Complex Looping and Branching

Jim Wilson
Mobile solutions developer & architect

@hedgehogjim jwhh.com

Overview

Nesting loops and if-else

Nesting for loops

Branching

Infinite loops

Course code can be downloaded as part of the course exercise files.

Nesting

- Placing one construct within another

Nesting is commonly used

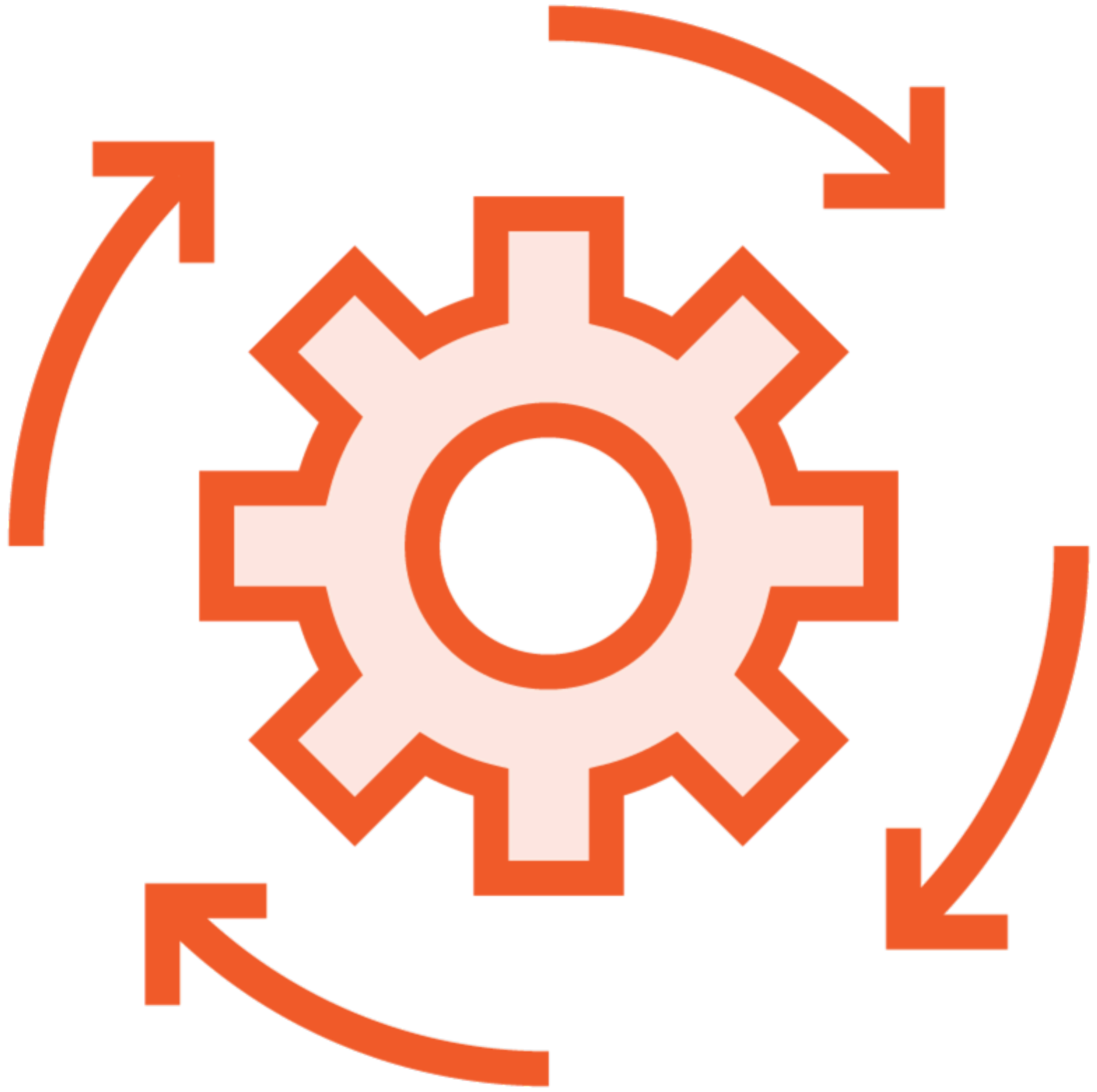
- if-else within if-else
- Loop within if-else
- if-else within loop
- Loop within loop

