

CS 106A, Lecture 7

Parameters and Return

suggested reading:

Java Ch. 5.1-5.4

Plan For Today

- Announcements
- Recap: Scope
- Parameters
- Return

Announcements

- Assignment 2 is out!
 - Due next Wednesday, 7/11
 - No partners
 - Use the output comparison tool
 - YEAH (Your Early Assignment Help) hours
 - Friday 7/6, 1:30PM-2:20PM in Gates B01

Plan For Today

- Announcements
- **Recap: Scope**
- Parameters
- Return

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);  
}
```

A Variable love story

By Chris Piech

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

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's scope

w goes away
here (at the end
of its code block)

Variable Scope

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

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

Variable Scope

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

```
public void run() {  
    for (int i = 0; i < 5; i++) {           // i ok here  
        int w = 2;                          // w ok here  
    }  
  
    for (int i = 0; i < 2; i++) {           // i ok here  
        int w = 3;                          // w ok here  
    }  
}
```

Variable Scope

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

```
public void run() {  
    int num = 5;  
    cow();  
    println(num);           // prints 5  
}  
  
private void cow() {  
    int num = 10;  
    println(num);         // prints 10  
}
```

Plan For Today

- Announcements
- Recap: Scope
- **Parameters**
- Return

Parameters

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

Methods = Toasters



parameter



Example: readInt

```
readInt( "Your guess? " );
```


Example: readInt

We call
readInt

We give readInt some
information in parentheses
(the text to print to the user)


`readInt ("Your guess? ");`


Declaring a parameter

Stating that a method requires a parameter in order to run

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

- Example:

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

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

Example: printGreeting

```
printGreeting(5);
```

(Prints a greeting a certain number of times)

Wouldn't it be nice if...

We call
printGreeting

We give printGreeting some
information (the number of
greetings to print)



printGreeting(5);

Methods with Parameters

Tells Java this method
needs one *int* in order to
execute.



```
private void printGreeting(int times) {  
    // use 'times' to print the greeting  
}
```

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
```

Methods with Parameters

```
printGreeting(5);
```



```
private void printGreeting(int times) {  
    // use 'times' to print the greeting  
}
```

Methods with Parameters


```
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!");  
    }  
}
```



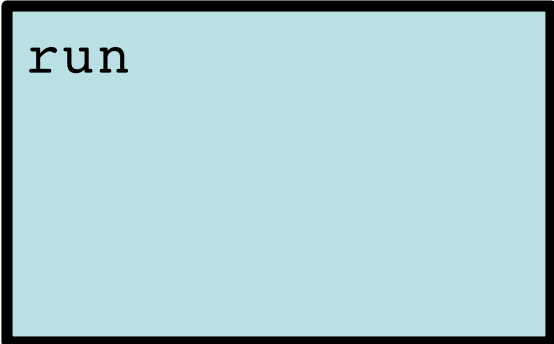
Methods with Parameters

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

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



run

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!");  
    }  
}
```

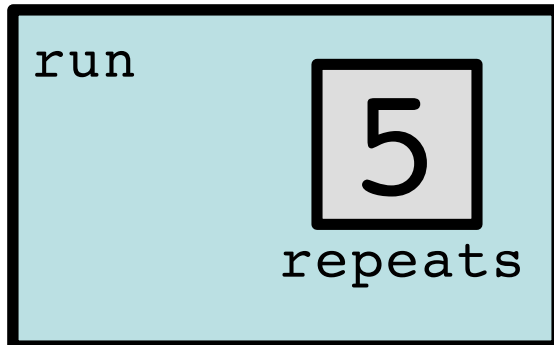


run

Methods with Parameters

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

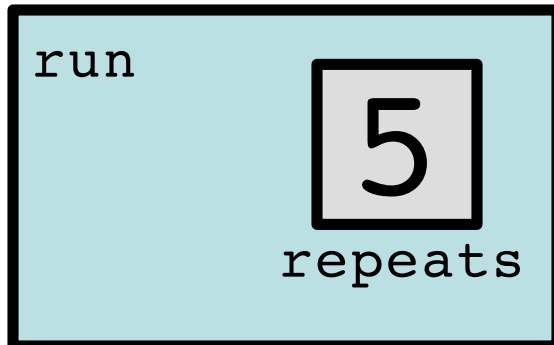
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

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

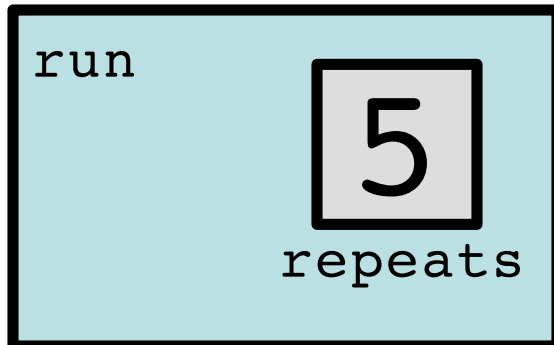
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

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

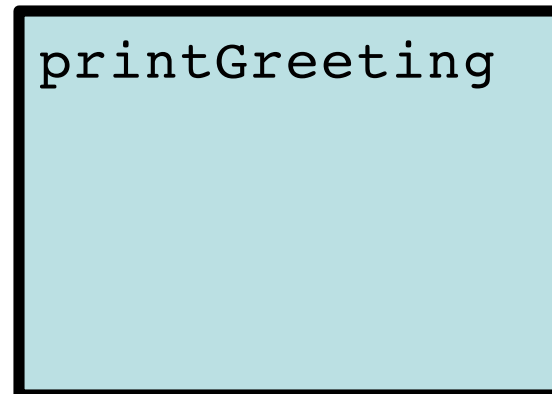
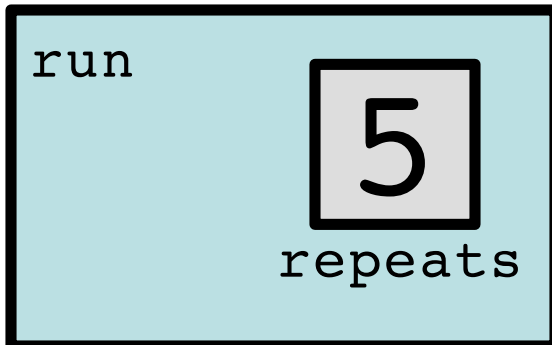
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

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

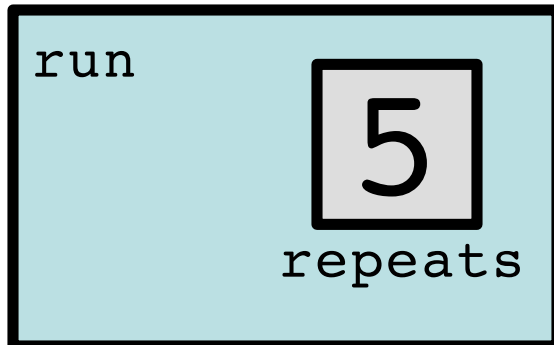
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

