JAVA 9

THE NEXT GENERATION OF THE JDK PLATFORM

JWORKS

POWERED BY ORDINA

ABOUT ME

Yannick De Turck
Java Developer
JVM Languages Competence Lead
JWorks, Ordina Belgium
@YannickDeTurck
https://github.com/YannickDeTurck
https://ordina-jworks.github.io

ROADMAP

- 2016/05/26: Feature Complete
- 2016/12/22: Feature Extension Complete
- 2017/01/05: Rampdown Start
- 2017/02/09: All Tests Run
- 2017/02/16: Zero Bug Bounce
- 2017/03/16: Rampdown Phase 2
- 2017/07/06: Final Release Candidate
- 2017/07/27: General Availability
- JDK 9 Schedule and Feature List

NEW FEATURES

Project Jigsaw: Modules

Project Kulla: JShell

Factory methods for collections

Diamond operator for anonymous inner classes

Try-with-resources enhancement

CompletableFuture API improvements

Underscore ('_') character is a keyword

SafeVarargs on private methods

NEW FEATURES

Private methods in interfaces

HTTP 2.0 Client

Process API improvements

Reactive streams

Optional improvements

Collectors improvements

Stream improvements

Deprecated

...

PROJECT JIGSAW: MODULES

JAVA PLATFORM MODULE SYSTEM (JPMS)

- Originally targeted at Java 7
- New concept: modules
  - Stepping down from monolithic JARs
  - Make Java more easily scalable down to small computing devices
  - Improved security and maintainability
  - Strong encapsulation and reliable configuration
  - Improved application performance
- JDK has been modularised, most internal APIs encapsulated
- The State of the Module System

WHAT'S A MODULE?

- A named, self-describing collection of **code** and **data**
  - **Code**: set of packages containing Java classes and interfaces
  - **Data**: resources and other kinds of static information
- Declares the modules it requires to be compiled and run
- Declares the packages it exports
- Versions are not declared

EXAMPLE OF A MODULE

- `module-info.java`
  - Module metadata
  - Located at the root of the source-file hierarchy

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

NAME

- Arbitrary
- Best to stick to inverse-URL naming schema of packages

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

REQUIRES

- The modules it depends on to compile and run
- `transitive` is used to require the readability of a module
  - Useful when exporting a package containing a type that points to another module
- Unlike classpath issues, any possible issues are discovered at compile time
- Every module automatically depends on `java.base`

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

EXPORTS

- The packages it exports
- Only public types are accessible from outside
- Non-public, non-exported types are not even accessible by Reflection

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

OPENS

- Makes the package accessible to code in other modules at runtime only, not at compile time
- Also makes the types accessible via reflection
- Useful for dependency injection frameworks

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

USES

- Specifies a service interface which the current module may discover via `java.util.ServiceLoader`
- Services allow for loose coupling between service consumers modules and service providers modules

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

PROVIDES

- Specifies a service interface with service implementations to `java.util.ServiceLoader`

```
module com.foo.bar {
    requires org.baz.qux;
    requires transitive org.foo.bar;

    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;

    opens com.bar.foo.model;

    uses com.some.foo.SPI;
    provides com.some.bar.SPI with com.some.bar.SPIImplementation;
}
```

COMPILATION

- With JDK 9, modular JARs are built

```
src/com.greetings/com/greetings/Main.java
src/com.greetings/module-info.java
```

```
$ javac -d mods/com.greetings \
    src/com.greetings/module-info.java \
    src/com.greetings/com/greetings/Main.java

$ java --module-path mods -m com.greetings/com.greetings.Main
```

- `-m` specifies the main module

COMPILATION

- Maven seems to support building modules already

```
$ mvn clean install
$ java --module-path target/my-project.jar \
    -m com.project/com.project.Runner
```

