CS 106A, Lecture 6
Scope and Parameters

suggested reading:

Java Ch. 5.1-5.4
Plan For Today

• Announcements
• Recap: Control Flow in Java
• Nested Loops
• Methods in Java
• Scope
• Parameters
Announcements

• Sections are finalized *today at 5PM*
  – Email *cs198@cs.stanford.edu* if you have a schedule conflict with your current section
  – Fill out Annie’s form if you have a partner in mind and want to swap into their section
Plan For Today

• Announcements
• Recap: Control Flow in Java
• Nested Loops
• Methods in Java
• Scope
• Parameters
Conditions in Java

```java
while(condition) {
    body
}

if(condition) {
    body
}
```

The condition should be a “boolean” which is either `true` or `false`
Booleans

1 < 2

true
# Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>1 + 1 == 2</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>3.2 != 2.5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>10 &lt; 5</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>10 &gt; 5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>126 &lt;= 100</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>5.0 &gt;= 5.0</td>
<td>true</td>
</tr>
</tbody>
</table>

* All have equal precedence
Practice: Sentinel Loops

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for numbers until the user types -1, then output the sum of the numbers.
  - In this case, -1 is the sentinel value.

Type a number: 10
Type a number: 20
Type a number: 30
Type a number: -1
Sum is 60
// fencepost problem!
// ask for number - post
// add number to sum - fence

int sum = 0;
int num = readInt("Enter a number: ");
while (num != -1) {
    sum += num;
    num = readInt("Enter a number: ");
}
println("Sum is " + sum);
// Solution #2: "break" out of the loop
// ONLY appropriate to use in fencepost cases

int sum = 0;
while (true) {
    int num = readInt("Enter a number: ");
    if (num == -1) {
        break; // immediately exits loop
    }
    sum += num;
}
println("Sum is " + sum);

Colin prefers this solution, but the debate of how to solve the “loop-and-a-half” problem has been raging for >50 years!
Compound Expressions

In order of precedence:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>!(2 == 3)</td>
<td>true</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>(2 == 3) &amp;&amp; (-1 &lt; 5)</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
</tbody>
</table>

Cannot "chain" tests as in algebra; use && or || instead

// assume x is 15
2 <= x <= 10
true  <= 10
Error!

// correct version
2 <= x && x <= 10
true && false
false
// Store expressions that evaluate to true/false
boolean x = 1 < 2; // true
boolean y = 5.0 == 4.0; // false

// Directly set to true/false
boolean isFamilyVisiting = true;
boolean isRaining = false;

// Ask the user a true/false (yes/no) question
boolean playAgain = readBoolean("Play again?", "y", "n");
if (playAgain) {
	...
}
Runs the first group of statements if `condition1` is true; otherwise, runs the second group of statements if `condition2` is true; otherwise, runs the third group of statements.

You can have multiple else if clauses together.
```java
int num = readInt("Enter a number: ");
if (num > 0) {
    println("Your number is positive");
} else if (num < 0) {
    println("Your number is negative");
} else {
    println("Your number is 0");
}
```
For Loops in Java

```java
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!" );
}
```

- This code is run once, just before the for loop starts.
- Repeats the loop if this condition passes.
- This code is run each time the code gets to the end of the 'body.'
// Launch countdown
for (int i = 10; i >= 1; i--) {
    println(i);
}
println("Blast off!");

Output:

10
9
8
...
Blast off!
Plan For Today

• Announcements
• Recap: Control Flow in Java
• Nested Loops
• Methods in Java
• Scope
• Parameters
Nested loops

- **nested loop**: A loop placed inside another loop.

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 10; j++) {
        print("*");
    }
    println();  // to end the line
}
```

- Output:

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

- The outer loop repeats 5 times; the inner one 10 times.
Nested loop question

**Q:** What output is produced by the following code?

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < i + 1; j++) {
        print("*");
    }
    println();
}
```

A. *****  
B. *****  
C. *  
D. 1  
E. 12345

(How would you modify the code to produce each output above?)
Nested loop question 2

• How would we produce the following output?