```
public void run() {  
    int 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(times);  
}  
  
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(times);  
}
```

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

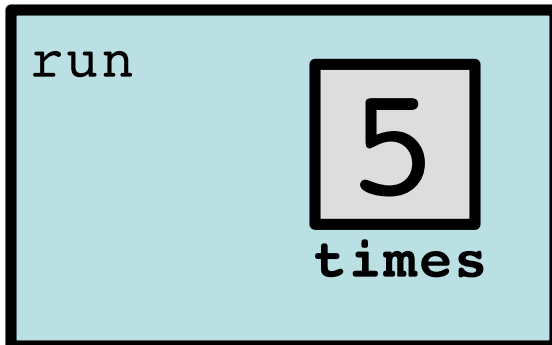


run

Methods with Parameters

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

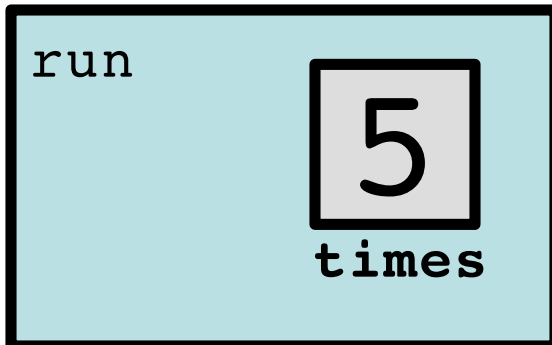
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

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

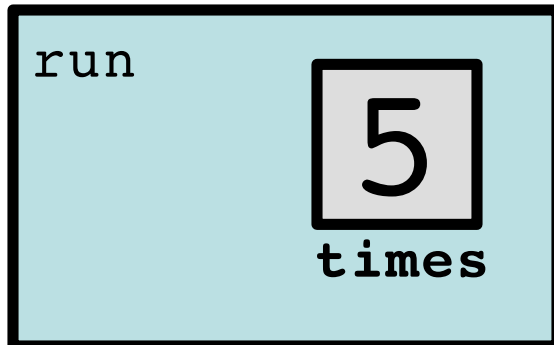
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

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

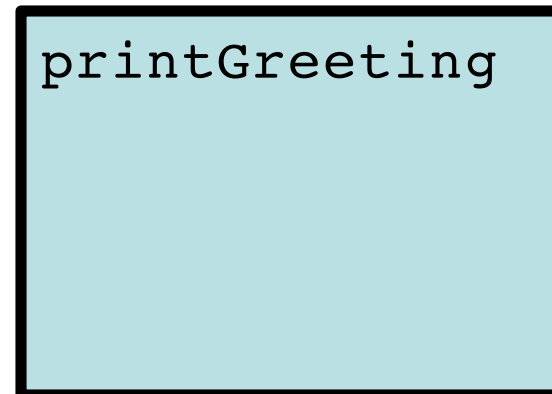
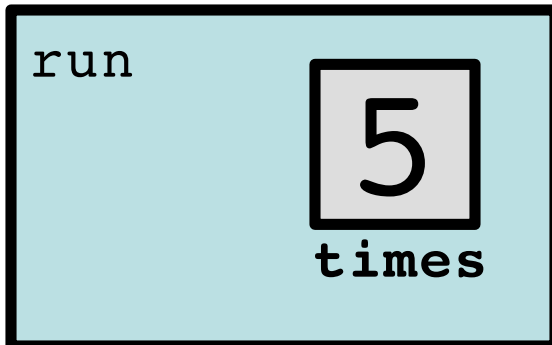
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

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

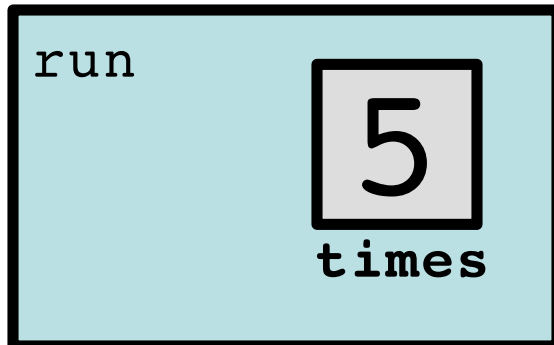
```
private void printGreeting(int times) {  
    for (int i = 0; i < times; i++) {  
        println("Hello world!");  
    }  
}
```



Methods with Parameters

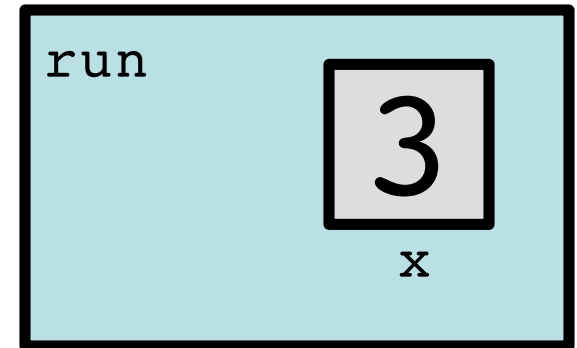
```
public void run() {  
    int times = 5;  
    printGreeting(times);  
}
```

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



Parameters are Copies

