NYU

Lecture 3

Mathematical functions, characters, strings, and loops

The Math Class

- Java provides many useful methods in the Math class for performing common mathematical functions.

- Class constants:
  - PI
  - E
- Class methods:
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods

Trigonometric Methods

- sin(double a)
- cos(double a)
- tan(double a)
- acos(double a)
- asin(double a)
- atan(double a)

```
Math.sin(0) returns 0.0
Math.sin(Math.PI / 6)
  returns 0.5
Math.sin(Math.PI / 2)
  returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6)
  returns 0.866
Math.cos(Math.PI / 2)
  returns 0
```

Exponent Methods

- exp(double a)
  - Returns e raised to the power of a.
- log(double a)
  - Returns the natural logarithm of a.
- log10(double a)
  - Returns the 10-based logarithm of a.
- pow(double a, double b)
  - Returns a raised to the power of b.
- sqrt(double a)
  - Returns the square root of a.

```
Math.exp(1) returns 2.71
Math.log(2.71) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(3.5, 2.5) returns
   22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 3.24
```

Rounding Methods

- `double ceil(double x)`
  - x rounded up to its nearest integer. This integer is returned as a double value.
- `double floor(double x)`
  - x is rounded down to its nearest integer. This integer is returned as a double value.
- `double rint(double x)`
  - x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.
- `int round(float x)`
  - Return (int)Math.floor(x+0.5).
- `long round(double x)`
  - Return (long)Math.floor(x+0.5).

Rounding Methods Examples

- `Math.ceil(2.1)` returns 3.0
- `Math.ceil(2.0)` returns 2.0
- `Math.ceil(-2.0)` returns -2.0
- `Math.ceil(-2.1)` returns -2.0
- `Math.floor(2.1)` returns 2.0
- `Math.floor(2.0)` returns 2.0
- `Math.floor(-2.0)` returns -2.0
- `Math.floor(-2.1)` returns -3.0
- `Math.rint(2.1)` returns 2.0
- `Math.rint(2.0)` returns 2.0
- `Math.rint(-2.0)` returns -2.0
- `Math.rint(-2.1)` returns -2.0
- `Math.rint(2.5)` returns 2.0
- `Math.rint(-2.5)` returns -2.0
- `Math.round(2.6f)` returns 3
- `Math.round(2.0)` returns 2
- `Math.round(-2.0f)` returns -2
- `Math.round(-2.6)` returns -3

min, max, and abs

- `max(a, b)` and `min(a, b)`
  - Returns the maximum or minimum of two parameters.
- `abs(a)`
  - Returns the absolute value of the parameter.
- `random()`
  - Returns a random double value in the range [0.0, 1.0).

```
Math.max(2, 3) returns 3
Math.max(2.5, 3) returns
  3.0
Math.min(2.5, 3.6) returns
  2.5
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1
```

The random Method

- Generates a random double value greater than or equal to 0.0 and less than 1.0 (0 <= Math.random() < 1.0).

- Example:

```
(int) (Math.random() *10); //returns a random between 0 and 9
50 + (int) (Math.random() *50); //between 50 and 99
```

- In general

```
a + Math.random() * b; //returns a random between a and a+b
```

Character Data Type

- `char letter = 'A'; (ASCII)`
- `char numChar = '4'; (ASCII)`
- `char letter = '\u0041'; (Unicode)`
- `char numChar = '\u0034'; (Unicode)`

- The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character.
- Example:

```
char ch = 'a';
System.out.println(++ch); //prints b
```

ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
`'0'` to `'9'`	48 to 57	\u0030 to \u0039
`'A'` to `'Z'`	65 to 90	\u0041 to \u005A
`'a'` to `'z'`	97 to 122	\u0061 to \u007A

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
`\b`	Backspace	`\u0008`	8
`\t`	Tab	`\u0009`	9
`\n`	Linefeed	`\u000A`	10
`\f`	Formfeed	`\u000C`	12
`\r`	Carriage Return	`\u000D`	13
`\\`	Backslash	`\u005C`	92
`\"`	Double Quote	`\u0022`	34

Using chars

```
int i = 'a'; // Same as int i = (int)'a';
char c = 97; // Same as char c = (char)97;

if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

Methods in the Character Class

Method	Description
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

The String Type

- The char type only represents one character.
- To represent a string of characters, use the data type called String.
- Example:

```
String message = "Welcome to Java";
```

- String is actually a predefined class in the Java library just like the System class and Scanner class.
- The String type is not a primitive type. It is known as a reference type.
- Any Java class can be used as a reference type for a variable.
- Reference data types will be thoroughly discussed “Objects and Classes.”
- For the time being, you just need to know how to:
  - declare a String variable,
  - assign a string to the variable,
  - concatenate strings,
  - perform simple operations for strings.

Simple Methods for String Objects

Method	Description
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

Simple Methods for String Objects

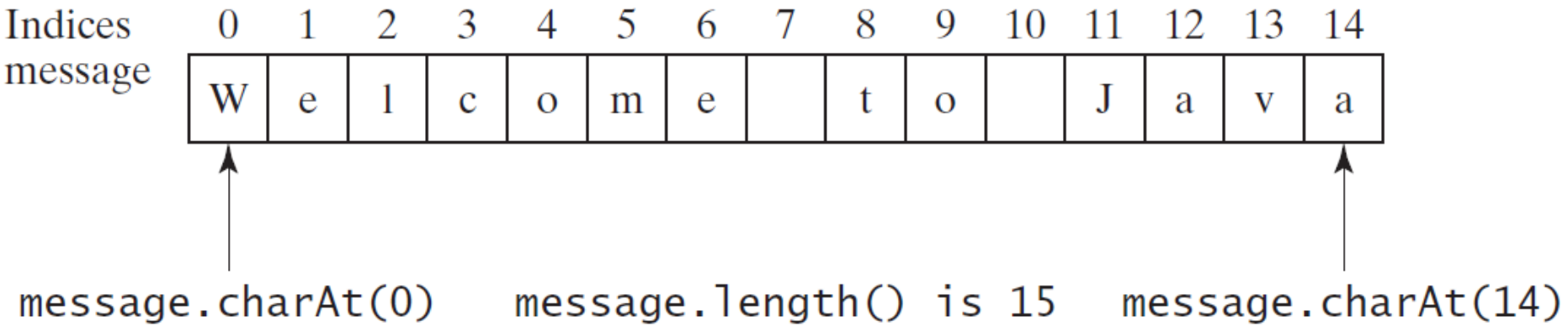
- Strings are objects in Java.
- The methods in the preceding table can only be invoked from a specific string instance.
- For this reason, these methods are called instance methods.
- A non-instance method is called a static method.
- A static method can be invoked without using an object.
- All the methods defined in the Math class are static methods.
- They are not tied to a specific object instance.
- The syntax to invoke an instance method is

```
referenceVariable.methodName(arguments)
```

- Example:

```
String message = "Welcome to Java";
int length = message.length();
```

Getting Characters from a String




```
String message = "Welcome to Java";
System.out.println("The first character in " +
"message is " + message.charAt(0));
```

String manipulation

- `"Welcome".toLowerCase()` returns a new string, welcome.
- `"Welcome".toUpperCase()` returns a new string, WELCOME.
- `"  Welcome  ".trim()` returns a new string, Welcome.

- `String message = "Welcome " + "to " + "Java";` Three strings are concatenated

- `String s = "Message" + 2;` `s` becomes Message2

- `String s1 = "Message" + 'B';` `s1` becomes MessageB

Reading from the Console

- String

