CS 106A, Lecture 6
Control Flow and Parameters

suggested reading:

Java Ch. 5.1-5.4

Plan For Today

- Announcements
- Recap: If and While in Java
- For Loops in Java
- Methods in Java
- Scope

HW2 Cutoff

- Parameters

Plan For Today

- Announcements
- Recap: If and While in Java
- For Loops in Java
- Methods in Java
- Scope
- Parameters

Conditions in 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

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

* All have equal precedence

Relational Operators

```
if (1 < 2) {
    println("1 is less than 2!");
}
```

```
int num = readInt("Enter a number: ");
if (num == 0) {
    println("That number is 0!");
} else {
    println("That number is not 0.");
}
```

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
```

Practice: Sentinel Loops

```
// 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);
```

Practice: Sentinel Loops

```
// Solution #2 (ok, but #1 is better)


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

Compound Expressions

In order of precedence:

Operator	Description	Example	Result
`!`	not	`!(2 == 3)`	`true`
`&&`	and	`(2 == 3) && (-1 < 5)`	`false`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`

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
```

Boolean Variables

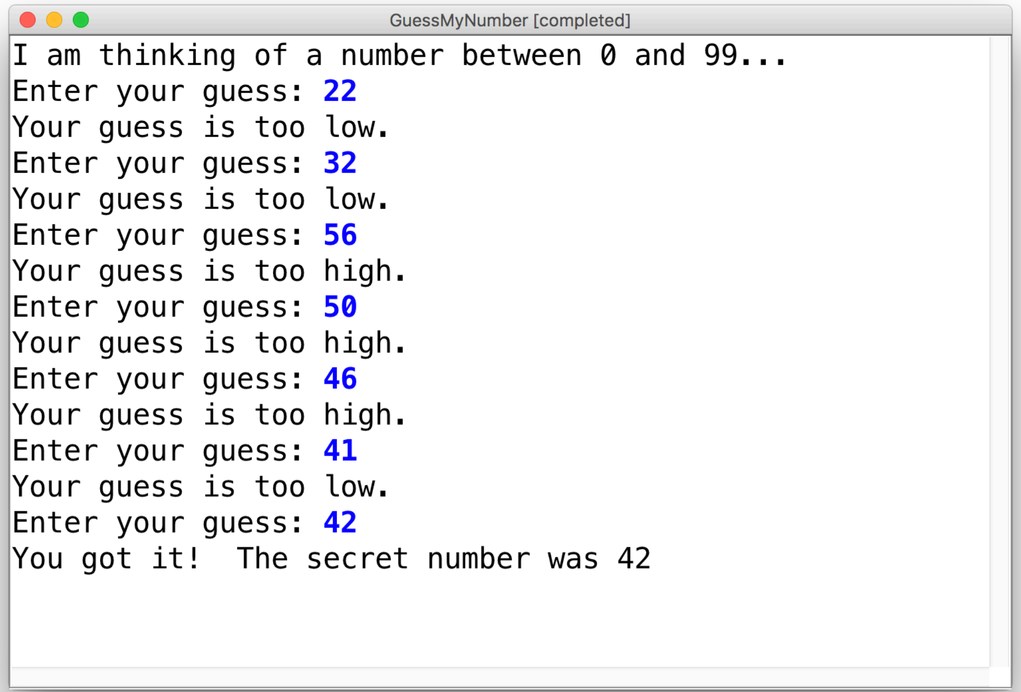
```
// Store expressions that evaluate to true/false
boolean x = 1 < 2;            // true
boolean y = 5.0 == 4.0;       // false

// 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) {
...
```

Practice: GuessMyNumber

- We wrote a program called *GuessMyNumber* that prompts the user for a number until they guess our secret number.
- If a guess is incorrect, the program provides a hint; specifically, whether the guess is too high or too low.



GuessMyNumber [completed]

```
I am thinking of a number between 0 and 99...
Enter your guess: 22
Your guess is too low.
Enter your guess: 32
Your guess is too low.
Enter your guess: 56
Your guess is too high.
Enter your guess: 50
Your guess is too high.
Enter your guess: 46
Your guess is too high.
Enter your guess: 41
Your guess is too low.
Enter your guess: 42
You got it!  The secret number was 42
```

If/Else If/Else

```
if (condition1) {
    ...
} else if (condition2) {        // NEW
    ...
} else {
    ...
}
```

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.

If/Else If/Else

