Java 14 update

Spis treści

Java 14 update . . . 1
- switch . . . 1
- NullPointerException . . . 4
- Pattern Matching instanceof (preview) . . . 6
- Records (preview) . . . 6
- Text Blocks (preview) . . . 6
- Podsumowanie . . . 6

Java 14 update

Java 14 została wydana w marcu 2020 i jest wersją **non-LTS**. Poniżej omówimy niektóre funkcjonalności udostępnione w tym wydaniu. Przy aktualizacji wersji Javy często poprawianych jest o wiele więcej funkcjonalności i dodawanych o wiele więcej klas lub metod niż te, które wymieniamy tutaj. W obrębie tych materiałów poruszamy tylko te kwestie, które są adekwatne do naszego poziomu zaawansowania jako Java developerów.

switch

Nowa wersja wyrażenia `switch` została zaproponowana w wersji 12, następnie w wersji 13 zostały dołożone kolejne funkcjonalności. Java w wersji 14 dodała nowy `switch` na stałe. Możemy zatem przejść do jego omówienia.

Nowa składnia wyróżnia się następującymi cechami:

- Eliminuje potrzebę stosowania instrukcji `break`.
- Instrukcja `yield` może być używana do wyjścia z jednego `case` na kilka sposobów. Początkowo planowano, żeby wyjście było realizowane przy wykorzystaniu słówka `break`, jednakże finalnie została podjęta decyzja o wprowadzeniu słówka `yield`.
- Możemy zdefiniować wiele stałych przy tym samym `case`.
- Wyrażenie `default` jest obowiązkowe.
- Nie popełnimy błędu polegającego na tym, że `switch` będzie się dalej wykonywał, jeżeli zapomnimy napisać `break`.

Przykład "starego" `switch`:

```
public class Example {

    public static void main(String[] args) {
        oldSwitch("A");
    }
```

```
    private static void oldSwitch(String input) {
        int result = 0;
        switch (input) {
            case "A":
            case "B":
            case "C":
                result = 1;
                break; ①
            case "D":
            case "E":
            case "F":
                result = 2;
                break;
        }

        System.out.println("Old switch: " + result);
    }
}
```

① Jeżeli zapomnimy postawić tutaj `break`, to na ekranie wydrukuje się *2*, a powinno wydrukować się *1*.

Poniżej "nowego" `switch`. Strzałka wygląda jak wyrażenie lambda, natomiast nie implementujemy tutaj interfejsu funkcyjnego. Przykład:

```
public class Example {

    public static void main(String[] args) {
        newSwitch("A");
    }

    private static void newSwitch(String input) {
        int result = switch (input) { ①
            case "A", "B", "C" -> 1; ②
            case "D", "E", "F" -> 2;
            default -> 0; ③
        };

        System.out.println("New switch: " + result);
    }
}
```

① Wynik działania `switch` możemy teraz przypisać do zmiennej.

② Możemy zdefiniować wiele stałych przy tym samym `case`.

③ Wyrażenie `default` jest obowiązkowe.

Kolejny przykład, tym razem z wykorzystaniem słówka `yield`:

```
public class Example {

    public static void main(String[] args) {
        newSwitch("A");
    }

    private static void newSwitch(String input) {
        int result = switch (input) { ①
```

```
            case "A", "B", "C": ②
                yield 1; ③
            case "D", "E", "F":
                yield 2;
            default:
                yield 0; ④
        }; ④

        System.out.println("Yield switch: " + result);
    }
}
```

① Wynik działania `switch` możemy ponownie przypisać do zmiennej.

② Możemy zdefiniować wiele stałych przy tym samym `case`.

③ Zostało wprowadzone nowe słówko kluczowe `yield`, które służy do zwrócenia wartości z danej gałęzi `switch`. Możemy je rozumieć analogicznie do `return`, ale nie wychodzimy z metody tylko ze `swticha`. Możemy również napisać warunek i `yield` w jednej linijce.

④ Wyrażenie `default` jest obowiązkowe.

Poniżej znajdziesz przykład `yield` jednolinijkowego:

```
private static void newSwitch(String input) {
    int result = switch (input) {
        case "A", "B", "C": yield 1;
        case "D", "E", "F": yield 2;
        default: yield 0;
    };

    System.out.println("Yield switch: " + result);
}
```

Jak można ze sobą połączyć → i `yield`:

```
private static void newSwitch(String input) {
    int result = switch (input) {
        case "A", "B", "C" -> 1;
        case "D", "E", "F" -> 2;
        default -> { ①
            if (input.length() < 10) {
                yield 10;
            } else {
                yield 20;
            }
        }
    };

    System.out.println("Yield switch: " + result);
}
```

① Nawiasy klamrowe mogą być również użyte przy innych `case`, żeby napisać wielolinijkowy rezultat dopasowania `case`.

Nie można natomiast ze sobą łączyć `yield` i →:

```
private static void newSwitch(String input) {
    int result = switch (input) {
        case "A", "B", "C": yield 1;
        case "D", "E", "F": yield 2;
        default -> { // błąd kompilacji
            if (input.length() < 10) {
                yield 10;
            } else {
                yield 20;
            }
        }
    };

    System.out.println("Yield switch: " + result);
}
```