```
Scanner input = new Scanner(System.in);
System.out.print("Enter two words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
```

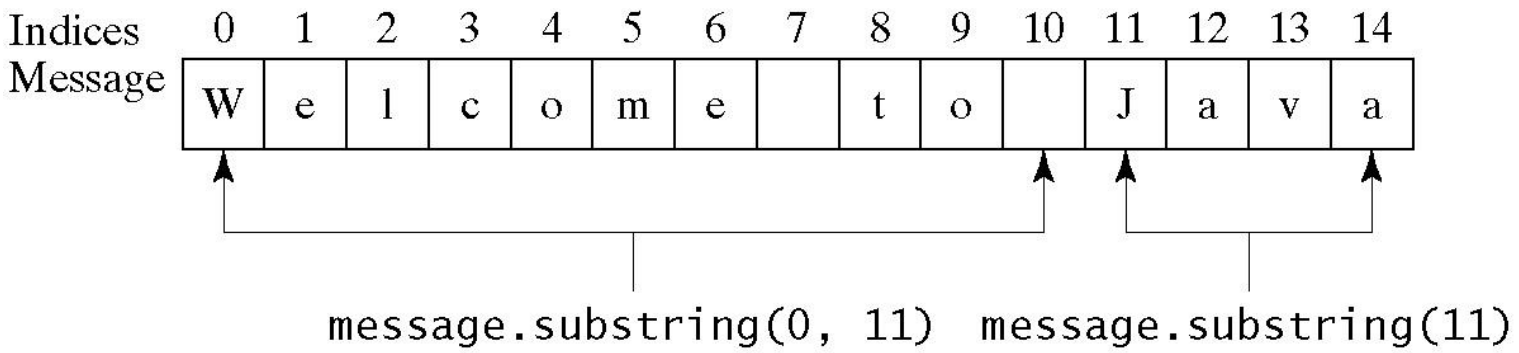
- Character

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

More operations on Strings

Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.

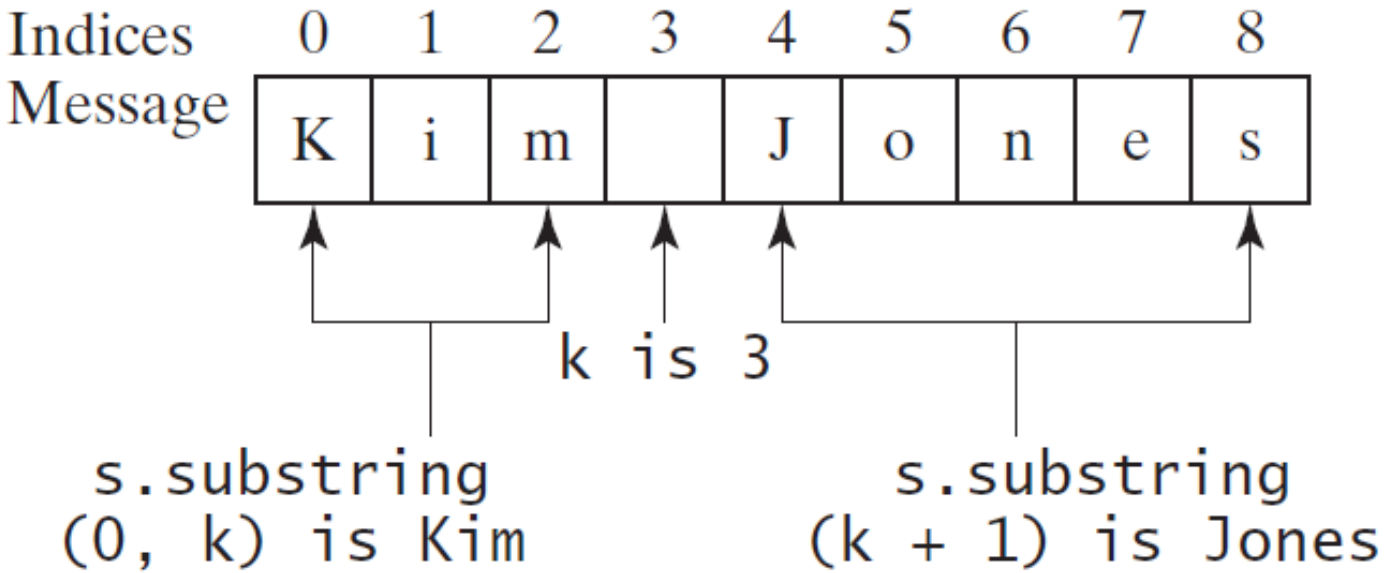


Finding a Character or a Substring in a String

Method	Description
`indexOf(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

Finding a Character or a Substring in a String

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



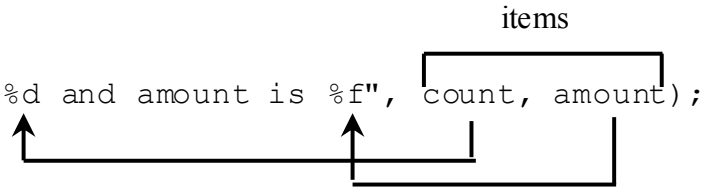
Conversion between Strings and Numbers

```
int intValue = Integer.parseInt(intString);
double doubleValue =
Double.parseDouble(doubleString);

String s = number + "";
```

Formatting Output

- Use the `printf` statement.
- `System.out.printf(format, items);`
- Where format is a string that may consist of substrings and format specifiers.
- A format specifier specifies how an item should be displayed:
  - `%b` a boolean
  - `%c` a character
  - `%d` a decimal integer
  - `%f` a floating-point number
  - `%e` a number in standard scientific notation
  - `%s` a string
- An item may be a numeric value, character, boolean value, or a string.

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```




```
display          count is 5 and amount is 45.560000
```

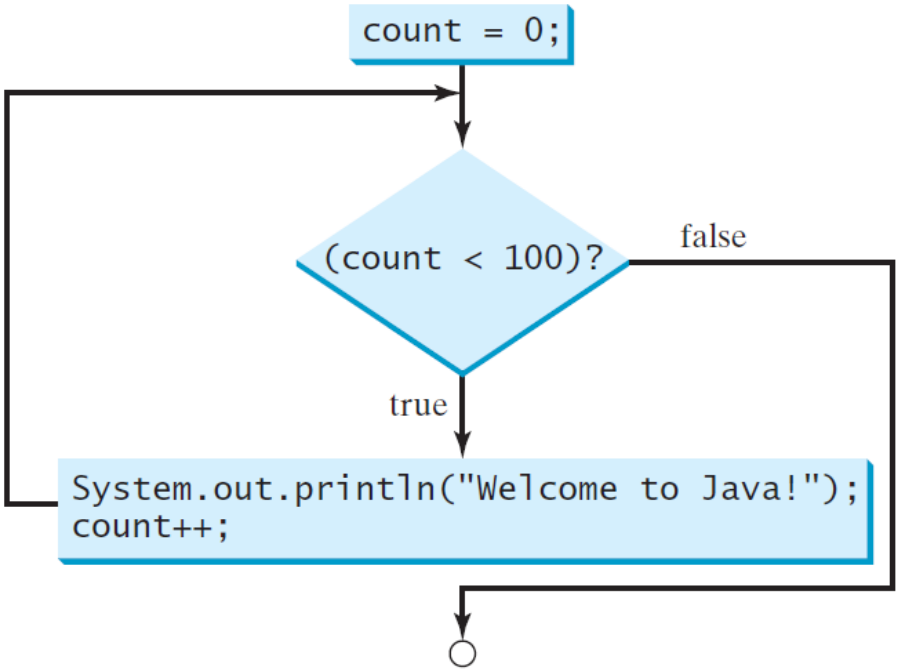
Problem

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!");
```

