IBM Software

IBM Java 8: What's coming next?

Ian Partridge
Java Technology Center, UK
i.partridge@uk.ibm.com

Disclaimer

- © Copyright IBM Corporation 2013. All rights reserved.
- U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

Agenda

- History
- Java 8
  - What's out
  - What's in
  - The future
- IBM Java 8 beta program
  - How to join
  - Changes so far
  - New features

History

- J2SE 1.4 *2002*
  - Regular expressions, JAXP (XML and XSLT), Web Start
- J2SE 5.0 *2004*
  - Generics, autoboxing, varargs, for-each loop
- Java SE 6 *2006*
  - Pluggable annotations, Swing UI remodel, JVMTI and JPDA enhancements, JAXB, JAX-WS
- Java SE 7 *2011*
  - Project Coin, fork/join framework, NIO2, invokedynamic
- Java SE 8 *2013*
  - *???*

What's NOT included

Project Jigsaw

Complex effort to:

1) Introduce a module system into Java
- Fix “JAR hell”
- Retire the classpath
- Evolve classloaders

2) Modularise the Java Platform itself
- API and implementation dependencies

All while maintaining compatibility with existing code, JARs, OSGi, maven etc.

Project Jigsaw

Mark Reinhold (Oracle) proposed deferring to Java SE 9 last year
- http://mreinhold.org/blog/late-for-the-train-qa
- Strong majority of EG agreed

- "Project Jigsaw started at Sun, way back in August 2008. Like many efforts during the final years of Sun, it was not well staffed. Jigsaw initially ran on a shoestring, with just a handful of mostly part-time engineers, so progress was slow. During the integration of Sun into Oracle all work on Jigsaw was halted for a time, but it was eventually resumed after a thorough consideration of the alternatives. Project Jigsaw was really only fully staffed about a year ago" - Mark Reinhold, August 2012

- "I therefore propose to defer the addition of a module system and the modularization of the Platform to Java SE 9. This is by no means a pleasant choice, but I think it's preferable to delaying Java SE 8 until the modularity work is complete." - Mark Reinhold, September 2012

What IS included?

JSR 337 defines the Java SE 8 platform – umbrella JSR

- JSR 308 – Annotations on Java Types
- JSR 310 – Date and Time API
- JSR 335 – Lambda Expressions for the Java Language
- Misc. JSRs
  - Updates to concurrency APIs, new Base64 APIs, Javadoc APIs
- Other maintenance JSRs
  - Unicode, JDBC, JAXP, JAXB, JAX-WS

- http://openjdk.java.net/projects/jdk8/features has a good list

- Let's look at those 3 main JSRs...

Project Lambda

Adding lambda expressions to Java

- Goal: evolve Java language to support functional-style “code as data” programming models
- Goal: enable parallel multi-core efficient libraries (leveraging fork/join from Java 7)
- Goal: keep the Java language relevant

Iteration – today

```
for (Account a : accounts) {
        if (a.currentBalance() < 0) {
                a.debit(20);  // profit!
        }
}
```

Is syntactic sugar for...

```
Iterator i = accounts.iterator();
while (i.hasNext()) {
        Account a = i.next();
        if (a.currentBalance() < 0) {
                a.debit(20);  // profit!
        }
}
```

Iteration – today

- Problem:
- Inherently sequential
  - Fixed iteration from beginning to end
  - Logic is fixed
- What do we really want?
  - Make the classlibraries deal with iteration
  - Pass a collection, and the operation to perform
  - Get the result back
- *The operation to perform*
  - How do you pass an “operation to perform” in Java?

```
for (Account a : accounts) {
    if (a.currentBalance() < 0) {
        a.debit(20);  // profit!
    }
}
```

Anonymous inner classes

```
public interface Invoker<T> {
        void invoke(T t);
}

accounts.forEach(new Invoker<Account>() {
        public void invoke(Account a) {
                if (a.currentBalance < 0) {
                        a.debit(20);
                }
        }
});
```

`Collections.forEach()` implemented by the classlibraries
- could use Iterators, or something else

Lambdas in Java

- Language designers decided to build on anonymous inner classes to implement lambdas
- Design principle: **pave the cowpaths**
- Interfaces which only define one method are given a special name:
  - Functional Interfaces (aka SAM-types (Single Abstract Methods))
- Programmer does not need to declare their interface as “functional”, it is inferred by the compiler

Functional Interfaces in Java 7

- Java 7 already contains plenty of these interfaces:

```
java.lang.Runnable
java.util.concurrent.Callable
java.security.PrivilegedAction
java.util.Comparator
java.io.FileFilter
java.nio.file.PathMatcher
java.lang.reflect.InvocationHandler
java.beans.PropertyChangeListener
java.awt.event.ActionListener
javax.swing.event.ChangeListener
```