USING A MODULAR JAR FILE

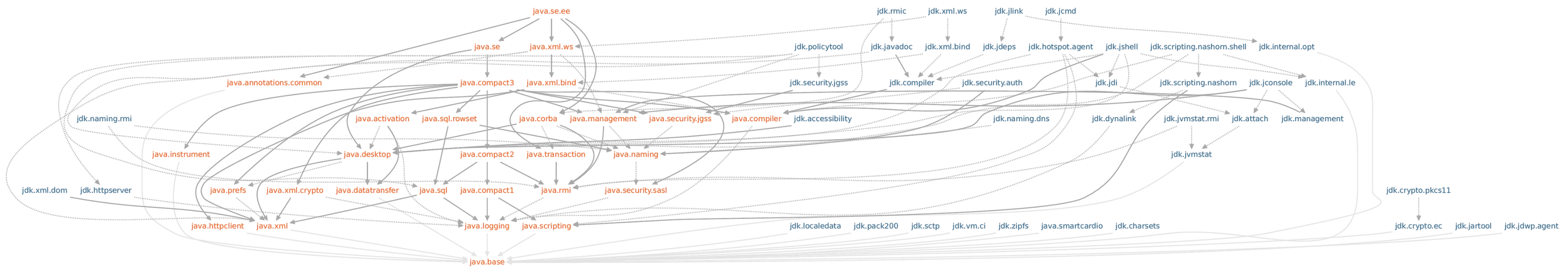
- Add it to the modulepath instead of the classpath
- But... You can still add it to the classpath (modular or not)
  - `module-info.class` will be ignored
  - All code is packaged up in the **unnamed module**, exporting all packages and can read all other modules
- Any JAR on the module path without module descriptor ends up as an **automatic module**
  - JDK generates a name depending on the JAR filename
  - Implicitly exports all packages and can read all other modules
  - Allows your Java 9 project to use pre-Java 9 libraries

SPLITTING MODULES

- Folder per module
- Each module has its `module-info.java`

```
src/org.module.a/module-info.java
src/org.module.a/org/module/a/Service.java
src/org.foo.bar/org/foo/bar/Main.java
src/org.foo.bar/module-info.java
```

JDK 9 MODULES



- The JDK itself is split up into ~92 modules
  - Check `$ java --list-modules`

EXISTING MODULE SYSTEMS

- What about OSGi or JBoss Modules?
  - Little resemblance
  - No direct support from the JVM
  - Each module is launched in its own classloader
- JPMS is a JVM extension, to be used for something that can be described as modules
- Blogpost: Critical Deficiencies in Jigsaw
- Blogpost: Is Jigsaw good or is it wack?

SUMMARY

- Modules are a new concept
- Not to be confused with Maven, OSGi or JBoss modules
- Will be interesting to see how it evolves and how the open source community embraces it

MORE INFORMATION

- JPMS: Modules in the Java Language and JVM
- Project Jigsaw: Module System Quick-Start Guide
- Java SE 9 modules (JPMS) - an introduction
- Java 9 modules - JPMS basics

PROJECT KULLA: JSHELL

PROJECT KULLA: JSHELL

- A REPL for Java
- Interactive tool to evaluate Java declarations, statements and expressions
- Supports tab-completion
- No need to use semicolons for single lines
- `$ jshell`
- `> /help`
- `> /exit`

STARTING UP

```
Yannicks-MacBook-Pro:~ yannickdeturck$ jshell
|  Welcome to JShell -- Version 9-ea
|  For an introduction type: /help intro

jshell> System.out.println("Hello JWorks!")
Hello JWorks!
```

COMMANDS

```
jshell> /help
|   /list [<name or id>|-all|-start]
|        list the source you have typed
|   /edit <name or id>
|        edit a source entry referenced by name or id
|   /drop <name or id>
|        delete a source entry referenced by name or id
|   /save [-all|-history|-start] <file>
|        Save snippet source to a file.
|   /open <file>
|        open a file as source input
|   /vars [<name or id>|-all|-start]
|        list the declared variables and their values
|   /methods [<name or id>|-all|-start]
|        list the declared methods and their signatures
|   /types [<name or id>|-all|-start]
|        list the declared types
|   /imports
|        list the imported items
```

COMMANDS

```
|  /exit
|       exit jshell
|  /env [-class-path <path>] [-module-path <path>] [-add-modules <modules>] ...
|       view or change the evaluation context
|  /reset [-class-path <path>] [-module-path <path>] [-add-modules <modules>]...
|       reset jshell
|  /reload [-restore] [-quiet] [-class-path <path>] [-module-path <path>]...
|       reset and replay relevant history -- current or previous (-restore)
|  /history
|       history of what you have typed
|  /help [<command>|<subject>]
|       get information about jshell
|  /set editor|start|feedback|mode|prompt|truncation|format ...
|       set jshell configuration information
|  /? [<command>|<subject>]
|       get information about jshell
|  /!
|       re-run last snippet
|  /<id>
|       re-run snippet by id
```

