#### 3.6 Оператор switch в Java 17

### Что вы узнаете

В этом материале вы познакомитесь с новыми возможностями оператора switch, которые появились в языке Java начиная с версии 17. Вы узнаете основные преимущества нового оператора switch и сможете использовать их далее в рамках обучения на курсе.

### Замена условий на оператор switch

В Java версии 17 появилось несколько новых функций языка, одна из них — улучшенный оператор switch. Этот оператор был в Java и ранее, но только обрабатывал данные и не возвращал значение.

Посмотрим на код с несколькими условными выражениями:

static String formatString(Object o) {  
    String result = "";  
    if (o instanceof Integer) {  
        result = String.format("int %d", (Integer)o);  
    } else if (o instanceof Long) {  
        result = String.format("long %d", (Long)o);  
    } else if (o instanceof Double) {  
        result = String.format("double %f", (Double)o);  
    } else if (o instanceof String) {  
        result = String.format("String %s", o);  
    }  
    return result;  
}

В этом методе мы применяем форматирование к строке в зависимости от аргумента. В каждом условии с помощью оператора instanceof проверяется принадлежность объекта к тому или иному классу, а затем вызывается метод String.format(), который форматирует строку.

В коде много условий, которые трудно читать. Перепишем тот же самый код с помощью switch:

static String formatterPatternSwitch(Object o) {  
    return switch (o) {  
        case Integer i -> String.format("int %d", i);  
        case Long l    -> String.format("long %d", l);  
        case Double d  -> String.format("double %f", d);  
        case String s  -> String.format("String %s", s);  
        default        -> o.toString();  
    };  
}

Код стал намного лучше: каждое условие вместе с действием теперь занимает одну строку. Как это читается:

        case Integer i -> String.format("int %d", i);

«Если переменная принадлежит классу Integer (задаём имя переменной, которое будем использовать в правой части switch), то вызываем метод format, в который передаём эту переменную». Такая проверка на типы называется pattern matching.

В отличие от предыдущих версий, новый switch можно использовать в return.

### Старый и новый switch

Рассмотрим, чем отличается старый switch от нового. Первое отличие: новое выражение использует стрелку ->, а старый оператор продолжает использовать двоеточие.

//Switch Expression example  
switch (o) {  
    case Integer i -> String.format("int %d", i);  
    case Long l    -> String.format("long %d", l);  
}  
//Switch Statement example  
switch (o) {  
  case Integer i:   
    String.format("int %d", i);  
    break;  
  case Long l:  
    String.format("long %d", l);   
    break;  
}

Можно заметить, что в старом операторе switch по-прежнему необходимо использовать оператор break для завершения работы выражения в каждой ветке. Если break не указан, оператор продолжит выполнять блоки кода следующих веток.

В новом операторе всегда используется одно выражение, break писать больше не нужно. Оба оператора продолжают работать с числами и константными значениями, а не только классами. Например, следующий код работает в обоих случаях:

// Предыдущая версия  
switch (value) {  
  case "A":   
    callMethod1();  
    break;  
  case "B":   
    callMethod2();  
    break;  
  default:  
    callMethod3();  
}  
// Новая версия  
switch (value) {  
  case "A"-> callMethod1();  
  case "B"-> callMethod2();  
  default -> callMethod3();  
}

### Интересные возможности switch

Обычный switch выбросит исключение NullPointerException, если переданное значение равно null. Это поведение справедливо и для текущего оператора, но, если мы захотим использовать проверку на типы, это можно обойти:

switch (o) {  
  case null     -> System.out.println("null!");  
  case String s -> System.out.println("String");  
  default       -> System.out.println("Something else");  
}

Иногда требуются дополнительные возможности в проверке типов. Например, проверить длину строки:

switch(value) {  
  case String s && (s.length < 3) -> System.out.println("Строка слишком короткая");  
  case String s && (s.length < 10) -> System.out.println("Нормальная строка");  
  default -> System.out.println("Слишком длинная строка");  
}