....1
...22
..333
.4444
55555
• How would we produce the following output?

```plaintext
.....1
...22
..333
 .4444
 55555
```

• Answer:

```plaintext
for (int i = 0; i < 5; i++) {
```
• How would we produce the following output?

....1
...22
..333
 .4444
 55555

• Answer:

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5 - i - 1; j++) {
        print(".");
    }
}
```
• How would we produce the following output?

....1
...22
..333
.4444
55555

• Answer:

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5 - i - 1; j++) {
        print(".");
    }
    for (int j = 0; j <= i; j++) {
        print(i + 1);
    }
}
```
Nested loop question 2

• How would we produce the following output?
  
  ....1  
  ...22  
  ..333  
  .4444  
  55555

• Answer:
  
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5 - i - 1; j++) {
      print(".");
    }
    for (int j = 0; j <= i; j++) {
      print(i + 1);
    }
  }
  println();
Plan For Today

• Announcements
• Recap: Control Flow in Java
• Nested Loops

• Methods in Java
• Scope
• Parameters
We can make new commands (or **methods**) for Karel. This lets us **decompose** our program into smaller pieces that are easier to understand.

```java
private void turnRight() {
    turnLeft();
    turnLeft();
    turnLeft();
}
```

For example:

```java
private void turnRight() {
    turnLeft();
    turnLeft();
    turnLeft();
    turnLeft();
}
```
We can define new **methods** in Java just like in Karel:

```java
private void name() {
    statement;
    statement;
    ...
}
```

For example:

```java
private void printGreeting() {
    println("Hello world!");
    println("I hope you have a great day.");
}
```
public void run() {
    int x = 2;
    printX();
}

private void printX() {
    // ERROR!  "Undefined variable x"
    println("X has the value " + x);
}
Plan For Today

• Announcements
• Recap: Control Flow in Java
• Nested Loops
• Methods in Java
• Scope
• Parameters
A Variable love story

By Chris Piech
Once upon a time...
There was a variable named x

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

5

5

x
...x was looking for love!

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

x was definitely looking for love.
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

Hi, I’m y
“Wow!”
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

Wow

\[
\begin{align*}
\text{Wow} & \quad \begin{array}{c}
5 \\
\backslash
\end{array} & \quad \begin{array}{c}
5 \\
\backslash
\end{array} \\
\begin{array}{c}
x \\
\backslash
\end{array} & \quad \begin{array}{c}
y \\
\backslash
\end{array}
\end{align*}
\]
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);

We have so much in common
```
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

We both have value 5!
```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

Maybe sometime we can...
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

println together?
It was a beautiful match...
...but then tragedy struck.
Tragedy Strikes

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

5
x

5
y
```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```
Noooooooooooooooooo!
You see...
when a program exits a code block, all variables declared inside that block go away!
Since y is inside the if-block...

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
...and doesn’t exist here.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
}
println(x + y);
```

Error. Undefined variable y.
The End
Sad times 😞
Variable Scope

- The **scope** of a variable refers to the section of code where a variable can be accessed.
- **Scope starts** where the variable is declared.
- **Scope ends** at the termination of the code block in which the variable was declared.

- A **code block** is a chunk of code between `{ }` braces.
Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ... some code
    }
    ... some other code
}
```
Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ... some code
    }
    ... some other code
}
```
Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ... some code
    }
    ... some other code
}
```

Comes to life here
Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ...
        ... some code
    }
    ...
    ... some other code
}
```

This is the inner most code block in which it was declared....
Variable Scope

Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ... some code
    }
    ... some other code
}
```

Still alive here...
Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ... some code
    }
    ... some other code
}
```

It goes away here (at the end of its code block)
Variable Scope

Variables have a lifetime (called scope):

```java
public void run() {
    double v = 8;
    if (condition) {
        v = 4;
        ... some code
    }
    ... some other code
}
```

It goes away here (at the end of its code block)
Variables have a lifetime (called scope):

```java
public void run() {
    ... some code
    if (condition) {
        int w = 4;
        ... some code
    }
    ... some other code
}
```