VARIABLES & EXPRESSIONS

```
jshell> int i=0
i ==> 0

jshell> i
i ==> 0

jshell> i++
$4 ==> 0

jshell> i
i ==> 1

jshell> $4
$4 ==> 1
```

CLASSES & METHODS

```
jshell> class HelloWorld {
   ...> public static void say(String message) {
   ...> System.out.println(message);
   ...> }
   ...> }
|  created class HelloWorld

jshell> HelloWorld.say("Hi JWorks!")
Hi JWorks!
```

DEFAULT IMPORTS

```
jshell> /imports
|     import java.io.*
|     import java.math.*
|     import java.net.*
|     import java.nio.file.*
|     import java.util.*
|     import java.util.concurrent.*
|     import java.util.function.*
|     import java.util.prefs.*
|     import java.util.regex.*
|     import java.util.stream.*
```

ADD IMPORTS

```
jshell> import static java.lang.System.out

jshell> out.println("hi")
hi
```

SHOW VARIABLES AND METHODS

```
jshell> /vars
|    int $4 = 0
|    int $8 = 18
|    int i = 1
|    int $15 = 0
|    int $19 = 18

jshell> /methods
|    int sum(int,int)
```

SAVE AND LOAD SESSIONS

```
jshell> /save my-first-jshell.txt

jshell> /open my-first-jshell.txt
Hello JWorks!
Hi JWorks!
hi

jshell>
```

ALSO WORKS FOR CLASSES

```
jshell> /open path/to/MyClass.java
```

SET THE CLASSPATH

```
jshell> /env -class-path target/some.jar
```

ADD MODULES

```
jshell> /env -module-path target/mods/ -add-modules org.foo.bar
```

EDIT IN EXTERNAL EDITOR

```
jshell> /edit

jshell> /edit sum
```

FACTORY METHODS FOR COLLECTIONS

FACTORY METHODS FOR COLLECTIONS

- Creating immutable `Collection` objects was cumbersome
- New utility methods for creating `List`, `Set` and `Map` interfaces

JAVA 8

```java
List<String> list = new ArrayList<>();
list.add("why");
list.add("hello");
list.add("there");
List<String> immutableList = Collections.unmodifiableList(list);
```

JAVA 9

```
List<String> emptyList = List.of();
List<String> list = List.of("foo", "bar");
Map<String, String> map = Map.of("key1", "value1", "key2", "value2");
Map<String, String> mapOfEntries = Map.ofEntries(Map.entry("key1", "value1"),
    Map.entry("key2", "value2"));
```

DIAMOND OPERATION FOR ANONYMOUS INNER CLASSES

JAVA 8

```
<T> Package<T> createPackage(T packageContent) {
    // type needs to be specified
    return new Package<T>(packageContent) { ... };
}
```

- Why can't the compiler infer the type here?
- Reason: non-denotable types

JAVA 9

- More relaxed, allowed if a denotable type is inferred

```
<T> Package<T> createPackage(T packageContent) {
    // we have our diamond :-)
    return new Package<>(packageContent) { ... };
}

// still can't do it here though as the inferred type is non-denotable
Package<?> createPackage(Object content) {
    List<?> innerList = Arrays.asList(content);
    // we can't do the following because the inferred type is non-denotable:
    // instead we have to denote the type we want:
    return new Package<List<?>>(innerList) { };
}
```

TRY-WITH-RESOURCES ENHANCEMENT

TRY-WITH-RESOURCES ENHANCEMENT

```java
// Java 7
BufferedReader reader1 = new BufferedReader(new FileReader("file.txt"));
try (BufferedReader reader2 = reader1) {
    System.out.println(reader2.readLine());
}
// Java 9
BufferedReader reader = new BufferedReader(new FileReader("file.txt"));
try (reader) {
    System.out.println(reader.readLine());
}
```

