JEP 361: Switch Expressions

Mục tiêu:
Mục tiêu phát triển của JEP 361 là hiện đại hóa và đơn giản hóa cấu trúc điều khiển switch trong ngôn ngữ Java. Trước đây, switch chỉ được sử dụng như một câu lệnh (statement), không thể trả về giá trị và thường gây lỗi do cơ chế *fall-through* khi lập trình viên quên thêm từ khóa break. JEP 361 được tạo ra để khắc phục những hạn chế này bằng cách mở rộng switch trở thành một biểu thức (expression), cho phép trả về giá trị trực tiếp, đồng thời giới thiệu cú pháp mới sử dụng toán tử -> thay cho dấu hai chấm truyền thống, giúp mã nguồn ngắn gọn và rõ ràng hơn. Ngoài ra, JEP 361 còn bổ sung khả năng gom nhiều nhãn case trên cùng một nhánh và phạm vi biến độc lập trong từng nhánh, giảm thiểu xung đột tên biến và lỗi logic. Mục tiêu sâu xa hơn của JEP 361 là đặt nền tảng cho các cải tiến ngôn ngữ trong tương lai, đặc biệt là các tính năng như *pattern matching* cho switch, góp phần làm cho Java trở nên hiện đại, an toàn và nhất quán hơn trong thiết kế cú pháp.

Chi tiết kỹ thuật:
Từ khi được đề xuất lần đầu tiên dưới dạng JEP 325 (năm 2017, phát hành thử nghiệm trong JDK 12), cho đến khi được chuẩn hóa trong JEP 361 (JDK 14, năm 2020), tính năng *switch expressions* đã trải qua nhiều giai đoạn thay đổi và hoàn thiện. Ở giai đoạn đầu (JEP 325), Java giới thiệu khả năng sử dụng switch như một biểu thức, cho phép trả về giá trị, nhưng việc sử dụng từ khóa break để trả giá trị đã gây ra sự nhầm lẫn với chức năng "thoát khỏi switch" vốn có trước đó. Tiếp thu phản hồi từ cộng đồng lập trình viên, JEP 354 (JDK 13 – *second preview*) đã thay đổi cơ chế này bằng cách giới thiệu từ khóa mới yield để trả giá trị, đồng thời giữ nguyên ý nghĩa truyền thống của break. Đến JEP 361 trong JDK 14, tính năng này được chính thức chuẩn hóa, đánh dấu sự hoàn thiện về cú pháp và hành vi của switch. Phiên bản này cho phép lập trình viên sử dụng song song cả hai cú pháp case L : (cũ) và case L -> (mới), hỗ trợ nhiều nhãn trên cùng một nhánh case, loại bỏ hoàn toàn hiện tượng *fall-through* mặc định, và mở rộng phân tích luồng dữ liệu (flow analysis) để đảm bảo mọi nhánh của switch expression đều có giá trị trả về hoặc ném ra ngoại lệ. Nhờ những cải tiến này, JEP 361 không chỉ hợp nhất những cải tiến từ JEP 325 và JEP 354 mà còn hoàn thiện cú pháp switch hiện đại, rõ ràng, an toàn hơn, đồng thời tạo nền tảng vững chắc cho các tính năng nâng cao như pattern matching được phát triển trong các phiên bản Java sau.

Ảnh hưởng:
JEP 361 – Switch Expressions mang lại nhiều lợi ích rõ rệt cho cả lập trình viên Java và hệ thống JVM. Đối với lập trình viên, cải tiến lớn nhất là cú pháp switch trở nên ngắn gọn, dễ đọc và ít lỗi hơn nhờ việc giới thiệu cú pháp mới với toán tử ->, giúp loại bỏ lỗi phổ biến như *fall-through* do quên break. JEP 361 cũng mở rộng switch từ một câu lệnh (statement) thành một biểu thức (expression), cho phép nó trả về giá trị và được sử dụng trực tiếp trong các phép gán hoặc logic điều kiện. Điều này giúp giảm mã thừa, tăng tính rõ ràng của ý đồ lập trình và hỗ trợ phong cách lập trình hàm hiện đại. Ngoài ra, việc hỗ trợ nhiều nhãn trong một case giúp giảm trùng lặp mã, trong khi mỗi nhánh case có phạm vi biến độc lập giúp tránh xung đột tên biến. Một lợi ích dài hạn khác là JEP 361 đóng vai trò nền tảng cho các tính năng nâng cao như *pattern matching* trong các phiên bản Java sau này (JEP 406, 420, 441).

Về phía JVM và trình biên dịch javac, JEP 361 chỉ thay đổi ở giai đoạn biên dịch mà không ảnh hưởng đến runtime. switch expression được biên dịch xuống bytecode tương tự switch truyền thống, nhưng có thêm cơ chế xử lý giá trị trả về, phân tích luồng dữ liệu và kiểm tra tính đầy đủ của các nhánh. Nhờ vậy, mã biên dịch an toàn và nhất quán hơn mà không cần bổ sung opcode mới hay thay đổi sâu trong JVM. Đồng thời, trình biên dịch có thể tối ưu hóa tốt hơn nhờ biết trước giá trị trả về của switch, giúp cải thiện hiệu năng trong một số tình huống.

Về tổng thể, JEP 361 giúp Java tiến gần hơn với các ngôn ngữ hiện đại như Kotlin hay Scala, làm cho mã nguồn dễ bảo trì, dễ dạy và dễ học hơn, đồng thời tăng khả năng tương thích và mở rộng cho các tính năng ngôn ngữ tương lai. Đây là một bước phát triển quan trọng trong việc hiện đại hóa Java mà vẫn giữ vững tính ổn định và tương thích ngược vốn là đặc trưng của hệ sinh thái này.

