Following are the features newly added in Java 8 - 

* Language-level support for lambda expressions; unofficially under Project Lambda
* Default methods (virtual extension methods) which make multiple inheritance possible in Java.
* a JavaScript runtime which allows developers to embed JavaScript code within applications.
* Annotation on Java Types.
* Unsigned Integer Arithmetic.
* Repeating annotations.
* Date and Time API.
* Statically-linked JNI libraries.
* Launch JavaFx applications.

Java 8 is packed full of some really exciting features at both the JVM and language level. While some of the features initially envisioned for this release got scoped out or pushed out to release 9, there are literally dozens of new features. Many of the new additions are under-the-hood improvements either at the compiler, JVM or help-system level. As such, while we may benefit from them, there’s nothing we need to actively do (other than install Java 8 of course) to enjoy them.

Having said that, let’s look at 5 features that we feel are an absolute must for you to know about:

#### 1. ****Lambda expressions****

Even if we really didn’t want to go mainstream here, there’s little doubt that from a developer’s perspective, the most dominant feature of Java 8 is the new support for Lambda expressions. This addition to the language brings Java to the forefront of functional programming, right there with other functional JVM-based languages such as Scala and Clojure.

We’ve previously looked into how [Java implemented Lambda expressions](http://blog.takipi.com/compiling-lambda-expressions-scala-vs-java-8/?utm_term=in-post-txt-1&utm_content=java8change), and how it compared to the approach taken by Scala. From Java’s perspective this is by far one of the biggest additions to the language in the past decade.

At minimum, it’s recommended you become familiar with the Lambda syntax, especially as it relates to array and collection operations, where Lambdas have been tightly integrated into the core language libraries. It is highly likely that you’ll start seeing more and more code like the snippet below in both 3rd party and within your organization’s code.

[?](http://blog.takipi.com/5-features-in-java-8-that-will-change-how-you-code/)

|  |  |
| --- | --- |
| 1  2 | Map<Person.Sex, List<Person>> byGender =  roster.stream().collect(Collectors.groupingBy(Person::getGender)); |

\* A pretty efficient way of grouping a collection by the value of a specific class field.

#### 2. ****Parallel operations****

With the addition of Lambda expressions to arrays operations, Java introduced a key concept into the language of internal iteration. Essentially as developers we’re used to use loop operations as one of the most basic programming idioms, right up there with if and else.

The introduction of Lambda expressions turned that paradigm around, with the actual iteration over a collection on which a Lambda function is applied now carried out by the core library itself (i.e. internal iteration).

You can think of this as an extension of iterators where the actual operation of extracting the next item from a collection on which to operate is carried out by an iterator. An exciting possibility opened by this design pattern is to enable operations carried out on long arrays such as sorting, filtering and mapping to be carried out in parallel by the framework. When dealing with server code that’s processing lengthy collections on a continuous basis, this can lead to major throughput improvements with relatively little work from your end.

Here’s the same snippet as above, but using the [framework’s](http://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html) new parallel processing capabilities –

[?](http://blog.takipi.com/5-features-in-java-8-that-will-change-how-you-code/)

|  |  |
| --- | --- |
| 1  2  3 | ConcurrentMap<Person.Sex, List<Person>> byGender =  roster.parallelStream().collect(  Collectors.groupingByConcurrent(Person::getGender)); |

\* It’s a fairly small change that’s required to make this algorithm run on multiple threads.

Find the Crap in Your Java App  
  
[Show me how >>](http://land.takipi.com/developers/)

[](http://land.takipi.com/developers/)

#### 3. ****Java + JavaScript =**** ❤

Java 8 is looking to right one of its biggest historical wrongs – the ever growing distance between Java and JavaScript, one that has only increased in the past few years. With this new release, Java 8 is introducing a completely new JVM JavaScript engine – Nashorn. This engine makes unique use of some of the [new features introduced in Java 7](http://blog.takipi.com/2014/02/10/java-8-compiling-lambda-expressions-in-the-new-nashorn-js-engine/) such as invokeDynamic to provide JVM-level speed to JavaScript execution right there with the likes of V8 and SpiderMonkey.

This means that the next time you’re looking to integrate JS into your backend, instead of setting up a node.js instance, you can simply use the JVM to execute the code. The added bonus here is the ability to have seamless interoperability between your Java and JavaScript code in-process, without having to use various IPC/RPC methods to bridge the gap.

#### 4. ****New date / time APIs****

The complexity of the current native Java library API has been a cause of pain for Java developers for many years. Joda time has been filling this vacuum for years now, and with Java 8. An immediate [question that arose](http://blog.joda.org/2009/11/why-jsr-310-isn-joda-time_4941.html) early on was why didn’t Java 8 adopt Joda as its native time framework. Due to what was perceived as a design flaw in Joda, Java 8 implemented its own new date / time API from scratch. The good news is that unlike Calendar.getInstance(), the [new API](http://java.dzone.com/articles/introducing-new-date-and-time)s were designed with simplicity in mind, and clear operations to operate on manipulated values in both human readable and machine time formats.