COMPLETABLEFUTURE API IMPROVEMENTS

COMPLETABLEFUTURE API IMPROVEMENTS

- Support for delays and timeouts
- Better support for subclassing
- Couple of new utility methods

NEWINCOMPLETEFUTURE()

- The "virtual constructor"
- Subclasses of `CompletableFuture` should implement this as it is used internally
- Establishes the concrete type returned by `CompletionStage` methods

```
CompletableFuture<U> newIncompleteFuture()
```

```
class MyCompletableFuture<T> extends CompletableFuture<T> {
    @Override
    public <U> CompletableFuture<U> newIncompleteFuture() {
        return new MyCompletableFuture<>();
    }
}
```

COMPLETEASYNC()

- Complete the `CompletableFuture` asynchronously using the value given by the Supplier provided

```
CompletableFuture<String> future = new CompletableFuture<>();
future.complete("calculated value1");
future.whenComplete((s, throwable) -> System.out.println(s));
future = new CompletableFuture<>();
CompletableFuture<String> completedAsync =
    future.completeAsync(() -> "calculated value2");
completedAsync.whenComplete((s, throwable) -> System.out.println(s));
```

ORTIMEOUT()

- Resolves the future exceptionally with `TimeoutException`

```
new CompletableFuture().orTimeout(1, TimeUnit.SECONDS);
```

COMPLETEONTIMEOUT()

- `CompletableFuture` will be resolved with an alternative result if it stays unresolved after x seconds

```java
CompletableFuture<Object> future = new CompletableFuture<>();
future.completeOnTimeout("alternative result", 3, TimeUnit.SECONDS);
```

STATIC UTILITY METHODS

```
Executor delayedExecutor(long delay, TimeUnit unit, Executor executor)
Executor delayedExecutor(long delay, TimeUnit unit)
<U> CompletionStage<U> completedStage(U value)
<U> CompletionStage<U> failedStage(Throwable ex)
<U> CompletableFuture<U> failedFuture(Throwable ex)
```

THE UNDERSCORE CHARACTER

THE UNDERSCORE CHARACTER

- As of release 9, '_' is a keyword and may not be used as an identifier
- Hinting that future Java versions may use it as a keyword or give it special semantics

SAFEVARARGS ON PRIVATE METHODS

SAFEVARARGS ON PRIVATE METHODS

- `@SafeVarargs`?
- Tells the compiler that your mixture of varargs and generics is safe
- Otherwise warnings will be given:

```
Foo.java uses unchecked or unsafe operations

Possible heap pollution from parameterized vararg type
```

- In Java 9: `@SafeVarargs` support for private methods

UNSAFE EXAMPLE

- **Safe**: Depending on the elements of the array being instances of `T`
- **Unsafe**: Depending on the array being an instance of `T[]`

```java
public static void main(String[] args) {
    for (String string : arrayOfTwo("a", "b")) {
        System.out.println(string);
        // java.lang.ClassCastException: java.base/[Ljava.lang.Object;
        // cannot be cast to java.base/[Ljava.lang.String;
    }
}

@SafeVarargs
static <T> T[] asArray(T... args) {
    return args;
}

static <T> T[] arrayOfTwo(T a, T b) {
    return asArray(a, b);
}
```

ABOUT VARARGS

- Varargs: syntactic sugar that undergoes a "re-writing" at compile-time
- Variable arguments are made into an array of `T[]`, but it is impossible to create an array of a type-parameter component type (`new T[] { ... }`)
- So instead, a `new Object[] { ... }` is used
- Problem: Arrays in Java know their component type at runtime so the passed array object will have the wrong component type at runtime
- So if you depend on the runtime component type of the array, it will not be safe

PRIVATE METHODS IN INTERFACES

PRIVATE METHODS IN INTERFACES

- Java 8 brought default methods
- But... Code reuse between default methods was rather unpleasant
- Private and private static method support added

JAVA 8

```java
public interface PackageSender {
    default void prepare() {
        // prepare stuff logic right here
    }

    default void cleanUp () {
        // cleanup stuff logic right here
    }

    default void sendPackage(Package thePackage) {
        prepare();
        // sending package logic right here
        cleanUp();
    }
}
```

