

Java I/O and Control Structures

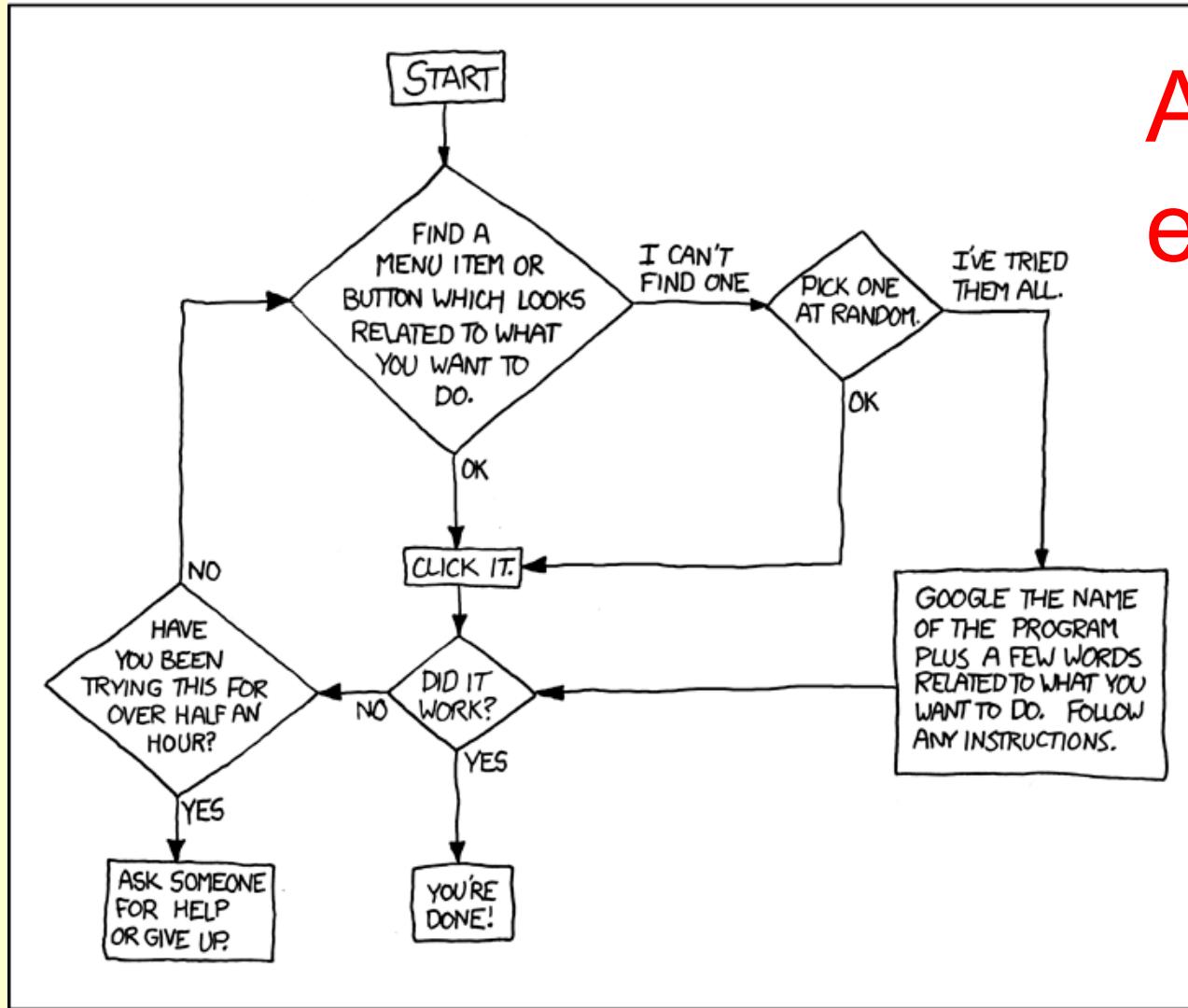
CSC 2014 – Java Bootcamp

Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Some slides in this presentation are adapted from the slides accompanying Java Software Solutions by Lewis & Loftus

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS,
AND OTHER "NOT COMPUTER PEOPLE."

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY
PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



Algorithms in everyday life

Source: <http://xkcd.com/627/>

PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN.
CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

Algorithms



An algorithm is a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point. Specific algorithms sometimes also go by the name method, procedure, or technique. The word "algorithm" is a distortion of al-Khwārizmī, a Persian mathematician who wrote an influential treatise about algebraic methods.

Sources: <http://mathworld.wolfram.com/Algorithm.html> and Wikipedia (http://en.wikipedia.org/wiki/Mu%E1%B8%A5ammad_ibn_M%C5%ABs%C4%81_al-Khw%C4%81rzm%C4%AB)

Algorithm Example: ***Input-Compute-Output*** pattern

GPA problem: Write a program that computes and outputs the GPA, given the credits and quality points earned.

Pseudocode: describe steps in simple, unambiguous language

Variables: qp, credits, gpa

Algorithm:

1. Input qp
2. Input credits
3. $\text{gpa} = \text{qp} / \text{credits}$
4. Print gpa

Java Program →

Variables: qp, credits, gpa

Algorithm:

1. Input qp
2. Input credits
3. $\text{gpa} = \text{qp} / \text{credits}$
4. Print gpa

Next: A closer look at input in Java

```
// GPA.java      Author: Joyce/Papalaskari
// Demonstrates the use of Scanner input and s
//*****
import java.util.Scanner;

public class GPA
{
    public static void main (String[] args)
    //-----
    // Inputs the quality points and credits and
    //-----
    {
        double qp, credits, gpa;

        Scanner scan = new Scanner(System.in);

        // get input
        System.out.print ("Enter Quality Points > ");
        qp = scan.nextInt();
        System.out.print ("Enter Credits > ");
        credits = scan.nextInt();

        // output information entered
        System.out.println ("\nQuality Points: " + qp);
        System.out.println ("Credits: " + credits);

        // calculate and output GPA
        gpa = qp / credits;
        System.out.println ("\n\tGPA: " + gpa);
    }
}
```

Interactive Programs – Input/Output

- Programs can use data obtained during runtime, eg:

```
int age;
```

```
String name;
```

```
System.out.print("Enter your name");  
name = scan.nextLine();
```

output method

input method

```
System.out.print("Enter your age");  
age = scan.nextInt();
```

Interactive Programs – Input/Output

- In Java, you first need to create a Scanner object

```
int age;
```

```
String name;
```

```
Scanner scan = new Scanner(System.in);
```

```
System.out.print("Enter your name");
```

```
name = scan.nextLine();
```

Scanner object

input method (for String)

```
System.out.print("Enter your age");
```

```
age = scan.nextInt();
```

input method (for int)

Reading Input

- The Scanner class is part of the `java.util` class library, and must be imported into a program in order to be used
- The import statement goes at beginning of your program (above class definition)

```
import java.util.Scanner;
```

Using the Scanner class

1. import the class, i.e., add this before the class definition of your program:

```
import java.util.Scanner;
```

2. In your main method, before doing any input, declare and initialize the Scanner object

```
Scanner scan = new Scanner(System.in);
```

3. Input away!

```
System.out.print("Enter your name");  
name = scan.nextLine();
```

```
System.out.print("Enter your age");  
age = scan.nextInt();
```

Example

```
import java.util.Scanner;

public class TellMeAboutYou
{
    public static void main(String[ ] args)
    {
        int age;
        String name;

        Scanner scan = new Scanner(System.in);

        System.out.print("Enter your name");
        name = scan.nextLine();

        System.out.print("Enter your age");
        age = scan.nextInt();

        System.out.println("Pleased to meet you, " + name + "!");
        System.out.println("Your age in dog years: " + age*10.5);
    }
}
```

Enter your name: **Fiona**
Enter your age: **17**
Pleased to meet you, Fiona!
Your age in dog years is 178.5

