ASMIQ

Best of Java 9 – 13

https://github.com/Michaeli71/JAX-London-Best-of-Java9-13.git

Michael Inden

CTO@ASMIQ AG

Speaker Intro

- Michael Inden, Year of Birth 1971
- Diploma Computer Science, C.v.O. Uni Oldenburg

- ~8 ¼ Years @ Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Years @ IVU Traffic Technologies AG in Aachen
- ~4 ¼ Years @ Zühlke Engineering AG in Zürich
- Since June 2017 @ **Direct Mail Informatics / ASMIQ** in Zürich

- Author @ dpunkt.verlag

E-Mail: michael.inden@asmiq.ch
Courses: https://asmiq.ch/
Blog: https://jaxenter.de/author/minden

https://github.com/Michaeli71/JAX-London-Best-of-Java9-13.git

Agenda

Workshop Contents

ASMIQ

- **PART 1:** Syntax Enhancements & News and API-Changes in Java 9
- **PART 2:** Multi-Threading with CompletableFuture and Reactive Streams
- **PART 3:** Additonal News and Changes in JDK 9

- **PART 4:** What's new in Java 10
- **PART 5:** What's new in Java 11
- **PART 6:** What's new in Java 12
- **PART 7:** What's new in Java 13

- **Separate: Java Modularization**

https://github.com/Michaeli71/JAX-London-Best-of-Java9-13.git

Workshop Contents

ASMIQ

JDK	Release Date	Development Time	LTS	Coverage in Workshop
Oracle JDK 8	3 / 2014	-	yes, ***in the mean time commercial***	-/-
Oracle JDK 9	9 / 2017	3,5 years	-	Part 1, 2, 3
Oracle JDK 10	3 / 2018	6 month	-	Part 4
Oracle JDK 11	9 / 2018	6 month	yes, ***commercial***	Part 5
Oracle JDK 12	3 / 2019	6 month	-	Part 6
Oracle JDK 13	9 / 2019	6 month	-	Part 7

PART 1: Syntax Enhancements & News and API-Changes in Java 9

Syntax Enhancements in Java 9

Anonymous inner classes and the diamond operator

```
final Comparator<String> byLength = new Comparator<String>()
{
    ...
};
```

```
final Comparator<String> byLength = new Comparator<>()
{
    ...
};
```

@Deprecated-Annotation

ASMIQ

- @Deprecated mark obsolete sourcecode

- JDK 8: no parameters
- JDK 9: two parameter @since and @forRemoval

```
/**
* @deprecated this method is replaced by someNewMethod() which is
more stable
*/
@Deprecated(since = "7.2", forRemoval = true)
private static void someOldMethod()
{
   // ...
}
```

Underscore as Identifier

ASMIQ

- _ is no valid identifier any longer (semantically it has never been ;-))
- final String _ = "Underline";

```
MyClass.java:2: error: as of release 9, '_' is a keyword, and may not be
used as an identifier
    static Object _ = new Object();
                  ^
```

Private Methods in Interfaces

Private Methods in Interfaces

```
public interface PrivateMethodsExample
{
    public abstract int method1();

    public default int calc(int a, int b) {
        return myCalc(a, b);
    }

    public default int calc2(int a, int b) {
        return myCalc(a, b);
    }

    private int myCalc(int a, int b) {
        return a + b;
    }
}
```

ASMIQ

News and API-Changes in Java 9

- Process API
- Stream-API
- Optional<T>
- Collection Factory Methods

Process API

Process Handling with Java 8 and earlier

ASMIQ

- **Limited control and management of operating system processes prior to Java 9**
- **Example Read PID: Often a different implementation is needed for each platform**

```
private static long getPidOldSytle() throws InterruptedException, IOException
{
    final Process proc = Runtime.getRuntime().exec(new String[]{ "/bin/sh", "-c", "echo $PPID" });
    if (proc.waitFor() == 0)
    {
        final InputStream in = proc.getInputStream();
        final byte[] outputBytes = new byte[in.available()];

        in.read(outputBytes);
        final String pid = new String(outputBytes);
        return Long.parseLong(pid.trim());
    }
    throw new IllegalStateException("PID is not accessible");
}
```

What do you think about this code? What is it not doing?

Simplification with Java 9

```
long pid = ProcessHandle.current().getPid();
```

Interface ProcessHandle

`ProcessHandle` can be used to **gather various other information** about processes besides the PID. The following methods are available for this purpose:

- `current()` – determines the current process as a `ProcessHandle`.
- `info()` – provides information about the process in the form of the internal interface `ProcessHandle.Info`, for example about user, command, etc.
- `info().command()` – returns the command as `Optional<String>`.
- `info().user()` – returns user as `Optional<String>` zurück
- `info().totalCpuDuration()` – extracts cpu time

Interface `ProcessHandle`

ASMIQ

In addition to information about the current process, information for all processes and all subprocesses for a process can be determined as follows:

- `allProcesses()` – Returns all processes as `Stream<ProcessHandle>`.
- `children()` – Determines for a process all its (direct) subprocesses as `Stream<ProcessHandle>`.
- `descendants()` – Determines for a process all its subprocesses as `Stream<ProcessHandle>`.

Stream API

Additions to Stream API in JDK 9

The comprehensive Stream API was one of the major new features in Java 8.

takeWhile(…)

dropWhile(…)

ofNullable(…)

iterate(… , …, …)

Additions to Stream API in JDK 9

takeWhile(…)

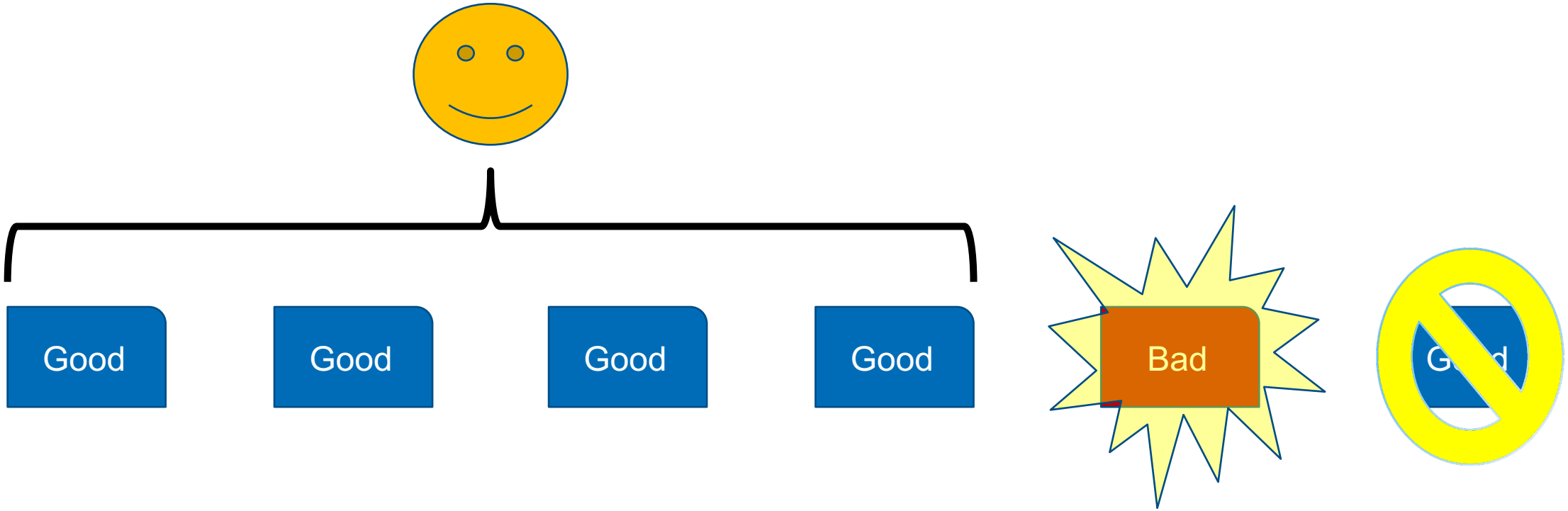
dropWhile(…)

`takeWhile(Predicate<T>)` – processes elements as long as the condition is met

ofNullable(…)

iterate(… ,…, …)

Stream – Scenario



Stream – Filter

JDK 8

```
public class StrictCustomer {

    public static void main(String[] args) {

        Stream<String> deliveredProductsQuality = Stream.of("1. Good",
                                                            "2. Good",
                                                            "3. Good",
                                                            "4. Good",
                                                            "5. Bad",
                                                            "6. Good");

        deliveredProductsQuality.
            filter(productQuality -> productQuality.contains("Good")).
            forEach(System.out::println);
    }

}
```

```
1. Good
2. Good
3. Good
4. Good
6. Good
```

Stream – Filter

JDK 8

```
public class StrictCustomer {

    public static void main(String[] args) {

        Stream<String> deliveredProductsQuality = Stream.of("1. Good",
                                                            "2. Good",
                                                            "3. Good",
                                                            "4. Good",
                                                            "5. Bad",
                                                            "6. Good");

        deliveredProductsQuality.
            filter(productQuality -> productQuality.contains("Good")).
            forEach(System.out::println);

    }

}
```

```
1. Good
2. Good
3. Good
4. Good
5. Good
```

So, what do we do?

Google

Google-Suche Auf gut Glück!

Google angeboten in: English Français Italiano Rumantsch

takeWhile(…)

ASMIQ

```
public class StrictCustomer {

    public static void main(String[] args) {

        Stream<String> deliveredProductsQuality = Stream.of("1. Good",
                                                            "2. Good",
                                                            "3. Good",
                                                            "4. Good",
                                                            "5. Bad",
                                                            "6. Good");

        deliveredProductsQuality.
                    takeWhile(productQuality -> productQuality.contains("Good")).
                    forEach(System.out::println);

    }

}
```

```
1. Good
2. Good
3. Good
4. Good
```

Additions to Stream API in JDK 9

takeWhile(…)

dropWhile(…)

`dropWhile(Predicate<T>)` – skips/drops elements as long as the condition is met

ofNullable(…)

iterate(… ,… ,…)

dropWhile(…)

ASMIQ

```
public class StrictCustomer {

    public static void main(String[] args) {

        Stream<String> deliveredProductsQuality = Stream.of("1. Bad",
                                                            "2. Bad",
                                                            "3. Bad",
                                                            "4. Good",
                                                            "5. Good",
                                                            "6. Good");

        deliveredProductsQuality.
                    dropWhile(productQuality -> productQuality.contains("Bad")).
                    forEach(System.out::println);

    }

}
```

dropWhile(…)

ASMIQ

```
public class StrictCustomer {

    public static void main(String[] args) {

        Stream<String> deliveredProductsQuality = Stream.of("1. Bad",
                                                            "2. Bad",
                                                            "3. Bad",
                                                            "4. Good",
                                                            "5. Good",
                                                            "6. Good");


        deliveredProductsQuality.
                    dropWhile(productQuality -> productQuality.contains("Bad")).
                    forEach(System.out::println);
                                                    4. Good
                                                    5. Good
                                                    6. Good
    }

}
```

Combination of Additions to Stream API in JDK 9

ASMIQ

- **Combination of the two methods for extracting data:**

```
public static void main(String[] args)
{
    Stream<String> words = Stream.of("ab", "BLA", "Xy", "<START>",
                                     "WELCOME", "TO", "JAX", "LONDON",
                                     "<END>", "x", "y", "z");

    Stream<String> extracted = words.dropWhile(str -> !str.startsWith("<START>")).
                                     skip(1).
                                     takeWhile(str -> !str.startsWith("<END>"));

    extracted.forEach(System.out::println);
}
```

```
WELCOME
TO
JAX
LONDON
```

Additions to Stream API in JDK 9

takeWhile(…)

dropWhile(…)

ofNullable(…)

iterate(…,… , …)

`ofNullable(T)` – returns a `stream<T>` with one element if the passed element is not `null`. Otherwise an empty stream is generated.

ofNullable(T)

ASMIQ

```
public class FirstNullableExample
{
    public static void main(final String[] args)
    {
        final Stream<Customer> customerStream1 = Stream.of(findCustomer());
        System.out.println(customerStream1.count());

        final Stream<Customer> customerStream2 = Stream.of(findCustomer());
        final Optional<Customer> optFirst = customerStream2.findFirst();
        optFirst.ifPresentOrElse(System.out::println,
                                 () -> System.out.println("No element"));
    }

    private static Customer findCustomer()
    {
         // complex logic for search with fallback ..
        return null;
    }
}
```

Please note: returning null is often considered bad style, but it is not uncommon in older code!!

ofNullable(T)

```
public class FirstNullableExample
{
    public static void main(final String[] args)
    {
        final Stream<Customer> customerStream1 = Stream.of(findCustomer());
        System.out.println(customerStream1.count());

        final Stream<Customer> customerStream2 = Stream.of(findCustomer());
        final Optional<Customer> optFirst = customerStream2.findFirst();
        optFirst.ifPresentOrElse(System.out::println,
                                 () -> System.out.println("No element"));
    }

    private static Customer findCustomer()
    {
        return null;
    }
}
```

```
<terminated> FirstNullableExample [Java Application] /Library/Java/JavaVirtualMachines/jd
1
Exception in thread "main" java.lang.NullPointerException
        at java.base/java.util.Objects.requireNonNull(Objects.java
        at java.base/java.util.Optional.of(Optional.java:111)
        at java.base/java.util.stream.FindOps$FindSink$OfRef.get(F
```

ofNullable(T)

ASMIQ

```
public class NullableExample
{
    public static void main(final String[] args)
    {
        final Stream<Customer> customerStream1 = Stream.ofNullable(findCustomer());
        System.out.println(customerStream1.count());

        final Stream<Customer> customerStream2 = Stream.ofNullable(findCustomer());
        final Optional<Customer> optFirst = customerStream2.findFirst();
        optFirst.ifPresentOrElse(System.out::println,
                                 () -> System.out.println("No element"));
    }

    private static Customer findCustomer()
    {
        return null;
    }
}
```

ofNullable(T)

```
public class NullableExample
{
    public static void main(final String[] args)
    {
        final Stream<Customer> customerStream1 = Stream.ofNullable(findCustomer());
        System.out.println(customerStream1.count());

        final Stream<Customer> customerStream2 = Stream.ofNullable(findCustomer());
        final Optional<Customer> optFirst = customerStream2.findFirst();
        optFirst.ifPresentOrElse(System.out::println,
                                 () -> System.out.println("No element"));
    }

    private static Customer findCustomer()
    {
        return null;
    }
}
```

```
<terminated> NullableExample (1) [Java Application] /Library/Java/JavaVirtualMachi
0
No element
```

Additions to Stream API in JDK 9

takeWhile(…)

dropWhile(…)

ofNullable(…)

iterate(…, … , …)

Additions to Stream API in JDK 9

ASMIQ

- `iterate(T, Predicate<? super T>, UnaryOperator<T>)` – **Creates a stream<T> with the given start value. The following values are calculated by the UnaryOperator<T> as long as the passed predicate<T> is fulfilled.**

```
final IntStream stream = IntStream.iterate(1, n -> n < 10, n -> n + 1);

System.out.println(stream.mapToObj(num -> "" + num).collect(joining(", ")));
```

- **=> 1, 2, 3, 4, 5, 6, 7, 8, 9**

- **Rather analog to for-loop :**

```
for (int n = 1; n < 10; n++)
iterate(1, n -> n < 10, n -> n + 1);
```

- **Since it's a stream, there are a variety of other possibilities.**

Stream API – Collectors

Stream API Collectors in JDK 9

ASMIQ

The extensive Stream API has a lot of collectors:

- **`toCollection(), toList(), toSet()` … – Collects the elements of the stream into the corresponding collection.**
- **`joining()` – Merge elements into a string**
- **`groupingBy()` – to prepare groupings. for example: histograms. Further collectors could also be used there (downstream collectors)**

In the context of `groupingBy()`, however, there are some special use cases for which there was no collector before Java 9.

Stream API Collectors in JDK 9

ASMIQ

Two newly available collectors

- `filtering()` – **Filtering the elements of the stream**
- `flatMapping()` – **Mapping and merging elements**

⇒ **Both are especially useful in the context of `groupingBy()`.**

⇒ **Let's first look at simple examples ...**

Stream API Collectors in JDK 9

ASMIQ

The new `Collectors.filtering()` with analogy `filter()`

Example for the entry:

```
final Set<String> result1 = programmming1.filter(name -> name.contains("Java")).
                                          collect(toSet());
```

With new collector:

```
final Set<String> result2 = programmming2.collect(
                            filtering(name -> name.contains("Java"), toSet()));
```

As a result:

```
[JavaFX, Java, JavaScript]
```

Second variant less intuitive and understandable than the first. Advantage only with `groupingBy()`

ASMIQ

Example for the entry:

Suppose we wanted to create some kind of histogram based on the programming languages and frameworks. Let's start with a Java-8 implementation:

```
final List<String> programming = List.of("Java", "JavaScript", "Groovy", "JavaFX",
                                          "Spring", "Java");

final Map<String, Long> result = programming.stream().
                                 filter(name -> name.contains("Java")).
                                 collect(groupingBy(name -> name, counting()));
System.out.println(result);
```

As a result:

```
{JavaFX=1, Java=2, JavaScript=1}
```

What do we do if during histogram preparation entries are also of interest that do not meet the condition?

Changed requirement: If entries that do not meet the condition are also of interest during histogram preparation, they would be lost if they were filtered beforehand. For our application, we must first group and then filter and only count those that are relevant:

```
final Map<String, Long> result = programming.stream().
                                 collect(groupingBy(name -> name,
                                         filtering(name -> name.contains("Java"),
                                                   counting())));
System.out.println(result);
```

As a result:

```
{JavaFX=1, Java=2, JavaScript=1, Spring=0, Groovy=0}
```

Stream API Collectors in JDK 9

ASMIQ

The new `Collectors.flatMapping()` with analogy `Collectors.mapping()` / `Stream.flatMap()`

Goal: All Hobbies as (one) set:

```
final Stream<Set<String>> lotsOfHobbies = Stream.of(Set.of("Java", "Movies"),
                                                    Set.of("Skating", "Tennis"),
                                                    Set.of("Karate", "Movies"),
                                                    Set.of("Music", "Movies"));
final Set<Set<String>> result =
                        lotsOfHobbies.collect(mapping(entry -> entry, toSet()));
System.out.println("Hobbies: " + result);
```

As a result:

```
Hobbies: [[Skating, Tennis], [Music, Movies], [Karate, Movies], [Java, Movies]]
```

This first variant delivers unwanted nesting!

ASMIQ

Goal: All Hobbies as one set:

```
final Stream<Set<String>> lotsOfHobbies = Stream.of(Set.of("Java", "Movies"),
                                                    Set.of("Skating", "Tennis"),
                                                    Set.of("Karate", "Movies"),
                                                    Set.of("Music", "Movies"));

final Set<String> result = lotsOfHobbies.collect(
                           flatMapping(value -> value.stream(), toSet()));
System.out.println("Hobbies: " + result);
```

As a result:

```
Hobbies: [Java, Tennis, Karate, Music, Movies, Skating]
```

This second variant is a bit less intuitive because of the call to `stream()`. Again advantage only with `groupingBy()`

Changed requirement and more practical example:
From a set of people with hobbies, we should group all those with the same firstname and create a collection of hobbies for each firstname:

```
final Stream<Map.Entry<String,Set<String>>> personsToHobbies =
                  Stream.of(Map.entry("Peter", Set.of("Groovy", "Movies")),
                            Map.entry("Peter", Set.of("Java", "Skating")),
                            Map.entry("Mike", Set.of("Java")));

final Map<String, Set<String>> collected = personsToHobbies.collect(
                                 groupingBy(entry -> entry.getKey(),
                                 flatMapping(entry -> entry.getValue().stream(),
                                             toSet())));
System.out.println(collected);
```

As a result:

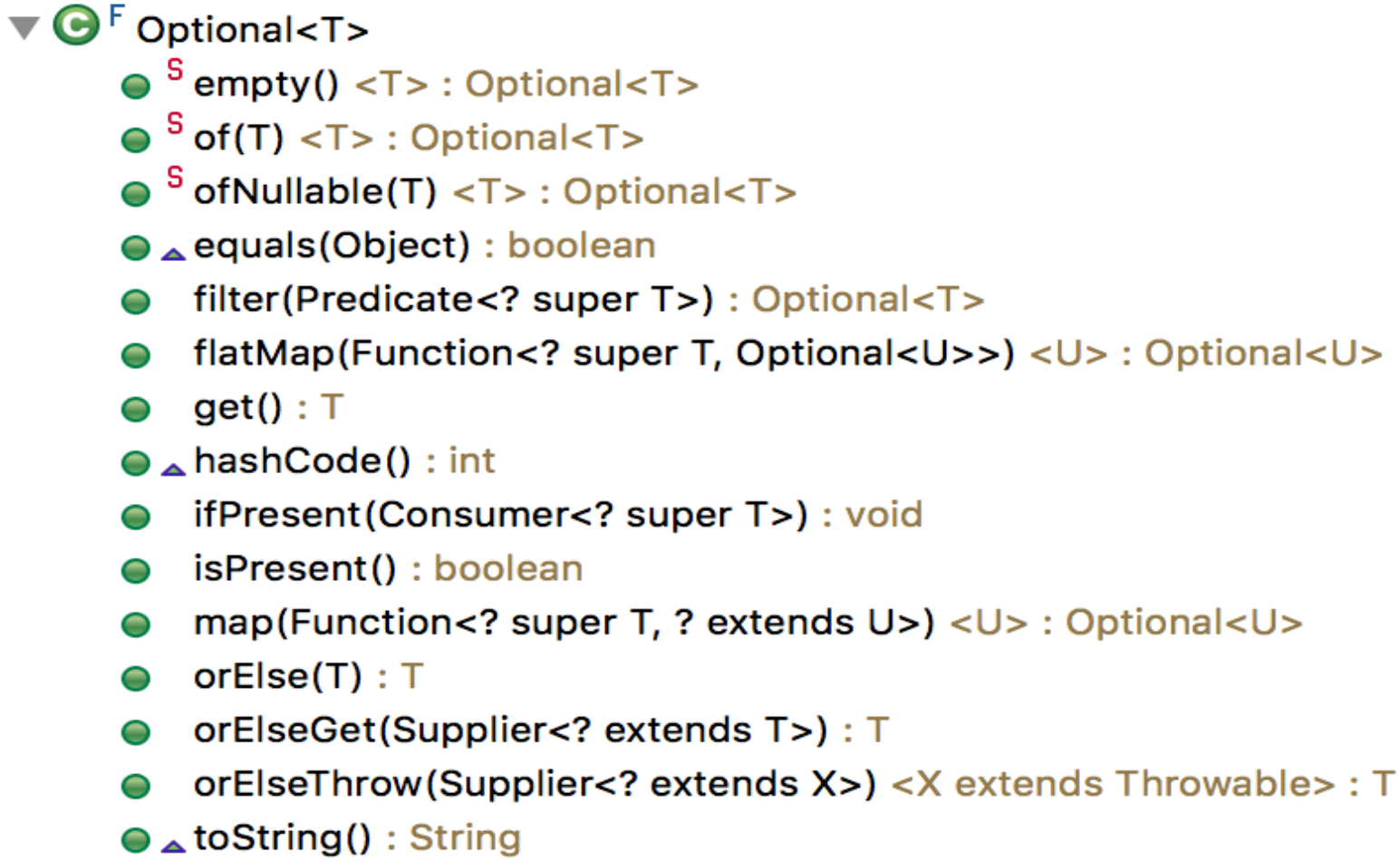
```
{Mike=[Java], Peter=[Java, Movies, Groovy, Skating]}
```