JAVA 9

```java
public interface PackageSender {
    private void prepare() {
        // prepare stuff logic right here
    }

    private void cleanUp () {
        // cleanup stuff logic right here
    }

    default void sendPackage(Package thePackage) {
        prepare();
        // sending package logic right here
        cleanUp();
    }
}
```

HTTP 2.0 CLIENT

HTTP 2.0 CLIENT

- Why HTTP/2?
  - Bidirectional communication using push requests
  - Multiplexing within a single TCP connection

HTTP 2.0 CLIENT

- Delivered as an incubator module
  - `requires jdk.incubator.httpclient;`
- New HTTP 2 Client API under `java.net.http`
- A more userfriendly API
- Supports
  - HTTP/1.1 and HTTP2/2 protocols
  - Synchronous and Asynchronous Modes (WebSocket API)
  - Long running connections
  - Stateful connections
- API documentation

EXAMPLE GET

```
HttpClient client = HttpClient.newHttpClient();
HttpResponse<String> response = client.send(
    HttpRequest
        .newBuilder(new URI("http://www.google.com/"))
        .headers("FooHeader", "fooValue", "BarHeader", "barValue")
        .GET()
        .build(),
    HttpResponse.BodyHandler.asString()
);
int statusCode = response.statusCode();
String body = response.body();
```

EXAMPLE POST

```
HttpClient client = HttpClient.newHttpClient();
HttpResponse<String> response = client.send(
    HttpRequest.newBuilder()
        .uri(new URI("http://localhost:8080/upload/"))
        .POST(HttpRequest.BodyProcessor.fromFile(Paths.get("/tmp/file-to-upload.txt"))
        .build(),
    HttpResponse.BodyHandler.discard(null));
int statusCode = response.statusCode();
String responseBody = response.body();
```

EXAMPLE ASYNCHRONOUS GET

```
HttpClient client = HttpClient.newHttpClient();
CompletableFuture<HttpResponse<String>> response = client.sendAsync(
        HttpRequest.newBuilder()
                .uri(new URI("https://labs.consol.de/"))
                .GET()
                .build(),
        HttpResponse.BodyHandler.asString());
response.whenComplete((resp, throwable) -> {
    System.out.println(resp.statusCode());
    System.out.println(resp.body());
});
```

ALTERNATIVE EXAMPLE
ASYNCHRONOUS GET

```java
HttpClient client = HttpClient.newHttpClient();
CompletableFuture<HttpResponse<String>> response = client.sendAsync(
        HttpRequest.newBuilder()
                .uri(new URI("https://labs.consol.de/"))
                .GET()
                .build(),
        HttpResponse.BodyHandler.asString());
Thread.sleep(3000);
if (response.isDone()) {
    System.out.println(response.get().statusCode());
    System.out.println(response.get().body());
} else {
    response.cancel(true);
    System.out.println("Timeout, cancelling the request...");
}
```

EXAMPLE SYSTEM PROXY SETTINGS

```java
HttpClient client = HttpClient.newBuilder()
    .proxy(ProxySelector.getDefault())
    .build();
HttpResponse<String> response = client.send(
    HttpRequest.newBuilder()
        .uri(new URI("https://localhost:8080"))
        .GET()
        .build(),
    HttpResponse.BodyHandler.asString());
int statusCode = response.statusCode();
String responseBody = response.body();
```

EXAMPLE BASIC AUTHENTICATION

```java
HttpClient client = HttpClient.newBuilder()
    .authenticator(new Authenticator() {
        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication("username", "password".toCharArray());
        }
    })
    .build();
HttpResponse<String> response = client.send(
    HttpRequest.newBuilder()
        .uri(new URI("https://localhost:8080"))
        .GET()
        .build(),
    HttpResponse.BodyHandler.asString());
int statusCode = response.statusCode();
String responseBody = response.body();
```

PROCESS API IMPROVEMENTS

SOME QUICK HISTORY

- Pretty primitive prior to Java 5
- In Java 5, `ProcessBuilder` API was introduced offering a cleaner way of spawning new processes
- Java 9 improves controlling and managing of OS processes
- For more info, see JEP 102