```
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");
}
```

Plan For Today

- Announcements
- Recap: If and While in Java
- For Loops in Java
- Methods in Java
- Scope
- Parameters

For Loops in Java

This code is run once, just before the for loop starts

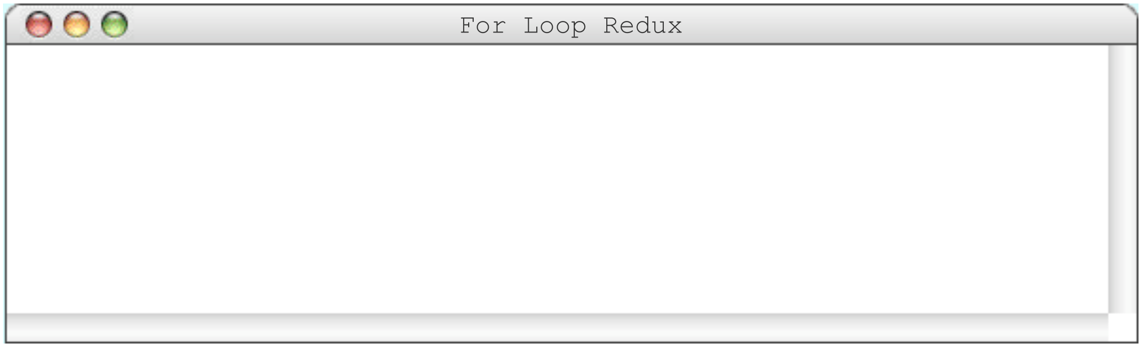
Repeats the loop if this condition passes

This code is run each time the code gets to the end of the 'body'

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

For Loops in Java

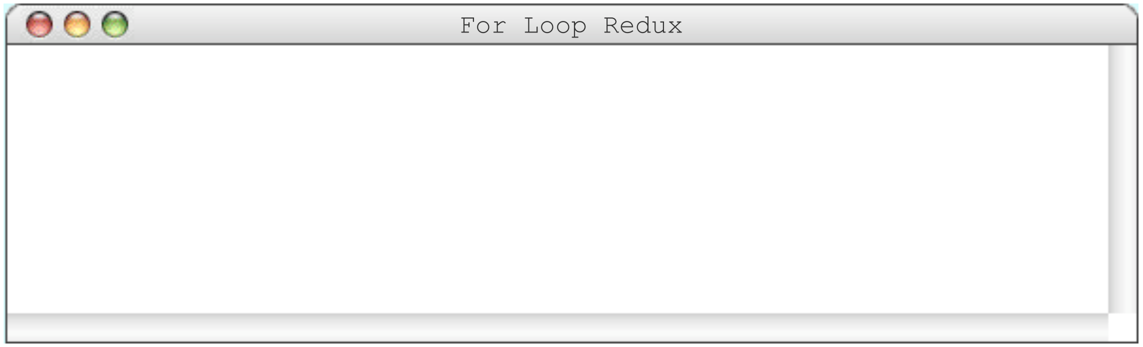
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



For Loops in Java

i 0

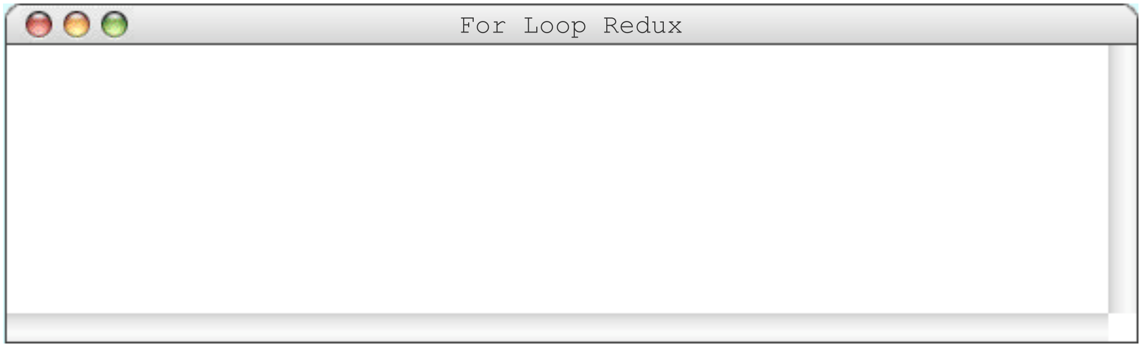
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



For Loops in Java

i $\boxed{0}$

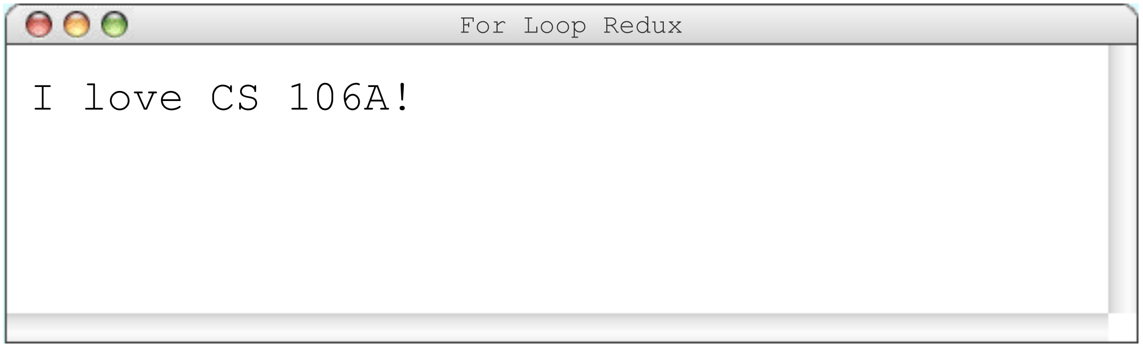
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



For Loops in Java

i $\boxed{0}$

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

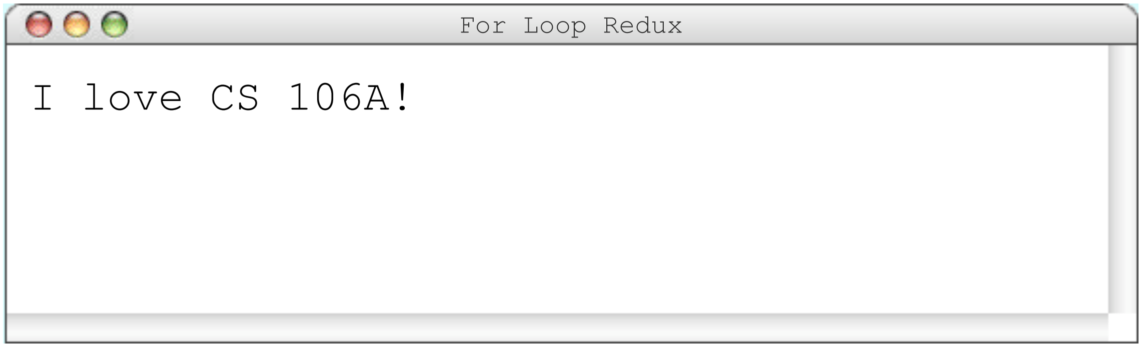


```
I love CS 106A!
```

For Loops in Java

i $\boxed{0}$

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

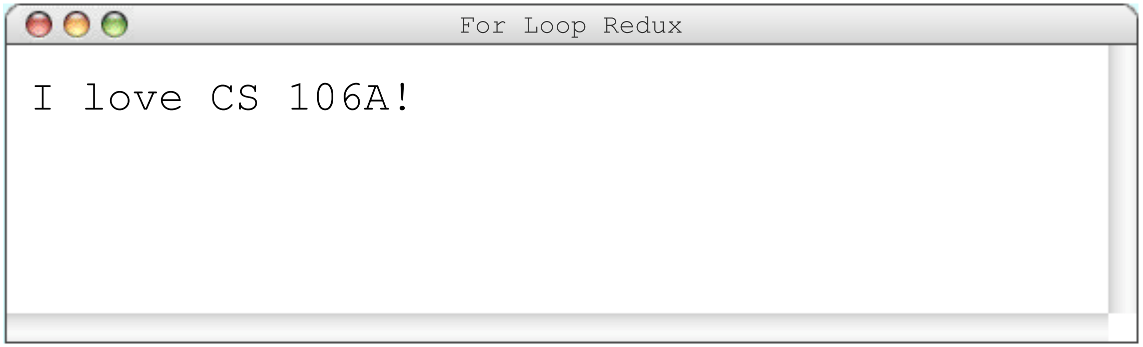


```
I love CS 106A!
```

For Loops in Java

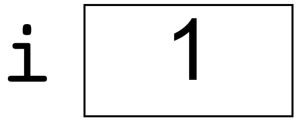
i 1

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

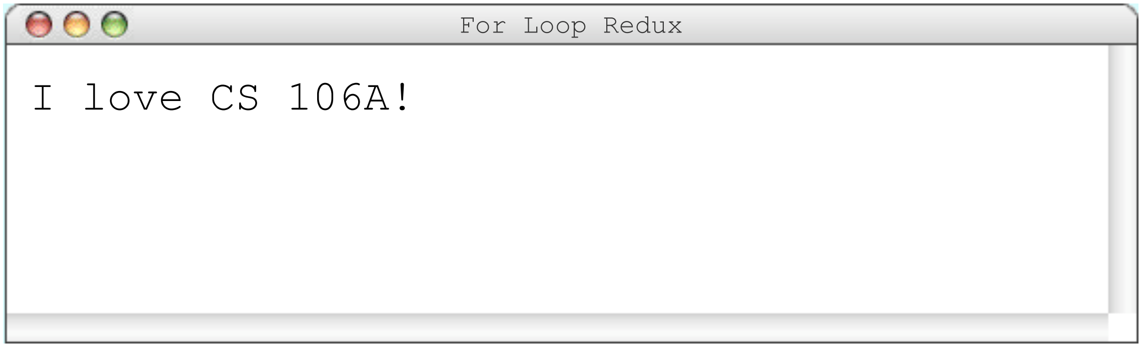


