JAVA 19

Pattern matching for switch

carlos.carneiro.dev

Before we talk about ***pattern matching for switch***, let's understand first what is ***pattern matching***. **A little bit of history**:

Firstly, ***pattern matching*** was proposed for the ***instanceof*** operator. It was proposed the first time in may 2017 through the JEP 305 (JDK enhancement proposal) and **was released as a feature preview in Java 14**. After that there was some changes in the proposal in 2019 through the JEP 375 and it was **released again as a second preview in Java 15**. Lastly, yet again in 2019, there was some more changes in the idea through the JEP 394 and it was **finally released as a feature in the version 16 of Java**.

To be more clear, let's see an example BEFORE and AFTER the ***pattern matching for instanceof***:

Let's suppose we have **an abstract class** called **Animal**.

```java
abstract class Animal { }
```

Let's say we have **two subclasses** as well: a class **Dog** and a class **Cat**.

```java
class Dog extends Animal {
    void bark() {
        System.out.println("Woof woof");
    }
}
```

```java
class Cat extends Animal {
    void meow() {
        System.out.println("Meow meow");
    }
}
```

carlos.carneiro.dev

Then, we have **a method makeSomeNoise**, which receives **a parameter** of type **Animal**. In this method we need to check the type of the parameter. If it's of type **Dog**, we call **bark**, if it's of type **Cat**, we call meow.

Before the ***pattern matching for instanceof*** we would do something like this:

```
void makeSomeNoise(Animal animal) {
    if(animal instanceof Dog) {
        var dog = (Dog) animal;
        dog.bark();
    } else if (animal instanceof Cat) {
        var cat = (Cat) animal;
        cat.meow();
    } else {
        throw new RuntimeException("WTH is it???");
    }
}
```

Look at the amount of code. There's too much noise. This is ugly as hell.

Just to make the example complete let's see how we would call this method:

```
var charles = new Cat();
makeSomeNoise(charles);
// Result: Meow meow
// PS: Yes, I know, why don't
// we just charles.meow()?
// Because it's an example
```

Perfect, now let's see how the method **makeSomeNoise** would be **using *pattern matching for instanceof***:

```
void makeSomeNoise(Animal animal) {
    if(animal instanceof Dog dog) {
        dog.bark();
    } else if (animal instanceof Cat cat) {
        cat.meow();
    } else {
        throw new RuntimeException("WTH is it???");
    }
}
```

Can you spot the difference?

carlos.carneiro.dev

Using ***pattern matching for instanceof*** **we don't need the explicitly casting anymore**, because **it'll be cast implicitly in case the result of the *instanceof* is true**.

The reasoning behind ***pattern matching*** is that you would always cast after an ***instanceof*** check. Quoting the JEP text:

"What else would you do after an instanceof test?"

But okay, now you may say: **"Code still ugly as Smeagol!!!"**

Exactly, and that's where the *pattern matching for switch* comes in.

Its idea is the same as for the ***pattern matching for instanceof***. Without further ado, let's see how the method **makeSomeNoise** would be using this **feature preview of Java 19**:

```java
void makeSomeNoise(Animal animal) {
    switch(animal) {
        case Dog dog -> dog.bark();
        case Cat cat -> cat.meow();
        default -> throw new RuntimeException("WTH is it???");
    }
}
```

Look at this beauty. Using this feature, we don't need no explicitly casting nor that bunch of *if (instanceof...*

This kind of code is rather usual, and it will be possible to simplify it in the future using this feature, when it will no longer be a preview.

Would you fancy another example?

This next example **is a snippet from the code base of the company I work for.**

carlos.carneiro.dev

```java
if (jaxbElement.getValue() instanceof CustomerHistoryTransaction) {
    CustomerHistoryTransaction value = (CustomerHistoryTransaction)
    if (value instanceof CDRMAType) {
        CDRMAType cdrma = (CDRMAType) value;
        /*...*/
    }
    else if (value instanceof AdjustmentMAType) {
        AdjustmentMAType adjustment = (AdjustmentMAType) value;
        /*...*/
    }
    else if (value instanceof AIRSRefillMAType) {
        AIRSRefillMAType refillMAType = (AIRSRefillMAType) value;
        /*...*/
    }
}
```

This piece of Java 8 code **can be improved in the future using the *pattern matching for switch* feature**. And not only this one, but many others as well.

Just a reminder...

***Pattern matching for switch* is still a preview, this means it cannot be used in production code**.

carlos.carneiro.dev

This feature is in its third preview. It first came out as a preview in Java 17, then again in Java 18 and now yet again in Java 19.

I did not dig this too deep, but **I believe this feature will no longer be a preview by October next week**, in other words, in Java 21, which will be the next LTS version.

There were a lot of other things we could talk about this feature, but I think for a little introduction this is okay for now.

So guys, did you know about this feature? What do you think about it? Useful? Not so much? Let me know what you think.

For any other question, you can reach out to me on instagram or at my email: **contato@carlos.carneiro.nom.br**

Thank you very much for reading!

 **carlos.carneiro.dev**