CS 112 – Introduction to Computing II

Wayne Snyder
Computer Science Department
Boston University

Today:

Java expressions and operators concluded

Java Statements:

- Conditionals: if/then, if/then/else
- Loops: while, for

Next Time: Arrays, methods, program structure, fields vs local variables, public vs private, the keyword static.

Reading assignments are posted on the web site!

Java Statements

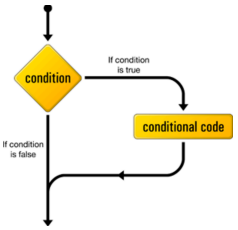
Recall: The core of a Java program is a **main** method consisting of a sequence of statements followed by semicolons; each statement is executed in sequence, and each has some effect on the state of the computer:

```
1  /* File: SampleProgram.java
2   * Author: Wayne Snyder (snyder@bu.edu)
3   * Date: 2/25/16
4   * Purpose: This is a sample problem for lecture 3 in CS 112.
5   */
6
7  public class SampleProgram {
8
9      public static void main(String[] args) {
10         int x;                  // declare x to be int
11         x = 3;                  // assign value to x
12         System.out.println(x);
13         double y = 3.4;         // combine declaration and assignment
14         double z = x + y;
15         System.out.println(z);
16     }
17 }
```

Flow of control: conditionals if/then, if/then/else

Computer Science

Conditional statements alter the flow of execution by testing some boolean condition and branching one way or the other; you just have to translate what you already know from Python into Java syntax:



Simple if/then statement:

Python:

```
x = 4

print("testing x....")

if( (x % 2) == 0):
    print("x is even")

print("done")
```

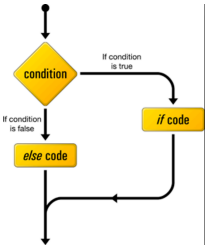
Java:

```
public static void main(String[] args) {

    int x;
    System.out.println("testing x....");

    if( (x % 2) == 0 )
        System.out.println("x is even");

    System.out.println("done");

}
```

Flow of control: conditionals if, if/then, if/then/else

Computer Science

Conditional statements alter the flow of execution by testing some boolean condition and branching one way or the other; you just have to translate what you already know from Python into Java syntax:



if/then/else statement:

Python:

```
print("testing x....")

if( (x % 2) == 0):
    print("x is even")
else:
    print("x is odd")

print("done")
```

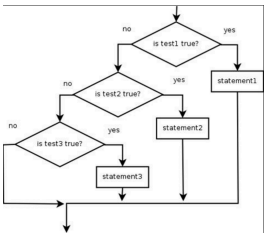
Java:

```
public static void main(String[] args) {

    int x = 4;
    System.out.println("testing x....");

    if( (x % 2) == 0 )
        System.out.println("x is even");
    else
        System.out.println("x is odd");

    System.out.println("done");
}
```

Flow of control: conditionals if, if/then, if/then/else

Computer Science

Conditional statements alter the flow of execution by testing some boolean condition and branching one way or the other; you just have to translate what you already know from Python into Java syntax:



Compound if/then/else statement:

Python:

```
print("testing x....")
if(x < 0):
    print("x is negative")
elif( x < 10):
    print("x is positive but less than 10")
elif(x < 100):
    print("x is positive but less than 100")
else:
    print("x is greater or equal to 100")

print("done")
```

Java:

```
int x = 4;
System.out.println("testing x....");

if( x < 0 )
    System.out.println("x is negative");
else if( x < 10 )
    System.out.println("x is positive but less than
else if( x < 100 )
    System.out.println("x is positive but less than
else
    System.out.println("x is greater or equal to 10
System.out.println("done");
```

5

Flow of control: conditionals if, if/then, if/then/else

Computer Science

Compound statements

Any single statement can be replaced by a compound statement, consisting of a sequence of statements delimited by curly braces, instead of indentation, to indicate that all these statements should be executed in sequence:

Python:

```
x = 4

print("testing x....")

if( (x % 2) == 0):
    print("x is even")
    print("proceeding to divide x by 2")
    x = x / 2;

print("done")
```

Java:

```
int x = 4;
System.out.println("testing x....");

if( (x % 2) == 0 ) {
    System.out.println("x is even");
    System.out.println("proceeding to divide
    x = x / 2;
}

System.out.println("done");
```

6

Flow of control: compound statements

Computer Science

Compound statements

Any single statement can be replaced by a compound statement, consisting of a sequence of statements delimited by curly braces, instead of indentation, to indicate that all these statements should be executed in sequence:

Python:

```
print("testing x....")

if( (x % 2) == 0):
    print("x is even")
    print("proceeding to divide x by 2")
    x = x / 2
else:
    print("x is odd")
    print("proceeding to add 1 to x")
    x += 1

print("done")
```

Java:

```
System.out.println("testing x....");

if( (x % 2) == 0 ) {
    System.out.println("x is even");
    System.out.println("proceeding to divide x by 2");
    x = x / 2;
}
else {
    System.out.println("x is odd");
    System.out.println("proceeding to add 1 to x");
    x += 1;
}

System.out.println("done");
```

Flow of control: compound statements

Computer Science

Programming Tip: I strongly recommend that you ALWAYS use parentheses, even to enclose a single statement:

Don't:

```
if(x < 0)
    System.out.println("x is negative");
else
    System.out.println("x is positive");
```

Do:

```
if(x < 0) {
    System.out.println("x is negative");
}
else {
    System.out.println("x is positive");
}
```

8

Flow of control: compound statements

Computer Science

Why? Because if you decide to add code later, you will avoid a nasty bug (especially if you are used to Python's grouping by indentation):

What you want:

```
if(x < 0) {
    System.out.println("x is negative");
}
else {
    System.out.println("x is positive");
    x = x * -1;
}
```

Not what you want!

```
if(x < 0)
    System.out.println("x is negative");
else
    System.out.println("x is positive");
    x = x * -1;
```

Will be executed no matter what!