Optional<T>

Optional<T>

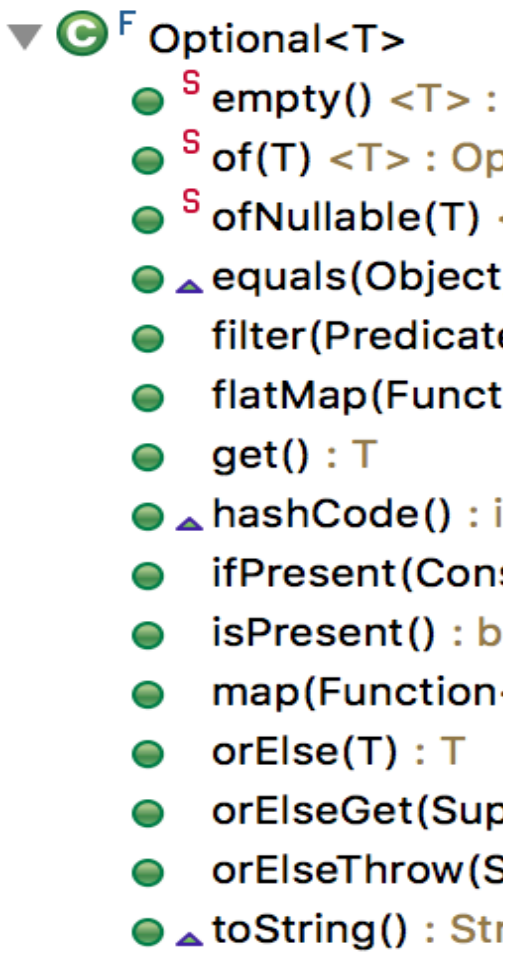
ASMIQ

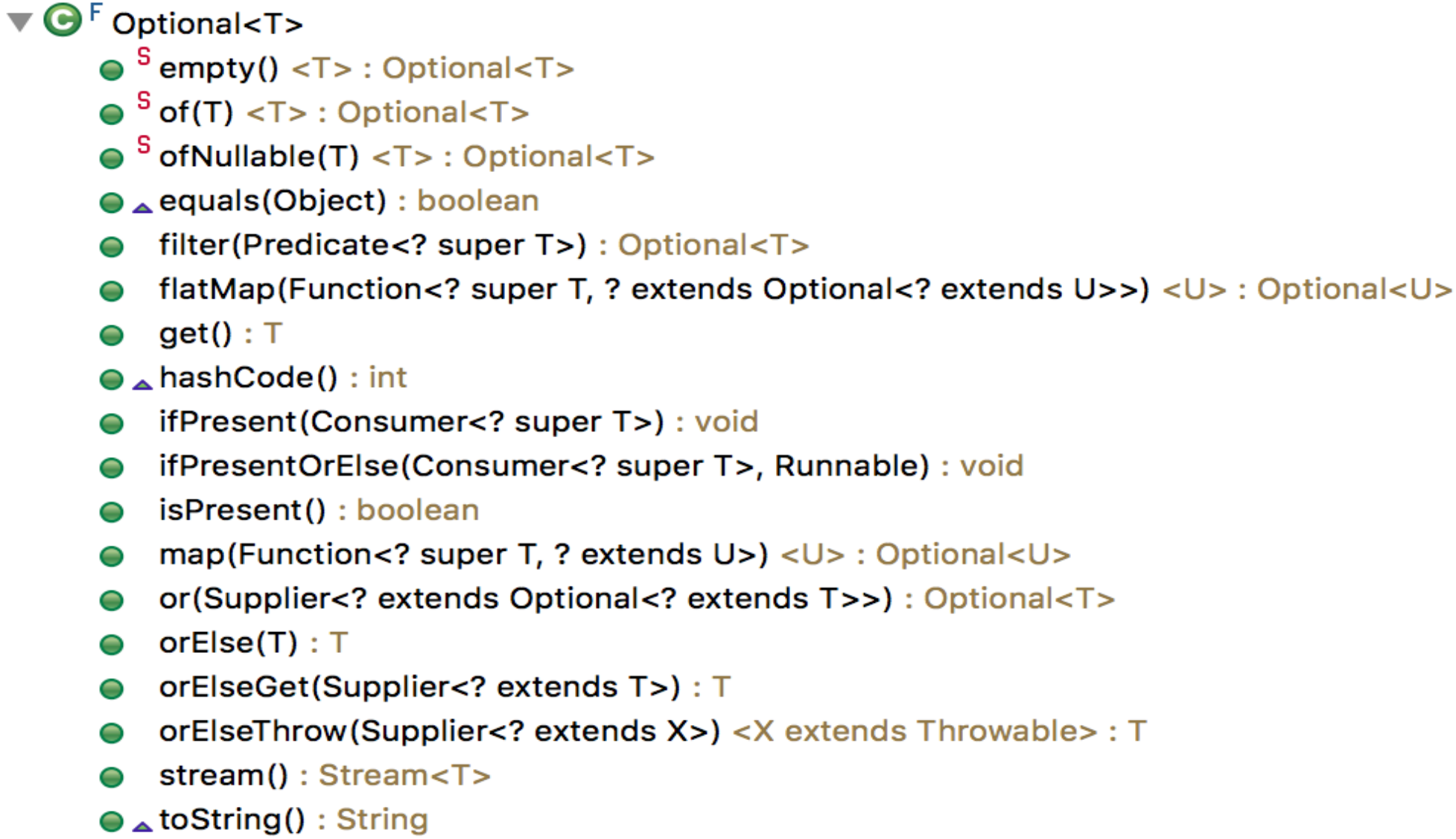
- **Was introduced with Java 8 and facilitates the processing and modelling of optional values**



Optional<T> in JDK 8 / JDK 9

ASMIQ





Optional<T>

ASMIQ

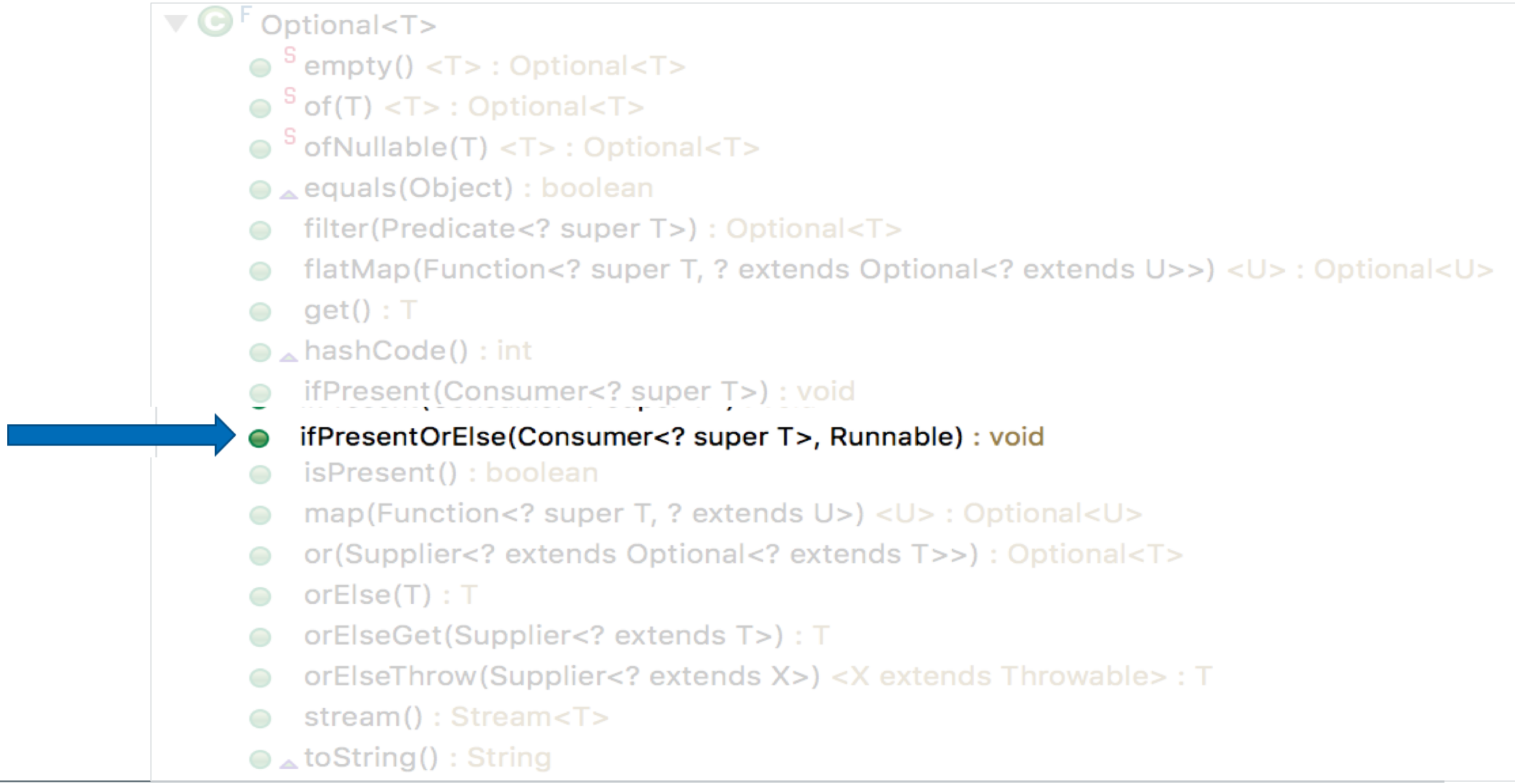
- **Initially with sound API, but 3 weak points for the following usages:**
  - **The execution of actions even in a negative case,**
  - **The combination of the results of several calculations that provide `Optional<T>`.**
  - **Conversion into a `Stream<T>`, for compatibility with the stream API, e.g. for frameworks working on streams**

Class Optional<T>

ASMIQ

- The enhancements to the `Optional<T>` class in JDK 9 address all three of the vulnerabilities listed before. The following methods are used for this purpose:
  - `ifPresentOrElse(Consumer<? super T>, Runnable)` – Allows an action to be executed in a positive or negative case.
  - `or(Supplier<Optional<T>> supplier)` – Elegantly combines multiple calculations.
  - `stream()` – converts the Optional<T> into a Stream<T>.

Optional<T> in JDK 9



why ifPresentOrElse(…) ?

ASMIQ

JDK 8

```
public class Example1 {

    public static void main(String[] args) {

        Optional<String> welcomeString = getWelcomeString();

        welcomeString.ifPresent(System.out::println);

    }

    private static Optional<String> getWelcomeString() {
        return Optional.of("Hi JUG Zürich");
    }

}
```

why ifPresentOrElse(…) ?

ASMIQ

JDK 8

```
public class Example1 {

    public static void main(String[] args) {

        Optional<String> welcomeString = getWelcomeString();

        if(welcomeString.isPresent()) {
            System.out.println(welcomeString.get());
        }else {
            System.out.println("Hi JUG Default ");
        }

    }

    private static Optional<String> getWelcomeString() {
        return Optional.of("Hi JUG Zürich");
    }

}
```

ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)

ASMIQ

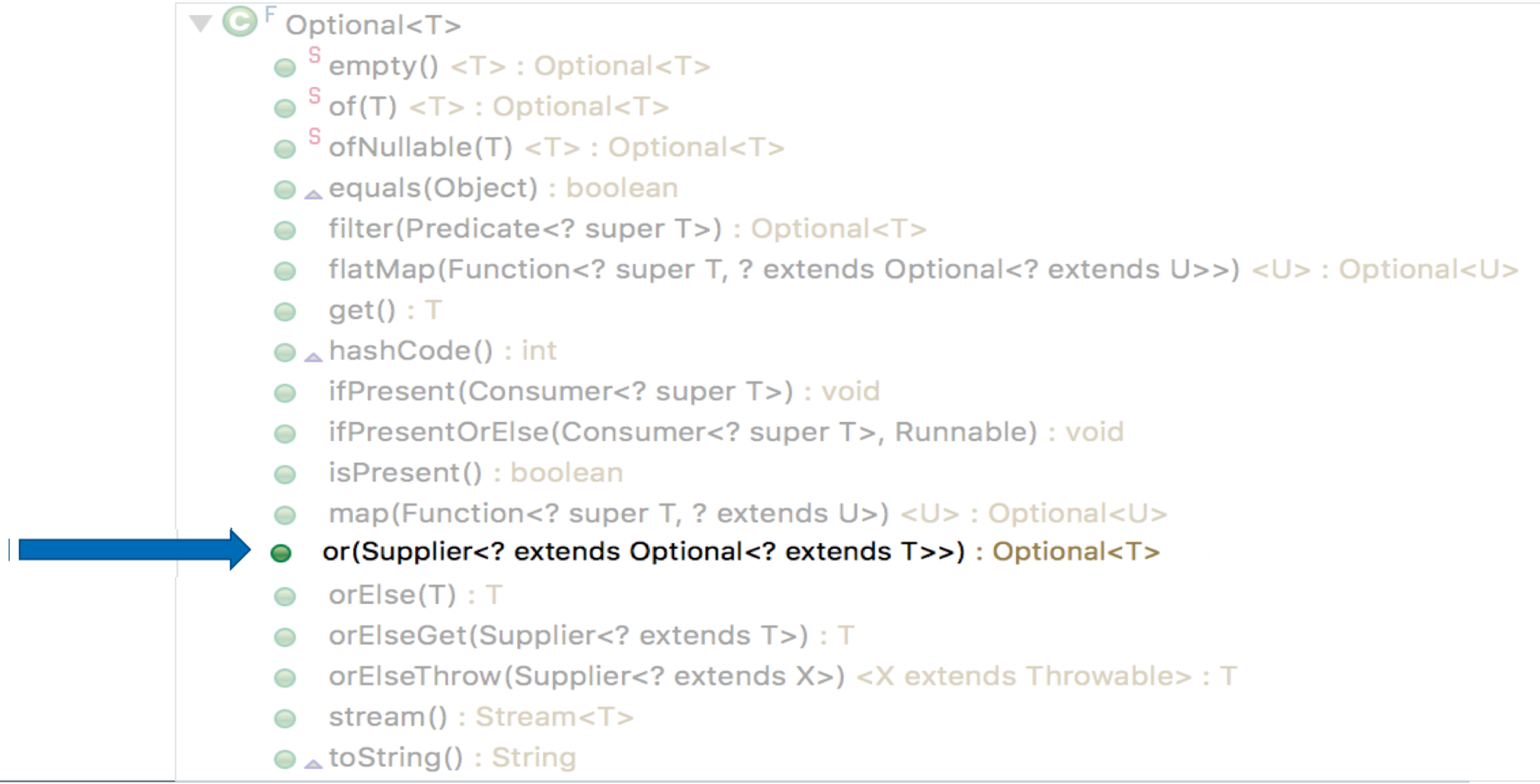
With Java 9 and the method `ifPresentOrElse()` the result evaluation of searches / actions can often be simplified:

JDK 9

```
public class Example1 {

public static void main(String[] args) {

        Optional<String> welcomeString = getWelcomeString();

        welcomeString.ifPresentOrElse(System.out::println, () -> System.out.println("Hi JUG XXX"));

    }

    private static Optional<String> getWelcomeString() {
        return Optional.of("Hi JUG Zürich");
    }


}
```

Optional<T> in JDK 9

ASMIQ



why or(…) ?

ASMIQ

JDK 8

```
public class PaymentProcessor {

    public static void main(String[] args) {
        Double balance = 0.0;
        Optional<Double> debitCardBalance = getDebitCardBalance();
        Optional<Double> creditCardBalance = getCreditCardBalance();
        Optional<Double> payPalBalance = getPayPalBalance();

        if (debitCardBalance.isPresent()) {

            balance = debitCardBalance.get();

        } else if (creditCardBalance.isPresent()) {

            balance = creditCardBalance.get();

        } else if (payPalBalance.isPresent()) {

            balance = payPalBalance.get();

        }

        if(balance > 0)
            System.out.println("Payment will be processed...");
        else
            System.out.println("Sorry insufficient balance");

    }
```

or(…)

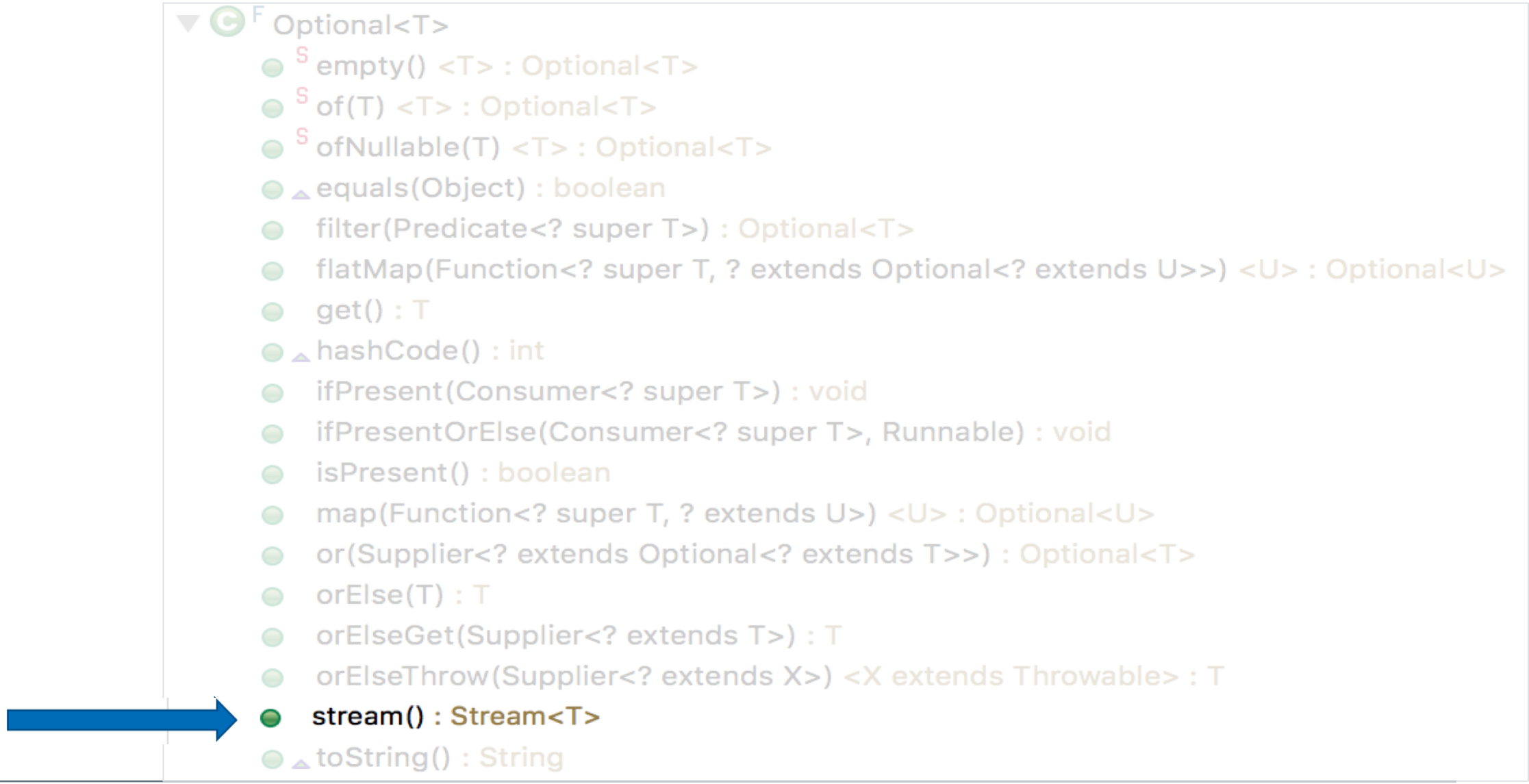
JDK 9

```
Optional<Double> optBalance = getDebitCardBalance().
                              or(()->getCreditCardBalance()).
                              or(()->getPayPalBalance());
```

```
optBalane.ifPresentOrElse(value -> System.out.println("Payment " + value +
                                                      " will be processed ...");,
                          () -> System.out.println("Sorry, insufficient balance"));
```

- `or(Supplier<Optional<T>> supplier)` seems insignificant at first glance
- But call chains can be described with fallback strategies in a readable and understandable way, as the above example impressively shows.

Optional<T> in JDK 9



why stream() ?

ASMIQ

JDK 8

```
public class CityPrinter {

    public static void main(String[] args) {

        Stream<Optional<String>> optionalCityNames = Stream.of(Optional.of("Zurich"), Optional.of("Bern"),
                                                               Optional.of("Basel"), Optional.empty());

        optionalCityNames.filter(Optional::isPresent).map(Optional::get).forEach(System.out::println);

    }

}
```

stream()

ASMIQ

- `Optional<T>` => `Stream<T>`: This is useful if you have a stream of optional values and want to keep only those entries with valid values (combination of methods `flatMap()` and `stream()`).

- Example of a stream consisting of `Optional<String>` elements, for example as a result of a parallel search. At the end, the results should be consolidated:

JDK 9

```
public class CityPrinter {

    public static void main(String[] args) {

        Stream<Optional<String>> optionalCityNames = Stream.of(Optional.of("Zurich"), Optional.of("Bern"),
                                                               Optional.of("Basel"), Optional.empty());


        optionalCityNames.flatMap(Optional::stream).forEach(System.out::println);
    }

}
```

Collection Factory Methods

Collection Factory Methods Intro

ASMIQ

- **Creating collections for a (smaller) set of predefined values can be a bit inconvenient in Java:**

```
// Variante 1
final List<String> oldStyleList1 = new ArrayList<>();
oldStyleList1.add("item1");
oldStyleList1.add("item2");

// Variante 2
final List<String> oldStyleList2 = Arrays.asList("item1", "item2");
```

- **Languages like Groovy or Python offer a special syntax for this, so-called collection literals ...**

Collection Factory Methods Intro

List<String> names = [“MAX“,“Moritz“,“Tim“];
Map<String, Integer> nameToAge = { “Tim“ : 45,“Tom“ : 23 };

```
newStyleList[0] = "newValue";
final String valueAtPos0 = newStyleList[0]; // => newValue
```

Collection Literals


```
List<String> names = [“MAX“,“Moritz“,“Tim“ ];
Map<String, Integer> nameToAge = { “Tim“ : 45,“Tom“ : 23 };
```

Already in 2009, people thought about such things for Java.
Unfortunately this was not realized until now...