Warto jest tutaj również zaznaczyć, że jeżeli obsłużymy wszystkie możliwe przypadki, nie jest konieczne dodawanie klauzuli `default`. Wyżej `switch` operował na `String`, w takim przypadku nie jest możliwe pokrycie wszystkich możliwych przypadków. Natomiast inaczej to już wygląda w przypadku `enum`. Spójrz na poniższy przykład:

```
public class SwitchNoDefault {

    public static void main(String[] args) {
        Operation operation = Operation.READ;
        newSwitchNoDefault(operation);
    }

    private static void newSwitchNoDefault(Operation operation) {
        int result1 = switch (operation) {
            case READ, UPDATE: yield 1;
            case DELETE, SAVE: yield 2;
        };
        System.out.println("Result1: " + result1);

        int result2 = switch (operation) {
            case READ, UPDATE -> 1;
            case DELETE, SAVE -> 2;
        };
        System.out.println("Result2: " + result2);
    }

    private enum Operation {
        READ,
        UPDATE,
        DELETE,
        SAVE
    }
}
```

NullPointerException

`NullPointerException` (w skrócie **NPE**) potrafią być w praktyce ciężkie do zrozumienia, gdy czytamy logi aplikacji. Szczególnie w sytuacji, gdy w logach był zapisany taki `Stacktrace`:

```
Exception in thread "main" java.lang.NullPointerException
    at Example.main(Example.java:12)
```

W takiej sytuacji w pierwszej kolejności sprawdzasz linijkę 12 w klasie `Example` i widzisz taki kod:

```
String result = input.getData().toString()
```

Nie wiadomo w takiej sytuacji, czy to `input` jest `null`, czy rezultat `getData()` zwrócił `null`. W Javie 14 został wprowadzony bardziej przejrzysty sposób drukowania `Stacktrace` w momencie, gdy zostanie wyrzucony `NullPointerException`.

Uruchom teraz poniższy kod i zobacz, co zostanie wydrukowane na ekranie:

```
public class Example {

    public static void main(String[] args) {
        String name = new Owner().getDog().getName();
    }
}
```

Klasa Owner

```
class Owner {
    private Dog dog;

    public Dog getDog() {
        return dog;
    }
}
```

Klasa Dog

```
class Dog {
    private String name;

    public String getName() {
        return name;
    }
}
```

Po uruchomieniu na ekranie zostanie wydrukowana wiadomość:

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "Dog.getName()" because the
return value of "Owner.getDog()" is null
    at Example.main(Example.java:5)
```

Pattern Matching instanceof (preview)

Java 14 przedstawiła również nową koncepcję zapisu wyrażenia `instanceof`, natomiast również zostało to wprowadzone jako **preview**. Finalna wersja tej zmiany została dodana w Javie 16, dlatego tam przejdziemy do jej omówienia.

Records (preview)

W Javie często tworzymy klasy, które nie realizują logiki, nie wykonują obliczeń, służą tylko do przetrzymywania informacji. Klasy takie są bardzo powtarzalne, posiadają pola, gettery, settery, konstruktory, nadpisują metody `equals()` i `hashCode()` oraz `toString()`. W java 14 jako **preview feature** wprowadzono rekordy (*records*), które mają uprościć tworzenie takich struktur. Rekord w uproszczeniu można rozumieć jak taką uproszczoną klasę, która skraca zapis w sytuacji opisanej powyżej. Z innej strony możemy na nie spojrzeć jak na `enum`, który też jest specyficznym rodzajem klasy. Z racji, że w Java 14 rekordy są **preview feature**, wrócimy do nich w Java 16, gdzie zostały wprowadzone na stałe.

Text Blocks (preview)

Java 14 wprowadziła pewne modyfikacje do **Text Blocks** i finalnie ta funkcjonalność została udostępniona na stałe. Dlatego też przejdziemy do jej omówienia w trakcie omawiania zmian w Java 15.

Podsumowanie

Mam nadzieję, że widzisz już na tym etapie, że pewne funkcjonalności są wprowadzane do języka jako **preview features**. Przypomnę, że oznacza to, że mogą się one przyjąć albo i nie. Programiści mogą takie funkcjonalności przetestować i zgłosić do nich feedback. Finalnie mogą one zostać wprowadzone w przyszłych wersjach w obecnej formie lub zmienionej. Mogą też zostać kompletnie usunięte i trzeba o tym pamiętać. Dlatego właśnie nie skupiamy się mocno na tłumaczeniu **preview features**. Funkcjonalności takie są przez nas wyjaśniane dopiero, jak zostają wprowadzone do języka na stałe.

Przypomnę, że przy aktualizacji wersji Javy często poprawianych jest o wiele więcej funkcjonalności i dodawanych o wiele więcej klas lub metod niż te, które wymieniamy tutaj. Z kolejnymi wersjami wprowadzane są również rozmaite poprawki lub usprawnienia w samym działaniu JVM albo przykładowo Garbage Collectora (w tym przypadku mogą to być, chociażby różne algorytmy, o których działanie oparty jest GC). Zmianom mogą ulegać również kwestie dotyczące zarządzania pamięcią. Oprócz tego kolejne wersje Javy mogą również wprowadzać dodatkowe narzędzia, które programista może wykorzystywać w swojej pracy. Do tego poprawkom mogą podlegać istniejące implementacje metod. W obrębie tych materiałów poruszamy tylko te kwestie, które są adekwatne do naszego poziomu zaawansowania jako Java developerów. Nie poruszamy też zagadnień, co do których twórcy Zajavki uznali, że z naszego punktu widzenia zmiany te nie są aż tak istotne i lepiej poświęcić ten sam czas na skupienie się na dalszych zagadnieniach.

Jeżeli natomiast interesuje Cię, jakie jeszcze zmiany są wprowadzane z każdą wersją — wystarczy, że wpiszesz w Google np. "Java 14 features" i znajdziesz dużo artykułów opisujących wprowadzone zmiany. Możesz również zerknąć na tę stronę JDK 14. Zaznaczam jednak, że wiele funkcjonalności będzie niezrozumiałych. 😉