



Best of Java 11 – 19

<https://github.com/Michaeli71/JAX-London-Best-of-Java-11-19>

Michael Inden

Independent SW consultant and author

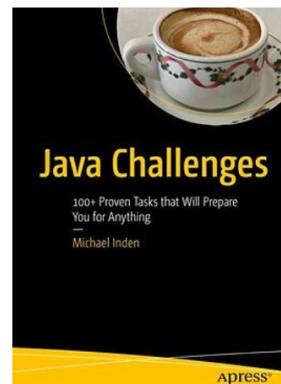
Speaker Intro



- **Michael Inden, Year of Birth 1971**
- **Diploma Computer Science, C.v.O. Uni Oldenburg**
- **~8 ¼ Years SSE at Heidelberger Druckmaschinen AG in Kiel**
- **~6 ¾ Years TPL, SA at IVU Traffic Technologies AG in Aachen**
- **~4 ¼ Years LSA / Trainer at Zühlke Engineering AG in Zurich**
- **~3 Years TL / CTO at Direct Mail Informatics / ASMIQ in Zurich**
- **Independent Consultant, Conference Speaker and Trainer**
- **Since January 2022 Head of Development at Adcubum in Zurich**
- **Author @ dpunkt.verlag and APress**

E-Mail: michael.inden@hotmail.com

Blog: <https://jaxenter.de/author/minden>





Agenda

Workshop Contents



- **Preliminary notes / Build Tools & IDEs**
- **PART 1:** Syntax Enhancements & News and API-Changes up to Java 11
- **PART 2:** Additional News and Changes up to Java 11

- **PART 3:** Syntax Enhancements in Java 12 to 17
- **PART 4:** News and API-Changes in Java 12 to 17
- **PART 5:** News in the JVM in Java 12 to 17
- **PART 6:** News in Java 18
- **PART 7:** News in Java 19

- **Separate:** Java Modularization

Workshop Contents



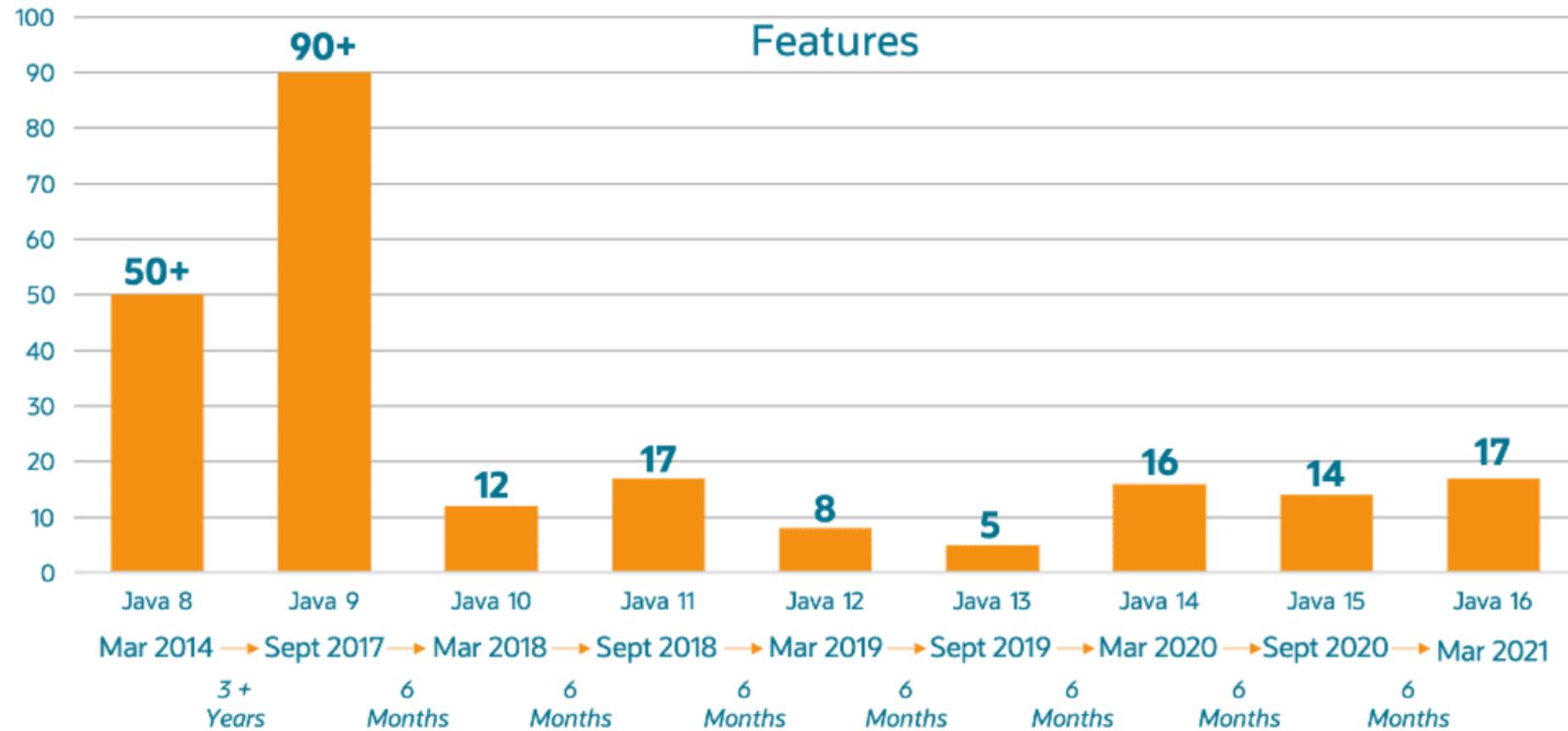
JDK	Release Date	Development Time	LTS	Workshop
Oracle JDK 8	3 / 2014	-	yes, <i>in the mean time commercial</i>	-/-
Oracle JDK 9	9 / 2017	3,5 year	-	Part 1, 2
Oracle JDK 10	3 / 2018	6 months	-	Part 1, 2
Oracle JDK 11	9 / 2018	6 months	yes, <i>commercial</i>	Part 1, 2
Oracle JDK 12	3 / 2019	6 months	-	Part 3, 4, 5
Oracle JDK 13	9 / 2019	6 months	-	Part 3, 4, 5
Oracle JDK 14	3 / 2020	6 months	-	Part 3, 4, 5
Oracle JDK 15	9 / 2020	6 months	-	Part 3, 4, 5
Oracle JDK 16	3 / 2021	6 months	-	Part 3, 4, 5
Oracle JDK 17	9 / 2021	6 months	yes, <i>commercial, but a lot laxer</i>	Part 3, 4, 5
Oracle JDK 18	3 / 2022	6 month	-	Part 6
Oracle JDK 19	9 / 2022	6 month	-	Part 7

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)
 - after Java 17 switching to a two years Its cadence => 2023 Java 21 LTS
- **other versions are "only" intermediate versions**
 - get updates only within 6 months
 - „Previews“ – features that are not finalized, integrated to get feedback
 - Ideal to get to know new features and to experiment (especially for private projects)

Classification 6 month release cycle





Build-Tools and IDEs



IDE & Tool Support for Java 17 (Java 18 / 19 is covered later)



- Current IDEs & Tools normally doing a good job



- Eclipse: Version 2021-09

- IntelliJ: Version 2021.2.1

<https://blog.jetbrains.com/idea/2021/09/java-17-and-intellij-idea/>

- Maven: 3.8.5, Compiler-Plugin: 3.8.1



- Gradle: 7.4.1



- Activation of preview features required

- In dialogs
- In the build script

IDE & Tool Support Java 17



- Eclipse 2021-09 with Plugin // since Eclipse 2021-12 included
- Activation of Preview Features necessary

Eclipse Marketplace

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

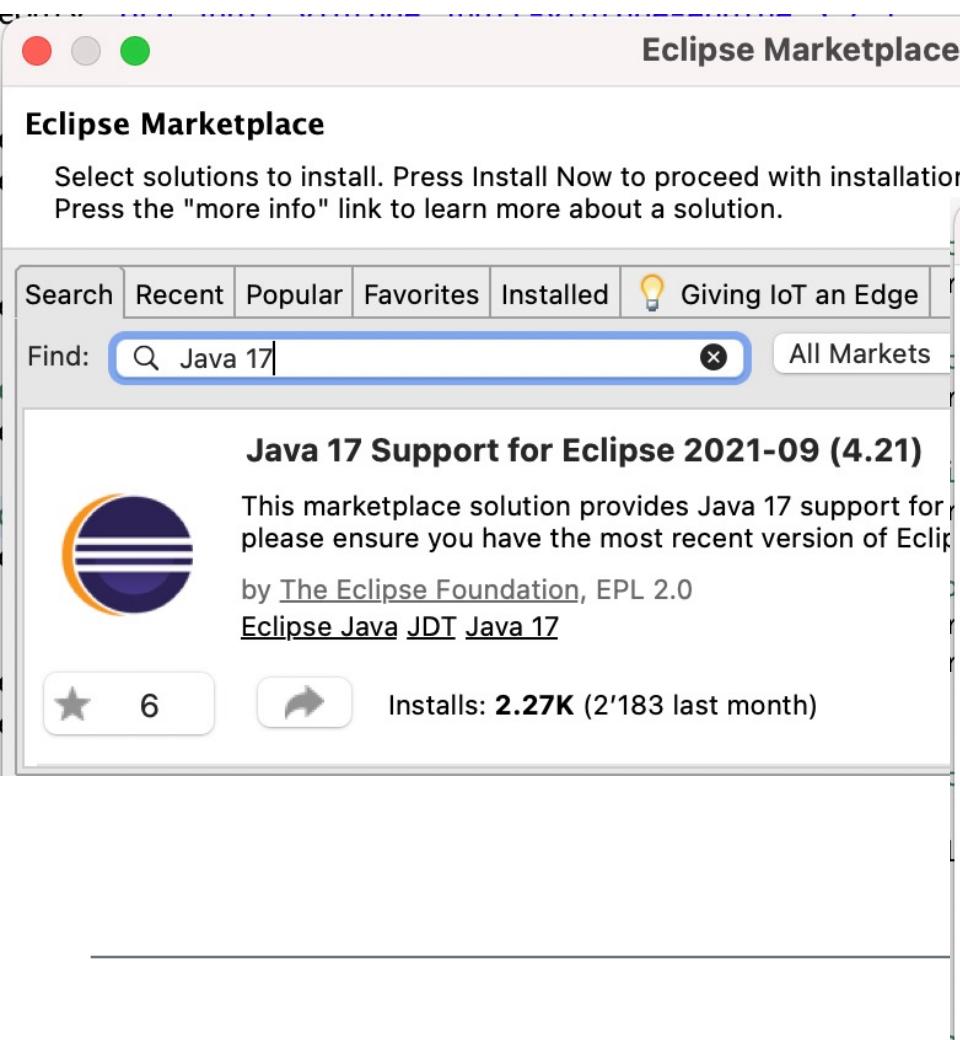
Search Recent Popular Favorites Installed Giving IoT an Edge

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 

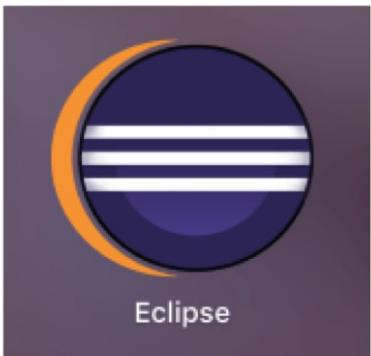
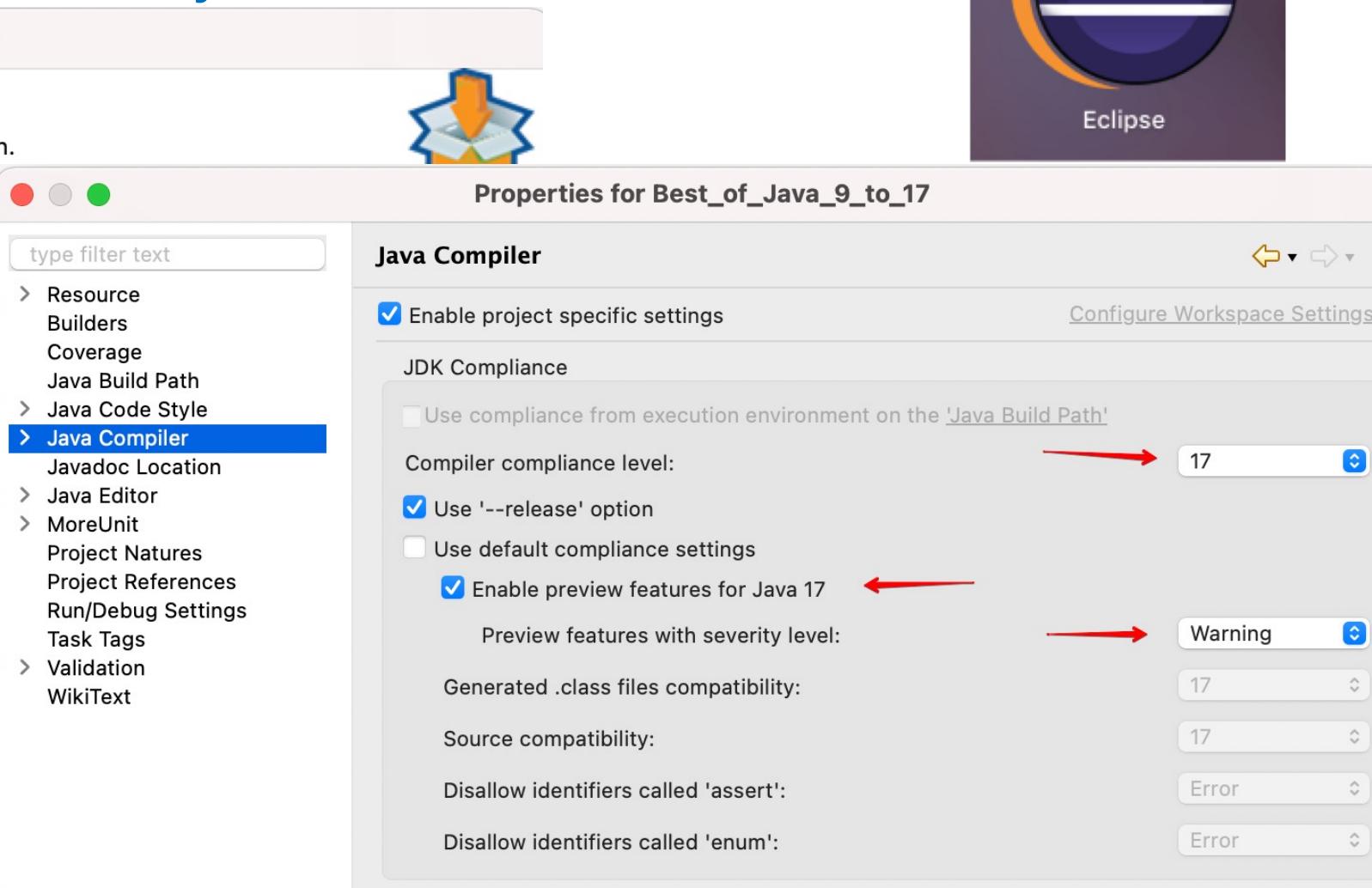
Preview features with severity level:  Warning

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.





- Activation of preview features required

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

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



```
sourceCompatibility=18  
targetCompatibility=18
```

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

IDE & Tool Support for Java 17 / 18



- 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>
      <!-- Important for some syntax news -->
      <compilerArgs>--enable-preview</compilerArgs>
    </configuration>
  </plugin>
</plugins>

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
      <source>18</source>
      <target>18</target>
      <!-- Wichtig für Java Syntax-Neuerungen -->
      <compilerArgs>--enable-preview</compilerArgs>
    </configuration>
  </plugin>
</plugins>
```





PART 1: **Syntax Enhancements & News and API-Changes up to Java 11**



Syntax Enhancements



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()  
{  
    // ...  
}
```

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.

🚫 `var justDeclaration; // no value => no definition`
`var numbers = {0, 1, 2}; // missing type`

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.