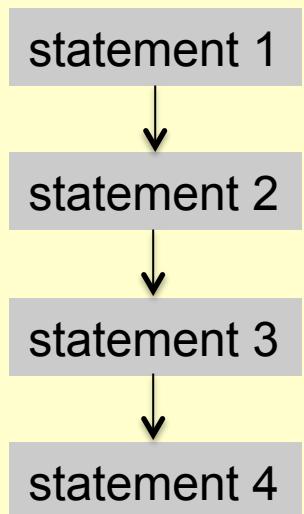
Inspired by: <http://www.onlineconversion.com/dogyears.htm>

Input methods

- `nextInt()` → input an `int`
- `nextDouble()` → input a `double`
- `nextLine()` → input a `String` (until end of line)
- `next()` → input a `String` **token** (one word or other delimited “chunk” of text)
- *White space (space, tab, new line) are used to separate input tokens*

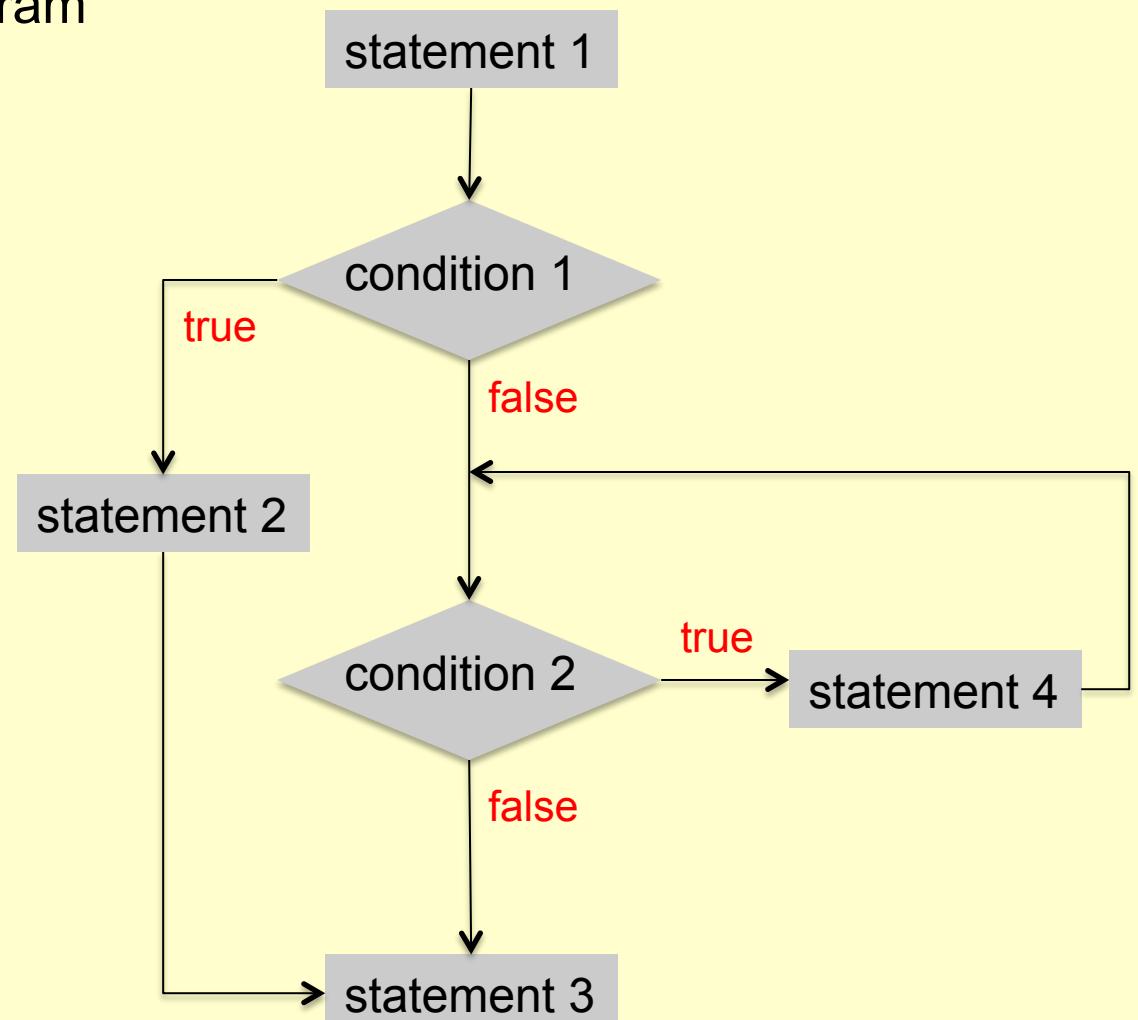
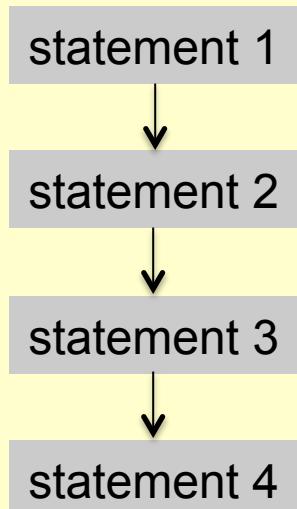
Control flow

- Sequence of statements that are actually executed in a program



Control flow

- Sequence of statements that are actually executed in a program



This slide adapted from Wayne&Sedgewick Princeton course: <http://www.cs.princeton.edu/courses/archive/spring13/cos126/lectures.php>

CSC 1051 M.A. Papalaskari, Villanova University

Java Conditional statements

alter the linear flow of control. They use *boolean expressions* to determine what to do next.

Example:

```
if (credits == 0)  
    System.out.println ("GPA: None") ; ← if true, do this  
  
else  
{  
    if more than one statement,  
    enclose in braces  
    gpa = qp / credits;  
    System.out.println ("\n\tGPA: " + gpa) ;  
    if false, do these  
}
```

Java relational operators

- relational operators can be used with numeric types and produce **boolean** results:

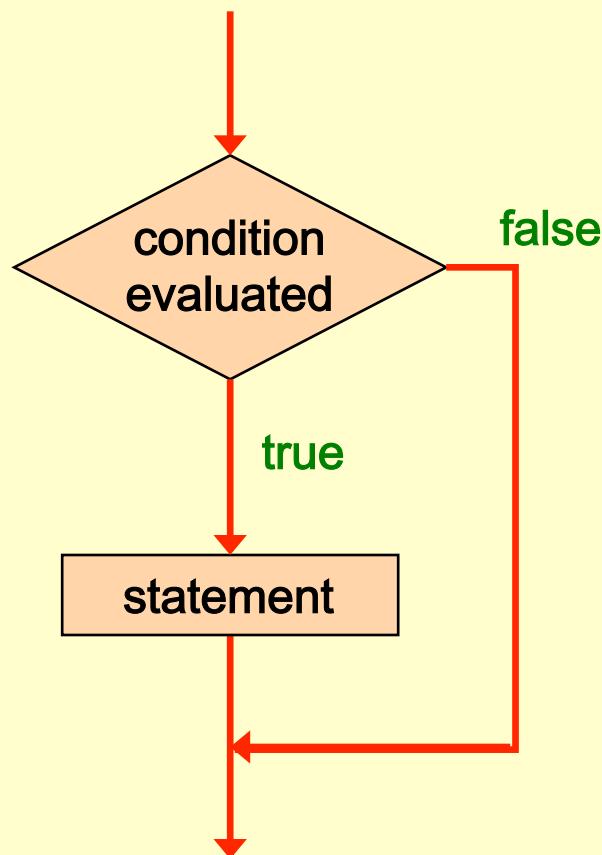
<code>==</code>	equal to
<code>!=</code>	not equal to
<code><</code>	less than
<code>></code>	greater than
<code><=</code>	less than or equal to
<code>>=</code>	greater than or equal to

- Note the difference between the equality operator (`==`) and the assignment operator (`=`)