```
I love CS 106A!
```

For Loops in Java



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

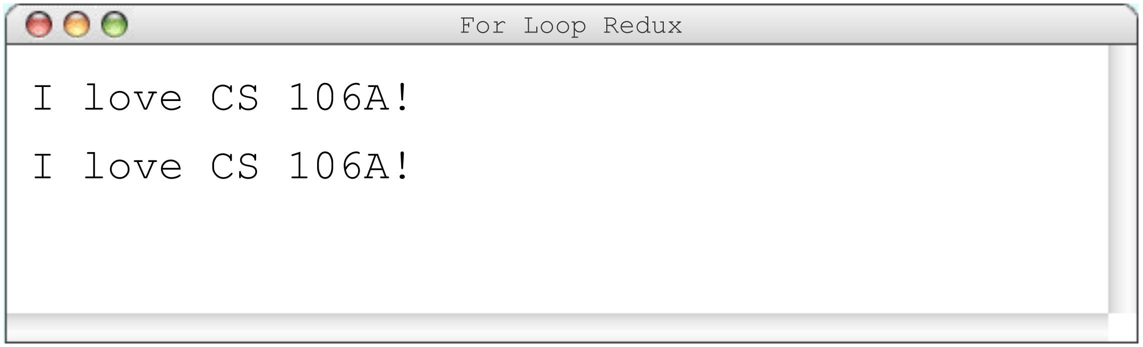


```
I love CS 106A!
```

For Loops in Java

```
i  1
```

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

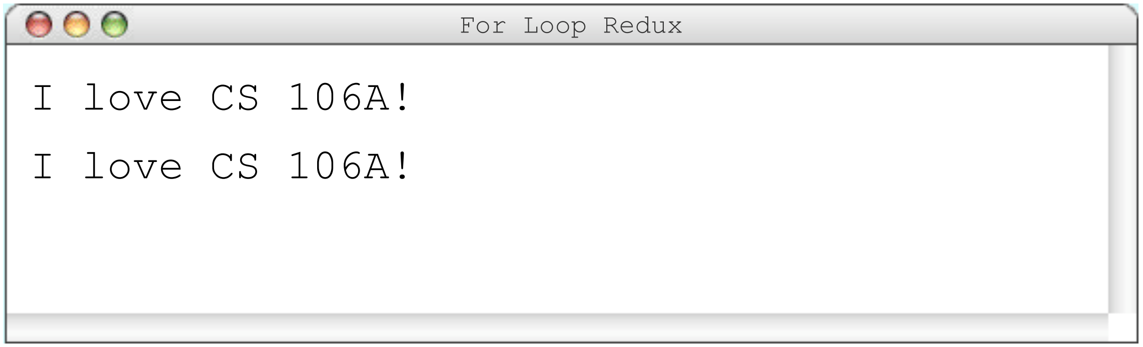


```
I love CS 106A!
I love CS 106A!
```

For Loops in Java

i $\boxed{2}$

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

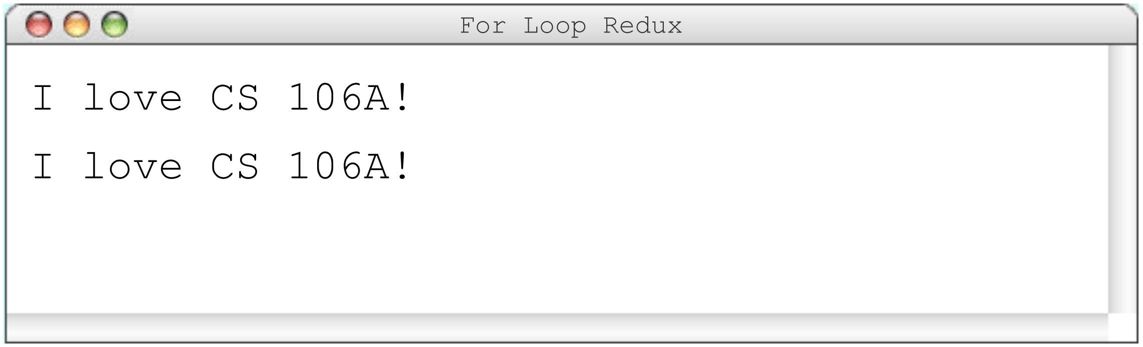


```
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  2
```

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

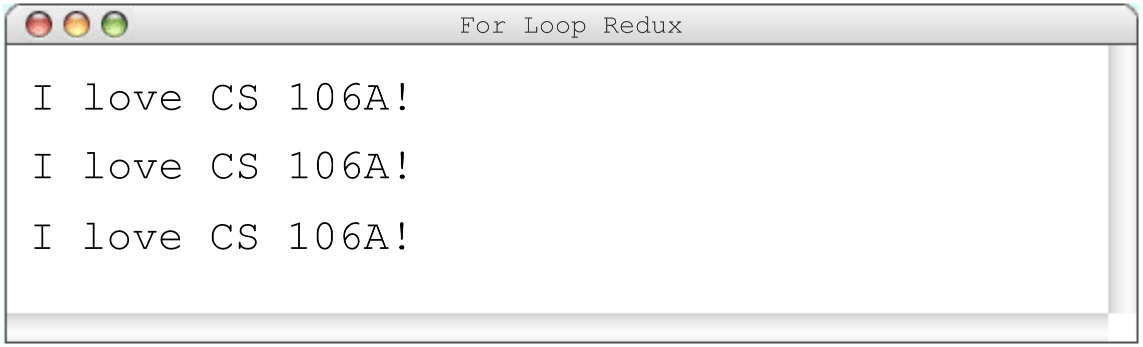


```
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  2
```

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

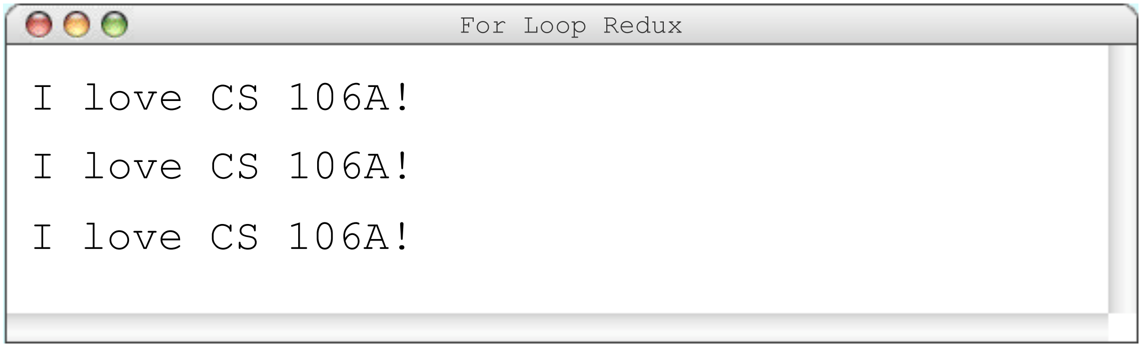


```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  3
```

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

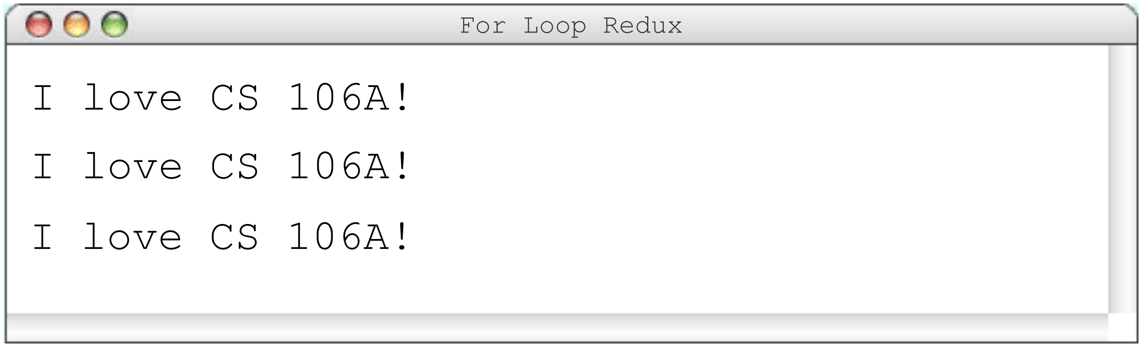