#### 5. ****Concurrent accumulators****

One of the most common scenarios in concurrent programming is updating of numeric counters accessed by multiple threads. There have been many idioms to do this over the years, starting from synchronized blocks (which introduce a high level of contention), to read/write locks to AtomicInteger(s). While the last ones are more efficient, as they rely directly on processor CAS instructions, they require a higher degree of familiarity to implement the required semantics correctly.

With Java 8 this problem is solved at the framework level with new concurrent[accumulator classes](http://download.java.net/jdk8/docs/api/java/util/concurrent/atomic/LongAccumulator.html) that enable you to very efficiently increase / decrease the value of a counter in a thread safe manner. This is really a case where it’s not a question of taste, or preference – using these new classes in your code is really a no-brainer.

|  |  |
| --- | --- |
| **Java 6** | **Java 7** |
| Features added:  - Support for older win9x versions dropped.  - Scripting lang support: Generic API for integration with scripting languages, & built-in mozilla javascript rhino integration  - Dramatic performance improvements for the core platform, and swing.  - Improved web service support through JAX-WS JDBC 4.0 support  - Java compiler API: an API allowing a java program to select and invoke a java compiler programmatically.  - Upgrade of JAXB to version 2.0: including integration of a stax parser.  - Support for pluggable annotations  - Many GUI improvements, such as integration of swingworker in the API, table sorting and filtering, and true swing double-buffering (eliminating the gray-area effect). | Features Added:  - Upgrade class-loader architecture: A method that frees the underlying resources, such as open files, held by a URLClassLoader  - Concurrency and collections updates: A lightweight fork/join framework, flexible and reusable synchronization barriers, transfer queues, concurrent linked double-ended queues, and thread-local pseudo-random number generators.  - Internationalization Upgrade: Upgrade on Unicode 6.0, Locale enhancement and Separate user locale and user-interface locale.  - More new I/O APIs for the Java platform (NIO.2), NIO.2 filesystem provider for zip/jar archives, SCTP, SDP, TLS 1.2 support.  - Security & Cryptography implemented Elliptic-curve cryptography (ECC).  - Upgrade to JDBC 4.1 and Rowset 1.1.  - XRender pipeline for Java 2D, Create new platform APIs for 6u10 graphics features, Nimbus look-and-feel for Swing, Swing JLayer component, Gervill sound synthesizer.  - Upgrade the components of the XML stack to the most recent stable versions: JAXP 1.4, JAXB 2.2a, and JAX-WS 2.2.  - Enhanced MBeans." Support for dynamically-typed languages (InvokeDynamic): Extensions to the JVM, the Java language, and the Java SE API to support the implementation of dynamically-typed languages at performance levels near to that of the Java language itself  - Strict class-file checking: Class files of version 51 (SE 7) or later must be verified with the typechecking verifier; the VM must not fail over to the old inferencing verifier.  - Small language enhancements (Project Coin): A set of small language changes intended to simplify common, day-to-day programming tasks: Strings in switch statements, try-with-resources statements, improved type inference for generic instance creation ("diamond"), simplified varargs method invocation, better integral literals, and improved exception handling (multi-catch). |

|  |  |
| --- | --- |
| **Java 5** | **Java 6** |
| Features added:  - Generics: provides compile-time (static) type safety for collections and eliminates the need for most typecasts (type conversion).  - Metadata: also called annotations; allows language constructs such as classes and methods to be tagged with additional data, which can then be processed by metadata-aware utilities.  - Autoboxing/unboxing: automatic conversions between primitive types (such as int) and primitive wrapper classes (such as integer).  - Enumerations: the enum keyword creates a typesafe, ordered list of values (such as day.monday, day.tuesday, etc.). Previously this could only be achieved by non-typesafe constant integers or manually constructed classes (typesafe enum pattern).  - Swing: new skinnable look and feel, called synth.  - Var args: the last parameter of a method can now be declared using a type name followed by three dots (e.g. Void drawtext(string... Lines)). In the calling code any number of parameters of that type can be used and they are then placed in an array to be passed to the method, or alternatively the calling code can pass an array of that type.  - Enhanced for each loop: the for loop syntax is extended with special syntax for iterating over each member of either an array or any iterable, such as the standard collection classesfix the previously broken semantics of the java memory model, which defines how threads interact through memory.  - Automatic stub generation for rmi objects.  - Static imports concurrency utilities in package java.util.concurrent.  - Scanner class for parsing data from various input streams and buffers.  - Assertions  - StringBuilder class (in java.lang package)  - Annotations | Features added:  - Support for older win9x versions dropped.  - Scripting lang support: Generic API for integration with scripting languages, & built-in mozilla javascript rhino integration  - Dramatic performance improvements for the core platform, and swing.  - Improved web service support through JAX-WS JDBC 4.0 support  - Java compiler API: an API allowing a java program to select and invoke a java compiler programmatically.  - Upgrade of JAXB to version 2.0: including integration of a stax parser.  - Support for pluggable annotations  - Many GUI improvements, such as integration of swingworker in the API, table sorting and filtering, and true swing double-buffering (eliminating the gray-area effect). |