NEW INTERFACES

```
java.lang.ProcessHandle
java.lang.ProcessHandle.Info
```

EXAMPLE: LS AND GREP

- `$ ls /Users/yannick/Desktop/foo/ | grep xml`

```
ProcessBuilder ls = new ProcessBuilder()
    .command("ls")
    .directory(Paths.get("/Users/yannick/Desktop/foo/").toFile());
ProcessBuilder grepXml = new ProcessBuilder()
    .command("grep", "xml")
    .redirectOutput(ProcessBuilder.Redirect.INHERIT);
List<Process> processes = ProcessBuilder
    .startPipeline(Arrays.asList(ls, grepXml));
// bar.xml
// secret-copy.xml
// secret.xml
```

EXAMPLE: WAIT UNTIL PROCESSES ARE DONE

```
CompletableFuture[] lsThenGrepFutures = processes.stream()
    .map(Process::onExit)
    .map(processFuture -> processFuture.thenAccept(
        process -> System.out.println("PID: " + process.getPid())))
    .toArray(CompletableFuture[]::new);
CompletableFuture
    .allOf(lsThenGrepFutures)
    .join();
```

EXAMPLE: PROCESSINFO

```java
ProcessHandle processHandle = ProcessHandle.current();
ProcessHandle.Info processInfo = processHandle.info();
processHandle.getPid(); // long
// 12138
processInfo.arguments(); // Optional<String[]>
// -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:62727,suspend=y,server=n
// -Dfile.encoding=UTF-8
// -classpath
// /Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar
processInfo.command(); // Optional<String[]>
// /Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java
processInfo.startInstant(); // Optional<Instant>
// 2017-04-11T19:01:10.099Z
processInfo.totalCpuDuration(); // Optional<Duration>
// PT0.465663S
processInfo.user(); // Optional<String[]>
// yannick
```

EXAMPLE: KILLING A PROCESS

```java
ProcessHandle.allProcesses()
    .filter(process -> process.info().command().filter(command ->
        command.toLowerCase().contains("twitter")).isPresent())
    .forEach(process -> process.info().command()
        .ifPresent(command ->
            System.out.println((process.destroyForcibly() ? "Successfully killed" : "
                + "process " + command))
    );
// Successfully killed process /Applications/Twitter.app/Contents/MacOS/Twitter
```

JAVADOC

- ProcessHandle
- Process

REACTIVE STREAMS

REACTIVE STREAMS

- The Reactive Manifesto
- New Reactive Streams API
- Publish/Subscribe Framework to implement asynchronous, scalable and parallel applications
- JDK provides only the interfaces and no implementations
- Full details see: JEP 266

REACTIVE STREAMS

```
java.util.concurrent.Flow
java.util.concurrent.Flow.Publisher
java.util.concurrent.Flow.Subscriber
java.util.concurrent.Flow.Processor
```

BUILDING BLOCKS - PUBLISHER

- Produces items for subscribers to consume
- `subscribe(Subscriber)`

BUILDING BLOCKS - SUBSCRIBER

- Subscribes to publishers to receive messages
- `onSubscription(Subscription)`
- `onNext(T)`
- `onError(Throwable)`
- `onComplete()`

BUILDING BLOCKS - SUBSCRIPTION

- Acts as both a publisher and a subscriber
- Sits between the publisher and subscriber, transforming one steam to another
- Multiple processors can be chained
- Final one is processed by the subscriber

```
interface Processor<T,R> extends Subscriber<T>, Publisher<R>
```

BUILDING BLOCKS - PROCESSOR

- Connection between a publisher and a subscriber
- `request(long)`
  - A request of `Long.MAX_VALUE` can be seen as effectively unbounded
- `cancel()`

BUILDING BLOCKS - PROTOCOL

- `onSubscribe onNext* (onError | onComplete)?`

EXAMPLE WITH SUBMISSIONPUBLISHER

```java
static class MySubscriber<T> implements Flow.Subscriber<T> {
    private Flow.Subscription subscription;

    @Override public void onSubscribe(Flow.Subscription subscription) {
        System.out.println("Received subscription: " + subscription);
        this.subscription = subscription;
        this.subscription.request(1);
    }

    @Override public void onNext(T item) {
        System.out.println("Treating item: " + item);
        this.subscription.request(1);
    }

    @Override public void onError(Throwable throwable) {
        System.out.println("Treating error: " + throwable);
    }

    @Override public void onComplete() {
        System.out.println("Completed!");
    }
}
```