```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

```
i  3
```

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

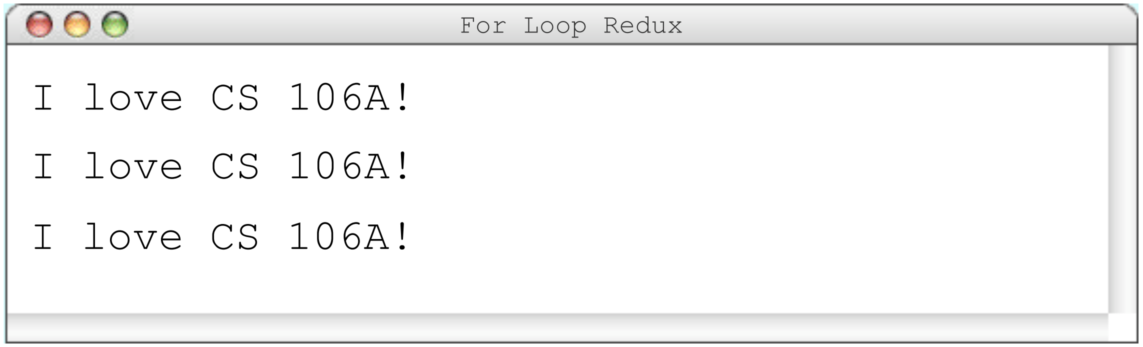


For Loop Redux

```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

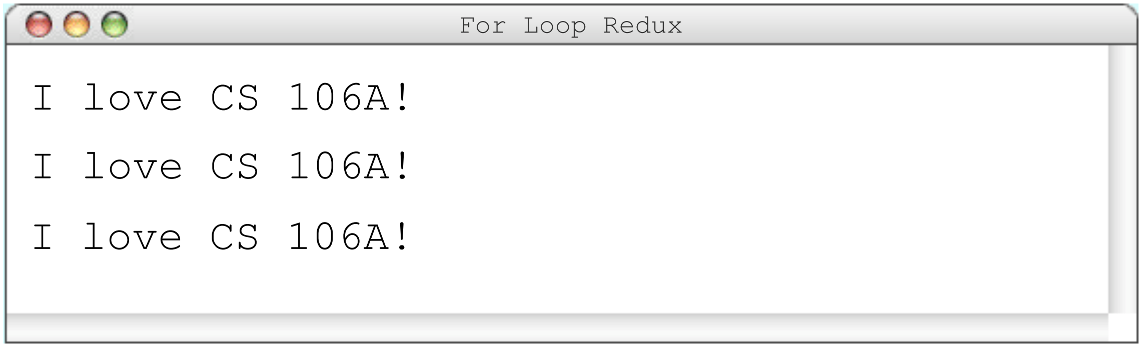
```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```



```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

For Loops in Java

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


For Loop Redux

```
I love CS 106A!
I love CS 106A!
I love CS 106A!
```

Using the For Loop Variable

```
// prints the first 100 even numbers
for(int i = 0; i < 100; i++) {
    println(i * 2);
}
```

Using the For Loop Variable

```
// Launch countdown
for(int i = 10; i >= 1; i--) {
    println(i);
}
println("Blast off!");
```

Output:

```
10
9
8
…
Blast off!
```

Using the For Loop Variable

```
// Adds up 1 + 2 + ... + 99 + 100
int sum = 0;
for(int i = 1; i <= 100; i++) {
     sum += i;
}
println("The sum is " + sum);
```

Nested loops

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

```
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?

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

A.
```
*****
*****
*****
*****
*****
```

B.
```
*****
****
***
**
*
```

C.
```
*
**
***
****
*****
```

D.
```
1
22
333
4444
55555
```

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
```

Nested loop question 2

- How would we produce the following output?

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

- Answer:

```
for (int i = 0; i < 5; i++) {








}
```

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(".");
    }




}
```

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);
    }

}
```

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: If and While in Java
- For Loops in Java
- Methods in Java
- Scope
- Parameters

Defining New Commands in Karel

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

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

For example:

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

Methods in Java

We can define new **methods** in Java just like in Karel:

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

For example:

```
private void printGreeting() {
     println("Hello world!");
     println("I hope you have a great day.");
}
```

Methods in Java

```
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: If and While in Java
- For Loops in Java
- Methods in Java
- Scope
- Parameters

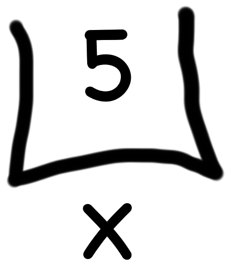
A Variable l♥ve story

By Chris Piech

Once upon a time…

...x was looking for love!

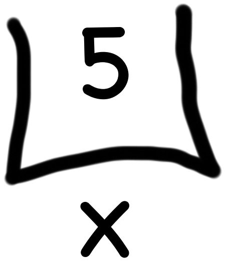
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



...x was looking for love!

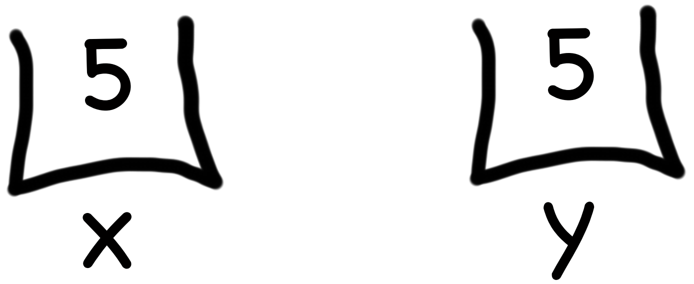
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

x was definitely looking for love



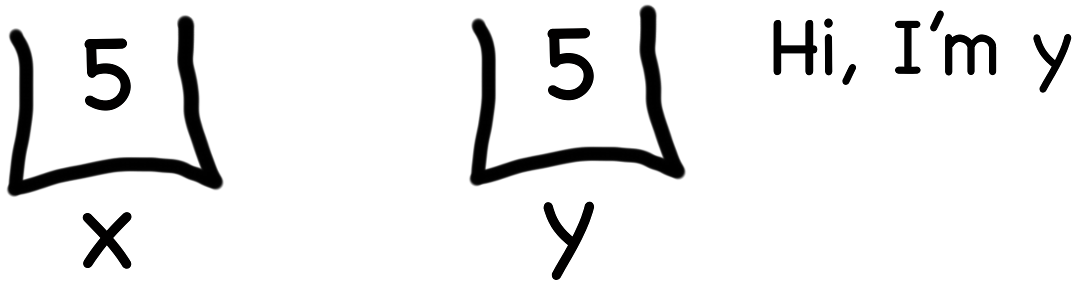
And met y.

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



And met y.

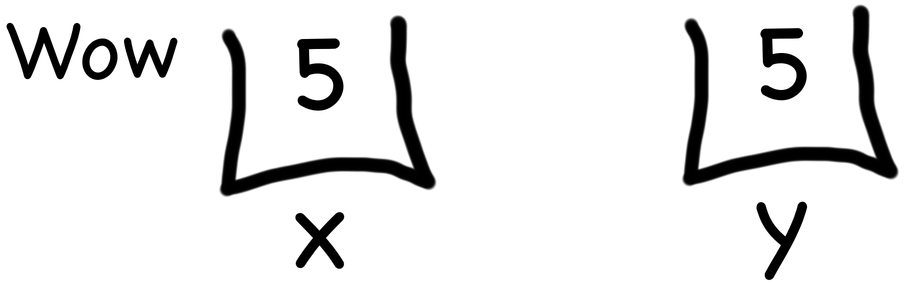
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



“Wow!”

