



Best of Java 9 – 17

<https://github.com/Michaeli71/JAX-London-Best-of-Java9-17>

Michael Inden

Independent SW consultant and author

Speaker Intro



- Michael Inden, Year of Birth 1971
- Diploma Computer Science, C.v.O. Uni Oldenburg
- ~8 ¼ Years SSE bei Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Years TPL, SA bei IVU Traffic Technologies AG in Aachen
- ~4 ¼ Years LSA / Trainer bei Zühlke Engineering AG in Zürich
- ~3 Years TL / CTO bei Direct Mail Informatics / ASMIQ in Zürich
- Independent Consultant, Conference Speaker and Trainer
- Author @ dpunkt.verlag

E-Mail: michael.inden@hotmail.ch
Blog: <https://jaxenter.de/author/minden>





Agenda

Workshop Contents



- **PART 1:** Syntax Enhancements & News and API-Changes in Java 9 to 11
 - **PART 2:** Multi-Threading with CompletableFuture and Reactive Streams
 - **PART 3:** Additional News and Changes in JDK 9 to 11
-
- **PART 4:** Syntax Enhancements & News and API-Changes in Java 12 to 16
 - **PART 5:** News and Changes in Java 12 to 16
 - **PART 6:** What's new in Java 17
-
- **Separate:** Java Modularization

Workshop Contents



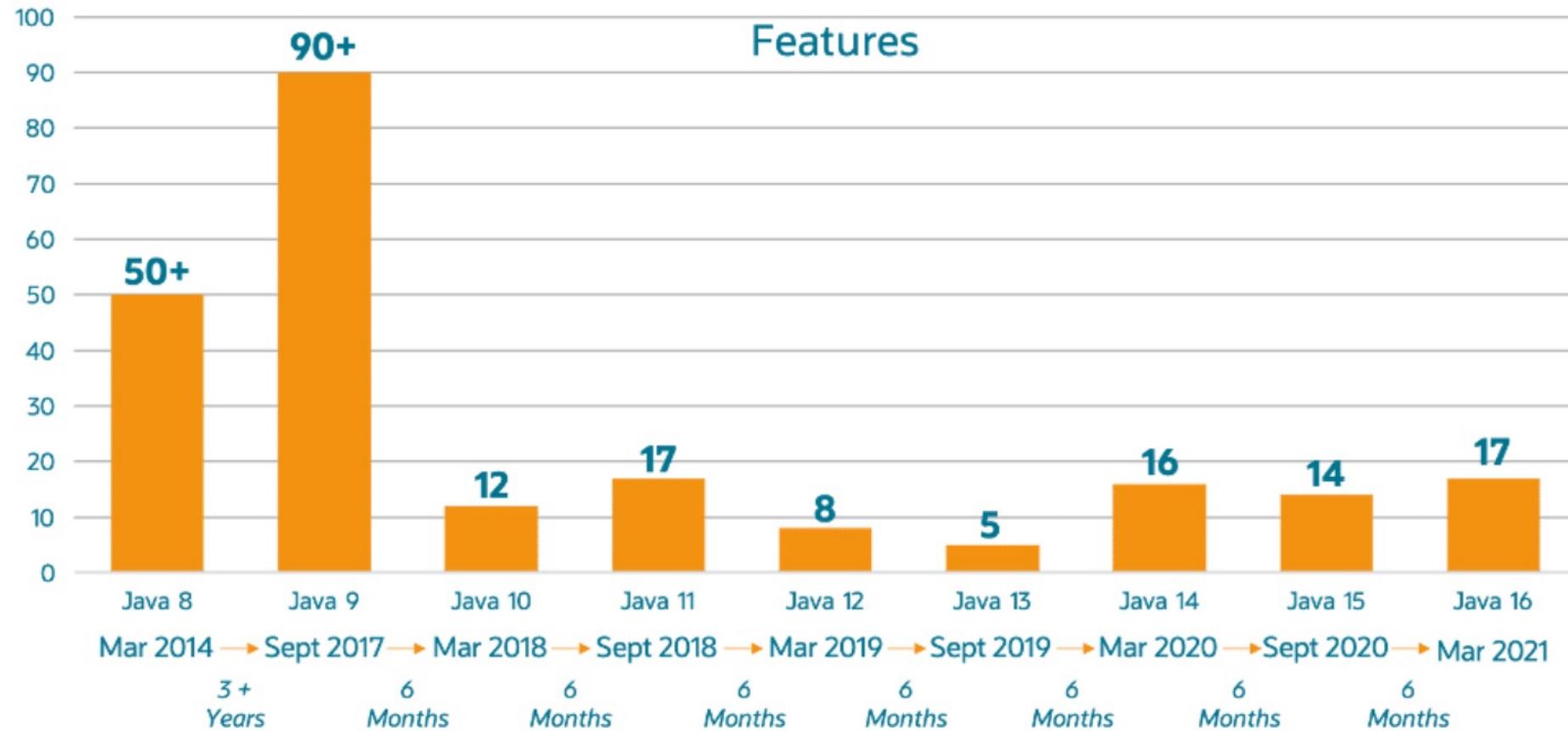
JDK	Release Date	Development Time	LTS	Workshop
Oracle JDK 8	3 / 2014	-	yes, <i>in the mean time commercial</i>	-/-
Oracle JDK 9	9 / 2017	3,5 year	-	Part 1, 2, 3
Oracle JDK 10	3 / 2018	6 months	-	Part 1, 2, 3
Oracle JDK 11	9 / 2018	6 months	yes, <i>commercial</i>	Part 1, 2, 3
Oracle JDK 12	3 / 2019	6 months	-	Part 4, 5
Oracle JDK 13	9 / 2019	6 months	-	Part 4, 5
Oracle JDK 14	3 / 2020	6 months	-	Part 4, 5
Oracle JDK 15	9 / 2020	6 months	-	Part 4, 5
Oracle JDK 16	3 / 2021	6 months	-	Part 4, 5
Oracle JDK 17	9 / 2021	6 months	yes, <i>commercial, but somewhat laxer</i>	Part 6

Long-Term Support Model



- **every three years Long Term Support (LTS) release**
 - get updates over a longer period of time
 - production versions
 - at the moment Java 8, 11 and 17
 - current LTS-Release is Java 17 (September 2021)
- **other versions are "only" intermediate versions**
 - get updates only within 6 months
 - „Previews“
 - Ideal to get to know new features and to experiment (especially privately)

Classification 6 month release cycle





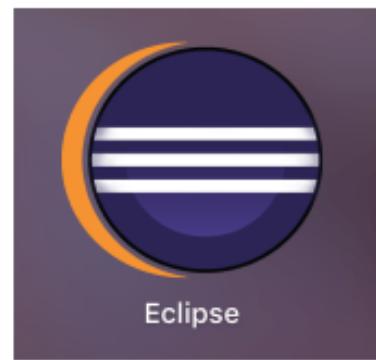
Build-Tools and IDEs



IDE & Tool Support für Java 16 (Java 17 is covered later)



- Current IDEs & Tools normally doing a good job
- Eclipse: Version 2021-06
- IntelliJ: Version 2021.X
- Maven: 3.6.1 / 3.8.2, Compiler-Plugin: 3.8.1
- Gradle: 7.x
- Activation of preview features required
 - In dialogs
 - In the build script



Maven™

 **Gradle**

IDE & Tool Support Java 16



- Eclipse 2021-06
- Activation of preview features required
- Only possible if not yet Java 17



Screenshot of the Eclipse Java Compiler properties dialog for project "Best_of_Java_9_bis_16". The "Java Compiler" tab is selected. The "JDK Compliance" section shows the following settings:

- Enable project specific settings
- Use compliance from execution environment 'JavaSE-16' on the [Java Build Path](#) (with a red arrow pointing to the "16" dropdown)
- Use '--release' option
- Use default compliance settings
- Enable preview features for Java 16 (with a red arrow pointing to this checkbox)

The "Compiler compliance level:" dropdown is set to "16". Other sections include "Preview features with severity level:", "Generated .class files compatibility:", and "Source compatibility:", each with a "16" dropdown.

IDE & Tool Support



- Activation of preview features required

The screenshot shows the IntelliJ IDEA Project Structure dialog. The left sidebar has 'Project Settings' selected, with options for Project, Modules, Libraries, Facets, Artifacts, Platform Settings, SDKs, Global Libraries, and Problems. The main area shows 'Project name: Java15-IDEA-Examples'. Under 'Project SDK:', '15 java version "15"' is selected, indicated by a red arrow. Under 'Project language level:', '15 (Preview) - Sealed types, records, patterns, local enums and interfaces' is selected, indicated by a red arrow. In the bottom right, under 'Java Compiler', 'Override compiler parameters' is set to '--enable-preview', indicated by a large red arrow. The background of the slide features a dark brown gradient with the IntelliJ IDEA logo and the text 'IntelliJ IDEA'.

IDE & Tool Support Java 16



- Activation of preview features required

```
sourceCompatibility=16  
targetCompatibility=16
```

```
// Important for some syntax news  
tasks.withType(JavaCompile) {  
    options.compilerArgs += ["--enable-preview"]  
}
```



IDE & Tool Support



- Activation of preview features required

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
      <source>16</source>
      <target>16</target>
      <!-- Important for some syntax news -->
      <compilerArgs>--enable-preview</compilerArgs>
    </configuration>
  </plugin>
</plugins>
```





New License Model





- If you are planning to or already distribute your software commercially, you should definitely consider the new release policy of Oracle when downloading Java 11!
- The Oracle JDK is now unfortunately NOT free for some scenarios - but during development it can still be used free of charge.
- Alternatives: OpenJDK (<https://openjdk.java.net/>) or Adopt Open JDK (<https://adoptopenjdk.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.



PART 1:

Syntax Enhancements &

News and API-Changes in

Java 9 to 11



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



- **@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()  
{  
    // ...  
}
```

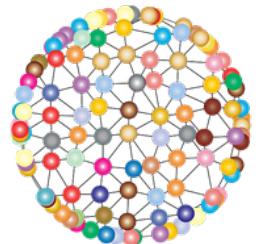
Underscore as Identifier



- `_` 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();  
          ^
```



Why?

Underscore as Identifier



- Idea: enable marking of unused parameters in lambdas for the future.
- Among others Python allows similar
- Syntax suggestion, but not implemented so far:

$(x, y, \textcolor{blue}{z}) \rightarrow x + y$

$(x, y, _) \rightarrow x + y$

$(x, \textcolor{blue}{y}, z) \rightarrow x * z$



$(x, _, z) \rightarrow x * z$

$(\text{person}, \textcolor{blue}{str}) \rightarrow \text{person.getAge}()$

$(\text{person}, _) \rightarrow \text{person.getAge}()$



Syntax Enhancements in Java 10 / 11



Local Variable Type Inference => var (JDK 10)



- 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



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

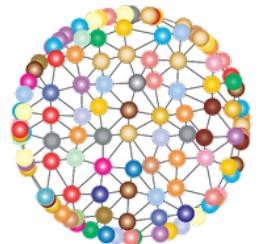
```
// var => Set<Map.Entry<String, Long>>
var entries = personAgeMapping.entrySet();
```

```
// 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?**

Local Variable Type Inference => var



- 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



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

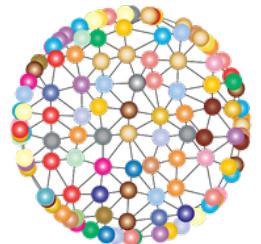
```
final List<String> names = new ArrayList<>();
names.add("Expected");
// names.add(42); // Compile error
```

- **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



- **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

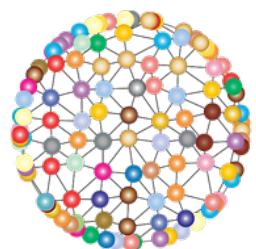


- Short recapitulation: var is intended for **local variables** that are initialized directly.
- **More things that cause compilation errors:**

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

- 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
```



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.



News and API-Changes in Java 9 to 11

- Process API
- Stream-API
- Optional<T>
- Collection Factory Methods
- Strings (JDK 11)



Process API



Process Handling with Java 8 and earlier

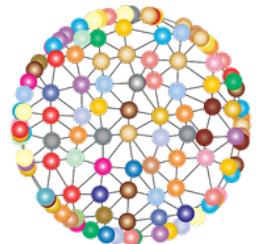


- Limited control and management of operating system processes prior to Java 9
- Example **Read PID: Often a different implementation is needed for each platform**

```
private static long getPidOldStyle() 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().pid();
```

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



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(...)`

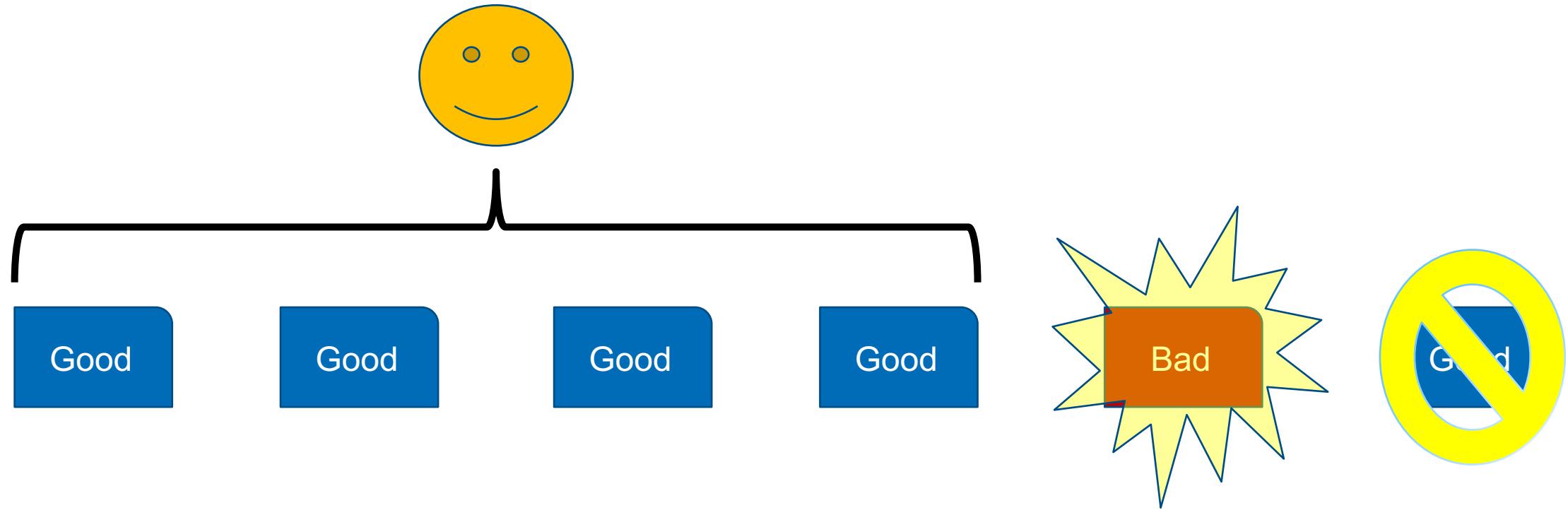
`takeWhile(Predicate<T>)` – processes elements as long
as the condition is met

`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?

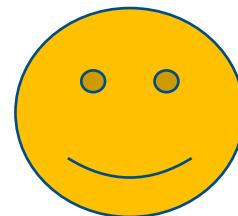


Google

Google-Suche

Auf gut Glück!

Google angeboten in: English Français Italiano Rumantsch



takeWhile(...)