Lambda expressions

- Lambda expressions are a new language feature which replace the bulkiness of anonymous inner classes with a more elegant approach
- Examples:

```
(int x) -> x*x

() -> System.out.println("Hello world");

(String s) -> System.out.println("Hello " + s);
```

Converting our example

```
for (Account a : accounts) {
        if (a.currentBalance() < 0) {
                a.debit(20);  // profit!
        }
}
```

Becomes...

```
accounts.stream().forEach(
        (Account a) -> { if (a.currentBalance < 0) a.debit(20); }
);
```

We can do better

```
accounts.stream().forEach(
        (Account a) -> { if (a.currentBalance < 0) a.debit(20); }
);
```

Could be...

```
accounts.stream().forEach(
        a -> { if (a.currentBalance < 0) a.debit(20); }
);
```

Thanks to a clever compiler.

Can it be clearer though?

Filtering and mapping

```
accounts.stream()
        .filter(a -> a.currentBalance < 0)
        .forEach(a -> { a.debit(20); });
```

```
List<Account> l = accounts.stream()
                        .filter(a -> a.surname() == "Branson")
                        .filter(a -> a.currentBalance > 1000000)
                        .into(new ArrayList<Account>());
```

```
int sum = accounts.stream()
                .map(a -> a.currentBalance())
                .sum();
```

Laziness

Filter and map operations can be **eager or lazy**

- **Eager** – filtering is complete when filter() returns
- **Lazy** – filtering is only done on-demand

- Stream operations which produce new streams lend themselves to lazy implementations
  - For example:

```
accounts.stream().filter(a -> a.surname == "Smith").findFirst();
```

- Operations like accumulation, or which save results to a new Collection are naturally eager
  - For example:

```
int sum = accounts.stream().map(a -> a.currentBalance()).sum();
```

Streams

```
package java.util.stream
public interface Stream<T>
```

- A sequence of elements supporting sequential and parallel bulk operations. Streams support lazy transformative operations (transforming a stream to another stream) such as **filter** and **map**, and consuming operations, such as **forEach**, **findFirst**, and **iterator**. Once an operation has been performed on a stream, it is considered *consumed* and no longer usable for other operations.

- *Streams are not data structures; they do not manage the storage for their elements, nor do they support access to individual elements. However, you can use the iterator() or spliterator() operations to perform a controlled traversal.*

What can Streams do?

Method	Description
`allMatch`	Return true if all elements of the stream match the predicate
`anyMatch`	Return true if any element of the stream matches the predicate
`filter`	Return a stream containing the subset of the elements matching the predicate
`findFirst`	Return the first element matching the predicate
`flatMap`	Return a stream where each element is transforming into 0 or more values
`forEach`	Perform an operation on each element (usually destructively)
`limit`	Return a stream containing no more than maxSize elements
`map`	Transform the stream into another, applying the given function to each element
`max/min`	Return the max/min element based on the supplied Comparator
`reduce`	Reduce the stream to a single value, performing the Reducer operation on each element
`sorted`	Sort the stream based on natural order or supplied Comparator
`toArray`	Convert stream to array

Evolving the Java Collections Framework

- If Java had lambdas from day 1, the Collections API would look very different
- How can the Collections take advantage of lambda expressions?
- Could start again – **Collections II !**
  - Major task
  - Developers would hate it
- Instead, Java 8 evolves existing interfaces like `Collection`, `Map` and `Iterable`

Evolving Interfaces

Evolving interfaces today is very difficult

- Say you have:

```
public interface MyInterface {
    void someMethod();
}
```

- and want to add...

```
public interface MyInterface {
    void someMethod();
    void anotherMethod();
}
```

Upgrading existing code works fine *until you recompile the implementors!*

Default methods

- **Default methods** are another new language feature – designed to allow interfaces to evolve
  - Most importantly, evolving the existing Collections classes
- They allow developers to add *default behaviour* to an interface

```
public interface MyInterface {
    void someMethod();

    default void anotherMethod() {
        System.out.println("Another method");
    }
}
```

- Existing implementers who do not implement anotherMethod() will use the default implementation
- New implementors can override anotherMethod() if they want

Default methods

- This is a big change to the inheritance model of Java
- Interfaces have never contained implementations before
- It's not multiple inheritance though:
  - Java already has multiple inheritance of *types* (you can implement multiple interfaces in the same class)
  - Default methods add multiple inheritance of *behaviour*
  - It does not add multiple inheritance of *state* (like C++ has)
    - You cannot add variables to interfaces