```
// NOTE: This program is buggy!!  
public void run() {  
    int x = 3;  
    addFive(x);  
    // prints "x = 3!"  
    println("x = " + x);  
}
```

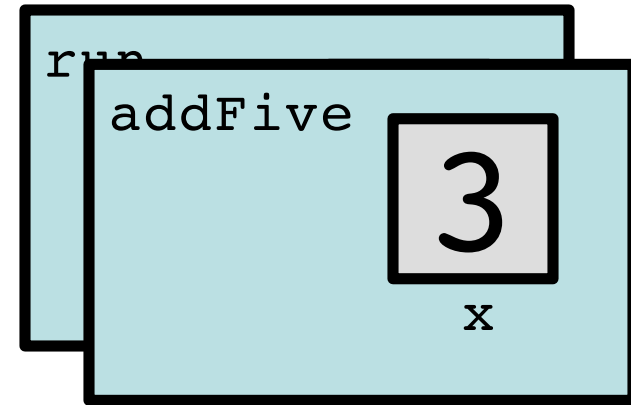


```
private void addFive(int x) {  
    x += 5;  
}
```

Parameters are Copies

```
// NOTE: This program is buggy!!  
public void run() {  
    int x = 3;  
    addFive(x);  
    // prints "x = 3!"  
    println("x = " + x);  
}
```

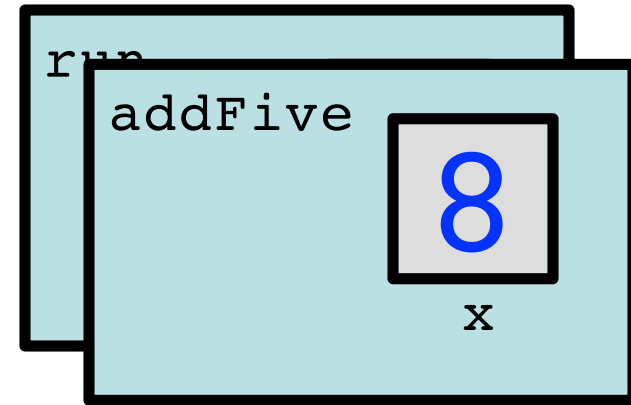
```
private void addFive(int x) {  
    x += 5;  
}
```



Parameters are Copies

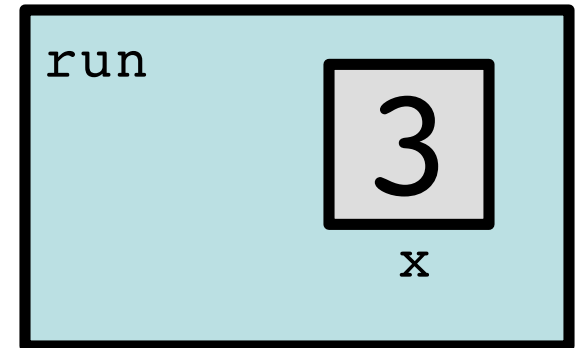
```
// NOTE: This program is buggy!!  
public void run() {  
    int x = 3;  
    addFive(x);  
    // prints "x = 3!"  
    println("x = " + x);  
}
```

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



Parameters are Copies

```
// NOTE: This program is buggy!!  
public void run() {  
    int x = 3;  
    addFive(x);  
    // prints "x = 3!"  
    println("x = " + x);  
}
```



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

Multiple parameters

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

- Declaration:

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

- Call:

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

Drawing boxes

- Let's write a program that uses methods and parameters to print the following boxes:

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

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

- The code to draw each box will be very similar.
 - Would variables help? Would constants help?

drawBox