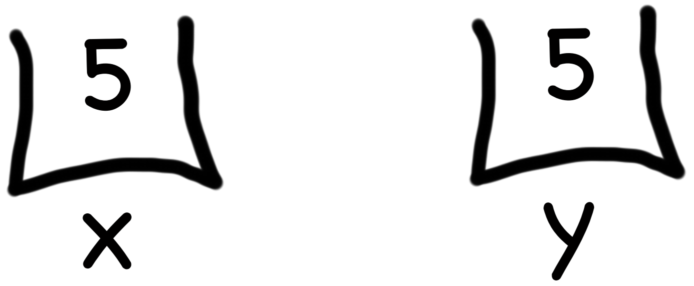
And met y.

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



And met y.

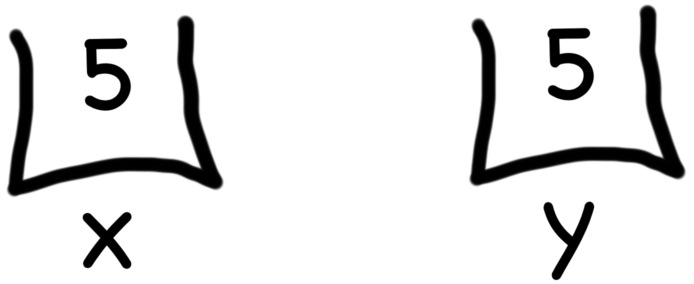
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



We have so much in common

And met y.

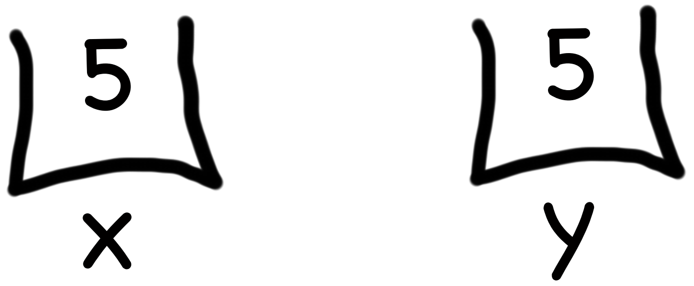
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



We both have value 5!

And met y.

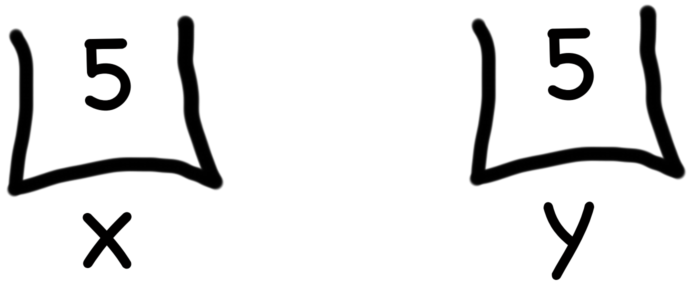
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



Maybe sometime we can...

And met y.

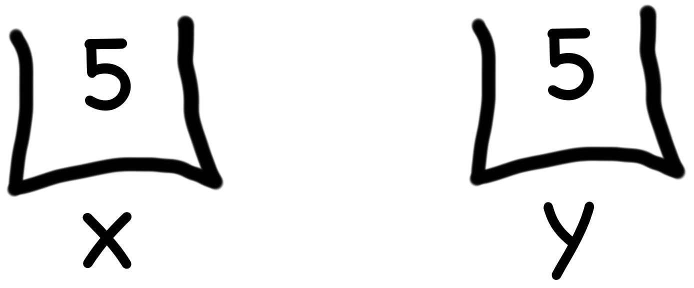
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



println together?

And met y.

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

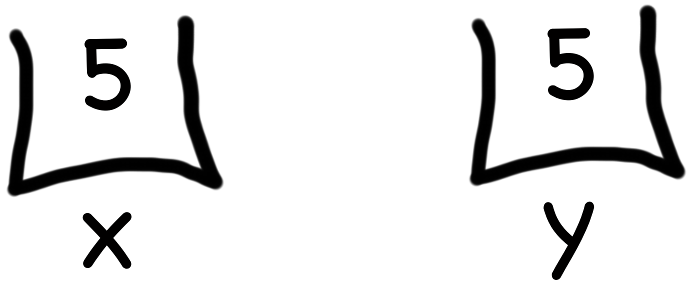


It was a beautiful match…

…but then tragedy struck.

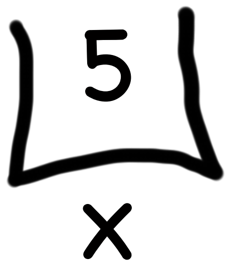
Tragedy Strikes

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



Tragedy Strikes

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

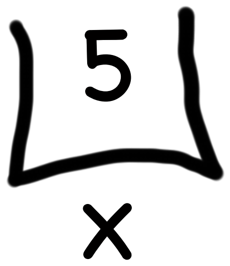


Noooooooooooooooo!

You see…
when a program exits a code block,
all variables declared inside that block go away!

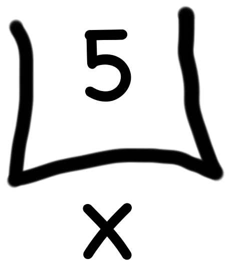
Since y is inside the if-block...

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



...it goes away here...

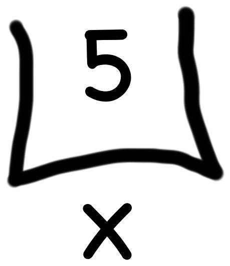
```
int x = 5;
if(lookingForLove()) {
   int y = 5;
}
println(x + y);
```



...and doesn't exist here.

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

Error. Undefined variable y.



The End

Sad times ☹

Variable Scope

Variables have a lifetime (called scope):

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

Variable Scope

Variables have a lifetime (called scope):

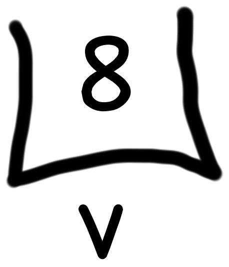
```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Variable Scope

Variables have a lifetime (called scope):

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

Comes to life here

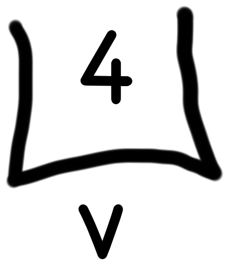


Variable Scope

Variables have a lifetime (called scope):

```
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....

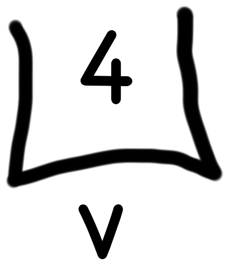


Variable Scope

Variables have a lifetime (called scope):

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

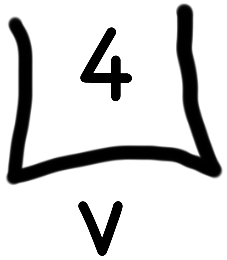
Still alive here...



