10. Branching Statements

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Outline

- The break Statement
- The continue Statement
- The return Statement
- References

The break Statement

The break Statement

- The break statement has two forms:
  - labeled
  - unlabeled
- You saw the unlabeled form in the previous discussion of the switch statement.
- You can also use an unlabeled break to terminate a for, while, or do-while loop

The break Statement

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

        int[] arrayOfInts = {32, 87, 3, 589, 12, 1076,
                2000, 8, 622, 127};
        int searchfor = 12;

        int i;
        boolean foundIt = false;

        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor +
                    " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

The break Statement

- This program searches for the number 12 in an array.
- The break statement terminates the for loop when that value is found.
- Control flow then transfers to the print statement at the end of the program.
- This program's output is:

```
Found 12 at index 4
```

The labeled break Statement

- An unlabeled break statement terminates the innermost switch, for, while, or do-while statement,
- But a labeled break terminates an outer statement.

The labeled break Statement

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

        int[][] arrayOfInts = { {32, 87, 3, 589},
                                {12, 1076, 2000, 8},
                                {622, 127, 77, 955}
                              };
        int searchfor = 12;
        int i;
        int j = 0;
        boolean foundIt = false;

        search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length; j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }
```

The labeled break Statement

```
        if (foundIt) {
            System.out.println("Found " +
                    searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

- The break statement terminates the labeled statement
- This is the output of the program:

  Found 12 at 1, 0

The continue Statement

The continue Statement

- The continue statement skips the current iteration of a for, while, or do-while loop.
- The continue statement has two forms:
  - labeled
  - unlabeled
- The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

The continue Statement

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

        String searchMe = "peter piper picked a peck of " +
                "pickled peppers";
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            //interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;

            //process p's
            numPs++;
        }
        System.out.println("Found " + numPs +
                " p's in the string.");
    }
}
```

The continue Statement

- ContinueDemo steps through a String, counting the occurrences of the letter "p".
- If the current character is not a p, the continue statement skips the rest of the loop and proceeds to the next character.
- If it is a p, the program increments the letter count.
- Here is the output of this program:

  Found 9 p's in the string.

The labeled continue statement

- A labeled continue statement skips the current iteration of an outer loop marked with the given label.

The labeled continue statement

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

        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;

        int max = searchMe.length() - substring.length();

        test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                    continue test;
                }
            }
            foundIt = true;
            break test;
        }
        System.out.println(foundIt ? "Found it" :
                "Didn't find it");
    }
}
```

The labeled continue statement

- ContinueWithLabelDemo uses nested loops to search for a substring within another string.
- Two nested loops are required: one to iterate over the substring and one to iterate over the string being searched.
- The program uses the labeled form of continue to skip an iteration in the outer loop.
- Here is the output from this program:

```
Found it
```

The return Statement

The return Statement

- The last of the branching statements is the return statement.
- The return statement exits from the current method, and control flow returns to where the method was invoked.
- The return statement has two forms:
  - one that returns a value
  - one that doesn't returns a value
- To return a value, simply put the value (or an expression that calculates the value) after the return keyword.

The return Statement

- Example:

  return ++count;

- The data type of the returned value must match the type of the method's declared return value.
- When a method is declared void, use the form of return that doesn't return a value.

  return;

- The Calling an Object's Methods will be discussed later.

References

References

- S. Zakhour, S. Hommel, J. Royal, I. Rabinovitch, T. Risser, M. Hoeber, **The Java Tutorial: A Short Course on the Basics**, 4th Edition, Prentice Hall, 2006. (Chapter 3)

The End