Revisiting Logical Operators

When searching you could say:
- Looking for results for Monet and his use of color
- Looking for movie results for Sci Fi or Crime
- Looking for a van or SUV - not both
Revisiting Logical Operators
Revisiting Logical Operators
Revisiting Logical Operators
Motivations

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

System.out.println("Welcome to Java");

So, how do you solve this problem?
Opening Problem

Problem:

100 times

```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

...
Introducing while Loops

```java
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java");
    count++;
}
```
while Loop Flow Chart

while (loop-continuation-condition) {
   // loop-body;
   Statement(s);
}

int count = 0;
while (count < 100) {
   System.out.println("Welcome to Java!");
   count++;
}
Trace while Loop

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```
Trace while Loop, cont.

```java
int count = 0;

while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

(count < 2) is true
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
Trace while Loop, cont.

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

Increase count by 1
count is 1 now
Trace while Loop, cont.

int count = 0;

while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}

(count < 2) is still true since count is 1
Trace while Loop, cont.

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

Print Welcome to Java
Trace while Loop, cont.

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

Increase count by 1

count is 2 now
Trace while Loop, cont.

```
int count = 0;

while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

(count < 2) is false since count is 2 now
Trace while Loop

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

The loop exits. Execute the next statement after the loop.
Ending a Loop with a Sentinel Value

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*. 
Caution

Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing $1 + 0.9 + 0.8 + ... + 0.1$:

```java
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```
do-while Loop

do {
    // Loop body;
    Statement(s);
} while (loop-continuation-condition);
Lecture 2
for Loops

for (initial-action;
    loop-continuation-condition;
    action-after-each-iteration) {
    // loop body;
    Statement(s);
}

int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
Trace for Loop

int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

animation
Declare i
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

Execute initializer
    i is now 0
Trace for Loop, cont.

int i;
for (i = 0; i < 2; i++) {
    System.out.println( "Welcome to Java!"); 
}

(i < 2) is true since i is 0
Trace for Loop, cont.

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```
Trace for Loop, cont.

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 1
Trace for Loop, cont.

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

(i < 2) is still true since i is 1
Trace for Loop, cont.

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```
Trace for Loop, cont.

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

Execute adjustment statement

i now is 2
Trace for Loop, cont.

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
```

(i < 2) is false since i is 2
Trace for Loop, cont.

int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

Exit the loop. Execute the next statement after the loop.
**Note**

The *initial-action* in a *for* loop can be a list of zero or more comma-separated expressions. The *action-after-each-iteration* in a *for* loop can be a list of zero or more comma-separated statements. Therefore, the following two *for* loops are correct. They are rarely used in practice, however.

```java
for (int i = 1; i < 100; System.out.println(i++));
```

```java
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```
Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```java
for (; ; ) {
    // Do something
}
```

Equivalent

```java
while (true) {
    // Do something
}
```

(a)  

(b)
Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

```java
for (int i=0; i<10; i++);  
{
    System.out.println("i is " + i);
}
```
Similarly, the following loop is also wrong:

```java
int i = 0;
while (i < 10); {  // Logic Error
    System.out.println("i is " + i);
    i++;
}
```

In the case of the `do` loop, the following semicolon is needed to end the loop.

```java
int i = 0;
do {
    System.out.println("i is " + i);
    i++;
} while (i<10);  // Correct
```
Which Loop to Use?

The three forms of loop statements, `while`, `do-while`, and `for`, are expressly equivalent; that is, you can write a loop in any of these three forms. For example, a `while` loop in (a) in the following figure can always be converted into the following `for` loop in (b):

```
while (loop-continuation-condition) {
    // Loop body
}
```

```
for (; loop-continuation-condition; ) {
    // Loop body
}
```

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):

```
for (initial-action;
    loop-continuation-condition;
    action-after-each-iteration) {
    // Loop body;
}
```