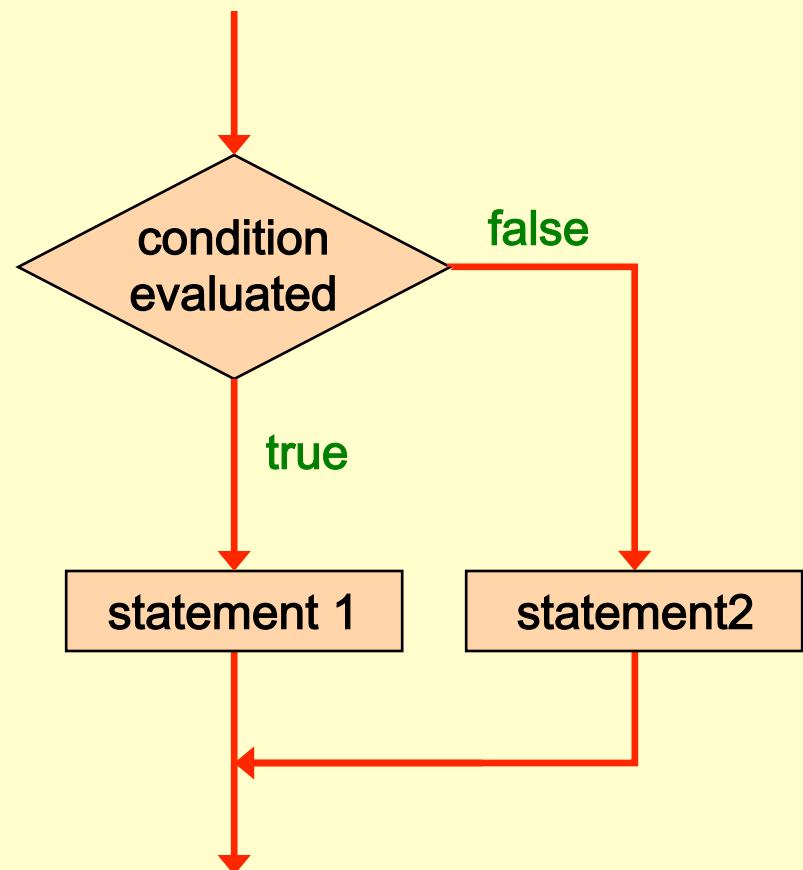
Conditional statements

```
if ( condition )
    statement;
```

// no else clause



```
if ( condition )
    statement1;
else
    statement2;
```



Example:

How do we fix output to use singular/plural as appropriate?

For example:

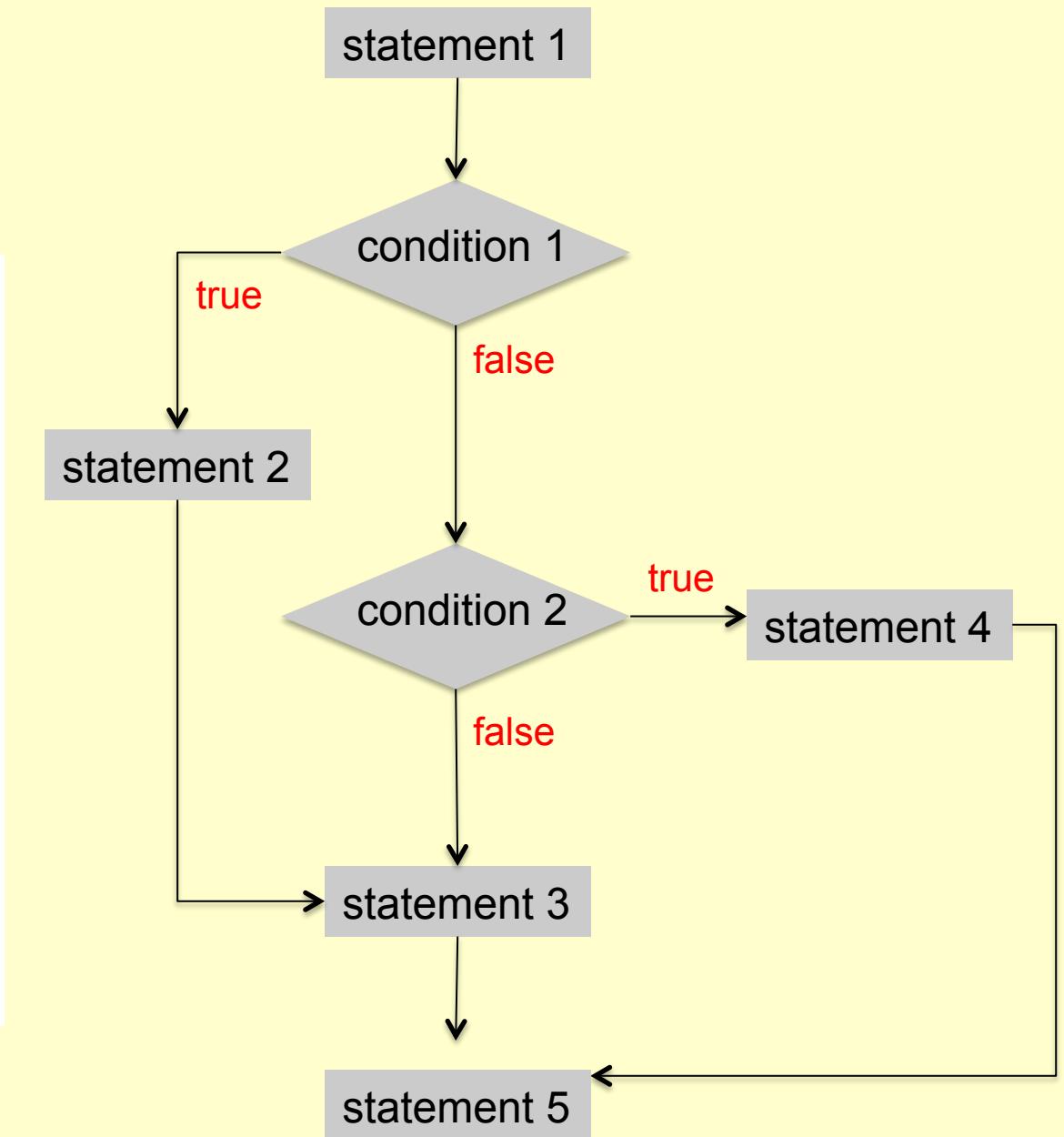
Enter the total amount to be given as change: **18**

That amount can be given as:

0 quarters
1 dime**get rid of this!**
1 nickel
3 pennies

Nested loops

```
statement 1;  
if (condition 1)  
    statement 2;  
else  
    if (condition 2)  
        statement 4;  
    else  
        statement 3;  
statement 5;
```



This slide adapted from Wayne&Sedgewick Princeton course: <http://www.cs.princeton.edu/courses/archive/spring13/cos126/lectures.php>

CSC 1051 M.A. Papalaskari, Villanova University

Another example:

Create an application called **Vacation** that prompts for and inputs an integer representing someone's age and then suggests an appropriate vacation destination. One of three destinations should be suggested depending on whether the answer is less than 20, between 20 and 50, or over 50.

For example, a run of the program might look like this:

```
How old is the traveler > 15
You should consider Hershey Park.
```

Java Logical Operators

- logical operators can be used with **boolean** operands to express more complex **boolean** conditions:

!	Logical NOT
&&	Logical AND
	Logical OR

Vacation example revisited:

Create an application called **Vacation** that prompts for and inputs an integer representing someone's age and then suggests an appropriate vacation destination. One of three destinations should be suggested depending on whether the answer is less than 20, between 20 and 50, or over 50.