```
drawBox(10, 4);
```



drawBox

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


drawBox

We call
drawBox

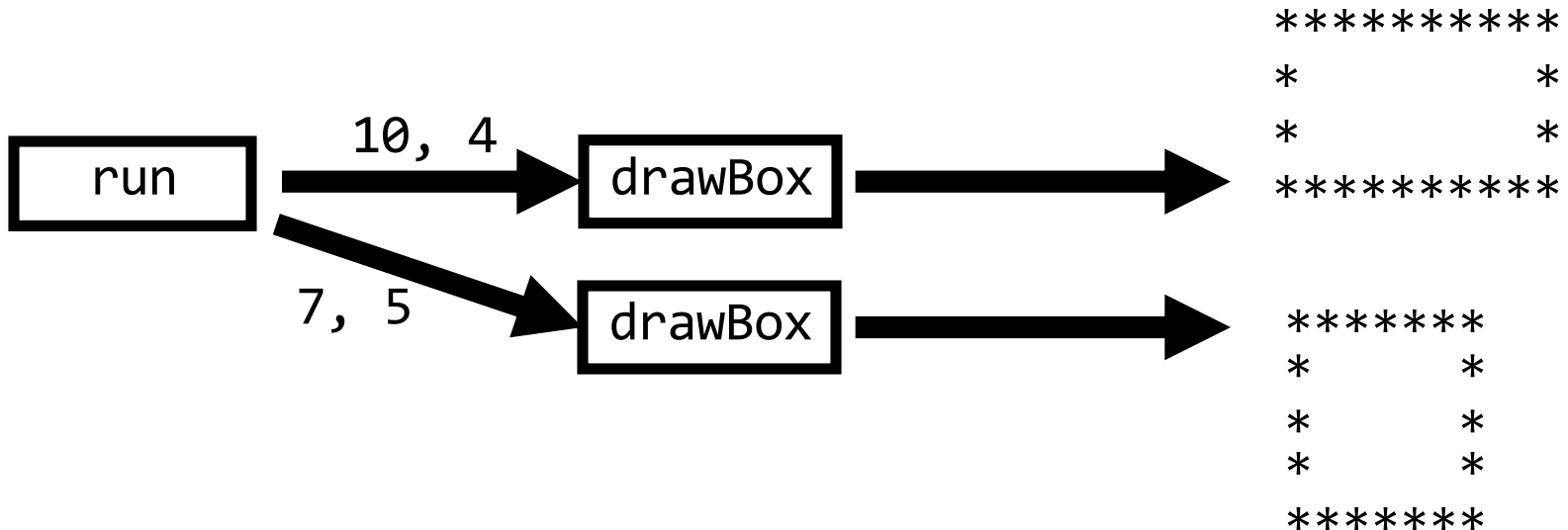
We give drawBox some
information (the size of
the box we want)



```
drawBox ( 10 , 4 ) ;
```

Parameters

- **parameter:** A value passed to a method by its caller.
 - Write a method **drawBox** 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.



drawBox

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

drawBox

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

```
private void drawBox(int width, int height) {  
    drawLine(width);  
    for (int line = 0; line < height - 2; line++) {  
        boxSide(width);  
    }  
    drawLine(width);  
}
```

drawBox

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

```
private void drawBox(int width, int height) {  
    drawLine(width);  
    for (int line = 0; line < height - 2; line++) {  
        boxSide(width);  
    }  
    drawLine(width);  
}
```


drawBox

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


```
private void drawBox(int width, int height) {  
    drawLine(width);  
    for (int line = 0; line < height - 2; line++) {  
        boxSide(width);  
    }  
    drawLine(width);  
}
```

drawBox

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



```
private void drawBox(int width, int height) {  
    drawLine(width);  
    for (int line = 0; line < height - 2; line++) {  
        boxSide(width);  
    }  
    drawLine(width);  
}
```



drawBox

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

```
private void drawBox(int width, int height) {  
    drawLine(width);  
    for (int line = 0; line < height - 2; line++) {  
        boxSide(width);  
    }  
    drawLine(width);  
}
```


line

```
private void drawLine(int width) {  
    for (int i = 0; i < width; i++) {  
        print("*");  
    }  
    println();  
}
```

boxSide

* *

```
private void boxSide(int width) {  
    print("*");  
    for (int i = 0; i < width - 2; i++) {  
        print(" ");  
    }  
    println("*");  
}
```

boxSide