```
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.  
            arrow [ ] 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(...)



```
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(...)



```
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



- 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
```



Optional<T>



Optional<T>



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

- ▼ C F Optional<T>

- empty() <T> : Optional<T>
- of(T) <T> : Optional<T>
- ofNullable(T) <T> : Optional<T>
- equals(Object) : boolean
- filter(Predicate<? super T>) : Optional<T>
- flatMap(Function<? super T, Optional<U>>) <U> : Optional<U>
- get() : T
- hashCode() : int
- ifPresent(Consumer<? super T>) : void
- isPresent() : boolean
- map(Function<? super T, ? extends U>) <U> : Optional<U>
- orElse(T) : T
- orElseGet(Supplier<? extends T>) : T
- orElseThrow(Supplier<? extends X>) <X extends Throwable> : T
- toString() : String

Optional<T> in JDK 8 / JDK 9



▼ C F Optional<T>

- S empty() <T> : void
- S of(T) <T> : Optional<T>
- S ofNullable(T) <T> : Optional<T>
- ▲ equals(Object) : boolean
- filter(Predicate<? super T>) : Optional<T>
- flatMap(Function<? super T, ? extends Optional<? extends U>>) <U> : Optional<U>
- get() : T
- ▲ hashCode() : int
- ifPresent(Consumer<? super T>) : void
- isPresent() : boolean
- map(Function<? super T, ? extends U>) <U> : Optional<U>
- orElse(T) : T
- orElseGet(Supplier<? extends T>) : T
- orElseThrow(Supplier<? extends X>) <X extends Throwable> : T
- ▲ toString() : String

▼ C F Optional<T>

- S empty() <T> : Optional<T>
- S of(T) <T> : Optional<T>
- S ofNullable(T) <T> : Optional<T>
- ▲ equals(Object) : boolean
- filter(Predicate<? super T>) : Optional<T>
- flatMap(Function<? super T, ? extends Optional<? extends U>>) <U> : Optional<U>
- get() : T
- ▲ hashCode() : int
- ifPresent(Consumer<? super T>) : void
- ifPresentOrElse(Consumer<? super T>, Runnable) : void
- isPresent() : boolean
- map(Function<? super T, ? extends U>) <U> : Optional<U>
- or(Supplier<? extends Optional<? extends T>>) : Optional<T>
- orElse(T) : T
- orElseGet(Supplier<? extends T>) : T
- orElseThrow(Supplier<? extends X>) <X extends Throwable> : T
- stream() : Stream<T>
- ▲ toString() : String

Optional<T>



- 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>



- 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



▼ C F Optional<T>

- S empty() <T> : Optional<T>
- S of(T) <T> : Optional<T>
- S ofNullable(T) <T> : Optional<T>
- ▲ equals(Object) : boolean
- filter(Predicate<? super T>) : Optional<T>
- flatMap(Function<? super T, ? extends Optional<? extends U>>) <U> : Optional<U>
- get() : T
- ▲ hashCode() : int
- ifPresent(Consumer<? super T>) : void
- **ifPresentOrElse(Consumer<? super T>, Runnable) : void**
- isPresent() : boolean
- map(Function<? super T, ? extends U>) <U> : Optional<U>
- or(Supplier<? extends Optional<? extends T>>) : Optional<T>
- orElse(T) : T
- orElseGet(Supplier<? extends T>) : T
- orElseThrow(Supplier<? extends X>) <X extends Throwable> : T
- stream() : Stream<T>
- ▲ toString() : String



why ifPresentOrElse(...) ?



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(...) ?



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)



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



▼ C F Optional<T>

- S empty() <T> : Optional<T>
- S of(T) <T> : Optional<T>
- S ofNullable(T) <T> : Optional<T>
- ▲ equals(Object) : boolean
- filter(Predicate<? super T>) : Optional<T>
- flatMap(Function<? super T, ? extends Optional<? extends U>>) <U> : Optional<U>
- get() : T
- ▲ hashCode() : int
- ifPresent(Consumer<? super T>) : void
- ifPresentOrElse(Consumer<? super T>, Runnable) : void
- isPresent() : boolean
- map(Function<? super T, ? extends U>) <U> : Optional<U>
- or(Supplier<? extends Optional<? extends T>>) : Optional<T>
- orElse(T) : T
- orElseGet(Supplier<? extends T>) : T
- orElseThrow(Supplier<? extends X>) <X extends Throwable> : T
- stream() : Stream<T>
- ▲ toString() : String



why or(...) ?



```
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");  
    }  
}
```

JDK 8

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



▼ C F Optional<T>

- S empty() <T> : Optional<T>
- S of(T) <T> : Optional<T>
- S ofNullable(T) <T> : Optional<T>
- ▲ equals(Object) : boolean
- filter(Predicate<? super T>) : Optional<T>
- flatMap(Function<? super T, ? extends Optional<? extends U>>) <U> : Optional<U>
- get() : T
- ▲ hashCode() : int
- ifPresent(Consumer<? super T>) : void
- ifPresentOrElse(Consumer<? super T>, Runnable) : void
- isPresent() : boolean
- map(Function<? super T, ? extends U>) <U> : Optional<U>
- or(Supplier<? extends Optional<? extends T>>) : Optional<T>
- orElse(T) : T
- orElseGet(Supplier<? extends T>) : T
- orElseThrow(Supplier<? extends X>) <X extends Throwable> : T
- **stream() : Stream<T>**
- ▲ toString() : String



why stream() ?



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()



- `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);  
    }  
}
```



Optional<T> in JDK 10 & 11



Optional<T> (Recap)



- Introduced with Java 8, facilitates the handling and modeling of optional values.
- Java 9 add three valuable methods

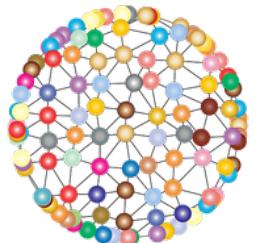
▼ **G F** `Optional<T>`

- `S empty() <T> : Optional<T>`
- `S of(T) <T> : Optional<T>`
- `S ofNullable(T) <T> : Optional<T>`
- `△ equals(Object) : boolean`
- `filter(Predicate<? super T>) : Optional<T>`
- `flatMap(Function<? super T, ? extends Optional<? extends U>>) <U> : Optional<U>`
- `get() : T`
- `△ hashCode() : int`
- `ifPresent(Consumer<? super T>) : void`
- `ifPresentOrElse(Consumer<? super T>, Runnable) : void`
- `isPresent() : boolean`
- `map(Function<? super T, ? extends U>) <U> : Optional<U>`
- `or(Supplier<? extends Optional<? extends T>>) : Optional<T>`
- `orElse(T) : T`
- `orElseGet(Supplier<? extends T>) : T`
- `orElseThrow(Supplier<? extends X>) <X extends Throwable> : T`
- `stream() : Stream<T>`
- `△ toString() : String`





**What is potentially
problematic with `get()`?**



Optional<T>



- The `get()` method for accessing a value seems to be harmless
- Sometimes `get()` get's called without check for existance with `isPresent()`
- If a value is missing this will result in a `NoSuchElementException`.
- **Normally a `get()` method is not necessarily expected to throw an exception.**
- **NEW IN JDK 10:**
- `orElseThrow()` as an alternative to `get()`, to express this fact directly in the API

Optional<T>



- Let's do some experiments 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
```

Class `java.util.Optional<T>`



- 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");
```



Collection Factory Methods



Collection Factory Methods Intro



- 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...**

Collection Factory Methods



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

Collection Factory Methods



- 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



- 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



```
static <E> List<E> of() {
    return ImmutableList.of();
}

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

...

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



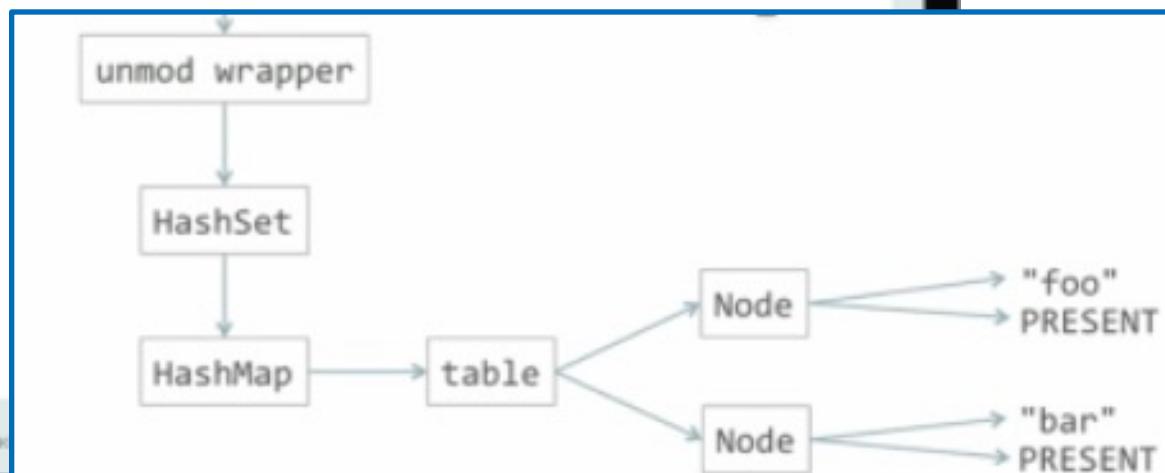
#JavaYoungPups

Space Efficiency

- Consider an unmodifiable set containing two strings

```
Set<String> set = new HashSet<>(3); // 3 is the number of buckets
set.add("foo");
set.add("bar");
set = Collections.unmodifiableSet(set);
```

- How much space does this take? Count objects.
 - 1 unmodifiable wrapper
 - 1 HashSet
 - 1 HashMap
 - 1 Object[] table of length 3
 - 2 Node objects, one for each element





#JavaYoungPups

Space Efficiency

- Proposed field-based set implementation
`Set<String> set = Set.of("foo", "bar");`
- One object, two fields
 - 20 bytes, compared to 152 bytes for conventional structure
- Efficiency gains
 - lower fixed cost: fewer objects created for a collection of any size
 - lower variable cost: fewer bytes overhead per collection element





Unmodifiable Collections Copies and Collectors



Unmodifiable Collections – Copies



- Java 9 brought Collection Factory Methods to create unmodifiable collections in a readable 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



```
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
```



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 new methods of class `java.util.stream.Collectors` provide exactly this:
 - `toUnmodifiableList()`,
 - `toUnmodifiableSet()` and
 - `toUnmodifiableMap()`

Unmodifiable Collections – Copies



```
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 duplicates



DEMO

ImmutableCollectionsAndCopyExample



Stream API – Special Collectors





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.



Two newly available collectors

- **filtering()** – Filtering the elements of the stream
- **flatMapting()** – 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



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

Example for the entry:

```
final Set<String> result1 = programming1.filter(name -> name.contains("Java")).  
                                collect(toSet());
```

With new collector:

```
final Set<String> result2 = programming2.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()`

Stream API Collectors in JDK 9



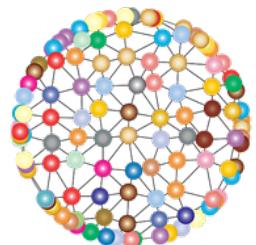
Example for the entry:

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?

Stream API Collectors in JDK 9



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



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!

Stream API Collectors in JDK 9



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()

Stream API Collectors in JDK 9



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]}
```



Extensions in the Class String



Enhancements in `java.lang.String`



- 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()`



- 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 delimiter = " -* ";
    System.out.println(delimiter.repeat(6));
}
```

=>

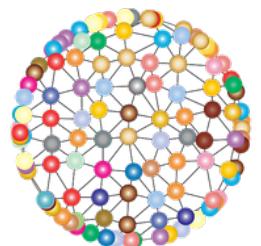
```
*****
 -* - -* - -* - -* - -* -
```

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



```
"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



```
"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()`



- 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'
```



Exercises PART 1

<https://github.com/Michaeli71/JAX-London-Best-of-Java9-17>



PART 2: Multi-Threading and with CompletableFuture

Multi-Threading and the Class CompletableFuture<T>



- 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>



- 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());
```

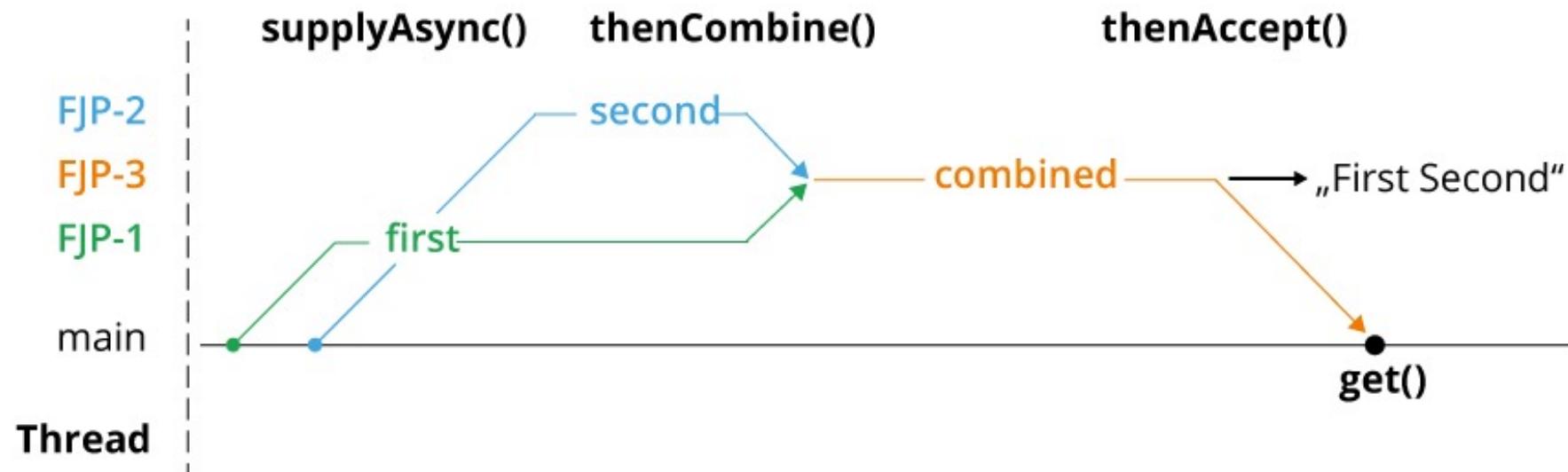
Introduction to CompletableFuture<T>