For Loop with Nested if-else

```
int evenCount = 0;
for(int i = 10; i > 0; i--)
  System.out.print(i);
  if (i % 2 == 0)
    System.out.println(" is even");
    evenCount++;
  }
  else
    System.out.println(" is odd");
}
```

if-else with Nested Do-while

```
int iVal = 1;
if(iVal < 5)
  do
    System.out.println("iVal = " + iVal++);
  while(iVal < 5);
else
  System.out.println("iVal is not less than 5");
```



Nested loops

- One loop contained within another
- The outer loop contains the inner loop

For each iteration of the outer loop

- Inner loop runs from start to finish

Nested For Loops

```
int[][] multi = {{100, 105, 110},
                 {200, 205, 210},
                 {300, 305, 310}};

for(int i = 0; i < multi.length; i++)
    for(int j = 0; j < multi[i].length; j++)
```

Nested For Loops

```
                    0    1    2
int[][] multi = {{100, 105, 110}, 0
                  {200, 205, 210},
                  {300, 305, 310}};
for(int i = 0; i < multi.length; i++)
    for(int j = 0; j < multi[i].length; j++)
        System.out.println(multi[i][j]);
```

Nested For Loops

```
                   0    1    2
int[][] multi = {{100, 105, 110},
                 {200, 205, 210}, 1
                 {300, 305, 310}};
for(int i = 0; i < multi.length; i++)
    for(int j = 0; j < multi[i].length; j++)
        System.out.println(multi[i][j]);
```

Nested For Loops

```
                0    1    2
int[][] multi = {{100, 105, 110},
                 {200, 205, 210},
                 {300, 305, 310}};2
for(int i = 0; i < multi.length; i++)
    for(int j = 0; j < multi[i].length; j++)
        System.out.println(multi[i][j]);
```

Nested For-each Loops

```
                value value value
int[][] multi = {{100, 105, 110}, simple
                 {200, 205, 210},
                 {300, 305, 310}};
for(int[] simple : multi)
    for(int value : simple)
        System.out.println(value);
```

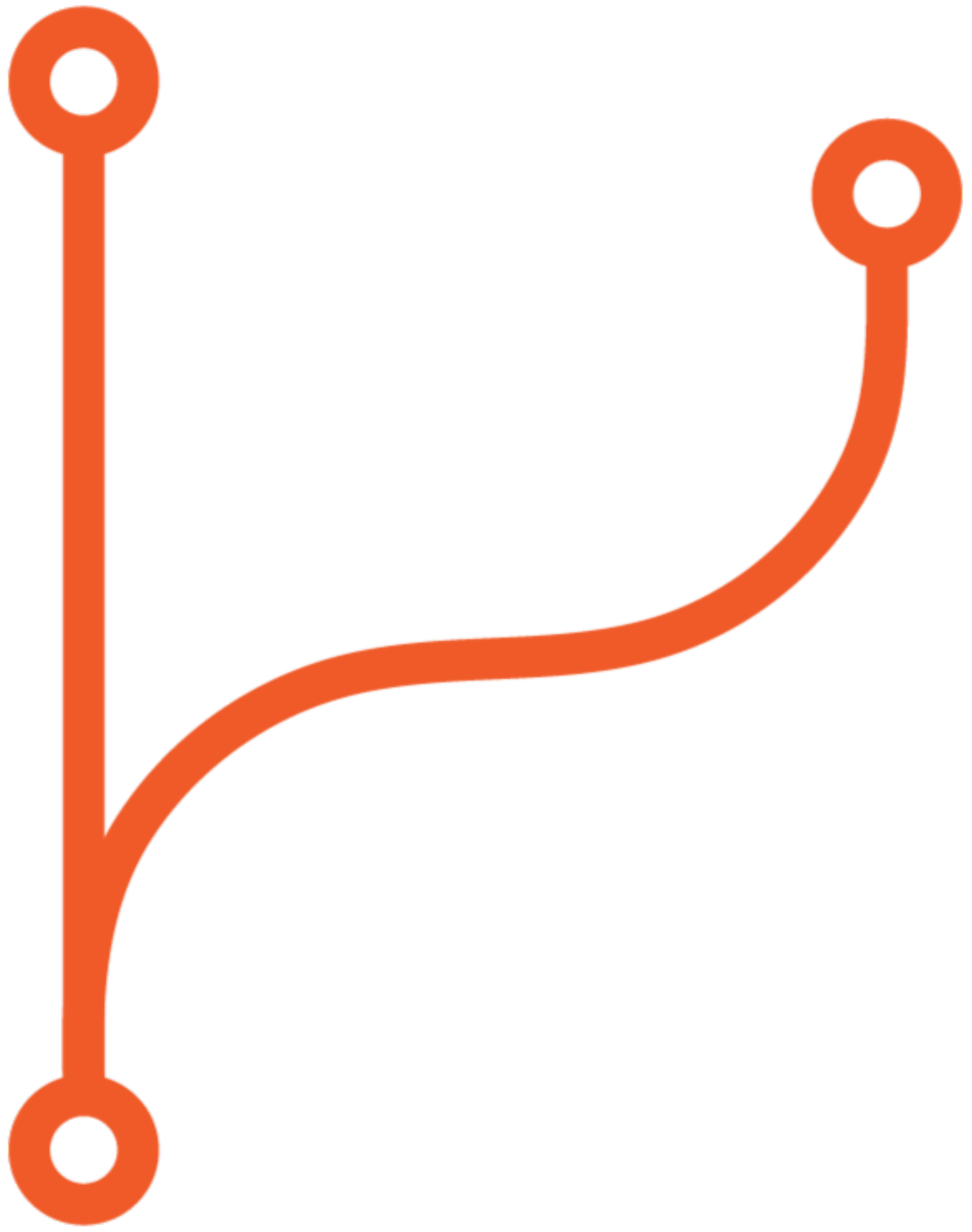
Nested For-each Loops

```
                 valuevaluevalue

int[][] multi = {{100, 105, 110},
                  {200, 205, 210},  simple
                  {300, 305, 310}};

for(int[] simple : multi)
    for(int value : simple)
        System.out.println(value);
```