ASMIQ

Collection Literals **LIGHT** a.k.a Collection Factory Methods

```
final List<String> names = List.of("Tim", "Tom", "Mike");
final Set<Integer> numbers = Set.of(1, 3, 4, 2);

final Map<Integer, String> mapping = Map.of(5, "five", 6, "six", 7, "seven");
final Map<Integer, String> mapping2 = Map.ofEntries(entry(5, "five"),
                                                    entry(6, "six"),
                                                    entry(7, "seven"));
```

Collection Factory Methods

ASMIQ

- **Behavior quite intuitive for lists ...**

```
List<String> names = List.of("MAX", "MORITZ", "MIKE");
names.forEach(name -> System.out.println(name));

MAX
MORITZ
MIKE
```

Collection Factory Methods

ASMIQ

- **Behavior quite strange for sets ...**

```
Set<String> names2 = Set.of("MAX", "MORITZ", "MAX");
names2.forEach(name -> System.out.println(name));


Exception in thread "main" java.lang.IllegalArgumentException:
duplicate element: MAX
    at java.util.ImmutableCollections$SetN.<init>(java.base@9-ea/
ImmutableCollections.java:329)
    at java.util.Set.of(java.base@9-ea/Set.java:500)
    at jdk9example.Jdk9Example.main(Jdk9Example.java:31)
```

Collection Factory Methods -- Specialities for construction

ASMIQ

```
static <E> List<E> of() {
    return ImmutableCollections.List0.instance();
}

static <E> List<E> of(E e1) {
    return new ImmutableCollections.List1<>(e1);
}

...

@SafeVarargs
@SuppressWarnings("varargs")
static <E> List<E> of(E... elements) {
    switch (elements.length) {
        case 0:
            return new ImmutableCollections.List0<>();
        case 1:
            return new ImmutableCollections.List1<>(elements[0]);
        case 2:
            return new ImmutableCollections.List2<>(elements[0], elements[1]);
        default:
            return new ImmutableCollections.ListN<>(elements);
    }
}
```

Exercises PART 1

https://github.com/Michaeli71/JAX-London-Best-of-Java9-13.git

ASMIQ

PART 2: Multi-Threading and Reactive Streams

- CompletableFuture<T>
- Flow-API

CompletableFuture

Multi-Threading and the Class CompletableFuture<T>

ASMIQ

- Since Java 5 there is the interface Future<T> in the JDK, but it often leads to blocking code by its method `get()`.
- Since JDK 8 `CompletableFuture<T>` helps with the definition of asynchronous calculations
- Describing processes, enabling parallel execution
- Actions on higher semantic level than with `Runnable` or `Callable<T>`

Introduction to CompletableFuture<T>

ASMIQ

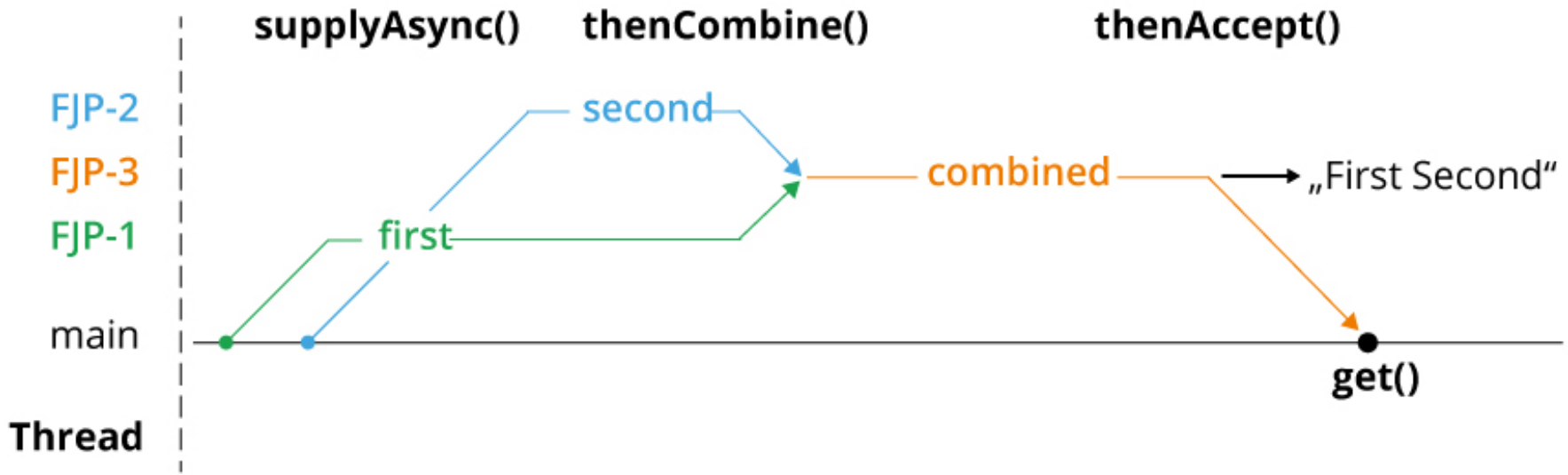
- Basic steps
  - **supplyAsync**(Supplier<T>) => Define calculations
  - **thenApply**(Function<T,R>) => Process result of calculation
  - **thenAccept**(Consumer<T>) => Process result, but without return
  - **thenCombine**(…) => Merge processing steps

- Example

```
CompletableFuture<String> firstTask = CompletableFuture.supplyAsync(() -> "First");
CompletableFuture<String> secondTask = CompletableFuture.supplyAsync((() -> "Second"));

CompletableFuture<String> combined = firstTask.thenCombine(secondTask,
                                                           (f, s) -> f + " " + s);

combined.thenAccept(System.out::println);
System.out.println(combined.get());
```

Introduction to CompletableFuture<T>

```
CompletableFuture<String> firstTask = CompletableFuture.supplyAsync(() -> "First");
CompletableFuture<String> secondTask = CompletableFuture.supplyAsync((() -> "Second"));

CompletableFuture<String> combined = firstTask.thenCombine(secondTask, (f, s) -> f + " " + s);

combined.thenAccept(System.out::println);
System.out.println(combined.get());
```



Example: The following actions are to take place:

- Read data from the server

- Calculate evaluation 1
- Calculate evaluation 2
- Calculate evaluation 3

- Merge results into a dashboard

How could a first realization look like?

Multi-Threading and the Class CompletableFuture<T>

ASMIQ

- **Read data from the server => `retrieveData()`**
- **Calculate evaluation 1 => `processData1()`**
- **Calculate evaluation 2 => `processData2()`**
- **Calculate evaluation 3 => `processData3()`**
- **Combine results in the form of a dashboard => `calcResult()`**

- **Simplifications: Data => List of strings, calculations => Result long => String**

```
public void execute()
{
    final List<String> data = retrieveData();
    final Long value1 = processData1(data);
    final Long value2 = processData2(data);
    final Long value3 = processData3(data);
    final String result = calcResult(value1, value2, value3);
    System.out.println("result: " + result);
}
```

Multi-Threading and the Class CompletableFuture<T>

```
public void execute()
{
    final List<String> data = retrieveData();
    final Long value1 = processData1(data);
    final Long value2 = processData2(data);
    final Long value3 = processData3(data);
    final String result = calcResult(value1, value2, value3);
    System.out.println("result: " + result);
}
```

However, the calculations in the example are very simplified ...

- **no parallelism**
- **no coordination of tasks (works because it is synchronous)**
- **no exception handling**

- **We avoid the trouble and hardly understandable and unmaintainable variants with Runnable, Callable<T>, Thread-Pools, ExecutorService etc., because this itself is often still complicated and error-prone.**

What must be changed for parallel processing with `CompletableFuture<T>`?

Multi-Threading and the Class CompletableFuture<T>

ASMIQ

```
public void execute() throws InterruptedException, ExecutionException
{
    final CompletableFuture<List<String>> cfData =
                                          CompletableFuture.supplyAsync(()->retrieveData());
    // print state
    cfData.thenAccept(System.out::println);

    // execute processing in parallel
    final CompletableFuture<Long> cFValue1 = cfData.thenApplyAsync(data -> processData1(data));
    final CompletableFuture<Long> cFValue2 = cfData.thenApplyAsync(data -> processData2(data));
    final CompletableFuture<Long> cFValue3 = cfData.thenApplyAsync(data -> processData3(data));

    // combine results
    final String result = calcResult(cFValue1.get(), cFValue2.get(), cFValue3.get());
    System.out.println("result: " + result);
}
```

How do we reflect real-world errors?

Multi-Threading and the Class CompletableFuture<T>

ASMIQ

- In the real world, exceptions sometimes occur
- Errors occur from time to time: Network problems, file system access, etc.

- **Assumption: An `IllegalStateException` would be triggered during data retrieval:**

```
Exception in thread "main" java.util.concurrent.ExecutionException:
java.lang.IllegalStateException: retrieveData() failed
at java.base/java.util.concurrent.CompletableFuture.reportGet(CompletableFuture.java:395)
at java.base/java.util.concurrent.CompletableFuture.get(CompletableFuture.java:1999)
```

- **Consequently, all processing would be interrupted and disrupted!**
- A simple integration of exception handling into the process flow is desirable.
- As well as less complicated handling than thread pools or `ExecutorService`

Multi-Threading and the Class CompletableFuture<T>

ASMIQ

- The class `CompletableFuture<T>` provides the method `exceptionally()`

- **Provide a fallback value if the data cannot be determined:**

```
final CompletableFuture<List<String>> cfData =
                        CompletableFuture.supplyAsync(()->retrieveData()).
                        exceptionally((throwable) -> Collections.emptyList());
```

- **Provide a fallback value if a calculation fails:**

```
final CompletableFuture<Long> cFValue2 =
                              cfData.thenApplyAsync(data -> processData2(data)).
                                     exceptionally((throwable) -> 0L);
```

- calculation can be continued even if one or more steps fail.

How do delays reflect the real world?

Multi-Threading and the Class `CompletableFuture<T>`

ASMIQ

- In the real world, access to external resources sometimes causes delays
- In the worst case, an external partner does not answer at all and you might wait indefinitely if a call is blocked.
- Desirable: Calculations should be able to be aborted with time-out
- Assumption: The data `retrieveData()` sometimes takes a few seconds.
- Consequently, the entire processing would be disturbed!

Multi-Threading and the Class `CompletableFuture<T>`

ASMIQ

Since JDK 9: The class `CompletableFuture<T>` provides the methods

- `public CompletableFuture<T> orTimeout(long timeout, TimeUnit unit)`

  => Processing is terminated with an exception if the result was not calculated within the specified time span.

- `public CompletableFuture<T> completeOnTimeout(T value, long timeout, TimeUnit unit)`

  => If the result was not calculated within the specified time span, a default value can be specified.

Multi-Threading and the Class `CompletableFuture<T>`

ASMIQ

Assumption: The data determination sometimes takes several seconds, but should be aborted after 1 second at the latest:

```
final CompletableFuture<List<String>> cfData =
                CompletableFuture.supplyAsync(()->retrieveData()).
                                  orTimeout(1, TimeUnit.SECONDS).
                                  exceptionally((throwable) -> Collections.emptyList());
```

=> The processing can be continued promptly (without much delay or even blocking) even if the data determination should take longer.

Assumption: The calculation sometimes takes several seconds - it should be aborted after 2 seconds at the latest and deliver the result 7:

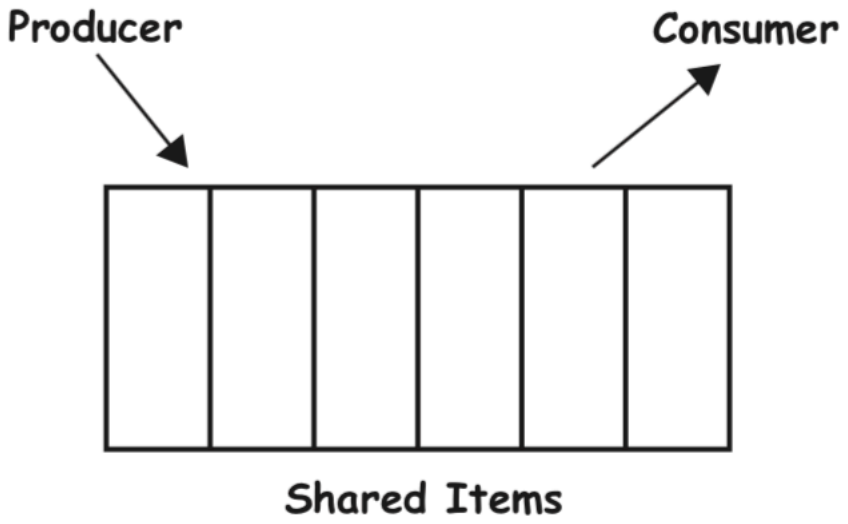
```
final CompletableFuture<Long> cFValue3 =
                              cfData.thenApplyAsync(data -> processData3(data)).
                                     completeOnTimeout(7L, 2, TimeUnit.SECONDS);
```

=> The calculation can be continued with a fallback value, even if one or more steps take longer.

ASMIQ

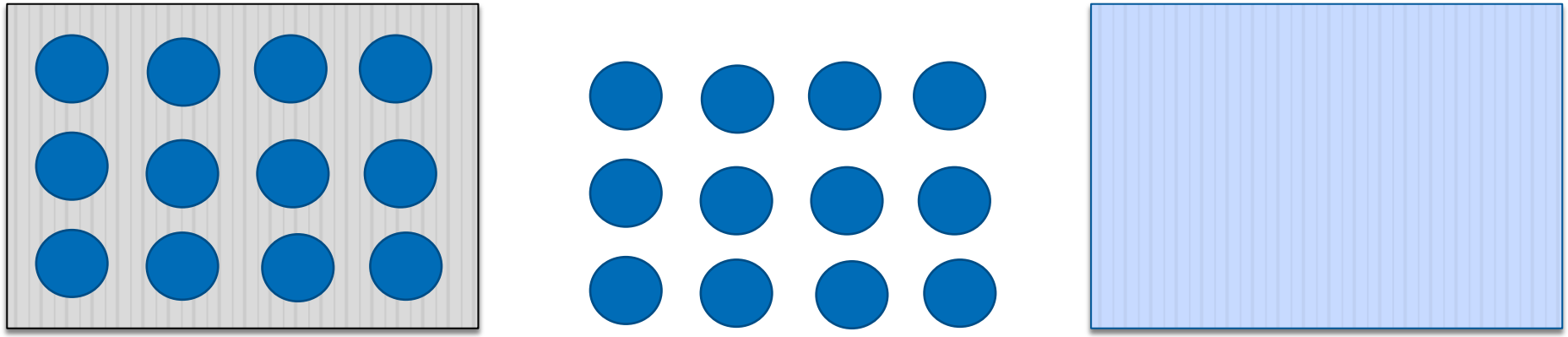
Reactive Streams

Producer - Consumer



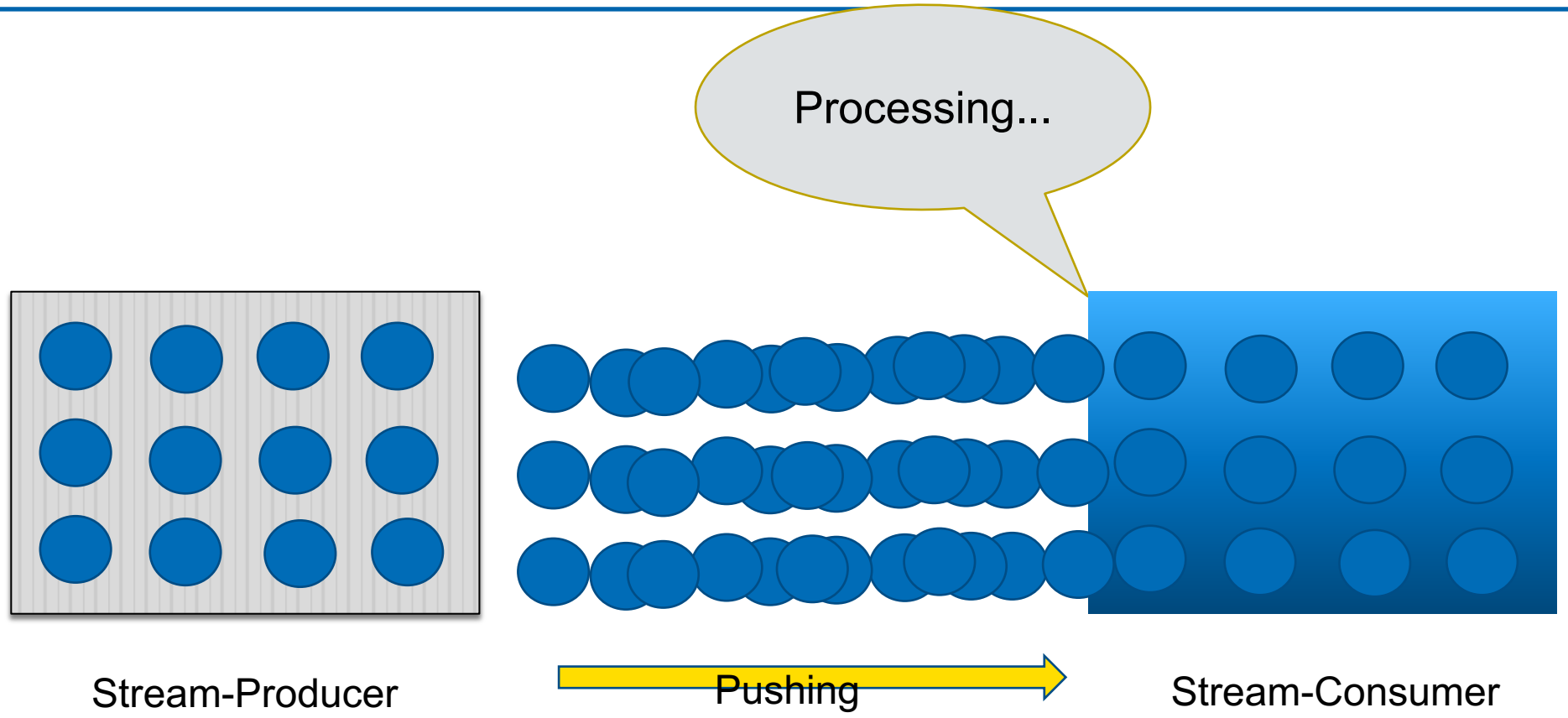
- **Coordination needed**
  - PUSH Producer pushes, Consumer processes directly
  - POLL Consumer polls, Produces creates on demand
  - **Buffering** needed