For example, a run of the program might look like this:

```
How old is the traveler > 15
You should consider Hershey Park.
```

Boolean Expressions

- The reserved words `true` and `false` are the only valid values for a `boolean` type
- Example: `boolean` variables:

```
boolean aboveAgeLimit = false;
```

```
boolean usePlural = hours > 1;
```

boolean expression using a *relational operator*

Example

```
if (total < MAX+5 && !found)
    System.out.println ("Processing...");
```

total 103

MAX 100

found false

- All logical operators have lower precedence than the relational operators
- The `!` operator has higher precedence than `&&` and `||`

Logical NOT

- The *logical NOT* operation is also called *logical negation* or *logical complement*
- If some boolean condition a is true, then $\neg a$ is false; if a is false, then $\neg a$ is true
- Logical expressions can be shown using a *truth table*:

a	$\neg a$
true	false
false	true

Logical AND and Logical OR

- The *logical AND* expression

a && b

is true if both a and b are true, and false otherwise

- The *logical OR* expression

a || b

is true if a or b or both are true, and false otherwise

Logical AND and Logical OR

- A truth table shows all possible true-false combinations of the terms
- Since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`

a	b	<code>a && b</code>	<code>a b</code>
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Quick Check 1

What do the following statements do?

```
if (total != stock + warehouse)  
    inventoryError = true;
```

total 20

stock 8

warehouse 12

inventoryError false

```
if (found || !done)  
    System.out.println("Ok");
```

found false

done true

Quick Check 2

Try again with different values

```
if (total != stock + warehouse)  
    inventoryError = true;
```

total 20

stock 7

warehouse 12

inventoryError false

```
if (found || !done)  
    System.out.println("Ok");
```

found false

done false

Quick Check 3

Try again with different values

```
if (total != stock + warehouse)  
    inventoryError = true;
```

total 20

stock 8

warehouse 12

inventoryerror true

```
if (found || !done)  
    System.out.println("Ok");
```

found true

done false

Boolean Expressions

- using truth tables – let's try this one:

found	done	!done	found !done
false	false		
false	true		
true	false		
true	true		

Boolean Expressions

- using truth tables – another example:

total < MAX	found	!found	total < MAX && !found
false	false		
false	true		
true	false		
true	true		

How much of a boolean expression do we need to evaluate before determining its value?

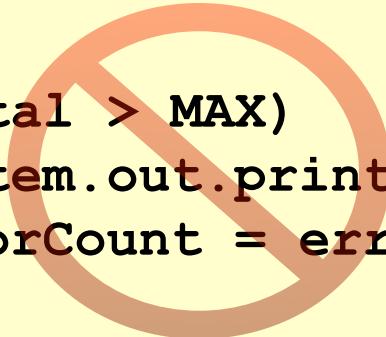
*** Short-Circuited Operators

- The processing of `&&` and `||` is “short-circuited” in cases where the left operand is sufficient to determine the result (the right operand is not evaluated at all)
- This can be both useful and dangerous!

```
if (count != 0 && total/count > MAX)
    System.out.println ("Testing.");
```

Indentation Revisited

- Remember that indentation is for the human reader, and is ignored by the computer



```
if (total > MAX)
    System.out.println ("Error!!");
    errorCount = errorcount + 1;;
```

Despite what is implied by the indentation, the increment will occur whether the condition is true or not

Selection structures in Java

- Conditional statement:

```
if (n > 0)
    System.out.print("positive");
else
    System.out.print("negative");
```

- Other selection structures (Chapter 6 in text)
 - the **conditional** operator
 - the **switch** statement

The Conditional Operator Syntax

condition ? expression1 : expression2

- If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated
- The value of the entire conditional operator is the value of the selected expression
- Example: Rewrite this →

```
if (n > 0)
    System.out.print("positive");
else
    System.out.print("negative");
```

The conditional operator is not a statement

WRONG!

~~(n > 0) ? System.out.print("positive"): System.out.print("negative");~~

More examples:

```
int bit = (ans == 'Y')? 1: 0;
```

```
String status = (age < 18 ? "child" : "adult");
```



- The conditional operator requires **three** operands so it is sometimes called the ***ternary*** operator

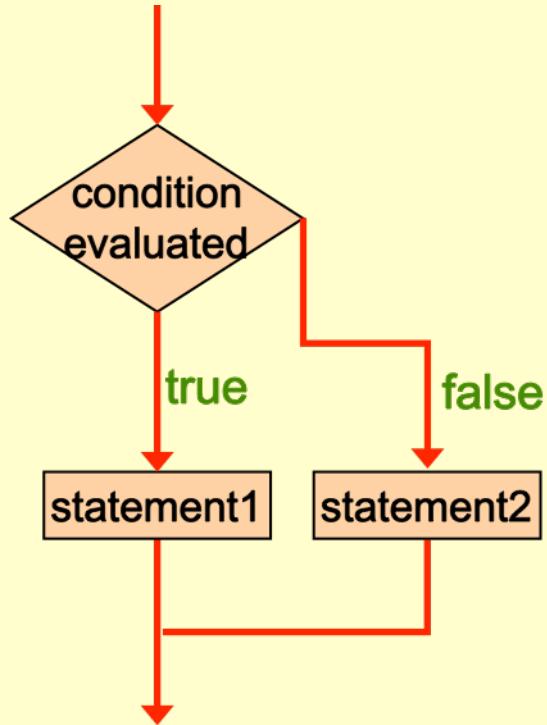
Quick Check

```
System.out.println ("Your change is " +  
                    num + "Dimes"));
```

Rewrite this statement so that "Dime" is printed if
num equals 1.

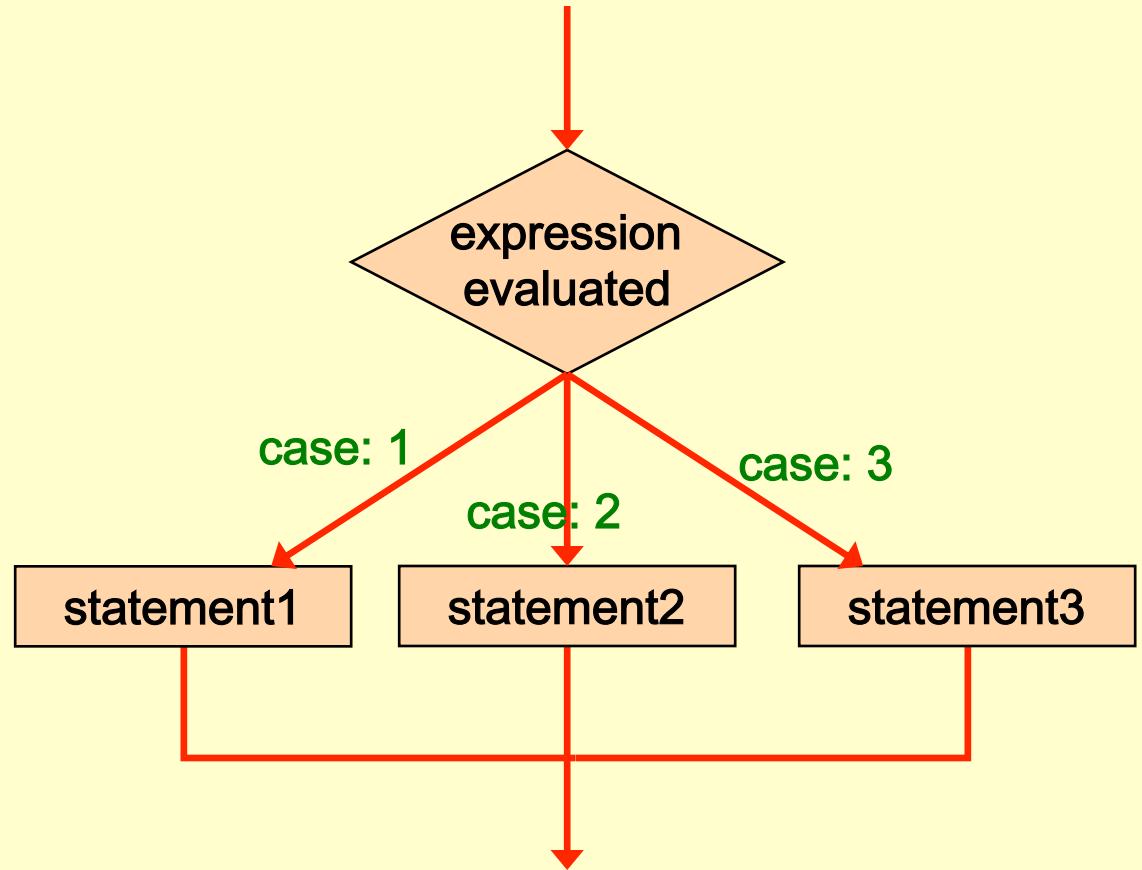
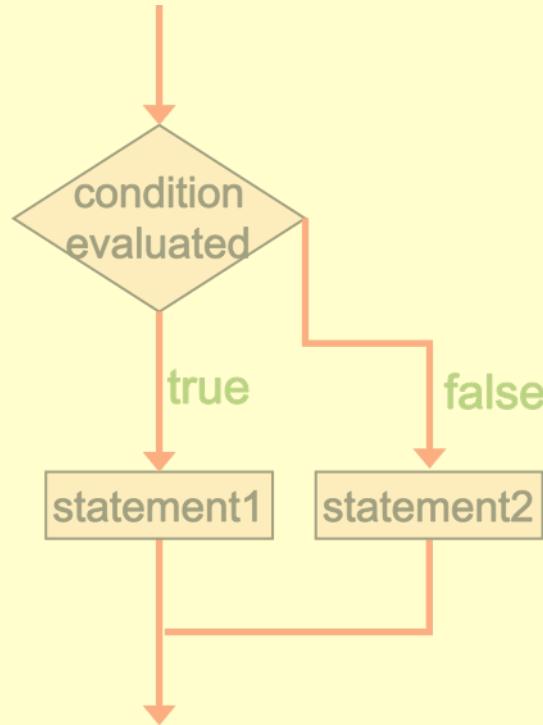
The `switch` Statement: multi-way branches