```
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());
```



Multi-Threading and the Class CompletableFuture<T>



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>



- **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);
}
```

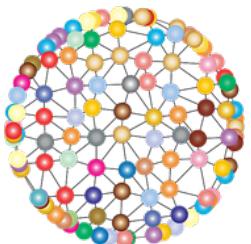
Multi-Threading and the Class CompletableFuture<T>



```
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>?**

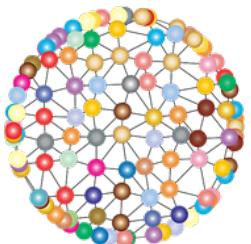
Multi-Threading and the Class CompletableFuture<T>



```
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>



- 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>



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

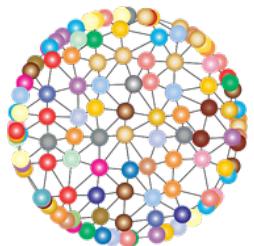
- Provide a fallback value if the data cannot be determined:

```
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>



- 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>



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>



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.

Multi-Threading and the Class CompletableFuture<T>



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.



Exercises PART 2

<https://github.com/Michaeli71/JAX-London-Best-of-Java9-17>



PART 3: Other News and Changes in APIs

- Date API
 - java.util.Arrays
 - java.io.InputStream and Reader
 - Files
 - Misc
 - HTTP/2
 - Direct Compilation
 - JShell
-



Date API Enhancements



class LocalDate



- **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 = myBirthday.datesUntil(christmas);
    daysUntil.skip(150).limit(4).forEach(System.out::println);

    System.out.println("\n3-Month-Stream");
    final Stream<LocalDate> monthsUntil =
        myBirthday.datesUntil(christmas, Period.ofMonths(3));
    monthsUntil.limit(3).forEach(System.out::println);
}
```

Class LocalDate



- 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 3 months:

Day-Stream

1971-07-07

1971-07-08

1971-07-09

1971-07-10

3-Month-Stream

1971-02-07

1971-05-07

1971-08-07

class Duration



- **divideBy()** – divides by given unit
- **truncateTo()** – truncates to given unit

```
public static void main(String[] args)
{
    final Duration tenDaysSevenHoursThirdMinutes = Duration.ofDays(10).
                                                plusHours(7).
                                                plusMinutes(30);

    // JDK 9: divideBy(Duration)
    final long wholeDays = tenDaysSevenHoursThirdMinutes.dividedBy(Duration.ofDays(1));
    final long wholeHours = tenDaysSevenHoursThirdMinutes.dividedBy(Duration.ofHours(1));
    final long wholeMinutes = tenDaysSevenHoursThirdMinutes.dividedBy(Duration.ofMinutes(15));
    System.out.println("wholeDays:      " + wholeDays);
    System.out.println("wholeHours:     " + wholeHours);
    System.out.println("whole15Minutes: " + wholeMinutes);

    // JDK 9: truncatedTo(TemporalUnit)
    System.out.println("truncatedTo(DAYS):   " + tenDaysSevenHoursThirdMinutes.truncatedTo(ChronoUnit.DAYS));
    System.out.println("truncatedTo(HOURS):  " + tenDaysSevenHoursThirdMinutes.truncatedTo(ChronoUnit.HOURS));
    System.out.println("truncatedTo(MINUTES): " + tenDaysSevenHoursThirdMinutes.truncatedTo(ChronoUnit.MINUTES));
}
```

class Duration



```
public static void main(String[] args)
{
    final Duration tenDaysSevenHoursThirdMinutes = Duration.ofDays(10).
                                                plusHours(7).
                                                plusMinutes(30);

    // JDK 9: divideBy(Duration)
    final long wholeDays = tenDaysSevenHoursThirdMinutes.dividedBy(Duration.ofDays(1));
    final long wholeHours = tenDaysSevenHoursThirdMinutes.dividedBy(Duration.ofHours(1));
    final long wholeMinutes = tenDaysSevenHoursThirdMinutes.dividedBy(Duration.ofMinutes(15));
    System.out.println("wholeDays: " + wholeDays);
    System.out.println("wholeHours: " + wholeHours);
    System.out.println("whole15Minutes: " + wholeMinutes);

    // JDK 9: truncatedTo(TemporalUnit)
    System.out.println("truncatedTo(DAYS): " + tenDaysSevenHoursThirdMinutes.truncatedTo(ChronoUnit.DAYS));
    System.out.println("truncatedTo(HOURS): " + tenDaysSevenHoursThirdMinutes.truncatedTo(ChronoUnit.HOURS));
    System.out.println("truncatedTo(MINUTES): " + tenDaysSevenHoursThirdMinutes.truncatedTo(ChronoUnit.MINUTES));
}
```

wholeDays:	10
wholeHours:	247
howMany15Minutes:	990
truncatedTo(DAYS):	PT240H
truncatedTo(HOURS):	PT247H
truncatedTo(MINUTES):	PT247H30M

class Duration



- **toXXX()** – converts into given unit
- **toXXXPart()** – extracts the part of the given unit

```
public static void main(final String[] args)
{
    final Duration tenDaysSevenHoursThirdMinutes = Duration.ofDays(10).
                                                plusHours(7).
                                                plusMinutes(30);

    // JDK 9:toXXX() und toXXXPart()
    System.out.println("toDays(): " + tenDaysSevenHoursThirdMinutes.toDays());
    System.out.println("toDaysPart(): " + tenDaysSevenHoursThirdMinutes.toDaysPart());
    System.out.println("toHours(): " + tenDaysSevenHoursThirdMinutes.toHours());
    System.out.println("toHoursPart(): " + tenDaysSevenHoursThirdMinutes.toHoursPart());
    System.out.println("toMinutes(): " + tenDaysSevenHoursThirdMinutes.toMinutes());
    System.out.println("toMinutesPart(): " + tenDaysSevenHoursThirdMinutes.toMinutesPart());
}
```

class Duration



```
public static void main(final String[] args)
{
    final Duration tenDaysSevenHoursThirdMinutes = Duration.ofDays(10).
                                                plusHours(7).
                                                plusMinutes(30);

    // JDK 9:toXXX() und toXXXPart()
    System.out.println("toDays():           " + tenDaysSevenHoursThirdMinutes.toDays());
    System.out.println("toDaysPart():      " + tenDaysSevenHoursThirdMinutes.toDaysPart());
    System.out.println("toHours():          " + tenDaysSevenHoursThirdMinutes.toHours());
    System.out.println("toHoursPart():     " + tenDaysSevenHoursThirdMinutes.toHoursPart());
    System.out.println("toMinutes():        " + tenDaysSevenHoursThirdMinutes.toMinutes());
    System.out.println("toMinutesPart():   " + tenDaysSevenHoursThirdMinutes.toMinutesPart());
}
```

toDays():	10
toDaysPart():	10
toHours():	247
toHoursPart():	7
toMinutes():	14850
toMinutesPart():	30



java.util.Arrays



Misc – Class 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



- **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
```

Misc – Class Arrays



- `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



java.io.InputStream / Reader



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`

class java.io.Reader (JDK 10)



- **long transferTo(Writer)**

All characters are transferred from the reader to the given writer - this functionality exists in the `InputStream` class since Java 9. Why get reader improved only in Java 10?

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

sr.transferTo(sw);

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

=>

Sw: Hello



Extensions in the Class Files



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
```

Utility class `java.nio.file.Files`



- **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
```



Misc





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: Enumeration => 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
```

```
Problems @ Javadoc Declaration Search Console X
<terminated> UnicodePropertiesExample [Java Application] /Library/Java/JavaVi
WARNING: Using incubator modules: jdk.incubator.httpclient
sigma = Σ / Σ
money = € / €
ohm = Ω / Ω
coffee = ☕ / ☕
seven = 7 / 7
sun = ☀ / ☀
```

Class `java.util.function.Predicate<T>`



- 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);
```

Misc – HTML 5 Java Doc



Java™ Platform Standard Ed. 9 DRAFT 9-ea+134

All Classes All Modules Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer

All Classes

AboutEvent
AboutHandler
AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractAnnotationValueVisitor9
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractElementVisitor9
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache

https://www.amazon.de/gp/bestsellers/books/166030031/ref=pd_zg_hrsr_b_1_5_just

javadoc jdk 9

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.httpclient
Package java.net.http

Class `HttpResponse`

`java.lang.Object`
`java.net.http.HttpResponse`

`public abstract class HttpResponse
extends Object`

Represents a response to a `HttpRequest`. A `HttpResponse` is available when the response status code and headers have been received, but before the response body is received.

Methods are provided in this class for accessing the response headers, and status code immediately and also methods for retrieving the response body. Static methods are provided which implement `HttpResponse.BodyProcessor` for standard body types such as `String`, `byte arrays`, `files`.

The `body` or `bodyAsync` which retrieve any response body must be called to ensure that the TCP connection can be re-used subsequently, and any response trailers accessed, if they exist, unless it is known that no response body was received.

Since:
9

Nested Class Summary



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



HTTP/2 API



HTTP-access with Java 8



- **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

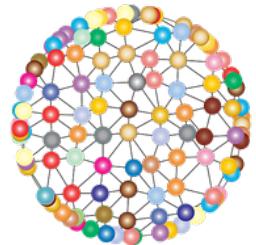


- **Read content as String**

```
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?**

HTTP/2 API Intro



```
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();

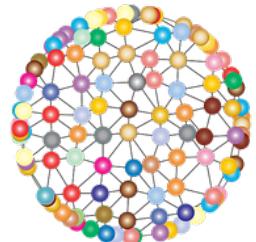
final HttpClient httpClient = HttpClient.newHttpClient();
final HttpResponse<String> response = httpClient.send(request, asString);

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

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



The PO enters the scene:
He likes to have it
asynchronous!



HTTP/2 API Async I



```
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);

wait2secForCompletion(asyncResponse);
if (asyncResponse.isDone())
{
    printResponseInfo(asyncResponse.get());
}
else
{
    asyncResponse.cancel(true);
    System.err.println("timeout");
}
```

HTTP/2 API Async I – waiting with premature termination

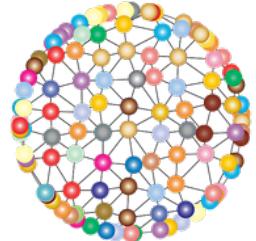


```
static void wait2secForCompletion(final CompletableFuture<?> asyncResponse)
    throws InterruptedException
{
    int i = 0;
    while (!asyncResponse.isDone() && i < 10)
    {
        System.out.println("Step " + i);
        i++;

        Thread.sleep(200);
    }
}
```



**One moment please:
Isn't that old school?
What can we improve on
waiting?**



HTTP/2 API Async II



```
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);

// Wait and process just with CompletableFuture
asyncResponse.thenAccept(response -> printResponseInfo(response)).
    orTimeout(2, TimeUnit.SECONDS).
    exceptionally(ex ->
{
    asyncResponse.cancel(true);
    System.err.println("timeout");
    return null;
});

// CompletableFuture should have time to complete
Thread.sleep(2_000);
```



- **HTTPRequest**

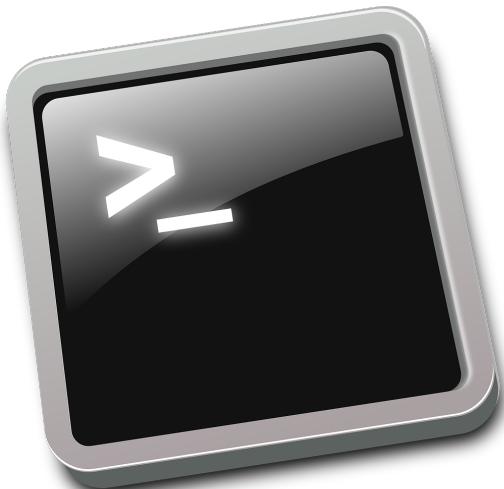
```
.timeout(Duration.ofSeconds(2))  
.header("Content-Type", "application/json")
```

- **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

```
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
```

Direct Compilation – Two Public Classes in 1 File



```
public class PalindromeChecker
{
    public static void main(final String[] args)
    {
        if (args.length < 1)
        {
            System.err.println("input is required!");
            System.exit(1);
        }

        System.out.printf("%s' is a palindrome? => %b %n",
                          args[0], StringUtils.isPalindrome(args[0]));
    }
}

public class StringUtils
{
    public static boolean isPalindrome(String word)
    {
        return new StringBuilder(word).reverse().toString().equalsIgnoreCase(word);
    }
}
```

Direct Compilation – A REST Call With HTTP/2 API



```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse.BodyHandlers;

public class PerformGetWithHttpClient
{
    public static void main(String[] args) throws Exception
    {
        var client = HttpClient.newBuilder().build();
        URI uri = URI.create("https://reqres.in/api/unknown/2");
        var request = HttpRequest.newBuilder().GET().uri(uri).build();

        var response = client.send(request, BodyHandlers.ofString());
        System.out.printf("Response code is: %d %n", response.statusCode());
        System.out.printf("The response body is:%n %s %n", response.body());
    }
}
```

Direct Compilation – Shebang Execution



```
#!/usr/bin/java --source 11
import java.nio.file.*;

public class DirectoryLister
{
    public static void main(String[] args) throws Exception
    {
        var dirName = ".";
        if (args == null || args.length < 1)
        {
            System.err.println("Using current directory as fallback");
        }
        else
        {
            dirName = args[0];
        }

        Files.walk(Paths.get(dirName)).forEach(System.out::println);
    }
}
```

- File must not end with ‘.java’
- File name independent of class name
- File must be executable (chmod +x)



DEMO



JShell