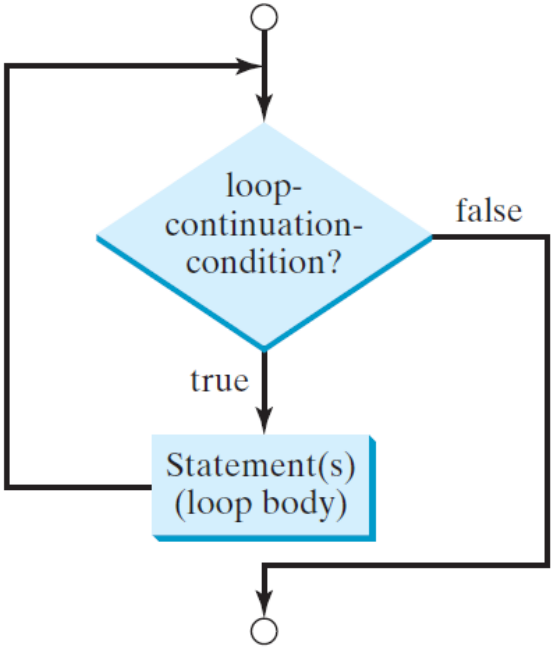
while Loops

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

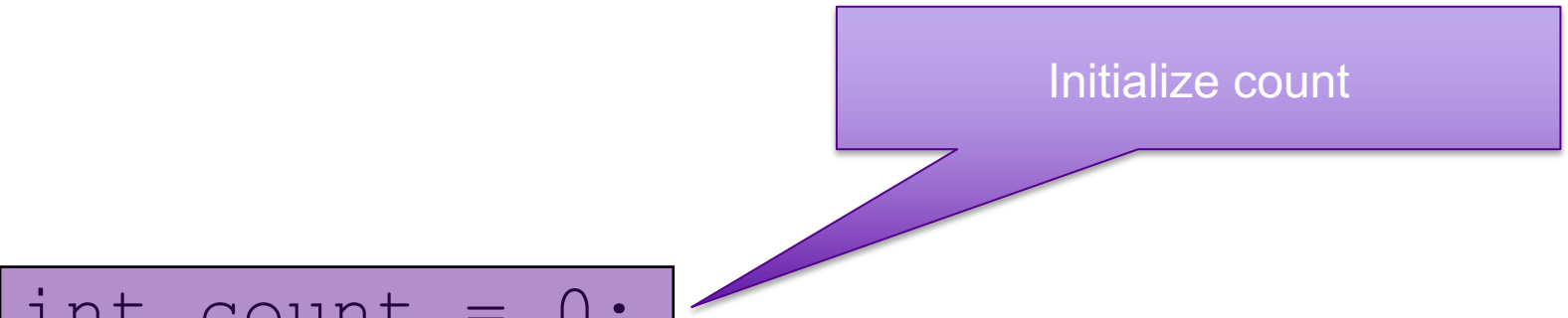


while Loop Flow Chart

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

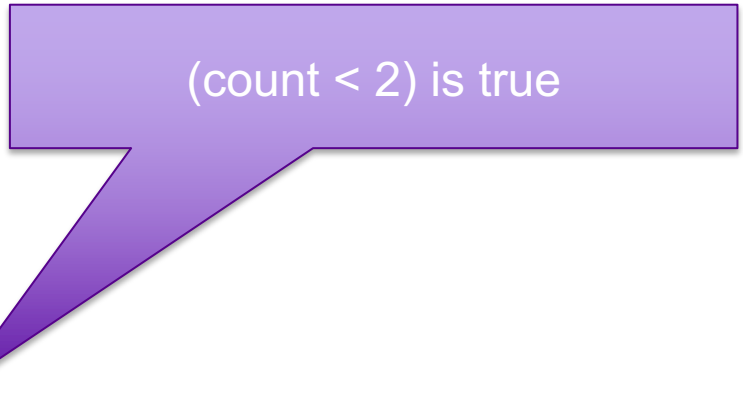


Trace while Loop



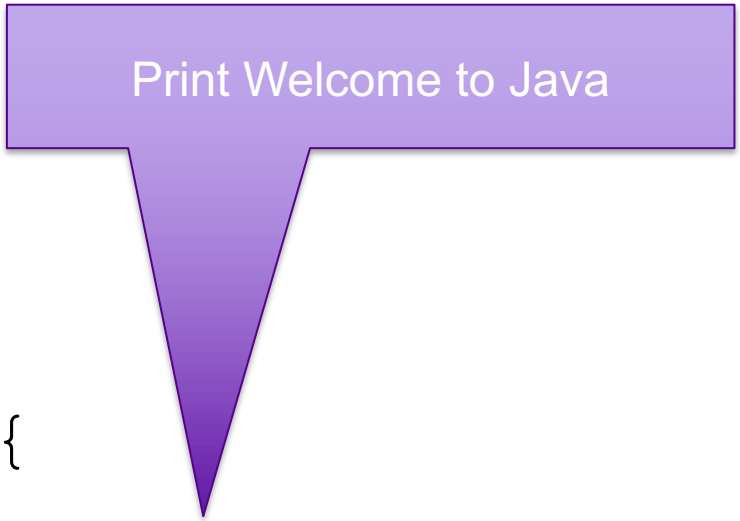

```
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Trace while Loop



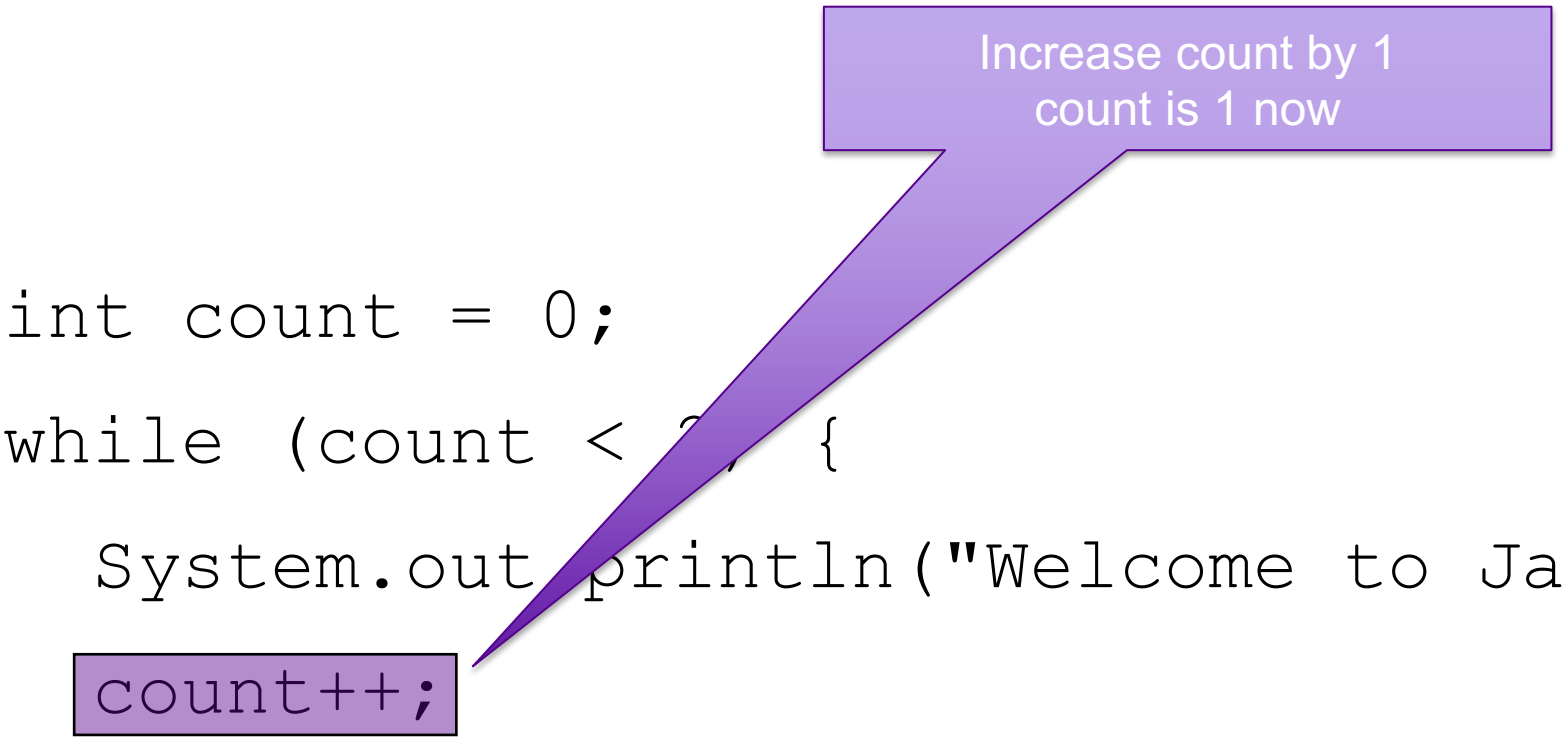

```
int count = 0;

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

Trace while Loop




```
int count = 0;

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

Trace while Loop

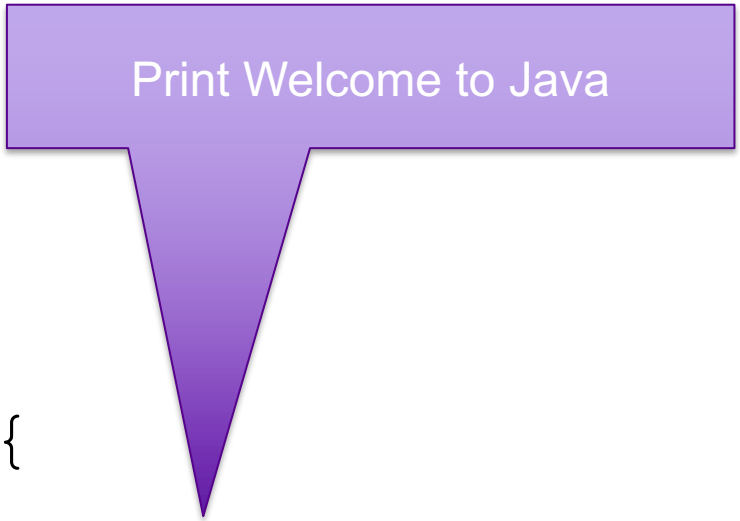


Trace while Loop

(count < 2) is still true since count is 1

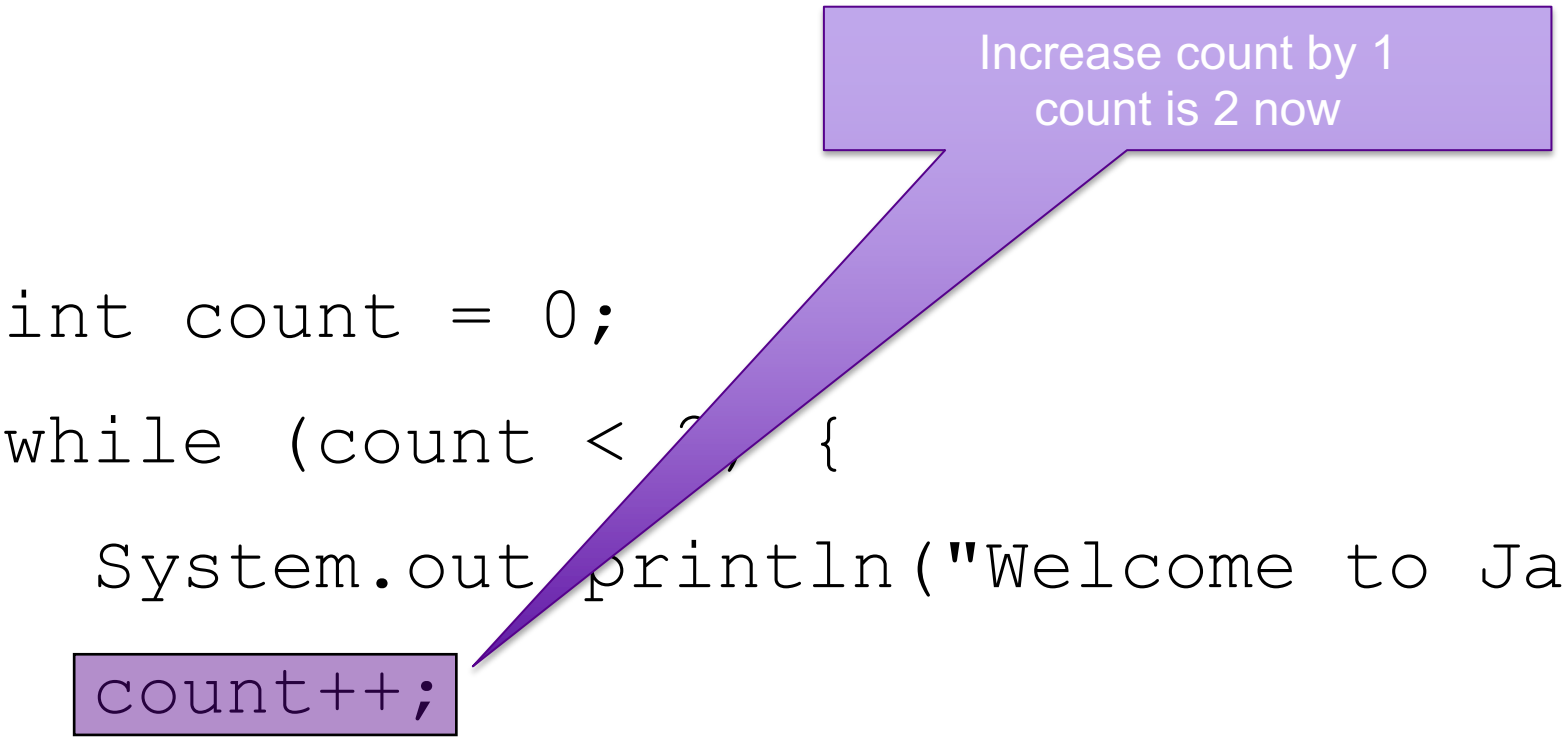
```
int count = 0;

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

Trace while Loop




```
int count = 0;

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

Trace while Loop



Trace while Loop

(count < 2) is false
since count is 2 now

```
int count = 0;

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

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.

Caution

- **Never use floating-point** values for equality checking in a loop control.
- Floating-point values are approximation
- Using them will result in imprecise counter values and inaccurate results.