Flow of control: loops **while** and **for**

Computer Science

while loop:

Python:

```
x = 6
while(x < 10):
    print(x)
    x += 1
```

Java:

```
x = 6;
while( x < 10 ) {
    System.out.println( x );
    x += 1;
}
```

for loop:

Python:

```
for y in range(6,10):
    print(y)
```

Java:

```
for(int y = 6; y < 10; ++y)
    System.out.println(y);
```

Flow of control: loops **while** and **for**

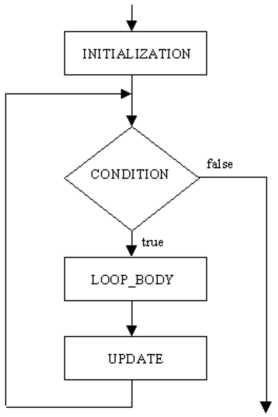
Computer Science

The **for** loop syntax is very different from Python:

Initialization: executed once before loop body

Condition: test at beginning of loop (exactly like while loop)

Update: executed once at end of loop body




```
for(int x = 6; x < 10; ++x) {
    System.out.println(x);
}
```

Similar to:

```
int x = 6;
while(x < 10) {
    System.out.println(x);
    ++x;
}
```

11

Flow of control: **break** and **continue**

Computer Science

More complex examples of **for** loops:

Multiple declarations and multiple updates:

```
System.out.println("Powers of two:");

for(int x = 0, p = 1; x < 11; ++x, p = p * 2) {
    System.out.println(x + "\t" + p);
}
```

```
> run SampleProgram
Powers of two:
0       1
1       2
2       4
3       8
4       16
5       32
6       64
7       128
8       256
9       512
10      1024
>
```

Same as:

```
int x = 0;
for(int p = 1; x < 11; p = p * 2) {
    System.out.println(x + "\t" + p);
    ++x;

}
```

```
int x = 0;
int p = 1;
while(x < 11) {
    System.out.println(x +
    ++x;
    p = p * 2;
}
```

Flow of control: **break** and **continue**

Computer Science

break and continue work exactly the same as in Python!

break jumps out of the loop entirely; **continue** jumps to end of loop and continues with next iteration:

```
x = 0
while(x < 10):
    print(x)
    if(x == 5):
        break
    x += 1
```

```
x = 0;
while( x < 10 ) {
    System.out.println( x );
    if( x == 5 )
        break;
    x += 1;
}
```

```
x = 0
while(x < 10):
    print(x)
    if(x == 5):
        x = 7
        continue
    x += 1
```

```
x = 0;
while( x < 10 ) {
    System.out.println( x );
    if( x == 5 ) {
        x = 7;
        continue;
    }
    x += 1;
}
```

13

Flow of control: loops: break, continue

Computer Science

Note carefully: when executing a continue in a **for** loop, the next statement to be executed is the update:

```
for(int x = 6; x < 10; ++x) {

    if(x == 5) {
        x = 7;
        continue;
    }

    System.out.println(x);

    // update ++x executed here
}
```

14

Local Variables and Scope

Computer Science

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```
{

    if(x > 7)                 Not Legal!
        ++x;                  Materials/SampleProgram.java:10: cannot find symbol
                              symbol  : variable x

    int x = 5;

    if(x > 7)                 Legal!
        ++x;

    int x = 2;                Not Legal!
                              Materials/SampleProgram.java:17: x is already defined in
                              main(java.lang.String[])
}
```

Scope of x

15

Local Variables and Scope

Computer Science

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```
public static void main(String[] args) {

    System.out.println("Powers of Two");

    int x = 0;
    while(x < 11) {
        int p = Math.pow(2.0,x);
        System.out.println(x + "\t" + p)
    }

    int j = 2;
    System.out.println( j );
}
```

Scope of x

16

Local Variables and Scope

Computer Science

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```
public static void main(String[] args) {

    System.out.println("Powers of Two");

    int x = 0;
    while(x < 11) {
        int p = Math.pow(2.0,x);
        System.out.println(x + "\t" + p)
    }

    int j = 2;
    System.out.println( j );
}
```

Scope of j

17

Local Variables and Scope

Computer Science

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```
public static void main(String[] args) {

    System.out.println("Powers of Two");

    int x = 0;
    while(x < 11) {
        int p = Math.pow(2.0,x);
        System.out.println(x + "\t" + p)
    }

    int j = 2;
    System.out.println( j );
}
```

Scope of p

18

Local Variables and Scope

Computer Science

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

Note carefully how this works in a **for** loop – the scope of the initialization variable includes the condition, the update, and the whole loop body:

Scope of i

```
public static void main(String[] args) {

    System.out.println("Powers of Two");

    for(int i = 0; i < 11; ++i) {
        int p = (int) Math.pow(2.0,i);
        System.out.println(i + "\t" + p);
    }

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

```
Materials/SampleProgram.java:13: cannot find symbol
symbol  : variable i
```

19

Flow of control: loops: break, continue

Computer Science

This leads us to a useful idiom: when you need to refer to the initialization after the **for** loop, declare it before the loop:

```
public static void main(String[] args) {

    System.out.println("Powers of Two");

    int i = 0;
    for( ; i < 11; ++i) {
        int p = (int) Math.pow(2.0,i);
        System.out.println(i + "\t" + p);
    }

    System.out.println("Now i has value: " + i);
}
```

```
> run SampleProgram
Powers of Two
0       1
1       2
2       4
3       8
4       16
5       32
6       64
7       128
8       256
9       512
10      1024
Now i has value: 11
```

20