```
Michaels-MBP-2:~ michaeli$ jshell
| Welcome to JShell -- Version 15
| For an introduction type: /help intro

jshell> 2 * 3 * 5 * 7
[$1 ==> 210

jshell> int add(int val1, int val2)
| ...> {
| ...>     return val1 + val2;
| ...> }
| created method add(int,int)

jshell> import java.time.*

jshell> boolean isSunday(LocalDate date)
| ...> {
| ...>     return date.
adjustInto(      atStartOfDay(    atTime(        compareTo(      datesUntil(    equals(        format(
get(            getChronology() getClass()      getDayOfMonth()  getDayOfWeek()  getDayOfYear()  getEra()
getLong(         getMonth()       getMonthValue()  getYear()       hashCode()      isAfter(       isBefore(
isEqual(        isLeapYear()     isSupported()   lengthOfMonth()  lengthOfYear()  minus(        minusDays(
minusMonths(    minusWeeks()    minusYears()    notify()        notifyAll()    plus(         plusDays(
plusMonths(    plusWeeks()     plusYears()    query()        range(        toEpochDay()  toEpochSecond(
jshell> boolean isSunday(LocalDate date){
| ...> {
| ...>     return date.getDayOfWeek() == DayOfWeek.SUNDAY;
| ...> }
| created method isSunday(LocalDate)

jshell> isSunday(LocalDate.of(1971, 2, 7))
[$5 ==> true
```

Java command line jshell



- Code execution without class and method declaration
- Semicolon at end of line can (partially) be omitted
- (partly) no exception handling necessary
- for each command a variable for the return value is automatically created (\$1, \$2, ...)
- declaration of methods and classes possible
- adding of modules possible at startup
- exit jshell: /exit

Java command line jshell



- Command history
 - `!<command>` repeats the last command
 - `/list` List of all entered commands
 - `/<nr>` executes the command with the given number
 - `/reset` clears history
 - `/methods` shows methods
 - `/imports` shows imports
 - `/help` shows help



DEMO

JShell - What we learned so far?



- Perform calculations on the fly
- Define variables
- Define methods
- Practical forward referencing: a method can call methods that are not yet defined
- Practical for **SMALL** experiments
- Since Java 14: editor is much more comfortable, before that rather catastrophic regarding multi-line editing
- 3rd Party & Preview-Features
 - `jshell --class-path myOwnClassPath --enable-preview`



- Create own instances of JShell programmatically (`create()`)
- Execute snippets of code (`eval()`)
- Perform dynamic calculations
- Use it as a kind of replacement for JavaScript-Engine

```
final JShell jshell = JShell.create();
final List<SnippetEvent> result1 = jshell.eval("var name = \"Mike\";");

for (final SnippetEvent se : result1)
{
    System.out.println(se.snippet());
    System.out.println(se.value());

    // leider kein (direkter) Zugriff auf Wert, nur über varValue()
    jshell.variables().map(varSnippet -> "variable: '" + varSnippet.name() + "' / "
                                + "type: " + varSnippet.typeName() + "' / "
                                + "value: " + jshell.varValue(varSnippet))
        .forEach(System.out::println);
}
```



```
final JShell jshell = JShell.create();
final List<SnippetEvent> result1 = jshell.eval("var name = \"Mike\";");

for (final SnippetEvent se : result1)
{
    System.out.println(se.snippet());
    System.out.println(se.value());

    // unfortunately no (direct) access on value, use varValue() instead
    jshell.variables().map(varSnippet -> "variable: '" + varSnippet.name() + "' / "
        "type: " + varSnippet.typeName() + "' / "
        "value: " + jshell.varValue(varSnippet))
        .forEach(System.out::println);
}
```

```
Snippet:VariableKey(name)#1-var name = "Mike";
"Mike"
variable: 'name' / type: String' / value: "Mike"
```

JShell-API



```
try (JShell js = JShell.create())
{
    // ATTENTION: we need the ';' otherwise not processed correctly
    String valA = js.eval("int a = 42;").get(0).value();
    System.out.println("valA = " + valA);
    String valB = js.eval("int b = 7;").get(0).value();
    System.out.println("valB = " + valB);
    String result = js.eval("int result = a / b;").get(0).value();
    System.out.println("Result = " + result);

    js.variables().map(varSnippet -> varSnippet.name() + " => " +
                      varSnippet.source()).foreach(System.out::println);
}
```

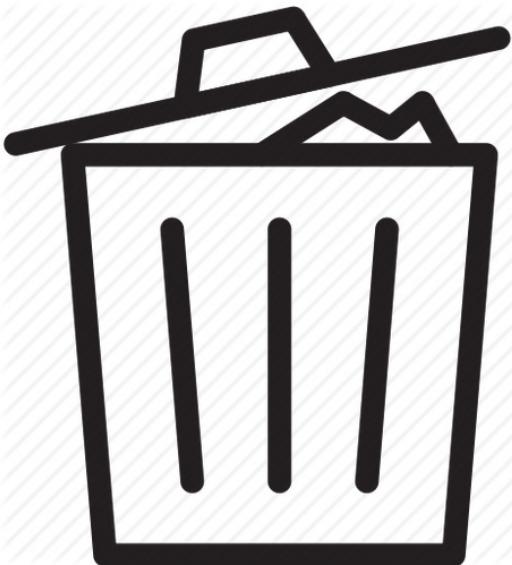
```
valA = 42
valB = 7
result = 6
a => int a = 42;
b => int b = 7;
result => int result = a / b;
```



DEMO



Deprecations



Removed APIs and libraries



- Modularization of the JDK enables the removal of individual modules
- Intersections with the Java EE specifications have been removed:
 - **java.activation** JAF
 - **java.corba** CORBA
 - **java.transaction** JTA
 - **java.xml.bind** JAXB
 - **java.xml.ws** JAX-WS, SAAJ
 - **java.xml.ws.annotation** Common Annotations
 - **java.se.ee** Aggregator-Modul for these moduls

Removed APIs and libraries



- tools related to removed modules got eliminated
- **jdk.xml.ws** (JAX-WS)
 - **wsgen**
 - **wsimport**
- **jdk.xml.bind** (JAXB)
 - **schemagen**
 - **xjc**
- **java.corba** (CORBA)
 - **idlj, orbd, servertool, tnamesrv**



DEMO

JAXBExample



- **Observer und Observable (java.util)**
- **Thread.destroy() und Thread.stop()**
- **Object.finalize()**
- Konstruktoren der Wrapperklassen, z.B. **new Integer()**, **new Long()**
 - stattdessen **valueOf()** oder **parseXXX()** verwenden
- Applets (**java.awt.Applet**, **javax.swing.JApplet**)



- new tool **jdeprscan** : finds usages of classes and methods marked as deprecated
- **jdeprscan --list** shows all parts of the JDK marked as deprecated
- even own projects can be checked if they use deprecated APIs:
 - **jdeprscan target/classes**
 - **jdeprscan myapp.jar**



Exercises PART 3

<https://github.com/Michaeli71/JAX-London-Best-of-Java9-17>



PART 4: Syntax Enhancements in Java 12 to 16

- Syntax Enhancements switch (FINAL)
- Syntax Enhancements Helpful NullPointerExceptions (FINAL)
- Syntax Enhancements Text Blocks (FINAL)
- Syntax Enhancements Records (FINAL)
- Syntax Enhancements bei instanceof (FINAL)
- Syntax Enhancements Local Enums und Interfaces (FINAL)



Syntax Enhancements



Switch Expressions



- **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 numLetters = -1;  
  
switch (day)  
{  
    case MONDAY:  
    case FRIDAY:  
    case SUNDAY:  
        numLetters = 6;  
        break;  
    case TUESDAY:  
        numLetters = 7;  
        break;  
    case THURSDAY:  
    case SATURDAY:  
        numLetters = 8;  
        break;  
    case WEDNESDAY:  
        numLetters = 9;  
        break;  
}
```

Switch Expressions as a Remedy



- Mapping of weekdays to their length... elegantly with modern Java:

```
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 as a Remedy



- Mapping of weekdays to their length... elegantly with modern Java:

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 as a Remedy



- Mapping of weekdays to their length... elegantly with modern Java:

```
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



- 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 as a Remedy



- Mapping months to their names... elegantly with modern Java:

```
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 numChars;

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

Switch Expressions: `yield` with return value



With modern Java it 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);
}
```

Switch Expressions: `yield` with return value



```
public static void main(final String[] args)
{
    DayOfWeek day = DayOfWeek.SUNDAY;

    int numOfLetters = switch (day)
    {
        case MONDAY, FRIDAY, SUNDAY -> {
            if (day == DayOfWeek.SUNDAY)
                System.out.println("SUNDAY is FUN DAY");
            yield 6;
        }
        case TUESDAY              -> 7;
        case THURSDAY, SATURDAY   -> 8;
        case WEDNESDAY             -> 9;
    };
    System.out.println(numOfLetters);
}
```

SUNDAY is FUN DAY
6



N P E

Helpful NullPointerExceptions



```
public static void main(final String[] args)
{
    SomeType a = null;
    a.value = "ERROR";
}
```

Exception in thread "main" java.lang.NullPointerException
at java14.NPE_Example.main(NPE_Example.java:8)

Helpful NullPointerExceptions



```
public static void main(final String[] args)
{
    SomeType a = null;
    a.value = "ERROR";
}
```

Exception in thread "main" java.lang.NullPointerException
at java14.NPE_Example.main(NPE_Example.java:8)

-XX:+ShowCodeDetailsInExceptionMessages

Exception in thread "main" java.lang.NullPointerException: Cannot assign field
"value" because "a" is null
at java14.NPE_Example.main(NPE_Example.java:8)

Helpful NullPointerExceptions



```
public static void main(final String[] args)
{
    try
    {
        final String[] stringArray = { null, null, null };
        final int errorPos = stringArray[2].lastIndexOf("ERROR");
    }
    catch (final NullPointerException e) { e.printStackTrace(); }
    try
    {
        final Integer value = null;
        final int sum = value + 3;
    }
    catch (final NullPointerException e) { e.printStackTrace(); }
}

java.lang.NullPointerException: Cannot invoke
"String.lastIndexOf(String)" because "stringArray[2]" is null
at java14.NPE_Second_Example.main(NPE_Second_Example.java:10)
java.lang.NullPointerException: Cannot invoke
"java.lang.Integer.intValue()" because "value" is null
at java14.NPE_Second_Example.main(NPE_Second_Example.java:20)
```



Text Blocks



Text Blocks



- 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!"
}
"""";
```

Text Blocks (Alignment)



```
jshell> var multiLine = """"  
...>           One  
...>           Two  
...>           Three""""  
multiLine ==> "One\n Two\n Three"
```

```
jshell> var multiLine = """"  
...>           _ One  
...>           _ Two  
...>           _ Three  
...>           _ Four  
...>           _ _ _ _  
multiLine ==> " _ One\n _ Two\n _ Three\n_ _ _ Four\n"
```

Text Blocks



```
String text = """"  
    This is a string splitted \  
    in several smaller \  
    strings.\  
    """";
```

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

This is a string splitted in several smaller strings.



Records





**Wouldn't it be cool to
define DTOs etc. in a
simple way?**

Enhancement Record



```
record MyPoint(int x, int y) { }
```

- simplified form of classes for simple data containers
- Very short, compact notation
- API is derived implicitly from the attributes defined as constructor parameters

```
MyPoint point = new MyPoint(47, 11);
System.out.println("point x:" + point.x());
System.out.println("point y:" + point.y());
```

- Implementations of accessor methods as well as equals() and hashCode() automatically generated and adhering to contract

Enhancement Record

```
record MyPoint(int x, int y) {}
```

```
Michaels-iMac:java14 michaeli$ javap MyPoint
Compiled from "MyPoint.java"
final class java14.MyPoint extends java.lang.Record {
    public java14.MyPoint(int, int);
    public int x();
    public int y();
    public java.lang.String toString();
    public int hashCode();
    public boolean equals(java.lang.Object);
}
```

```
public final class MyPoint
{
    private final int x;
    private final int y;

    public MyPoint(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o)
    {
        if (this == o)
            return true;
        if (o == null || getClass() != o.getClass())
            return false;

        MyPoint point = (MyPoint) o;
        return x == point.x && y == point.y;
    }

    @Override
    public int hashCode()
    {
        return Objects.hash(x, y);
    }

    @Override
    public String toString()
    {
        return "MyPoint[x=" + x + ", y=" + y + "]";
    }

    // Zugriffsmethoden auf x und y
}
```

Records additional constructors and methods



```
record MyPoint(int x, int y)
{
    public MyPoint(String values)
    {
        this(Integer.parseInt(values.split(",")[0]),
              Integer.parseInt(values.split(",")[1]));
    }

    public String shortForm()
    {
        return "[" + x + ", " + y + "]";
    }
}
```

```
var topLeft = new MyPoint(17, 19);
System.out.println(topLeft);
System.out.println(topLeft.shortForm());
```

MyPoint[x=17, y=19]
[17, 19]

Records for DTOs / Parameter Value Objects



```
record TopLeftWidthAndHeight(MyPoint topLeft, int width, int height) { }

record ColorAndRgbDTO(String name, int red, int green, int blue) { }

record PersonDTO(String firstname, String lastname, LocalDate birthday) { }

record Rectangle(int x, int y, int width, int height)
{
    Rectangle(TopLeftWidthAndHeight pointAndDimension)
    {
        this(pointAndDimension.topLeft().x, pointAndDimension.topLeft().y,
              pointAndDimension.width, pointAndDimension.height);
    }
}
```

Records for Complex Return Types or Parameters



```
record IntStringReturnValue(int code, String info) { }
record IntListReturnValue(int code, List<String> values) { }
```

```
record ReturnTuple(String first, String last, int amount) { }
record CompoundKey(String name, int age) { }
```

```
IntStringReturnValue calculateTheAnswer()
```

```
{  
    // Some complex stuff here  
    return new IntStringReturnValue(42, "the answer");  
}
```

```
IntListReturnValue calculate(CompoundKey inputKey)
```

```
{  
    // Some complex stuff here  
    return new IntListReturnValue(201,  
        List.of("This", "is", "a", "complex", "result"));  
}
```

Records modeling Tupels? – Excursion Pair<T>



- What's wrong with this self made pair?

