CS1150
Principles of Computer Science
Loops

Yanyan Zhuang

Department of Computer Science

http://www.cs.uccs.edu/~yzhuang

Review

- Boolean variables
  - Assume x=3, y=1, true or false? !(x<2) || y>3
- If statement
  - Be careful: multiple/nested if…else
  - By default: else is mathced with ___ if?
- Switch statement
  - Be careful: where to use break

Overview

- While loop
- Do…while loop
- For loop

Opening Problem: Why Loops?

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!");


…

…

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

Introducing while Loops

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

Introducing while Loops

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

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

How It Works

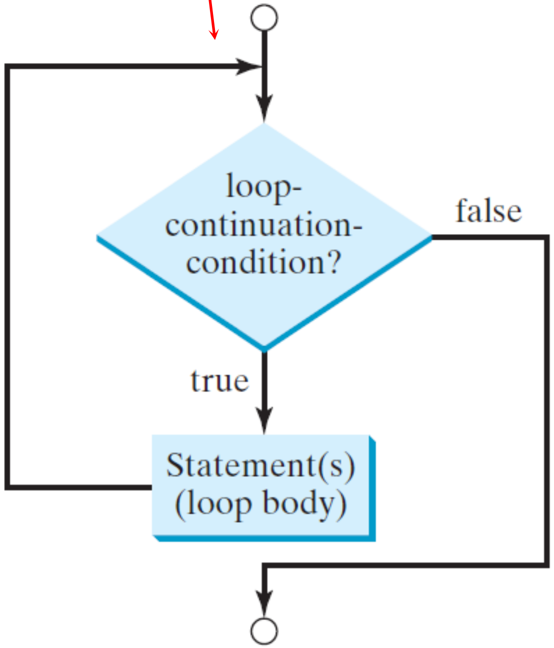
- The loop continuation condition - boolean expression - is evaluated
- If the condition is true, the statements in the loop body are executed
- When execution of loop body statements is complete, control returns to the loop condition
  - The loop continuation condition is evaluated again
- When the loop condition is false, control goes to statements following the loop

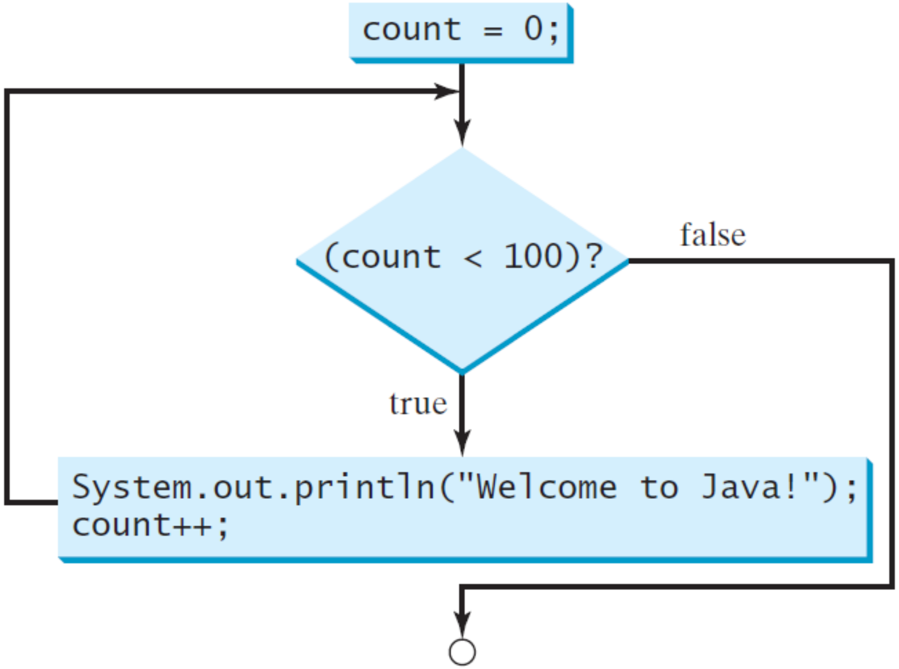
Note: if the loop continuation condition evaluates to false the first time, the entire while loop is skipped

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++;
}
```





Rules for While Loops

- The loop condition must be a boolean expression
  - Boolean expression must be in parentheses
  - Boolean expressions are formed using relational or logical operators
- Loop condition
  - Usually a statement **before** while loop "initializes" loop condition to *true*
  - Some statement within loop body eventually change the condition to *false*
- If the condition is never changed to false, the program is forever in the loop
  - This is called an "infinite loop"
- Curly braces are not necessary if only one statement in loop
  - But best practice is to always include curly braces

Trace while Loop

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

Initialize count
(which we often call control variable)

Trace while Loop, cont.

(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++;
}
```

Print Welcome to Java

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.

```
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.

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

Print Welcome to Java

Trace while Loop, cont.

```
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, cont.