Method resolution

- What does this print?

```
public interface A {
        default void run() { System.out.println("Interface A"); }
}

public interface B extends A {
        default void run() { System.out.println("Interface B"); }
}

public class C implements A, B {
        public static void main(String[] args) {
                new C().run();
        }
}
```

Answer: `"Interface B"`

Method resolution

Here are the rules:

- Classes always win. A declaration in the class or superclass beats any default method in an interface
- Otherwise, the *most specific default-providing interface* is chosen.
  - In the example before, this is B

Conflicts can still occur:

```
public interface A {
    default void run() { System.out.println("Interface A"); }
}

public interface B {
    default void run() { System.out.println("Interface B"); }
}

public class C implements A, B {
    public static void main(String[] args) {
        new C().run();
    }
}
```

Method resolution

Here's the javac error message:

```
class C inherits unrelated defaults for run() from types A
and B
reference to run is ambiguous, both method run() in A and
method run() in B match.
```

Wow, that's actually quite helpful!

Note that all this method resolution occurs at **compile time**

New Date and Time API – JSR 310

Current APIs have been around since forever

- System.currentTimeMillis()
  - Time in milliseconds from 1 January 1970
- java.util.Date
  - Most methods are now deprecated
- java.util.GregorianCalendar
  - Handles time offsets like “one week earlier”

- Working with these classes is difficult
  - They are widely disliked

Joda time

- Provides a quality replacement for the JDK date and time classes
  - http://joda-time.sourceforge.net/
- Easy API
  - It has getYear() instead of Calendar.get(Calendar.YEAR)!
- Supports multiple calendar systems out of the box
- Interoperates with JDK classes
  - System.currentTimeMillis()
- More predictable performance
  - System classes recalculate fields unexpectedly
- Open Source (ASL 2.0)

Designing the new Date/Time API

- As a long-running project, Joda time has encountered and found solutions for many subtle date/time issues
- Joda time is the inspiration for the new Java date/time classes
  - Same developers are involved in Joda time and JSR 310
- Fixes some architectural issues in Joda time as well
- New project is open source as well
  - http://threeten.sourceforge.net/
  - https://github.com/ThreeTen/threeten
- Javadoc is extensive
  - http://download.java.net/jdk8/docs/api/java/time/package-summary.html

Understanding the new API

- The new API is defined in the `java.time` package
- Most usecases are solved by these classes:

`Instant`
  - A numeric timestamp, stored with nanosecond resolution. Useful for capturing a point in time, similar to System.currentTimeMillis(). Instant is the closest equivalent class to java.util.Date. The instant when printed looks like '2000-12-01T12:30:00.000Z'.

`LocalDate`
  - A date without a time, offset or time zone. Useful for storing a birthday for example. The date when printed looks like '2000-12-01'

`LocalTime`
  - A time without a date, offset or time zone. Useful for storing store hours for example. The time when printed looks like '12:30:00.000'

`LocalDateTime`
  - A date and time without the offset or time zone. The date and time when printed looks like '2000-12-01T12:30:00.000'

`ZonedDateTime`
  - A date and time with offset and time zone. Useful for performing calculations that takes into account the time zone like 'America/New_York'. ZonedDateTime is the closest equivalent class to java.util.GregorianCalendar. The date and time when printed looks like '2000-12-01T12:30:00.000-05:00[America/New_York]'

Type Annotations – JSR 308

- Extends the annotations available in Java 7 from just **declarations** to **any use of a type**
- Before:

```
@Deprecated
public class OldClass {
    ...
    @Override
    String toString() { ... }
}
```

Type Annotations

Java 8 permits annotating any use of a type:

```
@Untainted String trustedString;

List<@NonNull String> strList;

myGraph = (@Immutable Graph) tmpGraph;

class UnmodifiableList<T> implements
    @Readonly List<@Readonly T> {}
```

Type Annotations

- Programmers can use type annotations to write better, more self-documenting code
- Compile-time tools can detect and prevent more errors
  - Null pointer errors
  - Unexpected side effects
  - Incorrect equality checks
- The Checker Framework is an open source tool which implements many error detectors
  - http://types.cs.washington.edu/checker-framework/
- Some of these checkers will be included in the Java 8 `javac`

Example

A simple null pointer bug

```
public class GetStarted {
    void sample() {
        @NonNull Object ref = new Object();
    }
}
```

Compile with:

```
javac -processor NullnessChecker GetStarted.java
```

Now modify this line:

```
@NonNull Object ref = null;
```

to get a helpful compilation error

The Future?

No-one really knows (even Oracle), but...