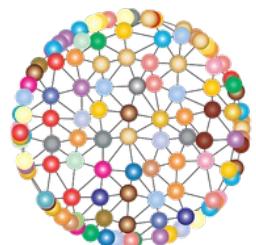
```
// var => Set<Map.Entry<String, Long>>
var entries = personAgeMapping.entrySet();
```

```
// 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

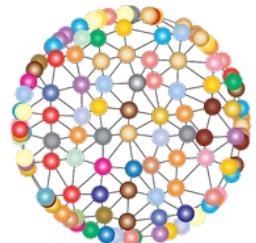


- Sometimes you may be tempted to just change the type with var:

```
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);
```



News and API-Changes up to Java 11

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



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(...)`

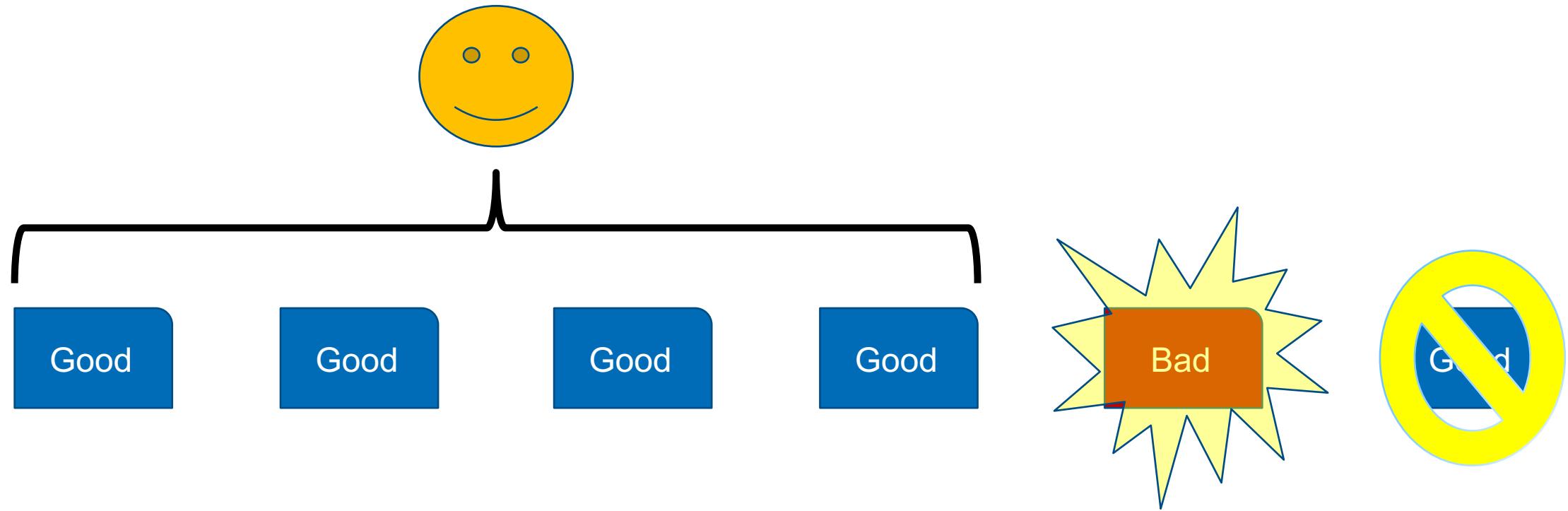
`takeWhile(Predicate<T>)` – processes elements as long
as the condition is met

`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?

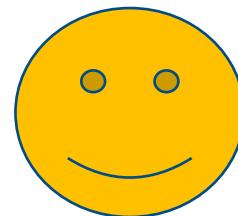


Google

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
```



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
- **flatMapping()** – Mapping and merging elements

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

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

Stream API Collectors in JDK 9



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 first. Advantage only with `groupingBy()`

Stream API Collectors in JDK 9



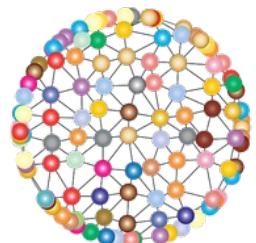
Example for the entry:

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}
```



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>



- 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()



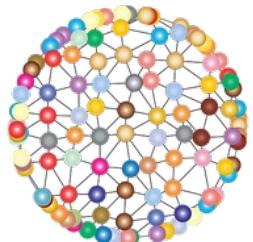
- `Optional<T> => Stream<T>`: This is useful if you have a stream of optional values and want to [keep only those entries with valid values](#) (combination of methods `flatMap()` and `stream()`).
- Example of a stream consisting of `Optional<String>` elements, for example as a result of a [parallel 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);  
    }  
  
}
```



**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)
```



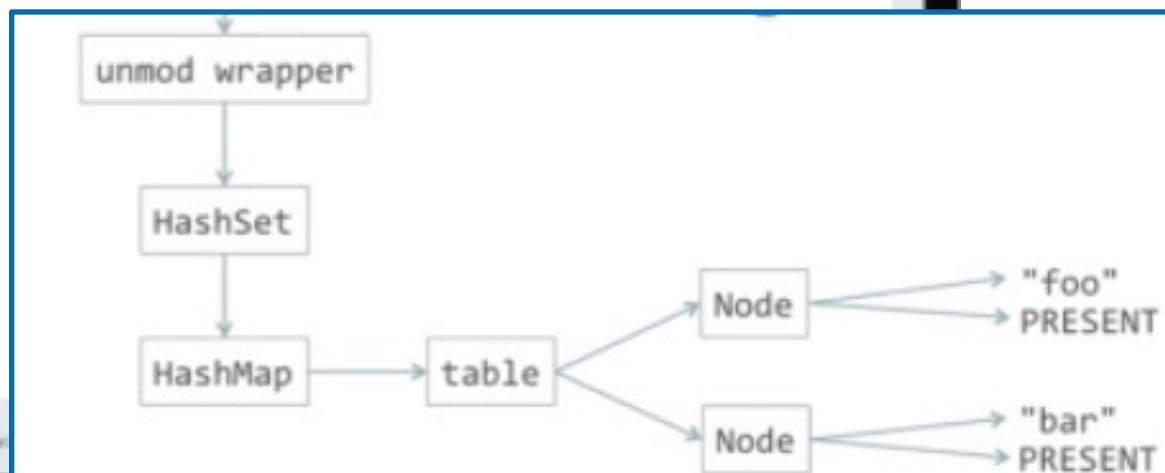
#JavaYoungPups

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

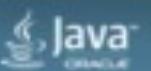




#JavaYoungPups

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





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));
}
```

=>

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

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-Java-11-19>



PART 2:

Additional News and

Changes up to Java 11

- Date API
- InputStream and Reader
- Files
- Multi-Threading with CompletableFuture
- 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



java.io.InputStream / Reader





- **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`
- **Since Java 10:**
 - `long transferTo(Writer)` in class Reader
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?



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

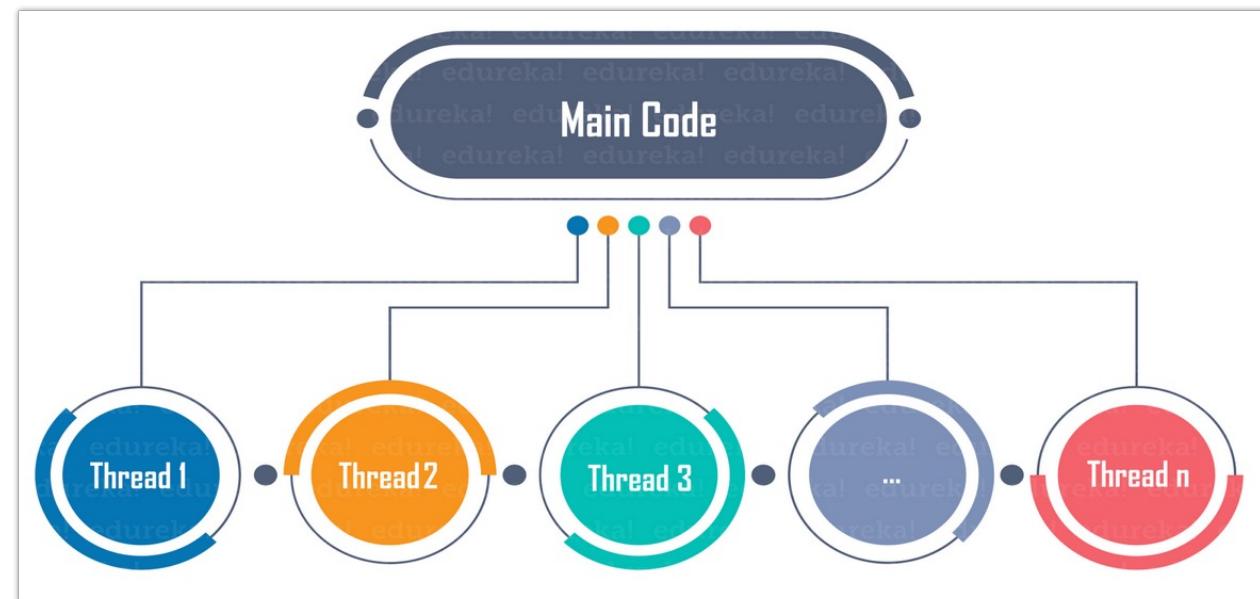
```
final String content = Files.readString(destDath);  
content.lines().forEach(System.out::println);
```

=>

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



Multi-Threading 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());
```

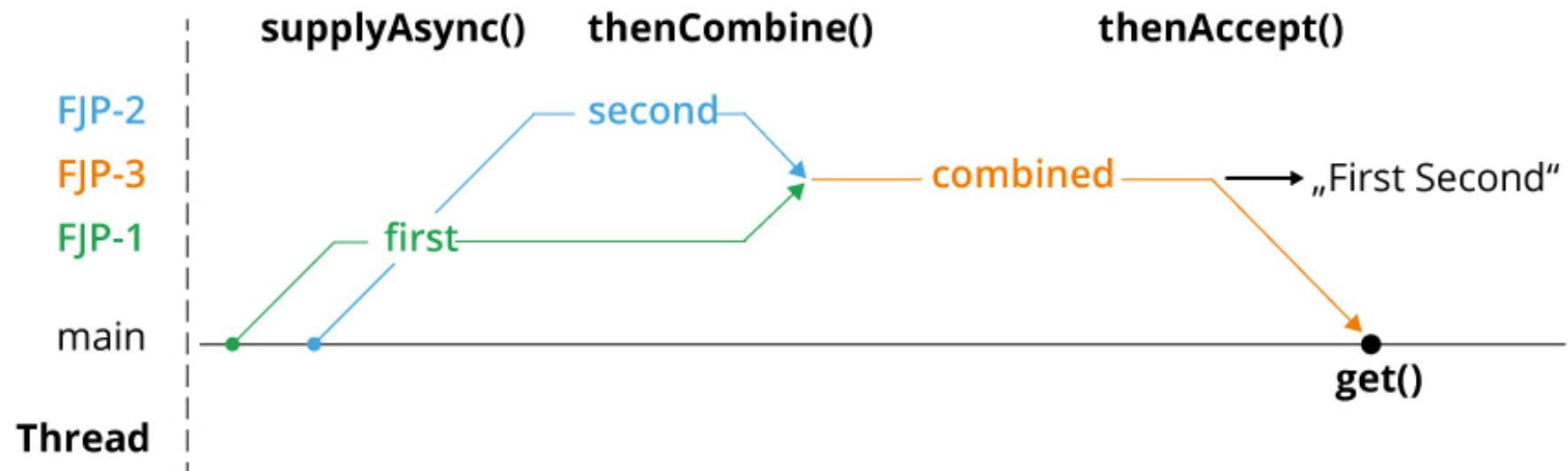
Introduction to CompletableFuture<T>



```
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);
}
```

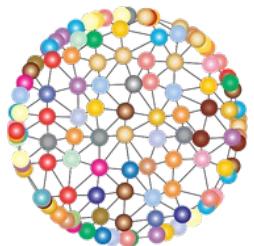
Multi-Threading and the Class CompletableFuture<T>



```
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>?**

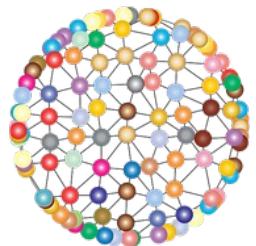
Multi-Threading and the Class CompletableFuture<T>



```
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()**

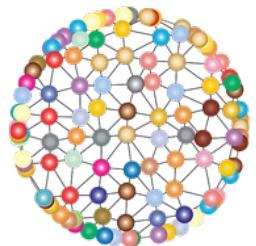
- Provide a fallback value if the data cannot be determined:

```
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.



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()));
}
```

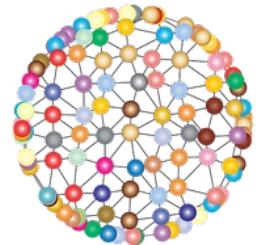


- **Read content as String**

```
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();

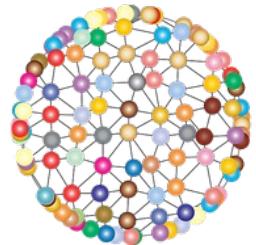
final HttpClient httpClient = HttpClient.newHttpClient();
final HttpResponse<String> response = httpClient.send(request, asString);

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

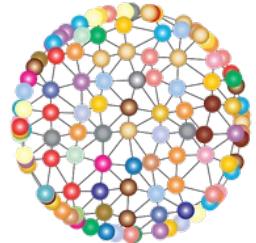


```
static void wait2secForCompletion(final CompletableFuture<?> asyncResponse)
    throws InterruptedException
{
    int i = 0;
    while (!asyncResponse.isDone() && i < 10)
    {
        System.out.println("Step " + i);
        i++;

        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**

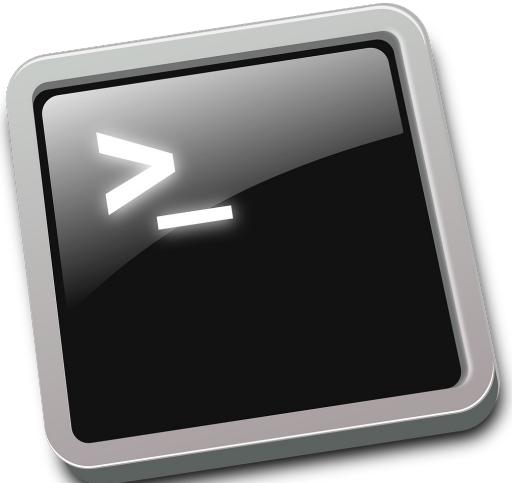
```
.timeout(Duration.ofSeconds(2))  
.header("Content-Type", "application/json")
```

- **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);
    }
}
```

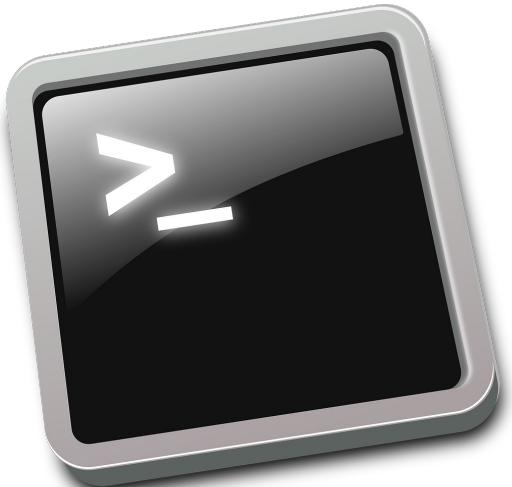
- File must not end with ‘.java’
- File name independent of class name
- File must be executable (chmod +x)



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

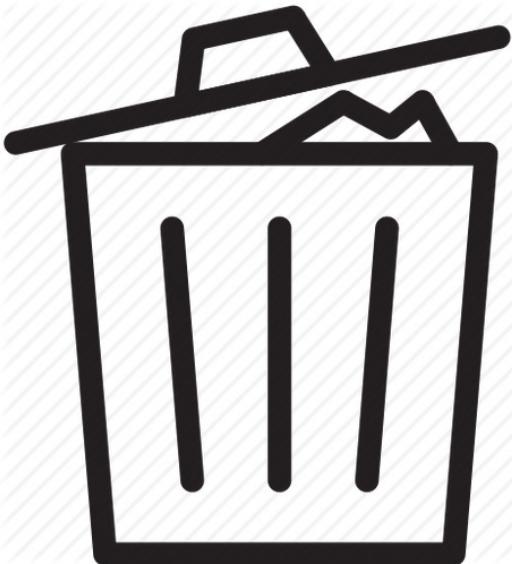
JShell - What we learned so far?



