# CS 1233 Object Oriented and Design

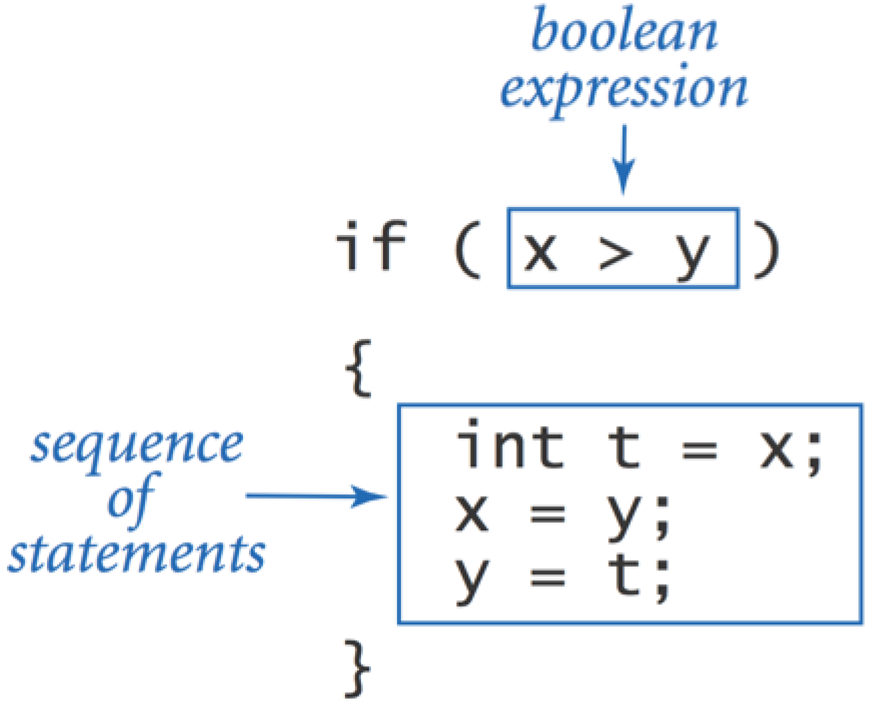
LIPSCOMB UNIVERSITY | COLLEGE OF COMPUTING & TECHNOLOGY

Created by Eddy C. Borera, PhD

## If Statement

The if statement. A common branching structure.

- Evaluate a boolean expression.
- If true, execute some statements.
- If false, execute other statements.



```
if ( x > y )
{
   int t = x;
   x = y;
   y = t;
}
```

## If Statement

```
if (x > y) {
    x = 10;
}
```

One line statement (brackets are optional)

```
if (x > y)
    x = 10;
```

## If … else

```
// Statements that execute before the branches

if (expression) {
   // Statements to execute when the expression is true
}
else {
   // Statements to execute when the expression is false
}

// Statements that execute after the branches
```

## Example

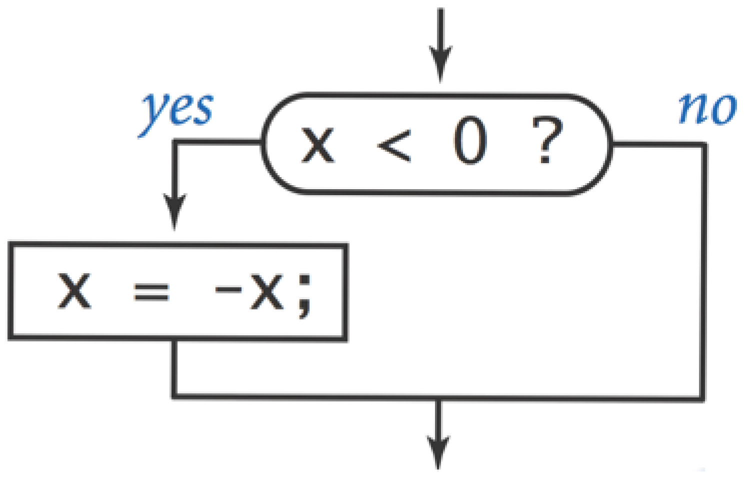
```
...

if (userAge < 25) {
     insurancePrice = PRICE_LESS_THAN_25;
     System.out.println("(executed first branch)");
}
else {
     insurancePrice = PRICE_25_AND_UP;
     System.out.println("(executed second branch)");
}

...
```