- Example:
  - Compute: 1 + 0.9 + 0.8 + ... + 0.1

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

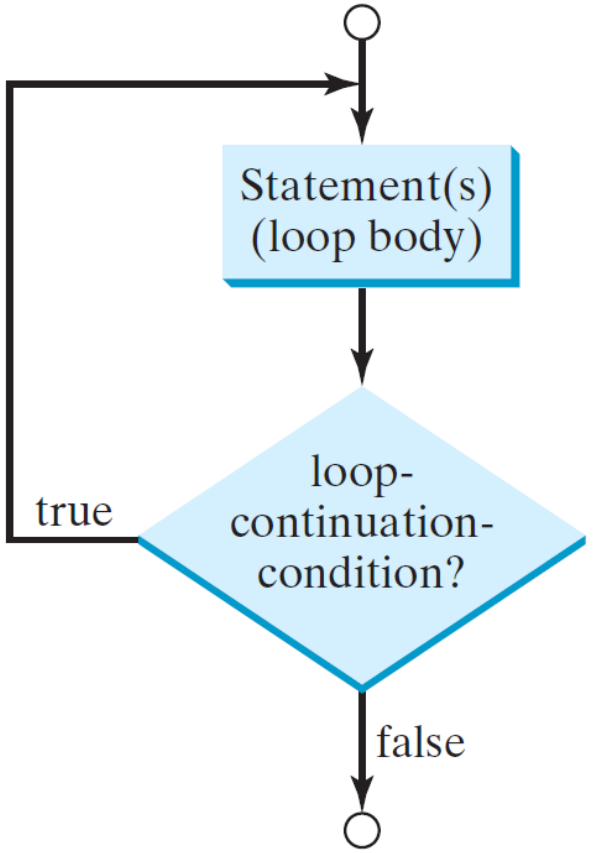
Ending a Loop with a Sentinel Value

- 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.

- Example
  - Write a program that reads and calculates the sum of an unspecified number of positive integers.
  - The input 0 signifies the end of the input (i.e. 0 is the sentinel value).

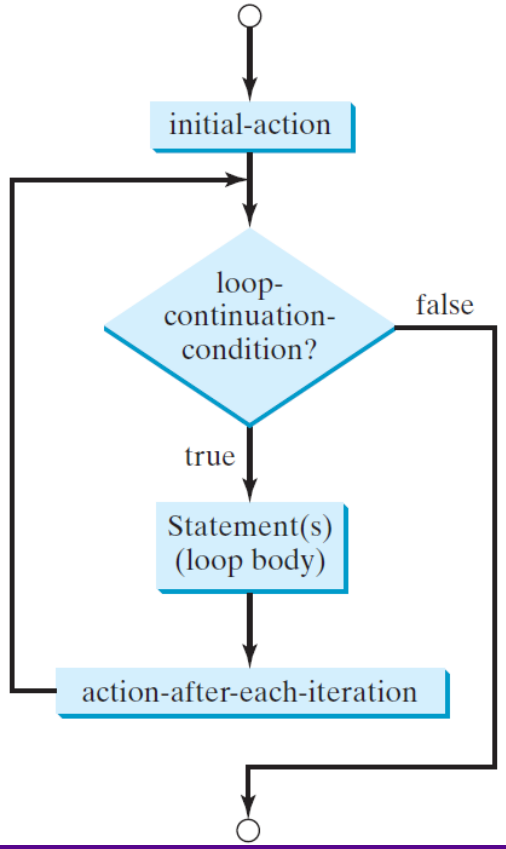
do-while Loop

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

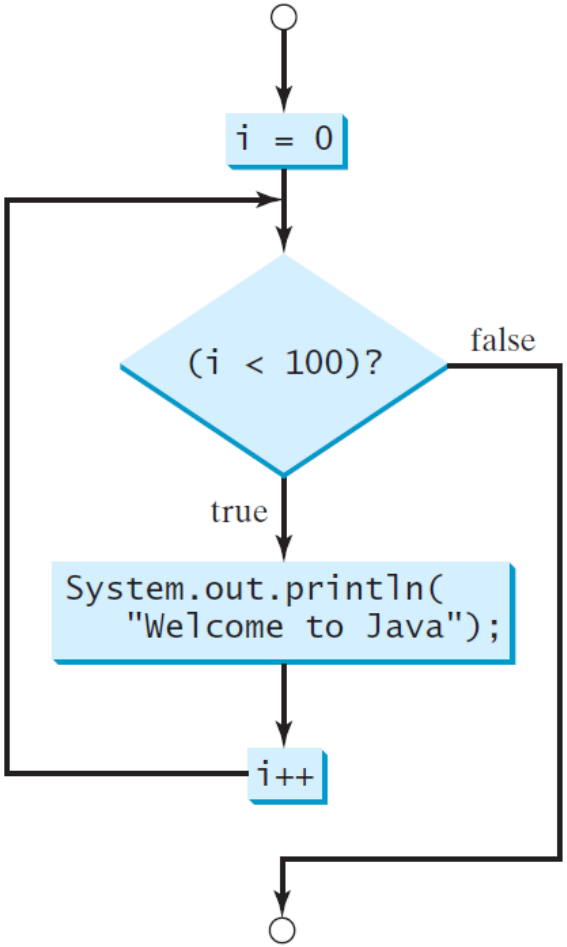


for Loops

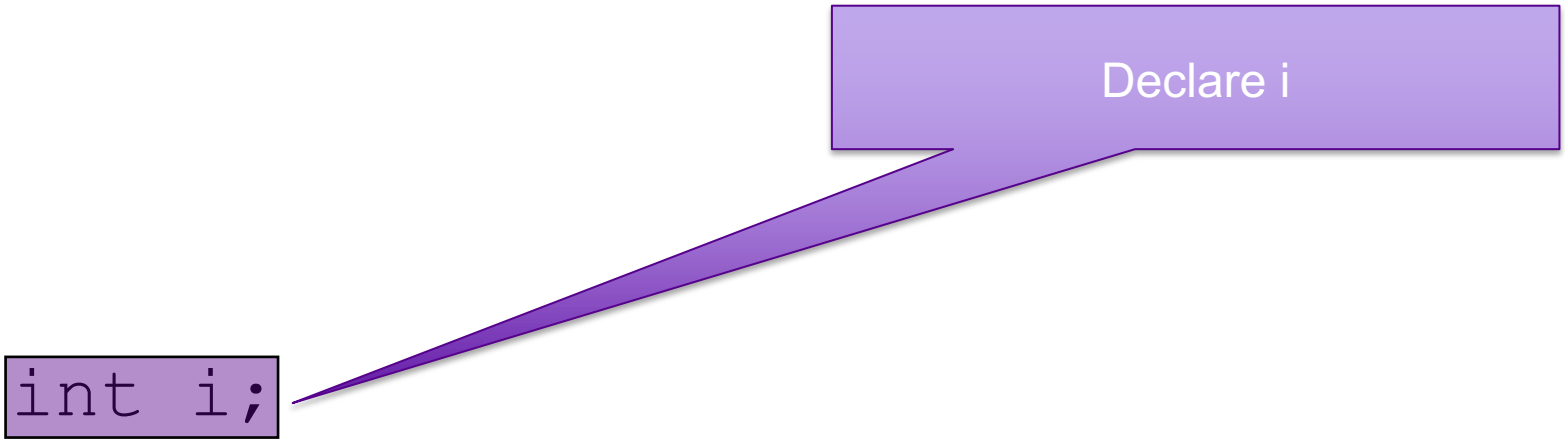
```
for (initial-action; loop-continuation-
condition; action-after-each-iteration) {
    // loop body;
    Statement(s);
}
```



