Switch Expressions is the second language enhancement from Project Amber to reach production status (the first was "var" in Java 10). Switch expressions allow a much more concise notation than before using arrow notation:

```
switch (day) {
  case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);
  case TUESDAY                -> System.out.println(7);
  case THURSDAY, SATURDAY     -> System.out.println(8);
  case WEDNESDAY              -> System.out.println(9);
}
```

As an expression, switch can also return a value:

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

To find out what other possibilities switch expressions offer, e.g. how to return a value from code blocks with `yield`, when default cases are necessary and when they are not, and how the compiler can perform a completeness analysis on enums, see the main article on switch expressions.

(Switch Expressions were first introduced as a preview feature in Java 12. In the second preview in Java 13, the keyword `break`, used initially to return values, was replaced by `yield`. Due to positive feedback, Switch Expressions were released as a final feature in Java 14 by JDK Enhancement Proposal 361 without further changes.)

Helpful NullPointerExceptions

We all know the following problem: Our code throws a `NullPointerException`:

```
Exception in thread "main" java.lang.NullPointerException
    at eu.happycoders.BusinessLogic.calculate(BusinessLogic.java:80)
```

And in the code, we find something like this:

```
long value = context.getService().getContainer().getMap().getValue();
```

What is `null` now?

- `context`?
- `context.getService()`?
- `Service.getContainer()`?
- `Container.getMap()`?
- `Map.getValue()`? (in case this method returns a `Long` object)

To fix the error, we have the following options:

- We could analyze the source code.
- We could debug the application (very costly if the error is difficult to reproduce or only occurs in production).
- We could split the code into multiple lines and rerun it (we would have to redeploy the application and wait for the error to occur again).

After upgrading to Java 14, you won't have to ask yourself this question anymore because then the error message will look like this, for example:

```
Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "Map.getValue()" because the return value of "Container.getMap()
    at eu.happycoders.BusinessLogic.calculate(BusinessLogic.java:80)
```

We can now see exactly where the exception occurred: `Container.getMap()` returned `null`, so `Map.getValue()` could not be called.

Something similar can happen when accessing arrays, as in the following line of code:

```
this.world[x][y][z] = value;
```

If a `NullPointerException` occurs in this line, it was previously not possible to tell whether `world`, `world[x]`, or `world[x][y]` was `null`. With Java 14, this is clear from the error message:

```
Exception in thread "main" java.lang.NullPointerException:
Cannot store to int array because "this.world[x][y]" is null
    at eu.happycoders.BusinessLogic.calculate(BusinessLogic.java:107)
```

In Java 14, "Helpful NullPointerExceptions" are disabled by default and must be enabled with `-XX:+ShowCodeDetailsInExceptionMessages`. In Java 15, this feature will be enabled by default.

(Helpful NullPointerExceptions are specified in JDK Enhancement Proposal 358.)

Experimental, Preview, and Incubator Features

The following features are not yet production-ready. They are shipped at a more or less advanced stage of development to be able to improve them based on feedback from the Java community.

The first three features (Records, Pattern Matching for instanceof, and Text Blocks) are being developed (like Switch Expressions) in Project Amber, whose goal is to make Java syntax more modern and concise.

I will not present the features in all details but only briefly outline each and refer to the Java version in which the features reach production readiness. I will then cover them in detail in the corresponding article of this series.