- 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`



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_JDK8**](#)
[**JAXBExample_JDK11**](#)



- **Observer und Observable (java.util)**
- **Thread.destroy() and Thread.stop()**
- **Object.finalize()**
- Constructors of the wrapper classes, e.g. `new Integer()`, `new Long()`, ...
 - use `valueOf()` or `parseXxx()` instead
- 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 2

<https://github.com/Michaeli71/JAX-London-Best-of-Java-11-19>



PART 3: Syntax Enhancements in Java 12 to 17

- Syntax Enhancements for `switch`
- Text Blocks
- Records
- Syntax Enhancements for `instanceof`
- Local Enums and Interfaces
- Sealed Types



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

```
Day0fWeek day = Day0fWeek.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



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



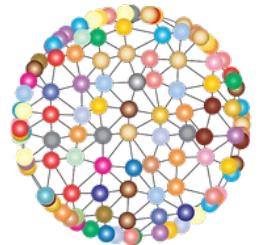
```
record ClosedInterval(int lower, int upper)
{
    public ClosedInterval(int lower, int upper)
    {
        if (lower > upper)
        {
            String errorMsg = String.format("invalid: %d (lower) >= %d (upper)",
                                             lower, upper);
            throw new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



(For now)
**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);
    }
}
```



Records

Beyond the Basics



Storage in different sets



- Let's define a simple record and two different data sets:

```
record SimplePerson(String name, int age, String city) {}
```

```
Set<SimplePerson> speakers = new HashSet<>();  
speakers.add(new SimplePerson("Michael", 51, "Zürich"));  
speakers.add(new SimplePerson("Michael", 51, "Zürich"));  
speakers.add(new SimplePerson("Anton", 42, "Aachen"));
```

```
System.out.print(speakers);
```

```
Set<SimplePerson> sortedSpeakers = new TreeSet<>();  
sortedSpeakers.add(new SimplePerson("Michael", 51, "Zürich"));  
sortedSpeakers.add(new SimplePerson("Michael", 51, "Zürich"));  
sortedSpeakers.add(new SimplePerson("Anton", 42, "Aachen"));
```

```
System.out.print(sortedSpeakers);
```



That should be the result, right?

```
[SimplePerson[name=Michael, age=51, city=Zürich],  
 SimplePerson[name=Anton, age=42, city=Aachen]]
```

Storage in different sets



```
[SimplePerson{name=Michael, age=51, city=Zürich}, SimplePerson{name=Anton, age=42, city=Aachen}]  
Exception in thread "main" java.lang.ClassCastException: class
```

b_slides.RecordInterfaceExample\$1SimplePerson cannot be cast to class java.lang.Comparable

(b_slides.RecordInterfaceExample\$1SimplePerson is in unnamed module of loader 'app';
java.lang.Comparable is in module java.base of loader 'bootstrap')

at java.base/java.util.TreeMap.compare(TreeMap.java:1569)

at java.base/java.util.TreeMap.addEntryToEmptyMap(TreeMap.java:776)

at java.base/java.util.TreeMap.put(TreeMap.java:785)

at java.base/java.util.TreeMap.put(TreeMap.java:534)

at java.base/java.util.TreeSet.add(TreeSet.java:255)

at b_slides.RecordInterfaceExample.main(RecordInterfaceExample.java:32)

Records and Interfaces



- Let's correct the definition of the simple record and implement an interface:

```
record SimplePerson2(String name, int age, String city)
    implements Comparable<SimplePerson2>
{
    @Override
    public int compareTo(SimplePerson2 other)
    {
        return name.compareTo(other.name);
    }
}
```

```
Set<SimplePerson2> sortedSpeakers = new TreeSet<>();
sortedSpeakers.add(new SimplePerson2("Michael", 51, "Zürich"));
sortedSpeakers.add(new SimplePerson2("Michael", 51, "Zürich"));
sortedSpeakers.add(new SimplePerson2("Anton", 42, "Aachen"));
System.out.println(sortedSpeakers);
```

```
[SimplePerson2[name=Anton, age=42, city=Aachen],
 SimplePerson2[name=Michael, age=51, city=Zürich]]
```

Records and Interfaces + Comparator



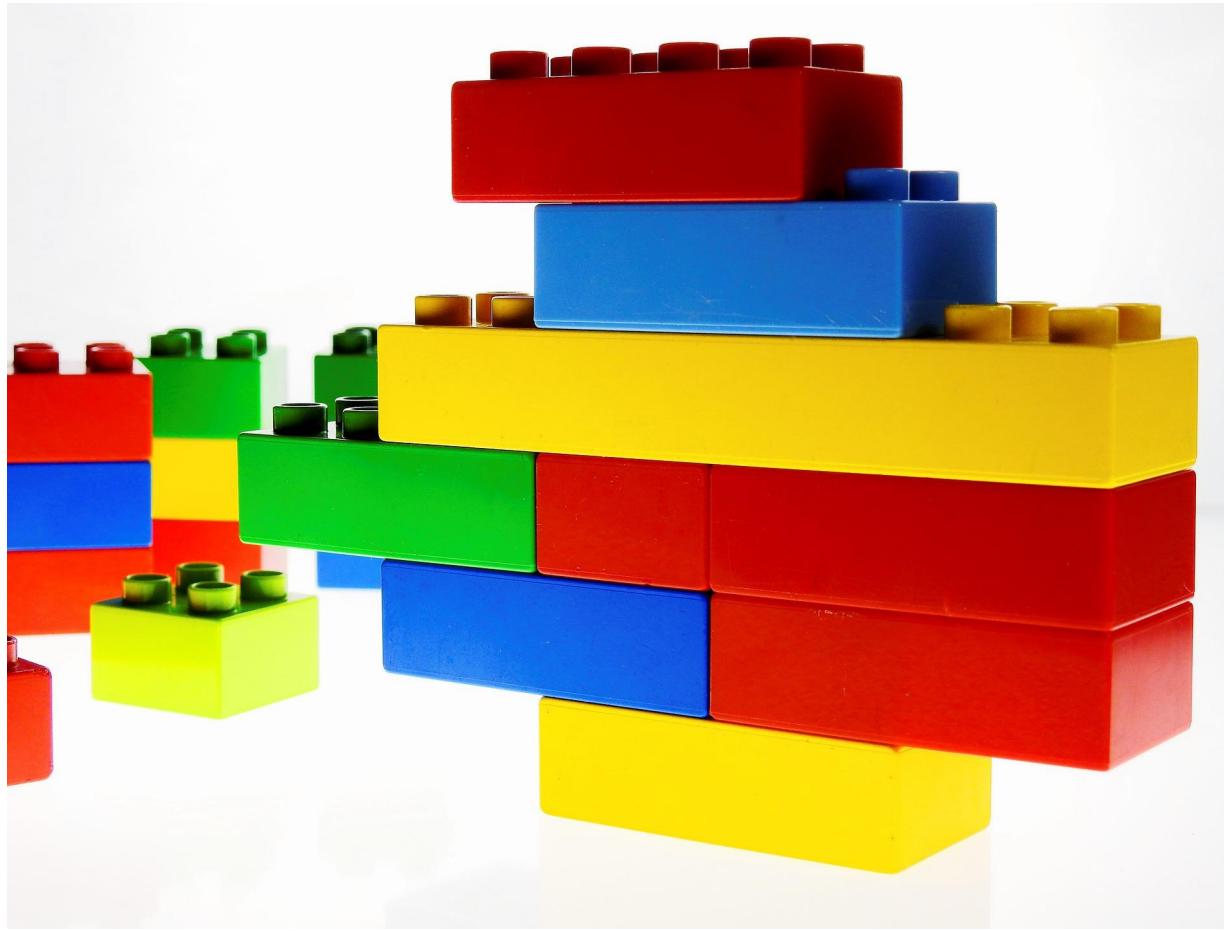
- Let's correct the definition of the comparator:

```
record SimplePerson3(String name, int age, String city)
    implements Comparable<SimplePerson3>
{
    static Comparator<SimplePerson3> byAllAttributes = Comparator.
        comparing(SimplePerson3::name).
        thenComparingInt(SimplePerson3::age).
        thenComparing(SimplePerson3::city);

    @Override
    public int compareTo(SimplePerson3 other)
    {
        return byAllAttributes.compare(this, other);
    }
}
```



Builder ...



Builder like



```
record SimplePerson(String name, int age, String city)
{
    SimplePerson(String name)
    {
        this(name, 0, "");
    }

    SimplePerson(String name, int age)
    {
        this(name, age, "");
    }

    SimplePerson withAge(int newAge)
    {
        return new SimplePerson(name, newAge, city);
    }

    SimplePerson withCity(String newCity)
    {
        return new SimplePerson(name, age, newCity);
    }
}
```

Builder like



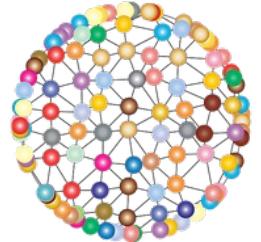
- Problem area, many attributes

```
record ComplexPerson(String firstname, String surname, LocalDate birthday,  
                     int height, int weight, String addressStreet,  
                     String addressNumber, String city)  
{  
    public static void main(String[] args)  
    {  
        ComplexPerson john = new ComplexPerson("John", "Smith", LocalDate.of(2010, 6, 21),  
                                              170, 90, "Fuldastrasse", "16a", "Berlin");  
  
        ComplexPerson mike = new ComplexPersonBuilder().withFirstName("Mike").  
                                         withBirthday(LocalDate.of(2021, 1, 21)).  
                                         withSurName("Peters").build();  
        System.out.println(mike);  
    }  
}
```

- But: ComplexPersonBuilder would have to be implemented by yourself



**What would be another
way to reduce complexity?**



Builder like



- Define records for individual components

```
record Address(String addressStreet, String addressNumber, String city) {}

record BodyInfo(int height, int weight) {}

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

record ReducedComplexPerson(PersonDTO person, BodyInfo bodyInfo, Address address)
{
    public static void main(String[] args)
    {
        var john = new PersonDTO("John", "Smith", LocalDate.of(2010, 6, 21));
        var bodyInfo = new BodyInfo(170, 90);
        var address = new Address("Fuldastrasse", "16a", "Berlin");

        var rcp = new ReducedComplexPerson(john, bodyInfo, address);
    }
}
```



Date Range ... Immutability

A large black rectangular sign with the date '12.12' written in large, white, stylized numbers with a red outline.

Record for Date range



```
public class RecordImmutabilityExample
{
    public static void main(String[] args)
    {
        record DateRange(Date start, Date end) {}

        DateRange range1 = new DateRange(new Date(71,1,7), new Date(71,2,27));
        System.out.println(range1);

        DateRange range2 = new DateRange(new Date(71,6,7), new Date(71,2,27));
        System.out.println(range2);
    }
}
```

```
DateRange[start=Sun Feb 07 00:00:00 CET 1971, end=Sat Mar 27 00:00:00 CET 1971]
DateRange[start=Wed Jul 07 00:00:00 CET 1971, end=Sat Mar 27 00:00:00 CET 1971]
```

- **Include validity check for invariant start < end**

Record for Date range



```
record DateRange(Date start, Date end)
{
    DateRange
    {
        if (!start.before(end))
            throw new IllegalArgumentException("start >= end");
    }
}

DateRange range1 = new DateRange(new Date(71,1,7), new Date(71,2,27));
System.out.println(range1);

DateRange range2 = new DateRange(new Date(71,6,7), new Date(71,2,27));
System.out.println(range2);
```

DateRange[start=Sun Feb 07 00:00:00 CET 1971, end=Sat Mar 27 00:00:00 CET 1971]
Exception in thread "main" java.lang.IllegalArgumentException: start >= end
at b_slides.RecordImmutabilityExample3\$1DateRange.<init>(RecordImmutabilityExample3.java:21)
at b_slides.RecordImmutabilityExample3.main(RecordImmutabilityExample3.java:28)

Record for Date range



```
record DateRange(Date start, Date end)
{
    DateRange
    {
        if (!start.before(end))
            throw new IllegalArgumentException("start >= end");
    }
}
```

```
DateRange range1 = new DateRange(new Date(71,1,7), new Date(71,2,27));
System.out.println(range1);
```

```
range1.start.setTime(new Date(71,6,7).getTime());
System.out.println(range1);
```

```
DateRange[start=Sun Feb 07 00:00:00 CET 1971, end=Sat Mar 27 00:00:00 CET 1971]
DateRange[start=Wed Jul 07 00:00:00 CET 1971, end=Sat Mar 27 00:00:00 CET 1971]
```

- **Immutability only for immutable attributes => ATTENTION: reference semantics in Java**

Record for Date range



```
record DateRange(LocalDate start, LocalDate end)
{
    DateRange
    {
        if (!start.isBefore(end))
            throw new IllegalArgumentException("start >= end");
    }
}

DateRange range1 = new DateRange(LocalDate.of(1971,1,7), LocalDate.of(1971,2,27));
System.out.println(range1);

DateRange range2 = new DateRange(LocalDate.of(1971,6,7), LocalDate.of(1971,2,27));
System.out.println(range2);

DateRange[start=1971-01-07, end=1971-02-27]
Exception in thread "main" java.lang.IllegalArgumentException: start >= end
at b_slides.RecordImmutabilityExample4$1DateRange.<init>(RecordImmutabilityExample4.java:21)
at b_slides.RecordImmutabilityExample4.main(RecordImmutabilityExample4.java:28)
```

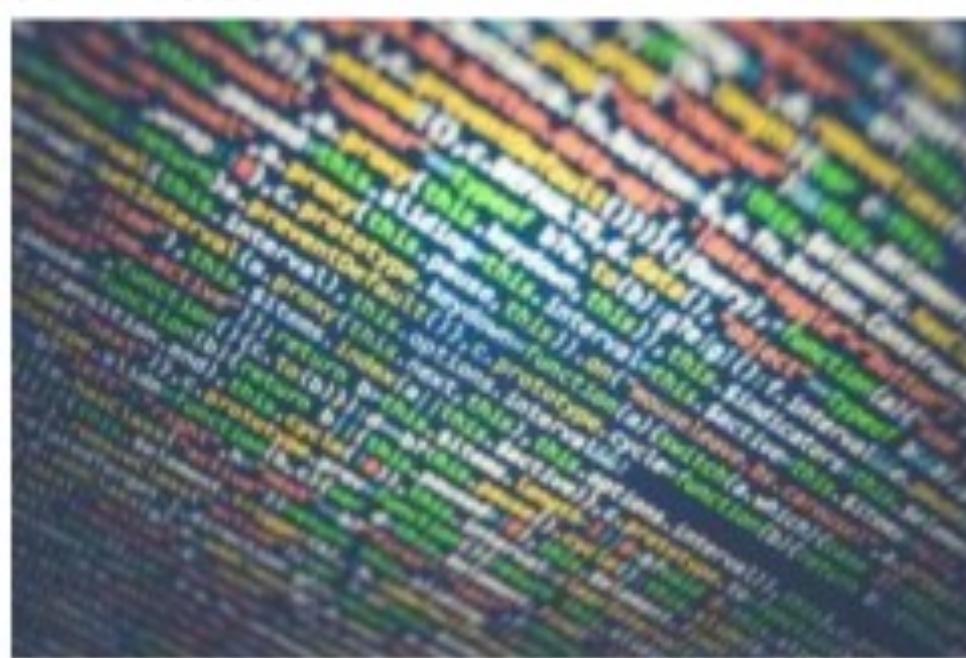
Records



DEMO & Hands on



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



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"));
```

```
}
```

```
}
```