```
public void run() {  
    drawBox(10, 4);  
    drawBox(7, 6);  
}
```

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

Plan For Today

- Announcements
- Recap: Scope
- Parameters
- **Return**

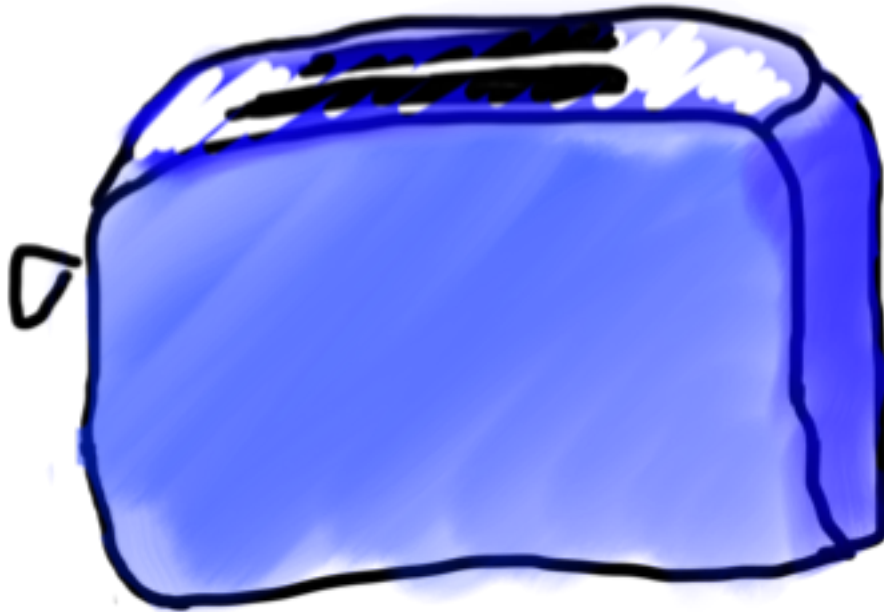
Return

Return values let you give back some information when a method is finished.

Methods = Toasters



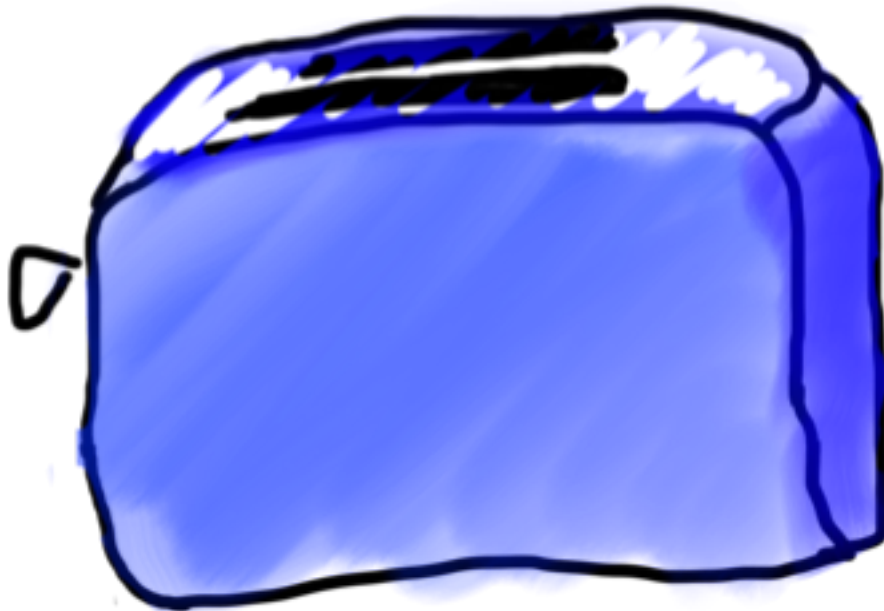
parameter



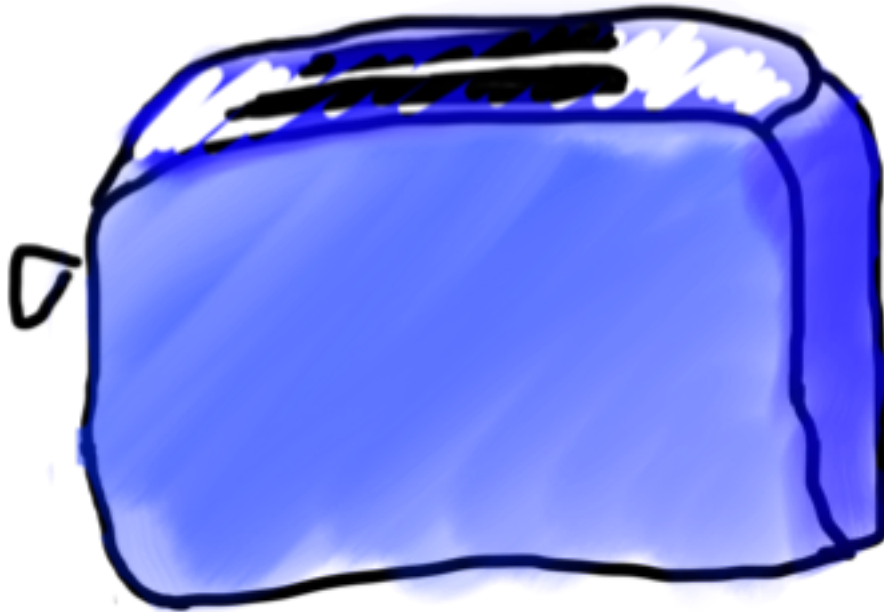
Methods = Toasters



parameter



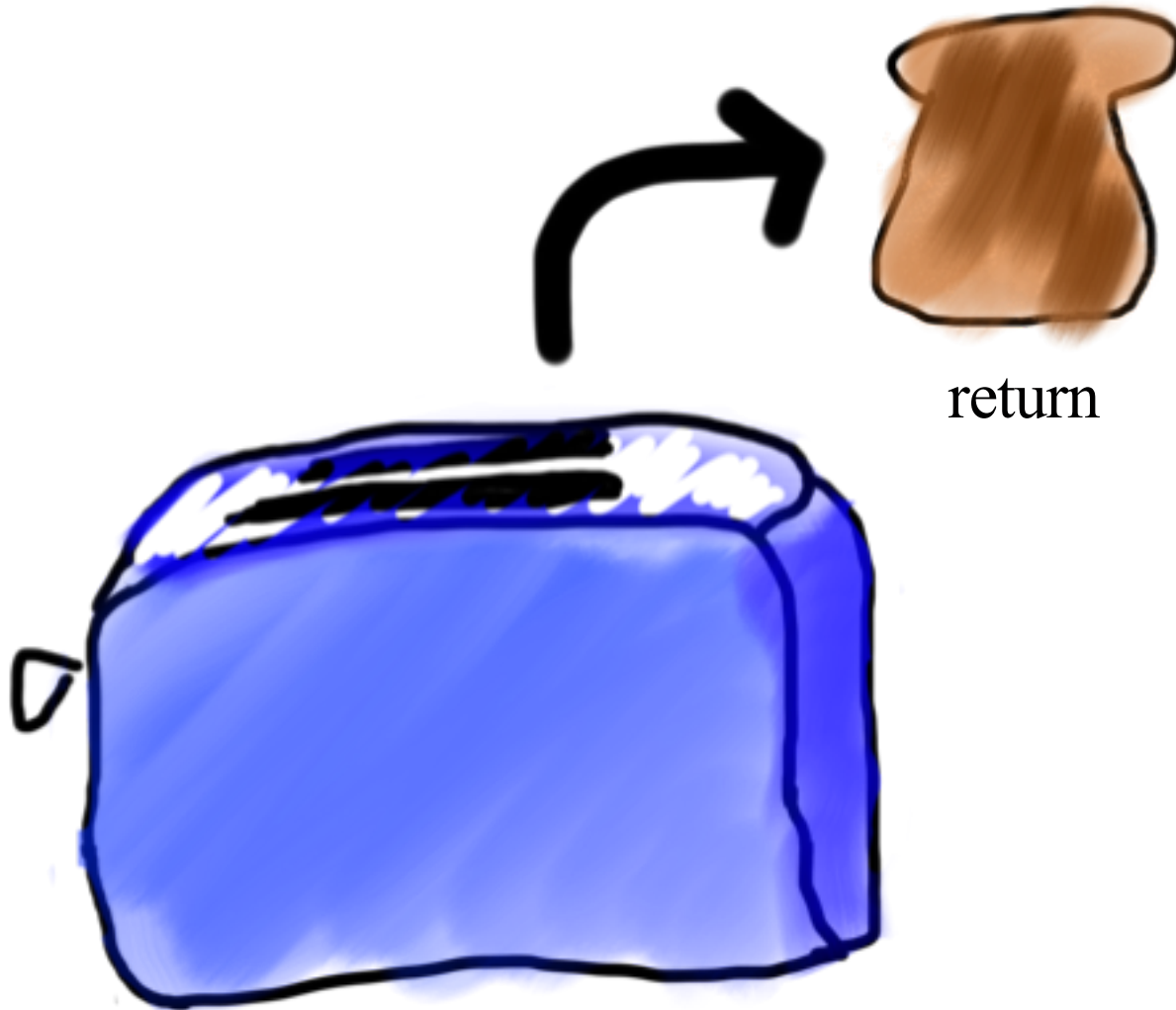
Methods = Toasters



Methods = Toasters



Methods = Toasters



Example: readInt