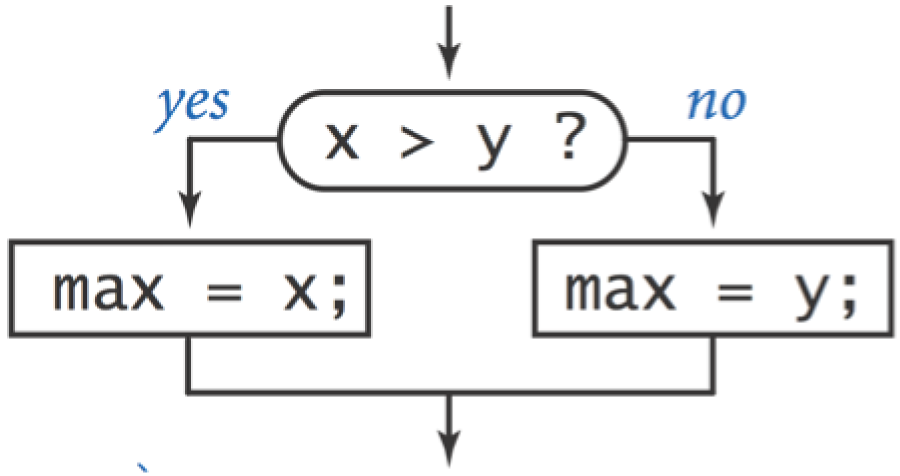
## Example

```
if (x < 0) x = -x;
```



## Example (2)

```
if (x > y) max = x;
else       max = y;
```



## Nested If Statement

What's wrong with the following for income tax calculation?

| Income | Rate |
|---|---|
| 0 - 47,450 | 22% |
| 47,450 – 114,650 | 25% |
| 114,650 – 174,700 | 28% |
| 174,700 – 311,950 | 33% |
| 311,950 - | 35% |

```
double rate = 0.35;
if (income <  47450) rate = 0.22;
if (income < 114650) rate = 0.25;
if (income < 174700) rate = 0.28;
if (income < 311950) rate = 0.33;
```

| Income | Rate |
|---|---|
| 0 - 47,450 | 22% |
| 47,450 – 114,650 | 25% |
| 114,650 – 174,700 | 28% |
| 174,700 – 311,950 | 33% |
| 311,950 - | 35% |

```
double rate = 0.35;

if (income <  47450){
    rate = 0.22;
}else if (income < 114650){
    rate = 0.25;
}else if (income < 174700){
    rate = 0.28;
}else  (income < 311950){
    rate = 0.33;
}
```

## String Comparisons

- Equal strings have the same number of characters, and each corresponding character is identical.
- Compare two strings using the notation `str1.equals(str2)`
- The `.equals()` method returns true if the two strings are equal.
- A common error is to use `==` to compare two strings, which behaves differently than expected.

## Example - String Comparisons

```java
import java.util.Scanner;

public class StringCensoring {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String userWord = "";

      System.out.print("Enter a word: ");
      userWord = scnr.next();

      if (userWord.equals("Voldemort")) {
         System.out.println("He who must not be named");
      }
      else {
         System.out.println(userWord);
      }
   }
}
```

## Comparing two Strings

| Relation | Expression to detect |
|---|---|
| str1 less-than str2 | `str1.compareTo(str2) < 0` |
| str1 equal-to str2 | `str1.compareTo(str2) == 0` |
| str1 greater-than str2 | `str1.compareTo(str2) > 0` |

## String Access Methods

| Operations | Description |
|---|---|
| `length()` | Number of characters |
| `isEmpty()` | true if length is 0 |
| `indexOf(item)` | Index of first item occurrence, else -1. |
| `substring(startIndex, endIndex)` | Returns substring starting at startIndex and ending at endIndex - 1 |