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

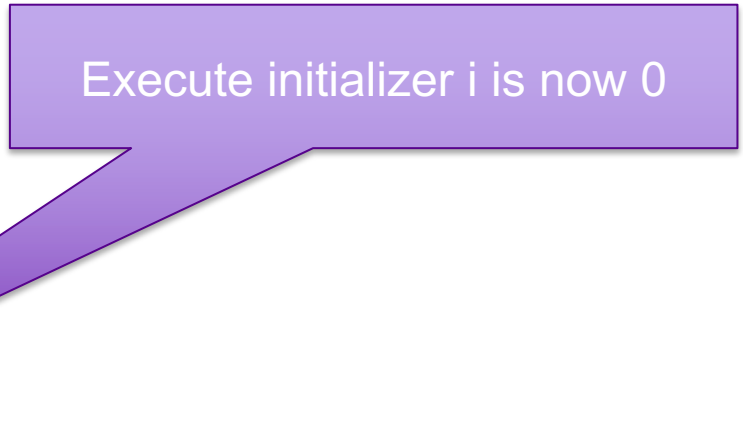


Trace for Loop



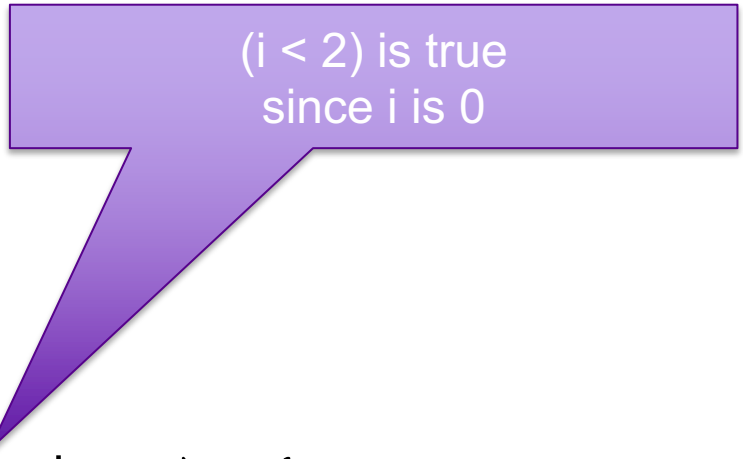

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



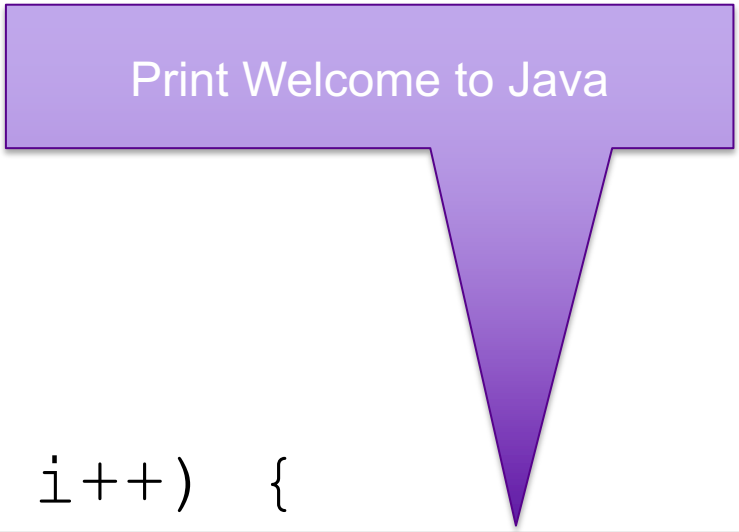

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



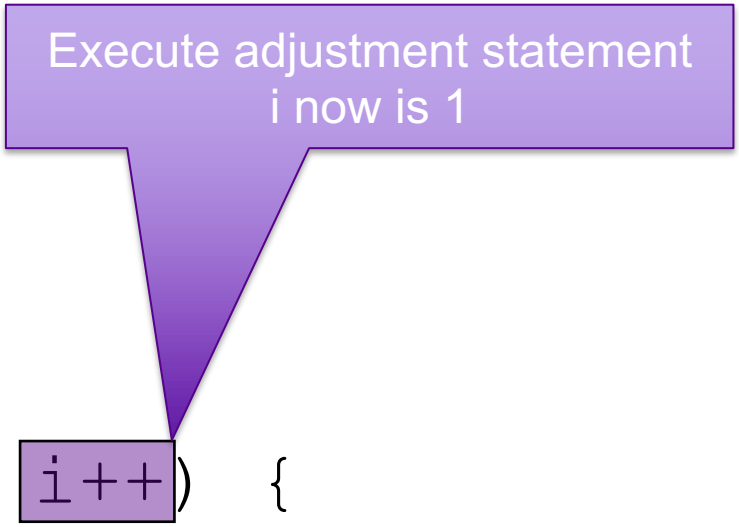

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



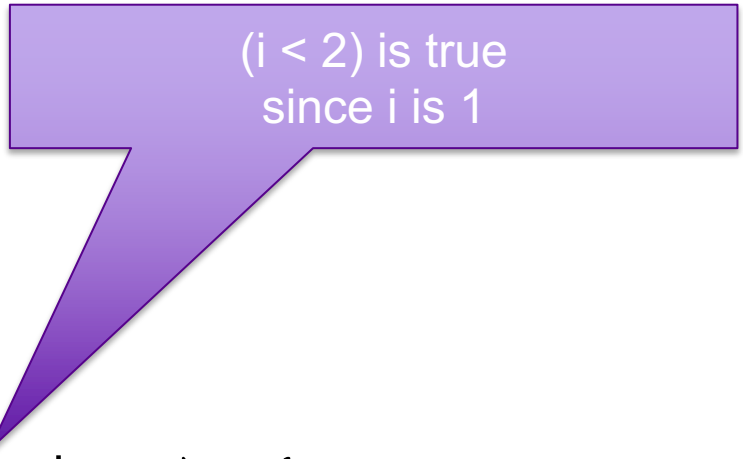

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



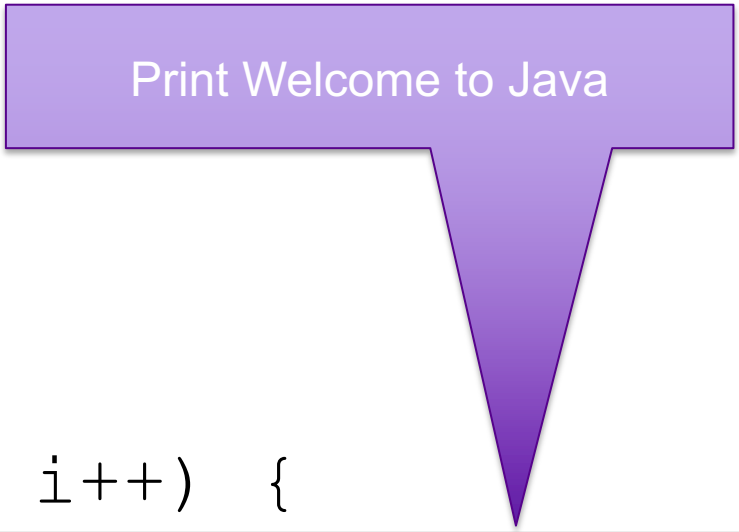

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



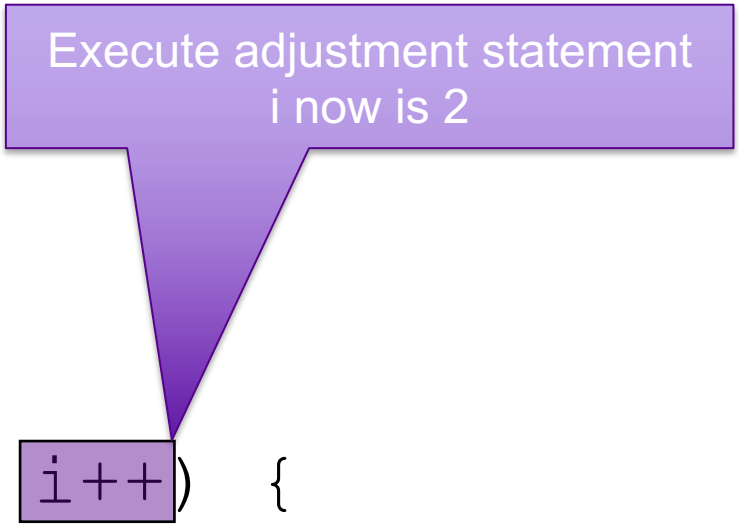

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



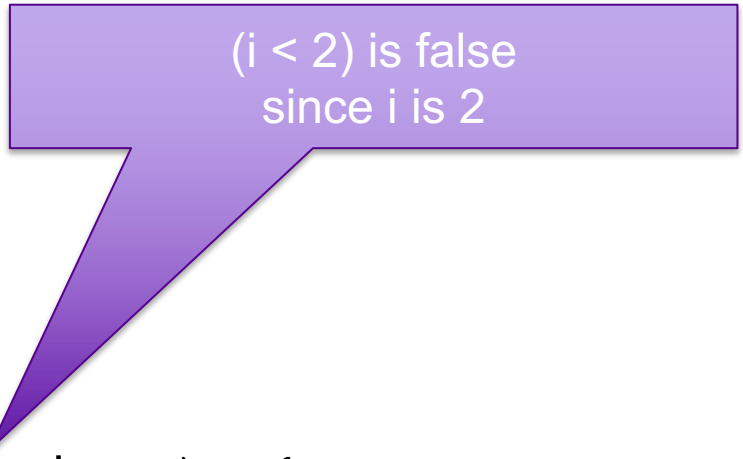

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



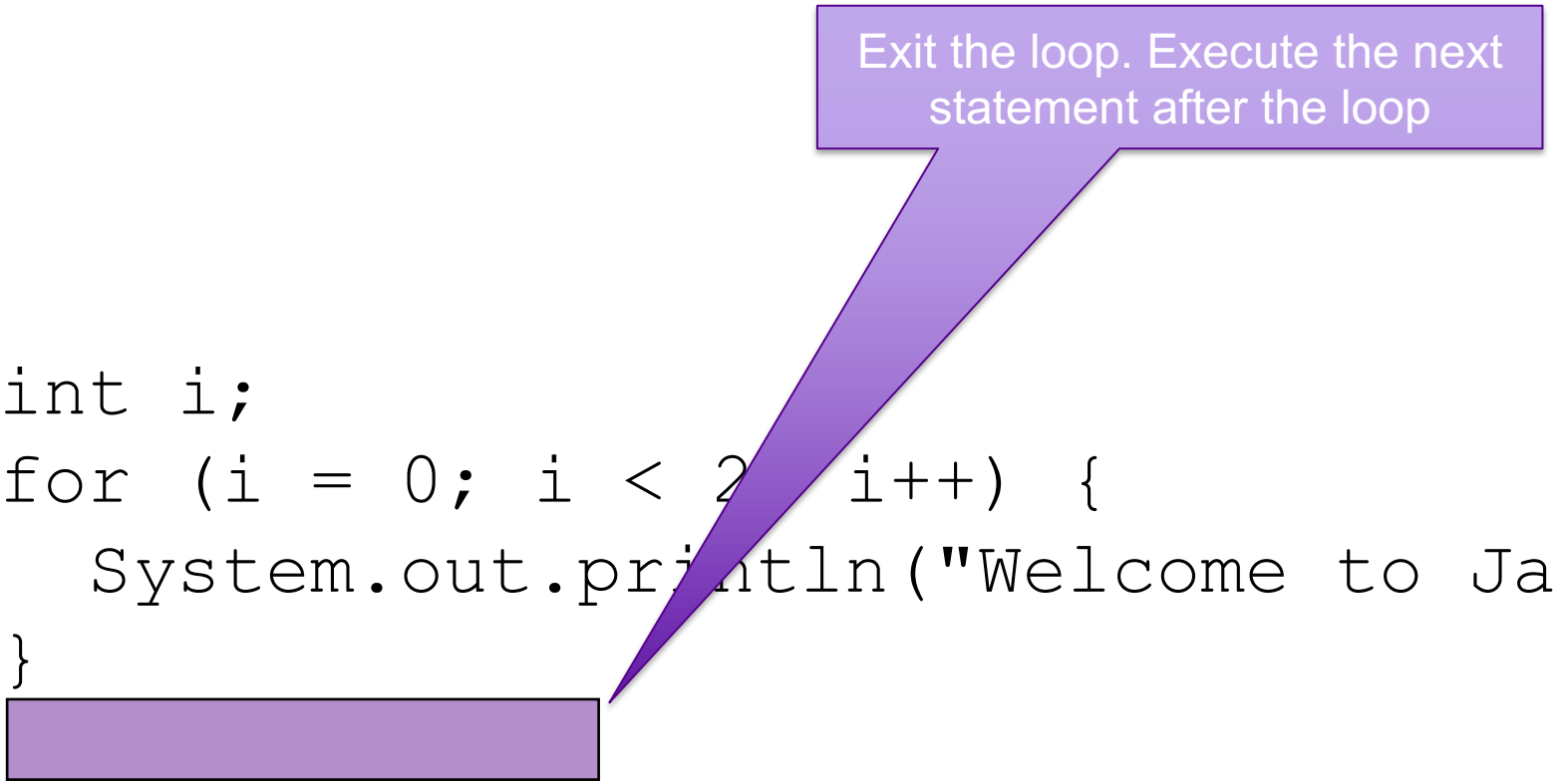

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Trace for Loop




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

Trace for 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.