Nested For-each Loops

value value value

```
int[][] multi = {{100, 105, 110},
                 {200, 205, 210},
                 {300, 305, 310}}; simple
for(int[] simple : multi)
    for(int value : simple)
        System.out.println(value);
```



Branching

- Alters standard code flow

The continue statement

- Skips remainder of current loop iteration

The break statement

- Terminates innermost switch or loop

The Continue Statement

Main.java

```java
int iVal = 0;
while (iVal < 10) {
  iVal++;
  if(iVal % 2 == 0)
    continue;
  System.out.println(iVal);
}
```

```
1
3
5
7
9
```

The Break Statement

Main.java

```
int sum = 0, iVal = 1;
while (iVal < 10) {
  sum += iVal;
  System.out.println("iVal=“ + iVal + " sum=“ + sum);
  if (sum > 5)
    break;
  iVal++;
}
```

```
iVal = 1 sum = 1

iVal = 2 sum = 3

iVal = 3 sum = 6

 << break terminates loop >>
```

Main.java

```
int iValStart = 1;
while (iValStart < 4) {
  System.out.println("iValStart = " + iValStart);
  int sum = 0, iVal = iValStart;
  while (iVal < 10) {
    sum += iVal;
    System.out.println("iVal = " + iVal + " sum = " + sum);
    if (sum > 5)
      break;
    iVal++;
  }
  iValStart++;
}
```

```
iValStart = 1
iVal = 1 sum = 1
iVal = 2 sum = 3
iVal = 3 sum = 6
iValStart = 2
iVal = 2 sum = 2
iVal = 3 sum = 5
iVal = 4 sum = 9
iValStart = 3
iVal = 3 sum = 3
iVal = 4 sum = 7
```

Main.java

```java
for (int iValStart = 1; iValStart < 4; iValStart++) {
  System.out.println("iValStart = " + iValStart);
  int sum = 0, iVal = iValStart;
  while (iVal < 10) {
    sum += iVal;
    System.out.println("iVal = " + iVal + " sum = " + sum);
    if (sum > 5)
      break;
    iVal++;
  }

}
```

```
iValStart = 1
iVal = 1 sum = 1
iVal = 2 sum = 3
iVal = 3 sum = 6
iValStart = 2
iVal = 2 sum = 2
iVal = 3 sum = 5
iVal = 4 sum = 9
iValStart = 3
iVal = 3 sum = 3
iVal = 4 sum = 7
```