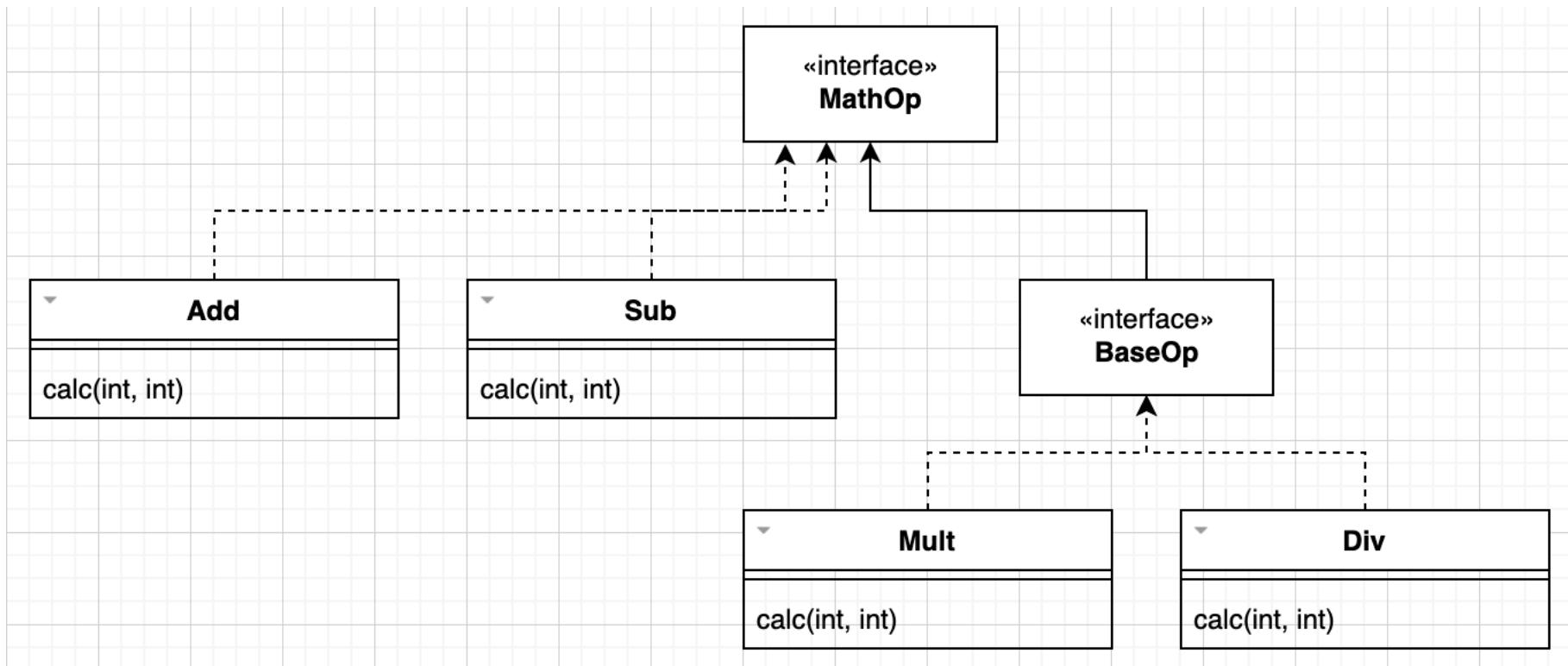
Sealed Types



Sealed Types



- Control inheritance and specify which classes can extend a base class, i.e. which other classes or interfaces may form subtypes of it.



Sealed Types – Control inheritance



- Specify which other classes or interfaces may subtype from it.

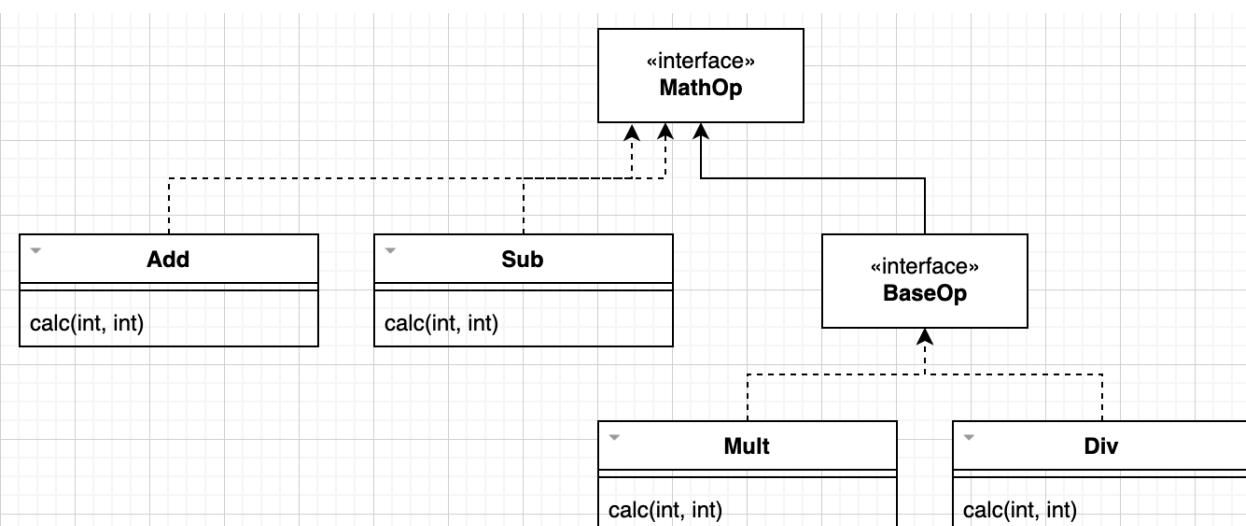
```
public class SealedTypesExamples
{
    sealed interface MathOp
        permits BaseOp, Add, Sub // <= permitted subtypes
    {
        int calc(int x, int y);
    }
}
```

// With non-sealed you can provide base classes within the inheritance hierarchy

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

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

With sealed we can seal an inheritance hierarchy and allow only the explicitly specified types. These must be sealed, non-sealed or final.



Sealed Types

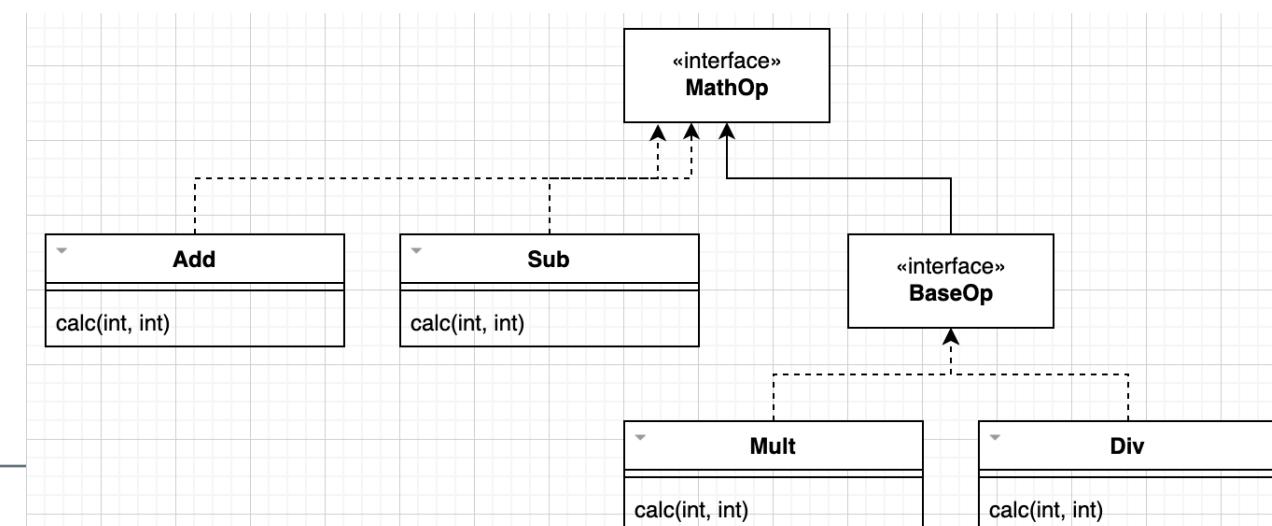


```
..  
// direct implementierung must be final  
final class Add implements MathOp  
{  
    @Override  
    public int calc(int x, int y)  
    {  
        return x + y;  
    }  
}  
final class Sub implements MathOp  
{  
    ...  
}  
final class Mult extends BaseOp  
{  
}  
final class Div extends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

A class that is marked as sealed must have subclasses.
which in turn are listed after permits

A class marked as non-sealed can function as a base class
and classes can be derived from it.

A class marked as final forms the end point of a derivation
hierarchy as usual.



Things to know for Sealed Types



- specify which other classes or interfaces may subtype from it.
 - Sealed types can expose behavior over interfaces in the development of libraries, but allow control over possible implementations.
 - Sealed types restrict with regard to the extensibility of class hierarchies and should therefore be used with caution. Flexibility not foreseen during implementation can be subsequently disruptive.
-



Exercises PART 3

<https://github.com/Michaeli71/JAX-London-Best-of-Java-11-19>



PART 4:

What's new in Java 12 to 17

- String APIs
 - CompactNumberFormat
 - Files
 - Teeing()-Kollektor
 - Stream.toList() / mapMulti()
 - Deprecations
-



Enhancement in class String



Extension in `java.lang.String`



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

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



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

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

- However, negative values are also allowed:

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

Extension in `java.lang.String`



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

```
'      Test
'
9
```

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

```
'6789
'
5
```

Extension in `java.lang.String`



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

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

Extension in `java.lang.String`



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

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

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

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



- Example: all characters of a string
 - Convert to UPPERCASE
 - Then remove T
 - Finally split
- Up to now, this has been realized as follows, for example:

```
var text = "This is a Test";  
  
var upperCase = text.toUpperCase();  
var noTs = upperCase.replaceAll("T", "");  
var result = noTs.split(" ");  
  
System.out.println(Arrays.asList(result));
```

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



- Example: all characters of a string
 - Convert to UPPERCASE
 - Then remove T
 - Finally split
- Up to now, this has been realized as follows, for example:

```
var text = "This is a Test";  
  
var upperCase = text.toUpperCase();  
var noTs = upperCase.replaceAll("T", "");  
var result = noTs.split(" ");  
  
System.out.println(Arrays.asList(result));
```

[HIS, IS, A, ES]

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



- Example: all characters of a string with `transform()`

- Convert to UPPERCASE
- Then remove T
- Finally split

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

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

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

- Analogous to `map()` in streams, for connecting transformations one after the other.
- Rather theoretical practical

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



Addition CompactNumberFormat



Utility-Class CompactNumberFormat



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

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

CompactNumberformat Style



```
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]



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. London too",
                  StandardCharsets.ISO_8859_1);
Files.writeString(fileEnc2, "Zürich is beautiful. London 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. London too",
                  StandardCharsets.ISO_8859_1);
Files.writeString(fileEnc2, "Zürich is beautiful. London 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



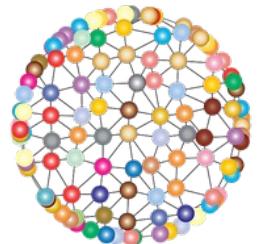
Teeing() Collector





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()!**



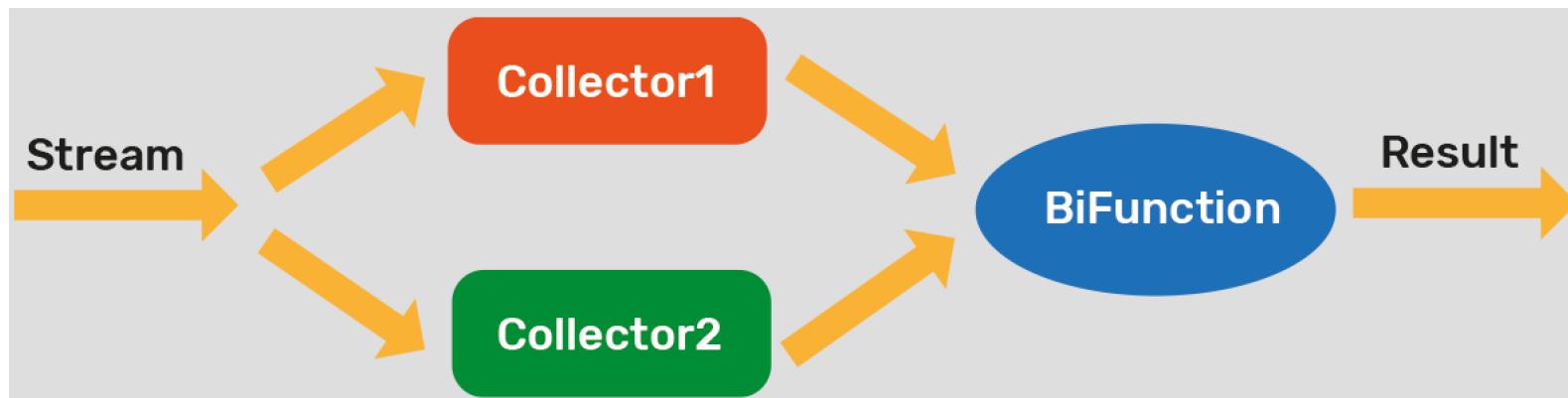
What is missing?



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(2, 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(2, 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=58]
```

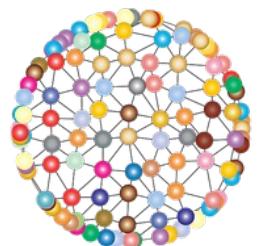
Excursion Pair<T>



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



Not so good!!

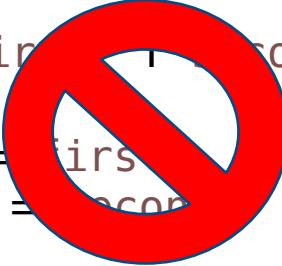
Instead of self-made pair, better use Records in modern Java!



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



```
record LongPair(long count, long sum)
{}
```



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]
    (list1, list2) -> List.of(list1, list2));

System.out.println(result);
    [[Michael, Mike], [Tim, Tom]]
```



Java 16



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



```
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();
```



mapMulti()

Why mapMulti()?