```
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 + "]";
    }
}
```



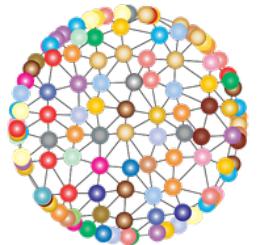
**What is actually missing
there? What is perhaps
disturbing there?**

Records for modelling Pairs and Tupels



```
record IntIntPair(int first, int second) {};
record StringIntPair(String name, int age) {};
record Pair<T1, T2>(T1 first, T2 second) {};
record Top3Favorites(String top1, String top2, String top3) {};
record CalcResultTuple(int min, int max, double avg, int count) {};
```

- **Extremely little writing effort**
- **Very practical for Pair, Tuples etc.**
- **Records work great with primitive types**
- **Implementations of accessor methods as well as equals() and hashCode() automatically and adhering to contracts**



**That's cool ... BUT: How
can I integrate validity
checks?**

Records with validity checks



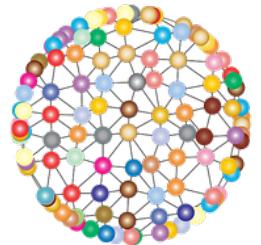
```
record ClosedInterval(int lower, int upper)
{
    public ClosedInterval(int lower, int upper)
    {
        if (lower > upper)
        {
            String errorMsg = String.format("invalid: %d (lower) >= %d (upper)",
                                             lower, upper);
            throw new IllegalArgumentException(errorMsg);
        }

        this.lower = lower;
        this.upper = upper;
    }
}
```

Records with validity checks (short cut)



```
record ClosedInterval(int lower, int upper)
{
    public ClosedInterval
    {
        if (lower > upper)
        {
            String errorMsg = String.format("invalid: %d (lower) >= %d (upper)",
                                             lower, upper);
            throw new IllegalArgumentException(errorMsg);
        }
    }
}
```



**Last question for records:
Is this all combinable?**

Example: All in



```
record MultiTypes<K, V, T> (Map<K, V> mapping, T info)
{
    public void printTypes()
    {
        System.out.println("mapping type: " + mapping.getClass());
        System.out.println("info type: " + info.getClass());
    }
}
```

```
record ListRestrictions<T> (List<T> values, int maxSize)
{
    public ListRestrictions
    {
        if (values.size() > maxSize)
            throw new IllegalArgumentException(
                "too many entries! got: " + values.size() +
                ", but restricted to " + maxSize);
    }
}
```



Pattern Matching instanceof



Pattern Matching instanceof



- OLD STYLE

```
final Object obj = new Person("Michael", "Inden");
if (obj instanceof Person)
{
    final Person person = (Person) obj;
    // ... Zugriff auf person...
}
```



**Always these casts...
Couldn't it be easier?**

Pattern Matching instanceof



- **OLD STYLE**

```
final Object obj = new Person("Michael", "Inden");
if (obj instanceof Person)
{
    final Person person = (Person) obj;
    // ... Access to person...
}
```

- **NEW STYLE**

```
if (obj instanceof Person person)
{
    // here is is possible to access variable person directly
}
```

Pattern Matching instanceof



```
Object obj2 = "Hello Java 14";
```

```
if (obj2 instanceof String str)
{
    // here it is possible to use str without Cast
    System.out.println("Length: " + str.length());
}
else
{
    // here we have no access to str
    System.out.println(obj.getClass());
}
```

Pattern Matching instanceof



```
if (obj2 instanceof String str2 && str2.length() > 5)
{
    System.out.println("Length: " + str2.length());
}
```

Local Enums and Interfaces



```
public class LocalEnumsAndInterfacesExamples
{
    public static void main(String[] args)
    {
        // Java 15 and later, vorher Compile-Error:
        // Local enums are not supported at language level '14'
        enum LocalEnumState
        {
            BAD, GOOD, UNKNOWN
        }

        // Java 15 and later, vorher Compile-Error:
        // Local interfaces are not supported at language level '14'
        interface Evaluationable
        {
            LocalEnumState evaluate(String info);
        }
    ...
}
```

Local Enums and Interfaces



```
public class LocalEnumsAndInterfacesExamples
{
```

```
...
```

```
class AlwaysBad implements Evaluationable
{
    @Override
    public LocalEnumState evaluate(String info)
    {
        return LocalEnumState.BAD;
    }
}
```

```
System.out.println(new AlwaysBad().evaluate("DOES NOT MATTER"));
```

```
}
```

```
}
```



PART 5: What's new in Java 12 to 16

- [CompactNumberFormat](#)
 - [Files](#)
 - [Teeing\(\)-Kollektor](#)
 - [JMH \(Microbenchmarks\)](#)
-



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



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

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

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

private static void formatNumbers(final String style,
                                 final DateFormat 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(DateFormat.Style.SHORT);
    formatNumbers("SHORT", shortFormat);

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

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

private static void formatNumbers(final String style,
                                 final DateFormat 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(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"));
}
```

US/SHORT parsing:

1
1000
1000000
1000000000

US/LONG parsing:

1000
1000000
1000000000

Utility-Class CompactNumberformat – rounding



```
System.out.println(compactNumberFormat.format(990L)); // 990  
System.out.println(compactNumberFormat.format(999L)); // 999
```

```
System.out.println(compactNumberFormat.format(1_499L)); // 1k  
System.out.println(compactNumberFormat.format(1_500L)); // 2k  
System.out.println(compactNumberFormat.format(1_999L)); // 2k
```

```
System.out.println(compactNumberFormat.format(1_499_999)); // 1m  
System.out.println(compactNumberFormat.format(1_500_000)); // 2m  
System.out.println(compactNumberFormat.format(1_567_890)); // 2m
```

Utility-Class CompactNumberformat – rounding



```
compactNumberFormat.setMinimumFractionDigits(1);

System.out.println(compactNumberFormat.format(990L)); // 990
System.out.println(compactNumberFormat.format(999L)); // 999

System.out.println(compactNumberFormat.format(1_499L)); // 1,5k
System.out.println(compactNumberFormat.format(1_500L)); // 1,5k
System.out.println(compactNumberFormat.format(1_999L)); // 2,0k

System.out.println(compactNumberFormat.format(1_499_999)); // 1,5m
System.out.println(compactNumberFormat.format(1_500_000)); // 1,5m
System.out.println(compactNumberFormat.format(1_567_890)); // 1,6m
```

Utility-Class CompactNumberformat – Customization



```
final String[] compactPatterns = { "", "", "",  
"0 [KB]", "00 [KB]", "000 [KB]", "0 [MB]", "00 [MB]", "000 [MB]",  
"0 [GB]", "00 [GB]", "000 [GB]", "0 [TB]", "00 [TB]", "000 [TB]" };  
  
final DecimalFormat decimalFormat = (DecimalFormat)  
NumberFormat.getNumberInstance(Locale.GERMANY);  
  
final CompactNumberFormat customCompactNumberFormat = new  
CompactNumberFormat(decimalFormat.toPattern(),  
decimalFormat.getDecimalFormatSymbols(), compactPatterns);  
  
customCompactNumberFormat.setMinimumFractionDigits(1);  
System.out.println(customCompactNumberFormat.format(990L));  
System.out.println(customCompactNumberFormat.format(999L));  
System.out.println(customCompactNumberFormat.format(1_499L));  
System.out.println(customCompactNumberFormat.format(1_500L));  
System.out.println(customCompactNumberFormat.format(1_999L));  
System.out.println(customCompactNumberFormat.format(1_499_999));  
System.out.println(customCompactNumberFormat.format(1_500_000));  
System.out.println(customCompactNumberFormat.format(1_567_890));
```

990
999
1,5 [KB]
1,5 [KB]
2,0 [KB]
1,5 [MB]
1,5 [MB]
1,6 [MB]

Utility-Class CompactNumberformat – Strange things



```
final DecimalFormat currencyFormat = (DecimalFormat)
                    NumberFormat.getCurrencyInstance();
final NumberFormat cnf = CompactNumberFormat.getCompactNumberInstance(Locale.GERMAN,
                    NumberFormat.Style.SHORT);
cnf.setMinimumFractionDigits(2);
```

```
System.out.println(currencyFormat.format(1234.455));
System.out.println(currencyFormat.format(1234.445));
System.out.println(currencyFormat.format(12345));
System.out.println(currencyFormat.format(1234567));
```

```
CHF 1'234.45
CHF 1'234.44
CHF 12'345.00
CHF 1'234'567.00
```

```
System.out.println(cnf.format(1234.455));
System.out.println(cnf.format(1234.445));
System.out.println(cnf.format(12345));
System.out.println(cnf.format(1234567));
```

```
1.234
1.234
12.345
1,23 Mio.
```



Enhancements in class Files



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



- 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 method **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`



```
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`



```
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`



```
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



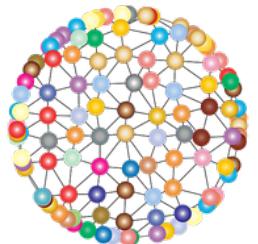
Teeing() Collector





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 toUnmodifiableXyz()!**



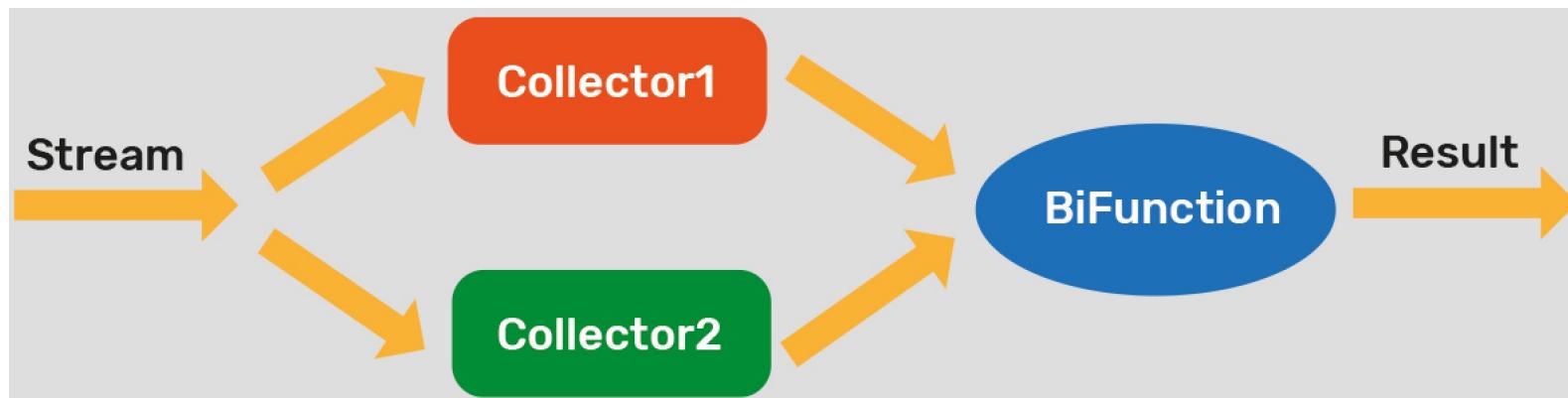
What is missing?



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



```
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]
```

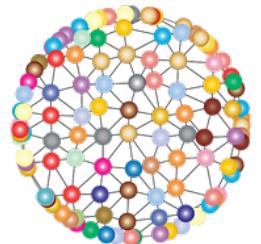
Excursion Pair<T>



```
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 + "]";
    }
}
```

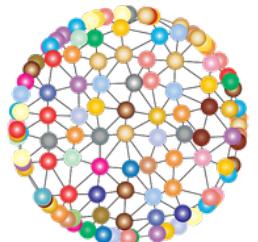


**How about Map.Entry as
a replacement for your
own pair?**



Not so good!!

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





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);
```



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()),
    [Michael, Mike]
    filtering(endsWithM, toList()),
    (list1, list2) -> List.of(list1, list2)));
System.out.println(result);
```



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()),
    [Michael, Mike]
    filtering(endsWithM, toList()),
    [Tim, Tom]
    (list1, list2) -> List.of(list1, list2));
System.out.println(result);
```



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()),  
                           [Michael, Mike])  
                         .filtering(endsWithM, toList()),  
                         [Tim, Tom])  
                         .toList((list1, list2) -> List.of(list1, list2));  
  
System.out.println(result);  
[[Michael, Mike], [Tim, Tom]]
```



JMH



Benchmarking Intro



- 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



- **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
-

Simple Start Stop measurements



- 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



- Repeated Start-/Stop measurements using more precise `System.nanoTime()`

```
// 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



- **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



- 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



- A **MyBenchmark** class is generated as the skeleton:

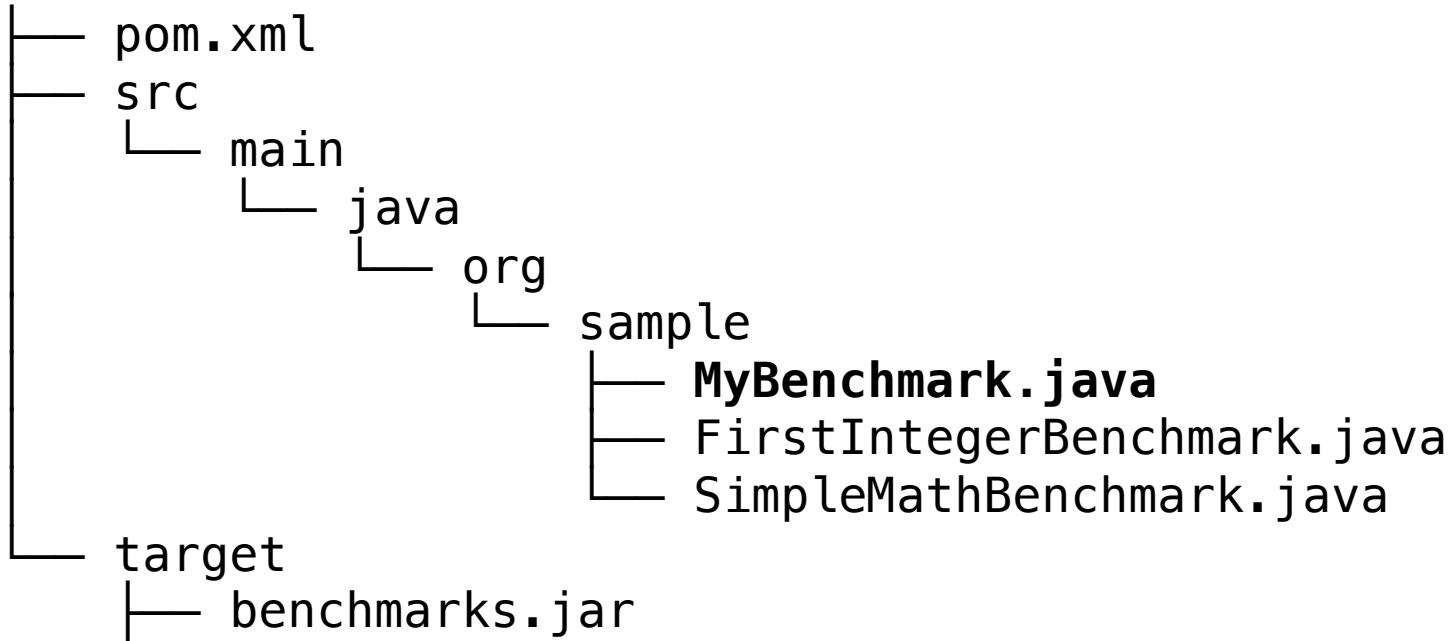
```
public class MyBenchmark
{
    @Benchmark
    public void testMethod()
    {
        // This is a demo/sample template for building your JMH benchmarks. Edit
        // as needed.
        // Put your benchmark code here.
    }
}
```

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

Microbenchmarks with JMH



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



- In 2 steps to a benchmark
 - 1) mvn clean package
 - 2) java -jar target/benchmarks.jar