Recall: Logic of an if-else statement



The **switch** Statement: multi-way branches

switch statement logic



Note: this is a simplified flowchart of the logic of the switch statement

The switch Statement - example

```
public String toString()
{
    String result = "";
    switch (faceValue)
    {
        case 1:
            result = "one";
            break;
        case 2:
            result = "two";
            break;
        case 3:
            result = "three";
            break;
    }
    return result;
}
```

The switch Statement in general

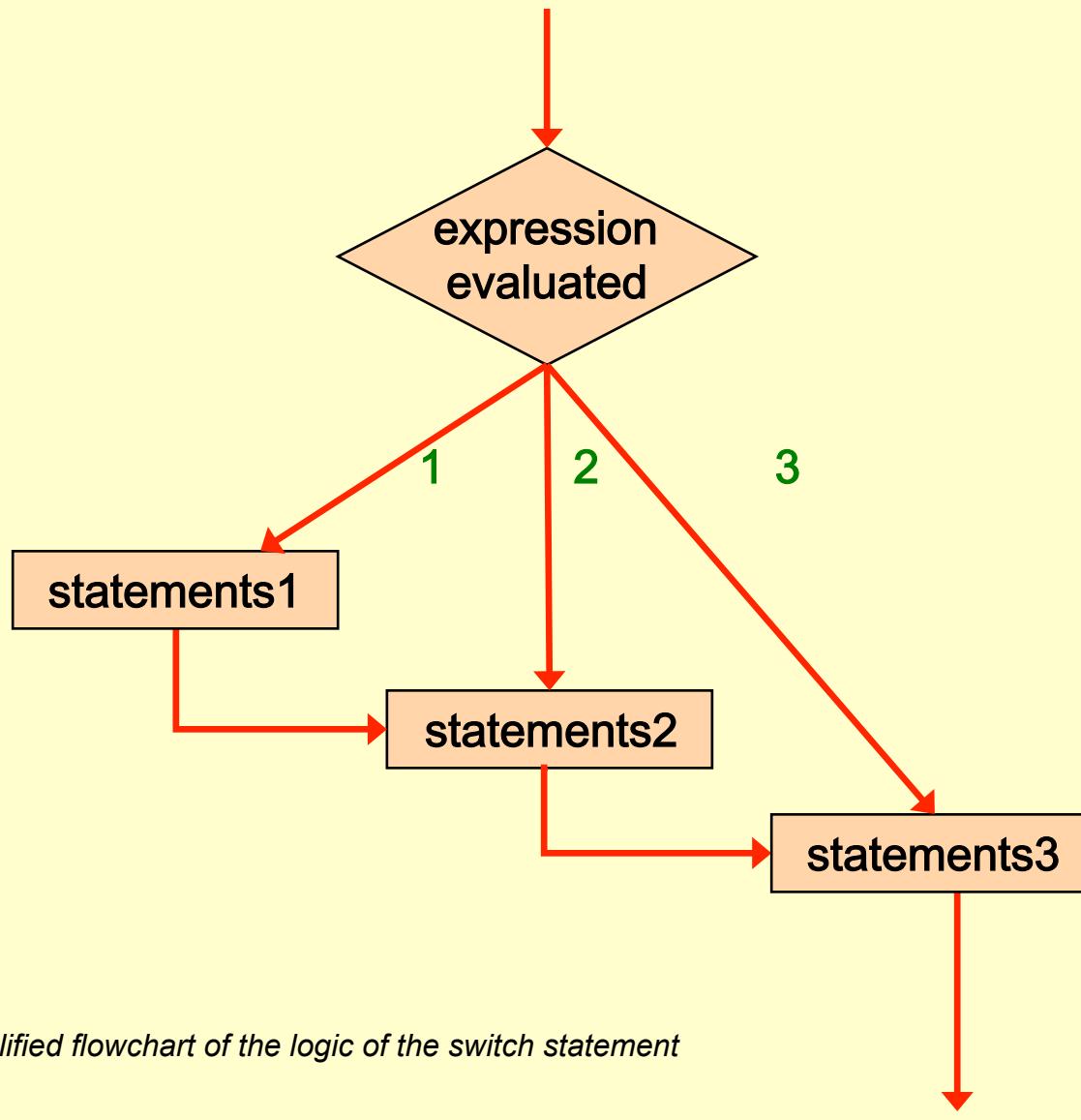
```
switch ( expression )  
{  
    case value1 :  
        statement-list1  
    case value2 :  
        statement-list2  
    case value3 :  
        statement-list3  
    .  
    .  
    .  
    default:  
        statement-list-n  
}
```

integer, char, or enumerated types
(Java 7 also allows Strings)
NO floating point values
NO ranges of values (eg: $0 < x < 10$)

If **expression**
matches **value2**,
control jumps
to here

If none of the values
match the **expression**,
control jumps to here

So... the logic of the switch is more like this:



Note: this is still a simplified flowchart of the logic of the switch statement

GradeReport.java

```
//*****  
// GradeReport.java      Author: Lewis/Loftus  
//  
// Demonstrates the use of a switch statement.  
//*****  
  
import java.util.Scanner;  
  
public class GradeReport  
{  
    //-----  
    // Reads a grade from the user and prints comments accordingly.  
    //-----  
    public static void main (String[] args)  
    {  
        int grade, category;  
  
        Scanner scan = new Scanner (System.in);  
  
        System.out.print ("Enter a numeric grade (0 to 100): ");  
        grade = scan.nextInt();  
  
        category = grade / 10;  
  
        System.out.print ("That grade is ");  
  
        continue  
    }  
}
```

Sample Run

continue

```
switch (category)
{
```

```
    case 10:
        System.out.println ("a perfect score. Well done.");
        break;
    case 9:
        System.out.println ("well above average. Excellent.");
        break;
    case 8:
        System.out.println ("above average. Nice job.");
        break;
    case 7:
        System.out.println ("average.");
        break;
    case 6:
        System.out.println ("below average. You should see the");
        System.out.println ("instructor to clarify the material "
                           + "presented in class.");
        break;
    default:
        System.out.println ("not passing.");
}
```

