switch case
new features

Objectives

After completing this lesson, you should be able to:

- Understand how to use switch-case statements to evaluate conditions.
- **Differentiate** between the traditional switch statement and the **switch expression** (introduced in Java 14)
- **Implement** the arrow syntax (->) to eliminate common errors like missing break statements (fall-through).
- **Utilize** the yield keyword to return values from complex code blocks within a switch.
- **Apply Pattern Matching for switch** to evaluate object types directly, eliminating instanceof checks and explicit casting.
- **Master the use of the when clause** to add additional logical conditions to a selection case, allowing for granular control without nesting if-else blocks.

Topics

- Traditional switch statements (using **break** statements)
- Traditional switch statements (using “**fall through**”)
- **switch expressions**
- Using **guards**
- **Pattern matching**
- Using code blocks and **yield**

Traditional switch statement (using break statements)

- Traditional switch statements evaluate constant values using case statements,
- You should use **break** statements to exit the switch construct when a case is evaluated to true:

```
int dayNum = 3;

String dayName;

switch (dayNum) {
    case 1:
        dayName = "Sunday";
        break;
    case 2:
        dayName = "Monday";
        break;
    ...
    default:
        dayName = "Invalid day";
}
```

Topics

- Traditional switch statements (using **break** statements)
- Traditional switch statements (using “**fall through**”)
- **switch expressions**
- Using **guards**
- **Pattern matching**
- Using code blocks and **yield**

Traditional switch statement (using “fall through”)

- “**fall through**” behavior allows to evaluate multiple cases without duplicating code:

```
int dayNum = 3;

String typeOfDay;

switch (dayNum) {
    case 1:  case 7:
        typeOfDay = ”Weekend";
        break;
    case 2: case 3: case 4: case 5: case 6:
        typeOfDay = ”Weekend";
        break;
    default:
        typeOfDay = "Invalid day";
}
```

Topics

- Traditional switch statements using **break** statements
- Traditional switch statements using “**fall through**”
- **switch expressions**
- Using **guards**
- **Pattern matching**
- Using code blocks and **yield**

switch expressions (introduced in Java SE 14)

- **switch expressions** allow direct assignment of values to a variable using the **arrow syntax.**
- All cases must be covered to avoid exceptions.
- The code is more concise and looks cleaner.

```
int dayNum = 3;

String typeOfDay =

        switch (dayNum) {
                case 1, 7 -> "Weekend";
                case 2, 3, 4, 5, 6 -> "Weekday";

        default -> "Invalid day";
};
```

switch expressions (introduced in Java SE 14)

- **switch expressions** can also be used to return the value from a method:

```
public static String returnTypeOfDay(int dayNum) {

// We can return the result directly
// from the switch expression

        return switch (dayNum) {
                case 1, 7 -> "Weekend";
                case 2, 3, 4, 5, 6 -> "Weekday";
                default -> "Invalid day number";
        };
}
public static void main(String[] args) {

        String typeOfDay = returnTypeOfDay(7);
}
```

Topics

- Traditional switch statements using **break** statements
- Traditional switch statements using “**fall through**”
- **switch expressions**
- Using **guards**
- **Pattern matching**
- Using code blocks and **yield**

Using guards

- Using **guards** it´s possible to evaluate conditions inside the switch expression:

```
int dayNum = 2; // Monday
boolean isHoliday = true;

String result = switch (dayNum) {

        // Guarded pattern:
        // checks the number AND the boolean condition
        case 2, 3, 4, 5, 6 when isHoliday -> "Holiday(Weekday)";
        case 1, 7 when !isHoliday -> "Holiday (Weekend)";

        // We still covering standard cases
        case 1, 7 -> "Weekend";
        case 2, 3, 4, 5, 6 -> "Weekday";
        default -> "Invalid day";
};
```

Topics

- Traditional switch statements using **break** statements
- Traditional switch statements using “**fall through**”
- switch expressions
- Using **guards**
- **Pattern matching**
- Using code blocks and **yield**

Pattern matching

- In previous releases (before Java SE 17), to dynamically determine the data type and cast a variable, developers had to use the **instanceof** operator followed by an **explicit cast**, which often led to verbose and error-prone code:

```
// evaluate if it´s a String using instanceof operator
if (obj instanceof String) {

    String s = (String) obj; // requires explicit casting

    System.out.println(s.toLowerCase());
}
else if (obj instanceof Integer) {

    Integer i = (Integer) obj; // requires explicit casting

    System.out.println(i*i));
}
...
```

Pattern matching (continued)

- Now we can use **pattern matching** to avoid explicit evaluation and apply automatic casting:

```
public void processInput(Object obj) {
    switch (obj) {
        case String s -> // implicit casting
            System.out.println("The length of the String is " +
            s.length());
        case Integer i -> // implicit casting
            System.out.println("The square of the Integer value is "
            + (i * i));
        case null -> // avoid NullPointerException
            System.out.println("null value!");
        default ->  // cover all cases
            System.out.println(”Unknown type: " +
            obj.getClass().getName());
...
```

Topics

- Traditional switch statements using **break** statements
- Traditional switch statements using “**fall through**”
- switch expressions
- Using **guards**
- **Pattern matching**
- Using code blocks and **yield**

Using code blocks and yield

- You can use **yield t**o directly assign a value when executing a code block inside a switch expression:

```
int day = 3;

    String dayName = switch (day) {
        case 1 -> "Monday";
        case 2 -> "Tuesday";
        case 3 -> {
            System.out.println("Midweek");
            yield "Wednesday";
        }
        case 4 -> "Thursday";
        case 5 -> "Friday";
        case 6, 7 -> "Weekend";
        default -> throw new IllegalArgumentException("Invalid day");
    };
```

Summary

In this lesson, you should have learned how to:

- Use a **switch statement** to apply conditions using **break** statements and applying “fall through”.
- Use **switch expressions** with the **arrow syntax** to directly assign values to a variable or to return data from a method.
- Apply **guard conditions**.
- Use **pattern matching** to avoid using excessive **instanceof** operators.

  and avoid explicit casting.
- How to use **yield** to return data from code blocks in switch expressions.