EXAMPLE WITH SUBMISSIONPUBLISHER

```java
SubmissionPublisher<String> publisher = new SubmissionPublisher<>();

Flow.Subscriber<String> subscriber = new MySubscriber<>();
publisher.subscribe(subscriber);

List<String> messages = List.of("these", "are", "a", "couple", "of", "words");
messages.forEach(publisher::submit);

Thread.sleep(1000L);
publisher.close();

// Received subscription: java.util.concurrent.SubmissionPublisher$BufferedSubscripti
// Treating item: these
// Treating item: are
// Treating item: a
// Treating item: couple
// Treating item: of
// Treating item: words
// Completed!
```

FLOW

1. Create a `Publisher` and a `Subscriber`
2. Subscriber subscribes with `Publisher::subscribe`
3. The publisher creates a `Subscription` and calls `Subscriber::onSubscription` which the subscriber stores
4. Subscriber calls `Subscription::request` to request a number of items
5. Publisher hands items to the subscriber by calling `Subscriber::onNext`
6. Publisher might at some point be depleted or run into trouble and call `Subscriber::onComplete` or `Subscriber::onError`, respectively
7. Subscriber might either continue to request more items every now and then or cut the connection by calling `Subscription::cancel`

FLOW

- Reactive Programming with JDK 9 Flow API

OPTIONAL CLASS IMPROVEMENTS

OPTIONAL CLASS IMPROVEMENTS

- New methods to `java.util.Optional class`
  - `stream()`
  - `ifPresentOrElse()`
  - `or()`

STREAM()

- Returns a sequential `Stream` containing the value if present otherwise an empty `Stream`

```
Stream<Optional<BlogPost>> findBlogPosts(String category) {...}

// java 8
Stream<BlogPost> blogPosts = findBlogPosts("java")
     .filter(Optional::isPresent)
     .map(Optional::get);
// java 9
Stream<BlogPost> blogPosts = findBlogPosts("java")
     .flatMap(Optional::stream);
```

IFPRESENTORELSE()

- Combines `ifPresent()`, `isPresent()` and `orElse()`

```
public void ifPresentOrElse(Consumerl<? super Tl> action,
                            Runnable emptyAction)

Optional<Integer> optionalInt = Optional.of(7);
optionalInt.ifPresentOrElse(
    i -> System.out.println("number is " + i),
    () -> System.out.println("it's empty")
);
```

OR()

- Use the current value if present, otherwise fallback on another value

```
Optional<BlogPost> blogPost = blogPostService.findBlogPost(id)
    .or(remoteBlogPostService.findBlogPost(id))
    .or(archivedBlogPostService.findBlogPost(id));
```

COLLECTORS CLASS IMPROVEMENTS

NEW METHODS

- New methods to `java.util.stream.Collectors`
  - `Collectors.filtering()`
  - `Collectors.flatMapping()`

COLLECTORS.FILTERING()

- Similar to `filter()` but used for different scenarios
- Designed to be used along with `groupingBy`
- Specify a function for filtering the input elements and a collector to collect the filtered elements

```
// counting the numbers in a list
// filter
List<Integer> numbers = List.of(1, 2, 3, 5, 5);
Map<Integer, Long> result = numbers.stream()
    .filter(val -> val > 3)
    .collect(Collectors.groupingBy(i -> i, Collectors.counting()));
result.forEach((key, val) -> System.out.println(key + " -> " + val));
// 5 -> 2

// filtering
result = numbers.stream()
    .collect(Collectors.groupingBy(i -> i,
        Collectors.filtering(val -> val > 3, Collectors.counting())));
result.forEach((key, val) -> System.out.println(key + " -> " + val));
// 1 -> 0, 2 -> 0, 3 -> 0, 5 -> 2
```

COLLECTORS.FLATMAPPING()

