Object Oriented Programming in Java

Branching with Switch Statement & Expression

Suhel Hammoud

Branching with Switch Statement

Switch Statement Basics:

- Controls program flow based on selector variable
- Supported types:
  - `byte`, `short`, `char`, `int`
  - `Character`, `Byte`, `Short`, `Integer`
  - `enum` types
  - `String` (Java SE 7+)

Branching with Switch Statement

```
int quarter = 2;
String quarterLabel;
switch (quarter) {
    case 0: quarterLabel = "Q1 - Winter"; break;
    case 1: quarterLabel = "Q2 - Spring"; break;
    case 2: quarterLabel = "Q3 - Summer"; break;
    case 3: quarterLabel = "Q4 - Fall"; break;
    default: quarterLabel = "Unknown quarter";
}
```

Branching with Switch Statement

▓▓ Break and Fall-Through

- break terminates switch block
- Without break, execution falls through to next case

```java
int month = 2;
List<String> futureMonths = new ArrayList<>();
switch (month) {
    case 1: futureMonths.add("January");
    case 2: futureMonths.add("February");
    case 8: futureMonths.add("August");break;
    default: break;
}
// futureMonths contains [February, August]
```

Branching with Switch Statement

■ Multiple Case Labels

- Single block can handle multiple cases

```java
switch (month) {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
        numDays = 31; break;
    case 4: case 6: case 9: case 11:
        numDays = 30; break;
    case 2: // February calculation
        break;
    default:
        System.out.println("Invalid month");
}
```

Branching with Switch Statement

switch vs if-then-else

- **Switch**: single discrete values

```java
// Good for switch
switch (month) {
    case 1: System.out.println("Jan"); break;
    // ...
}
```

Branching with Switch Statement

■ switch vs if-then-else

- **if-then-else**: ranges or conditions

```java
// Requires if-else
if (temp < 0) {
    System.out.println("Ice");
} else if (temp < 100) {
    System.out.println("Liquid");
} else {
    System.out.println("Vapor");
}
```

Branching with Switch Statement

▓ String in Switch

- Available since Java SE 7
- Case-sensitive comparison (use `toLowerCase()`)

```java
String month = "February";
int monthNumber;
switch (month.toLowerCase()) {
    case "january": monthNumber = 1; break;
    case "february": monthNumber = 2; break;
    // ...
    default: monthNumber = 0; break;
}
```

Branching with Switch Statement

▓ Handling Null Values

- Switch throws NullPointerException with null selector
- Always check for null first

```java
String month = null;
if (month == null) {
    System.out.println("Month is null");
} else {
    switch (month.toLowerCase()) {
        // cases...
    }
}
```

Branching with Switch Expressions

Switch Expressions (Java SE 14+)

- More concise syntax
- No fall-through behavior
- Can return values
- Multiple constants per case

Branching with Switch Expressions

▓ **Traditional switch statement**

```java
Day day = Day.MONDAY;
int len = 0;
switch (day) {
  case MONDAY, FRIDAY, SUNDAY:
    len = 6;
    break;
  // ... other cases ...
}
```

Branching with Switch Expressions

▦ **Switch expression**

```
int len = switch (day) {
  case MONDAY, FRIDAY, SUNDAY -> 6;
  case TUESDAY                -> 7;
  case THURSDAY, SATURDAY     -> 8;
  case WEDNESDAY              -> 9;
};
```

Branching with Switch Expressions

Producing Values

- Entire switch can return a value
- Single expression: implicit return
- Block: use yield keyword

Simple expression

```
String quarterLabel = switch (quarter) {
    case 0 -> "Q1 - Winter";
    case 1 -> "Q2 - Spring";
    default -> "Unknown quarter";
};
```

Branching with Switch Expressions

Block with yield

```java
String label = switch (quarter) {
    case 0 -> {
        System.out.println("Q1 - Winter");
        yield "Q1 - Winter";
    }
    default -> "Unknown quarter";
};
```

Branching with Switch Expressions

■ Exhaustiveness

- Switch expressions must cover all cases
- Default clause often needed
- Exception for complete enum coverage

```java
enum Season { SPRING, SUMMER, FALL, WINTER }
String weather = switch (season) {
  case SPRING -> "Rainy";
  case SUMMER -> "Sunny";
  case FALL   -> "Windy";
  case WINTER -> "Snowy";
  // No default needed - all cases covered
};
```

Branching with Switch Expressions

Colon Syntax

- Traditional `case L:` syntax supported
- Requires explicit `yield`
- Fall-through still applies

```java
String quarterLabel = switch (quarter) {
  case 0:
    System.out.println("Winter");
    yield "Q1 - Winter";
  case 1:
    yield "Q2 - Spring";
  default:
    yield "Unknown quarter";
};
```

Branching with Switch Expressions

▓▓ Null Handling

- Traditional switch throws NPE on null
- Java 17+ preview: null case support

```java
String month = null;
String season = switch (month) {
  case "Jan", "Feb", "Dec" -> "Winter";
  case null -> "No month provided";  // Java 17+ preview
  default -> "Unknown season";
};
```

Best Practice: Always check for null before switching