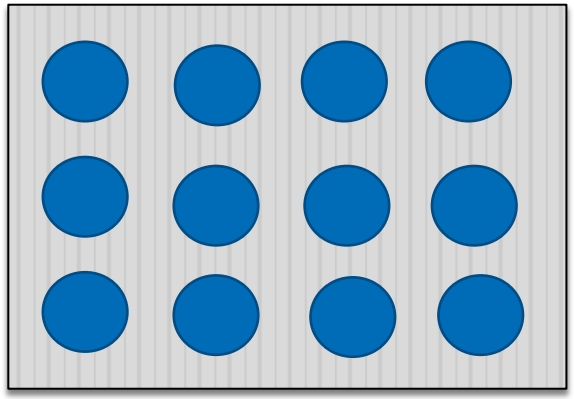
Scenario 1 - Push



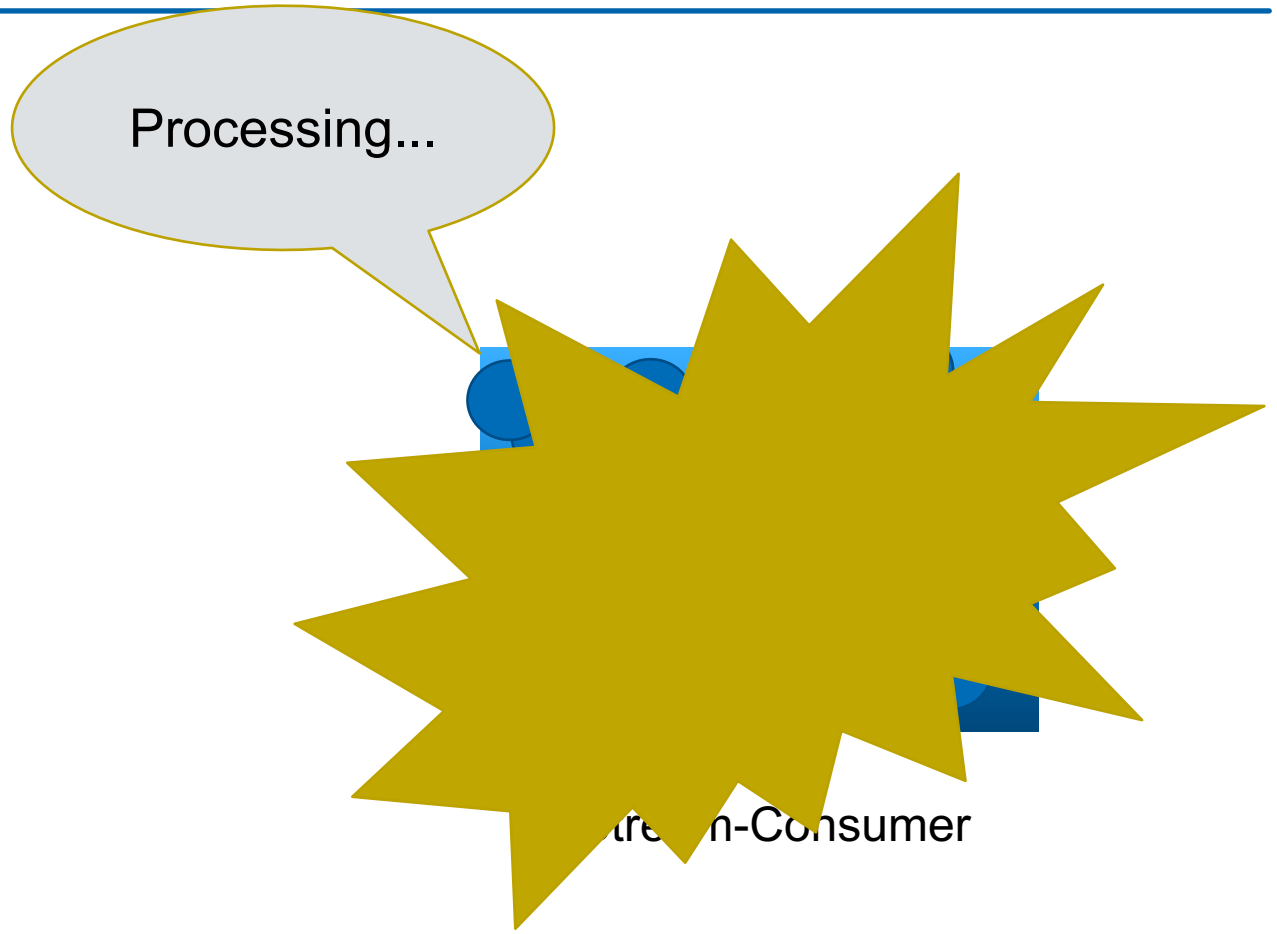
Stream-Producer

Stream-Consumer


Processing...
Stream-Producer
Pushing
Stream-Consumer

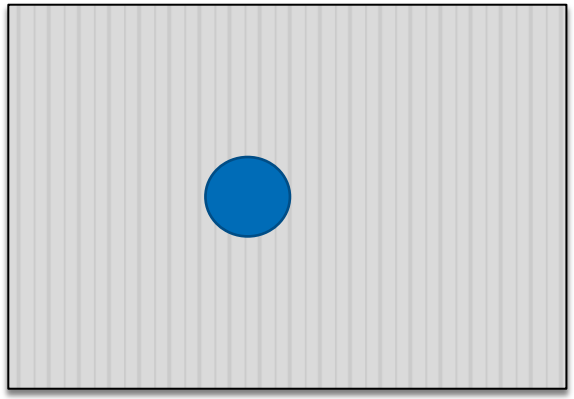


Stream-Producer



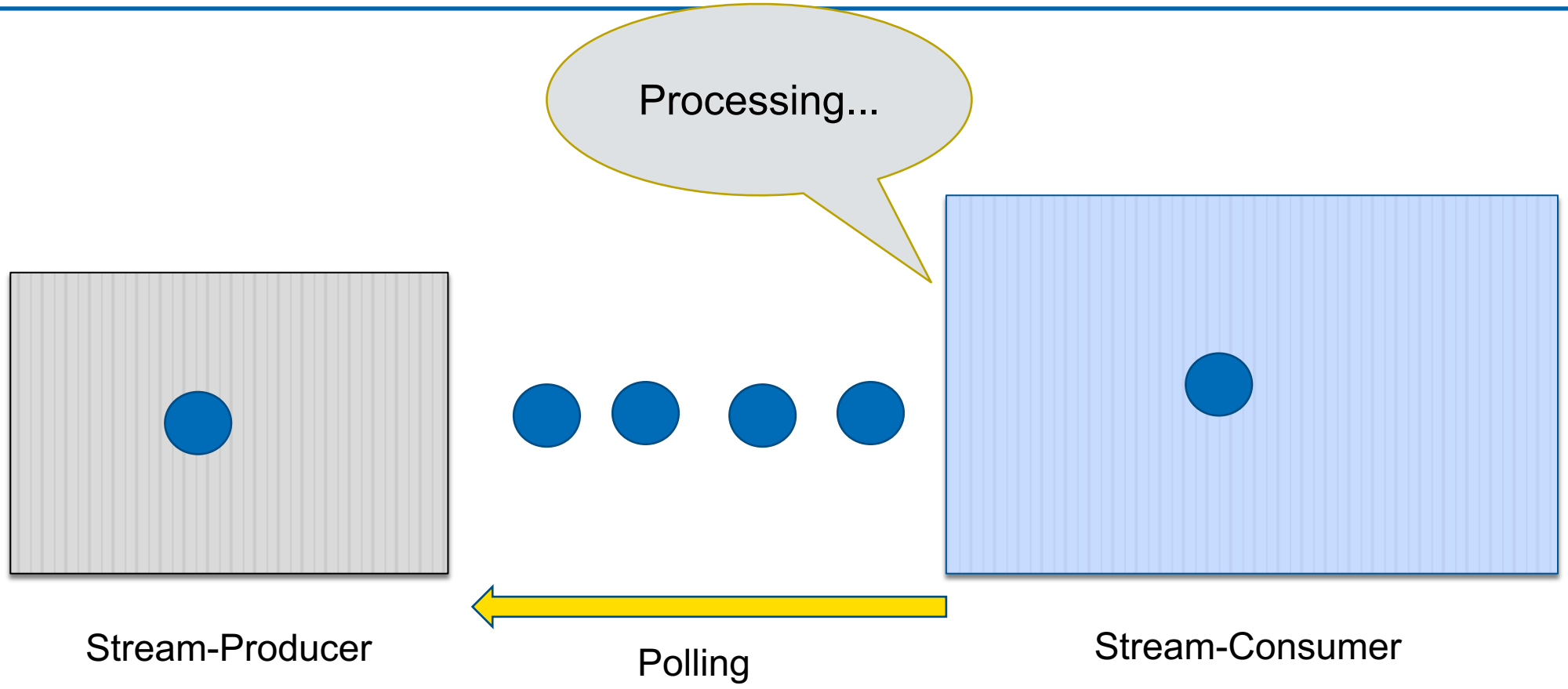
tre n-Consumer

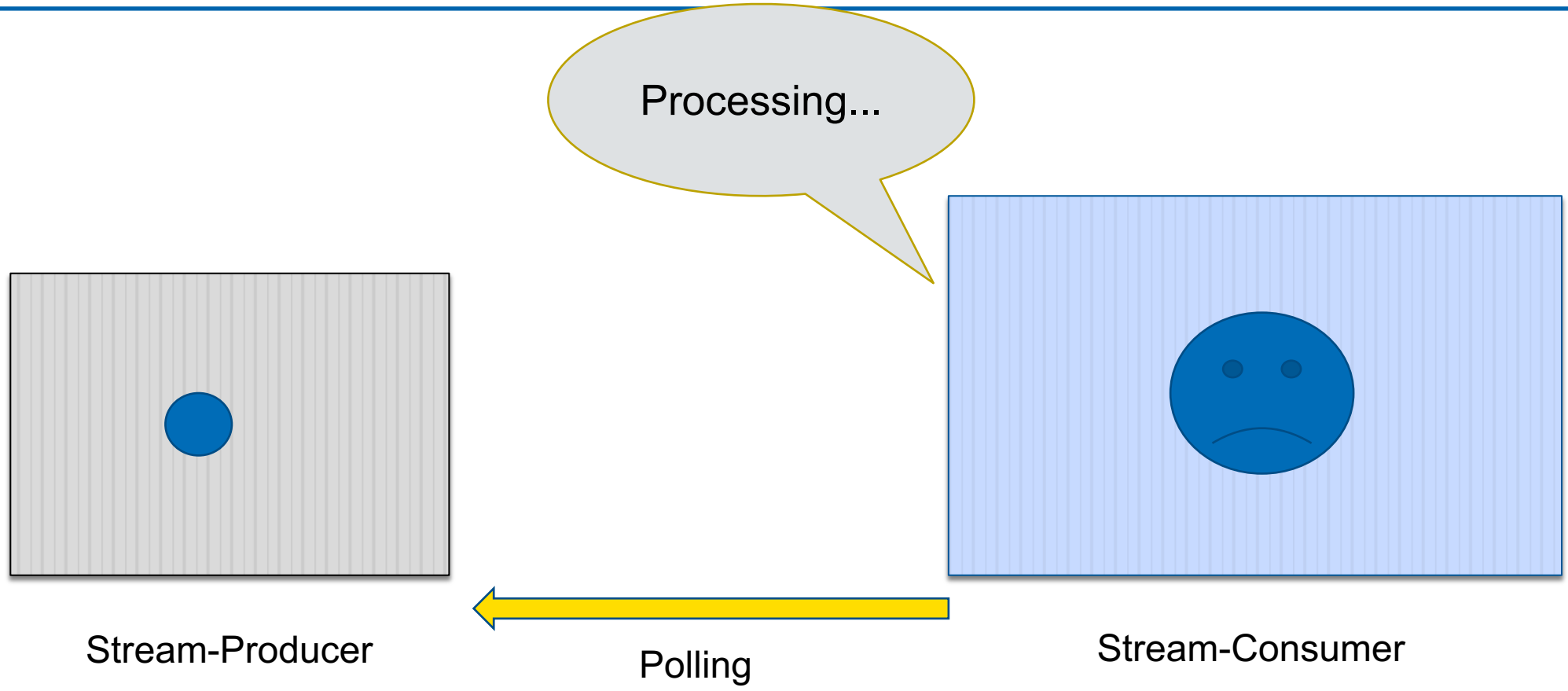
Scenario 2 - Polling



Stream-Producer

Stream-Consumer


Processing...
Stream-Producer
Polling
Stream-Consumer


Processing...
Stream-Producer
Polling
Stream-Consumer

Consequences

ASMIQ

PUSH

1) consumer has to process very fast in sync with the fast producer
2) Or consumer has to buffer at the consumer side
3) Or Prducer has to slow down, but how does he knows this?

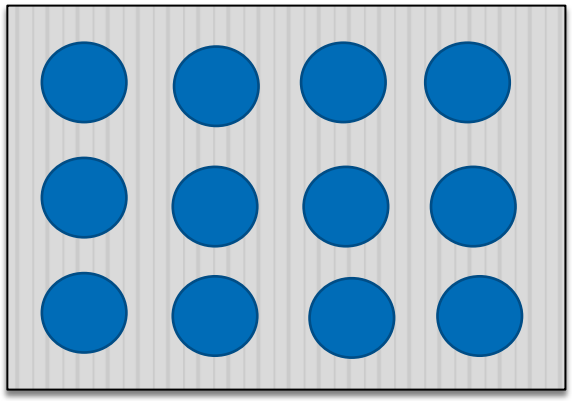
POLL

1) Fast consumer has to wait for slower producer
2) Slow consumer forces the producer to pause or to buffer data

- ✓ One-way communication
  - ✓ Producer -> Consumer (PUSH)
  - ✓ Consumer -> Producer (POLL)

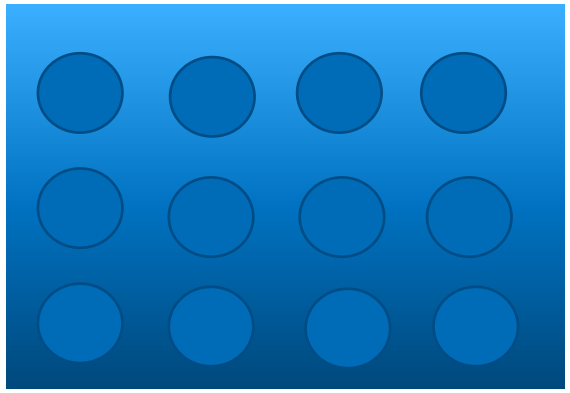
What if ?

Consumer can inform to the producer about it‘s capabilties and Producer can react according to Consumer‘s capabilties and needs.

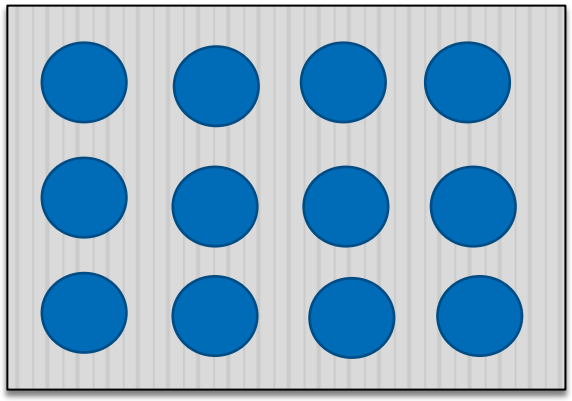


Stream-Producer

Hey I can handle only 12 at a time !

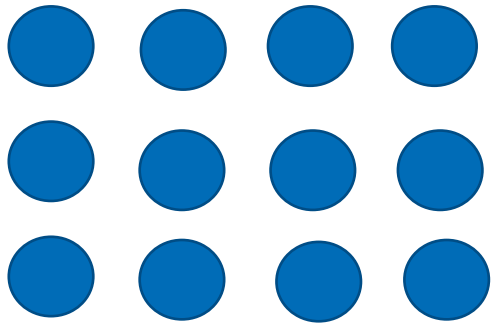


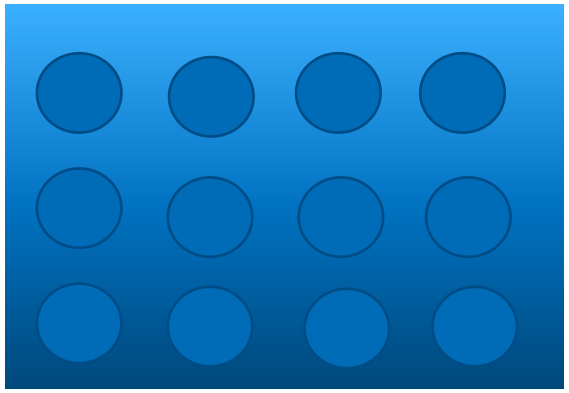
Stream-Consumer



Stream-Producer

Here you go! But ask me if you want more with your new capabilities !





Stream-Consumer

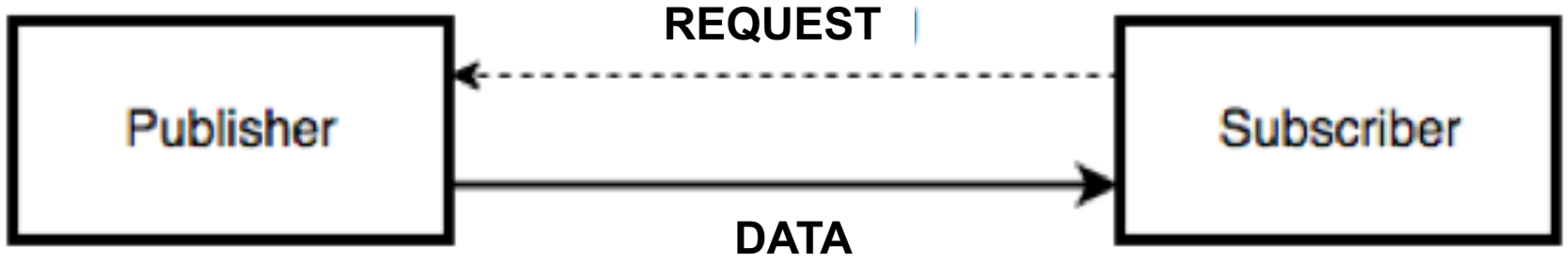
ASMIQ

"… provide a standard for asynchronous stream processing with non-blocking back pressure. … aimed at runtime environments (JVM and JavaScript) .. "

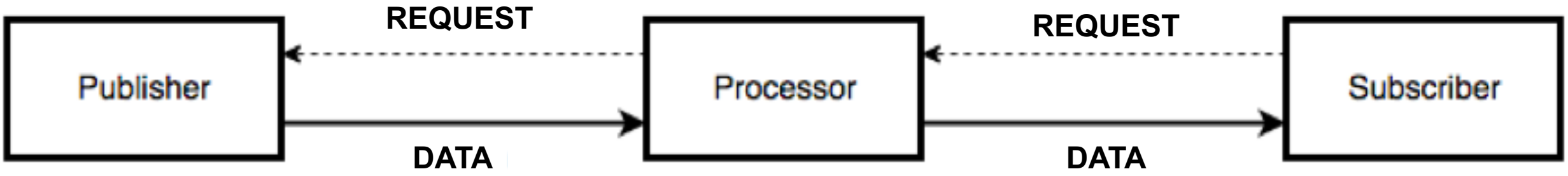
https://www.reactive-streams.org

ASMIQ

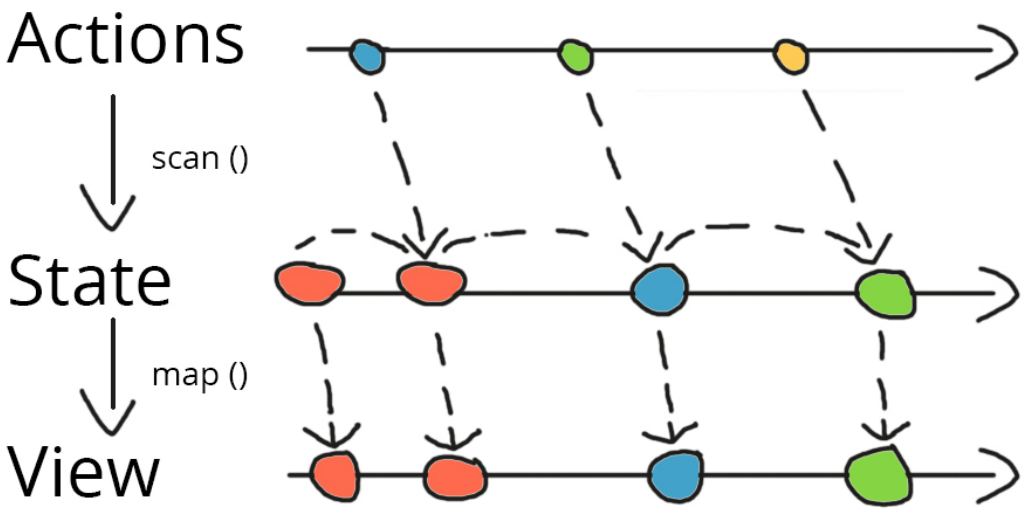
- Publishers are publishing data and subscribers are processing data



- Expansion into a processing chain, similar to a relocation a chain of helpers

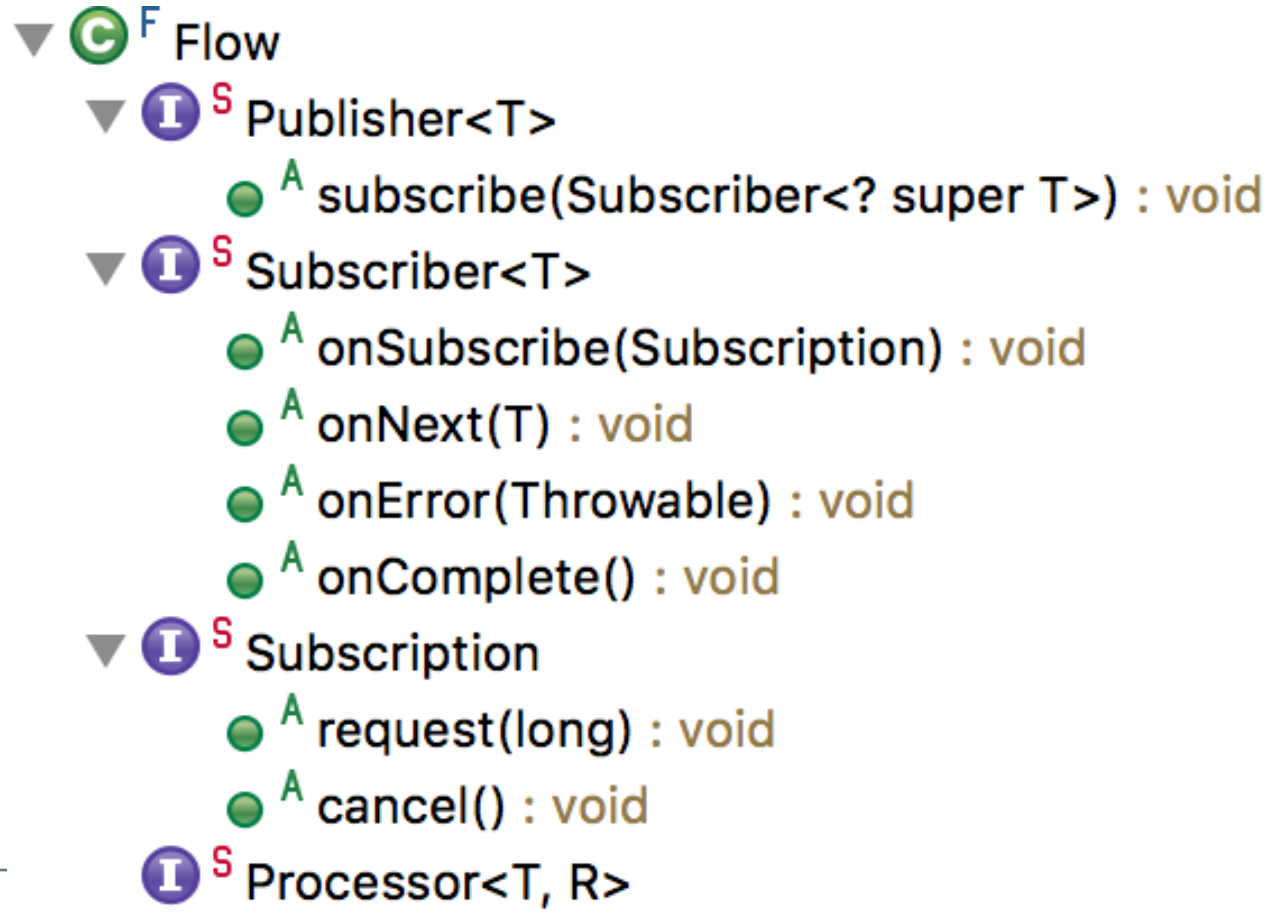


Reactive Streams in the JDK

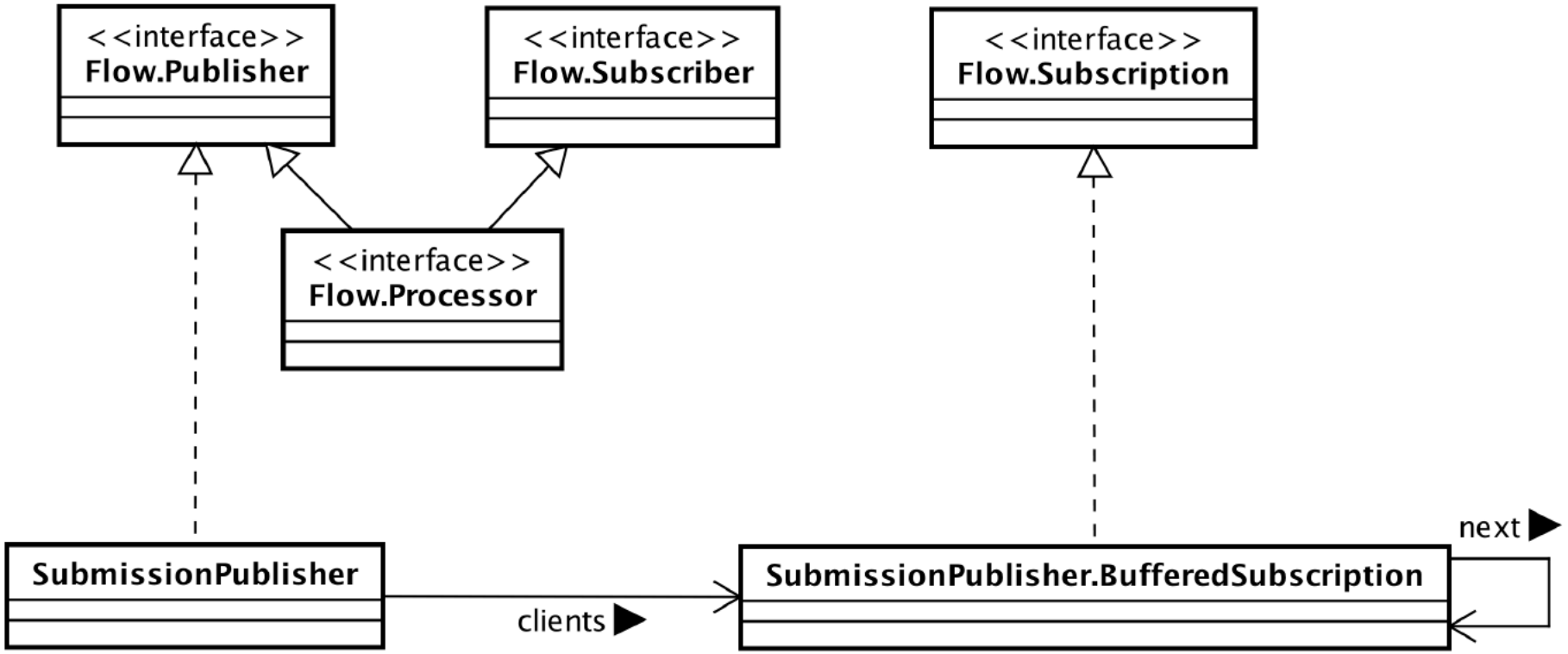


ASMIQ

- Publish Subscribe Framework base on the Flow class and a SubmissionPublisher utility class



Flow API / Reactive Streams API in JDK 9



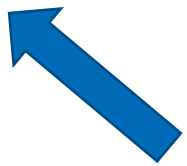
Simple Example

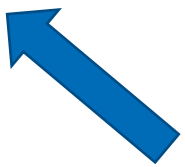
ASMIQ

```
public class SubmissionPublisherExample
{
    public static void main(String[] args) throws InterruptedException
    {
        try (SubmissionPublisher<Integer> publisher =
                                              new SubmissionPublisher<>())

        {
            publisher.subscribe(new ConsoleOutNumberSubscriber());
            System.out.println("Submitting items ...");
            for ( int i = 0; i < 10; i++)
            {
                publisher.submit(i);
            }
            TimeUnit.SECONDS.sleep(1);
        }
    }
}
```

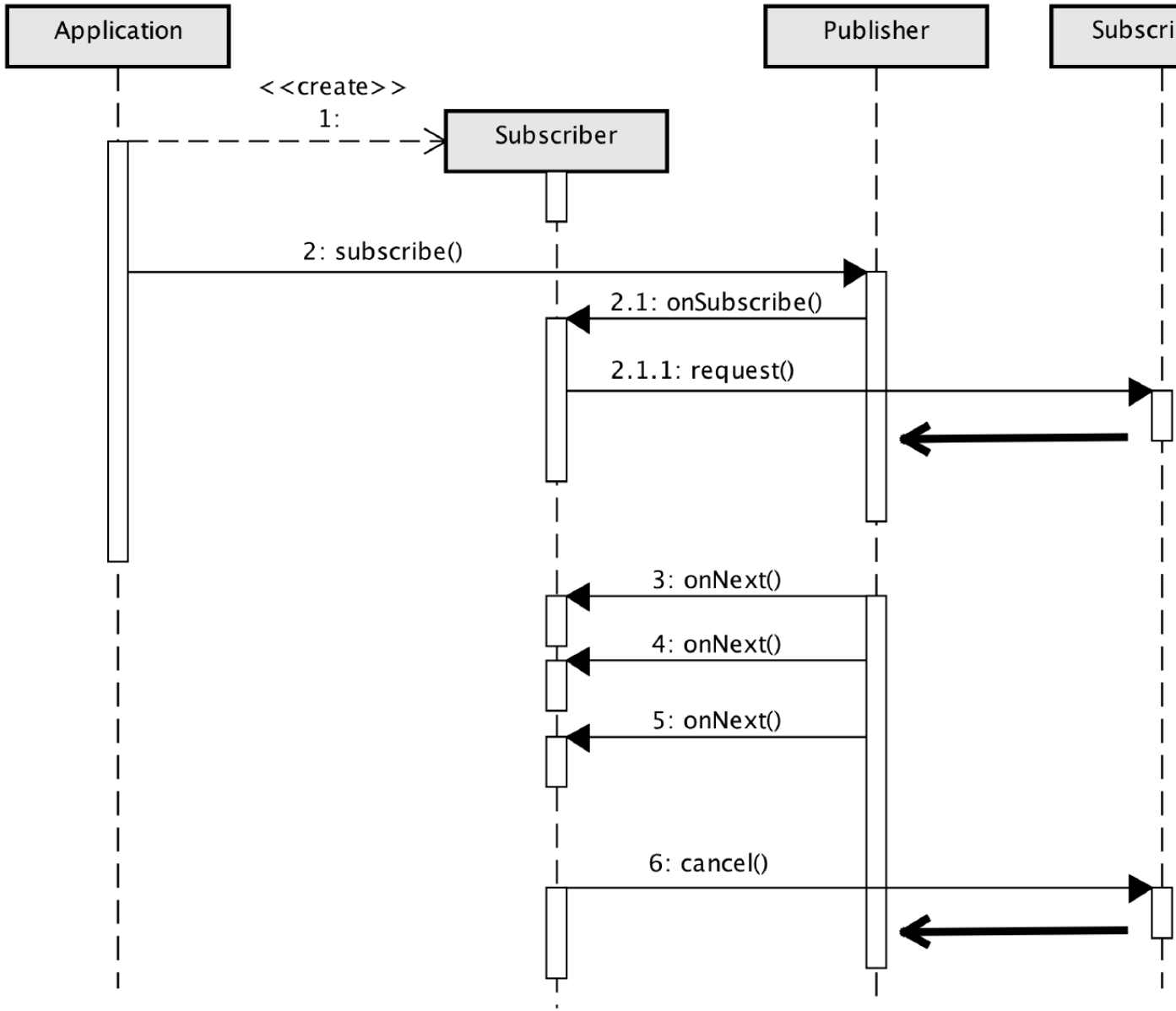
Simple Subscriber

```
public class ConsoleOutNumberSubscriber implements Flow.Subscriber<Integer>
{
    private Flow.Subscription subscription;

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

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

    // onError() / onComplete()
}
```