The return statement

- Exits current method
- Terminates all switches or loops within that method

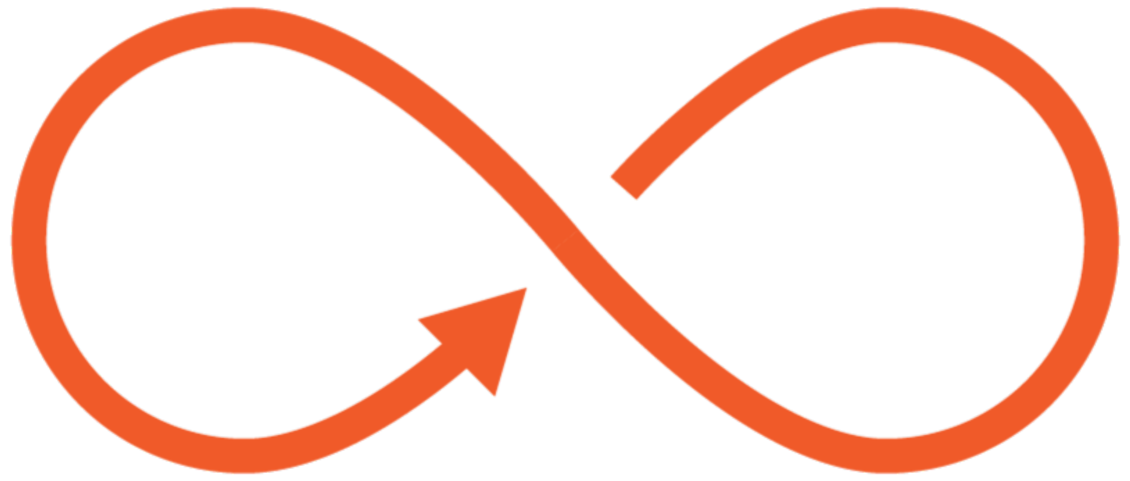
Main.java

```java
private static main(String[] args) {
  System.out.println("Before method call");
  methodWithLoops();
  System.out.println("After method call");
}
private static void methodWithLoops() {
    for (int iValStart = 1; iValStart < 4; iValStart++) {
        System.out.println("iValStart = " + iValStart);
        for (int iVal = iValStart; iVal < 10; iVal++) {
            System.out.println("iVal = " + iVal);
            if (iVal == 3) return;
        }
    }
}
```

Main.java

```java
private static main(String[] args) {
  System.out.println("Before method call");
  methodWithLoops();
  System.out.println("After method call");
}
private static void methodWithLoops() {
    for (int iValStart = 1; iValStart < 4; iValStart++) {
        System.out.println("iValStart = " + iValStart);
        for (int iVal = iValStart; iVal < 10; iVal++) {
            System.out.println("iVal = " + iVal);
            if (iVal == 3) return;
        }
    }
}
```

```
Before method call
iValStart = 1
iVal = 1
iVal = 2
iVal = 3
After method call
```



Infinite loop

- A loop that will repeatedly execute without ever terminating

Sometimes created intentionally

- Generally rely on some external event to terminate the loop

Commonly created inadvertently

- Loop processing never causes the control condition to become false

Infinite Loops

InfiniteWhile.java

```
while(true)
  System.out.println(“Looping...”);
```

InfiniteFor.java

```
for(;;)
  System.out.println(“Looping...”);
```

Techniques For Avoiding Infinite Loops

Avoid overly specific control conditions

Check for infinite loop condition before start

Avoid Overly Specific Control Conditions

Main.java

```java
int iVal = 1;
while(iVal != 4) {
  System.out.println("iVal = " + iVal);
  iVal += 2;
}
```

```
iVal = 1
iVal = 3
iVal = 5
   ⋮
```

Avoid Overly Specific Control Conditions

Main.java

```
int iVal = 1;
while(iVal <= 4) {
  System.out.println("iVal = " + iVal);
  iVal += 2;
}
```

```
iVal = 1
iVal = 3
```

Check Infinite Loop Condition Before Start

Main.java

```java
int stepVal =   1;

  for(int iVal = 1; iVal < 10; iVal += stepVal)
    System.out.println("iVal = " + iVal);
```

Check Infinite Loop Condition Before Start

Main.java

```
int stepVal = -1;
if (stepVal > 0)
  for(int iVal = 1; iVal < 10; iVal += stepVal)
    System.out.println("iVal = " + iVal);
else
  System.out.println("Invalid stepVal: " + stepVal);
```

Summary

Nesting

- Placing one construct within another

Nested loops

- One loop contained within another
- The outer loop contains the inner loop

For each iteration of the outer loop

- Inner loop runs from start to finish

Summary

The continue statement

- Skips remainder of current iteration

The break statement

- Terminates innermost switch or loop

The return statement

- Exits current method
- Terminates all switches and loops within that method

Summary

Infinite loop

- Repeatedly executes without ever terminating

Techniques for avoiding infinite loops

- Avoid overly specific control conditions
- Check for infinite loop condition before loop start

Continuing Your Preparation

Working with Classes and Interfaces in Java