @interface Demo {

}
```

Repeating Annotations

```
@Foo("a")
@Foo("b")
public void doSomething() {

}
```

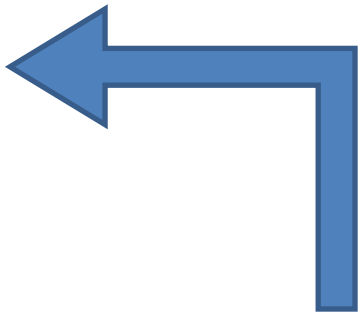
Demo

Nashorn & Annotations

Lambda expressions(JSR 335)

```
final Runnable r = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello!");
    }
};

r.run();
```



Functional Interface

```
Runnable r = () -> System.out.println("Hello!");
r.run();
```

Lambda expressions(JSR 335)

- Closures, anonymous methods/functions
- Function that takes input parameters and produces a result
- **<u>Are not anonymous classes</u>**
- Invoked as a new method(*invokedynamic*)
- Cannot throw checked exceptions
- Local variables must be final or effectively final
- Functional interfaces

- Syntax of Lambda Expressions:

(params) -> statement

Lambda expressions: under the hood

```java
import java.util.Comparator;

public class Example0 {
    Comparator<Object> comparator = new Comparator<Object>() {
        @Override
        public int compare(final Object o1, final Object o2) {
            return 0;
        }
    };
}
```

- javap -c -v Example0

```
 0: aload_0
 1: invokespecial #12 // Method java/lang/Object."<init>":()V
 4: aload_0
 5: new           #14 // class jug/zg/java8/lambda/anonymousClass/Example0$1
 8: dup
 9: aload_0
10: invokespecial #16 // Method jug/zg/java8/lambda/anonymousClass/Example0$1."
13: putfield      #19 // Field comparator:Ljava/util/Comparator;
16: return
```

Lambda expressions: under the hood(2)

```java
import java.util.Comparator;

public class Example1 {
    Comparator<Object> comparator = (o1, o2) -> 0;
}
```

- javap -c -v Example1

```
 0: aload_0
 1: invokespecial #12     // Method java/lang/Object."<init>":()V
 4: aload_0
 5: invokedynamic #17,  0 // InvokeDynamic #0:compare:()Ljava/util/Comparator;
10: putfield      #18     // Field comparator:Ljava/util/Comparator;
13: return
```

Lambda expressions: runtime

Debug | Type Hierarchy

- jug.zg.java8.lambda.anonymousClass.Demo2 at localhost:51629
  - Thread [main] (Suspended (breakpoint at line 12 in Demo2))
    - Demo2.lambda$0(String) line: 12
    - 48612937.accept(Object) line: not available
    - Arrays$ArrayList<E>.forEach(Consumer<? super E>) line: 3880
    - Demo2.main(String[]) line: 12
  - C:\Program Files\Java\jdk1.8.0_20\bin\javaw.exe (23 gru 2014, 13:42:27)

Demo2.java

```
4  package jug.zg.java8.lambda.anonymousClass;
5
6  import java.util.Arrays;
7  import java.util.List;
8
9  public class Demo2 {
10     public static void main(final String[] args) {
11         final List<String> list = Arrays.asList("a", "b");
12         list.forEach(i -> System.out.println(i));
13     }
14 }
```

Method References(JSR 335, part C)

- Lambda expressions for existing methods

Kind	Example
Reference to a static method	**Person::staticMethodName**
Reference to an instance method of a particular object	**person::getAge**
Reference to an instance method of an arbitrary object of a particular type	**String::compareToIgnoreCase**
Reference to a constructor	**Person::new**

Default methods(JSR 335, part H)

- Allow for adding new methods to existing interfaces
- Default and static methods do not break the functional interface contract
- Can be dangerous(multiple inheritance?)

- Examples(java.util.Collection):

Collection.class

```
577      * @return a sequential {@code Stream} over the elemen
578      * @since 1.8
579      */
580     default Stream<E> stream() {
581         return StreamSupport.stream(spliterator(), false);
582     }
```

Demo

Lambdas & Method References & Default methods

Functional programming

- **@FunctionalInterface**
  - *„Functional interface has exactly one abstract method."*

- **Functional interfaces in Java 8:**
  - Optional<T>
  - Supplier<T>
  - Consumer<T>
  - Function<T, R>
  - BiFunction<T, U, R>
  - Predicate<T>
  - ...

Stream API

- *„A sequence of elements supporting sequential and parallel aggregate operations"*

- Streams are not:
  - Collections
  - Sequences
  - Iterators

- Stream is an abstraction over objects
  - *„a query on the stream source"*

Stream API

- *java.util.stream.Stream* Interface

- Streams can be obtained in a number of ways:
  - *stream()* method
  - *Arrays.stream(Object[]);*
  - *Stream.of(Object[]);*
  - *java.nio.file.Files, java.io.BufferedReader etc.*

Stream API

- A stream consists of:
  - zero or more *intermediate* operations
  - a *terminal* operation

Streams are lazy: computation on the source data is only performed when the terminal operation is initiated.

- **Example:**

```
List<Integer> list = Arrays.asList(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
list.stream()
    .filter(i -> i > 5)
    .forEach(System.out::print);
```

Parallel Streams

- Fork/Join framework
- Parallel streams use a common ForkJoinPool
- Long-running tasks should be avoided
- Parallel streams can be slower than *sequential* streams

Demo

Functional programming & Streams

What else?

- **Concurrency**
  - java.util .Arrays.parallelSort(...) methods
  - StampedLock, LongAdder, CompletableFuture etc.
- **A new Date and Time API**
- **Java ME 8 Device Access API**
  - GPIO (General Purpose Input Output) pins
  - I2C (Inter-Integrated Circuit Bus)
- **Tools**
  - Jdeps: Command-line Static Dependency Checker

Thanks!