Lecture 8: Loops

CS 170, Section 000, Fall 2009
22 September 2009
Lecture Plan

- Homework 1 last questions?
- Loops
- Eclipse (software)
Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times.

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

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!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
A Better Approach

Main point of today’s lecture: LOOP
Objectives

- To use `while`, `do-while`, and `for` loop statements to control the repetition of statements (§ § 4.2-4.4).
- To know the similarities and differences between three types of loops (§ 4.5).
- To write nested loops (§ 4.6).
- To learn the techniques for minimizing numerical errors (§ 4.7).
- To implement program control with `break` and `continue` (§ 4.9).
while Loop Flow Chart

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

while (count < 100) {
    System.out.println("Welcome");
    count += 1;
}

count = 0;

(count < 100)?
true
false

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

false
true
false

9/22/2009  CS170, Section 000, Fall 2009
Trace while Loop

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

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.

```java
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++;
}
```
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.

```java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

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

The loop exits. Execute the next statement after the loop.

System.out.println("exit the loop");
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. The program then tells the user whether the input is correct, too low or too high.

Modify the above program so that it 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.
public static void main(String[] args) {
    int number = (int)(Math.random() * 101);

    Scanner input = new Scanner(System.in);
    System.out.println("Guess a magic number between 0 and 100");

    System.out.print("\nEnter your guess: "); //prompt user
    int guess = input.nextInt();

    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
}
while (guess != number) {
    System.out.print("Enter your guess: "); //prompt user
    int guess = input.nextInt();
    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
}

~cs170000/inclass/Sept22/GuessNumber.java
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*.

Write a program that reads and calculates the sum of an *unspecified number of integers*. The input 0 signifies the end of the input.
System.out.print("Enter int value (0 to exit): ");
int data = input.nextInt();

int sum = 0;
while (data != 0) {
    data = input.nextInt();
    if (data == 0) {
        sum += data;
    } else {
        sum += data;
    }
}
print(sum);
System.out.print("Enter int value (0 to exit): ");
int data = input.nextInt();

int sum = 0;
while (data != 0) {// Keep reading until input=0
    sum += data;
    System.out.print("Enter value (0 to exit): ");
    data = input.nextInt();
}
System.out.println("The sum is " + sum);

~cs170000/inclass/Sept22/SentinelValue.java
do–while loop*

```
do {
    printf("Welcome");
    count ++;
} while (count < 2);
```
Sentinel Example, Reloaded

Use do-while loop to rewrite the program that reads
and calculates the sum of an unspecified number of
integers. The input 0 signifies the end of the input.

```java
int sum = 0;
int input = scanner.nextInt();

do {
    sum += input;
    input = scanner.nextInt();
} while (input != 0);
```

~cs170000/inclass/Sept22/SentinelValueDoWhile.java
for (initial-action; loop-continuation-condition; action-after-each-iteration) {
    // loop body;
    Statement(s);
}

*for Loops*

Use whenever *(avoid if possible)*
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

Declare i
int i;
for (i = 0; i < 2, i++) {
    System.out.println("Welcome to 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!");
}
```
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to 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!");
}
```
int i;
for (i = 0; i < 2 {  // Increment 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!");
}
```

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

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

Exit the loop. Execute the next statement after the loop.
Additional Notes on **for** loop

The **initial-action** and **action-after-each-iteration** in a **for** loop can be a list of zero or more comma-separated expressions.

```java
    int i = i
    for (int i = 1; i < 100; System.out.println(i++));
    for (int i = 0, j = 0; (i + j < 10); i++, j++) {
        // Do something
    }
```

The **control variable must** always be declared inside the control structure of the loop or before the loop. If the loop control variable is used only in the loop, it is good **programming practice** to declare it in the initialisation--assignment of the **for** loop. If the variable is declared inside the loop control structure, it cannot be referenced outside the loop. For example, you cannot reference `i` outside the 2nd `for` loop, because it is declared inside the loop.
If the loop-continuation-condition in a for loop is omitted, it is implicitly true.

```java
for (int i = 0; i < 100; i++) {
    print("blah");
    if (i < 100) break;
}
```

(a) 

Equivalent

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

(b) Better!
Caution

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

```
for (int i=0; i<10; i++);
{
    System.out.println("i is "+i);
}
```

Logic Error
Caution, cont.

int i=0;
while (i < 10);
{
    System.out.println("i is " + i);
    i++;
}

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

- while and for loops are pre-test loops, do-while are post-test loops.
- The three loops are expressively equivalent.

A while loop in (a) below can always be converted into the for loop in (b):

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

Equivalent

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

A for loop in (a) below can generally be converted into the 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;
}
```

Equivalent

```
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.
Summary So Far

- While loop
- Do-while loop
- For loop

- Nested loops
Nested Loops

- **Print Triangle**
- *Outer* loop for triangle rows
- *Inner* loop for triangle columns for each row

```java
for (int i = 1; i <= n; i++) {
    // draw one row
    for (int j = 0; j < i; j++) {
        print("[]"); // print an element
    }
    println(); // move to the next line
}
```

- Ex: Modify code to print a square instead of a triangle
Example: Multiplication Table

Write a program that uses nested for loops to print a multiplication table.

<table>
<thead>
<tr>
<th>Multiplication Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2  3  4  5  6  7  8  9</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
Multiplication Table: Implementation

```java
for (int i = 1; i <= 9; i++) {
    System.out.println(i);
    for (int j = 1; j <= 9; j++) {
        int val = i * j;
        System.out.println(val);
    }
}
```

MultiplicationTable.java
Summary: Implementing Loops

• **Step 1.** List the work that needs to be done in every step of the loop body

• **Step 2.** Find out how often the loop is repeated and where we can determine the loop is finished—determine whether to use while or for loops

• **Step 3.** Implement the loop by putting the operations from Step 1 into the loop body

• **Step 4.** Double check variable initializations and updates and check for infinite loop and off-by-one errors
Example: compute the sum for a series of numbers: 0.01, 0.02, ..., 1.0.

// Initialize sum

// Add 0.01, 0.02, ..., 0.99, 1 to sum
for (float f = 0.01; f <= 1.0f; f += 0.01) {
    sum += f;
}

// Display result
System.out.println("The sum is " + sum);
TestSum.java
Compare floating-point numbers

Don’t use == or != to compare floating-point numbers

double data = Math.pow(Math.sqrt(2), 2) - 2;

if (data == 0) // wrong
    System.out.println("data is zero");
else
    System.out.println("data is not zero");
Comparing Floating-Point Numbers

To compare floating-point numbers test whether they are close enough:

$$|x - y| \leq \varepsilon$$

where $\varepsilon$ is a small number such as $10^{-14}$

```java
final double EPSILON = 1E-14;
if (Math.abs(x - y) <= EPSILON)
    // x is approximately equal to y
```
Debugging

- Etymology:
  - De (remove) bugs (?)

- A software bug is used to describe an error, flaw, mistake or fault in a computer program or system that produces an incorrect or unexpected result.
Lab 4 Preview: Eclipse

- **Eclipse**: Integrated Development Environment (IDE)
  - Note: requires X-windows access (in lab or from home)
- Helps with syntax, common errors
- Allows to **step** through program code, one statement at a time
- Can **examine** variable values
- **Extremely powerful** (lots of options/features): don’t be intimidated!