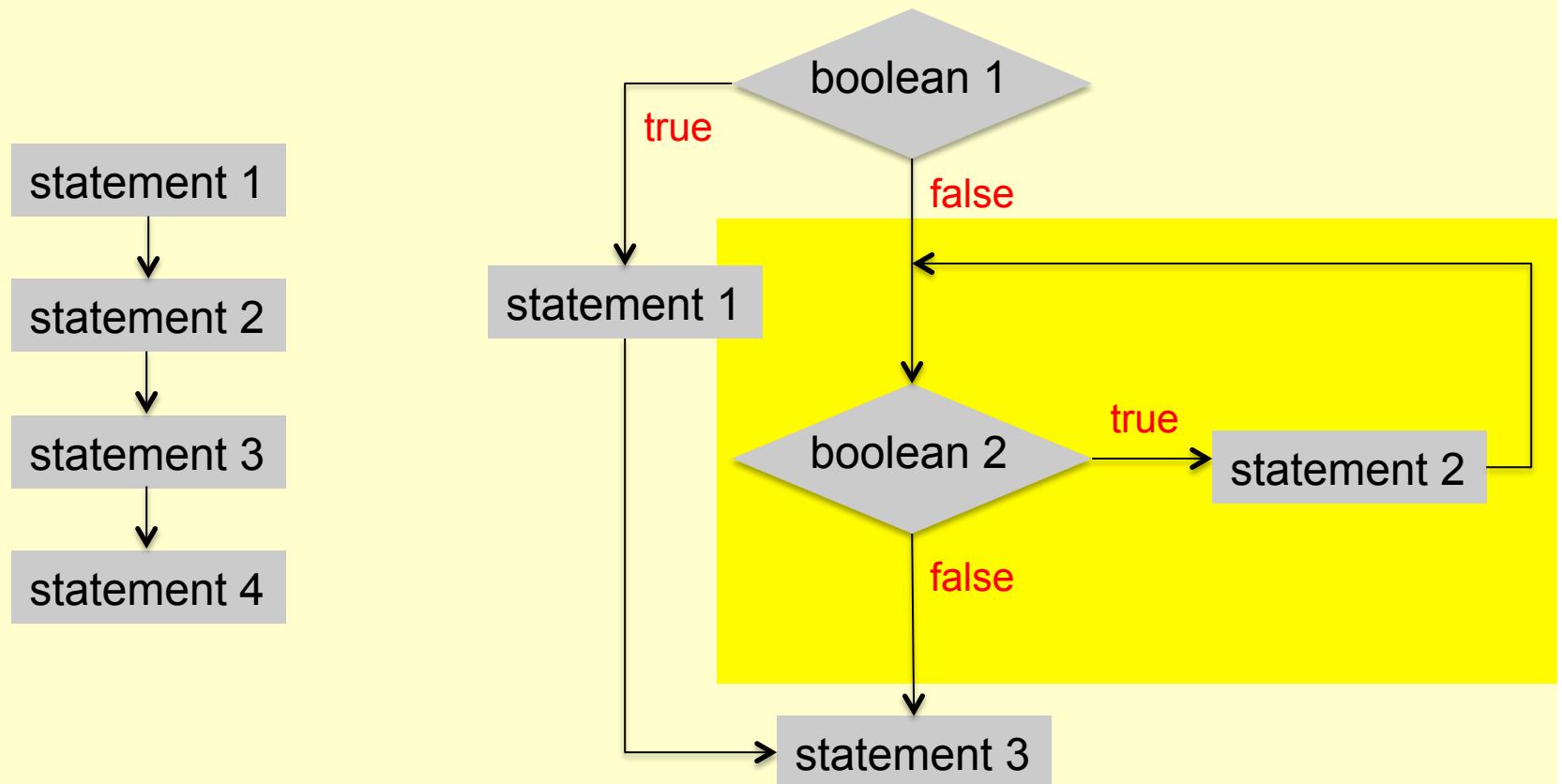
Enter a numeric grade (0 to 100): 91

That grade is well above average. Excellent.

Hands on: try removing one of the break statements

Control flow: Repetition

- Sequence of statements that are actually executed in a program
- Conditional and Repetition statements: enable us to alter control flow



This slide adapted from Doug Clark's course <http://www.cs.princeton.edu/courses/archive/spring13/cos126/lectures.php>

CSC 1051 M.A. Papalaskari, Villanova University

Example

- Investment problem: You put \$10,000 into a bank account that earns 5% interest per year.

year	interest	balance
0		\$10,000.00
1	\$500.00	\$10,500.00
2	\$525.00	\$11,025.00
3	\$551.25	\$11576.25
4	\$578.81	\$12,155.06

- ... How many years does it take for the account balance to be double the original?

Example

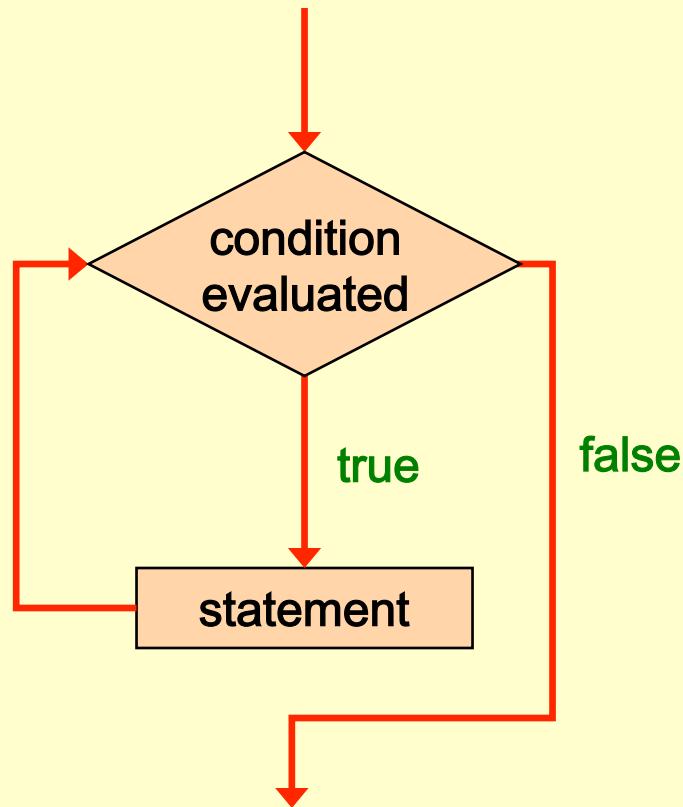
- Investment problem: You put \$10,000 into a bank account that earns 5% interest per year. How many years does it take for the account balance to be double the original?
- Algorithm:

The while Statement

- A *while statement* has the following syntax:

```
while ( condition )
      statement;
```
- If the *condition* is true, the *statement* is executed
- Then the condition is evaluated again, and if it is still true, the *statement* is executed again
- The *statement* is executed repeatedly until the *condition* becomes false

Logic of a while Loop



Example

- A counting loop that prints the numbers 1, 2, 3, ...

Algorithm:

- initialize a counter to 1
- while the counter \leq upper limit
 - print counter
 - increment counter

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count
1

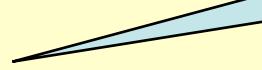
Initialize count

The while Statement

count
1

```
int count = 1;  
while (count <= 3) {  
    System.out.println (count);  
    count++;  
}
```

count <= 3 is true

A blue rounded rectangle contains the text "count <= 3 is true". A blue arrow originates from the bottom right corner of this box and points towards the "count" variable in the first line of code.