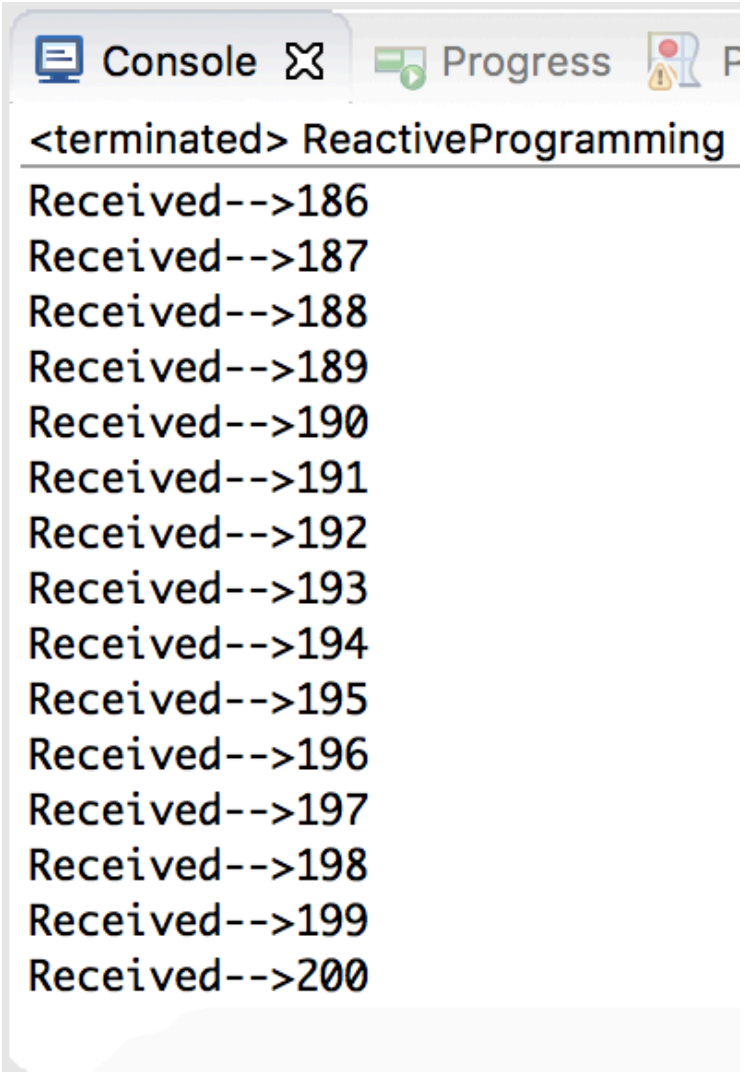


```
Submitting items...
onNext() received item: 0
onNext() received item: 1
onNext() received item: 2
onNext() received item: 3
onNext() received item: 4
onNext() received item: 5
onNext() received item: 6
onNext() received item: 7
onNext() received item: 8
onNext() received item: 9
onComplete()
```


Application
Publisher
Subscription
<<create>>
1:
Subscriber
2: subscribe()
2.1: onSubscribe()
2.1.1: request()
3: onNext()
4: onNext()
5: onNext()
6: cancel()

Subscription - Cancel

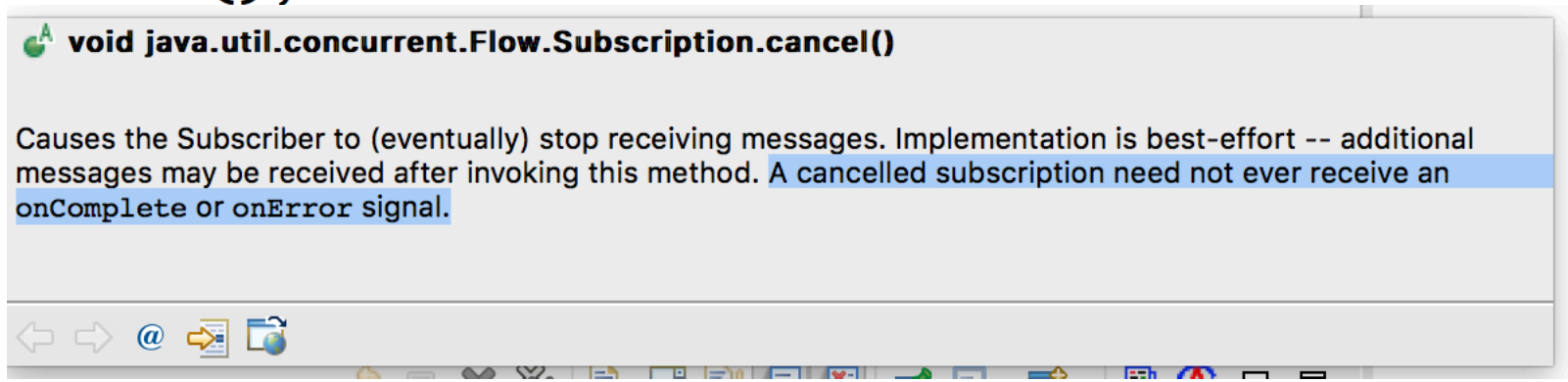
```
@Override
public void onNext(Integer value) {
    System.out.println("Received-->"+value);
    if(value<200) {
        subscription.request(1);
    }else {
        subscription.cancel();
    }
}
```



Where's the "Completed!"?

Flow-API - <Interface> Subscription

```
@Override
public void onNext(Integer value) {
    System.out.println("Received-->"+value);
    if(value<200) {
        subscription.request(1);
    }else {
        subscription.cancel();
    }
}
```



DEMO

Exercises PART 2

https://github.com/Michaeli71/JAX-London-Best-of-Java9-13.git

PART 3: Other News and Changes in APIs

- Date API
- java.utils.Arrays
- java.io.InputStream
- Misc

Date API Enhancements

The Class LocalDate

ASMIQ

- `datesUntil()` – creates a `Stream<LocalDate>` between two `LocalDate` instances and allows you to optionally specify a step size:

```
public static void main(final String[] args)
{
    final LocalDate myBirthday = LocalDate.of(1971, Month.FEBRUARY, 7);
    final LocalDate christmas = LocalDate.of(1971, Month.DECEMBER, 24);

    System.out.println("Day-Stream");
    final Stream<LocalDate> daysUntil = birthday.datesUntil(christmas);
    daysUntil.skip(150).limit(4).forEach(System.out::println);

    System.out.println("\nMonth-Stream");
    final Stream<LocalDate> monthsUntil =
                            birthday.datesUntil(christmas, Period.ofMonths(1));
    monthsUntil.limit(3).forEach(System.out::println);
}
```

Class LocalDate

ASMIQ

- Start February 7th => Skip 150 days into the future => July 7th

- **Day-Stream:** **Per day iteration limited to 4**
- **Month-Stream:** **Monthly iteration limited to 3**
  => Specification of an alternative step size, here months:

```
Day-Stream
1971-07-07
1971-07-08
1971-07-09
1971-07-10

Month-Stream
1971-02-07
1971-03-07
1971-04-07
```

java.utils.Arrays

- **Enhancements in `java.util.Arrays`:**
  - `equals()` – Compares arrays to equality (related to ranges)
  - `compare()` – Compares arrays (also related to ranges)
  - `mismatch()` – Determines the first difference in array (also related to ranges)

Misc – Class Arrays

ASMIQ

- **`Arrays.equals()` existed for a long time in JDK, but …**

- **Unfortunately, it was not possible to limit the comparison to specific ranges**

```
final byte[] string1 = "ABC-DEF-GHI".getBytes();
final byte[] string2 = "DEF".getBytes();

System.out.println(Arrays.equals(string1, string2));      => false

System.out.println(Arrays.equals(string1, 4, 7,
                                 string2, 0, 3));         => true
```

Misc – Class Arrays

ASMIQ

- **`Arrays.compare()` new in JDK**

- **Compares according to `Comparator<T>` – it is also possible to limit the comparison to specific ranges**

```
final byte[] string1 = "ABC-DEF-GHI".getBytes();
final byte[] string2 = "DEF".getBytes();

System.out.println(Arrays.compare(string1, string2));     => ABC < DEF => -3

System.out.println(Arrays.compare(string1, 4, 7,
                                  string2, 0, 3));        => DEF = DEF => 0

System.out.println(Arrays.compare(string1, 8, 11,
                                  string2, 0, 3));        => GHI > DEF => 3
```

Misc – Class Arrays

- **`Arrays.mismatch()` new in JDK**

- **Detects differences / mismatches in two arrays – it is also possible to do this just for specific ranges**

```
//                          0123401234567890123 4
final byte[] string1 = "ABC-DEF-GHIJKLMNOPQT".getBytes();
final byte[] string2 = "ABC-dEF-GHIJKLMNOPqT".getBytes();

System.out.println(Arrays.mismatch(string1, string2));     => 4

System.out.println(Arrays.mismatch(string1, 5, 20,
                                   string2, 5, 20));       => 13
```

java.io.InputStream

Misc – Class InputStream

- **There are some new helpful methods in class `InputStream` – some of them are long awaited:**
  - `public long transferTo(OutputStream out) throws IOException`
  - `public byte[] readAllBytes() throws IOException`
  - `public int readNBytes(byte[] b, int off, int len) throws IOException`

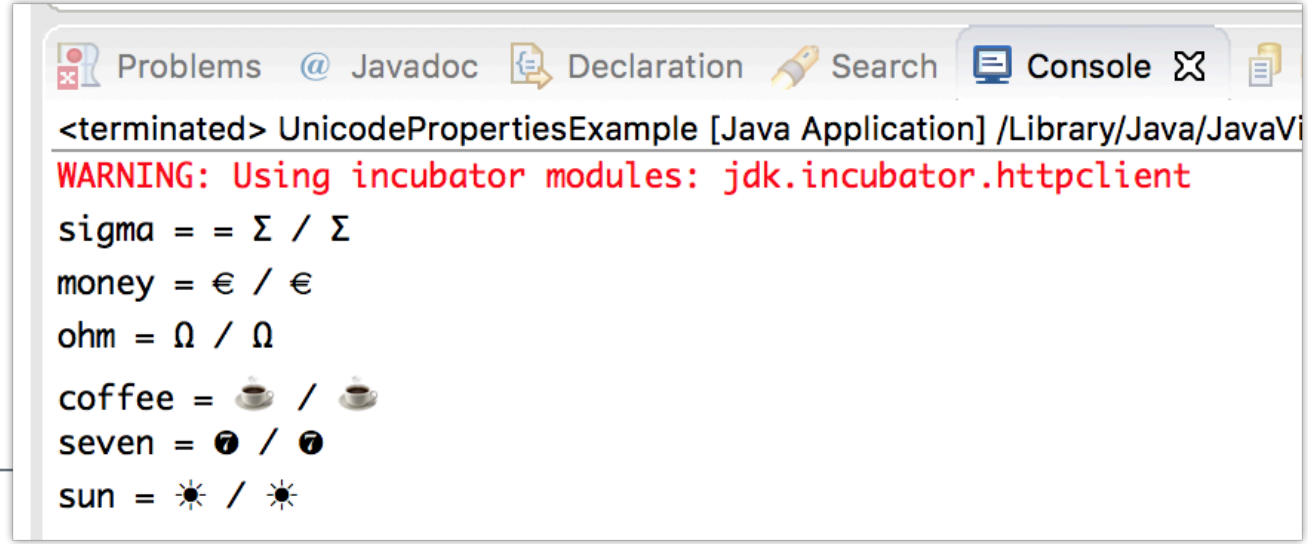
Misc

ASMIQ

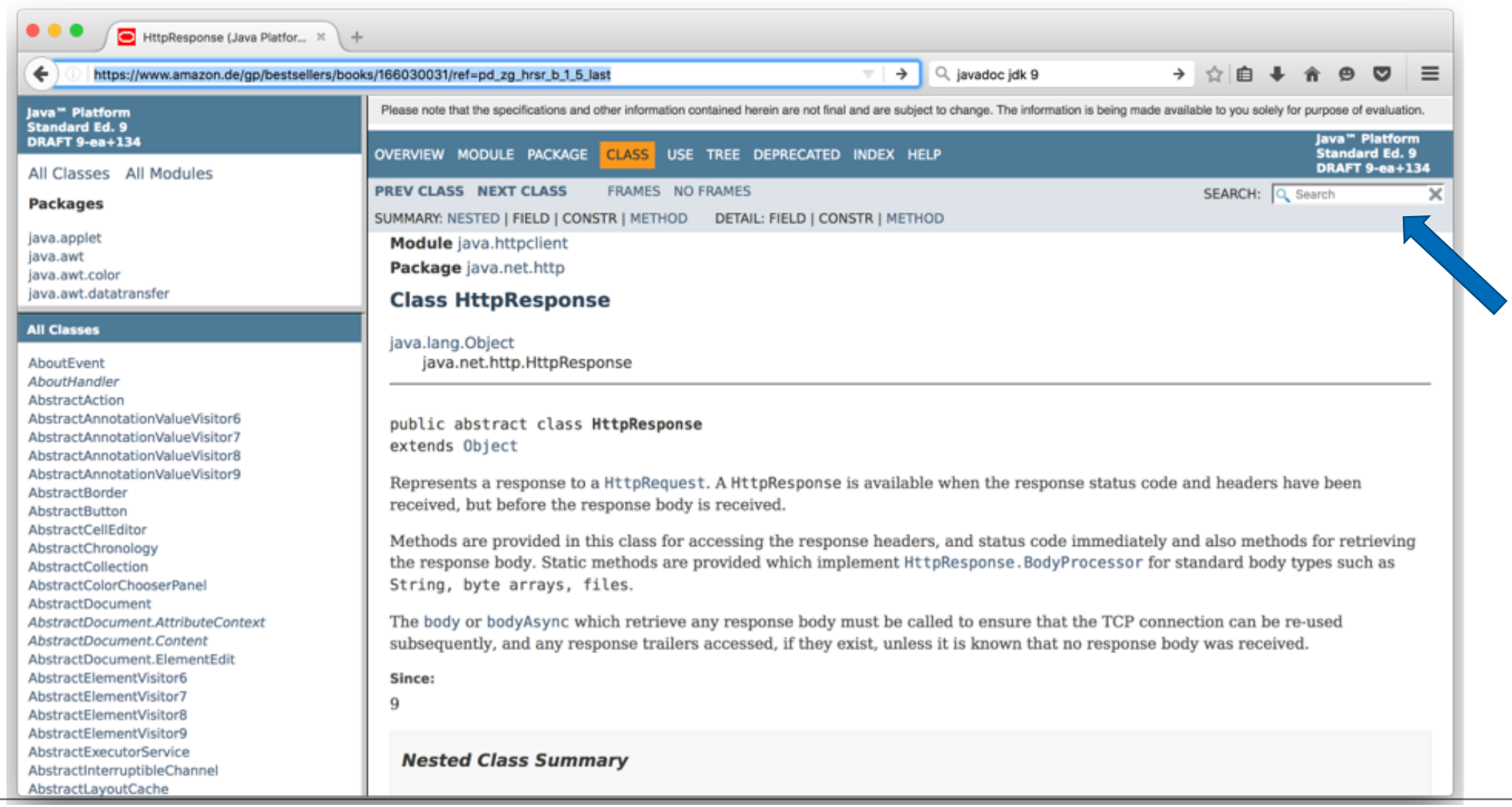
UTF-8 Resource Bundles:

```
public static void main(final String[] args) throws Exception
{
    try (final InputStream propertyFile =
          new FileInputStream("src/ch3_2_4/unicode/config.properties"))
    {
        final ResourceBundle properties = new PropertyResourceBundle(propertyFile);

        // JDK 9: Enumration => Iterator
        properties.getKeys().asIterator().forEachRemaining(key ->
        {
            System.out.println(key + " = " + properties.getString(key));
        });
    }
}
```

```
money = € / \u20AC
coffee =  / \u2615
sun =  / \u2600
seven = 7 / \u277c
ohm = Ω / \u2126
sigma = = Σ / \u03a3
```



Misc – HTML 5 Java Doc



https://docs.oracle.com/javase/9/docs/api/jdk/incubator/http/HttpResponse.html

Exercises PART 3

ASMIQ

PART 4:
What's new in Java 10

- Syntax Enhancement `var`
- Unmodifiable Collections
- Optional<T>
- Misc

Syntax Enhancements

Local Variable Type Inference => var (JDK 10)

ASMIQ

- **Local Variable Type Inference**

- **New reserved word var for defining local variables, instead of explicit type specification**

```
var name = "Peter";                    // var => String
var chars = name.toCharArray();        // var => char[]

var mike = new Person("Mike", 47);     // var => Person
var hash = mike.hashCode();            // var => int
```

- **possible if the concrete type for a local variable can be determined by the compiler using the definition on the right side of the assignment.**

Local Variable Type Inference => var

- **Especially useful in the context of generics spelling abbreviations:**

```
// var => ArrayList<String> names

ArrayList<String> namesOldStyle = new ArrayList<String>();
var names = new ArrayList<String>();
names.add("Tim"); names.add("Tom");


// var => Map<String, Long>

Map<String, Long> personAgeMappingOldStyle = Map.of("Tim", 47L, "Tom", 12L,
                                                    "Michael", 47L);
var personAgeMapping = Map.of("Tim", 47L, "Tom", 12L, "Michael", 47L);
```

Local Variable Type Inference => var

ASMIQ

- **Especially if the type specifications include several generic parameters, `var` can make the source code significantly shorter and sometimes more readable.**

```
// var => Set<Map.Entry<String, Long>>
var entries = personAgeMapping.entrySet();

// var => Map<Character, Set<Map.Entry<String, Long>>>
var filteredPersons = personAgeMapping.entrySet().stream().
                                       collect(groupingBy(firstChar,
                                               filtering(isAdult, toSet())));
```

- **But we have to use these Lambdas above:**

```
final Function<Map.Entry<String, Long>, Character> firstChar =
                            entry -> entry.getKey().charAt(0);

final Predicate<Map.Entry<String, Long>> isAdult =
                                       entry -> entry.getValue() >= 18;
```

Wouldn't it be nice to use var here too?

ASMIQ

- Yes!!!

- But the compiler cannot determine the concrete type purely based on these Lambdas

- Thus no conversion into var is possible, but leads to the error message> «Lambda expression needs an explicit target-type“.

- To avoid this mistake, the following cast could be inserted:

```
var isAdultVar = (Predicate<Map.Entry<String, Long>>)
                 entry -> entry.getValue() >= 18;
```

- **Overall, we see that var is not suitable for Lambda-expressions.**

Local Variable Type Inference Pitfall

ASMIQ

- Sometimes you may be tempted to just change the type with var:

```
final List<String> names = new ArrayList<>();
names.add("Expected");
// names.add(42); // Compile error
```

- **Let's replace the concrete type with var and see what happens if we uncomment the line**

```
final var mixedContent = new ArrayList<>();
mixedContent.add("Strange with var");
mixedContent.add(42);
```

Does this compile?
If yes, why?

Local Variable Type Inference Pitfalls

ASMIQ

- **This will compile without problems? Why?**

- We use the Diamond Operators which has no clue of the types, so the compiler uses Object as Fallback:

```
// var => ArrayList<Object>
final var mixedContent = new ArrayList<>();
mixedContent.add("Strange with var");
mixedContent.add(42);
```

Local Variable Type Inference Recap / Limitations

ASMIQ

- Short recapitulation: var is intended for **local variables** that are initialized directly.

- When using var, the **exact** type is always used and not a basic type

```
// var => ArrayList<String> und nicht List<String>
var names = new ArrayList<String>();
names = new LinkedList<String>(); // error
```

- **More things that cause compilation errrors:**

```
var justDeclaration;      // no declaration of value/definition
var numbers = {0, 1, 2}; // missing declaration of type
var appendSpace = str -> str + " "; // type not clear
```

Is it desirable or possible to use var to declare attributes, parameters or return types?

YES … but this is not possible because the type can't be determined unambiguously by the compiler.

Unmodifiable Collections Copies and Collectors

Unmodifiable Collections – Copies

ASMIQ

- Java 9 brought Collection Factory Methods to create unmodifiable collections in a readbale short manner

- **But there was no way to create unchangeable copies of any collections!**

- **Remedy 1**

```
final List<String> newCopyOfCollection = new ArrayList<>(names);
```

- **Remedy 2**

```
final Set<String> names = new HashSet<>();
names.add("Tim");
names.add("Tom");
names.add("Mike");
final Set<String> immutableNames = Collections.unmodifiableSet(names);
```

Unmodifiable Collections – Copies

ASMIQ

```
final var names = List.of("Tim", "Tom", "Mike", "Peter");
final List<String> copyOfNames = List.copyOf(names);
System.out.println("copyOfList: " +copyOfNames.getClass());

final var colors = Set.of("Red", "Green", "Blue");
final Set<String> copyOfColors = Set.copyOf(colors);
System.out.println("copyOfSet:  " + copyOfColors.getClass());

final var personAgeMapping = Map.of("Tim", 47L, "Tom", 12L, "Mike", 47L, "Max", 25L);

final Map<String, Long> copyOfMap = Map.copyOf(personAgeMapping);
System.out.println("copyOfMap:  " + copyOfMap.getClass());
```

=>

```
copyOfList:   class java.util.ImmutableCollections$ListN
copyOfSet:    class java.util.ImmutableCollections$SetN
copyOfMap:    class java.util.ImmutableCollections$MapN
```

Unmodifiable Collections – Stream API Collectors in JDK 10

ASMIQ

Stream-API offers a lot of collectors:

- **`toCollection(), toList(), toSet()` … – all of them collect the elements of a stream into the mentioned corresponding collection.**

But there was no way to create unchangeable copies of any collections!