- Examples:

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

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.

- Example: to create an infinite loop, is correct

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

- Better to use while (equivalent)

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

Where to put the semicolon

- Adding a semicolon at the end of the for clause before the loop body is a common mistake

- Example

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

Where to put the semicolon

- For while loops:

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

- For do loop, the semicolon is needed to end the loop:

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

Which Loop to Use?

- The three **equivalent** forms of loop statements (i.e., you can write a loop in any of these three forms)
  - While
  - do-while
  - For

- Example:

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

=

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

```
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**.
- In general, a **for loop** may be used if the number of repetitions is known
  - 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
  - For example, when 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.

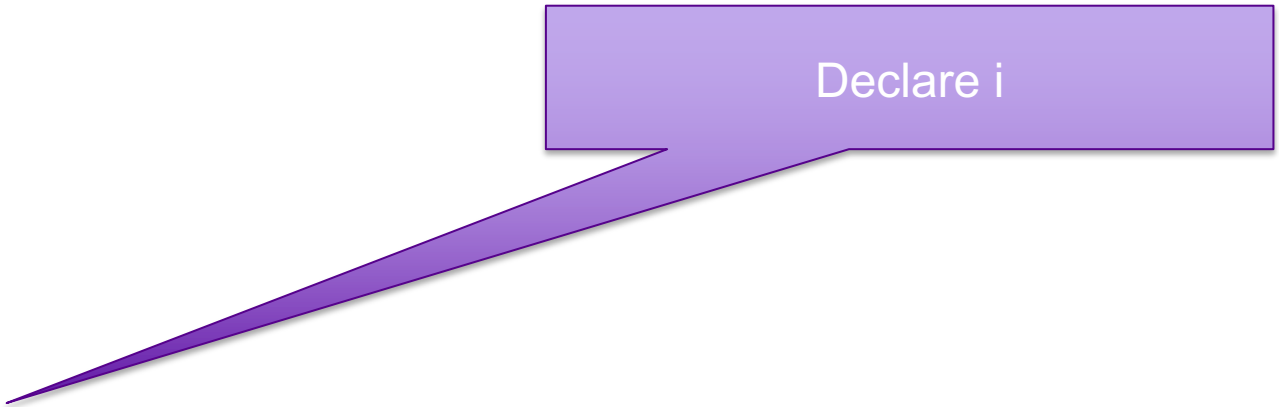
Break and continue

- Break and continue can be used in loops to provide additional control
- It might simplify programming
- Overusing or improper usage leads to spaghetti code

- Break
  - Exits the current loop

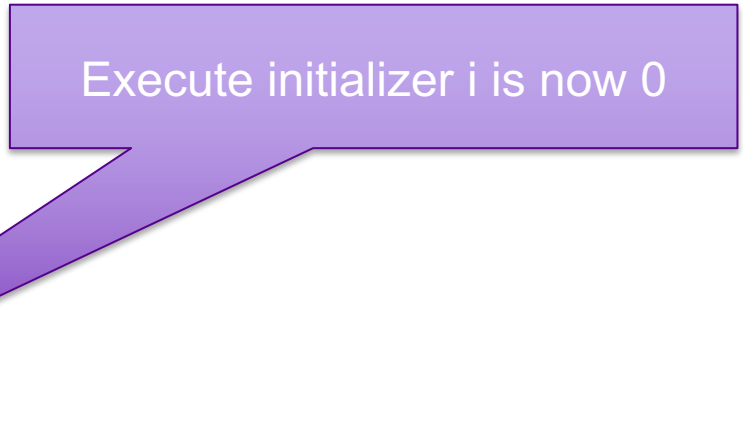
- Continue
  - Moves back the execution to the beginning of the loop

Trace for Loop



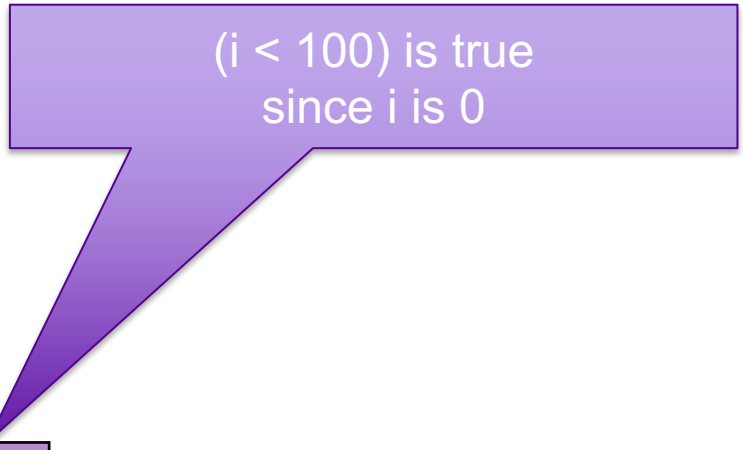

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



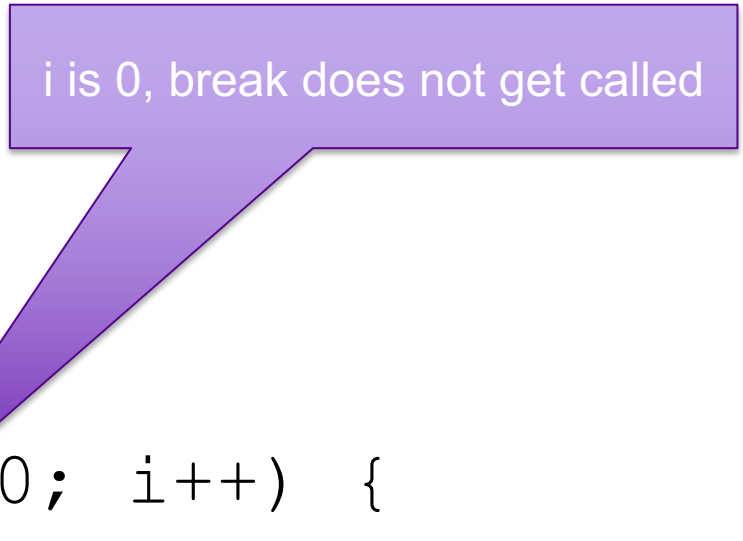

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



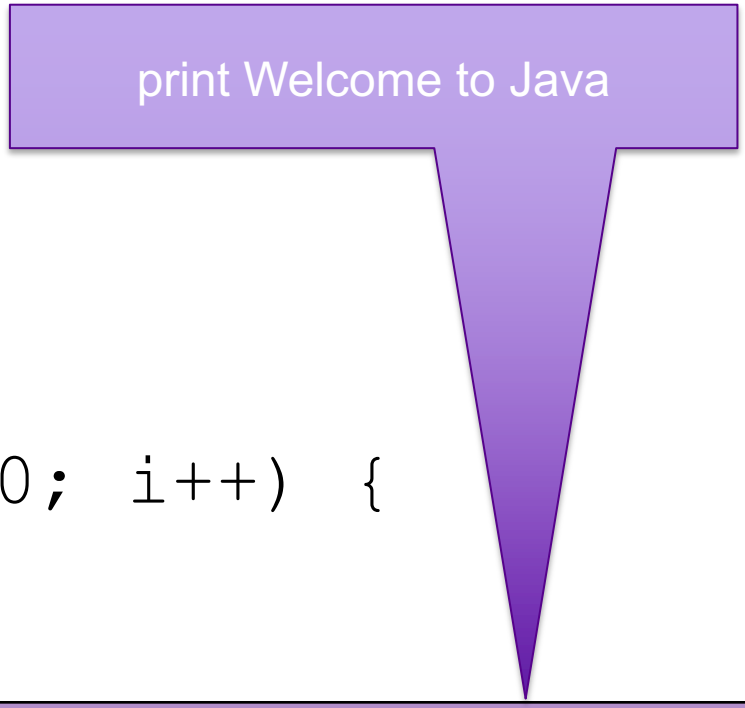

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



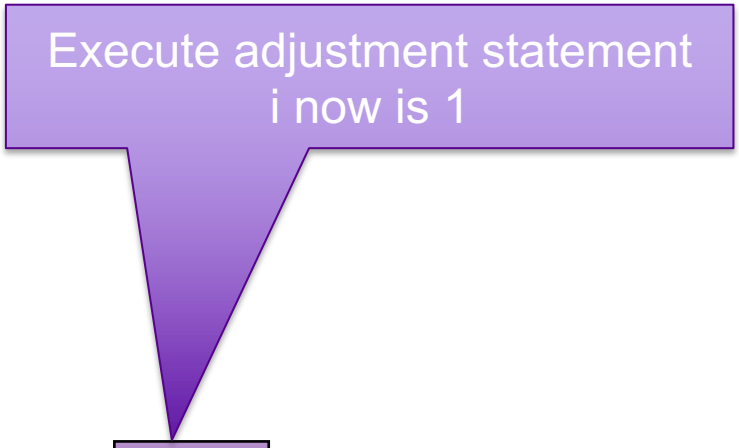

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



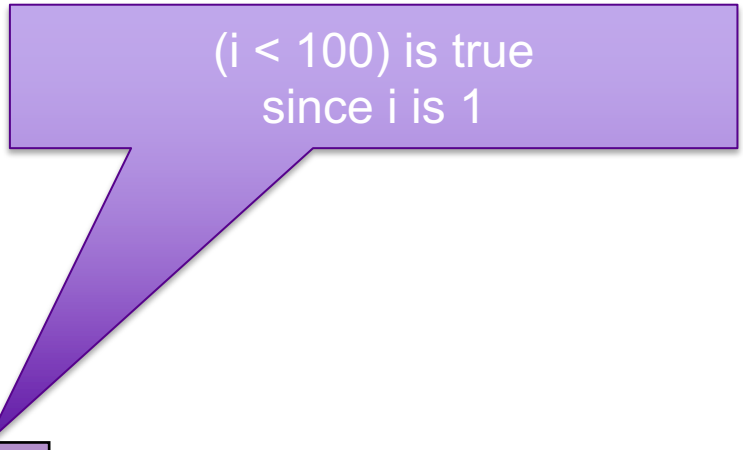

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



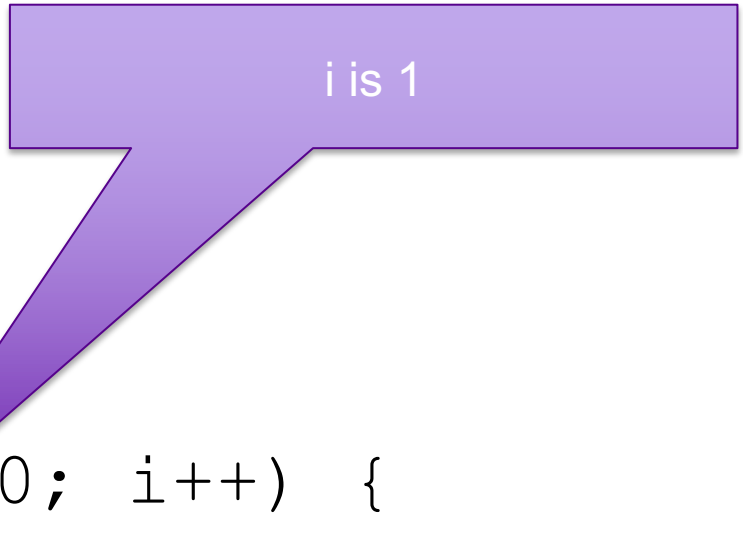

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



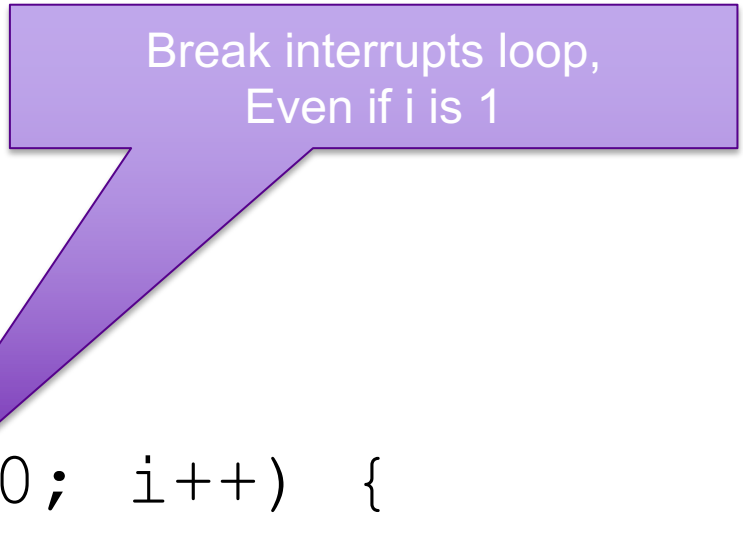

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop