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count
1

Print count

Output:

1

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

Increment count

Output:

1

count
2

The while Statement

count
2

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count <= 3 is true

Output:

1

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count
2

Print count

Output:

1
2

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

Increment count

Output:

1
2

count
3

The while Statement

count
3

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count <= 3 is true

Output:

1

2

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count
3

Print count

Output:

1
2
3

The while Statement

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

Increment count

Output:

1
2
3

count
4

The while Statement

count
4

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println (count);  
    count++;  
}
```

count <= 3 is false

Output:

1
2
3

The while Statement “unraveled”

```
int count = 1;  
while (count <= 3)  
{  
    System.out.println(count);  
    count++;  
}
```

Output:
1
2
3

int count = 1;
TEST: (count <= 3) ➔ true

{

System.out.println(count);
count++;

}

TEST: (count <= 3) ➔ true

{

System.out.println(count);
count++;

}

TEST: (count <= 3) ➔ true

{

System.out.println(count);
count++;

}

TEST: (count <= 3) ➔ false

count
1

count
2

count
3

count
4

EXIT LOOP

ity

Example

- Table of squares and cubes:

N	N^2	N^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

Example

- Table of powers: Compute the powers of 2 and the powers of 3 and print a table like this:

N	2^N	3^N
0	1	1
1	2	3
2	4	9
3	8	27
4	16	81

What's wrong with this code?

```
int count = 1;  
while (count <= 10)  
    System.out.println (count) ;  
    count++ ;
```

What's wrong with this code?

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

If the condition of a while loop is false initially, the statement is never executed

```
System.out.println("input a number >5");
int num = scan.nextInt();

// input validation
while (num <= 5)
{
    System.out.println ("type a number >5");
    num = scan.nextInt();
}
```

- Therefore, the body of a while loop will execute zero or more times

Nested loops

Example: Investment problem repetition

→ the repeated action (calculating the number of years it takes for investment to double) involves repetition

General pattern for algorithms: A nested loop

while (condition for repeating action)

 initialize variables

while (condition for reaching goal)

 calculations

 print results

Quick Check

How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println ("Here");
        count2 = count2++;
    }
    count1 = count1++;
}
```

More repetition structures: do & for loops

Repetition structure in Java, so far: **while** loop

```
int count = 0;  
while (count < 5)  
{  
    System.out.println (count);  
    count++;  
}
```

- Other repetition structures (Chapter 6 in text)
 - the **do** loop
 - the **for** loop

The **do** Statement in Java

- A **do** statement has the following syntax:

```
do
{
    statement-list;
} while (condition); //end do
```

- The ***statement-list*** is executed once initially, and then the ***condition*** is evaluated
- The ***statement-list*** is executed repeatedly until the condition becomes **false**

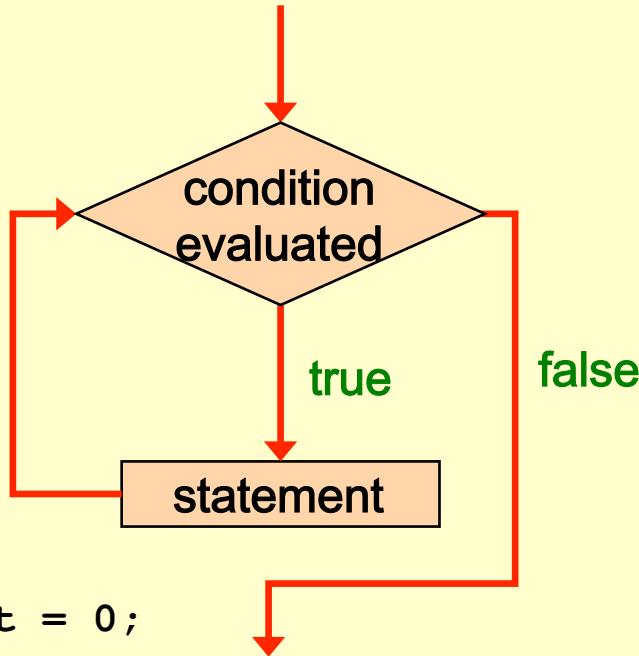
The **while** and **do** loops are similar.

```
int count = 0;
while (count < 5)
{
    System.out.println (count);
    count++;
}
```

```
int count = 0;
do
{
    System.out.println (count);
    count++;
} while (count < 5);
```

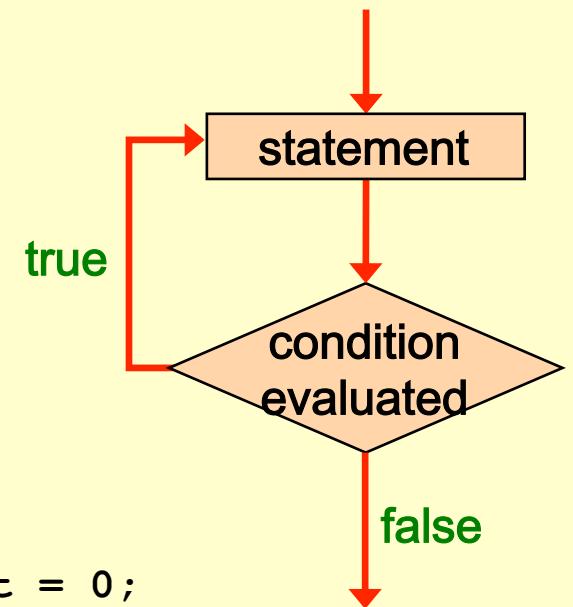
Similar – but not the same:

while Loop



```
int count = 0;  
while (count < 5)  
{  
    System.out.println (count);  
    count++;  
}
```

do Loop



```
int count = 0;  
do  
{  
    System.out.println (count);  
    count++;  
} while (count < 5);
```

- The body of a **do** loop executes at least once

For some things the **do** loop is more appropriate:

```
System.out.println("input a number >5");
int num = scan.nextInt();

while (num <= 5)
{
    System.out.println ("type a number >5");
    num = scan.nextInt();
}
```

```
do
{
    System.out.println ("type a number >5");
    num = scan.nextInt();
} while (num <= 5)
```

input validation

CSC 1051 M.A. Papalaskari, Villanova University

For some things the **do** loop is more appropriate:

```
boolean more = true;  
while (more)  
{  
    System.out.print("Enter Quality Points ");  
    qp = scan.nextInt();  
  
    System.out.print ("Enter Credits ");  
    credits = scan.nextInt();  
  
    gpa = (double) qp /credits;  
    System.out.println("GPA = " + gpa);  
    System.out.print ("Again? 1=yes, 0=no ");  
    more = (1 == scan.nextInt());  
}  
  
System.out.println("Thank you. Goodbye. ");
```

```
do {  
    System.out.print("Enter Quality Points ");  
    qp = scan.nextInt();  
  
    System.out.print ("Enter Credits ");  
    credits = scan.nextInt();  
  
    gpa = (double) qp /credits;  
    System.out.println("GPA = " + gpa);  
    System.out.print ("Again? 1=yes, 0=no ");  
} while (1 == scan.nextInt())  
System.out.println("Thank you. Goodbye. ");
```

