Lecture 3: Selection Statements

Based on Chapter 3 in Introduction to Java Programming, Y. Daniel Liang, Brief Version, 9/E

You should be familiar with most of the concepts covered in this chapter. We will quickly go over the following topics:

- Boolean data type and comparison operators
- Different types of selection statements in Java
- Logical operators
- Formatted output
- Debugging techniques

1 Boolean data type and comparison operators

Boolean data type is used for a variable whose value can be either true or false. In Java the boolean type is used for such variables:

    boolean isGreater = true;

Comparison operators compare two values and produce (or return) true or false (assume radius is equal to 5, in the examples below):

- less than: radius < 0 is false
- less than or equal: radius <= 0 is false
- greater than: radius > 0 is true
- greater than or equal: radius >= 0 is true
- equal to: radius == 0 is false
- not equal to: radius != 0 is true

WARNING: == (double equal sign) is used for comparison, = (single equal sign) is used for assignment; confusing these two results in a common logical error that is not caught by the compiler.

2 Different types of selection statements in Java

2.1 if ... statement

Syntax:

    if ( Boolean-expression )
    {
        statements to execute if Boolean-expression is true;
    }
Example:

```java
...
if ( radius >= 0 )
{
    area = radius * radius * PI;
    System.out.println("The area is " + area);
}
...
```

### 2.2 if ... else ... statement, a.k.a. two way if statement

Syntax:

```java
if ( Boolean-expression )
{
    statements to execute if Boolean-expression is true;
}
else
{
    statements to execute if Boolean-expression is false;
}
```

### 2.3 if ... else if ... else ... statement, a.k.a. multi-way if statement

Syntax:

```java
if ( Boolean-expression1 )
{
    statements to execute if Boolean-expression1 is true;
}
else {
    if ( Boolean-expression2 )
    {
        statements to execute if Boolean-expression2 is true;
    }
    else {
        if ( Boolean-expression3 )
        {
            statements to execute if Boolean-expression3 is true;
        }
        else {
            ...
        }
    }
}
```
Alternative, better, notation:

```java
if ( Boolean-expression1 )
{
    statements to execute if Boolean-expression1 is true;
}
else if ( Boolean-expression2 )
{
    statements to execute if Boolean-expression2 is true;
}
else if ( Boolean-expression3 )
{
    statements to execute if Boolean-expression3 is true;
}
else if ( ... )
{
    ...
}
else
{
    statements to execute if all Boolean-expressions are false;
}
```

### 2.4 switch statements

Syntax:

```java
switch (switch-expression) {
    case value1: statements to execute if switch-expression has value1;
                  break;
    case value2: statements to execute if switch-expression has value2;
                  break;
    ...
    case valueN: statements to execute if switch-expression has valueN;
                  break;
    defaults:    statements to execute if switch-expression does not match any of the above values
                 ...
                 break;
}
```
3 Logical operators

Logical operators are used in combination with comparison operators and boolean variables to produce more compound Boolean expressions. The logical operators in Java are:

- `!` not
- `&&` and
- `||` or
- `^` exclusive or (not exponentiation)

### Truth tables

|   |   | !a | a && b | a || b | a ^ b |
|---|---|----|--------|--------|-------|
| false | false | true | false | false | false |
| false | true  | true | false | true  | true  |
| true | false | false | false | true  | true  |
| true | true  | false | true  | true  | false |

### Short-circuit evaluation:

- If the first operand of the `&&` operator is false, the second operand is not evaluated.
- If the first operand of the `||` operator is true, the second operand is not evaluated.

4 Common errors

Forgetting braces (especially for the previous Python users):

```java
if ( radius >= 0 )
area = radius * radius * PI;
System.out.println( area );
```

Wrong

```java
if ( radius >= 0 )
{
area = radius * radius * PI;
System.out.println( area );
}
```

Correct

In Java a block of code has not be surrounded by the curly braces, not just indented!
The only exception to this rule is when the block of code consists of a single statement.

Semicolon at the end of the if line:

```java
if ( radius >= 0 ) ;
{
    area = radius * radius * PI;
    System.out.println( area );
}
```

Wrong

```java
if ( radius >= 0 ) {
}
{
    area = radius * radius * PI;
    System.out.println( area );
}
```

Correct
Dangling else ambiguity:

```java
int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
    else
        System.out.println("B");

OR

int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
    else
        System.out.println("B");
```

The `else` always matches the `if` immediately before it, unless curly braces are used to change that.

Redundant testing of Boolean values:

```java
boolean even = true;
if (even == true)
    System.out.println("Even");
else
    System.out.println("Odd");
```

This works, but ...

```java
boolean even = true;
if (even)
    System.out.println("Even");
else
    System.out.println("Odd");
```

This is better (and equivalent)!

Comparison of Boolean variables to true or false is redundant.

Forgetting the break statement in the case of switch statement:

```java
switch(grade) {
    case 'A': System.out.println("A");
    case 'B': System.out.println("B");
    case 'C': System.out.println("C");
    case 'D': System.out.println("D");
    case 'F': System.out.println("F");
    default: System.out.println("Wrong");
}
```

This compiles, but when grade has value of A, A B C D E F is printed.

With break statement, when grade has value of A only A is printed.

This is known as the **fall-through behavior**.

4.1 Code examples

See QuadraticEquationSolver02.java for an improved version of our QuadraticEquationSolver program from the previous lecture.

5 Formatting console output

We saw `System.out.print()` and `System.out.println()` methods - they both print text to the console. `System.out.printf()` allows us to print formatted output.

Syntax:

```
System.out.printf( format, item1, item2, ..., itemN);
```

*format* is a string that consists of substrings and *format specifiers*.

*item1*...*itemN* are values (numeric, character, string, Boolean), either variables or constants
Simple format specifier consists of a percent sign followed by a letter: b, c, d, f, e, or s that indicates the type of the item corresponding to the format specifier.

\%b a Boolean value
\%c a character
\%d a decimal integer
\%f a floating point number
\%e a number in scientific notation
\%s a string

Example:

```java
System.out.printf("Area of circle with radius %f is %f.", radius, area);
System.out.printf("Given that x is %b and y is %b, the value of x && y is %b.", x, y, x&&y);
System.out.printf("Student: %s, tuition: $%f.", name, tuition);
```

Output:

```
Area of circle with radius 5.250000 is 86.587594.
Given that x is true and y is false, the value of x && y is false.
Student: John Smith, tuition: $25036.210000.
```

Format specifier can include field width and precision in addition to the conversion code (indicating the type of the value):

\%W.PC

where W is the field width, P is the precision and C is the conversion code.

Examples:

\%5c Output the character and add four spaces before the character item.
\%6b Output the Boolean value and add one space before the false value and two spaces before the true value.
\%5d Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
\%10.2f Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7 add spaces before the number. If the number of digits before the decimal point in the item is > 7 the width is automatically increased.
\%12s Output the string with width at least 12 characters. If the string item has less than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

**Code example:** see PrintfExample.java

6 Debugging