// OLD

```
Stream.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
         Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3"))
    .flatMap(Optional::stream)
    .forEach(System.out::print);
```

// NEW

```
Stream.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
         Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3"))
    .mapMulti(Optional::ifPresent). // !!!
    .forEach(System.out::print);
```

Why mapMulti()?



```
// NEW
Stream.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
         Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3"))
    .mapMulti(Optional::ifPresent). // !!!
    forEach(System.out::print);

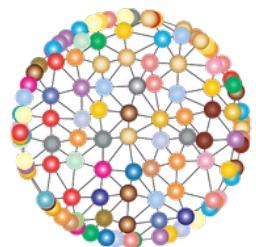
// More the thought like imperative for loop
var elements = List.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
                      Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3"));

for (Optional optElem : elements )
{
    optElem.ifPresent(System.out::print);
}
```



How do we build that?

```
Stream.of(List.of(1, 2, 3),  
         "ABC", null,  
         Set.of("X", "Y", "Z"));
```



=>

```
[1, 2, 3, ABC, null, Z, Y, X]
```

Why mapMulti()? Example expandIterables()



```
// OLD  
var stream = Stream.of(List.of(1, 2, 3), "ABC", null, Set.of("X", "Y", "Z"));
```

```
Stream<Object> expandedStream = stream.flatMap(e -> {  
    if (e instanceof Iterable<?> iterable) {  
        // Trick to convert Iterable to Stream  
        return StreamSupport.stream(iterablespliterator(), false);  
    }  
    return Stream.of(e);  
});
```

```
System.out.println(expandedStream.toList());
```



[1, 2, 3, ABC, null, Z, Y, X]

Why mapMulti()? Example expandIterables()



```
// NEW
var stream2 = Stream.of(List.of(1, 2, 3), "ABC", null, Set.of("X", "Y", "Z"));

Stream<Object> expandedStream2 = stream2.mapMulti(MapMultiExample2::expandIterable);

System.out.println(expandedStream2.toList());
```

```
static void expandIterable(Object e, Consumer<Object> c) {
    if (e instanceof Iterable<?> iterable) {
        for (Object elem : iterable) {
            expandIterable(elem, c);
        }
    } else {
        c.accept(e);
    }
}
```

[1, 2, 3, ABC, null, Z, Y, X]



Deprecations



Deprecations



- **Construction of wrappers:**

```
/Users/michaelinden/java8-
workspace/Java17Examples/src/main/java/primitives/PrimitiveCtorDeprecati
onExample.java:6: warning: [removal] Long(long) in Long has been
deprecated and marked for removal
```

```
    Long myLong = new Long(1234L);
               ^
```

```
/Users/michaelinden/java8-
workspace/Java17Examples/src/main/java/primitives/PrimitiveCtorDeprecati
onExample.java:8: warning: [removal] Integer(int) in Integer has been
deprecated and marked for removal
```

```
    Integer myInt = new Integer(1234);
                   ^
```



Exercises PART 4

<https://github.com/Michaeli71/JAX-London-Best-of-Java-11-19>



PART 5: News in the JVM in Java 12 to 17

- Helpful NullPointerExceptions
 - JMH (Microbenchmarks)
 - JPackages
 - JavaScript-Engine (got removed)
-



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)`

Helpful NullPointerExceptions



```
public static void main(final String[] args)
{
    int width = getWindowManager().getWindow(5).size().width();
    System.out.println("Width: " + width);
}
```

Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"jvm.NPE_Third_Example\$Window.size()" because the return value of
"jvm.NPE_Third_Example\$WindowManager.getWindow(int)" is null
at jvm.NPE_Third_Example.main(NPE_Third_Example.java:7)

Helpful NullPointerExceptions



```
public static WindowManager getWindowManager()
{
    return new WindowManager();
}
```

```
public static class WindowManager
{
    public Window getWindow(final int i)
    {
        return null;
    }
}
```

```
public static record Window(Size size) {}
public static record Size(int width, int height) {}
```



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.

Simple Start Stop measurements (Please don't do it!!)



- 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 (Avoid if possible!)



- 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 (Becomes complicated)



- **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;
```



- **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
-

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:

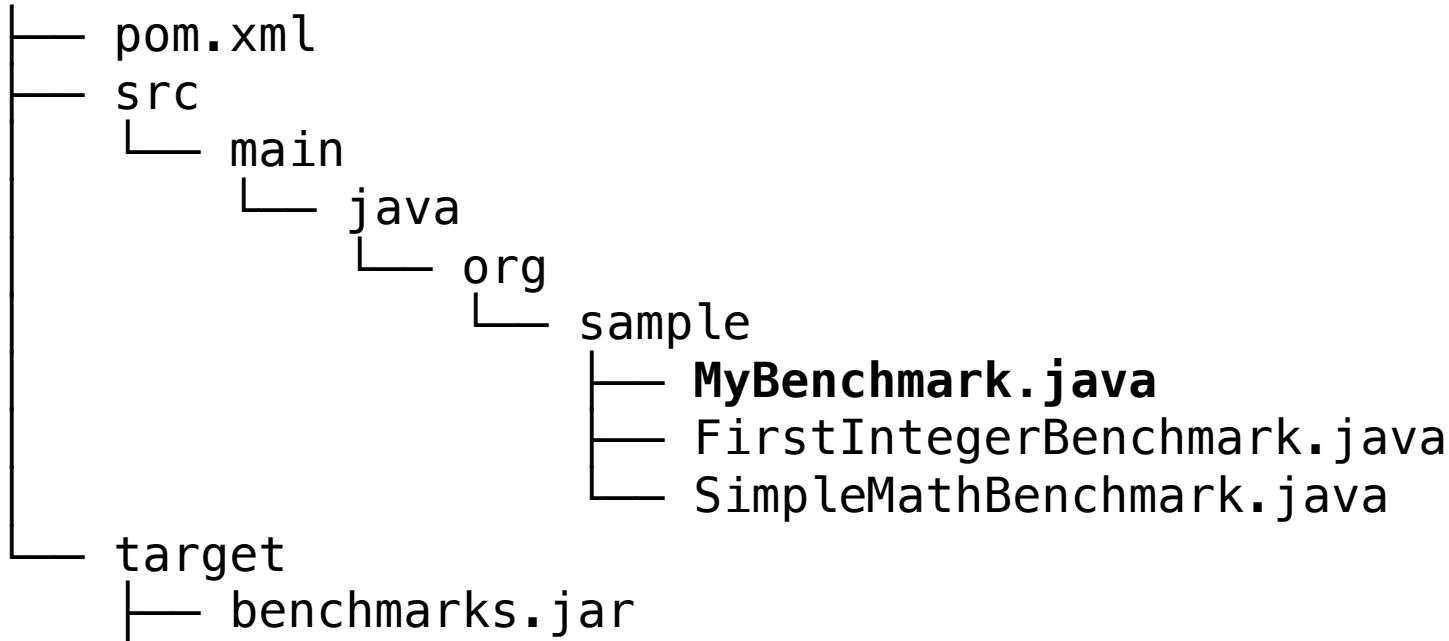
```
public class MyBenchmark
{
    @Benchmark
    public void testMethod()
    {
        // This is a demo/sample template for building your JMH benchmarks. Edit
        // as needed.
        // Put your benchmark code here.
    }
}
```

- 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()`
-



Sample 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	



Sample results

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



Sample results

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

Benchmark	(value)	Mode	Cnt	Score	Error	Units
StringJoinBenchmark.concatUsingStringBuilder	10000	avgt	10	0.016	± 0.001	ms/op
StringJoinBenchmark.concatUsingStringBuilder	100000	avgt	10	0.172	± 0.009	ms/op
StringJoinBenchmark.stringConcatABC	10000	avgt	10	9.634	± 0.812	ms/op
StringJoinBenchmark.stringConcatABC	100000	avgt	10	662.953	± 4.029	ms/op
StringJoinBenchmark.stringPlusABC	10000	avgt	10	7.926	± 0.149	ms/op
StringJoinBenchmark.stringPlusABC	100000	avgt	10	662.343	± 2.912	ms/op



DEMO & Hands on



JPackage





▼ PackagingDemo

► JRE System Library [JavaSE-16]

▼ src/main/java

 ▼ de.java17

 ▼ ApplicationExample.java

 ▼ ApplicationExample

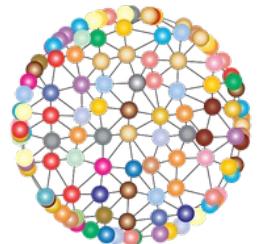
 main(String[]) : void

► src

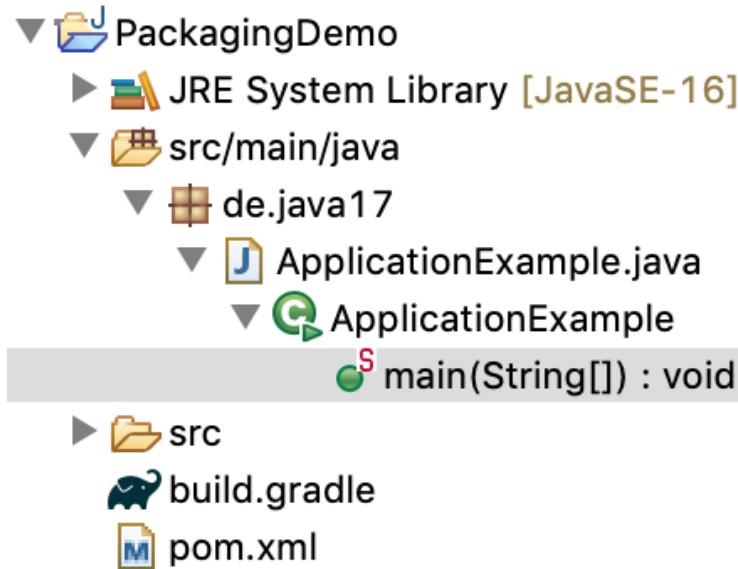
 build.gradle

 pom.xml

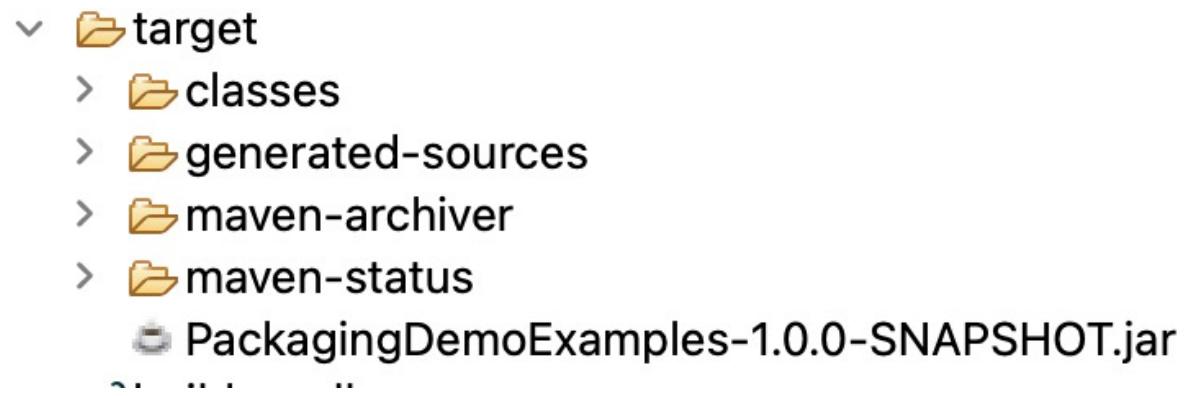
```
public class ApplicationExample {  
    public static void main(String[] args) {  
        System.out.println("First JPackage");  
  
        JOptionPane.showConfirmDialog(null, "Generated by jpackage", "DEMO",  
            JOptionPane.OK_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE, null)  
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



How to build?

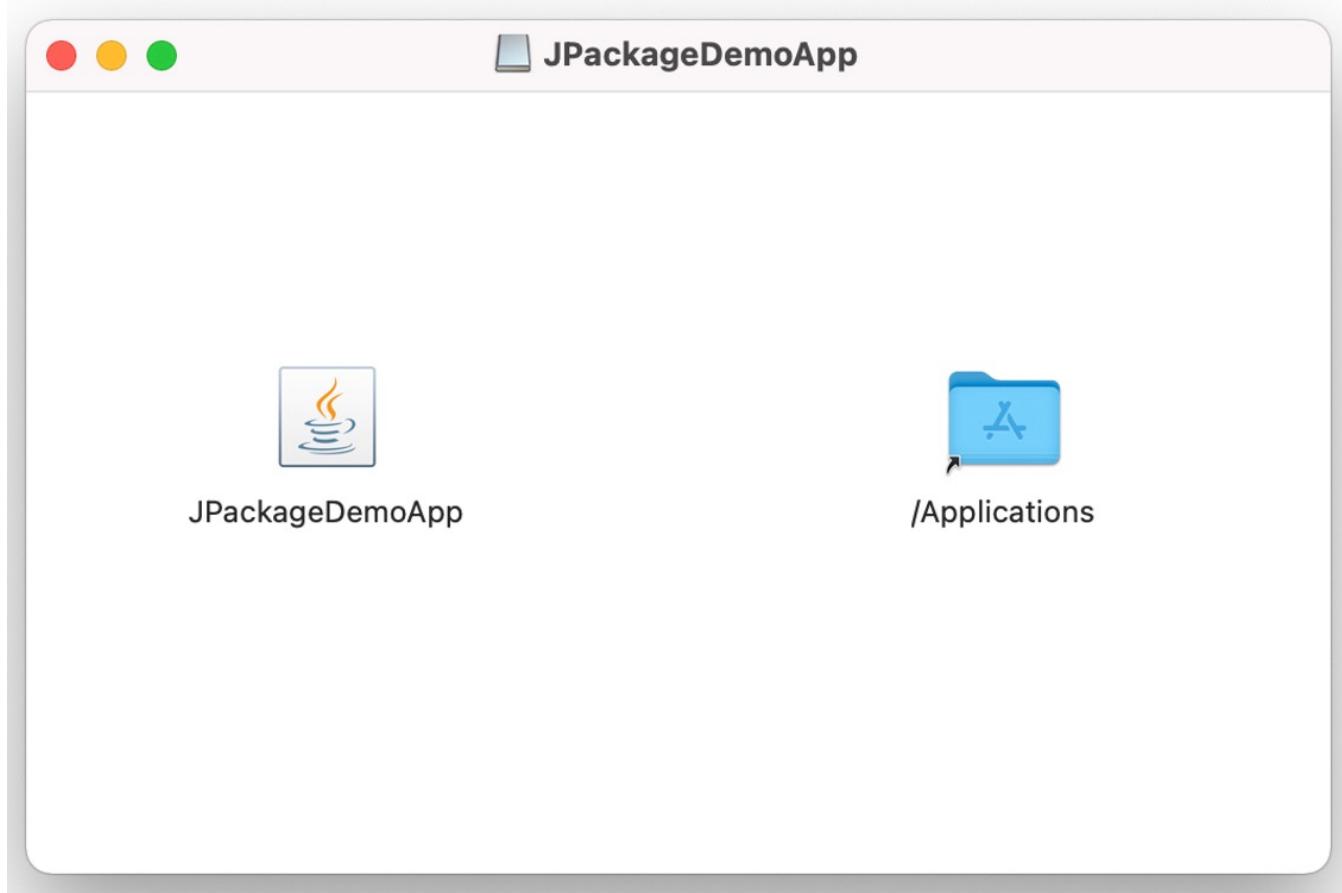


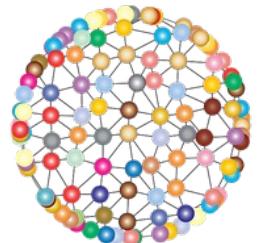
`mvn clean install`





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





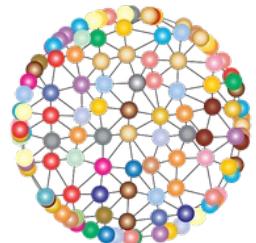
What about 3rd party libraries?



```
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?



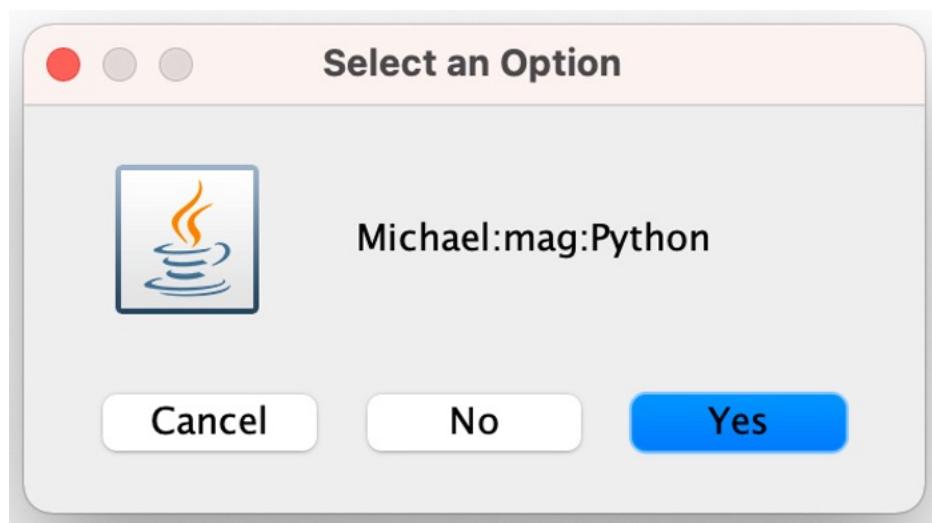


```
<plugin>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <phase>process-sources</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>

      <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



Nashorn Java Script Engine

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





- 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());

    // unfortunately no (direct) access to value, only via 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



Exercises PART 5

<https://github.com/Michaeli71/JAX-London-Best-of-Java-11-19>



Recap: News in Java 17

- Overview: What's inside?
 - Syntax Enhancement 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](#)

Which of these is really relevant to us?