repeating a computation

Another example: ReverseNumber.java

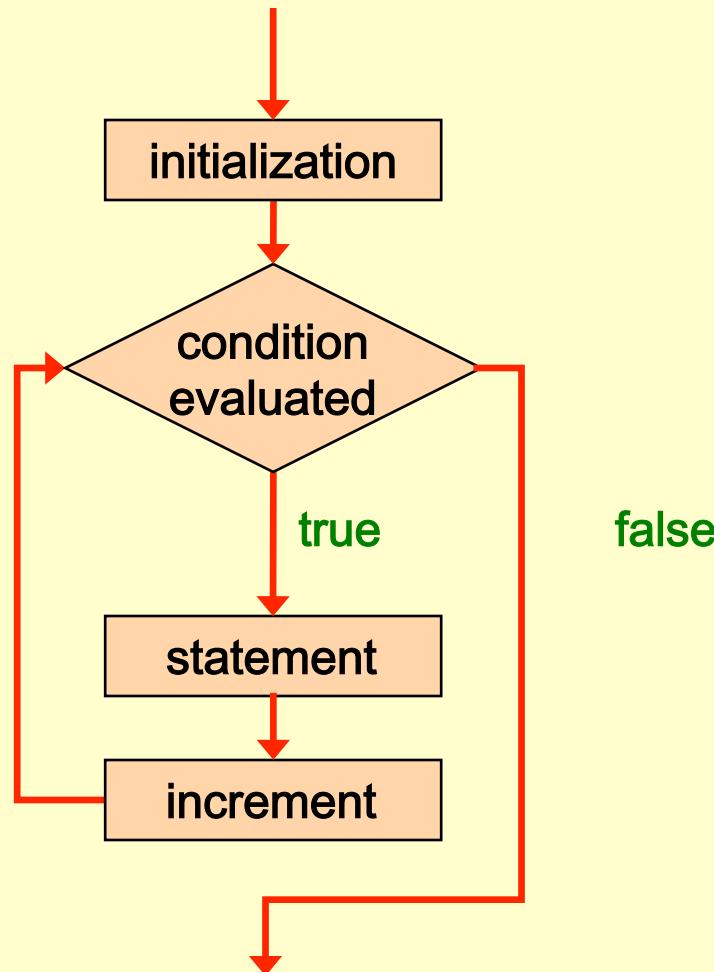
```
//*****
//  ReverseNumber.java    Author: Lewis
//  Demonstrates the use of a do loop.
//*****
import java.util.Scanner;
public class ReverseNumber
{
    //-----
    // Reverses the digits of an integer mathematically.
    //-----
    public static void main (String[] args)
    {
        int number, lastDigit, reverse = 0;
        Scanner scan = new Scanner (System.in);
        System.out.print ("Enter a positive integer: ");
        number = scan.nextInt();
        do
        {
            lastDigit = number % 10;
            reverse = (reverse * 10) + lastDigit;
            number = number / 10;
        }
        while (number > 0);

        System.out.println ("That number reversed is " + reverse);
    }
}
```

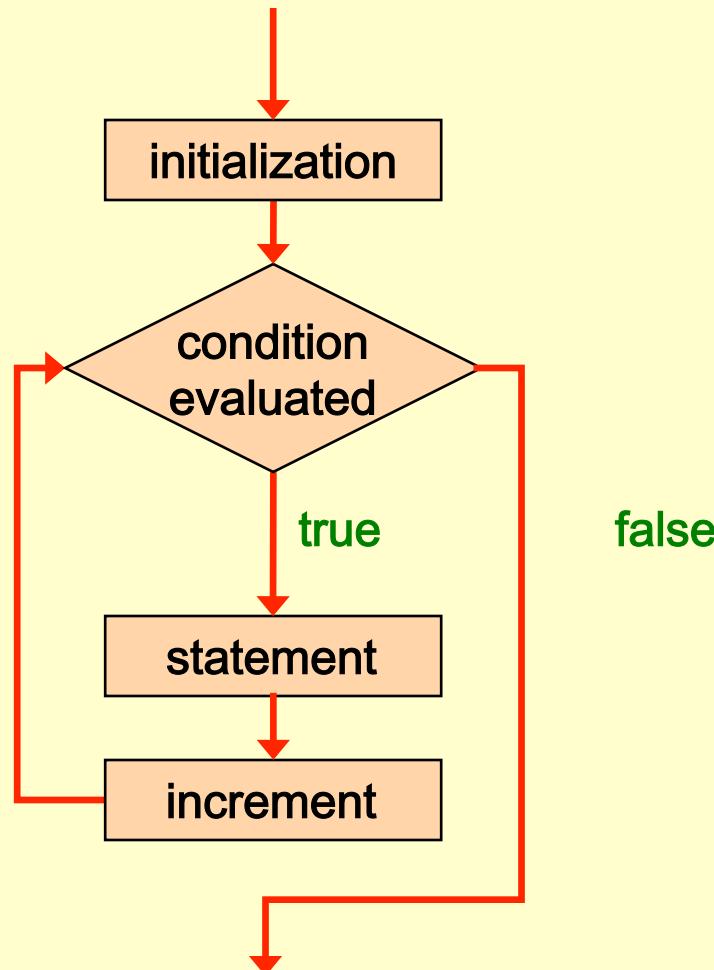
Sample Run

Enter a positive integer: 2896
That number reversed is 6982

for: a loop with built in “counter”



for: a loop with built in “counter”



```
int count = 0;
```

Example

```
while (count < 5)
```

```
{
```

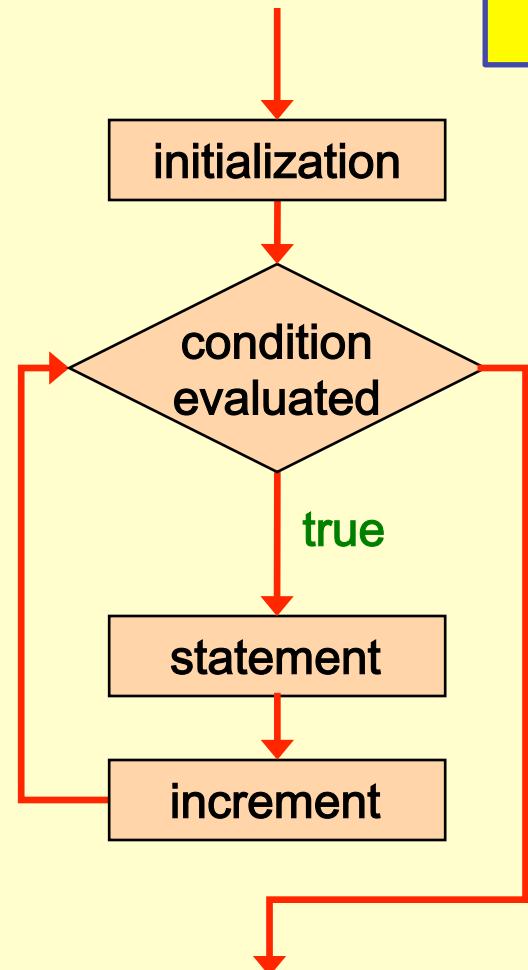
```
    System.out.println (count);
```

```
    count++;
```

```
}
```

for: a loop with built in “counter”

```
for (int count = 0; count < 5; count++)  
    System.out.println (count);
```



```
int count = 0;
```

Example

```
while (count < 5)
```

```
{
```

```
    System.out.println (count);
```

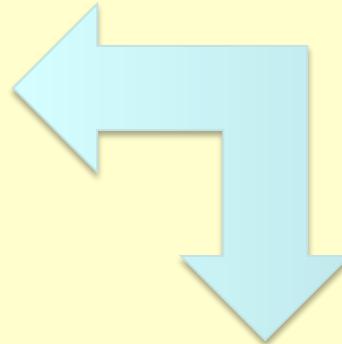
```
    count++;
```

```
}
```

The **for** Statement

- A **for** loop is functionally equivalent to the following while loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```



```
for ( initialization ; condition ; increment )
    statement;
```

The for Statement

- A *for statement* has the following syntax:

The *initialization* is executed once before the loop begins

```
for ( initialization ; condition ; increment )  
    statement;
```

The *statement* is executed until the *condition* becomes false

The *increment* portion is executed at the end of each iteration

The for Statement

- A *for statement* has the following syntax:

The *initialization* is executed once before the loop begins

```
for (int count = 0; count < 5; count++)  
    System.out.println (count);
```

The *statement* is executed until the *condition* becomes false

The *increment* portion is executed at the end of each iteration

The for Statement

- The increment section can perform any calculation:

```
for (int num=100; num > 0; num -= 5)  
    System.out.println (num) ;
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance

Example: Stars.java

```
//*****  
// Stars.java          Author: Lewis/Loftus  
//  
// Demonstrates the use of nested for loops.  
//*****  
  
public class Stars  
{  
    //-----  
    // Prints a triangle shape using asterisk (star) characters.  
    //-----  
    public static void main (String[] args)  
    {  
        final int MAX_ROWS = 10;  
  
        for (int row = 1; row <= MAX_ROWS; row++)  
        {  
            for (int star = 1; star <= row; star++)  
                System.out.print ("*");  
  
            System.out.println();  
        }  
    }  
}
```

Output

```
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

Exercise: can you make it print the row number in (1, 2, 3...) at the beginning of each line?