Features of Java 8

- Lambda expressions
- Method references
- Default methods
- New Streams API
- New DateTime API
- Nashorn

Why Lambda? λ

- Lambda calculus <-> Turing Machines
  - Alonzo Church – Alan Turing
- JVM Closures: Groovy, Clojure, Scala, JRuby
- Functional programming
  - Multi-core processors
  - Concurrency
  - Lazy evaluation

Pre-Lambda Expressions

- Syntax: Anonymous classes
- ```
  Runnable r = new Runnable() {
    public void run() {
      doSomething(); }}
  ```
- ```
  ActionListener l = new ActionListener() {
    public void actionPerformed(ActionEvent event){
        handle(event); }}
  ```
- ```
  Comparator<Long,Long> c = new Comparator<> {
    public int compare(Long a, Long b) {
       return a – b; }}
  ```
  ```

Lambda Expressions

- Syntax
- `Runnable r = () -> doSomething();`
  - Equivalent: `() -> { doSomething(); }`
- `ActionListener l = event -> handle(event);`
  - Equivalent: `(ActionEvent event) -> handle(event);`
- `Comparator<Long,Long> c = (a,b) -> a – b;`
  - Equivalent:

```
(Long a, Long b) -> {
  return a – b;
}
```

Lambda Expressions

- Scope
- this refers to containing object
  - `Runnable r = () -> System.out.println(this);`
  - `Runnable r = () -> System.out.println(toString());`
- You can refer to final or *effectively* final variables
  - `int g = 10;`
  - `Predicate<Integer> p = (num) -> num > g;`

Lambda Expressions

- Byte-code
- Lambda expressions become methods (lambda$0, lambda$1, …)
- Invoked using invokedynamic

Functional Interfaces

- A functional interface has one abstract method
- SAM = Single Abstract Method
- @FunctionalInterface enforces
- New in Java 8: java.util.function.
  - `Function<T,R>`
  - `Supplier<T>`
  - `Predicate<T>`
  - `Consumer<T>`

Method References

- Three forms
  - Static methods: Math::max
  - Instance methods: str::compareTo
  - Type methods: String::length
  - “For a reference to an instance method of an arbitrary object, the type to which the method belongs precedes the delimiter, and the invocation's receiver is the first parameter of the functional interface method”
- Constructor: TreeSet::new

DEMO

- Hello.java
- Func.java

Default Methods

- Defender methods/ Virtual Extension methods
- Syntax

```
interface Iterable {
default void forEach(Consumer<? Super T> action)
```

- Multiple defaults:
  - Class X implements A, B // both implement foo()
  - Override foo and call A.super.foo();

Static Methods on Interface

- Makes helper/util methods easier to locate
  - (Stream instead of StreamUtil or Streams)
- Syntax