Variable Scope

Variables have a lifetime (called scope):

```
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):

```
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):

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

This is the scope of **w**

Variable Scope

Variables have a lifetime (called scope):

```
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)

Variable Scope

```
public void run() {
      int x = 2;
      printX();
}

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

A Variable love story

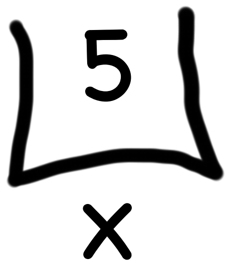
Chapter 2

By Chris

The programmer fixed the bug

...x was looking for love!

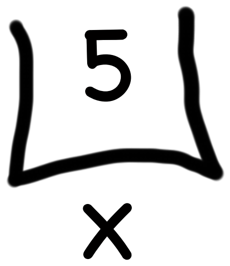
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



...x was looking for love!

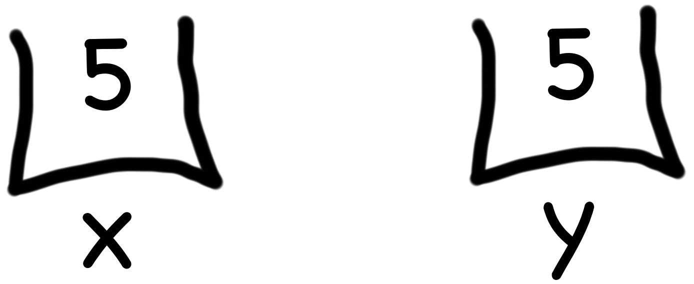
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

x was definitely looking for love



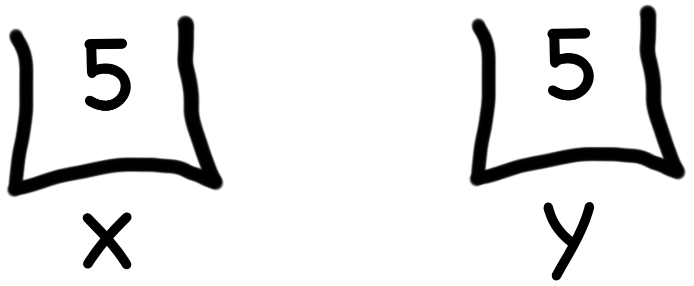
And met y.

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



Since they were both "in scope"…

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



…they lived happily ever after.
The end.

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 { } brackets

Variable Scope

You *cannot* have two variables with the same name in the *same scope*.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                  // ERROR
    print("/");
}
```

Variable Scope

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

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

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

Variable Scope

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

```
private 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*.

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

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

Revisiting Sentinel Loops

```
// sum must be outside the while loop!
// Otherwise it will be redeclared many times.
int sum = 0;
int num = readInt("Enter a number: ");
while (num != -1) {
     sum += num;
     num = readInt("Enter a number: ");
}
println("Sum is " + sum);
```

Plan For Today

- Announcements
- Recap: If and While in Java
- For Loops in Java
- Methods in Java
- Scope
- Parameters

