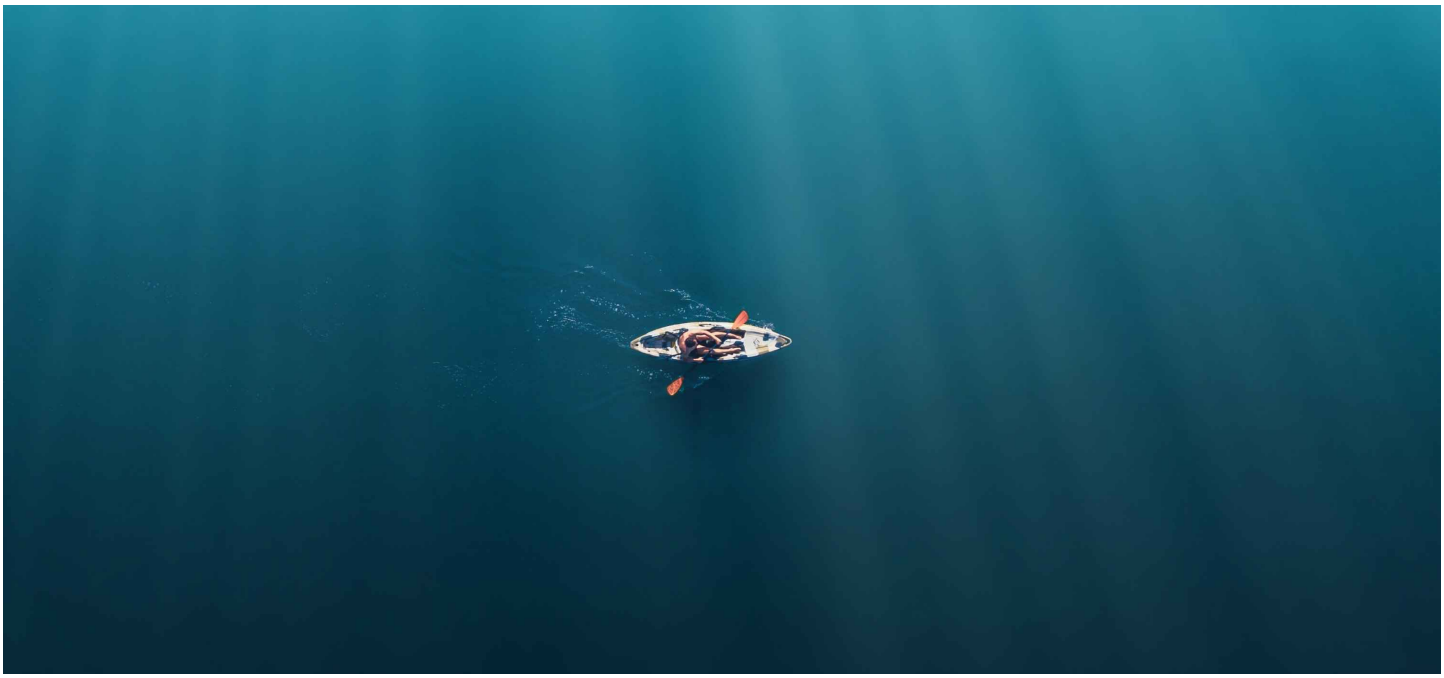

36-Java 14: Switch Expression

Author: Vincent Lau

Learning Objectives

> Understand the ways to use var in Java
>
> Understand what var can do, and cannot do
>
> Understand what exactly compile-time type inference means

Introduction

Switch expressions were introduced in Java 12 as a preview feature and then became a standard feature starting from Java 14. They provide a more concise and expressive way to write switch statements, making code easier to read and write. Switch expressions are a replacement for traditional switch statements, and they can be used both for branching logic and returning values.

Traditional Switch Statement

In a traditional switch statement, you provide a value to be evaluated against different cases. Each case contains a constant value, and if the value matches a case, the corresponding block of code is executed. The `break` statement is used to exit the switch after a case is matched.

```
int day = 2;
String dayName = "";

switch (day) {
    case 1:
        dayName = "Monday";
        break;
    case 2:
        dayName = "Tuesday";
        break;
    // ... other cases ...
    default:
        dayName = "Invalid day";
}
```

Switch Expression

Enum

When you use a **switch expression with an enum type**, you are required to provide a case for each enum constant. This ensures that you handle all possible values of the enum.

Compiler enforces the rule that you must cover all possible values of the enum in your switch expression. Now the switch expression covers all possible values of the `Day` enum, and the code will compile successfully.

```
enum Day {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}

public class SwitchEnumExample {
    public static void main(String[] args) {
        Day day = Day.SATURDAY;

        String dayType = switch (day) {
            // Compile-time check if it handled all possible values of the enum
            case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> "Weekday";
            case SATURDAY, SUNDAY -> "Weekend";
        };
```

```
14
15          System.out.println("Day type: " + dayType);
16      }
17  }
18
```

Lambda

With switch expressions, the case labels can directly return values or expressions. You use the `->` arrow syntax to indicate the value that should be returned when a case is matched. There's no need for `break` statements, as control automatically exits the switch expression when a case is matched.

It's important to note that the lambda syntax in switch expressions makes code more compact and expressive, but like any feature, it should be used judiciously for readability and maintainability.

```
1  // Lambda is just a way to
2  public class SwitchLambdaExample {
3      public static void main(String[] args) {
4          String day = "Monday";
5
6          String result = switch (day) {
7              case "Monday", "Tuesday" -> "Weekday";
8              case "Wednesday" -> "Midweek";
9              case "Thursday", "Friday" -> "Almost Weekend";
10             default -> "Weekend";
11         };
12
13         System.out.println("Day: " + day);
14         System.out.println("Description: " + result);
15     }
16 }
```

yield

Switch expressions also support the use of `yield` to specify the value to be returned:

```
1  public class SwitchYieldExample {
2      public static void main(String[] args) {
3          int day = 2;
4          String dayName = switch (day) {
5          case 1:
```

```
            yield "Monday";
        case 2:
            yield "Tuesday";
        // ... other cases ...
        default: yield "Invalid day";
        };
        System.out.println("Day Name=" + dayName);
    }
}
```

Lambda & yield

When you use Lambda expression in switch, you can use `code block {}` to group the lines of code to represent a case, which is same as normal lambda expression.

But `yield` is required in code block, but NOT `return`.

```
public class SwitchLambdaExample {
    public static void main(String[] args) {
        Integer day = 3;

        String result = switch (day) {
            case 1 -> {
                // other codes ...
                yield "Monday";
            }
            case 2 -> {
                // other codes ...
                yield "Tuesday";
            }
            case 3 -> {
                // other codes ...
                yield "Wednesday";
            }
            // other cases
            default -> {
                // other codes ...
                yield "Weekend";
            }
        };
        System.out.println("Description: " + result);
    }
}
```

To conclude, switch expressions allow more flexible branching, including the ability to return values directly from cases, which makes them particularly useful for concise and functional code.