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

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

Let’s look at the first example PrintNTimes.java

Infinite loop example

- In this example, nothing in the loop body changes the value of the control variable

```
count = 1;   // Initializes the loop control variable
while (count <= 5) {
  System.out.println("The value of count is " + count);
}
```

- This is an infinite loop because (count <= 5) is always true
  - Nothing changes the value of count in the loop body
- If you accidentally create an infinite loop, use terminate button (red square) in Eclipse to make it stop

- Placing a semicolon at the end of the while-clause creates an infinite loop - be careful!

```
int iteration = 1;
while (iteration <= 10); {
        System.out.println("Iteration  = " + iteration);
        iteration++;
}
```

Off-by-one Error

- Common issue with loops: Loop body executes one more/less than expected
- Example:

```
System.out.println("I'm going to count to three, ready set....");
count = 1;
while (count < 3) {
  System.out.println(count);
  count++;
}
```

Off-by-one Error

- Common issue with loops: Loop body executes one more/less than expected
- Example:

```
System.out.println("I'm going to count to three, ready set....");
count = 1;
while (count < 3) {
   System.out.println(count);
   count++;
}
```

Output:

```
I'm going to count to three, ready set....
1
2
```

Problem: Repeat Addition Until Correct

See RepeatAdditionQuiz.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.

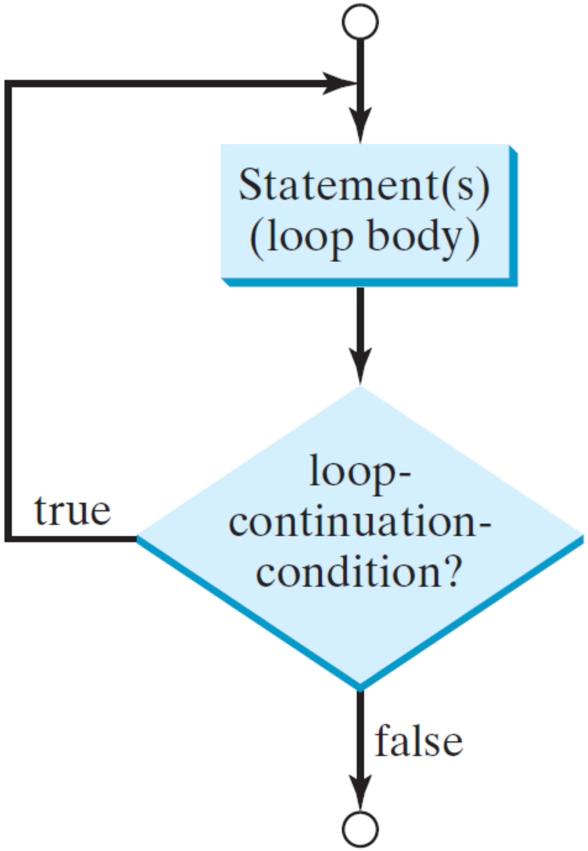
Such a value is known as a *sentinel value*.

Write a program that reads and calculates the sum of an unspecified number of integers (e.g., the sum of 2, 3, 5, 7, 11…). The input 0 signifies the end of the input.

See SentinelValue.java.

do-while Loop

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



Example: TestDoWhile.java

- The loop body is executed
- The loop condition - boolean expression - is evaluated
- If the loop condition is true, then loop body is executed again
- If the loop condition is false, control is transferred to the statement following the loop

Do…While Loop Rules (same as while loop)

- The loop condition must be a boolean expression
  - Boolean expression must be in parentheses
  - Boolean expression is formed using relational and logical operators
- Loop condition
  - Generally, some statement before the while loop "initializes" the loop condition to true
  - Some statement within the loop body must eventually change the condition to false
- If the condition is never changed to false, the program will be forever stuck in the loop
  - This is called an "infinite loop"
- Curly braces are not necessary if only one statement in loop but best practice is to always include curly braces

Note

- Recall how placing a semicolon at the end of the while-clause creates an infinite loop

```
int iteration = 1;
while (iteration <= 10); {   // Unnecessary semicolon
        System.out.println("Iteration  = " + iteration);
        iteration++;
}
```

Note

- In the case of do-while you must include the semicolon since it ends the loop!

```
int iteration = 1;
do {
        iteration++;
        System.out.println("Iteration = " + iteration);
} while (iteration <= 5); // Necessary semicolon
```

Loop Design Strategies

- Four steps when writing a loop.
  - Step 1: Identify what statements need to be repeated
  - Step 2: Wrap these statements in a loop using while or do...while:

    -
      ```
      while (true) {
        Statements;
      }
      ```
  - Step 3: Determine what condition the code should check (replace true)
  - Step 4: Add code in the body that eventually causes the condition to become false

    -
      ```
      while (loop-continuation-condition) {
      Statements;
      Additional statements for controlling the loop;
      }
      ```

Example: Powers.java

Summary

- While loop
- Do…while loop