- **The ne methode toUnmodifiableList(), toUnmodifiableSet() and toUnmodifiableMap() of class java.util.stream.Collectors provide exactly this!**

Unmodifiable Collections – Copies

ASMIQ

```
final var names = new ArrayList<>(List.of("Tim", "Tom", "Mike",
                                          "Peter", "Tom", Tim"));
final var immutableNames = names.stream().
                                 collect(Collectors.toUnmodifiableList());
System.out.println("immutableNames type: " + immutableNames.getClass());

final var uniqueImmutableNames = names.stream().
                                   collect(Collectors.toUnmodifiableSet());
System.out.println("uniqueImmutableNames content: " + uniqueImmutableNames);
System.out.println("uniqueImmutableNames type: " + uniqueImmutableNames.getClass());
```

=>

```
immutableNames type: class java.util.ImmutableCollections$ListN
uniqueImmutableNames content: [Peter, Mike, Tim, Tom]
uniqueImmutableNames type: class java.util.ImmutableCollections$SetN
```

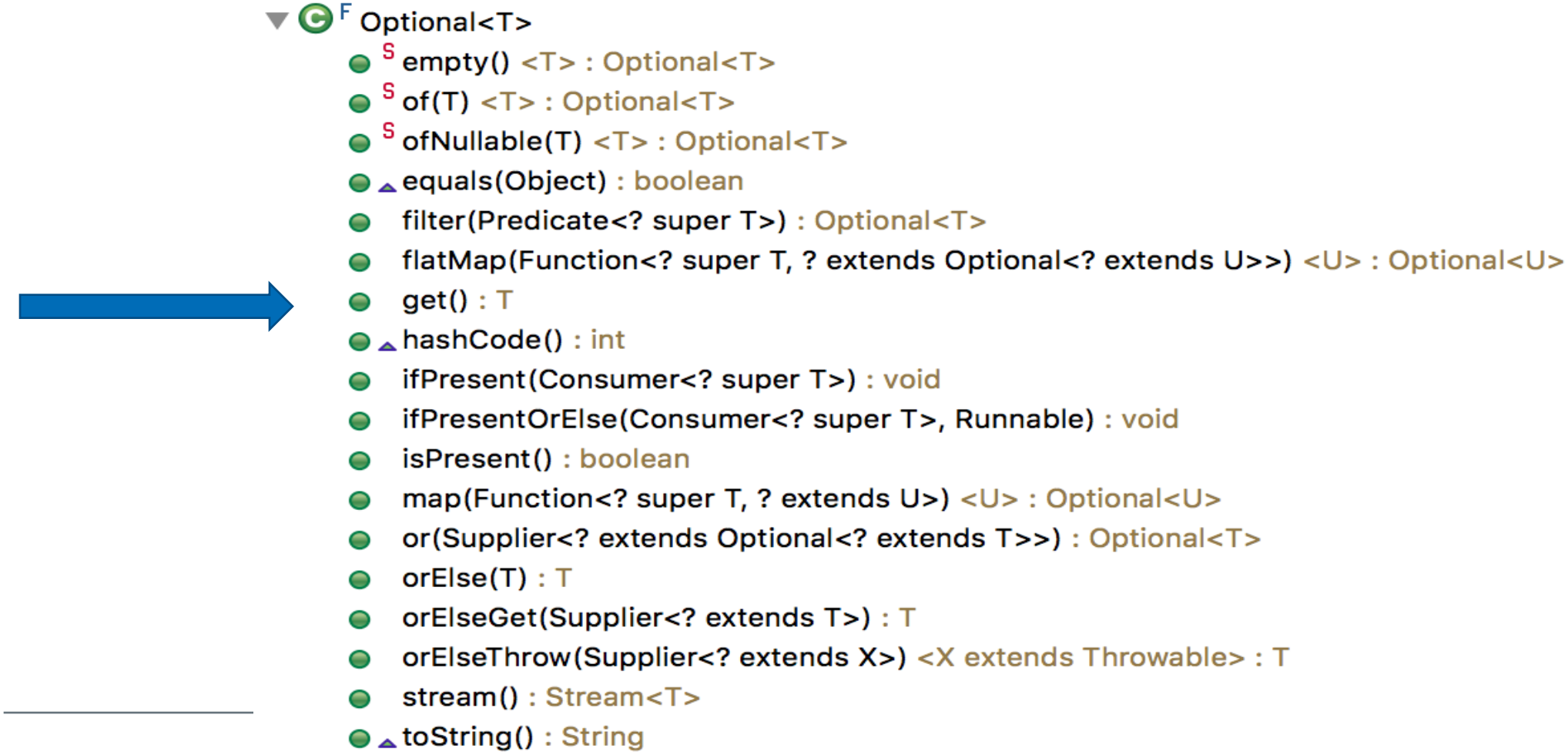
- **Interestingly the behaviour of the collector differs from the result of the of() method, which raises an exception when passing duplictes**

Optional<T>

Class Optional<T>

ASMIQ

- **Was introduced with Java 8 and facilitates the processing and modelling of optional values, Java 9 added some valuable methodes**



What may be problematic with `get()`?

Class Optional<T>

ASMIQ

- **The method `get()` looks to harmless**
- That‘s why `get()` is sometimes called without prior checking the existence with `isPresent()`
- However, this then raises a `NoSuchElementException` when there is no value.
- **Normally, a `get()` method is not expected to raise an exception.**
- **NEW IN JDK 10:**
- `orElseThrow()` as an alternative to get() to express this directly in the API.

Class Optional<T>

ASMIQ

- **Let's try it out in JShell**

```
jshell> Optional<String> optValue = Optional.of("ABC");
optValue ==> Optional[ABC]

jshell> String value = optValue.orElseThrow();
value ==> "ABC"

jshell> Optional<String> empty = Optional.empty();
empty ==> Optional.empty

jshell> empty.orElseThrow();
|  java.util.NoSuchElementException thrown: No value present
|        at Optional.orElseThrow (Optional.j
```

Misc

Class java.io.Reader

ASMIQ

- **long transferTo(Writer)**
  All characters from the reader are transferred to the specified writer - this functionality exists analogously in the InputStream class since Java 9.

```
var sr = new StringReader("Hello");
var sw = new StringWriter();

sr.transferTo(sw);

System.out.println("SW: " + sw.toString());

=>

SW: Hello
```

PART 5:
What's new in Java 11

- New license model
- Syntax enhancement `var` in Lambdas
- API-Enhancements in String
- API-Changes in Optional<T> , Files and Predicate<T>
- HTTP/2 API (already in JDK 9 as preview)

New license model

New licence model

ASMIQ

- **If you plan to distribute your software commercially, you should take the new Oracle release policy into account when downloading Java 11!**
- **Unfortunately, the Oracle JDK is now normally not free of charge -- during development, however, it can still be used free of charge.**
- **Alternatively, you can use the OpenJDK (https://openjdk.java.net/)**

Java SE Development Kit 11 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Important changes in Oracle JDK 11 License

With JDK 11 Oracle has updated the license terms on which we offer the Oracle JDK.
The new Oracle Technology Network License Agreement for Oracle Java SE is substantially different from the licenses under which previous versions of the JDK were offered. Please review the new terms carefully before downloading and using this product.

Oracle also offers this software under the GPL License on jdk.java.net/11

Syntax Enhancements

Local Variable Type Inference => var in Lambdas

ASMIQ

- **Java 10: you can't use var in Lambdas, just the following:**

```
IntFunction<Integer> doubleItTyped = (final int x) -> x * 2;
IntFunction<Integer> doubleItNoType = x -> x * 2;
```

- **Java 11: it's possible to use var in Lambdas**

```
IntFunction<Integer> doubleItTyped = (var x) -> x * 2;
```

- **What is the advantage?**

```
Function<String, String> trimmer = (@SomeAnnotation var str) -> str.trim();
```

Extensions in the Class `String`

Enhancements in `java.lang.String`

ASMIQ

- **Class String exists since JDK 1.0 with only a few changes and additions**

- **Java 11 contains the following new methods:**

  - `isBlank()`
  - `lines()`
  - `repeat(int)`
  - `strip()`
  - `stripLeading()`
  - `stripTrailing()`

Enhancement in java.lang.String: isBlank()

ASMIQ

- For strings, it was previously difficult or only possible with the help of external libraries to check whether they only contain whitespaces.
- Java 11 offers the method `isBlank()` which is based on `Character.isWhitespace(int)`

```
private static void isBlankExample()
{
	final String exampleText1 = "";
	final String exampleText2 = "    ";
	final String exampleText3 = " \n \t ";

	System.out.println(exampleText1.isBlank());
	System.out.println(exampleText2.isBlank());
	System.out.println(exampleText3.isBlank());
}
```

- All of them print out `true`.

Enhancements in java.lang.String: lines()

- When processing data from files, information often has to be broken down into individual lines. There is a method called `Files.lines(Path)` for this purpose.
- However, if the data source is a string, this functionality did not exist until now. JDK 11 provides the method `lines()`, which returns a `Stream<String>`:

```
private static void linesExample()
{
        final String exampleText = "1 This is a\n2 multi line\r“ +
                                   "3 text with\r\n4 four lines!";
        final Stream<String> lines = exampleText.lines();
        lines.forEach(System.out::println);
}

=>

1 This is a
2 multi line
3 text with
4 four lines!
```

Enhancements in java.lang.String: repeat()

- Every now and then you are faced with the task of repeating an existing string n times.
- Up to now, this has required auxiliary methods of their own or those from external libraries. With Java 11 you can use the method `repeat(int)` instead:

```
private static void repeatExample()
{
	final String star = "*";
	System.out.println(star.repeat(30));

	final String delimeter = " -*- ";
	System.out.println(delimeter.repeat(6));
}

=>

******************************
 -*-  -*-  -*-  -*-  -*-  -*-
```

Enhancement in java.lang.String: repeat() Corner Cases

ASMIQ

```
"ERROR".repeat(-1);
```

```
"ERROR".repeat(Integer.MAX_VALUE);
```

What happens?

Enhancement in java.lang.String: repeat() Corner Cases

```
"ERROR".repeat(-1);
```

```
Exception in thread "main" java.lang.IllegalArgumentException: count is negative: -1
at java.base/java.lang.String.repeat(String.java:3149)
at Java11Examples/snippet.Snippet.main(Snippet.java:16)
```

```
"ERROR".repeat(Integer.MAX_VALUE);
```

```
Exception in thread "main"
java.lang.OutOfMemoryError: Repeating 5 bytes String 2147483647 times will
produce a String exceeding maximum size.
at java.base/java.lang.String.repeat(String.java:3164)
at Java11Examples/snippet.Snippet.main(Snippet.java:14)
```

Enhancement in java.lang.String: repeat() Corner Cases

ASMIQ

```
"ERROR".repeat(Integer.MAX_VALUE);
```

```
Exception in thread "main"
java.lang.OutOfMemoryError: Repeating 5 bytes String 2147483647 times will
produce a String exceeding maximum size.
at java.base/java.lang.String.repeat(String.java:3164)
at Java11Examples/snippet.Snippet.main(Snippet.java:14)
```

```
if (Integer.MAX_VALUE / count < len)
{
    throw new OutOfMemoryError("Repeating " + len + " bytes String " + count +
                    " times will produce a String exceeding maximum size.");
}
```

Addition in java.lang.String: strip()/-Leading()/-Trailing()

ASMIQ

- **The methods `strip()`, `stripLeading()` und `stripTrailing()` remove leading, trailing whitespace or both:**

```
private static void stripExample()
{
	final String exampleText1 = "  abc ";
	final String exampleText2 = " \t XYZ \t ";

	System.out.println("'" + exampleText1.strip() + "'");
	System.out.println("'" + exampleText2.strip() + "'");
	System.out.println("'" + exampleText2.stripLeading() + "'");
	System.out.println("'" + exampleText2.stripTrailing() + "'");
}

=>

'abc'
'XYZ'
'XYZ    '
'	XYZ'
```

Addition in `Optional<T>`

Class java.util.Optional<T>

ASMIQ

- **Was updated up to Java 9 / 10 with valuable additions.**
- **Java 11 includes another new method: `isEmpty()`**
- **This makes the API analogous to that of collections and String**
- **Using this method you do not need to negate `isPresent()`**

```
final Optional<String> optEmpty = Optional.empty();

if (!optEmpty.isPresent())
        System.out.println("check for empty JDK 10 style");

if (optEmpty.isEmpty())
        System.out.println("check for empty JDK 11 style");
```

Extensions in the Class `Files`

Utility class `java.nio.file.Files`

ASMIQ

- In Java 11, the processing of strings related to files has been made easier.
- Now it is easy to write strings into and to read them from a file.
- The utility class Files provides the methods **`writeString()`** und **`readString()`** for that.

```
final Path destDath = Path.of("ExampleFile.txt");

Files.writeString(destDath, "1: This is a string to file test\n");
Files.writeString(destDath, "2: Second line");

final String line1 = Files.readString(destDath);
final String line2 = Files.readString(destDath);

System.out.println(line1);
System.out.println(line2);

=>

2: Second line
2: Second line
```

Utility class `java.nio.file.Files`

- In Java 11, the processing of strings related to files has been made easier.
- Now it is easy to write strings into and to read them from a file.
- The utility class Files provides the methods **`writeString()`** und **`readString()`** for that.

```
final Path destDath = Path.of("ExampleFile.txt");

Files.writeString(destDath, "1: This is a string to file test\n");
Files.writeString(destDath, "2: Second line");

final String line1 = Files.readString(destDath);
final String line2 = Files.readString(destDath);

System.out.println(line1);
System.out.println(line2);

=>

2: Second line
2: Second line
```

Attention for very large files (2 GB):
OutOfMemoryError

Utility class `java.nio.file.Files`

ASMIQ

- **Correction 1**: APPEND-Mode

```
Files.writeString(destDath, "2: Second line", StandardOpenOption.APPEND);

=>

1: This is a string to file test
2: Second line
1: This is a string to file test
2: Second line
```

- **Correction 2**: Read string only once

```
final String content = Files.readString(destDath);
content.lines().forEach(System.out::println);

=>

1: This is a string to file test
2: Second line
```

Addition in `Predicate<T>`

Class java.util.function.Predicate<T>

ASMIQ

- **The Predicate<T> class is very useful for expressing filter conditions for stream processing.**

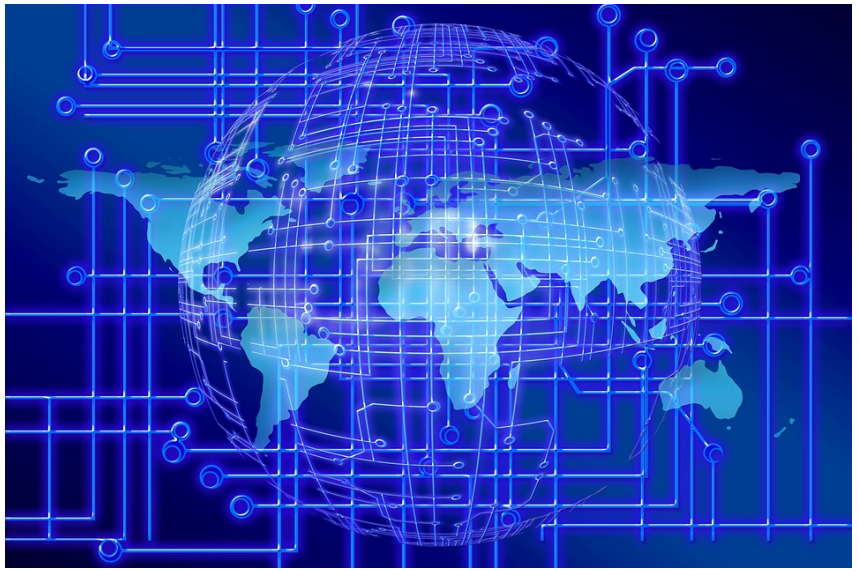
- **Besides the combination with and() and or() a negation could be expressed with negate() . This was a bit cumbersome and difficult to read:**

```
// JDK 10 style
final Predicate<String> isEmpty = String::isEmpty;
final Predicate<String> notEmptyJdk10 = isEmpty.negate();
```

- **With Java 11 this becomes more readable and no need for an additional (artificial) Explaining variable isEmpty**

```
// JDK 11 style
final Predicate<String> notEmptyJdk11 = Predicate.not(String::isEmpty);
// with static import
final Predicate<String> notEmptyJdk11 = not(String::isEmpty);
```

HTTP/2 API



HTTP-access with Java 8

ASMIQ

- **URL + openStream()**

```
public static void main(final String[] args) throws Exception
{
    final URL oracleUrl = new URL("https://www.oracle.com/index.html");

    final String contents = readContent(oracleUrl.openStream());
    System.out.println(contents);
}
```

- **URL + URLConnection + openConnection() + getInputStream()**

```
public static void main(final String[] args) throws Exception
{
    final URL oracleUrl = new URL("https://www.oracle.com/index.html");

    final URLConnection urlConnection = oracleUrl.openConnection();
    System.out.println(readContent(urlConnection.getInputStream()));
}
```

HTTP-access with Java 8

ASMIQ

- **Read content as String**

```
public static String readContent(final InputStream is) throws IOException
{
   try (final InputStreamReader isr = new InputStreamReader(is);
        final BufferedReader br = new BufferedReader(isr))
   {
      final StringBuilder content = new StringBuilder();

      String line;
      while ((line = br.readLine()) != null)
      {
         content.append(line + "\n");
      }

      return content.toString();
   }
}
```

What do you say about this code? What does it not do?

```
final URI uri = new URI("https://www.oracle.com/index.html");

final HttpRequest request = HttpRequest.newBuilder(uri).GET().build();
final BodyHandler<String> asString = HttpResponse.BodyHandlers.ofString();

final HttpClient httpClient = HttpClient.newHttpClient();
final HttpResponse<String> response = httpClient.send(request, asString);

final int responseCode = response.statusCode();
final String responseBody = response.body();

System.out.println("Status:  " + responseCode);
System.out.println("Body:    " + responseBody);
```

HTTP/2 API Async

ASMIQ

```
final URI uri = new URI("https://www.oracle.com/index.html");

final HttpRequest request = HttpRequest.newBuilder(uri).GET().build();
final BodyHandler<String> asString = HttpResponse.BodyHandlers.ofString();

final HttpClient httpClient = HttpClient.newHttpClient();
final CompletableFuture<HttpResponse<String>> asyncResponse =
                        httpClient.sendAsync(request, asString);

waitForCompletion();
if (asyncResponse.isDone())
{
    printResponseInfo(asyncResponse.get());
}
else
{
    asyncResponse.cancel(true);
    System.err.println("timeout");
}
```

HTTP/2 API other

ASMIQ

- **HTTPRequest**

```
.timeout(Duration.ofSeconds(2))
.header("Content-Type", "application/json")
```

- **HTTPClient**

```
.followRedirects(Redirect.SAME_PROTOCOL)
.proxy(ProxySelector.of(new InetSocketAddress("www-proxy.com", 8080)))
.authenticator(Authenticator.getDefault())
```

Direct Compilation Launch Single-File Source-Code Programs

Direct Compilation

- Allows you to compile and run single file Java applications directly in one go.
- saves you work and you don't have to know anything about bytecode and .class files.
- especially useful for executing smaller Java files as scripts and for getting started with Java

```java
package direct.compilation;

public class HelloWorld
{
    public static void main(String... args)
    {
        System.out.println("Hello Execute After Compile");}
}
```

```
java ./HelloWorld.java
```

```
Hello Execute After Compile
```

DEMO

Exercises PART 4 + 5

PART 6:
What's new in Java 12

- Build-Tools and IDEs
- Syntax Enhancementi `switch`
- Various APIs: Strings, `CompactNumberFormat`, `Files`
- `Teeing()-ColleCtor`
- `JMH (Microbenchmarks)`

Build-Tools and IDEs

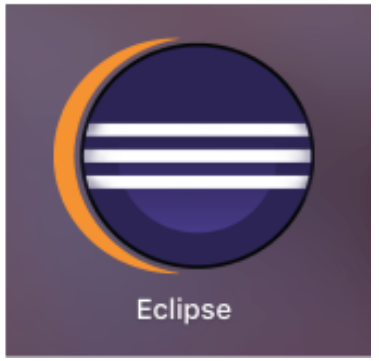
IDE & Tool Support

ASMIQ

- **Current IDEs & Tools are fundamentally good**

- **Eclipse: Version 2019-09**
- **IntelliJ: Version 2019.2.3**
- **Maven: 3.6.2, Compiler-Plugin 3.8.1**
- **Gradle: 5.6.2**

- **Activation of preview features required**
  - In dialogues
  - In the build script



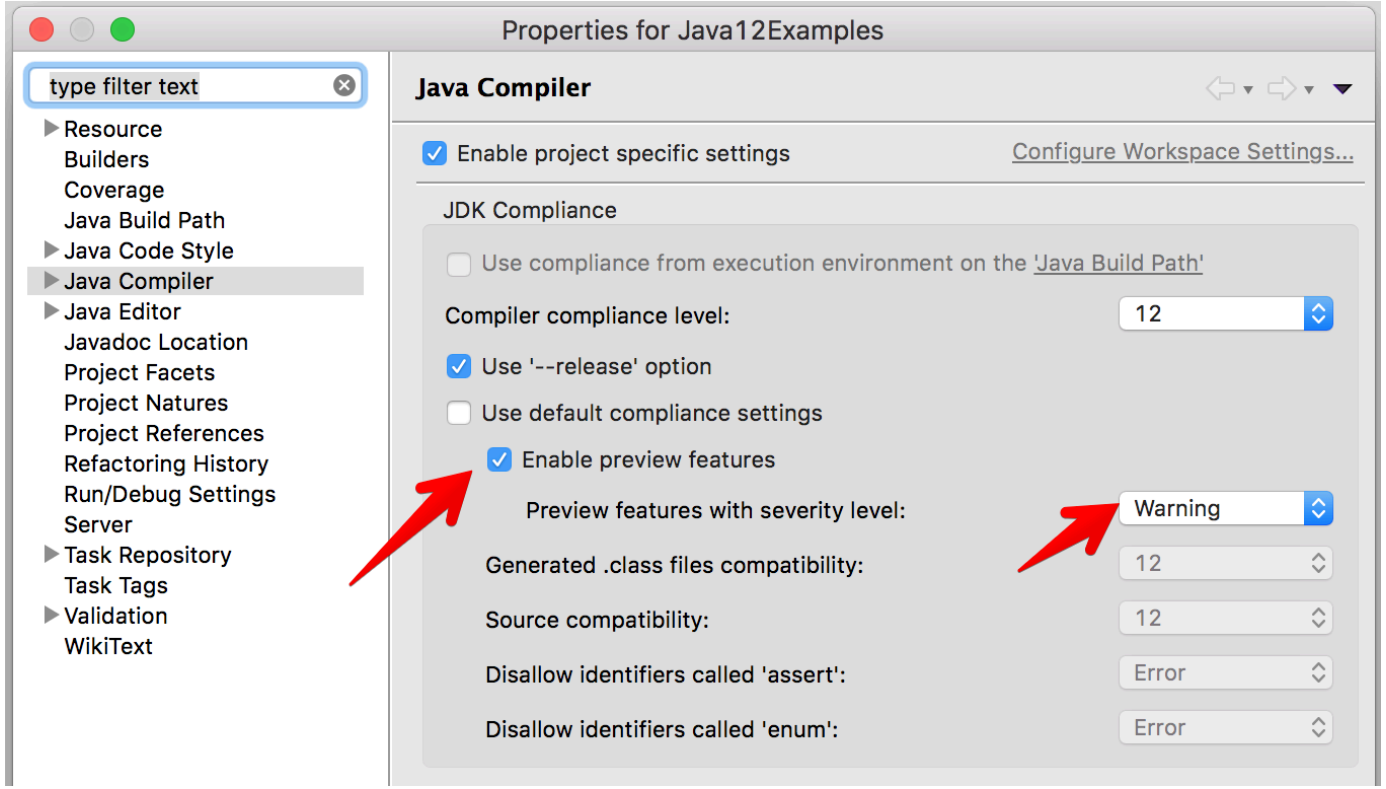
Maven™

Gradle

IDE & Tool Support

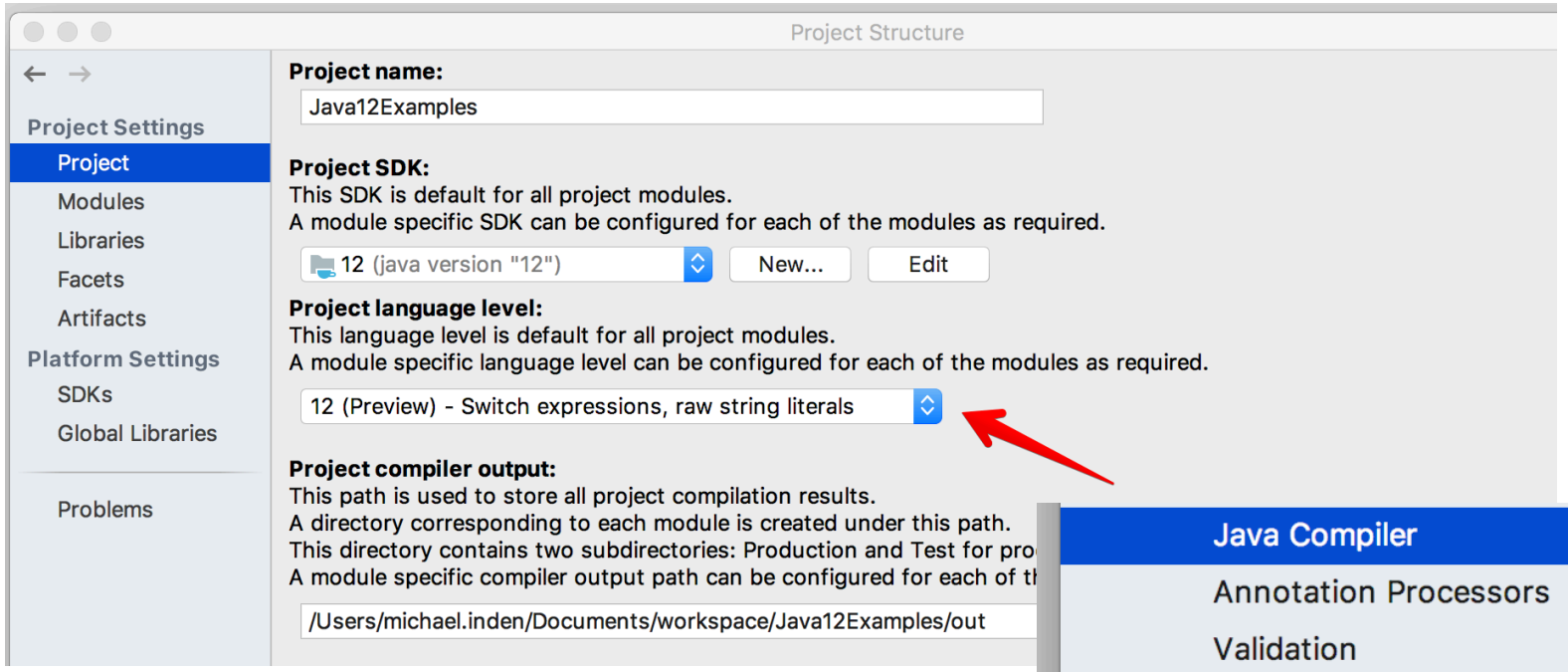
ASMIQ

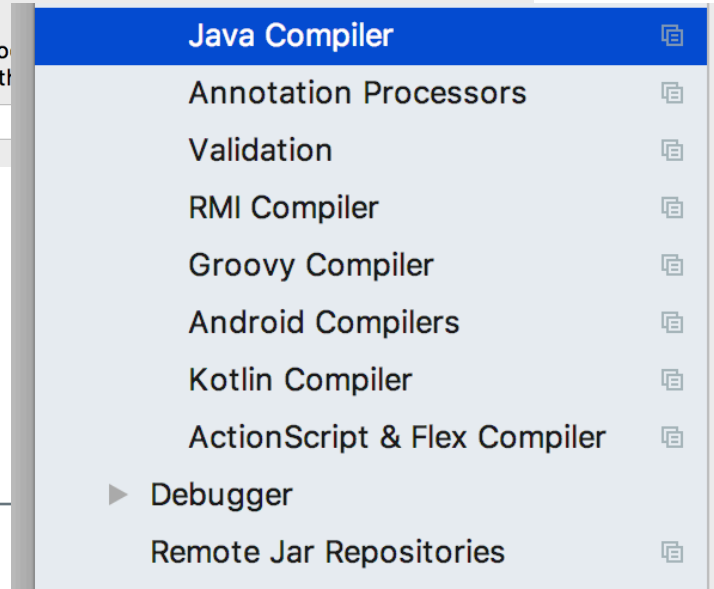
- **Activation of preview features required**