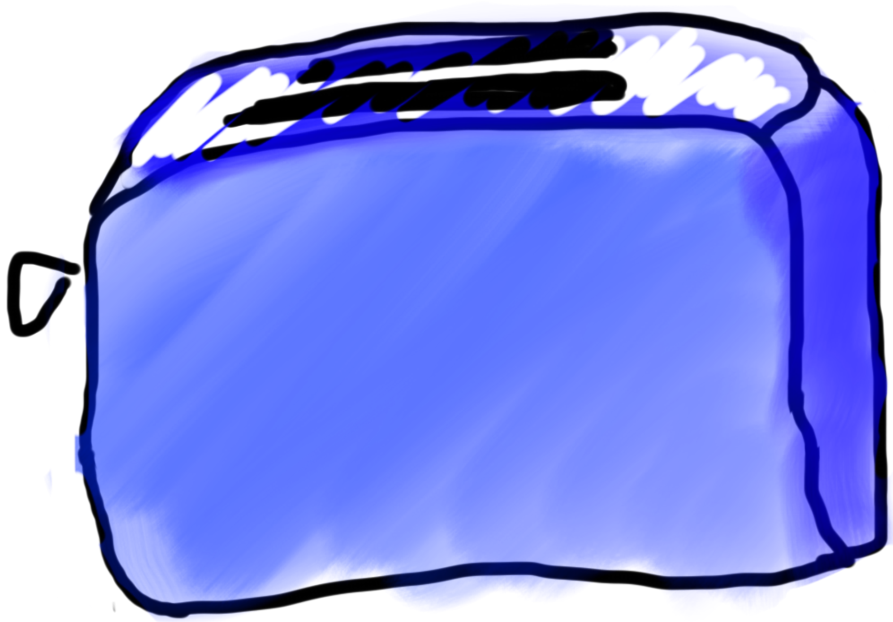
Parameters

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

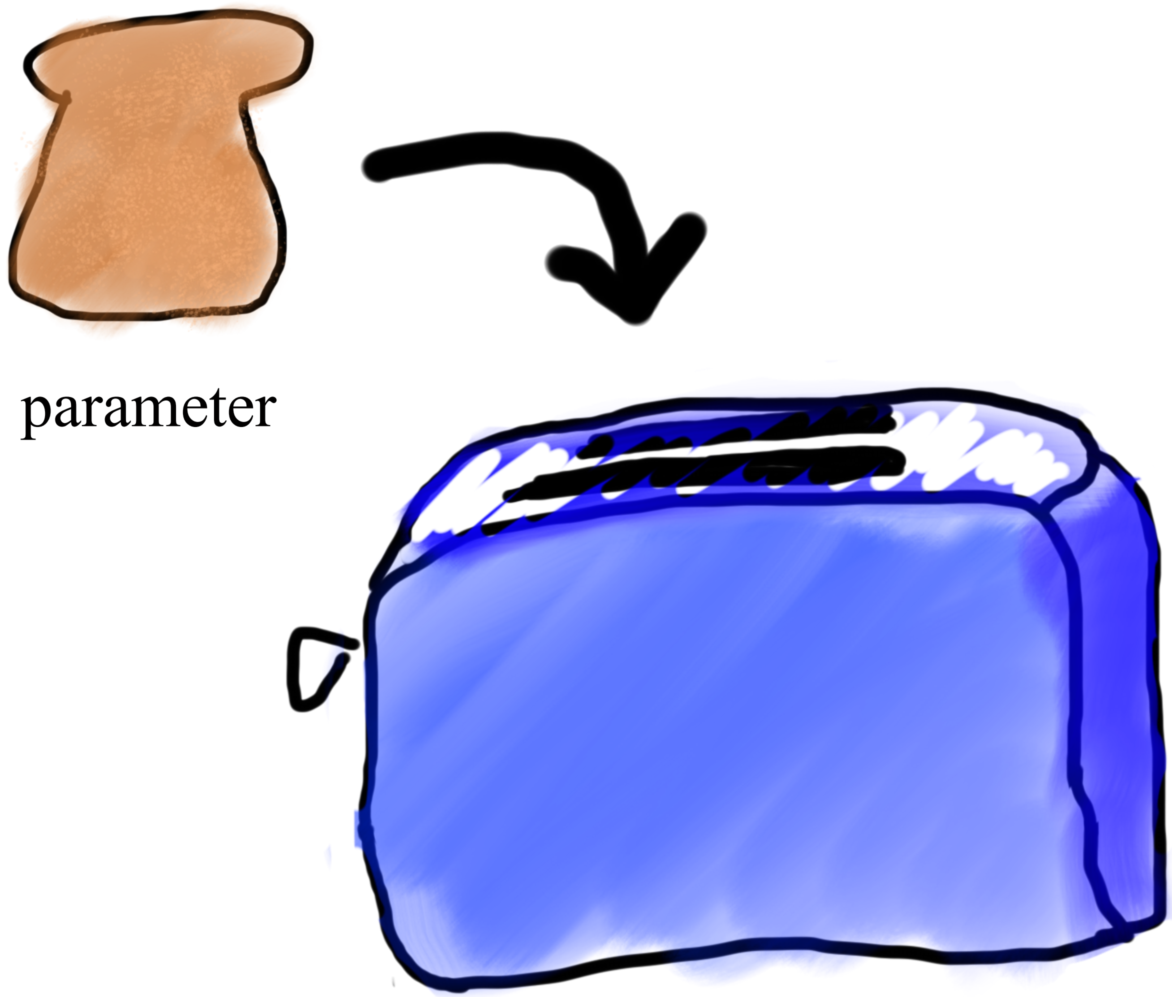
Methods = Toasters

Methods = Toasters

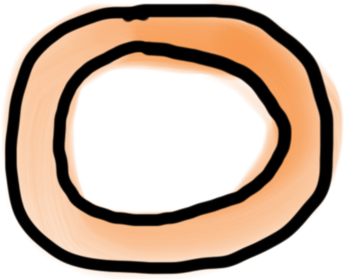
parameter



Methods = Toasters

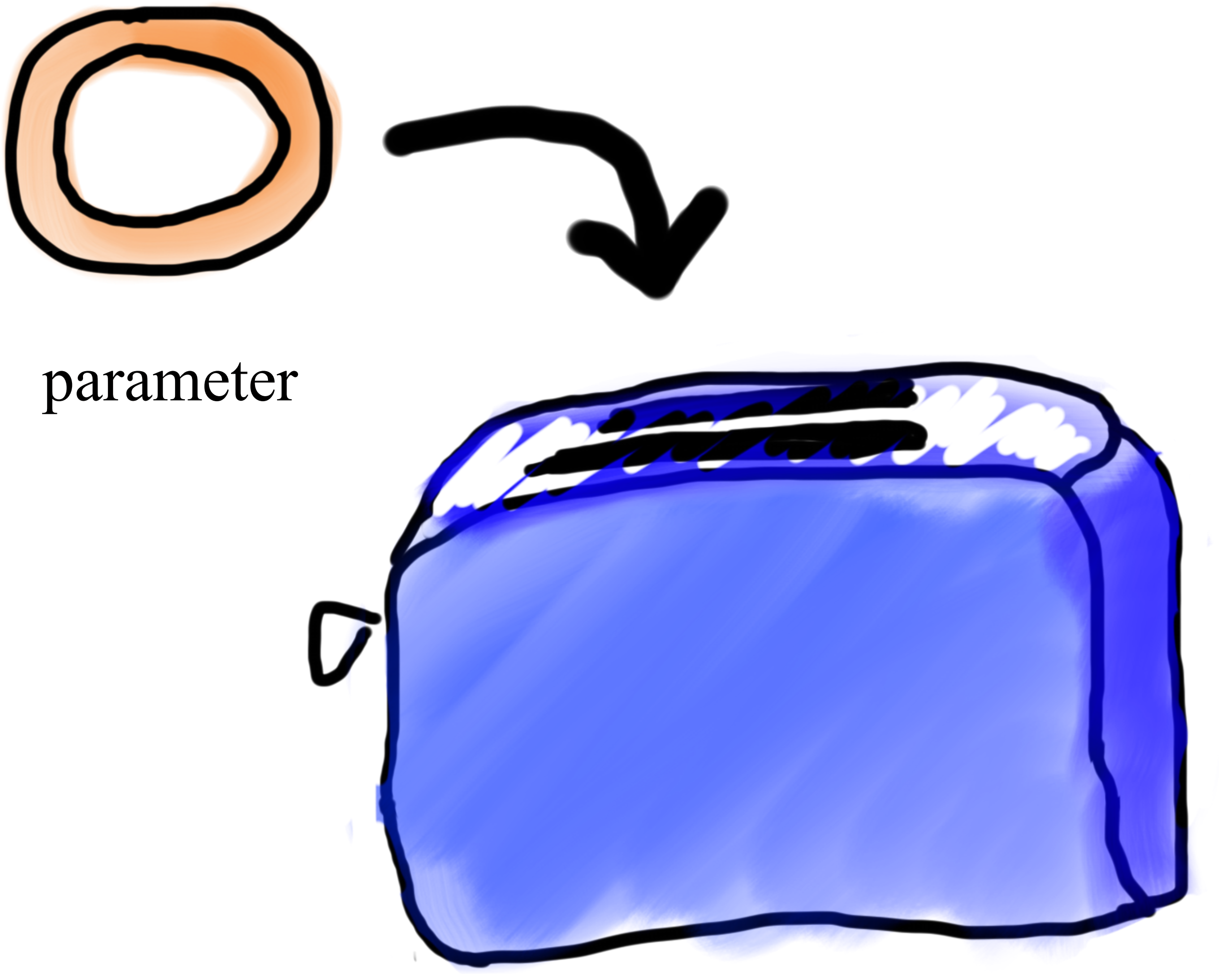


Methods = Toasters

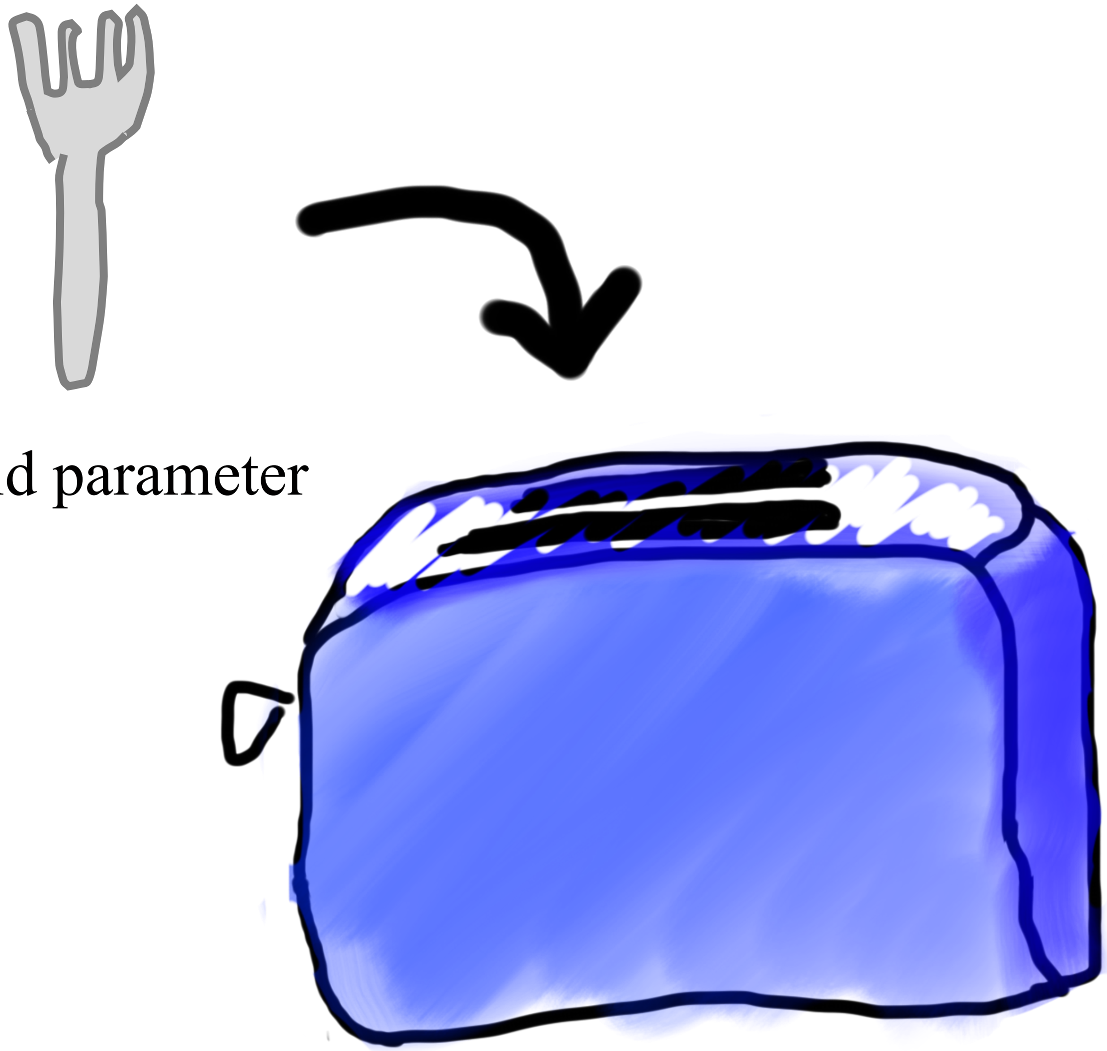


parameter

Methods = Toasters



Methods = Toasters



Invalid parameter

Methods = Toasters

Drawing boxes

- 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);
```

Methods with Parameters

```
private void drawBox(int width, int height) {
     // use width and height to draw box
}
```

Methods with Parameters

```
public void run() {
     printGreeting(5);
}

private void printGreeting(int times) {
     for (int i = 0; i < times; i++) {
          println("Hello world!");
     }
}
```

Methods with Parameters

```
public void run() {
      printGreeting(5)
}

private void printGreeting(int times) {
      for (int i = 0; i < times; i++) {
            println("Hello world!");
      }
}
```

Methods with Parameters

```
public void run() {
     int repeats = 5;
     printGreeting(repeats);
}

private void printGreeting(int times) {
     for (int i = 0; i < times; i++) {
          println("Hello world!");
     }
}
```

Methods with Parameters

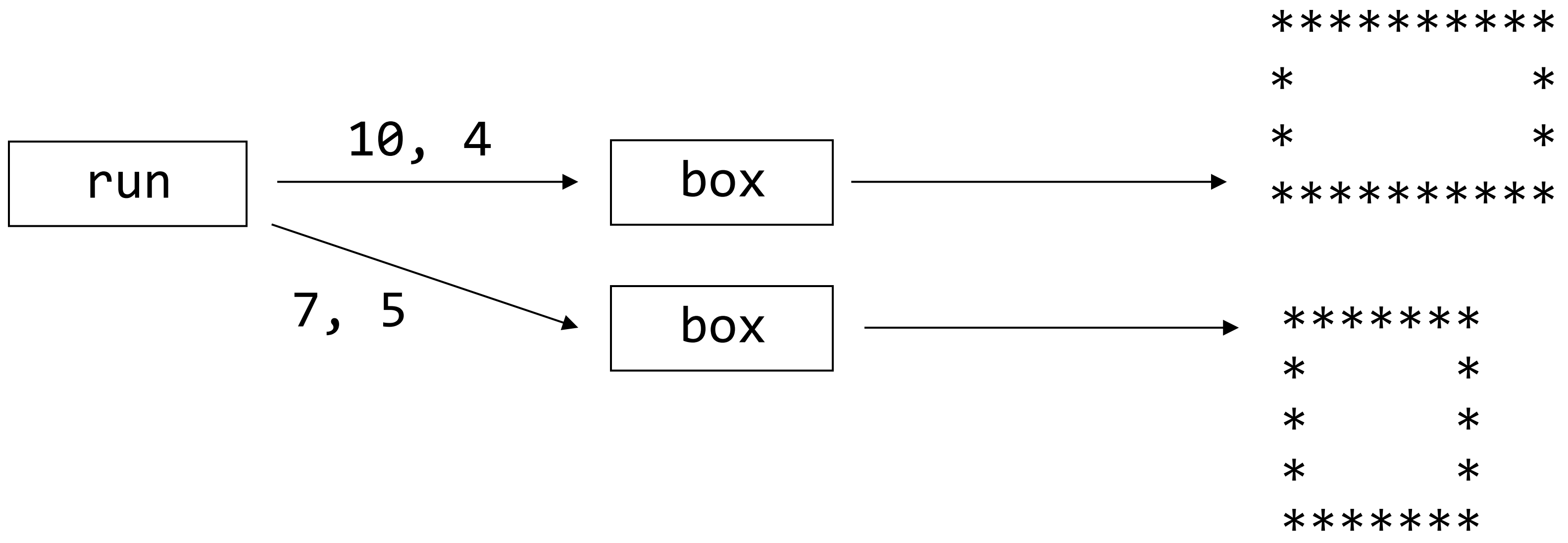
```
public void run() {
    int times = 5;
    printGreeting(repeats);
}

private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}
```

Parameter names do not affect how the program runs!

Parameters

- **parameter**: A value passed to a method by its caller.
  - Write a method **box** to draw a box of any size.
    - When *declaring* the method, we will state that it requires the caller to tell it the width and height of the box.
    - When *calling* the method, we will specify the width and height to use.



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

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

Declaring a parameter

Stating that a method requires a parameter in order to run