- `w` is created here:
- `w` goes away here (at the end of its code block)
Variables have a lifetime (called scope):

```java
public void run() {
    ... some code
    if (condition) {
        int w = 4;
        ... some code
    }
    ... some other code
}
```
This is the scope of \( w \)
You cannot have two variables with the same name in the same scope.

```java
for (int i = 1; i <= 100; i++) {
    int i = 2;              // ERROR
    print("/");
}
```
You *cannot* have two variables with the same name in the same scope.

```c
for (int i = 1; i <= 100; i++) {
    while (...) {
        int i = 5; // ERROR
    }
}
```
You can have two variables with the same name in separate scopes.

```java
public void run() {
    for (int i = 0; i < 5; i++) {
        int w = 2;
        // i ok here
        // w ok here
    }

    for (int i = 0; i < 2; i++) {
        int w = 3;
        // i ok here
        // w ok here
    }
}
```
You *can* have two variables with the same name in *separate* scopes.

```java
public void run() {
    int num = 5;
    cow();
    println(num); // prints 5
}

private void cow() {
    int num = 10;
    println(num); // prints 10
}
```
Variable Scope

You *can* have two variables with the same name in *different* scopes.

```java
public void run() {
    int num = 5;
    cow();
    println(num);
}

private void cow() {
    int num = 10;
    println(num);
}
```
You *can* have two variables with the same name in *different scopes*.

```java
public void run() {
    int num = 5;
    cow();
    println(num);
}

private void cow() {
    int num = 10;
    println(num);
}
```
You can have two variables with the same name in different scopes.

```java
public void run() {
    int num = 5;
    cow();
    println(num);
}

private void cow() {
    int num = 10;
    println(num);
}
```
You *can* have two variables with the same name in *different scopes*.

```java
public void run() {
    int num = 5;
    cow();
    println(num);
}

private void cow() {
    int num = 10;
    println(num);
}
```
You *can* have two variables with the same name in *different scopes*.

```java
public void run() {
    int num = 5;
    cow();
    println(num);
}

private void cow() {
    int num = 10;
    println(num);
}
```
Revisiting Sentinel Loops

// sum must be declared outside of the loop!  
// Otherwise, it will be redeclared many times  
// num must be declared outside of the loop!  
// Otherwise, the loop condition makes no sense

int sum = 0;

int num = readInt("Enter a number: ");

while (num != -1) {
    sum += num;
    num = readInt("Enter a number: ");
}

println("Sum is " + sum);
Revisiting Sentinel Loops

// Here, num goes out of scope at the end of each loop iteration. At that point, we have already used its value and can discard it.

int sum = 0;
while (true) {
    int num = readInt("Enter a number: ");
    if (num == -1) {
        break; // immediately exits loop
    }
    sum += num;
}
println("Sum is " + sum);
Chapter 2
By Chris
The programmer fixed the bug
There was a variable named x.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

\[ x = 5 \]
...x was looking for love!

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

x was definitely looking for love
And met y.

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

\[ x = 5 \quad \text{and} \quad y = 5 \]
Since they were both “in scope”...

```java
int x = 5;
if (lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

5
---

x

5
---

y
…they lived happily ever after.
The end.
Plan For Today

- Announcements
- Recap: Control Flow in Java
- Nested Loops
- Methods in Java
- Scope
- Parameters
public void run() {
    int x = 2;
    printX();
}

private void printX() {
    // ERROR!  "Undefined variable x"
    println("X has the value " + x);
}
Parameters

Parameters let you provide a method some information when you are calling it.
Methods = Toasters
Methods = Toasters

parameter
Methods = Toasters

parameter
Methods = Toasters

parameter
Methods = Toasters

drawings of bread and toaster with a parameter label
Methods = Toasters

Invalid parameter
Methods = Toasters
• Consider the task of printing the following boxes:

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

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

– The code to draw each box will be very similar.
  • Would variables help? Would constants help?
Wouldn’t it be nice if…

drawBox(10, 4);

Continued next time…