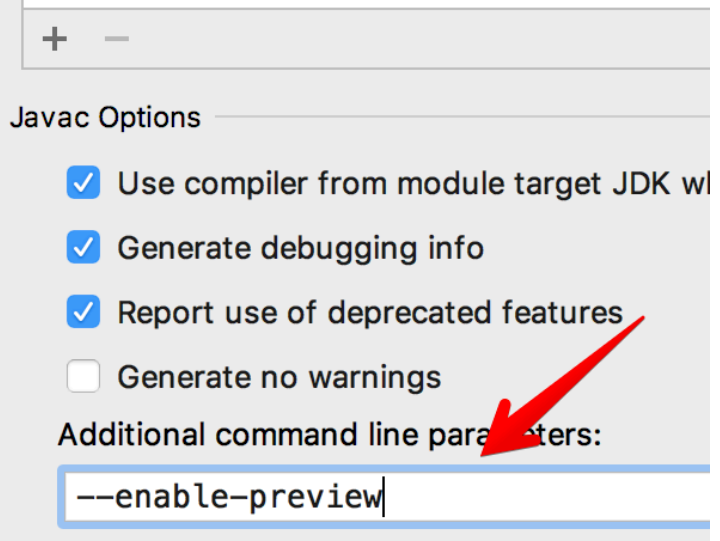


IDE & Tool Support

- **Activation of preview features required**







IDE & Tool Support

ASMIQ

- **Activation of preview features required**


```
sourceCompatibility=12
targetCompatibility=12

// Activation of Switch Expressions Preview
tasks.withType(JavaCompile) {
   options.compilerArgs += ["--enable-preview"]
}
```

IDE & Tool Support

ASMIQ

- **Activation of preview features required**

Maven™

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
            <source>12</source>
            <target>12</target>
            <!– Needed for Java 12 Syntax-Neuerungen –->
            <compilerArgs>--enable-preview</compilerArgs>
        </configuration>
    </plugin>
</plugins>
```

Syntax Enhancements

Switch Expressions (Preview)

ASMIQ

- switch-case-expression still at the ancient roots from the early days of Java
- Compromises in language design that should make the transition easier for C++ developers.
- Relicts like the break and in the absence of such the Fall-Through
- Error prone, mistakes were made again and again
- In addition, the `case` was quite limited in the specification of the values.
- This all changes fortunately with Java 12 / 13. The syntax is slightly changed and now allows the specification of one expression and several values in the case

Switch Expressions: Looking back … Retrospect

ASMIQ

- **Mapping of weekdays to their length...**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numOfLetters = -1;

switch (day)
{
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numOfLetters = 6;
        break;
    case TUESDAY:
        numOfLetters = 7;
        break;
    case THURSDAY:
    case SATURDAY:
        numOfLetters = 8;
        break;
    case WEDNESDAY:
        numOfLetters = 9;
        break;
}
```

Switch Expressions: Java 12 / 13 as a Remedy

- **Mapping of weekdays to their length… in a more elegant way with Java 12 / 13:**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = -1;

switch (day)
{
    case MONDAY, FRIDAY, SUNDAY -> numLetters = 6;
    case TUESDAY                -> numLetters = 7;
    case THURSDAY, SATURDAY     -> numLetters = 8;
    case WEDNESDAY              -> numLetters = 9;
};
```

Enhancements in class `String`

Enhancement in `java.lang.String`

ASMIQ

- **The String class exists since JDK 1.0 and has only undergone a few API changes in the meantime.**
- **With Java 11 this changed. 6 new methods were introduced.**
- **In Java 12 the following two methods are added:**
  - **indent**() - Adjusts the indentation of a string
  - **transform**() - Allows actions to transform a string

Enhancement in `java.lang.String: indent()`

- this method appends 'n' number of space characters (U+00200) in front of each line, then suffixed with a line feed "\n"

```
var test = "Test".indent(4);
System.out.println("'" + test + "'");
System.out.println(test.length());
```

- Es sind aber auch negative Werte erlaubt:

```
var removeTest = "      6789".indent(-5);
System.out.println("'" + removeTest + "'");
System.out.println(removeTest.length());
```

Enhancement in `java.lang.String`

ASMIQ

```
var test = "Test".indent(4);
System.out.println("'" + test + "'");
System.out.println(test.length());
```

```
'    Test
'
9
```

```
var removeTest = "      6789".indent(-5);
System.out.println("'" + removeTest + "'");
System.out.println(removeTest.length());
```

```
'6789
'
5
```

Enhancement in java.lang.String

```
private static void indent_Helpful_For_MultiLines()
{
    var header = "Report";
    var infoMessage = "This is a message\nThis is line 2\nLine3".indent(4);

    System.out.println(header);
    System.out.println(infoMessage);
}
```

Enhancement in java.lang.String

ASMIQ

```
private static void indent_Helpful_For_MultiLines()
{
    var header = "Report";
    var infoMessage = "This is a message\nThis is line 2\nLine3".indent(4);

    System.out.println(header);
    System.out.println(infoMessage);
}
```

```
Report
    This is a message
    This is line 2
    Line3
```

Enhancement in java.lang.String: transform()

ASMIQ

- **Example: Transform all chars of a string**
  1. convert to UPPERCASE
  2. Remove the letter ‘T’
  3. split it up

- **Until now you can do it as follows:**

```
var text = "This is a Test";

var upperCase = text.toUpperCase();
var noTs = upperCase.replaceAll("T", "");
var result =  noTs.split(" ");

System.out.println(Arrays.asList(result));
```

Enhancement in java.lang.String: transform()

ASMIQ

- **Example: Transform all chars of a string**
  1. convert to UPPERCASE
  2. Remove the letter ‘T’
  3. split it up

- **Until now you can do it as follows:**

```
var text = "This is a Test";

var upperCase = text.toUpperCase();
var noTs = upperCase.replaceAll("T", "");
var result =  noTs.split(" ");

System.out.println(Arrays.asList(result));
```

```
[HIS, IS, A, ES]
```

Enhancement in `java.lang.String: transform()`

ASMIQ

- **Usage of `transform()` to process all chars of a string**
  1. convert to UPPERCASE
  2. Remove the letter ‘T’
  3. split it up

```
var text = "This is a Test";

// chaining of operations
var result = text.transform(String::toUpperCase).
                  transform(str -> str.replaceAll("T", "")).
                  transform(str -> str.split(" "));

System.out.println(Arrays.asList(result));
```

```
[HIS, IS, A, ES]
```

- **Analogy `map()` in Streams**
- **Rather theoretically practical ☺**

```
public <R> R transform(Function<? super String,
                                 ? extends R> f)
{
   return f.apply(this);
}
```

Addition
`CompactNumberFormat`

Utility-Class CompactNumberFormat

- **CompactNumberFormat** is a subclass of **NumberFormat**
- formats a decimal number in a compact form, Locale sensitive
- NumberFormat is the abstract base class for all number formats which provides the interface for formatting and parsing numbers.
- An example of a SHORT compact form would be writing 10,000 as 10K,
- **There is a constructor, but easier to use factory method:**

```
NumberFormat compactFormat =
                NumberFormat.getCompactNumberInstance(Locale.US,
                                                      NumberFormat.Style.SHORT);
```

CompactNumberformat Style

ASMIQ

```
public static void main(final String args[])
{
    var shortFormat = getUsCompactNumberFormat(NumberFormat.Style.SHORT);
    formatNumbers("SHORT", shortFormat);

    var longFormat = getUsCompactNumberFormat(NumberFormat.Style.LONG);
    formatNumbers("LONG", longFormat);
}

private static NumberFormat getUsCompactNumberFormat(NumberFormat.Style style)
{
    return NumberFormat.getCompactNumberInstance(Locale.US, style);
}

private static void formatNumbers(final String style,
                                  final NumberFormat shortFormat)
{
    System.out.println("\nNumberFormat " + style);
    System.out.println("Result: " + shortFormat.format(10_000));
    System.out.println("Result: " + shortFormat.format(123_456));
    System.out.println("Result: " + shortFormat.format(1_234_567));
    System.out.println("Result: " + shortFormat.format(1_950_000_000));
}
```

CompactNumberformat Style

```
public static void main(final String args[])
{
    var shortFormat = getUsCompactNumberFormat(NumberFormat.Style.SHORT);
    formatNumbers("SHORT", shortFormat);

    var longFormat = getUsCompactNumberFormat(NumberFormat.Style.LONG);
    formatNumbers("LONG", longFormat);
}

private static NumberFormat getUsCompactNumberFormat(NumberFormat.Style style)
{
    return NumberFormat.getCompactNumberInstance(Locale.US, style);
}

private static void formatNumbers(final String style,
                                  final NumberFormat shortFormat)
{
    System.out.println("\nNumberFormat " + style);
    System.out.println("Result: " + shortFormat.format(10_000));
    System.out.println("Result: " + shortFormat.format(123_456));
    System.out.println("Result: " + shortFormat.format(1_234_567));
    System.out.println("Result: " + shortFormat.format(1_950_000_000));
}
```

```
NumberFormat SHORT
Result: 10K
Result: 123K
Result: 1M
Result: 2B


NumberFormat LONG
Result: 10 thousand
Result: 123 thousand
Result: 1 million
Result: 2 billion
```

Utility-Class CompactNumberformat

```
public static void main(String[] args) throws ParseException
{
    System.out.println("US/SHORT parsing:");

    var shortFormat = NumberFormat.getCompactNumberInstance(Locale.US, Style.SHORT);
    System.out.println(shortFormat.parse("1 K")); // ATTENTION
    System.out.println(shortFormat.parse("1K"));
    System.out.println(shortFormat.parse("1M"));
    System.out.println(shortFormat.parse("1B"));


    System.out.println("\nUS/LONG parsing:");

    var longFormat = NumberFormat.getCompactNumberInstance(Locale.US, Style.LONG);
    System.out.println(longFormat.parse("1 thousand"));
    System.out.println(longFormat.parse("1 million"));
    System.out.println(longFormat.parse("1 billion"));
}
```

Utility-Class CompactNumberformat

```
public static void main(String[] args) throws ParseException
{
    System.out.println("US/SHORT parsing:");

    var shortFormat = NumberFormat.getCompactNumberInstance(Locale.US, Style.SHORT);
    System.out.println(shortFormat.parse("1 K")); // ATTENTION
    System.out.println(shortFormat.parse("1K"));
    System.out.println(shortFormat.parse("1M"));
    System.out.println(shortFormat.parse("1B"));


    System.out.println("\nUS/LONG parsing:");

    var longFormat = NumberFormat.getCompactNumberInstance(Lo
    System.out.println(longFormat.parse("1 thousand"));
    System.out.println(longFormat.parse("1 million"));
    System.out.println(longFormat.parse("1 billion"));
}
```

```
US/SHORT parsing:
1
1000
1000000
1000000000


US/LONG parsing:
1000
1000000
1000000000
```

Enhancements in class `Files`

Utility-Class `java.nio.file.Files`

ASMIQ

- Methods for comparing arrays were introduced in Java 9.
- In Java 11, the processing of strings related to files has been made easier.
- In Java 12 and in the following example, these are combined in methode **mismatch()**

```
Path filePath1 = Files.createTempFile("test1", ".txt");
Path filePath2 = Files.createTempFile("test2", ".txt");
Path filePath3 = Files.createTempFile("test3", ".txt");

Files.writeString(filePath1, "Same Content");
Files.writeString(filePath2, "Same Content");
Files.writeString(filePath3, "Same Start / Different Content");

long mismatchPos1 = Files.mismatch(filePath1, filePath2);
System.out.println("File1 mismatch File2 = " + mismatchPos1);

long mismatchPos2 = Files.mismatch(filePath1, filePath3);
System.out.println("File1 mismatch File3 = " + mismatchPos2);
```

Utility-Class java.nio.file.Files

ASMIQ

```
Path filePath1 = Files.createTempFile("test1", ".txt");
Path filePath2 = Files.createTempFile("test2", ".txt");
Path filePath3 = Files.createTempFile("test3", ".txt");

Files.writeString(filePath1, "Same Content");
Files.writeString(filePath2, "Same Content");
Files.writeString(filePath3, "Same Start / Different Content");

long mismatchPos1 = Files.mismatch(filePath1, filePath2);
System.out.println("File1 mismatch File2 = " + mismatchPos1);

long mismatchPos2 = Files.mismatch(filePath1, filePath3);
System.out.println("File1 mismatch File3 = " + mismatchPos2);
```

```
File1 mismatch File2 = -1
File1 mismatch File3 = 5
```

Utility-Class java.nio.file.Files

ASMIQ

```
Path fileEnc1 = Files.createTempFile("enc1", ".latin1");
Path fileEnc2 = Files.createTempFile("enc2", ".utf8");

Files.writeString(fileEnc1, "Zürich is beautiful. Mainz too",
                  StandardCharsets.ISO_8859_1);
Files.writeString(fileEnc2, "Zürich is beautiful. Mainz too",
                  StandardCharsets.UTF_8);

var mismatchPos = Files.mismatch(fileEnc1, fileEnc2);
System.out.println("enc1 mismatch enc2 = " + mismatchPos);


var oneFile = Files.createTempFile("oneFile", ".txt");

var mismatchPosSameFile = Files.mismatch(oneFile, oneFile);
System.out.println("oneFile mismatch oneFile = " + mismatchPosSameFile);
```

Utility-Class java.nio.file.Files

ASMIQ

```
Path fileEnc1 = Files.createTempFile("enc1", ".latin1");
Path fileEnc2 = Files.createTempFile("enc2", ".utf8");

Files.writeString(fileEnc1, "Zürich is beautiful. Mainz too",
                  StandardCharsets.ISO_8859_1);
Files.writeString(fileEnc2, "Zürich is beautiful. Mainz too",
                  StandardCharsets.UTF_8);

var mismatchPos = Files.mismatch(fileEnc1, fileEnc2);
System.out.println("enc1 mismatch enc2 = " + mismatchPos);


var oneFile = Files.createTempFile("oneFile", ".txt");

var mismatchPosSameFile = Files.mismatch(oneFile, oneFile);
System.out.println("oneFile mismatch oneFile = " + mismatchPosSameFile);
```

```
enc1 mismatch enc2 = 1
oneFile mismatch oneFile = -1
```

Utility-Class `java.nio.file.Files` – Pitfalls

- Return values consistent to expectation with same encoding
- Therefore …

File 1	File 2	Encoding		Result
ABCD	ABCD	same		-1
ABCDEF	ABCDXY	same		4
Zürich	Zürich	different		Positive value, first character with umlaut or encoding deviation

- If the paths point to the same file, it is of course also the same

Teeing() Collector

ASMIQ

The powerful Stream API provides a set of predefined collectors:

- **`toCollection(), toList(), toSet() ...` – Group the items of the stream into the appropriate collection.**
- **Java 10 also added `toUnmodifableXyz()`!**

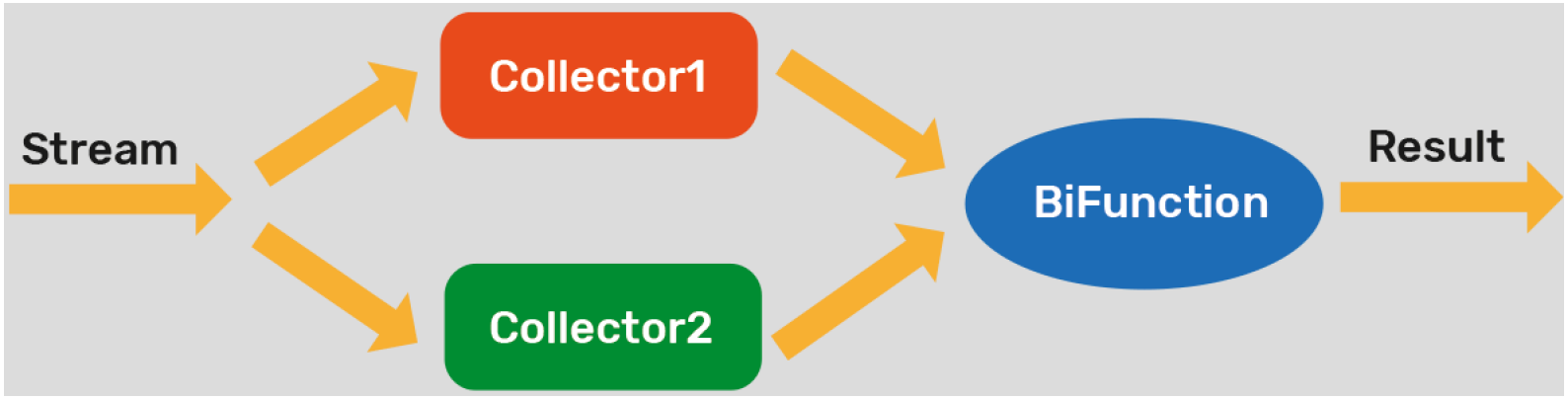
What is missing?

ASMIQ

What about combining streams?

"...returns a Collector that is a composite of two downstream collectors. Every element passed to the resulting collector is processed by both downstream collectors, then their results are merged using the specified merge function into the final result."

```
static <T, R1, R2, R> Collector<T, ?, R> teeing(Collector<? super T, ?, R1> downstream1,
                                                Collector<? super T, ?, R2> downstream2,
                                                BiFunction<? super R1, ? super R2, R> merger)
```



Collectors

ASMIQ

```
public static void main(String[] args)
{
	var firstSixNumbers = Stream.of(1, 2, 3, 4, 5, 6);
	System.out.println(calcCountAndSum(firstSixNumbers));

	var primeNumbers = Stream.of(1, 3, 5, 7, 11, 13, 17);
	System.out.println(calcCountAndSum(primeNumbers));
}

private static Pair<Long> calcCountAndSum(Stream<Integer> numbers)
{
	return numbers.collect(Collectors.teeing(
				Collectors.counting(),
				Collectors.summingLong(n -> n),
				// (count, sum) -> new Pair<Long>(count, sum))
				Pair<Long>::new));
}
```

Collectors

```
public static void main(String[] args)
{
        var firstSixNumbers = Stream.of(1, 2, 3, 4, 5, 6);
        System.out.println(calcCountAndSum(firstSixNumbers));

        var primeNumbers = Stream.of(1, 3, 5, 7, 11, 13, 17);
        System.out.println(calcCountAndSum(primeNumbers));
}

private static Pair<Long> calcCountAndSum(Stream<Integer> numbers)
{
        return numbers.collect(Collectors.teeing(
                                        Collectors.counting(),
                                        Collectors.summingLong(n -> n),
                                        Pair<Long>::new));
}
```

```
Pair<T> [first=6, second=21]
Pair<T> [first=7, second=57]
```

Excursion Pair<T>

```
static class Pair<T>
{
        public T first;
        public T second;

        public Pair(T first, T second)
        {
                this.first = first;
                this.second = second;
        }

        @Override
        public String toString()
        {
            return "Pair [first=" + first + ", second=" + second + "]";
        }
}
```

How about Map.Entry as a replacement for your own pair?

Not so good!!

Why? Map.Entry is intended for maps and should only be used in their context.

ASMIQ

Task: From a stream of strings, different elements should be filtered out and then the results should be combined.

- **This can be achieved combining collectors `filtering()` and `teeing()`:**

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");
final BiFunction<List<String>, List<String>, List<List<String>>> combineLists =
                                        (list1, list2) -> List.of(list1, list2);

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  // (list1, list2) -> List.of(list1, list2)
                                  combineLists));
System.out.println(result);
```

ASMIQ

Task: From a stream of strings, different elements should be filtered out and then the results should be combined.

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  (list1, list2) -> List.of(list1, list2));
System.out.println(result);
```

[Michael, Mike]

ASMIQ

Task: From a stream of strings, different elements should be filtered out and then the results should be combined.

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  (list1, list2) -> List.of(list1, list2));
System.out.println(result);
```

[Michael, Mike]

[Tim, Tom]

ASMIQ

Task: From a stream of strings, different elements should be filtered out and then the results should be combined.