- Java SE 9 and 10
  - Project Jigsaw!
  - Multi-tenancy?
  - Better multi-language support?
  - Hypervisor-awareness?
  - Full 64-bit addressability??
  - Self-tuning JVMs???
  - Removing primitive types???
- “Java is not the new Cobol” - Oracle

IBM Java 8 beta program

- Getting early feedback from IBM customers and business partners about IBM Java 8

Objectives

- IBM
  - Obtain feedback on the new features in the upcoming release
  - Use feedback to influence our development effort
- Customers
  - Early product knowledge and experience
  - Opportunity to shape future directions
- Win – win

Deliverables

- IBM is providing:
  - Beta code for you to download and install
  - Draft documentation
  - Support via a developerWorks Community on a best-effort basis
- Supported platforms:
  - AIX
    - 32-bit IBM POWER
    - 64-bit IBM POWER
  - Linux
    - 31-bit IBM System z
    - 64-bit IBM System z
    - 32-bit IBM POWER
    - 64-bit IBM POWER
    - 32-bit x86
    - 64-bit AMD64/EM64T
  - z/OS
    - 31-bit IBM System z
    - 64-bit IBM System z

How to join

https://ibm.biz/BdxPpH

developerWorks Technical topics Evaluation software Community Events

Public Groups My Groups Help This Group

IBM SDK, Java Technology Edition Version 8 Leave G

Overview
Members
Message Board
Feeds
Bookmarks
Files

We're upgrading the developerWorks community applications to the latest version of IBM Connections in the first half of year 2013.
Get a glimpse of the improvements to come!

Overview

Welcome to the IBM SDK for Java 8.0 Beta Program

If you'd like to take part in the IBM Java 8 beta just use the url below to access the files

https://www.ibm.com/services/forms/preLogin.do?source=swg-beta-ibmjte

This new version of Java aims to provide Java SE 8.0 compatibility, while exploiting the unique capabilities of IBM platforms to achieve performance and usability improvements.

New in IBM SDK, Java Technology Edition, Version 8.0:
- Compatibility with the new Java SE 8.0 (JSR 337)
- Leveraging new IBM hardware
- Improved performance for workload optimized runtimes
- Enhanced support for Cloud & Multi-tenancy environments
- Improved efficiency of manipulating native data records/types directly from Java code

Development process

- Oracle is developing Java 8 concurrently with IBM
  - http://openjdk.java.net/projects/jdk8/
- Oracle's plan targets GA in September
  - IBM's GA will follow as soon as possible
- IBM betas are not necessarily based directly upon Oracle milestone builds
  - For example, our Beta 1 corresponds to a level between Oracle M4 and M5
  - Plus additional changes!

Current limitations

- Java plugin, Applet view and WebStart
  - These are current unavailable
- Security limitations
  - Kerberos
  - hwkeytool on z/OS
  - IBMPKCS11Impl provider
- Java Communications API
  - Not currently available
- Uninstallation leaves some files behind on Linux
- Documentation LaunchPad utility
  - Not yet updated for Java 8

IBM changes

Removal of legacy and deprecated functionality

- JRIO on z/OS
  - Use JZOS record I/O instead
- Annotation Processing Tool (apt) and com.sun APIs
  - Use Pluggable Annotation Processing API instead (available since Java 6)

IBM serviceability enhancements

- Enhancements to JVM dump API
- Specify dump filename at runtime:

```
com/ibm/jvm/Dump.javaDumpToFile(fileName)
com/ibm/jvm/Dump.heapDumpToFile(fileName)
com/ibm/jvm/Dump.systemDumpToFile(fileName)
com/ibm/jvm/Dump.snapDumpToFile(fileName)
```

- Setting and querying of dump options at runtime:

```
com/ibm/jvm/Dump.setDumpOptions(options)
com/ibm/jvm/Dump.queryDumpOptions()
com/ibm/jvm/Dump.resetDumpOptions()
com/ibm/jvm/Dump.triggerDump(options)
```

IBM serviceability enhancements

New information available in javacore.txt files

- Hypervisor information:

```
2CISYSINFO      Hypervisor name = PowerVM
```

- Supported hypervisors:
  - KVM
  - VMWare
  - PowerVM
  - Hyper-V
  - z/VM
  - PR/SM

IBM serviceability enhancements

- Include PID information in the body of javacore.txt files
  - Previously only available in filename, but that was easily lost via renaming or custom -Xdump settings

```
0SECTION                ENVINFO subcomponent dump routine
NULL                    ================================
1CIJAVAVERSION JRE 1.8.0 Linux amd64-64 build
nonproduction_20121211_131477 (pxa6480-20121205_01 )
...
1CIRUNNINGAS      Running as a standalone JVM
1CIPROCESSID      Process ID: 5306 (0x14BA)
1CICMDLINE
/java/ras/vm/jvmxa6480_ibuild_131477/jre/bin/java
-Xdump:java:events=vmstop -version
```