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

Version overview – <https://javaalmanac.io/>



The Java Version Almanac

javaalmanac.io



[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	17	16	15	14	13	12	11	10	9	8	...
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		1.1	...										
Java 1.0	EOL	API Lang VM		...											

Data Source



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:^{*}

```
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;  
}
```

^{*}the whole thing would be much worse without Java 16, since we would have to add a line of cast for each type

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");  
    }  
}
```



PART 6: News in Java 18

A large, bold, red number '18' is displayed in a three-dimensional perspective, casting a soft shadow on a light gray surface below it. The digits are slightly rounded and have a metallic texture, giving them a prominent appearance.



Build-Tools and IDEs



IDE & Tool Support for Java 18



- Current IDEs & Tools basically good
- Eclipse: Version 2022-06 (2022-03 with additional plugin)
- IntelliJ: Version 2022.1
- Maven: 3.8.5, Compiler-Plugin: 3.8.1 (prefer 3.10)
- Gradle: 7.4.2, officially Gradle 7.5
- Activation of preview features necessary
 - In dialogs
 - In build scripts



Maven™

 **Gradle**

The Gradle logo consists of a stylized blue elephant icon followed by the word "Gradle" in a bold, sans-serif font.

IDE & Tool Support Java 18



- Eclipse 2022-03 with Plugin
- Activation of preview features necessary

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: Search All Markets All Categories Go

Featured

Java 18 Support for Eclipse 2022-03 (4.23)

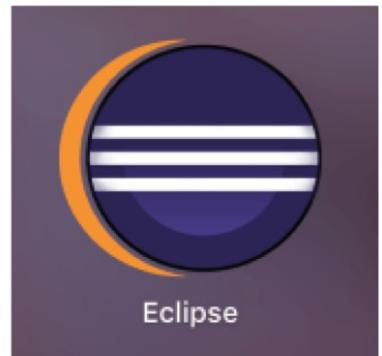
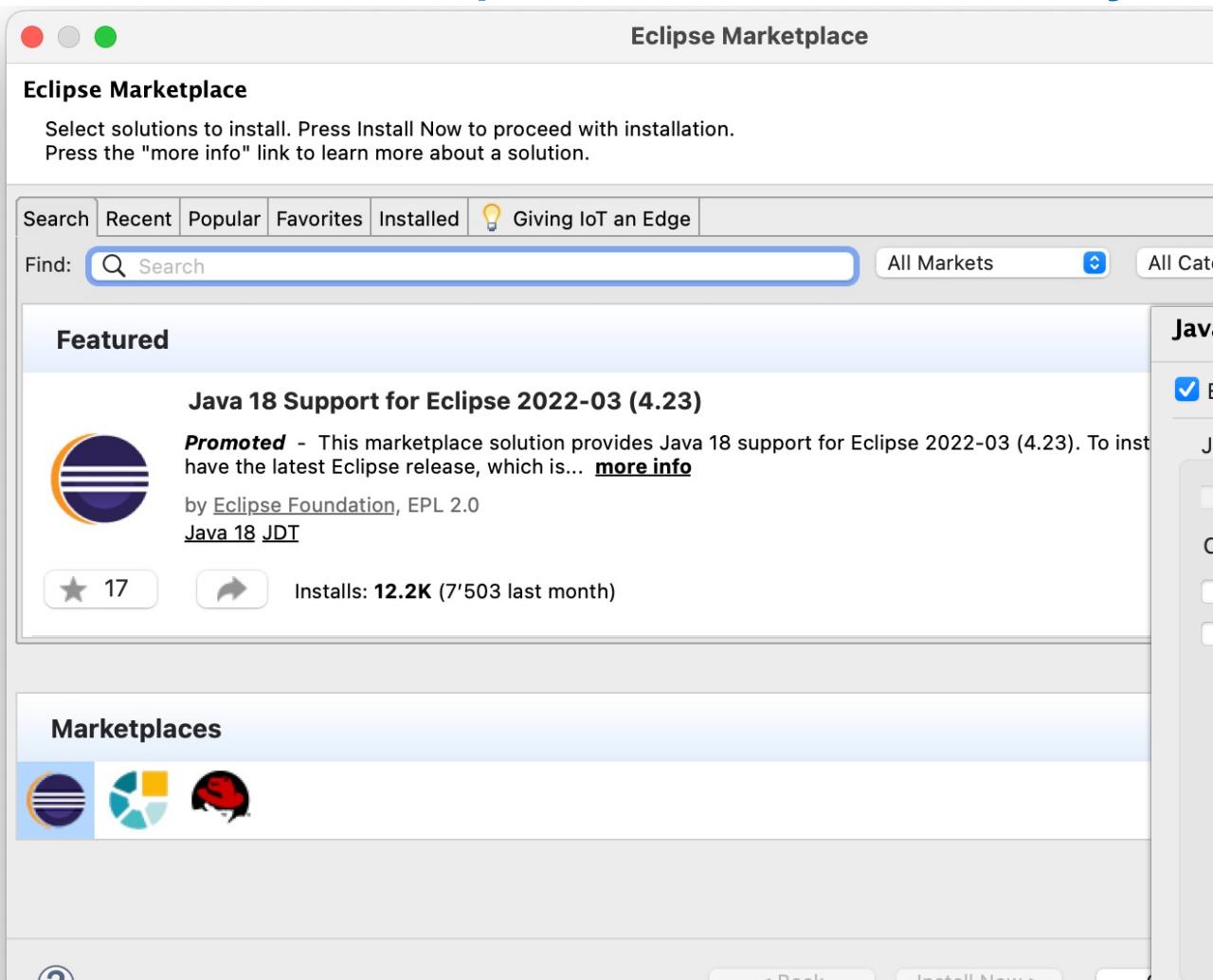
Promoted - This marketplace solution provides Java 18 support for Eclipse 2022-03 (4.23). To install have the latest Eclipse release, which is... [more info](#)

by Eclipse Foundation, EPL 2.0
[Java 18 JDT](#)

17 installs: 12.2K (7'503 last month)

Marketplaces

② Back Install Now



Java Compiler

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

JDK Compliance

Use compliance from execution environment on the ['Java Build Path'](#)

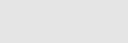
Compiler compliance level: **18** 

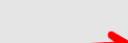
Use '--release' option

Use default compliance settings

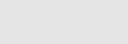
Enable preview features for Java 18 

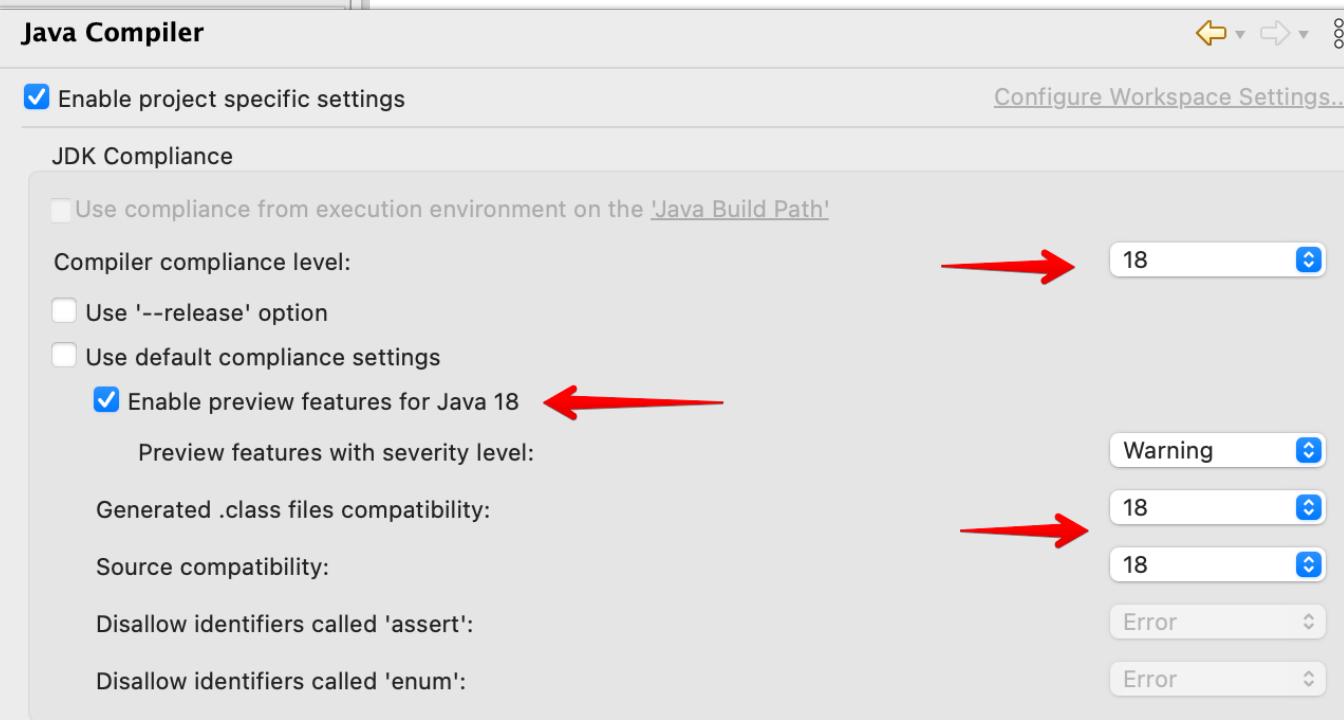
Preview features with severity level:

Generated .class files compatibility: **Warning** 

Source compatibility: **18** 

Disallow identifiers called 'assert': **18** 

Disallow identifiers called 'enum': **Error** 



IDE & Tool Support Java 18 for Vector API Incubator



Run Configurations

Create, manage, and run configurations

Run a Java application

Name: VectorApiExample

Main Arguments JRE Dependencies Source Environment Common Prototype

Program arguments:

VM arguments:

--add-modules jdk.incubator.vector

Use the -XstartOnFirstThread argument when launching with SWT

Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching

Use @argfile when launching

Working directory:

Default: \${workspace_loc:Java18Examples}

Other: _____

Workspace... File System... Variables...

A screenshot of the Eclipse IDE's "Run Configurations" dialog. The configuration is named "VectorApiExample". In the "VM arguments" section, the argument "--add-modules jdk.incubator.vector" is entered, with a red arrow pointing to it from the left. Below this, there are three checked checkboxes for launching with SWT and showing code details, and one unchecked checkbox for using an argfile. The "Working directory" section shows "Default" selected with the path "\${workspace_loc:Java18Examples}". At the bottom, there are buttons for "Workspace...", "File System...", and "Variables...".



IDE & Tool Support



- Activation of preview features necessary

Project Structure

← →

Project Settings

- Project** (selected)
- Modules
- Libraries
- Facets
- Artifacts

Platform Settings

- SDKs
- Global Libraries

Project

Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: Java18Examples

SDK: 18 Oracle OpenJDK version 18.0.1

Language level: 18 (Preview) - Pattern matching for switch (second preview)

Compiler output: ~/java8-workspace/Java18Examples/out

Used for modules' subdirectories, Production and Test directories for the corresponding sources.



IDE & Tool Support for Vector API Incubator



Preferences

Build, Execution, Deployment > Compiler > Java Compiler

Use compiler: Javac

Use '--release' option for cross-compilation (Java 9 and later)

Project bytecode version: Same as language level

Per-module bytecode version:

Module	Target bytecode version
Java18Examples	18

Javac Options

Use compiler from module target JDK when possible

Generate debugging info

Report use of deprecated features

Generate no warnings

Additional command line parameters: (!' recommended in paths for cross-platform configurations)

--add-modules jdk.incubator.vector

Override compiler parameters per-module:

Module	Compilation options
Java18Examples	--enable-preview

Cancel Apply OK



IDE & Tool Support for Vector API Incubator



Run/Debug Configurations

Name: VectorApiExample Store as project file

Build and run

java 18 SDK of 'Java18Exam' --add-modules jdk.incubator.vector ↓ Modify options

apis.VectorApiExample

Program arguments

VM options. CLI arguments to the 'Java' command. Example: -ea -Xmx2048m. VM

Working directory: _JAVA-BEST-OF/2022/ch.open_Java_11_bis_19_WORKSHOP_2022/Java18Exam Working directory

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started OK

Code Coverage

Packages and classes to include in coverage data Modify

- + apis.*

Edit configuration templates... Cancel Apply OK





- Activation of preview features / incubator necessary

```
sourceCompatibility=18  
targetCompatibility=18
```

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

```
tasks.withType(JavaCompile).configureEach {  
    options.compilerArgs += ["--enable-preview", "--add-modules", "jdk.incubator.vector"]  
}
```



IDE & Tool Support



- Activation of preview features / incubator necessary

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.10</version>
    <configuration>
      <source>18</source>
      <target>18</target>
      <!-- Important for Java Syntax-News -->
      <compilerArgs>--enable-preview</compilerArgs>
    </configuration>
  </plugin>
</plugins>
```

A screenshot of an IDE showing the configuration of the maven-compiler-plugin. The configuration block includes source and target versions set to 18, and compilerArgs set to --enable-preview. It also includes compilerArgs with additional arguments: --add-modules and jdk.incubator.vector. The incubator module argument is highlighted in blue.

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.10.0</version>
  <configuration>
    <source>18</source>
    <target>18</target>
    <compilerArgs>
      <arg>--enable-preview</arg>
      <arg>--add-modules</arg>
      <arg>jdk.incubator.vector</arg>
    </compilerArgs>
  </configuration>
</plugin>
```



JEPs in Java

18

Java 18 – What's inside?



- JEP 400: UTF-8 by Default
 - [JEP 408: Simple Web Server](#)
 - [JEP 413: Code Snippets in Java API Documentation](#)
 - JEP 416: Reimplement Core Reflection with Method Handles
 - [JEP 417: Vector API \(Third Incubator\)](#)
 - [JEP 418: Internet-Address Resolution SPI](#)
 - JEP 419: Foreign Function & Memory API (Second Incubator)
 - [JEP 420: Pattern Matching for switch \(Second Preview\)](#)
 - [JEP 421: Deprecate Finalization for Removal](#)
-



[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 19	DEV	API Notes	JDK JRE	18	17	16	15	14	13	12	11	10	9	...
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												



Java 18

Status	In Development
Release Date	2022-03-15
EOL Date	2022-09-15
Bytecode Version	62.0
API Changes	Compare to 17 - 16 - 15 - 14 - 13 - 12 - 11 - 10 - 9 - 8 - 7 - 6 - 5 - 1.4 - 1.3 - 1.2 - 1.1
Documentation	Release Notes , JavaDoc
SCM	git

[Data Source](#)

New Features

JVM

- UTF-8 by Default ([JEP 400](#))

Language

- Pattern Matching for switch (Second Preview) [Preview](#) ([JEP 420](#))

API

- Reimplement Core Reflection with Method Handles ([JEP 416](#))
- Internet-Address Resolution SPI ([JEP 418](#))
- Deprecate Finalization for Removal ([JEP 421](#))

Tools

- Simple Web Server ([JEP 408](#))
- Code Snippets in Java API Documentation ([JEP 413](#))



Sandbox

Instantly compile and run Java 18 snippets without a local Java installation.

Java18.java

▶ Run

18+36-2087

```
import javax.lang.model.SourceVersion;

public class Java18 {

    public static void main(String[] args) {
        System.out.println("Runtime required for this: " + SourceVersion.RELEASE_18.runtimeVersion());

        switchSupportingNull(null);
        switchSupportingNull("JAVA");
    }

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



Sandbox

Instantly compile and run Java 18 snippets without a local Java installation.