```
public interface Stream {
 static<T> Stream<T> of(T... values) {
  return Arrays.stream(values);
 }
```

DEMO

- Iface.java
- Twoface.java

New Stream API

- java.util.stream.Stream
- Lazily evaluated
- list.stream() //parallelStream()
  .map(student -> student.numberOfAwards)
  .filter(num -> num > 0)
  .reduce(0, (n1, n2) -> n1 + n2); //sum
- peek(System.out::println);
- sorted() //sorts the stream
- limit(n)

New Stream API: Collect

- stream.collect
  - “Performs a mutable reduction operation on the elements of this stream.”
- Two Forms:
  - `R collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)`
  - `R collect(Collector<? super T,A,R> collector)`

New Stream API: Collect

- stream.collect(Collector)
- `java.util.stream.Collectors`
  - `toList()`
  - `toCollection(TreeSet::new)`
  - `averagingInt(String::length)`
  - `summarizingInt(String::length)`
    - `IntSummaryStatistics`
    - `Average, max, min, count, sum`
  - `joining(String)`
  - `groupingBy(Function) //Map result`

DEMO

- D.java
- Nio.java

Generating Streams

- collection.stream()
- bufferedReader.lines()
- Files.lines(path)
- Files.walk(path) //depth-first file tree
- pattern.splitAsStream(string)
- Stream.generate(() -> new Thing());
- Stream.of(...)
- Arrays.stream(...)

Generating Streams 2

- `Stream.iterate(0, x -> x + 1)`
- `IntStream.range(0, 11)`
- `Random rnd = new Random();`
  - `rnd.ints()`
  - `rnd.longs()`
  - `rnd.doubles()`
  - `rnd.ints(1, 10) // 1 to 9`
- `Stream.empty()`

Stream reduce

- Question: Stream.empty().reduce((a,b)->a) returns what?

Optional

- java.util.Optional //avoiding null return values (and thus NullPointerException).
  - Guava's Optional
  - Scala's Option
  - Nat Pryce's Maybe
  - ...
- Generating:
  - `Optional.of(x)`
  - `Optional.empty()`

Optional

- Methods:
  - `isPresent()`
  - `get()`
  - `orElse(T)`
  - `orElseGet(Supplier<T>)`
  - `orElseThrow(Supplier<X extends Throwable>)`
  - `ifPresent(Consumer)`
  - `map(Function)`

Nashorn Javascript Engine

- Nashorn replaces Rhino as the default Javascript engine for the Oracle JVM.
  - ECMAScript-262 Edition 5.1
  - Faster, less memory, invokedynamic
  - Object, Function, Number, String, Date, Array, RegExp, etc.
- Command line tool (jjs).
  - $ jjs script.js
- Integrates with Java; Java<->Javascript

Nashorn Javascript Engine

- Using ScriptEngine:
  - ScriptEngineManager engineManager = new ScriptEngineManager();
  - ScriptEngine engine = engineManager.getEngineByName("nashorn");
  - engine.eval(new FileReader('library.js'));
- Javascript:
  - var imports = new JavaImporter(java.util, java.io, java.nio.file);
  - with (imports) { var paths = new LinkedList(); }
  - var Callable = Java.type("java.util.concurrent.Callable");
  - var task = Java.extend(Callable, {call: function() {print("task "+i)}})

DEMO

- Nashorn.java

New Date-Time

- New Classes (Immutable)
  - LocalDate – Day, month, year.
  - LocalTime – Time of day only.
  - LocalDateTime – Both date and time.
  - ZonedDateTime – Date/time with time-zone.
- Creation
  - LocalTime now = LocalTime.now();
  - LocalDate date = LocalDate.of(2014, Month.MARCH, 27);
  - LocalTime time = LocalTime.of(12, 15, 0);

New Date-Time

- Editing
  - LocalTime later = now.plus(8, HOURS);
  - LocalTime time = LocalTime.of(12, 15, 0);
  - LocalDateTime datetime = date.atTime(time);
- Duration = A time-based amount of time, such as '34.5 seconds'.
- Period = A date-based amount of time, such as '2 years, 3 months and 4 days'.
  - `Period.between(date1, date2)`
  - `Duration.between(time1, time2)`

DEMO

- ThreeTen.java

New Date-Time: Clock

- Useful for testing
- Clocks
  - Default: `Clock.systemDefaultZone();`
  - Fixed: `Clock.fixed(Instant.now(), ZoneId.systemDefault());`
- `LocalDate.now(clock)`

Bye-bye PermGen

- No more Permanent generation
  - Most allocations for the class metadata are now allocated out of native memory.
    - “The proposed implementation will allocate class meta-data in native memory and move interned Strings and class statics to the Java heap” -http://openjdk.java.net/jeps/122
  - java.lang.OutOfMemoryError: Metadata space
  - Convergence of Oracle JRockit & HotSpot

Miscellaneous

- java.util.Base64
- Cryptography upgrades (lots)
- JDBC 4.2
- Repeatable Annotations
- Annotations on types
- ....?
- http://openjdk.java.net/jeps/0

Thanks!

Questions? Comments?

- https://github.com/adamd/hellojava8
- @adamldavis
- https://leanpub.com/u/adamldavis