- The bigger brother of `mapping()`
- Also designed to be used along with `groupingBy()`
- Gets rid of the intermediate collection and writes directory to a single container mapped to the group

```
// get the comments for each blogpost category
List<BlogPost> blogposts = service.getBlogPosts();
Map<String, List<List<String>>> blogPostComments = blogposts.stream()
    .collect(Collectors.groupingBy(BlogPost::getCategory,
        Collectors.mapping(BlogPost::getComments, Collectors.toList())));
// junit -> [[comment0], [comment0, comment1, comment2]]

Map<String, List<String>> blogPostComments2 = blogposts.stream()
    .collect(Collectors.groupingBy(BlogPost::getCategory,
        Collectors.flatMapping(blog -> blog.getComments().stream(),
            Collectors.toList())));
// junit -> [comment0, comment0, comment1, comment2]
```

STREAM IMPROVEMENTS

STREAM IMPROVEMENTS

- New methods to `java.util.stream.Stream`
  - `Stream::takeWhile`
  - `Stream::dropWhile`
  - `Stream::iterate`
  - `Stream::ofNullable`

STREAM::TAKEWHILE

- Takes elements from the stream while the predicate holds

```
Stream<T< takeWhile(Predicate<? super T< predicate);

Stream.of(1, 4, 2, -8, 6, 3)
    .takeWhile(i -> i > 0)
    .forEach(System.out::println);
// 1, 4, 2
```

STREAM::DROPWHILE

- Drops elements from the stream while the predicate holds

```
Stream<T< dropWhile(Predicate<? super T< predicate);

Stream.of(1, 4, 2, -8, 6, 3)
     .dropWhile(i -> i > 0)
     .forEach(System.out::println);
// -8, 6, 3
```

STREAM::ITERATE

- Already exists based on a seed and a function
- This stream however is infinite
- A new overload was added with a `hasNext` predicate in addition

```
<T> Stream<T> iterate(T seed, UnaryOperator<T> f)

<T> Stream<T> iterate(T seed, Predicate<? super T> hasNext,
                      UnaryOperator<T> next)

Stream.iterate(1, i -> i < 20, i -> i + 2)
    .forEach(System.out::println);
// 1, 3, 5, ..., 19
```

STREAM::OFNULLABLE

- Creates a stream with the given element
- Unless... It is null, then it is empty

```
Stream.ofNullable("hello");
Stream.ofNullable(null);
```

- So... What's the point?

STREAM::OFNULLABLE

- Useful with `flatMap`

```
BlogPost post = findBlogPost(1L); // evil method that may return null
Stream<String> comments =
    post == null ? Stream.empty() : post.getComments().stream();

BlogPost post = findBlogPost(1L); // evil method that may return null
Stream<String> comments = Stream.ofNullable(post)
    .flatMap(bpost -> bpost.getComments().stream());
```

DEPRECATED

@DEPRECATED: NEW FLAGS

- `forRemoval`: whether the annotated element is subject to removal
- `since`: the version in which the annotated element became deprecated

EXAMPLE

```java
@Deprecated(since = "1.0", forRemoval = false)
    public void someAncientMethod(){
        System.out.println("Doing ancient stuff...");
    }
```

NEW DEPRECATIONS

- Applet API
- CORBA
- Explicit constructors for primitive wrappers (`new Integer(1)`) in favour of `valueOf` and `parse`
- `Observer` and `Observable`
- SHA-1 certificates becoming phased out
- Underscore character is no longer a valid character identifier
- Most of the JDK's internal APIs have been made inaccessible by default
- Removal of `rt.jar` and `tools.jar`
  - Moved into modules
- Full list

ANYTHING ELSE?

OTHER FEATURES

- Multi-Release JARs
- Multi-Resolution Image API
- GC (Garbage Collector) Improvements
- Stack-Walking API
- Filter Incoming Serialization Data
- Enhanced Method Handles
- Platform Logging API and Service
- Compact Strings
- Parser API for Nashorn
- Javadoc Search
- HTML5 Javadoc

FEATURES THAT DIDN'T MAKE IT

Standardized lightweight JSON API

Money and Currency API

GETTING STARTED WITH JAVA 9

- Just download and install Java 9
- Already well supported in IntelliJ
- Java Platform, Standard Edition What’s New in Oracle JDK 9

QUESTIONS?

- Sources and presentation available on GitHub: https://github.com/yannickdeturck/java9workshop

THANKS FOR WATCHING!