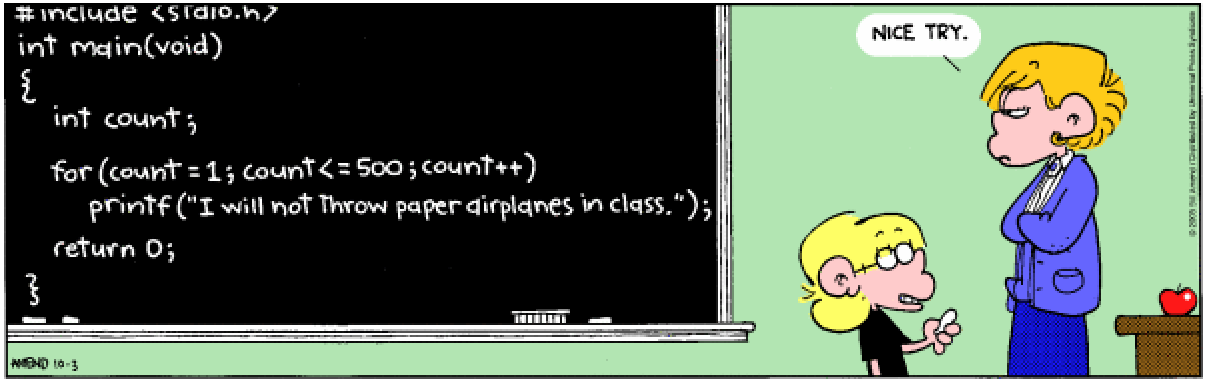
## While Loop

A common repetition structure:

- Evaluate a boolean expression.
- If true, execute some statements.
- Repeat.

```
while (boolean expression) {
   statement 1;
   statement 2;
}
```

## For loops



```
#include <stdio.h>
int main(void)
{
   int count;
   for (count = 1; count <= 500; count++)
      printf("I will not Throw paper airplanes in class.");
   return 0;
}
```