Java18.java

▶ Run

18+36-2087

```
Java18.java:14: warning: [preview] null in switch cases is a preview feature and may be removed in a future release
      case null -> System.out.println("null is allowed in preview");
                         ^
Runtime required for this: 18
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.toUpperCase()" because "<parameter name="s">" is null
  at Java18.switchSupportingNull(Java18.java:13)
  at Java18.main(Java18.java:8)
```



JEP 408: Simple Web Server



JEP 408: Simple Web Server



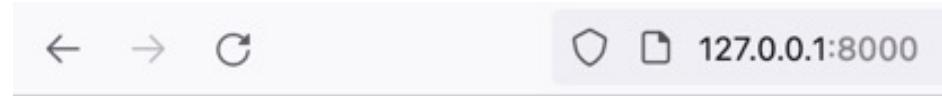
- The goal for this JEP was to integrate a minimalistic web server into the JDK without much need for configuration, which can only provide static files. This facilitates prototyping, file sharing or can also be useful for simple tests.

```
michaelinden@MBP-von-Michael ~ % jwebserver
Binding to loopback by default. For all interfaces use "-b 0.0.0.0" or "-b ::".
Serving /Users/michaelinden and subdirectories on 127.0.0.1 port 8000
URL http://127.0.0.1:8000/
```

```
jshell> import com.sun.net.httpserver.SimpleFileServer;
...> import com.sun.net.httpserver.SimpleFileServer.OutputLevel;

jshell> var server = SimpleFileServer.createFileServer(new
InetSocketAddress(8000), Path.of("/Users/michaelinden"), OutputLevel.VERBOSE)
server ==> sun.net.httpserver.HttpServerImpl@6659c656

jshell> server.start()
```



Directory listing for /

- [Music/](#)
- [JavaDateAndTimeOverview_RO.astar](#)
- [spring-boot-test-workspace/](#)
- [JDK_SET.txt](#)
- [spacerocks.dat](#)
- [java8-workspace/](#)
- [h2](#)
- [spring-boot-persistence-data-workspace/](#)
- [Pictures/](#)
- [zshrc.textClipping](#)
- [JavaDateAndTimeOvervie_IFs.png](#)
- [Desktop/](#)
- [Library/](#)
- [eclipse-workspace/](#)
- [mongodb-macos-x86_64-5.0.3/](#)
- [spring-framework-intro-workspace/](#)
- [Sites/](#)
- [splash.png](#)
- [my.properties](#)
- [PycharmProjects/](#)



DEMO & Hands on



JEP 413: Code Snippets in Java API Documentation



JEP 413: Code Snippets in Java API Documentation



- Until now, there is no standard to include code fragments in the HTML documentation generated with Java Doc. In the context of this JEP, the inline tag `@snippet` is introduced. This allows code fragments to be included in a Java-Doc comment:

```
/**  
 * The following code shows how to use {@code Optional.isPresent}:  
 * {@snippet :  
 * if (optValue.isPresent())  
 * {  
 *     System.out.println("value: " + optValue.get());  
 * }  
 * }  
 */  
  
public static void method1(String[] args) {  
    //  
    //  
    //  
    //  
}  
}
```

 **void java18.misc.JavaDocExample.method1(String[] args)**

The following code shows how to use `Optional.isPresent`:

```
if (optValue.isPresent())  
{  
    System.out.println("value: " + optValue.get());  
}
```



- **Highlighting**

```
/**  
 * The following code shows how to use {@code Optional.isPresent} and  
 * {@code Optional.get} in combination  
 *  
 * {@snippet :  
 * if (optValue.isPresent()) // @highlight substring="isPresent"  
 * {  
 *     System.out.println("value: " + optValue.get()); // @highlight substring="get"  
 * } }  
 */  
  
public static void newJavaDocExample(String[] args) {  
    //  
    //  
    //  
    //  
}  
}
```

 **void java18.misc.JavaDocExample.newJavaDocExample(String[] args)**

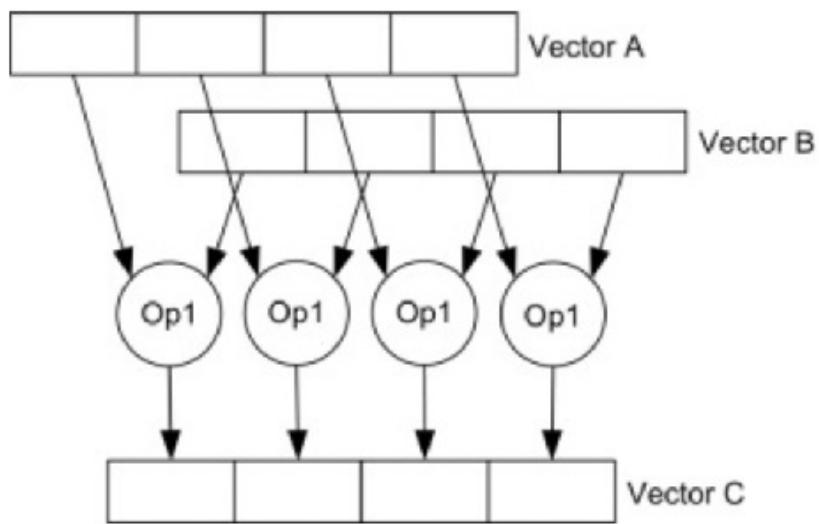
The following code shows how to use `Optional.isPresent` and `Optional.get` in combination

```
if (optValue.isPresent())  
{  
    System.out.println("value: " + optValue.get());  
}
```

Parameters:
args

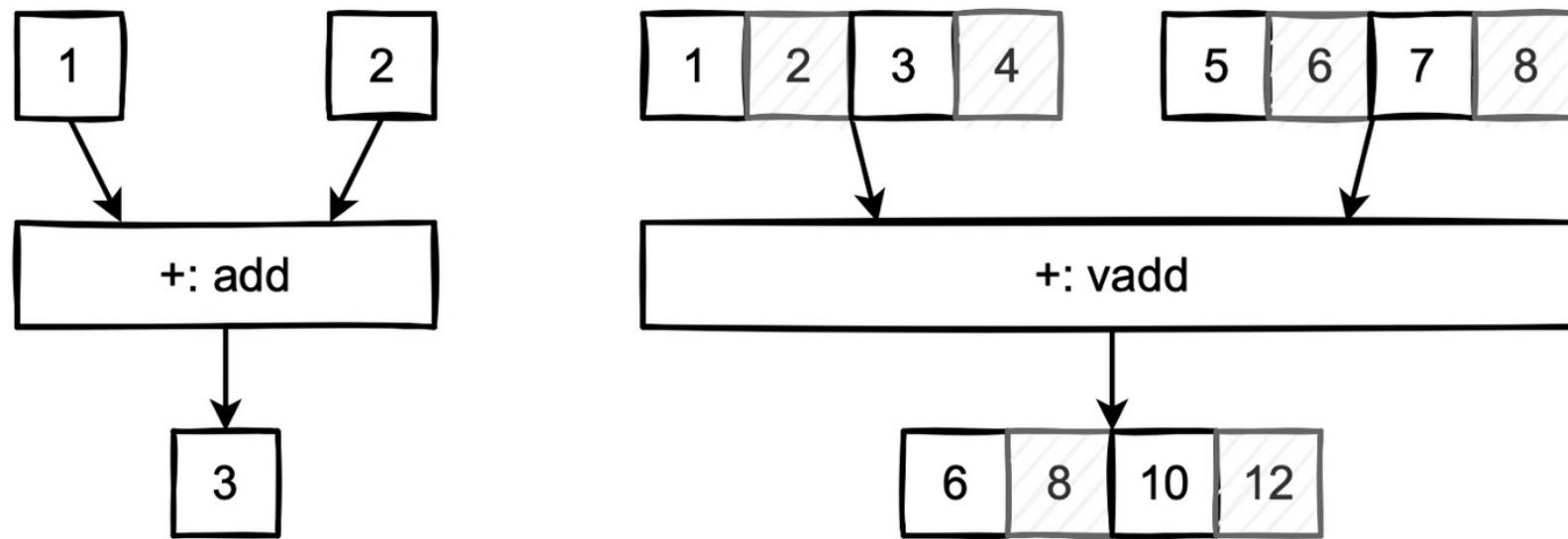


JEP 417: Vector API





- This JEP has nothing to do with the class `java.util.Vector`! Rather, it is about a platform-independent support of so-called vector calculations.
- Modern processors are able, for example, to perform an addition or multiplication not only for two values, but for a large number of values. This is also known as **Single Instruction Multiple Data (SIMD)**. The following graphic visualizes the basic principle:





- The Vector API aims to improve the performance of vector calculations.
- A vector calculation consists of a sequence of operations with vectors. You can think of a vector as an array of primitive values.
- If you wanted to combine two vectors or arrays with a mathematical operation, you would use a loop over all values.

```
int[] a = { 1, 2, 3, 4, 5, 6, 7, 8 };
int[] b = { 1, 2, 3, 4, 5, 6, 7, 8 };

var c = new int[a.length];
for (int i = 0; i < a.length; i++)
{
    c[i] = a[i] + b[i];
}
```



- With the Vector API it is possible to exploit the special features and optimizations in modern processors. Therefore one provides certain help for the computations.

```
int[] a = { 1, 2, 3, 4, 5, 6, 7, 8 };
int[] b = { 1, 2, 3, 4, 5, 6, 7, 8 };
```

```
var c = new int[a.length];
var vectorA = IntVector.fromArray(IntVector.SPECIES_256, a, 0);
var vectorB = IntVector.fromArray(IntVector.SPECIES_256, b, 0);
var vectorC = vectorA.add(vectorB);
// var vectorC = vectorA.mul(vectorB);
vectorC.intoArray(c, 0);
```

- The controlling factor here is the size of the vector, which we set to 256 bits by the argument `IntVector.SPECIES_256`.
- After that the appropriate action here can be `add()` or `mul()` or others.



- Let's consider the example from JEP 417 and the scalar calculation as a starting point:

```
void scalarComputation(float[] a, float[] b, float[] c)
{
    for (int i = 0; i < a.length; i++)
    {
        c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
    }
}
```

- The algorithm is absolutely comprehensible and easy to follow.

JEP 417: Vector API



```
static final VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;

void vectorComputation(float[] a, float[] b, float[] c)
{
    int i = 0;
    int upperBound = SPECIES.loopBound(a.length);
    for (; i < upperBound; i += SPECIES.length())
    {
        var va = FloatVector.fromArray(SPECIES, a, i);
        var vb = FloatVector.fromArray(SPECIES, b, i);
        var vc = va.mul(va)
                  .add(vb.mul(vb))
                  .neg();
        vc.intoArray(c, i);
    }

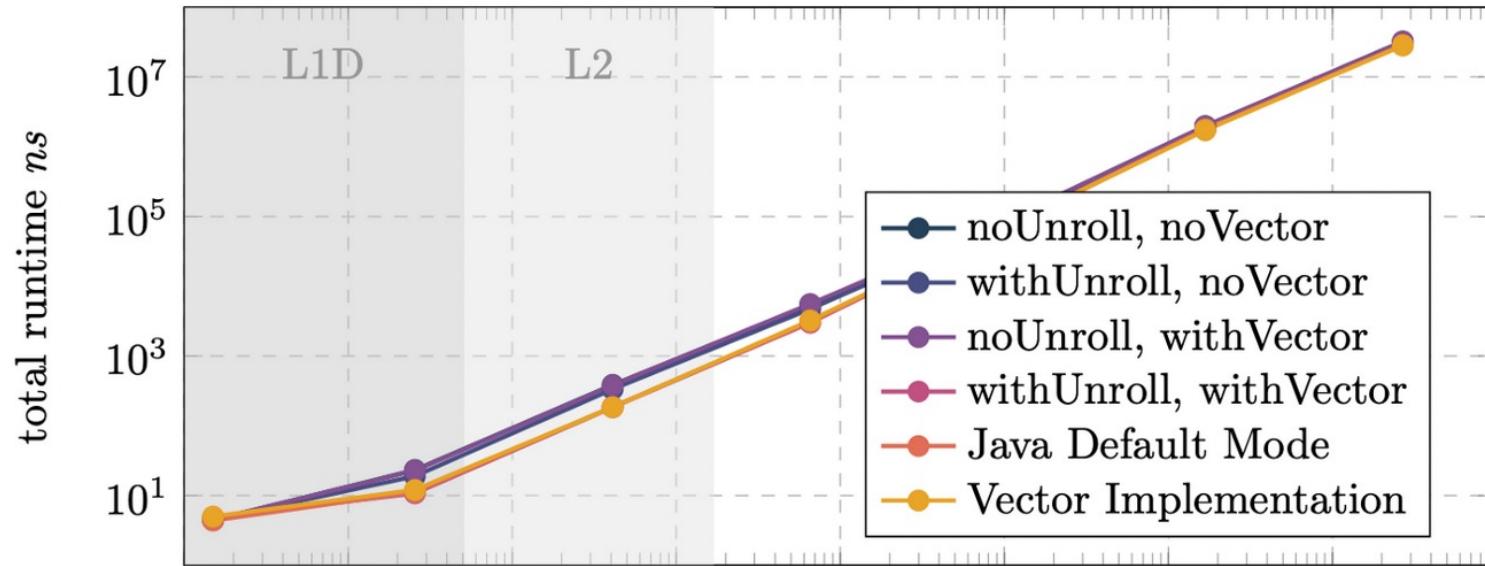
    for (; i < a.length; i++)
    {
        c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
    }
}
```

- If the initial data does not fit perfectly and is larger, the actions must be divided to fit.

JEP 417: Vector API Benchmarking



Benchmark: $c[n] = a[n] + b[n]$



Method	Peak Speed-Up
No Unroll, No Vector	1.00x
With Unroll, No Vector	1.22x
No Unroll, With Vector	1.01x
With Unroll, With Vector	2.15x
Java Default	2.06x
Vector Implementation	2.08x



JEP 418: Internet-Address Resolution SPI



JEP 418: Internet-Address Resolution SPI



- As part of this JEP, an SPI (Service Provider Interface) was implemented for host and name address resolution.
- The address resolution so far uses a hardcoded implementation within the class `java.net.InetAddress`.

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

public class InetAddressExmaple
{
    public static void main(String[] args) throws UnknownHostException
    {
        InetAddress[] addresses = InetAddress.getAllByName("www.dpunkt.de");
        System.out.println("addresses = " + Arrays.toString(addresses));
    }
}
```

JEP 418: Internet-Address Resolution SPI



```
public class OwnInetAddressResolver implements InetAddressResolver
{
    @Override
    public Stream<InetAddress> lookupByName(String host, LookupPolicy lookupPolicy)
        throws UnknownHostException
    {
        if (host.equals("www.dpunkt.de"))
            return Stream.of(InetAddress.getByAddress(new byte[] { 127, 0, 0, 1 }));

        throw new UnsupportedOperationException();
    }

    @Override
    public String lookupByAddress(byte[] addr)
    {
        throw new UnsupportedOperationException();
    }
}
```

JEP 418: Internet-Address Resolution SPI



```
import java.net.spi.InetAddressResolver;
import java.net.spi.InetAddressResolverProvider;

public class OwnInetAddressResolverProvider extends InetAddressResolverProvider
{
    @Override
    public InetAddressResolver get(Configuration configuration) {
        return new OwnInetAddressResolver();
    }

    @Override
    public String name() {
        return "Own Internet Address Resolver Provider";
    }
}
```





DEMO & Hands on



JEP 420: Pattern Matching for switch (PREVIEW II)



JEP 420: Pattern Matching for switch



- JEP 420 leads to two changes in the evaluation of the cases within switch during compilation:
 - on the one hand the so-called dominance check changes,
 - on the other hand the completeness analysis was corrected.
- However, the whole thing is again implemented in the form of a preview feature and to follow up you need to activate them appropriately, for example as follows in the JShell:

```
michaelinden@MBP-von-Michael ~ % jshell --enable-preview
| Welcome to JShell -- Version 18.0.1
| For an introduction type: /help intro
```

JEP 420: Pattern Matching for switch: Dominance check



- Several patterns can match one input. The one that matches "best" is called the dominating pattern. In the example, the shorter pattern `String s` dominates the longer one specified before it.