```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



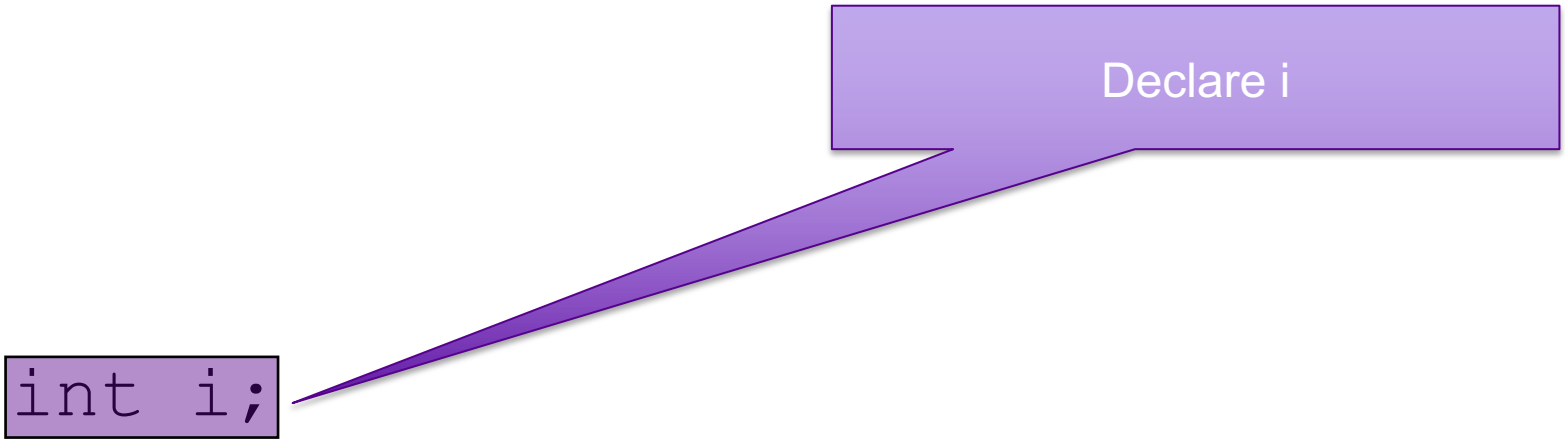

```
int i;
for (i = 0; i < 100; i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop

Exit the loop. Execute the next statement after the loop

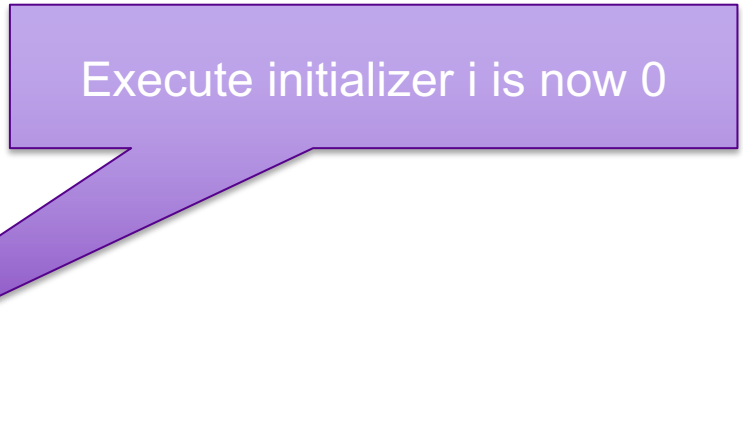
```
int i;
for (i = 0; i < 10;    i++) {
  if ( i == 1 )
    break;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