Copyright 2004, FoxTrot by Bill Amend

www.ucomics.com/foxtrot/2003/10/03

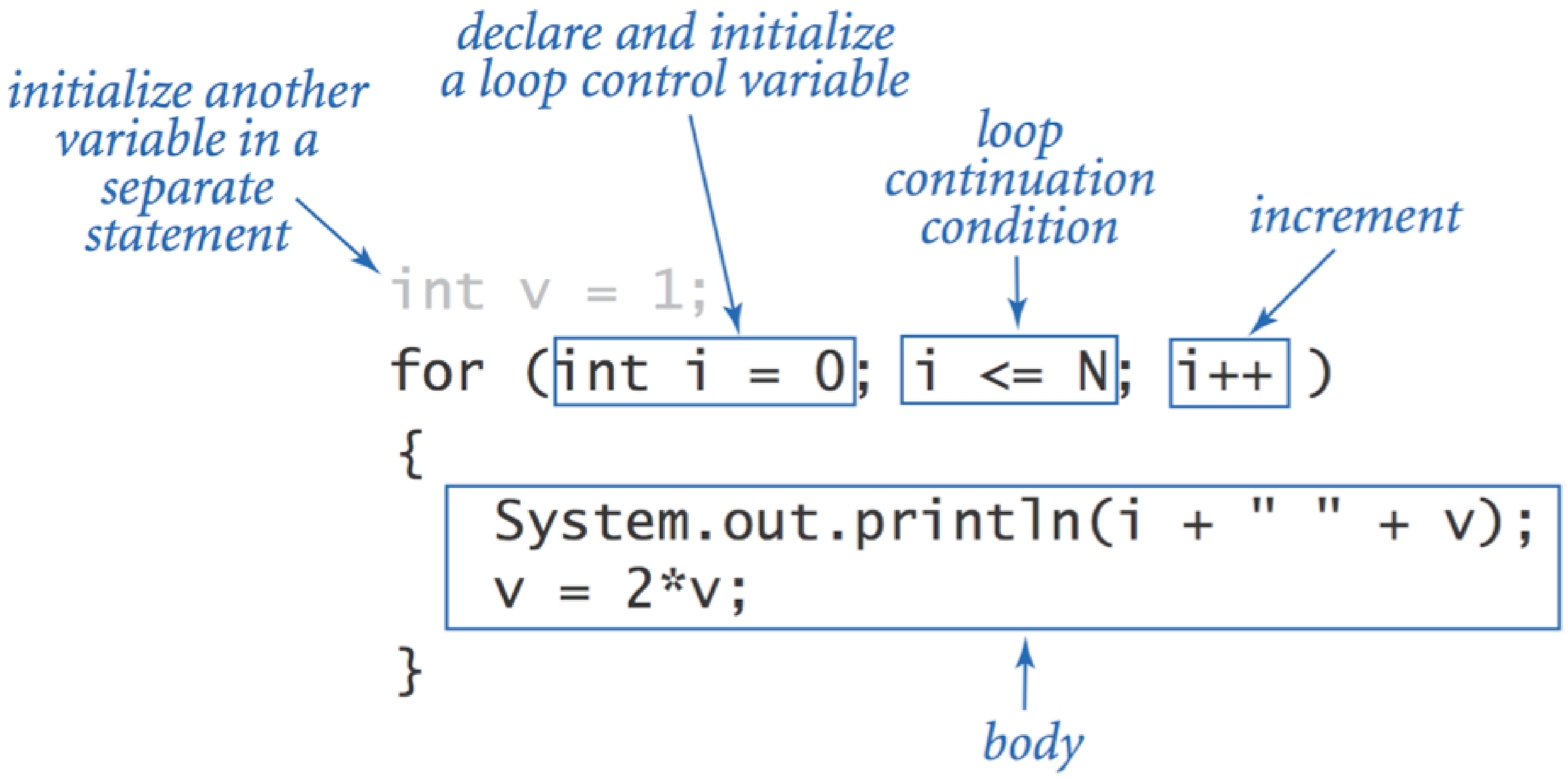
## For loop

Another common repetition structure.

1. Execute initialization statement.
2. Evaluate a boolean expression.
3. If true, execute some statements.
4. And then the increment statement.
5. Repeat 1

```
for (init; boolean expression; increment) {
   statement 1;
   statement 2;
}
```

## For loop


initialize another variable in a separate statement
declare and initialize a loop control variable
loop continuation condition
increment
body

```
int v = 1;
for (int i = 0; i <= N; i++ )
{
   System.out.println(i + " " + v);
   v = 2*v;
}
```

Q: What does it print?

A:

## For loop examples

| | |
|---|---|
| *print largest power of two less than or equal to N* | `int v = 1;`<br>`while (v <= N/2)`<br>`   v = 2*v;`<br>`System.out.println(v);` |
| *compute a finite sum* *($1 + 2 + \ldots + N$)* | `int sum = 0;`<br>`for (int i = 1; i <= N; i++)`<br>`   sum += i;`<br>`System.out.println(sum);` |
| *compute a finite product* *($N! = 1 \times 2 \times \ \ldots \ \times N$)* | `int product = 1;`<br>`for (int i = 1; i <= N; i++)`<br>`   product *= i;`<br>`System.out.println(product);` |
| *print a table of function values* | `for (int i = 0; i <= N; i++)`<br>`   System.out.println(i + " " + 2*Math.PI*i/N);` |

Print powers of 2 that are <= $2^N$.

- Increment i from 0 to N.
- Double v each time.