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  (list1, list2) -> List.of(list1, list2));

System.out.println(result);
```

[Michael, Mike]

[Tim, Tom]

[[Michael, Mike], [Tim, Tom]]

JMH

Benchmarking Intro

ASMIQ

- **Sometimes some parts of the software are not as performant as needed.**
- **There are various tools and levels for optimizing performance**
- **In general, one should first measure thoroughly and optimize only with caution.**

- **Why?**
  - There are already various optimizations built into the JVM
  - Optimization is not trivial, since the measurements should be performed under the same conditions (CPU load, memory consumption, etc.), for comparable results
  - purely on the basis of assumptions one is often wrong

- **by no means optimize only based on assumptions, make sure it is based on measurements:**
  - simple start/stop measurements
  - More sophisticated processes with several runs are more recommendable.

JEP 2330: Micobenchmark Suite / JMH

ASMIQ

- JEP 230 add a basic suite of microbenchmarks to the JDK source code, and make it easy for developers to run existing microbenchmarks and create new ones.
- Based on Java Microbenchmark Harness (JMH)
- Framework for creating microbenchmark tests
- Microbenchmarking = optimization on the level of single or fewer statements
- Considers various external interferences and fluctuations
- Comprehensive, but mostly easy to configure
- Makes writing benchmarks almost as easy as unit testing with JUnit

Einfache Start-/Stop-Messungen

- Simple start/stop measurements with System.currentTimeMillis()

```
// ACHTUNG: keine gute Variante
final long startTimeMs = System.currentTimeMillis();

someCodeToMeasure();

final long stopTimeMs = System.currentTimeMillis();
final long duration = stopTimeMs - startTimeMs;
```

- surround the program part to be measured with System.currentTimeMillis()
- Use as a kind of stopwatch by determining and the difference between the values

Repeated Start-/Stop Measurements

ASMIQ

- Repeated Start-/Stop measurements

```
// bessere Variante
final long startTimeNs = java.lang.System.nanoTime();

for (int i = 0; i < MAX_ITERATIONS; i++)
{
   someCodeToMeasure();
}

final long stopTimeNs = java.lang.System.nanoTime();

final long avgDurationNs = (stopTimeNs - startTimeNs) / MAX_ITERATIONS;
```

- Less susceptible to system load fluctuations or other interference due to multiple runs and averaging.
- It is also quite easy to determine minimum and maximum duration or standard deviation.

Repeated Start-/Stop Measurements with Warm-up

ASMIQ

- **settling effects**: Only after a certain number of runs does functionality show its optimum runtime:

```
// noch bessere Variante der Messung durch Warm-up
for (int i = 0; i < MAX_WARMUP_ITERATIONS; i++)
{
   someCodeToMeasure();
}

// eigentliche Messung nach Warm-up
final long startTimeNs = java.lang.System.nanoTime();

for (int i = 0; i < MAX_ITERATIONS; i++)
{
   someCodeToMeasure();
}
final long stopTimeNs = java.lang.System.nanoTime();

final long avgDurationNs = (stopTimeNs - startTimeNs) / MAX_ITERATIONS;
```

Microbenchmarks with JMH

ASMIQ

- JMH can create a performance test environment with the following Maven command:

```
mvn archetype:generate \
> -DinteractiveMode=false \
> -DarchetypeGroupId=org.openjdk.jmh \
> -DarchetypeArtifactId=jmh-java-benchmark-archetype \
> -DgroupId=org.sample \
> -DartifactId=jmh-test \
> -Dversion=1.0-SNAPSHOT
```

Microbenchmarks with JMH

ASMIQ

- A MyBenchmark class is generated as the skeleton:

```
public class MyBenchmark
{
    @Benchmark
    public void testMethod()
    {
        // This is a demo/sample template for building your JMH benchmarks. Edit
            as needed.
        // Put your benchmark code here.
    }
}
```

- JMH works with annotations and integrates different measurements based on them.

ASMIQ

- **You can create your own benchmark classes based on the skeleton:**

```
├── pom.xml
├── src
│   └── main
│       └── java
│           └── org
│               └── sample
│                   ├── FirstIntegerBenchmark.java
│                   └── SimpleMathBenchmark.java
└── target
    ├── benchmarks.jar
```

- **In 2 steps to a benchmark**

```
1) mvn clean package
2) java -jar target/benchmarks.jar
```

What do we want to measure?

- `indexOf(String), indexOf(char), contains(String)`
- `for, forEach, while, Iterator`
- `String +=, Sting.concat(), StringBuilder.append()`

Example Results

Benchmark	Mode	Cnt	Score	Error	Units
LoopBenchmark.loopFor	avgt	10	5.039	± 0.134	ms/op
LoopBenchmark.loopForEach	avgt	10	5.308	± 0.322	ms/op
LoopBenchmark.loopIterator	avgt	10	5.466	± 0.528	ms/op
LoopBenchmark.loopWhile	avgt	10	5.026	± 0.218	ms/op

Benchmark	Mode	Cnt	Score	Error	Units
SearchBenchmark.testContains	avgt	15	7.712	± 0.241	ns/op
SearchBenchmark.testIndexOf	avgt	15	7.797	± 0.475	ns/op
SearchBenchmark.testIndexOfChar	avgt	15	7.046	± 0.070	ns/op

Benchmark	Mode	Cnt	Score	Error	Units
StringAddBenchmark.stringBuilderAppend_Dump	avgt	10	22.349	± 0.091	ms/op
StringAddBenchmark.stringBuilderAppend_Multiple	avgt	10	15.872	± 0.055	ms/op
StringAddBenchmark.stringConcat	avgt	10	636655.091	± 80873.906	ms/op
StringAddBenchmark.stringPlus	avgt	10	673963.206	± 64323.287	ms/op

ASMIQ

PART 5:
What's new in Java 13

- Build-Tools and IDEs
- Syntax Enhancements

Build-Tools and IDEs

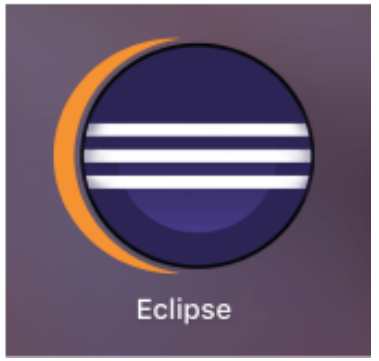
IDE & Tool Support

ASMIQ

- **Current IDEs & Tools are fundamentally good**

- **Eclipse: Version 2019-09**
- **IntelliJ: Version 2019.2.3**
- **Maven: 3.6.2, Compiler-Plugin 3.8.1**
- **Gradle: 5.6.2**

- **Activation of preview features required**
  - In dialogues
  - In the build script



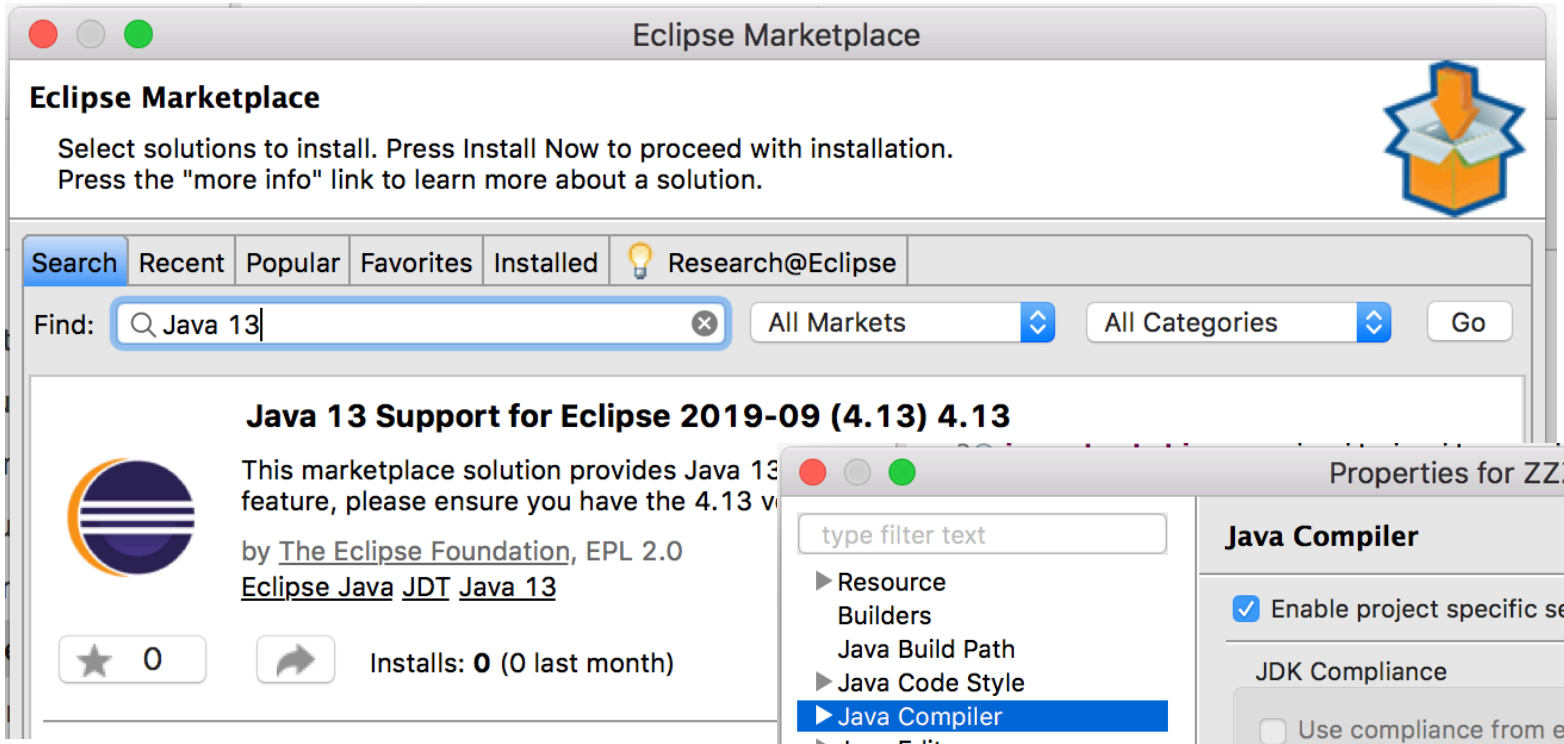
Maven™

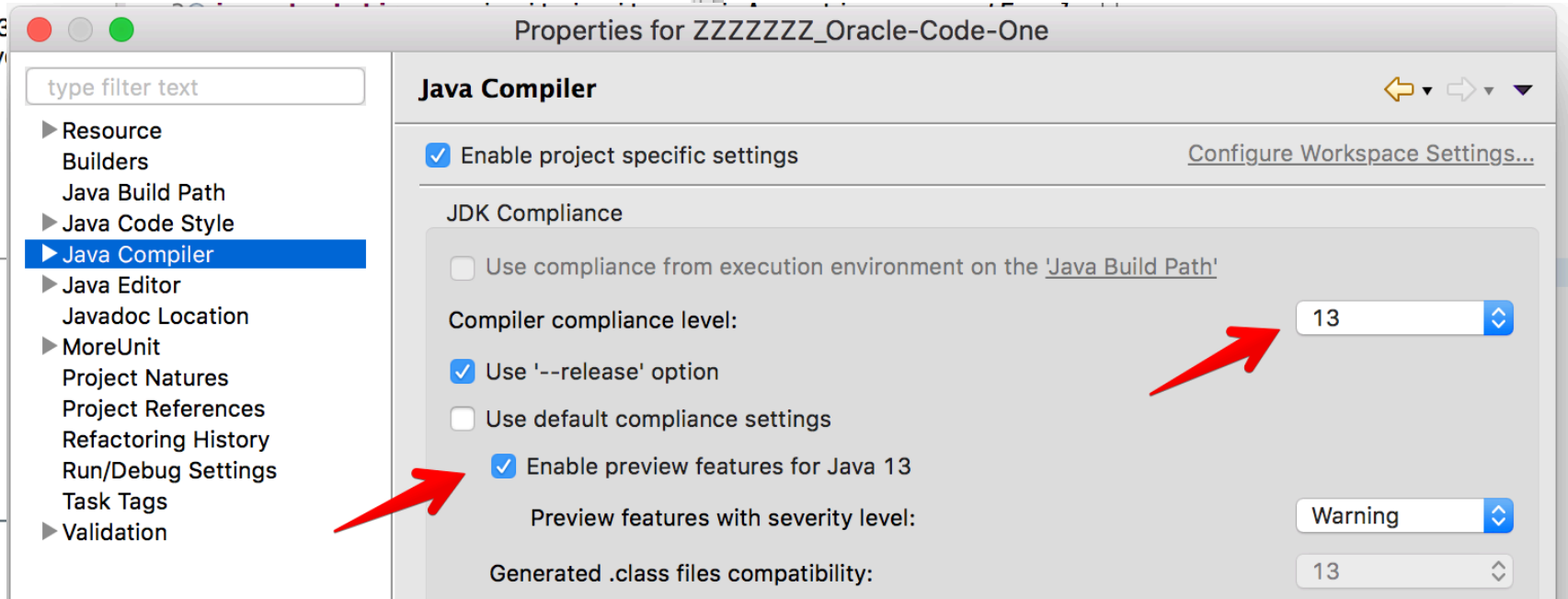
Gradle

IDE & Tool Support

ASMIQ

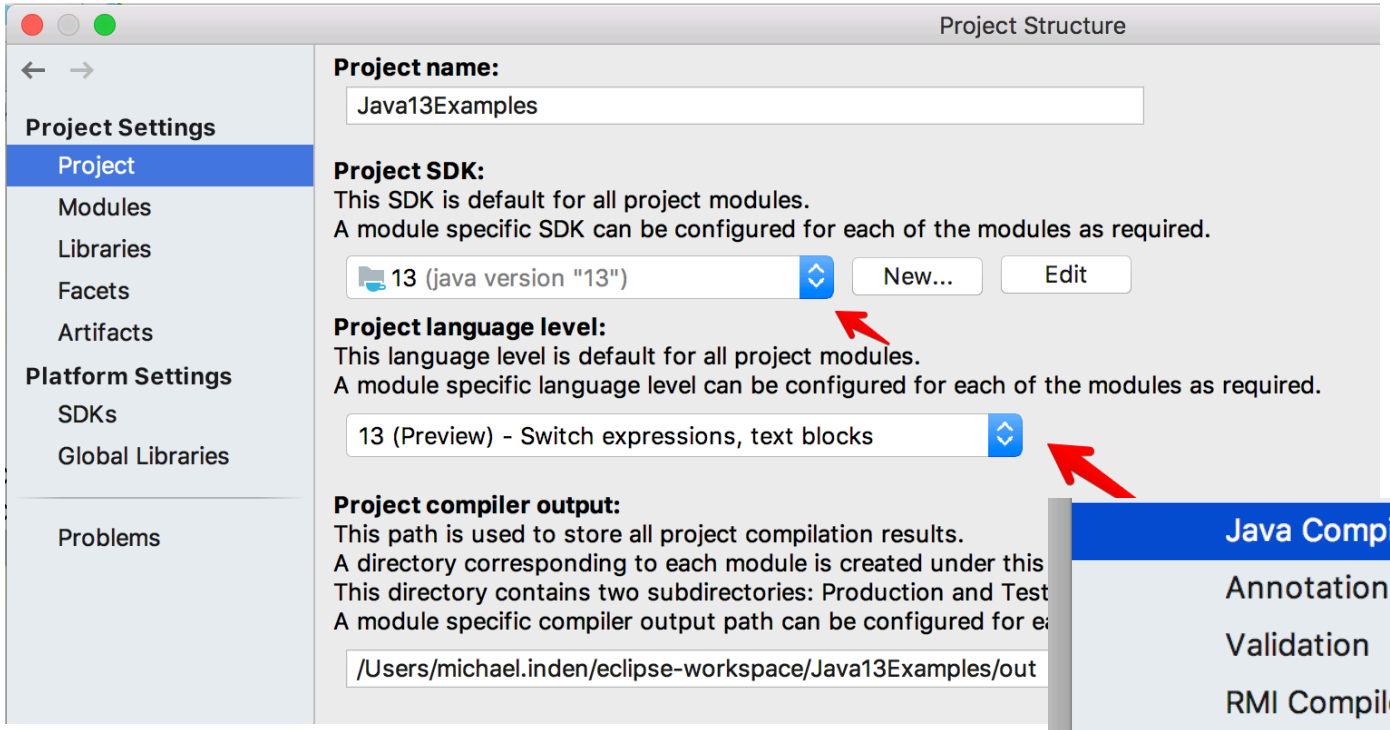
- **Installation of Plugin & Activation of Preview-Features needed**

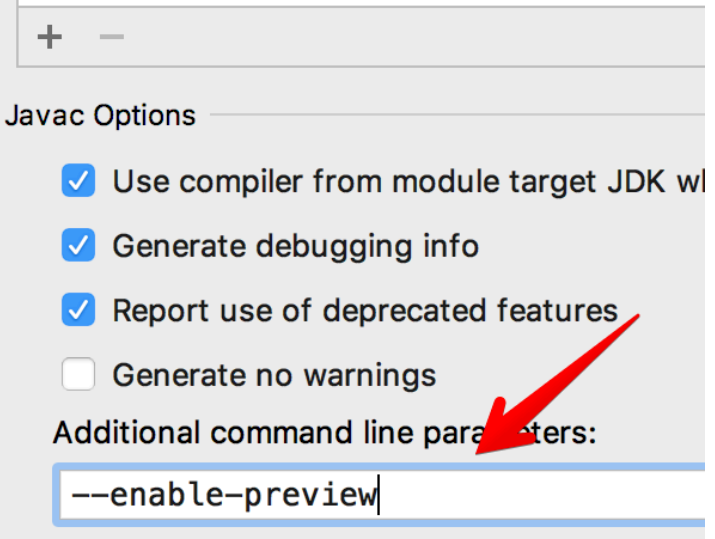




IDE & Tool Support

- **Activation of Preview-Features is needed**





IDE & Tool Support

ASMIQ

- **Activation of Preview-Features is needed**


```
sourceCompatibility=13
targetCompatibility=13

// Activation of Switch Expressions Preview
tasks.withType(JavaCompile) {
   options.compilerArgs += ["--enable-preview"]
}
```

```
* What went wrong:
Could not compile build file '/Users/michael.inden/Desktop/Vorträge/ZZZZZZ_JAX-
London Java 9 bis 13 WORKSHOP 2019/Übungen-Teil1-
APIs/Best_of_Java_9_bis_13/build.gradle'.
> startup failed:
  General error during semantic analysis: Unsupported class file major version 57

  java.lang.IllegalArgumentException: Unsupported class file major version 57
        at groovyjarjarasm.asm.ClassReader.<init>(ClassReader.java:184)
```

IDE & Tool Support – Work around not working anymore!

- **Additionally Gradle has to be started with Java 12 or lower**

```
/usr/libexec/java_home -V

=>

export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home
```

- **Additionally Gradle has to be configures with gradle.properties:**

```
// JAVA_HOME points to Java 12 or below in the console
// For compilation use Java 13
org.gradle.java.home=/Library/Java/JavaVirtualMachines/jdk-13.jdk/Contents/Home
```

IDE & Tool Support

ASMIQ

- **Activation of Preview-Features is needed**

Maven™

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
            <source>13</source>
            <target>13</target>
            <!-- Important for Java 12/13 Syntax -->
            <compilerArgs>--enable-preview</compilerArgs>
        </configuration>
    </plugin>
</plugins>
```

Syntax Enhancements

Switch Expressions

ASMIQ

- switch-case-expression still at the ancient roots from the early days of Java
- Compromises in language design that should make the transition easier for C++ developers.
- Relicts like the break and in the absence of such the Fall-Through
- Error prone, mistakes were made again and again
- In addition, the `case` was quite limited in the specification of the values.
- This all changes fortunately with Java 12 / 13. The syntax is slightly changed and now allows the specification of one expression and several values in the case

Switch Expressions: Retrospect

- **Mapping of weekdays to their length...**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numOfLetters = -1;

switch (day)
{
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numOfLetters = 6;
        break;
    case TUESDAY:
        numOfLetters = 7;
        break;
    case THURSDAY:
    case SATURDAY:
        numOfLetters = 8;
        break;
    case WEDNESDAY:
        numOfLetters = 9;
        break;
}
```

Switch Expressions: Java 12 / 13 as a Remedy

ASMIQ

- **Mapping of weekdays to their length… elegantly with Java 12 / 13:**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = -1;

switch (day)
{
    case MONDAY, FRIDAY, SUNDAY -> numLetters = 6;
    case TUESDAY                -> numLetters = 7;
    case THURSDAY, SATURDAY     -> numLetters = 8;
    case WEDNESDAY              -> numLetters = 9;
};
```

Switch Expressions: Java 12 / 13 as a Remedy

ASMIQ

- **Mapping of weekdays to their length… elegantly with Java 12 / 13:**

Return-Value

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = switch (day)
{
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```

Switch Expressions: Java 13 als Abhilfe

ASMIQ

- **Mapping of weekdays to their length… elegantly with Java 12 / 13:**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = switch (day)
{
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```

- **More elegance using case:**
  - Besides the obvious arrow instead of the colon also several values allowed
  - No break necessary, no case-through either
  - switch can now return a value, avoids artificial auxiliary variables

Switch Expressions: Retrospect… Pitfalls

ASMIQ

- **Mapping of month to their names...**

```
// ATTENTION: Sometimes very bad error: default in the middle of cases
String monthString = "";
switch (month)
{
    case JANUARY:
        monthString = "January";
        break;
    default:
        monthString = "N/A";  // here HIDDEN Fall Through
    case FEBRUARY:
        monthString = "February";
        break;
    case MARCH:
        monthString = "March";
        break;
    case JULY:
        monthString = "July";
        break;
}

System.out.println("OLD: " + month + " = " + monthString); // February
```

Switch Expressions: Java 12 / 13 as a Remedy

ASMIQ

- **Mapping months to their names... elegantly with Java 12 / 13:**

```
static String monthToName(final Month month)
{
    return switch (month)
    {
        case JANUARY -> "January";
        default -> "N/A";  // here is NO Fall Through
        case FEBRUARY -> "February";
        case MARCH -> "March";
        case JULY -> "JULY";
    };
}
```

Switch Expressions: Pitfalls with break in older Java versions

- **here is an enumeration of colors:**

```
enum Color { RED, GREEN, BLUE, YELLOW, ORANGE };
```

- **Let's map them to the number of letters:**

```
Color color = Color.GREEN;
int numOfChars;

switch (color)
{
    case RED: numOfChars = 3; break;
    case GREEN: numOfChars = 5; /* break; UPS: FALL-THROUGH */;
    case YELLOW: numOfChars = 6; break;
    case ORANGE: numOfChars = 6; break;
    default: numOfChars = -1;
}
```

Switch Expressions: `yield` with return value

ASMIQ

With Java 12 / 13 will again all be clear and easy:

```
public static void switchYieldReturnsValue(Color color)
{
    int numOfChars = switch (color)
    {
        case RED: yield 3;
        case GREEN:  yield 5;
        case YELLOW, ORANGE: yield 6;
        default: yield -1;
    };

    System.out.println("color: " + color + " ==> " + numOfChars);
}
```

Text Blocks

Text Blocks

ASMIQ

- **Long-awaited extension, namely to be able to define multiline strings without laborious links and to dispense with error-prone escaping.**

- **Facilitates, among other things, the handling of SQL commands, or the definition of JavaScript in Java source code.**

- **OLD**

```
String javaScriptCodeOld = "function hello() {\n" +
        "   print(\"Hello World\");\n" +
        "}\n" +
        "\n" +
        "hello();\n";
```

Text Blocks

- **NEW**

```
String javascriptCode = """
                        void print(Object o)
                        {
                            System.out.println(Objects.toString(o));
                        }
                        """;

String multiLineString = """
                         THIS IS
                         A MULTI
                         LINE STRING
                         WITH A BACKSLASH \\
                         """;
```

Text Blocks

- https://openjdk.java.net/jeps/326

Traditional String Literals

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

```
String multiLineHtml = """
                       <html>
                           <body>
                               <p>Hello, world</p>
                           </body>
                       </html>
                       """;
```

Text Blocks

- **NEW**

```
String multiLineSQL = """
                      SELECT `ID`, `LAST_NAME` FROM `CUSTOMER`
                      WHERE `CITY` = 'ZÜRICH'
                      ORDER BY `LAST_NAME;
                      """;


String multiLineStringWithPlaceHolders = """
                        SELECT %s
                          FROM %s
                          WHERE %s
                       """.formatted(new Object[]{"A", "B", "C"});
```

Text Blocks

- **NEW**

```
String jsonObj = """
                 {
                     name: "Mike",
                     birthday: "1971-02-07",
                     comment: "Text blocks are nice!"
                 }
                 """;
```

Exercises PART 6 & 7

Questions?


HELP
IS ON THE
WAY


Michael Inden
Java
Die Neuerungen in Version 9 bis 12
Modularisierung, Syntax- und API-Erweiterungen
dpunkt.verlag


4.
Auflage
Michael Inden
Der Weg zum Java-Profi
Konzepte und Techniken für die professionelle Java-Entwicklung
Aktuell zu Java 9
dpunkt.verlag

Thank You