Own Microbenchmark with JMH



```
@Measurement(iterations = 3, time = 1000, timeUnit = TimeUnit.MILLISECONDS)
@Warmup(iterations = 7, time = 1000, timeUnit = TimeUnit.MICROSECONDS)
@Fork(4)
public class FirstIntegerBenchmark
{
    @Benchmark
    public String numberAsHexString()
    {
        int dec = 123456789;
        return Integer.toHexString(dec);
    }

    @Benchmark
    public String numberAsBinaryString()
    {
        int dec = 123456789;
        return Integer.toBinaryString(dec);
    }
}
```

Own Microbenchmark with JMH



```
// Based on https://www.retit.de/continuous-benchmarking-with-jmh-and-junit-2/
public class SearchBenchmark {
    @State(Scope.Thread)
    public static class SearchState {
        public String text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ__abcdefghijklmnopqrstuvwxyz";
    }

    @Benchmark
    public int testIndex0f(SearchState state) {
        return state.text.indexOf("M");
    }

    @Benchmark
    public int testIndex0fChar(SearchState state) {
        return state.text.indexOf('M');
    }

    @Benchmark
    public boolean testContains(SearchState state) {
        return state.text.contains("M");
    }
}
```

Own Microbenchmark with JMH



```
@BenchmarkMode(Mode.AverageTime)
@Fork(2)
@State(Scope.Benchmark)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class SimpleStringJoinBenchmark
{
    private String from = "Michael";
    private String to = "Participants";
    private String subject = "Benchmarking with JMH";

    @Benchmark
    public String stringPlus(Blackhole blackhole)
    {
        String result = "From: " + from + "\nTo: " + to + "\nSubject: " + subject;
        blackhole.consume(result);
        return result;
    }
}
```

Inspired by <http://alblue.bandlem.com/2016/04/jmh-stringbuffer-stringbuilder.html>,
but now using BlackHole and slightly modified

Own Microbenchmark with JMH



```
@Benchmark
public String stringPlusEqual(Blackhole blackhole)
{
    String result = "From: " + from;
    result += "\nTo: " + to;
    result += "\nSubject: " + subject;

    blackhole.consume(result);
    return result;
}

@Benchmark
public String builderAppendChained(Blackhole blackhole)
{
    String result = new StringBuilder().append("From: ").append(from).
                                         append("\nTo: ").append(to).
                                         append("\nSubject: ").append(subject).
                                         toString();

    blackhole.consume(result);
    return result;
}
```

ATTENTION – Own Microbenchmarks mit JMH



```
@State(Scope.Benchmark)
public static class MyBenchmarkState {
    @Param({ "10000", "100000" })
    public int value;
}

@Benchmark
public String stringPlusABC(MyBenchmarkState state, Blackhole blackhole) {
    String result = "";
    for (int i = 0; i < state.value; i++) {
        result += "ABC";
    }

    blackhole.consume(result);
    return result;
}
```

ATTENTION – Own Microbenchmarks mit JMH



```
@Benchmark
public String stringConcatABC(MyBenchmarkState state, Blackhole blackhole) {
    String result = "";
    for (int i = 0; i < state.value; i++) {
        result = result.concat("ABC");
    }
    blackhole.consume(result);
    return result;
}
```

```
@Benchmark
public String concatUsingStringBuilder(MyBenchmarkState state, Blackhole blackhole) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < state.value; i++) {
        sb.append("ABC");
    }
    String result = sb.toString();
    blackhole.consume(result);
    return result;
}
```

Microbenchmarks With JMH, adjust to support Java 17



- POM adjustment for java 17:

```
<!-- Java source/target to use for compilation. -->
    <javac.target>1.8</javac.target>
=>
<javac.target>17</javac.target>

<jmh.version>1.33</jmh.version>

<groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
=>
<version>3.8.1</version>
```



What do we want to measure?

- `indexOf(String)`, `indexOf(char)`, `contains(String)`
 - `for`, `forEach`, `while`, `Iterator`
 - `String +=`, `String.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
SimpleStringJoinBenchmark.builderAppendChained	avgt	10	46.308	± 7.950	ns/op
SimpleStringJoinBenchmark.builderMultipleAppend	avgt	10	127.312	± 14.935	ns/op
SimpleStringJoinBenchmark.stringConcat	avgt	10	83.888	± 18.979	ns/op
SimpleStringJoinBenchmark.stringPlus	avgt	10	38.871	± 2.823	ns/op
SimpleStringJoinBenchmark.stringPlusEqual	avgt	10	39.346	± 3.015	ns/op



Nashorn Java Script Engine

(deprecated since Java 11 deprecated, removed with Java 15)





Java 16



Stream => List ... it was so awkward ...



```
List<String> namesMi = Stream.of("Tim", "Tom", "Mike", "Michael").  
    filter(str -> str.startsWith("Mi")).  
collect(Collectors.toList());
```

FINALLY... `toList()`



```
List<String> namesMi = Stream.of("Tim", "Tom", "Mike", "Michael").  
                           filter(str -> str.startsWith("Mi")).  
                           toList();
```

FINALLY... `toList()` ... but better not look into the sources!



```
@SuppressWarnings("unchecked")
default List<T> toList() {
    return (List<T>) Collections.unmodifiableList(
        new ArrayList<>(Arrays.asList(this.toArray())));
}
```



Exercises PART 4 & 5

<https://github.com/Michaeli71/JAX-London-Best-of-Java9-17>



PART 6:

What's new in Java 17

- Overview: What's included?
 - Build tools and IDEs
 - Syntax extension Sealed Types / Classes (FINAL)
 - Tooling jpackage (FINAL)
 - Syntax extensions for switch (PREVIEW)
-

Java 17 – What's included?



- JEP 306: [Restore Always-Strict Floating-Point Semantics](#)
- JEP 356: [Enhanced Pseudo-Random Number Generators](#)
- JEP 382: [New macOS Rendering Pipeline](#)
- JEP 391: [macOS/AArch64 Port](#)
- JEP 398: [Deprecate the Applet API for Removal](#)
- JEP 403: [Strongly Encapsulate JDK Internals](#)
- JEP 406: [Pattern Matching for switch \(Preview\)](#)
- JEP 407: [Remove RMI Activation](#)
- JEP 409: [Sealed Classes](#)
- JEP 410: [Remove the Experimental AOT and JIT Compiler](#)
- JEP 411: [Deprecate the Security Manager for Removal](#)
- JEP 412: [Foreign Function & Memory API \(Incubator\)](#)
- JEP 414: [Vector API \(Second Incubator\)](#)
- JEP 415: [Context-Specific Deserialization Filters](#)

Was ist davon wirklich wichtig für uns?

- JEP 406: [Pattern Matching for switch \(Preview\)](#)
 - ?? JEP 409: [Sealed Classes](#) ??
 - ?? JEP 391: [macOS/AArch64 Port](#) ??
-



[Feedback on this page?](#)

The Java Version Almanac

Collection of information about the history and future of Java.

Details	Status	Documentation	Download	Compare API to
Java 18	DEV	API Notes	JDK JRE	17 16 15 14 13 12 11 10 9 8 ...
Java 17	LTS	API Lang VM Notes	JDK JRE	16 15 14 13 12 11 10 9 8 7 ...
Java 16	EOL	API Lang VM Notes	JDK JRE	15 14 13 12 11 10 9 8 7 6 ...
Java 15	EOL	API Lang VM Notes	JDK JRE	14 13 12 11 10 9 8 7 6 5 ...
Java 14	EOL	API Lang VM Notes	JDK JRE	13 12 11 10 9 8 7 6 5 1.4 ...
Java 13	EOL	API Lang VM Notes	JDK JRE	12 11 10 9 8 7 6 5 1.4 1.3 ...
Java 12	EOL	API Lang VM Notes	JDK JRE	11 10 9 8 7 6 5 1.4 1.3 1.2 ...
Java 11	LTS	API Lang VM Notes	JDK JRE	10 9 8 7 6 5 1.4 1.3 1.2 1.1
Java 10	EOL	API Lang VM Notes	JDK JRE	9 8 7 6 5 1.4 1.3 1.2 1.1
Java 9	EOL	API Lang VM Notes	JDK JRE	8 7 6 5 1.4 1.3 1.2 1.1
Java 8	LTS	API Lang VM Notes	JDK JRE	7 6 5 1.4 1.3 1.2 1.1
Java 7	EOL	API Lang VM Notes	JDK JRE	6 5 1.4 1.3 1.2 1.1
Java 6	EOL	API Lang VM Notes	JDK JRE	5 1.4 1.3 1.2 1.1
Java 5	EOL	API Lang VM Notes		1.4 1.3 1.2 1.1
Java 1.4	EOL	API		1.3 1.2 1.1
Java 1.3	EOL	API		1.2 1.1
Java 1.2	EOL	API Lang		1.1
Java 1.1	EOL	API		
Java 1.0	EOL	API Lang VM		



Build-Tools and IDEs



IDE & Tool Support for Java 17



- Current IDEs & tools basically good
- Eclipse: Version 2021-09 with additional plug in
- IntelliJ: Version 2021.2.1
<https://blog.jetbrains.com/idea/2021/09/java-17-and-intellij-idea/>
- Maven: 3.8.2, Compiler-Plugin: 3.8.1
- Gradle: 7.2
- Activation of preview features required
 - In dialogs
 - In the build script



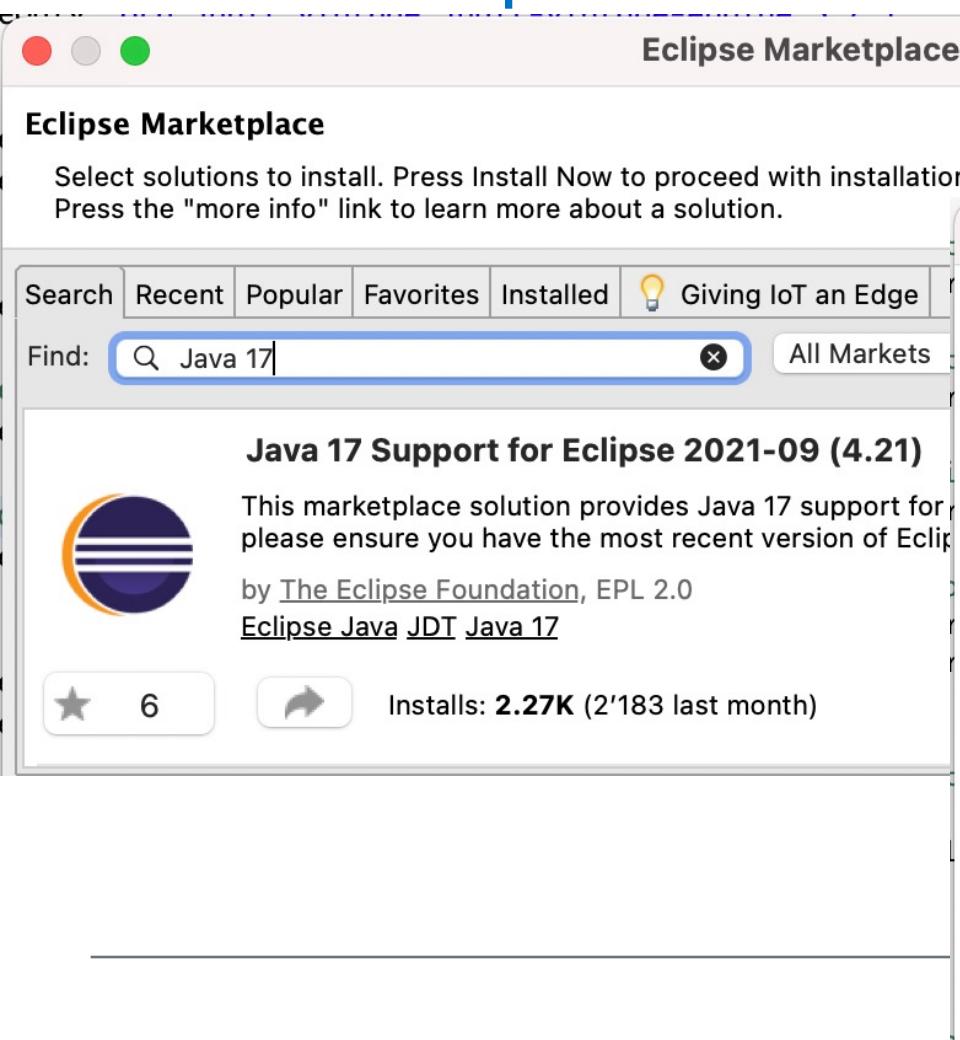
Maven™



IDE & Tool Support Java 17



- Eclipse 2021-09 with Plugin
- Activation of preview features required



Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

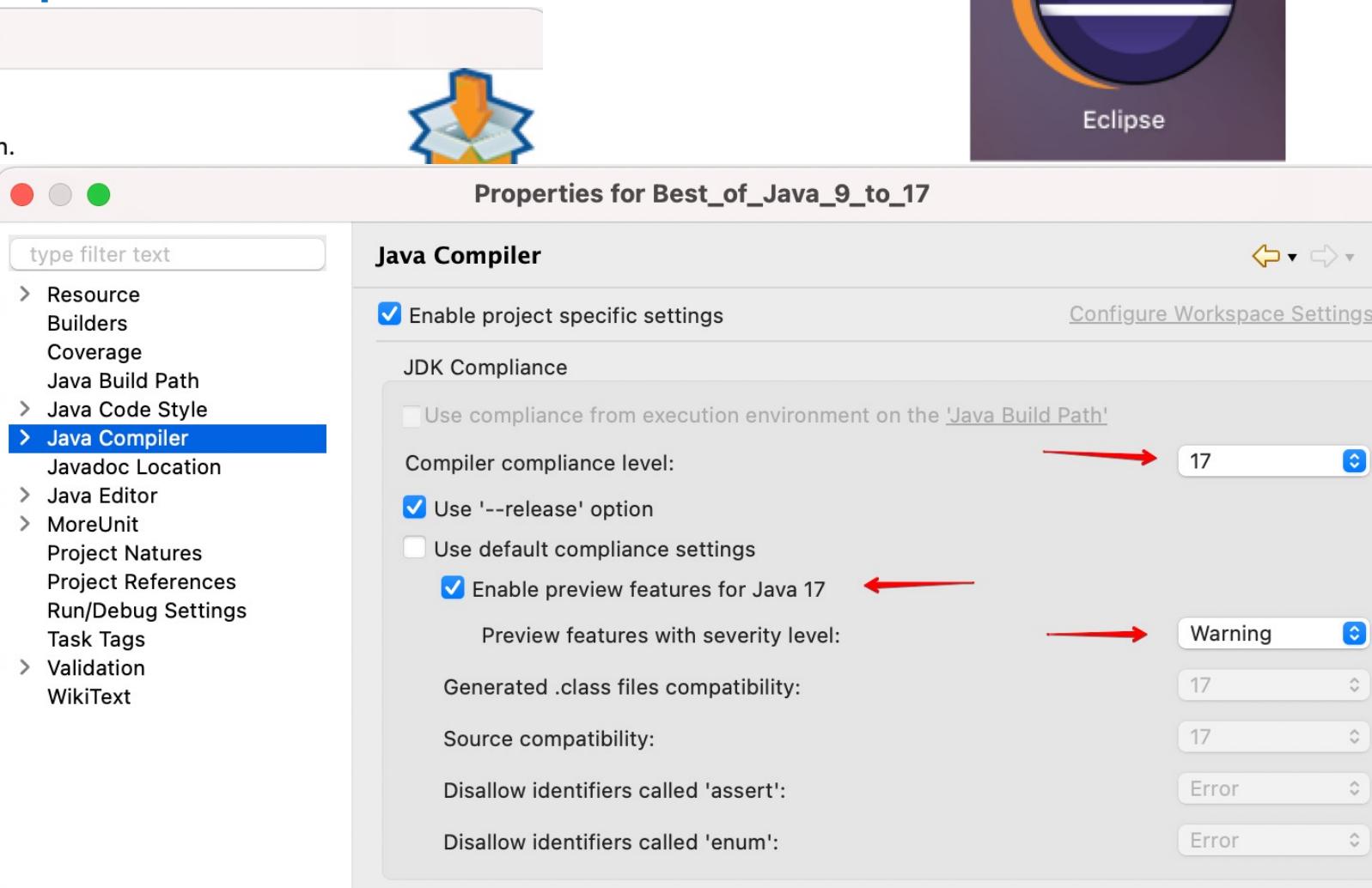
Search Recent Popular Favorites Installed Giving IoT an Edge