Anything wrong with the following code for printing powers of 2?

```
int i = 0;
int v = 1;
while (i <= N) {
   System.out.println(i + " " + v);
   i = i + 1;
   v = 2 * v;
}
```

## Questions

Q: Anything wrong with the following code for printing powers of 2?

```
int i = 0;
int v = 1;
while (i <= N)
   System.out.println(i + " " + v);
   i = i + 1;
   v = 2 * v;
```

## Questions

Q: Anything wrong with the following code?

```
int i = 0;
while (i <= N);
   i = i + 1;
```

## Switch statements

```
public class SwitchDemo
{
   public static void main(String[] args)
   {
        int day = Integer.parseInt(args[0]);

        switch(day)
        {
            case 0:
               System.out.println(“Sunday");
               break;

            case 6:
               System.out.println(“Saturday");
               break;

            default:
               break;
        }
   }
}
```

## Nested Loop

- A nested loop is a loop that appears in the body of another loop.
- The nested loops are commonly referred to as the inner loop and outer loop.

```
while (expr1) {
    while (expr2) {
        // Inner Loop
    }
}
```

```
for ( init1; expr1 ; increment1) {
    for ( init2; expr2 ; increment2 ){
        // inner loop
    }
}
```

## Example (1)

```java
char letter1 = 'a';
while (letter1 <= 'z') {
   char letter2 = 'a';
   while (letter2 <= 'z') {
      System.out.println("" + letter1 + "" + letter2 + ".com");
      ++letter2;
   }
   ++letter1;
}
```

## Example (2)

```
for (int i = 0; i < M; i++) {
   for (int j = 0; j < N; j++) {
        System.out.println(i + " " + j);
   }
}
```

## Exo (1)

Given the following code, how many times will the inner loop body execute?

```
int row = 0;
int col = 0;
for(row = 0; row < 2; row = row + 1) {
    for(col = 0; col < 3; col = col + 1) {
        // Inner loop body
    }
}
```

## Exo (2)

Given the following code, how many times will the inner loop body execute?

```
char letter1 = '?';
char letter2 = '?';

letter1 = 'a';
while (letter1 <= 'f') {
   letter2 = 'c';
   while (letter2 <= 'f') {
      // Inner loop body
      ++letter2;
   }
   ++letter1;
}
```

## Break

- A break statement in a loop causes an immediate exit of the loop.

Example:

```
for (int i=0; i<10; i++) {
    if (i == 6){
        break;
    }

    System.out.println(i);
}
```

This program prints numbers from 0 to 5.

## Continue

- A continue statement in a loop causes an immediate jump to the loop condition check

Example:

```
for (int i=0; i<10; i++) {
    if ((i % 2) == 0){
        continue;
    }

    System.out.println(i);
}
```

This program prints all odd numbers less than 10.

## Break / Continue

- Break and continue statements can avoid excessive indenting/nesting within a loop.
- But they could be easily overlooked, and should be used sparingly, when their use is clear to the reader.

Example:

```
for (i = 0; i < 5; ++i) {
    if (i < 10) {
        continue;
    }
    System.out.println(i);
}
```

This code will not print any output.

## Precedence rules for logical and relational operators.

| Convention | Description | Explanation |
|---|---|---|
| *( )* | Items within parentheses are evaluated first. | In `!(age > 16)`, age > 16 is evaluated first, then the logical NOT. |
| *!* | Next to be evaluated is *!*. | |
| ** / % + -* | Arithmetic operator are then evaluated using the precedence rules for those operators. | `z - 45 < 53` is evaluated as `(z - 45) < 53`. |

## Appendix (Precedence)

| Convention | Description | Explanation |
|---|---|---|
| & | Then, the bitwise AND operator is evaluated. | `x == 5 \| y == 10 & z != 10` is evaluated as `(x == 5) \| ((y == 10) & (z != 10))` because & has precedence over \|. |
| / | Then, the bitwise OR operator is evaluated. | `x == 5 \| y == 10 && z != 10` is evaluated as `((x == 5) \| (y == 10)) && (z != 10))` because \| has precedence over &&. |
| && | Then, the logical AND operator is evaluated. | `x == 5 \|\| y == 10 && z != 10` is evaluated as `(x == 5) \|\| ((y == 10) && (z != 10))` because && has precedence over \|\|. |

## In Class Programming

- String Comparisons
- Do while
- Switch