```
int x = readInt( "Your guess? " );
```

Example: readInt

We call
readInt



We give readInt some
information (the text to
print to the user)



```
int x = readInt( "Your guess? " );
```

Example: readInt

When finished, readInt gives us information back (the user's number) and we put it in x.



```
int x = readInt( "Your guess? " );
```

Example: readInt

When we set a variable equal to a method, this tells Java to save the return value of the method in that variable.

```
int x = readInt( "Your guess? " );
```

Example: metersToCm

```
double cm = metersToCm(5);
```

(Returns the given number of m as cm)

Example: metersToCm

We call
metersToCm

We give
metersToCm some
information (the
number of meters)

double cm = metersToCm(5);

A diagram illustrating the call to the `metersToCm` function. A blue arrow points from the text "We call metersToCm" to the function name `metersToCm` in the code. Another blue arrow points from the text "We give metersToCm some information (the number of meters)" to the argument `5` in the code.

Example: metersToCm

When metersToCm finishes, it returns the number of cm, and we put that in this variable.

```
double cm = metersToCm(5);
```



Methods and Return

Tells Java this method
needs one *double* in order
to execute.



```
private double metersToCm(double meters) {  
    ...  
}
```

Methods and Return

Tells Java that, when this method finishes, it will return a *double*.
(Void meant returns nothing)



```
private double metersToCm(double meters) {  
    ...  
}
```

Methods and Return

Tells Java that, when this method finishes, it will return a *double*.



```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```

Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```

Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```

run



meters

Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```

run

5

meters

Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```

run

5

meters

metersToCm

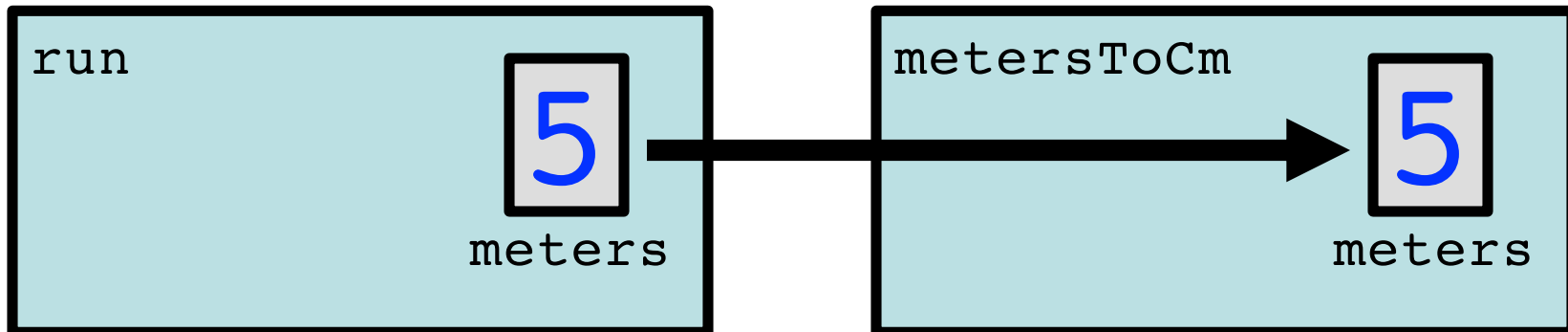
?