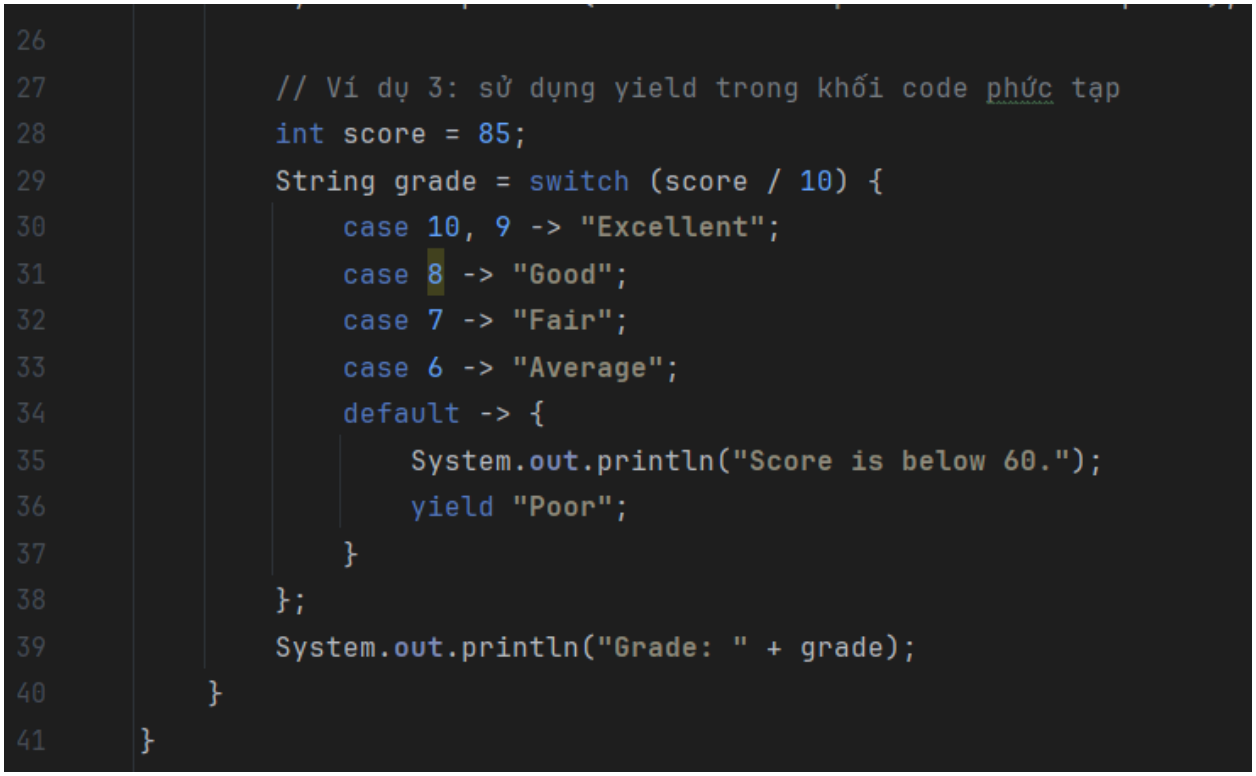
Code demo:

SwitchExpressionDemo.java

```java
public class SwitchExpressionDemo {
    public static void main(String[] args) {

        // Ví dụ 1: switch như một biểu thức (expression)
        String day = "SATURDAY";
        String typeOfDay = switch (day) {
            case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY" -> "Weekday";
            case "SATURDAY", "SUNDAY" -> "Weekend";
            default -> {
                // Có thể dùng khối lệnh phức tạp và yield kết quả
                System.out.println("Unknown day: " + day);
                yield "Invalid";
            }
        };
        System.out.println("Type of day: " + typeOfDay);

        // Ví dụ 2: switch trả về giá trị, không cần biến tạm
        int number = 3;
        String description = switch (number) {
            case 1 -> "One";
            case 2 -> "Two";
            case 3 -> "Three";
            default -> "Other";
        };
        System.out.println("Number description: " + description);
```



```
26
27          // Ví dụ 3: sử dụng yield trong khối code phức tạp
28          int score = 85;
29          String grade = switch (score / 10) {
30              case 10, 9 -> "Excellent";
31              case 8 -> "Good";
32              case 7 -> "Fair";
33              case 6 -> "Average";
34              default -> {
35                  System.out.println("Score is below 60.");
36                  yield "Poor";
37              }
38          };
39          System.out.println("Grade: " + grade);
40      }
41  }
```

```
public class SwitchExpressionDemo {
    public static void main(String[] args) {

        // Ví dụ 1: switch như một biểu thức (expression)
        String day = "SATURDAY";
        String typeOfDay = switch (day) {
            case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"
-> "Weekday";
            case "SATURDAY", "SUNDAY" -> "Weekend";
            default -> {
                // Có thể dùng khối lệnh phức tạp và yield kết quả
                System.out.println("Unknown day: " + day);
                yield "Invalid";
            }
        };
        System.out.println("Type of day: " + typeOfDay);

        // Ví dụ 2: switch trả về giá trị, không cần biến tạm
```

```
        int number = 3;
        String description = switch (number) {
            case 1 -> "One";
            case 2 -> "Two";
            case 3 -> "Three";
            default -> "Other";
        };
        System.out.println("Number description: " + description);

        // Ví dụ 3: sử dụng yield trong khối code phức tạp
        int score = 85;
        String grade = switch (score / 10) {
            case 10, 9 -> "Excellent";
            case 8 -> "Good";
            case 7 -> "Fair";
            case 6 -> "Average";
            default -> {
                System.out.println("Score is below 60.");
                yield "Poor";
            }
        };
        System.out.println("Grade: " + grade);
    }
}
```