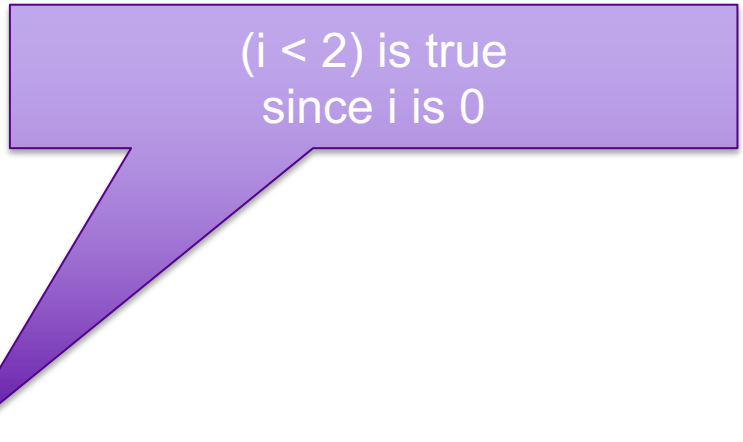

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



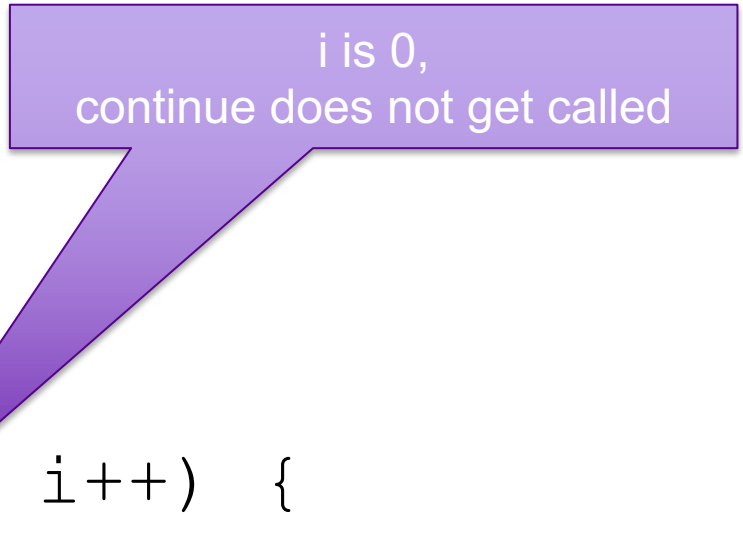

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



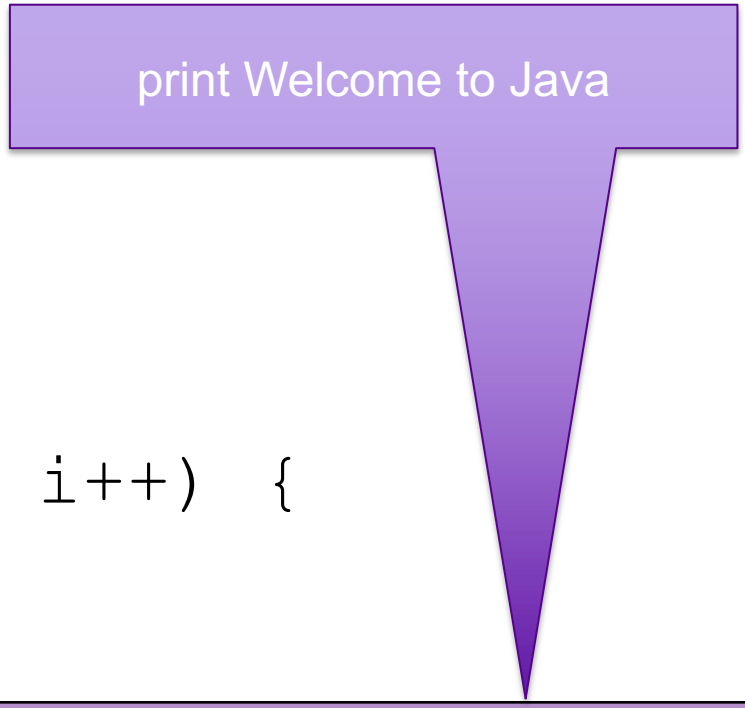

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



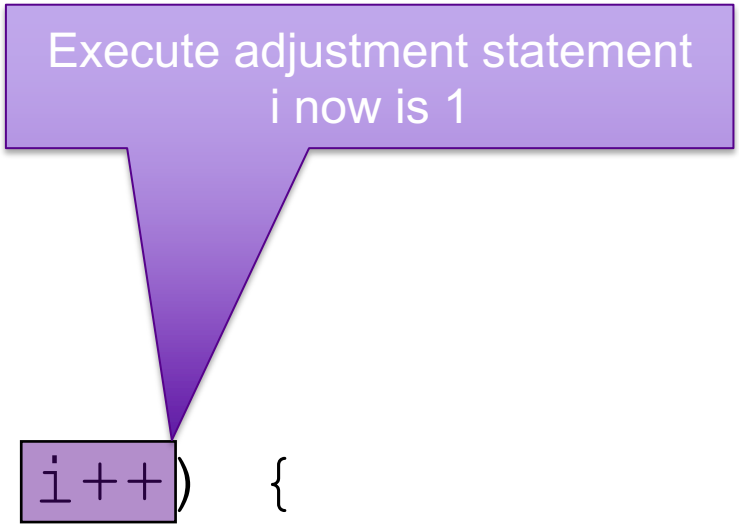

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



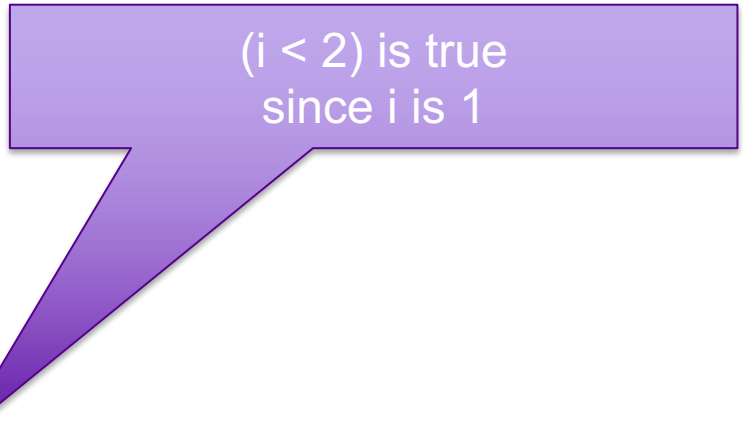

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



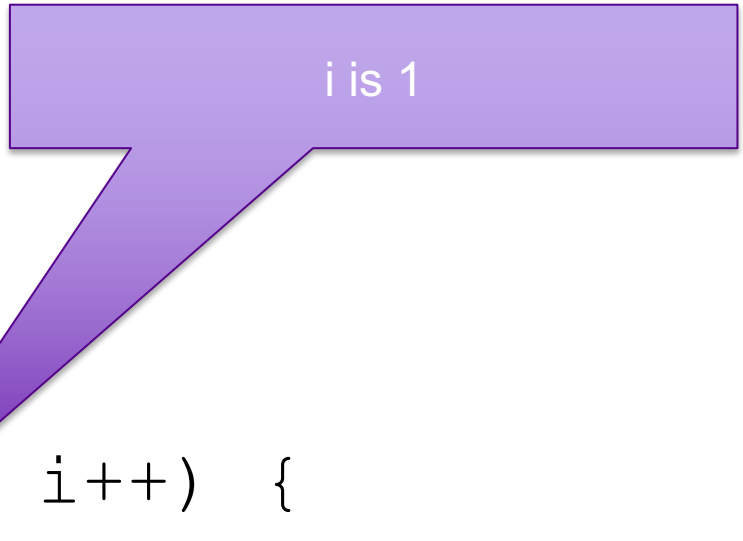

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



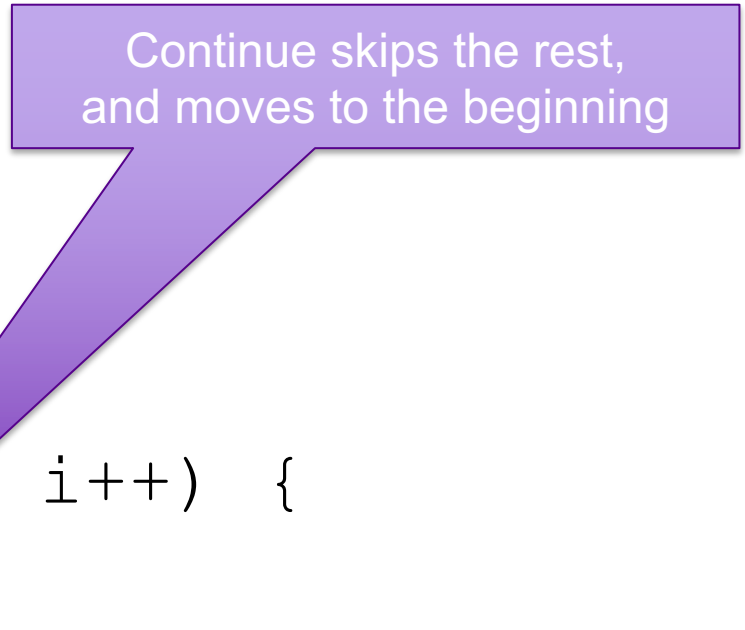

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



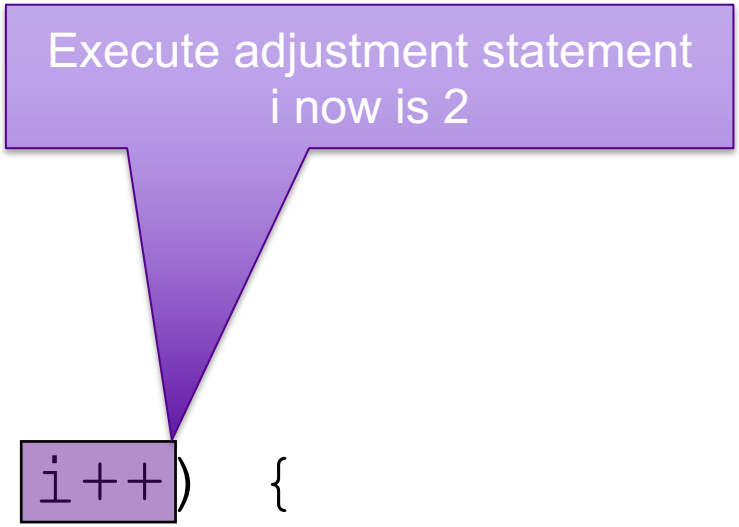

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop



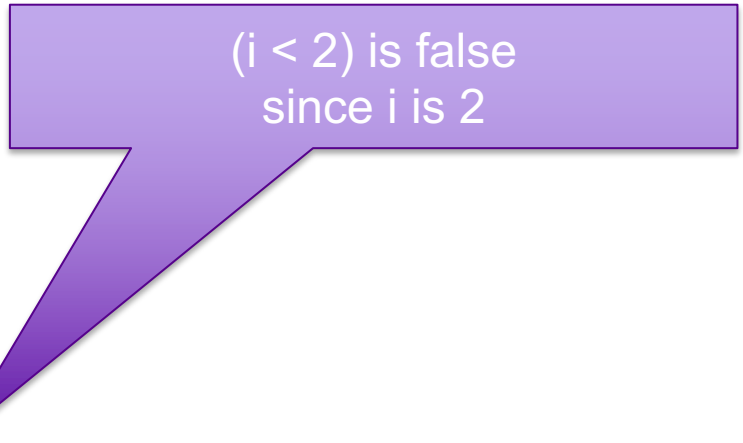

```
int i;
for (i = 0; i < 4; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop




```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop




```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  if ( i == 1 )
    continue;
  System.out.println("Welcome to Java!");
}
```

Exit the loop. Execute the next statement after the loop