meters

Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

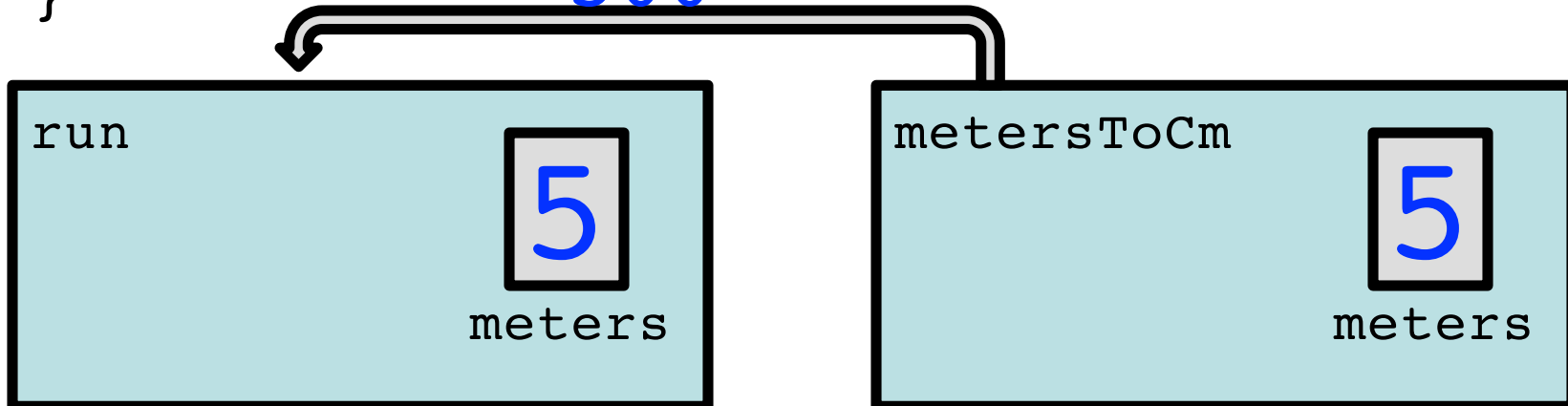
```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```



Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

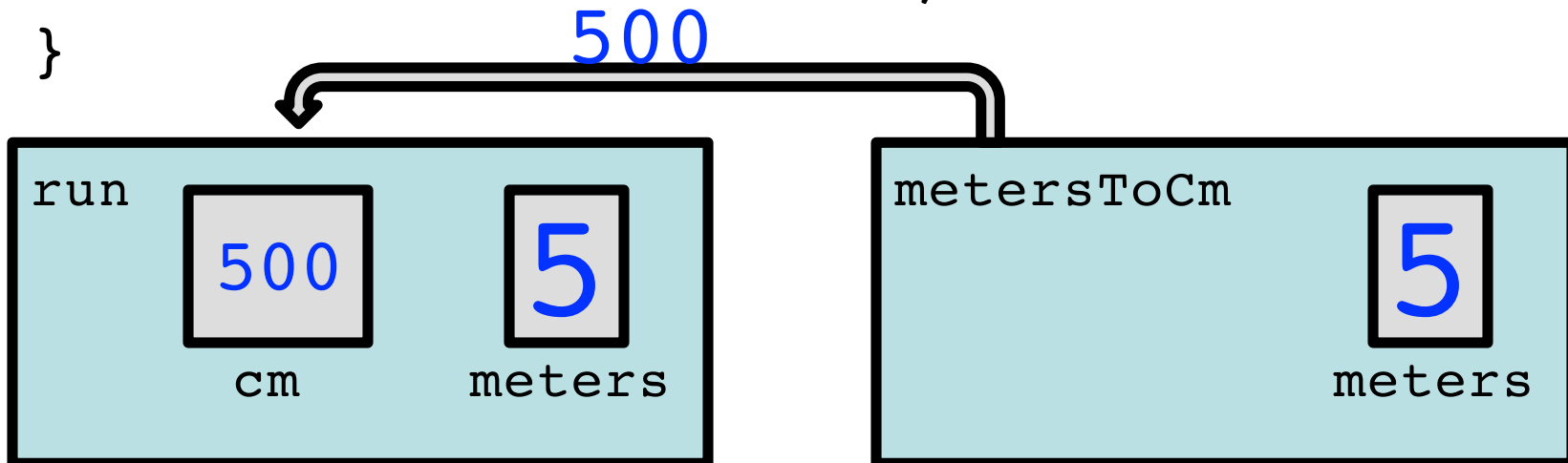
```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```



Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

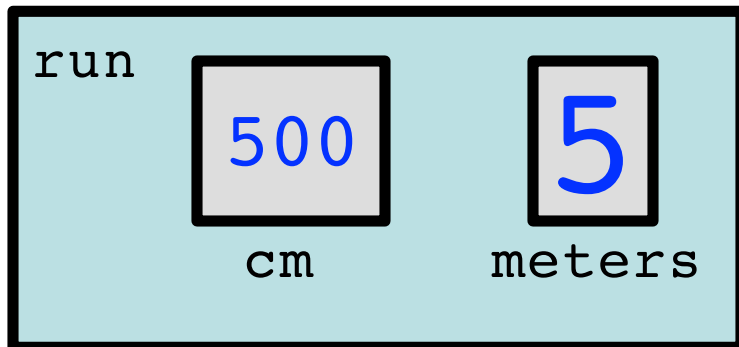
```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```



Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}
```

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```



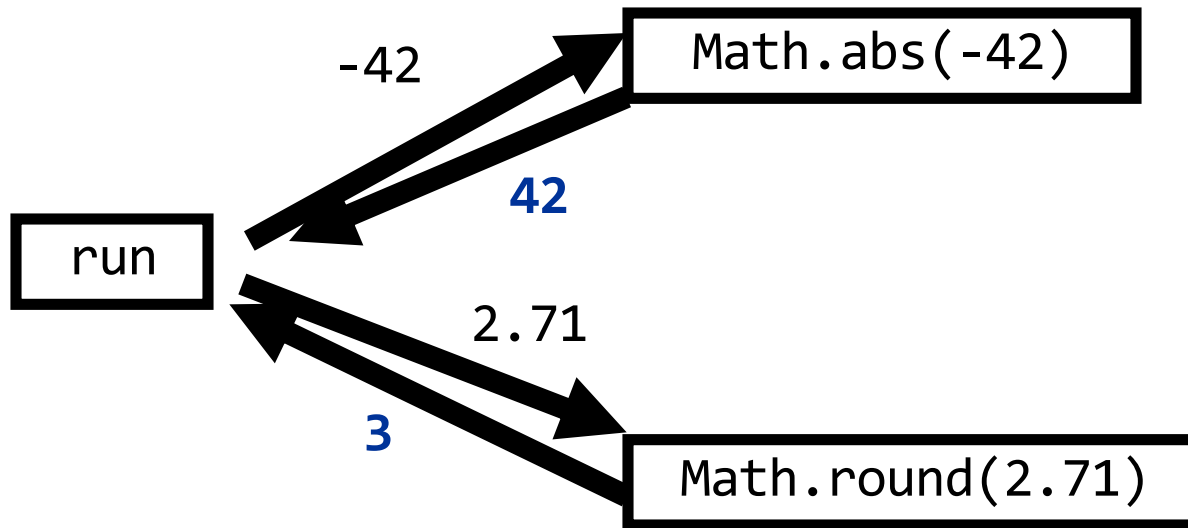
Methods and Return

```
public void run() {  
    double meters = readDouble("#meters? ");  
    println(metersToCm(meters) + "cm.");  
}  
  
private double metersToCm(double meters) {  
    return 100 * meters;  
}
```

If a method returns something, you can use it directly in an expression!

Parameters vs. Return

- **return:** To send out a value as the result of a method.
 - Parameters send information *in* from the caller to the method.
 - Return values send information *out* from a method to its caller.
 - A call to the method can be used as part of an expression.



- **Q:** Why return? Why not just println the result value?

Methods

```
visibility type nameOfMethod(parameters) {  
    statements  
}
```

- *visibility*: usually **private** or **public**
- *type*: type returned by method (e.g., **int**, **double**, *etc.*)
 - Can be **void** to indicate that nothing is returned
- *parameters*: information passed into method

Returning Booleans

```
private boolean isEven(int number) {  
  
}
```


Returning Booleans

```
private boolean isEven(int number) {  
    if (number % 2 == 0) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Returning Booleans

```
private boolean isEven(int number) {  
    if (number % 2 == 0) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

```
public void run() {  
    int num = readInt("? ");  
    if (isEven(num)) {  
        println("Even!");  
    } else {  
        println("Odd!");  
    }  
}
```

Returning Booleans

```
private boolean isEven(int number) {  
    return number % 2 == 0;  
}
```

```
public void run() {  
    int num = readInt("? ");  
    if (isEven(num)) {  
        println("Even!");  
    } else {  
        println("Odd!");  
    }  
}
```

Return

Return *ends* a method's execution.

```
private int multiplyByTwo(int num) {  
    return num * 2;  
    println("Hello world?"); // not executed!  
}
```

Return

Return *ends* a method's execution.

```
private int max(int num1, int num2) {  
    if (num1 >= num2) {  
        return num1;  
    }  
    return num2; // here only if num1 < num2  
}
```

```
public void run() {  
    println(max(2,3));  
}
```

Revisiting a Bug

```
// NOTE: This program is buggy!!  
public void run() {  
    int x = 3;  
    addFive(x);  
    // prints "x = 3"  
    println("x = " + x);  
}  
  
private void addFive(int x) {  
    x += 5;  
}
```

Fixed!

```
// NOTE: This program is feeling just fine
public void run() {
    int x = 3;
    x = addFive(x);
    // prints "x = 5"!
    println("x = " + x);
}

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

Recap

- Announcements
- Recap: Scope
- Parameters
- Return

Next time: Strings (new variable type!)