Find: All Markets

Java 17 Support for Eclipse 2021-09 (4.21)

This marketplace solution provides Java 17 support for please ensure you have the most recent version of Eclipse by [The Eclipse Foundation](#), EPL 2.0 [Eclipse Java JDT Java 17](#)

6  Installs: 2.27K (2'183 last month)



Eclipse Marketplace

Properties for Best_of_Java_9_to_17

Java Compiler

Enable project specific settings [Configure Workspace Settings](#)

JDK Compliance

Use compliance from execution environment on the 'Java Build Path'

Compiler compliance level:  17

Use '--release' option

Use default compliance settings

Enable preview features for Java 17 

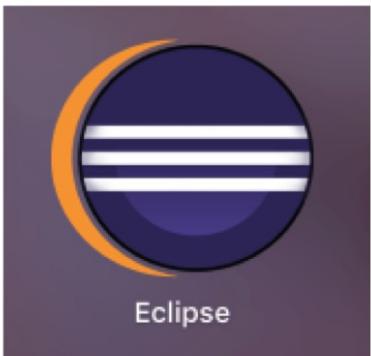
Preview features with severity level:  Warning 

Generated .class files compatibility: 17

Source compatibility: 17

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error



IDE & Tool Support



- Activation of preview features required

Project Structure

Project name: Java17Examples

Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.
17 version 17 Edit

Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.
SDK default (17 - Sealed types, always-strict floating-point semantics)

Project compiler output:
This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.
/Users/michaeli/Java17Examples/out

Project language level:

This language level is default for all project modules.

A module specific language level can be configured for each of the modules as required.



IDE & Tool Support Java 17



- Activation of preview features required

```
sourceCompatibility=17  
targetCompatibility=17
```

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



IDE & Tool Support



- Activation of preview features required

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
      <source>17</source>
      <target>17</target>
      <!-- Wichtig für Java Syntax-Neuerungen -->
      <compilerArgs>--enable-preview</compilerArgs>
    </configuration>
  </plugin>
</plugins>
```





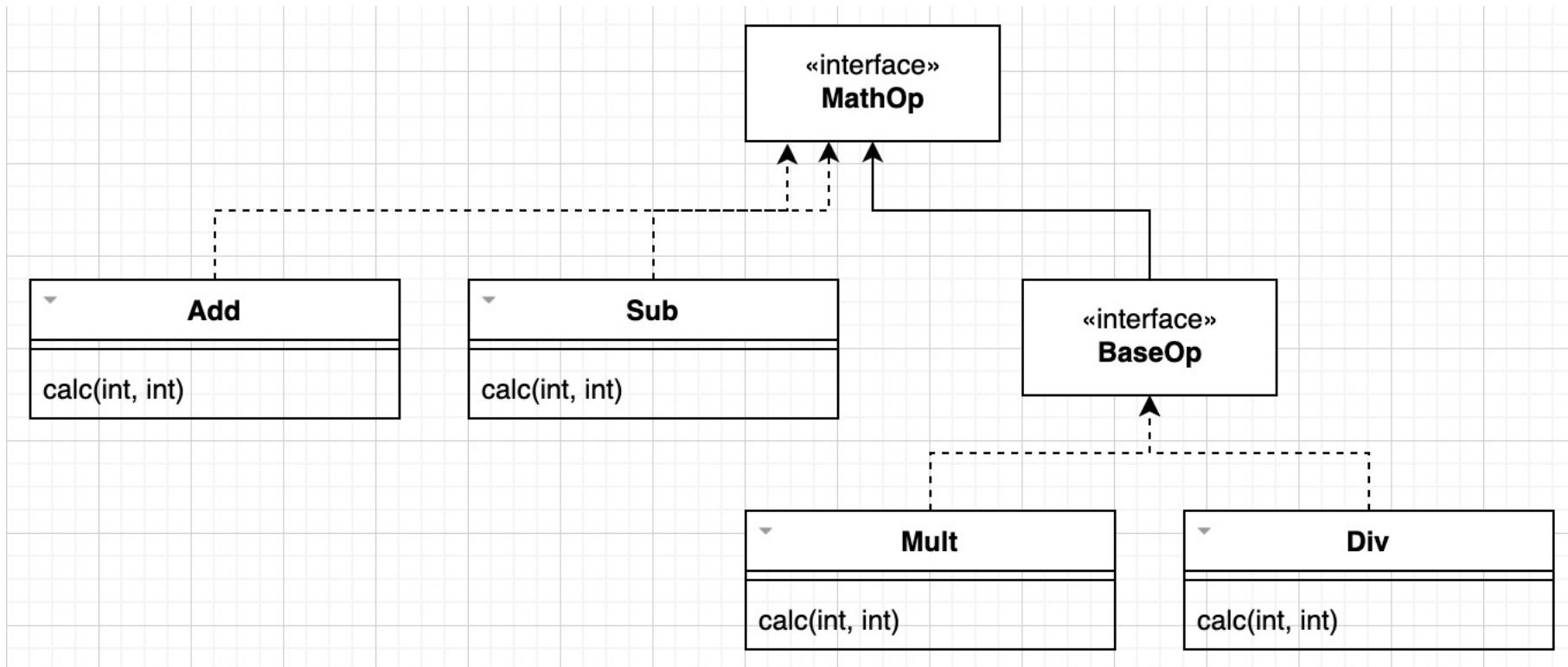
Sealed Types



Sealed Types



- Control of inheritance and specify which classes can extend a base class



Sealed Types



- **Control of inheritance** and **specify** which classes can extend a base class

```
public class SealedTypesExamples
```

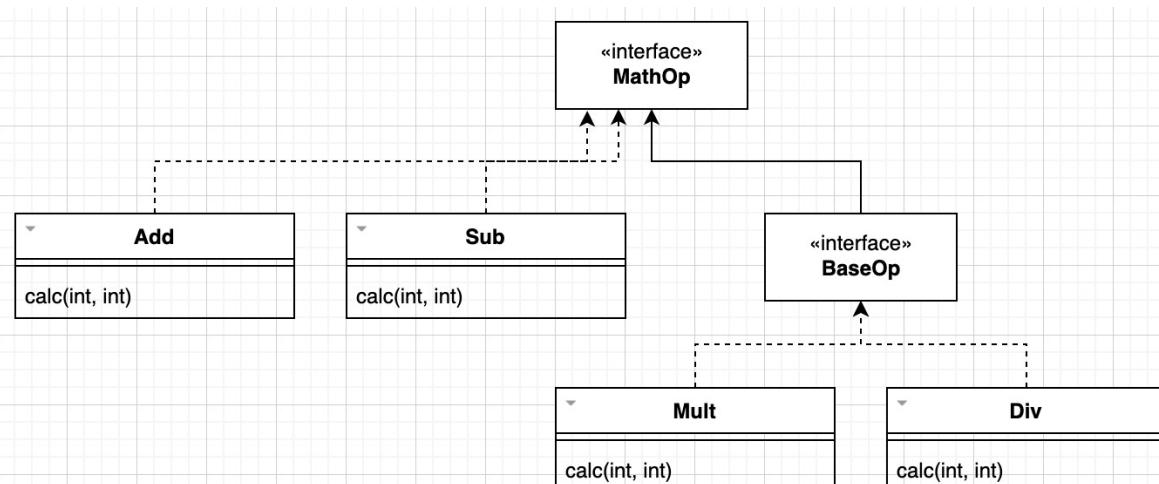
```
{  
    // A sealed class must have subclasses  
    sealed interface MathOp  
        permits BaseOp, Add, Sub // <= permitted sub types  
    {  
        int calc(int x, int y);  
    }  
}
```

// non-sealed allows to provide base class(es)

```
non-sealed class BaseOp implements MathOp // <= base class not sealed
```

```
{  
    @Override  
    public int calc(int x, int y)  
    {  
        return 0;  
    }  
}  
...
```

sealed allows to seal an inheritance hierarchy and to accept only explicitly noted types. These have to be sealed, non-sealed or final.

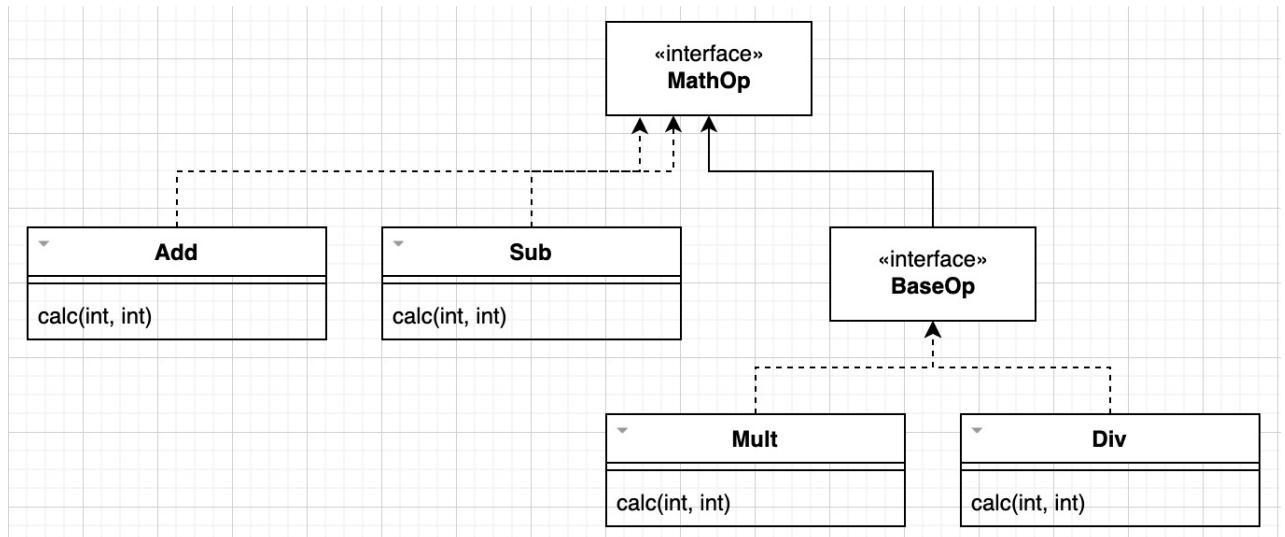


Sealed Types



```
..  
// direct implementation must be final  
final class Add implements MathOp  
{  
    @Override  
    public int calc(int x, int y)  
    {  
        return x + y;  
    }  
}  
  
final class Sub implements MathOp  
{  
    ...  
}  
// derived from base class must be final  
final class Mult extends BaseOp  
{  
}  
  
final class Div extends BaseOp  
{  
}
```

- A sealed class must have subclasses which in turn are listed after permits
- A non-sealed class can act as base class and subtypes can be derived
- A final class is – as usual – an endpoint in a class hierarchy



Knowledge of Sealed Types



- **define which other classes or interfaces may form subtypes**
 - **Sealed Types can expose behavior via interfaces when developing libraries, but keep control over possible implementations**
 - **Sealed Types restrict the extensibility of class hierarchies and should be used with care. Flexibility not foreseen during implementation can be disturbing later. (e.g. Unit Testing and Extract and Override)**
-



JPackage





▼ PackagingDemo

► JRE System Library [JavaSE-16]

▼ src/main/java

 ▼ de.java17

 ▼ ApplicationExample.java

 ▼ ApplicationExample

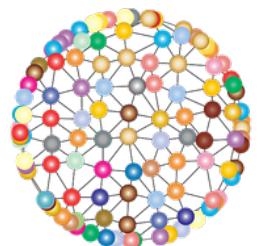
 main(String[]) : void

► src

 build.gradle

 pom.xml

```
public class ApplicationExample {  
    public static void main(String[] args) {  
        System.out.println("First JPackage");  
  
        JOptionPane.showConfirmDialog(null, "Generated by jpackage", "DEMO",  
            JOptionPane.OK_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE, null)  
    }  
}
```



How to build?