```
initial-action;
while (loop-continuation-condition) {
    // Loop body;
    action-after-each-iteration;
}
```
Recommendations

Use the one that is most intuitive and comfortable for you.

- A **for** loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.
- A **while** loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A **do-while** loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.
Misc Slides
Minimizing Numerical Errors

Numeric errors involving floating-point numbers are inevitable. This section discusses how to minimize such errors through an example.

Here is an example that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.
Problem: Finding the Greatest Common Divisor

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest common divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.
Problem: Predicting the Future Tuition

Problem: Suppose that the tuition for a university is $10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?
Problem: Predicating the Future Tuition

double tuition = 10000;  int year = 0  // Year 0

  tuition = tuition * 1.07; year++;    // Year 1
  tuition = tuition * 1.07; year++;    // Year 2
  tuition = tuition * 1.07; year++;    // Year 3

  ...

Case Study: Converting Decimals to Hexadecimals

Hexadecimals are often used in computer systems programming (see Appendix F for an introduction to number systems). How do you convert a decimal number to a hexadecimal number? To convert a decimal number \(d\) to a hexadecimal number is to find the hexadecimal digits \(h_n, h_{n-1}, h_{n-2}, \ldots, h_2, h_1, h_0\) such that

\[
d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \ldots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0
\]

These hexadecimal digits can be found by successively dividing \(d\) by 16 until the quotient is 0. The remainders are \(h_0, h_1, h_2, \ldots, h_{n-2}, h_{n-1}, h_n\) and \(h_n\).
Problem: *Monte Carlo Simulation*

The Monte Carlo simulation refers to a technique that uses random numbers and probability to solve problems. This method has a wide range of applications in computational mathematics, physics, chemistry, and finance. This section gives an example of using the Monte Carlo simulation for estimating $\pi$.

$$\frac{\text{circleArea}}{\text{squareArea}} = \frac{\pi}{4}.$$  

$\pi$ can be approximated as $4 \times \frac{\text{numberOfHits}}{\text{numberOfTrials}}$. 

Companion Website

MonteCarloSimulation

Run
Using `break` and `continue`

Examples for using the `break` and `continue` keywords:

- TestBreak.java
- TestContinue.java
```java
public class TestBreak {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

        while (number < 20) {
            number++;
            sum += number;
            if (sum >= 100)
                break;
        }

        System.out.println("The number is " + number);
        System.out.println("The sum is " + sum);
    }
}
```
public class TestContinue {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

        while (number < 20) {
            number++;
            if (number == 10 || number == 11)
                continue;
            sum += number;
        }

        System.out.println("The sum is " + sum);
    }
}
Guessing Number Problem Revisited

Here is a program for guessing a number. You can rewrite it using a `break` statement.

GuessNumberUsingBreak
Run
Problem: Checking Palindrome

A string is a palindrome if it reads the same forward and backward. The words “mom,” “dad,” and “noon,” for instance, are all palindromes.

The problem is to write a program that prompts the user to enter a string and reports whether the string is a palindrome. One solution is to check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-to-last character. This process continues until a mismatch is found or all the characters in the string are checked, except for the middle character if the string has an odd number of characters.
Problem: Displaying Prime Numbers

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is prime if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

• For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
• Determine whether a given number is prime.
• Count the prime numbers.
• Print each prime number, and print 10 numbers per line.
Debugging Loops in IDE Tools

Supplements II.C, II.E, and II.G.
Problem: Repeat Addition Until Correct

Recall that Listing 3.1 AdditionQuiz.java gives a program that prompts the user to enter an answer for a question on addition of two single digits. Using a loop, you can now rewrite the program to let the user enter a new answer until it is correct.

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidernote.doc.
Problem: Guessing Numbers

Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently. Here is a sample run:
Problem: An Advanced Math Learning Tool

The Math subtraction learning tool program generates just one question for each run. You can use a loop to generate questions repeatedly. This example gives a program that generates five questions and reports the number of the correct answers after a student answers all five questions.