```
public static void main(String[] args) {
    multiMatch("Python");
    multiMatch(null);
}

static void multiMatch(Object obj) {
    switch (obj) {
        case null -> System.out.println("null");
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());
        case String s
            -> System.out.println(s.toLowerCase());
        case Integer i
            -> System.out.println(i * i);
        default -> {}
    }
}
```

JEP 420: Pattern Matching for switch: Dominance check



- Several patterns can match one input. The one that matches "best" is called the dominating pattern. In the example, the shorter pattern `String s` dominates the longer one specified before it.

```
static void dominaceExample(Object obj) {  
    switch (obj) {  
        case null -> System.out.println("null");  
        case String s  
        case String s && s.length() > 5 -> System.out.println(s.toLowerCase());  
        case Integer i  
        default -> {}  
    }  
}
```

A screenshot of an IDE showing a Java code editor. The code above is highlighted. A tooltip window is open over the line `case String s && s.length() > 5 ->`. The tooltip contains the message: "✖ This case label is dominated by one of the preceding case label" and "Press 'F2' for focus".

- With Java 17 is was not detected as error!

JEP 420: Pattern Matching for switch: Dominance check



- Problems with constants (was not detected with Java 17)

```
static void dominaceExampleWithConstant(Object obj) {  
    switch (obj.toString()) {  
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());  
        case "Sophie" -> System.out.println("My lovely daughter");  
        default -> System.out.println("FALLBACK");  
    }  
}
```

A screenshot of an IDE showing a Java code editor. The code is identical to the one above, but the second case statement 'case "Sophie"' has a tooltip overlaid on it. The tooltip contains the text 'This case label is dominated by one of the preceding case label' with a red 'X' icon, and 'Press 'F2' for focus' at the bottom right.

- Correction:

```
static void dominaceExampleWithConstant(Object obj) {  
    switch (obj.toString()) {  
        case "Sophie" -> System.out.println("My lovely daughter");  
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());  
        default -> System.out.println("FALLBACK");  
    }  
}
```

JEP 420: Pattern Matching for switch: Completeness analysis



- bug fix in the area of completeness analysis = checking if all possible paths are covered by the cases in the switch
- sometimes wrong error message "switch statement does not cover all possible input values"

```
static void performAction(BaseOp op) {  
    switch (op) {  
        case Add a -> System.out.println(a);  
        case Sub s -> System.out.println(s);  
        // default -> System.out.println("FALLBACK")  
    }  
}
```

```
static void performAction(BaseOp op) {  
    switch (op) {  
        case  
        case  
        // de  
    }  
}
```

'switch' statement does not cover all possible input values
Insert 'default' branch ↗↔ More actions... ↗↔
@NotNull ↗
SwitchPreviewExample.BaseOp op
main



- bug fix in the area of completeness analysis = checking if all possible paths are covered by the cases in the switch

```
static sealed abstract class BaseOp permits Add, Sub {  
}  
  
static final class Add extends BaseOp {  
}  
  
static final class Sub extends BaseOp {  
}  
  
static void performAction(BaseOp op) {  
    switch (op) {  
        case Add a -> System.out.println(a);  
        case Sub s -> System.out.println(s);  
    }  
}
```



DEMO & Hands on



JEP 421: Deprecate Finalization for Removal



JEP 421: Deprecate Finalization for Removal



- Originally, finalization was intended to serve as a safety net and enable cleanup operations or releasing of resources when an object dies. For this purpose, the respective finalizer is called during garbage collection in the form of a `finalize()` method before the object's memory is returned.
- However, there are various hurdles and difficulties in doing so:
 - not defined when exactly the call happens (possibly some time after the object is no longer referenced).
 - the object can be put back into an active state, quasi revived.
- In the context of this JEP, the finalization is considered deprecated as well as `forRemoval`. Although the finalization remains active by default, it can now be deactivated.
- The background of this JEP is explained in detail by Nicolai Parlog online at <https://www.youtube.com/watch?v=eDgBnjOid-g>



Other Deprecations



Other Deprecations



- As a result of JEP 421, all `finalize()` methods have already been deprecated.
 - In the `Runtime` class, various overloaded `exec()` methods have been deprecated. Likewise the method `runFinalization()`.
 - In the class `Thread` the method `stop()` is now not only deprecated, but `forRemoval=true`.
-



PART 7: News in Java 19



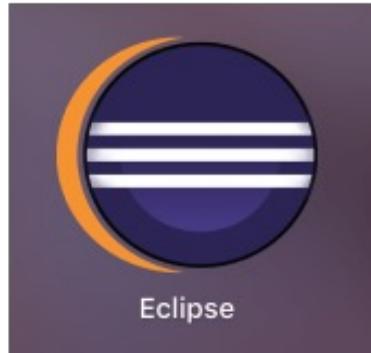
Build-Tools und IDEs



IDE & Tool Support für Java 19 (zukünftig)



- Still some problems remaining
- Eclipse: Version 2022-09 with additional plugin
- IntelliJ: Version 2022.2.1
- Maven: 3.8.6, Compiler plugin: 3.10
- Gradle: 7.5.1 => 7.6!
- Activation of preview features necessary
 - In dialogs
 - In build scripts



Maven™

 **Gradle**

The Gradle logo consists of a stylized blue elephant icon followed by the word "Gradle" in a bold, sans-serif font.

IDE & Tool Support



- Activation of preview features required

Project Structure

Project
Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: InstanceofRecordMatching.java

SDK: 19 java version "19" A red arrow points to this button.

Language level: 19 (Preview) - Record patterns, pattern matching for switch (third preview)

Compiler output: ~/java19/out

Used for modules' subdirectories, Production and Test directories for the corresponding sources.

Cancel Apply OK

Project Settings

- Project
- Modules
- Libraries
- Facets
- Artifacts

Platform Settings

- SDKs
- Global Libraries

Problems





- Activation of preview features required

```
sourceCompatibility=19  
targetCompatibility=19
```

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



> **BUG!** exception in phase 'semantic analysis' in source unit '_BuildScript_'
Unsupported class file major version 63

Gradle Release Notes

The Gradle team is excited to announce Gradle 7.6-20220924231105+0000.

This release includes [building and running code with Java 19](#), a flag to [rerun tasks individually](#), and a new [strongly-typed dependencies block](#) for JVM test suites.

IDE & Tool Support



- Activation of preview features required

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.10.0</version>
  <configuration>
    <source>19</source>
    <target>19</target>
    <compilerArgs>
      <arg>--enable-preview</arg>
      <arg>--add-modules</arg>
      <arg>jdk.incubator.concurrent</arg>
    </compilerArgs>
  </configuration>
</plugin>
```



```
    <release>19</release>
```



Java 19 JEPs

Java 19 – What is included?



- **JEP 405: Record Patterns (Preview)**
 - JEP 422: Linux/RISC-V Port
 - JEP 424: Foreign Function & Memory API (Preview)
 - **JEP 425: Virtual Threads (Preview)**
 - JEP 426: Vector API (Fourth Incubator)
 - **JEP 427: Pattern Matching for switch (Third Preview)**
 - **JEP 428: Structured Concurrency (Incubator)**
-



JEP 405: Record Patterns (Preview)



JEP 420: Record Patterns



- This JEP extends the pattern matching for instanceof from Java 16:

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

static void printCoordinateInfo(Object obj)
{
    if (obj instanceof Point point)
    {
        int x = point.x();
        int y = point.y();

        System.out.println("x: %d y: %d, sum: %d".formatted(x, y, x + y));
    }
}
```

- The goal is to be able to decompose records into their components and access them.
-

JEP 420: Pattern Matching for switch



- The goal is to be able to decompose records into their components and access them.

```
static void printCoordinateInfo(Object obj)
{
    if (obj instanceof Point point)
    {
        int x = point.x();
        int y = point.y();
        System.out.println("x: %d y: %d, sum: %d".formatted(x, y, x + y));
    }
}
```

- =>

```
static void printCoordinateInfoNew(Object obj)
{
    if (obj instanceof Point(int x, int y))
    {
        System.out.println("x: %d y: %d, sum: %d".formatted(x, y, x + y));
    }
}
```

JEP 420: Pattern Matching for switch



- Record patterns can be nested to provide a declarative, powerful, and combinable form of data navigation and processing.

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

```
enum Color { RED, GREEN, BLUE }
```

```
record ColoredPoint(Point p, Color c) {}
```

```
record Rectangle(ColoredPoint upperLeft, ColoredPoint lowerRight) {}
```

```
static void printColorOfUpperLeftPoint(Rectangle rect)
```

```
{  
    if (rect instanceof Rectangle(ColoredPoint(Point p, Color c), ColoredPoint lr))  
    {  
        System.out.println(c);  
    }  
}
```



DEMO & Hands on

InstanceOfRecordMatching.java

InstanceOfRecordMatchingAdvanced.java



JEP 425: Virtual Threads (Preview)



JEP 425: Virtual Threads



- This preview feature introduces the concept of **lightweight virtual threads** that do not map directly to operating system threads.
- Virtual threads allow server programming to once again work with one separate thread per request and more lightly support an asynchronous programming style.
- Even better: Existing code that uses the previous thread API can be converted to virtual threads with minimal changes.
- Factory methods such as `newVirtualThreadPerTaskExecutor()` can be used to choose whether to use virtual threads or platform threads (e.g. with `Executors.newCachedThreadPool()`).

JEP 425: Virtual Threads



```
public static void main(String[] args)
{
    System.out.println("Start");

    try (var executor = Executors.newVirtualThreadPerTaskExecutor())
    {
        for (int i = 0; i < 10_000; i++)
        {
            final int pos = i;
            executor.submit(() -> {
                Thread.sleep(Duration.ofSeconds(1));
                return pos;
            });
        }
    }
    // executor.close() is called implicitly, and waits
    System.out.println("End");
}
```



DEMO & Hands on

VirtualThreads.java



JEP 427: Pattern Matching for switch (Preview III)



JEP 427: Pattern Matching for switch



- This JEP already has two predecessors that brought significant improvements to switch.
- This JEP is about a few refinements, namely the way to specify further queries, so-called guarded patterns, in switch.
- So far intuitively and as known from if with &&:

```
case String str && str.startsWith("INFO") -> System.out.println("just an info");
```
- Now the key word when has been newly introduced for this purpose.

```
case String str when str.startsWith("INFO") -> System.out.println("just an info");
```
- A bit special syntax:

```
case String str when str.startsWith("INFO") && str.contains("SPECIAL") ->  
    System.out.println("a very special info");
```

JEP 427: Pattern Matching for switch



```
interface Shape {}

record Rectangle() implements Shape {}

record Triangle() implements Shape { int calculateArea() { return 7271; } }

static void testTriangleAndString(Object obj)
{
    switch (obj)
    {
        case Triangle t when t.calculateArea() > 100 -> System.out.println("Large triangle");
        case String str when str.startsWith("INFO") -> System.out.println("just an info");
        default -> System.out.println("Something else: " + obj);
    }
}
```



DEMO & Hands on

SwitchWhen.java



JEP 428: Structured Concurrency (Incubator)





- This JEP brings Structured Concurrency as a simplification of multithreading.
- Here, different tasks that are executed in multiple threads are considered as a single unit. This improves reliability, reduces the risk for errors and simplifies their handling.
- To illustrate the functionality and advantages, let's assume the following actions for handling a request, which is later to determine the matching user along with the associated order in parallel based on a user id:

```
static Response handleSynchronously(Long userId)
{
    String user = findUser(userId);
    Integer order = fetchOrder(userId);

    return new Response(user, order);
}
```



- **Traditional implementation with ExecutorService:**

```
static Response handleOldStyle(Long userId) throws ExecutionException, InterruptedException
{
    var executorService = Executors.newCachedThreadPool();

    var userNameFuture = executorService.submit(() -> findUser(userId));
    var orderNoFuture = executorService.submit(() -> fetchOrder(userId));

    var user = userNameFuture.get(); // Join findUser
    var order = orderNoFuture.get(); // Join fetchOrder

    return new Response(user, order);
}
```

- Because the subtasks are executed in parallel, they can succeed or fail independently. Then the handling can become quite complicated.
- Often, for example, one does not want the second get() to be called if an exception has already occurred during the processing of the findUser() method.



- **Implementation of Structurd Concurrency with class StructuredTaskScope:**

```
static Response handle(Long userId) throws ExecutionException, InterruptedException
{
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        var userNameFuture = scope.fork(() -> findUser(userId));
        var orderNoFuture = scope.fork(() -> fetchOrder(userId));

        scope.join();      // Join both forks
        scope.throwIfFailed(); // ... and propagate errors

        // Here, both forks have succeeded, so compose their results
        return new Response(userNameFuture.resultNow(), orderNoFuture.resultNow());
    }
}
```

- **With structured concurrency, one splits off competing subtasks with fork().**
- **The results are collected again with a blocking call to join(), which waits until all subtasks are processed or an error occurred. Afterwards one should handle all errors that occurred and otherwise process their results.**

JEP 428: Structured Concurrency



Run/Debug Configurations

Name: StructuredConcurrency Store as project file

Build and run

java 19 SDK of 'Java19Exam' --add-modules jdk.incubator.concurrent

StructuredConcurrency

Program arguments

Press ⌘ for field hints

Working directory: _JAVA-BEST-OF/2022/ch.open_Java_11_bis_19_WORKSHOP_2022/Java19Exam

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started

Edit configuration templates...

?

Cancel Apply OK

A screenshot of the "Run/Debug Configurations" dialog in an IDE. The configuration is named "StructuredConcurrency". In the "Build and run" section, the command is set to "java 19 SDK of 'Java19Exam'" followed by "--add-modules jdk.incubator.concurrent". A red arrow points to the "Modify options" button next to the command line. The "Working directory" is set to "_JAVA-BEST-OF/2022/ch.open_Java_11_bis_19_WORKSHOP_2022/Java19Exam". The "Environment variables" field is empty. At the bottom, there is a checkbox for "Open run/debug tool window when started". The footer of the dialog includes standard buttons: a question mark icon, "Cancel", "Apply", and "OK".



DEMO & Hands on

StructuredConcurrency.java



Conclusion

Positive things



- Some parts of project coin (syntax)
- Switch / Records
- A lot of practical extensions in the APIs
- JPackage
- HTTP 2 (Java 11)
- Modularization with Project JIGSAW -- but until now a lot of room for improvement concerning tools



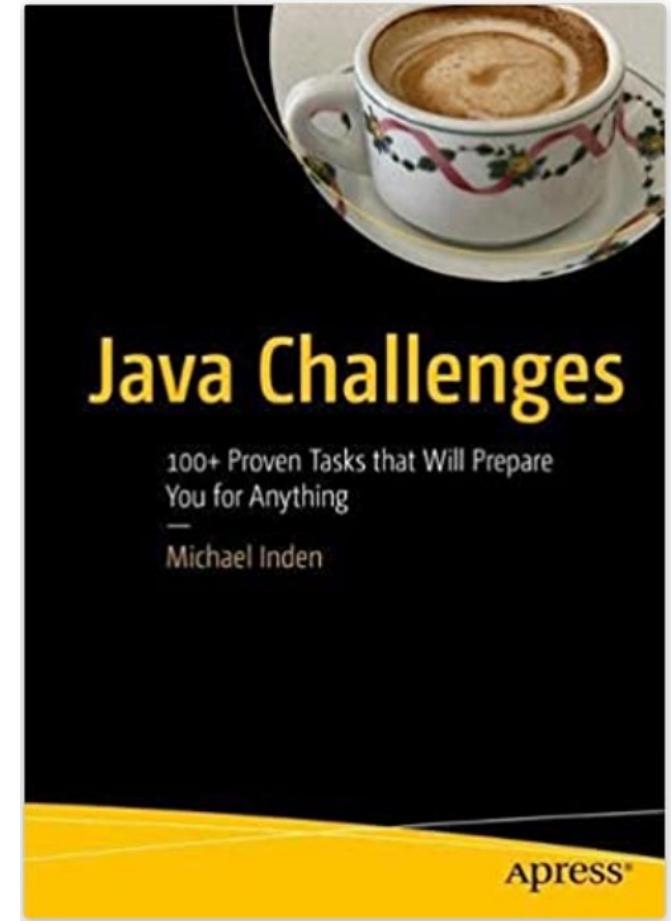
Negative things



- **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**



Help





Questions?



Thank You