```
public void name(type name) {
    statements;
}
```

- Example:

```
public void password(int code) {
    println("The password is: " + code);
}
```

  - When `password` is called, the caller must specify the integer code to print.

Multiple parameters

- A method can accept multiple parameters separated by commas: `,`
  - When calling it, you must pass values for each parameter.

- Declaration:

```
public void name(type name, ..., type name) {
    statements;
}
```

- Call:

```
name(value, value, ..., value);
```

Passing a parameter

Calling a method and specifying values for its parameters

```
methodName(expression);
```

- Example:

```
public void run() {
    password(42);
    password(12345);
}
```

Output:

```
The password is 42
The password is 12345
```

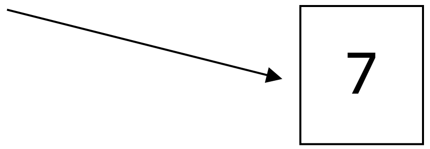
- Illegal to call without passing an `int` for that parameter.

```
password();        // Error
password(3.7);     // Error
```

How params are passed

- When the method is called:
  - The value is stored into the parameter variable.
  - The method's code executes using that value.

```
public void run() {
  chant(7);
}
```



```
public void chant(int times) {
    for (int i = 0; i < times; i++) {
        println("Java is great!");
    }
}
```

Parameters are Copies

```
// NOTE: This program is buggy!!

private void addFive(int x) {
  x += 5;
}


public void run() {
   int x = 3;
   addFive(x);
  println("x = " + x);
}
```

Parameters are Copies

```
// NOTE: This program is buggy!!

private void addFive(int x) {
  x += 5;
}


public void run() {
   int x = 3;
   addFive(x);
  println("x = " + x);    // prints "x = 3"!
}
```

Drawing boxes

- Lets write a program that uses methods and parameters to print the following boxes:

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

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

  - Note: the code to draw each box will be very similar!

Recap

- Announcements
- Recap: If and While in Java
- For Loops in Java
- Methods in Java
- Scope
- Parameters

Next time: Methods and `return`