```
▼-PackagingDemo
  ► JRE System Library [JavaSE-16]
  ▼-src/main/java
    ▼-de.java17
      ▼-ApplicationExample.java
        ApplicationExample
          main(String[]) : void
  ► src
  🐘 build.gradle
  📄 pom.xml
```

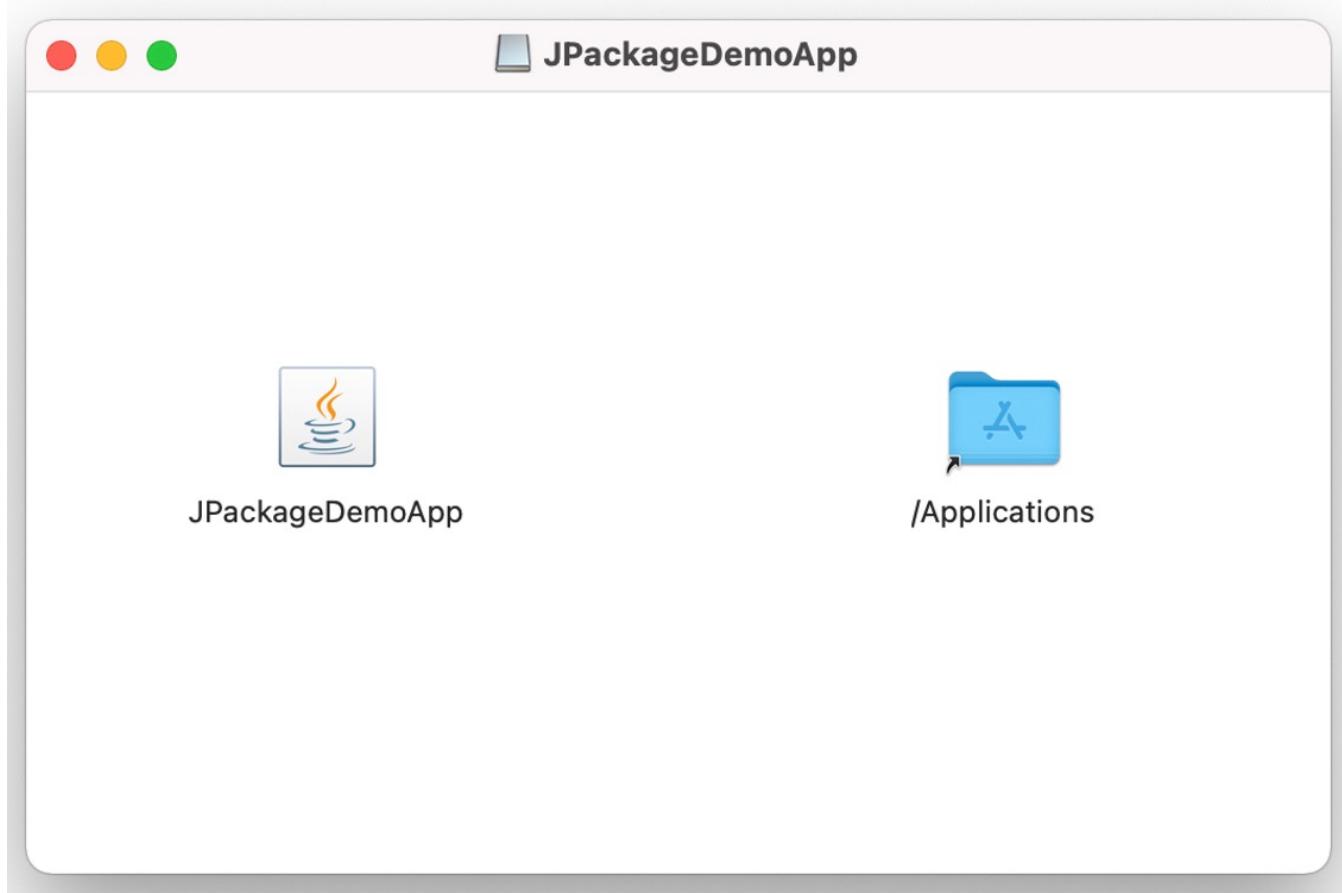
mvn clean install

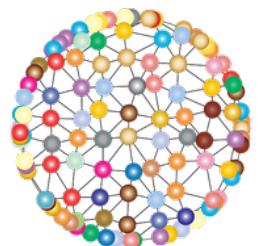
```
✓ target
  > classes
  > generated-sources
  > maven-archiver
  > maven-status
  > PackagingDemoExamples-1.0.0-SNAPSHOT.jar
  ...
```





```
jpackage --input target/ --name JPackageDemoApp --main-jar PackagingDemoExamples-1.0.0-SNAPSHOT.jar --main-class de.java17.ApplicationExample --type dmg --java-options '--enable-preview'
```





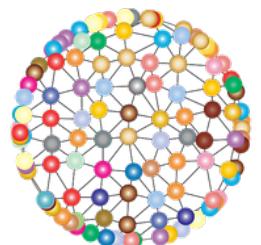
What about 3rd party libraries?



```
public class ApplicationExample {  
  
    public static void main(String[] args) {  
        System.out.println("First JPackage");  
  
        Joiner joiner = Joiner.on(":");  
        String result = joiner.join(List.of("Michael", "mag", "Python"));  
        System.out.println(result);  
        JOptionPane.showConfirmDialog(null, result);  
    }  
}  
  
<!-- https://mvnrepository.com/artifact/com.google.guava/guava -->  
<dependency>  
  <groupId>com.google.guava</groupId>  
  <artifactId>guava</artifactId>  
  <version>31.0.1-jre</version>  
</dependency>
```



How to include the lib in the Application?





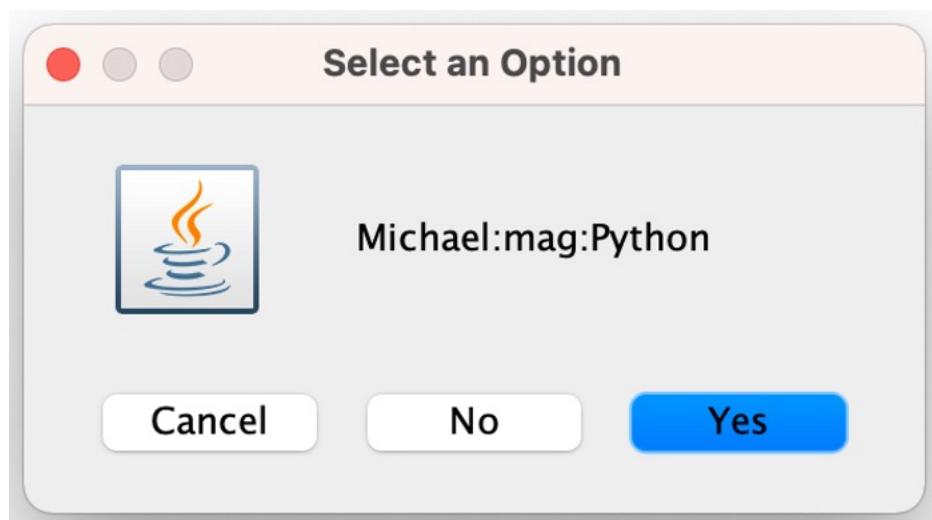
```
<plugin>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <phase>process-sources</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>

      <configuration>
        <outputDirectory>target</outputDirectory>
        <includeScope>runtime</includeScope>
        <excludeScope>test</excludeScope>
      </configuration>
    </execution>
  </executions>
</plugin>
```



✓ target

- > classes
- > generated-sources
- > maven-archiver
- > maven-status
 - ⌚ checker-qual-3.12.0.jar
 - ⌚ error_prone_annotations-2.7.1.jar
 - ⌚ failureaccess-1.0.1.jar
 - ⌚ guava-31.0.1-jre.jar
 - ⌚ j2objc-annotations-1.3.jar
 - ⌚ jsr305-3.0.2.jar
 - ⌚ listenablefuture-9999.0-empty-to-avoid-conflict-with-guav
 - ⌚ PackagingDemoExamples-1.0.0-SNAPSHOT.jar





DEMO & Hands on



Pattern Matching bei switch (PREVIEW)



Pattern Matching for switch



- With Java 16, pattern matching was introduced for instanceof. With this it is possible to accept a so-called type pattern and to perform a pattern matching. This saves annoying casts.
- This syntax innovation is also available for switch with Java 17.
- However, the whole thing is initially implemented in the form of preview features and to verify them you have to activate them appropriately, for example as follows in the JShell:

```
michaelinden@Air-von-Michael ~ % jshell --enable-preview
| Welcome to JShell -- Version 17
| For an introduction type: /help intro
```

Pattern Matching for switch



- Suppose we needed to write a generic method for formatting values. Provided we use Java 16, we can write this reasonably readably using pattern matching and instanceof as follows -- the whole thing would be much worse without Java 16, since we would have to add a line of cast for each type:

```
static String formatterJdk16instanceof(Object obj) {  
    String formatted = "unknown";  
    if (obj instanceof Integer i) {  
        formatted = String.format("int %d", i);  
    } else if (obj instanceof Long l) {  
        formatted = String.format("long %d", l);  
    } else if (obj instanceof Double d) {  
        formatted = String.format("double %f", d);  
    } else if (obj instanceof String s) {  
        formatted = String.format("String %s", s);  
    }  
    return formatted;  
}
```

Pattern Matching for switch



- Until now it was not possible to handle the value null in the cases of a switch, instead the following special handling was necessary:

```
static void switchSpecialNullSupport(String str) {  
    if (str == null) {  
        System.out.println("special handling for null");  
        return;  
    }  
  
    switch (str) {  
        case "Java", "Python" -> System.out.println("cool language");  
        default -> System.out.println("everything else");  
    }  
}
```

Pattern Matching for switch



- With Java 17 and preview features enabled, we can now specify null-Werte values and unify the whole construct:

```
static void switchSupportingNull(String str) {  
    switch (str) {  
        case null -> System.out.println("null is allowed in preview"); ←  
        case "Java", "Python" -> System.out.println("cool language");  
        default -> System.out.println("everything else");  
    }  
}
```

Pattern Matching for switch



- Analogous to instanceof, queries like the following are now also possible in the cases:

```
static void processData(Object obj) {  
    switch (obj) {  
        case String str && str.startsWith("V1") -> System.out.println("Processing V1");  
        case String str && str.startsWith("V2") -> System.out.println("Processing V2");  
        case Integer i && i > 10 && i < 100 -> System.out.println("Processing ints");  
        default -> throw new IllegalArgumentException("invalid input");  
    }  
}
```





Conclusion

Positive things



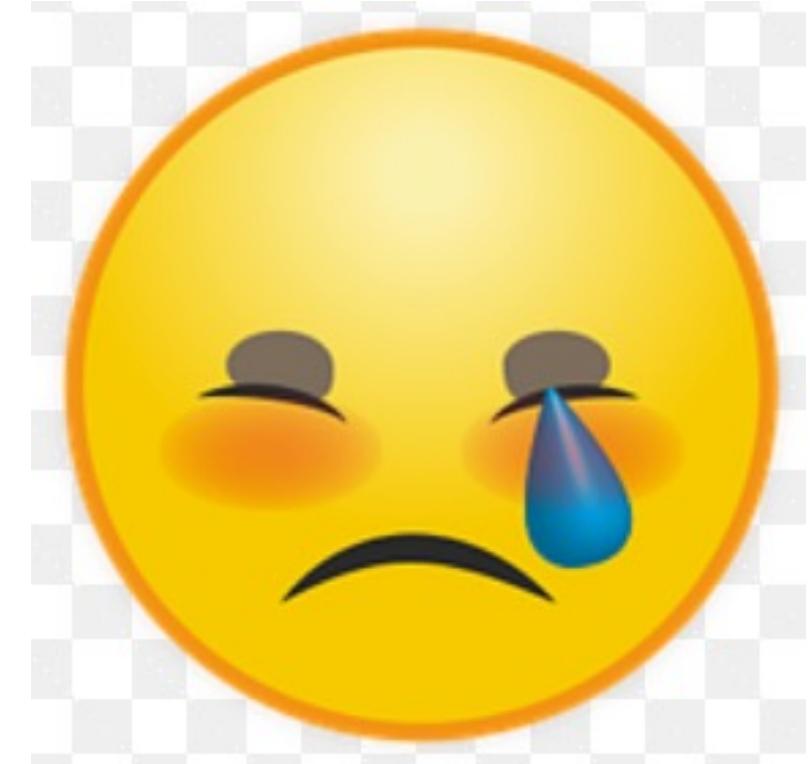
- Some parts of project coin (syntax)
- Switch / Records
- A lot of practical extensions in the APIs
- HTTP 2
- Reactive Streams
- Modularization with Project JIGSAW -- but until now a lot of room for improvement concerning tools

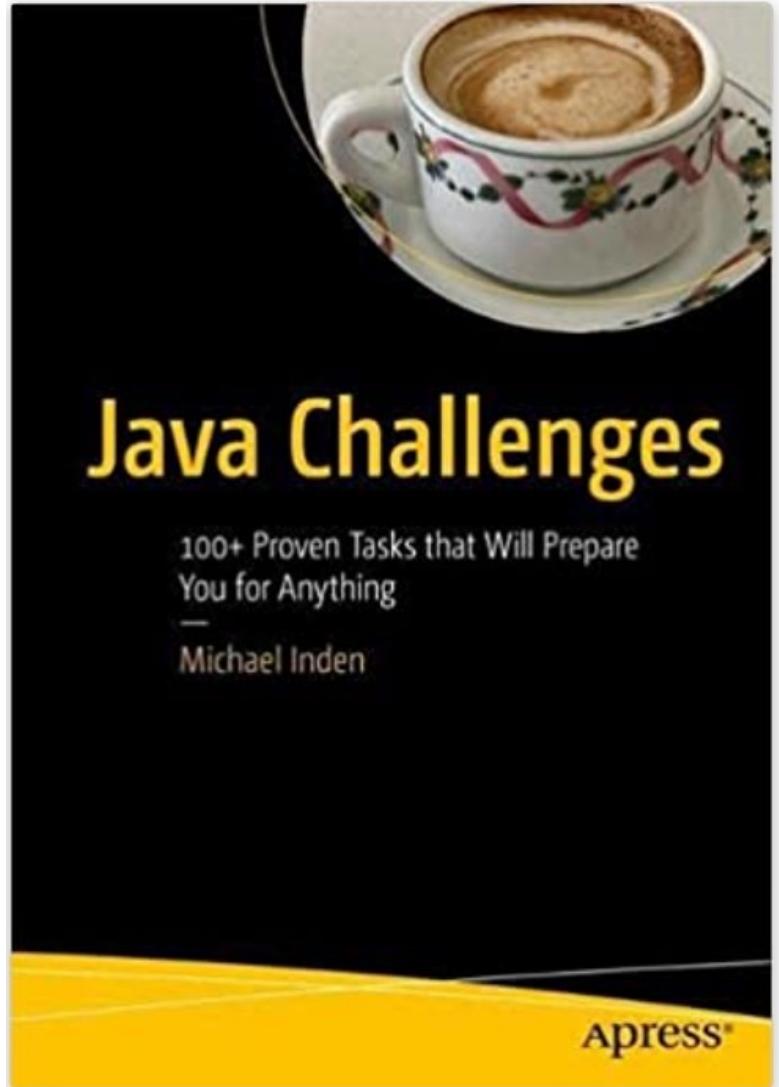


Negative things



- Multiple delay of Java 9
Sept. 2016 => March 2017 => July 2017 => Sept. 2017
- Next releases were on time, but sometimes bringing just a few new features (except Java 14)
- Often less than planned
 - no versioning with JIGSAW
 - no JSON support
 - instead of collection literals only convenience methods
 - Instead of ZIP only TEE (ing) collector







Questions?



Thank You