

Lecture Jan 22 MW - Methods and Ifs

Bentley James Oakes

January 21, 2018

- Due **January 31st**
- Should cover all the material by the end of today
- Please ask for help if you need it
- Discussion forum, TAs, my office hours



Office Hours and Tutorials

Giulia Alberini Jan 12, 2018 10:45

Here is a Google Calendar with all the Office Hours and tutorials offered for COMP 202 this semester.

COMP 202 Winter 2018 - Office Hours & Tutorials

Today	◀	▶	January 2018	▼	Print	Week	Month	Agenda	▼
Sun	Mon	Tue	Wed	Thu	Fri	Sat			
31	1 Jan	2	3	4	5	6			
7	8	9	10	11	12	13			
14	15	16	17	18	19	20			
	13:00 Office Hours (J)	10:00 Office Hours (P)	08:30 Office Hours (S)	09:00 Office Hours (T)	08:30 Office Hours (Z)				
	13:30 Office Hours (E)	11:00 Tutorial	13:00 Office Hours (J)	10:00 Office Hours (J)	09:30 Office Hours (T)				
	14:00 Office Hours (M)	12:30 Office Hours (X)	13:30 Office Hours (E)	11:00 Office Hours (A)	11:30 Tutorial				
	15:00 Tutorial	13:30 Office Hours (F)	15:00 Office Hours (S)	12:00 Tutorial	13:00 Office Hours (A)				
	16:30 Office Hours (I)	16:00 Office Hours (C)	16:00 Office Hours (F)	+2 more	+2 more				

- See the Assistance forum on MyCourses for a calendar of office hours and tutorials

- 1 Input Arguments
- 2 Methods
- 3 Methods Overview
- 4 Methods Example
- 5 If Statements
- 6 Writing Proper Code
- 7 Scope and Other Details

Section 1

Input Arguments

- Let's look at the main method header

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

- This is a method that does not return anything
- And has one parameter called *args* of type `String[]`

- String[] args are parameters passed to the program
- This is specially done when Java starts the program
- This is why we need to have *String[] args* as the main method's parameter

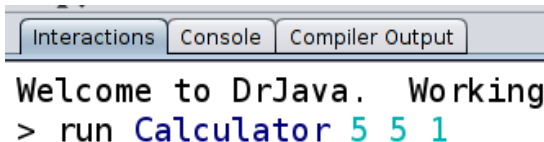
```
public static void main(String[] args)
{
    System.out.println("First argument: " + args[0]);
}
```

Assignment Input Arguments

- In the assignment, we input data using these input arguments

To use:

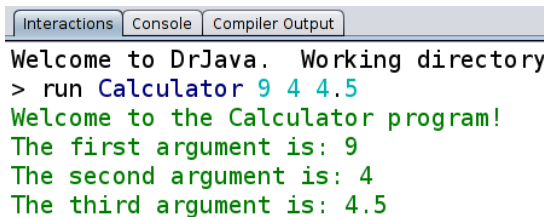
- Open up Calculator.java
- Type *run Calculator 5 5 1* and press Enter



The screenshot shows the DrJava IDE interface. At the top, there are three tabs: 'Interactions', 'Console', and 'Compiler Output'. The 'Console' tab is selected. Below the tabs, the text 'Welcome to DrJava. Working' is displayed. In the console area, the command '> run Calculator 5 5 1' has been entered. The word 'run' is in black, 'Calculator' is in blue, and the numbers '5 5 1' are in red.

Assignment Input Arguments

- Changing these input arguments changes the parameters given to main

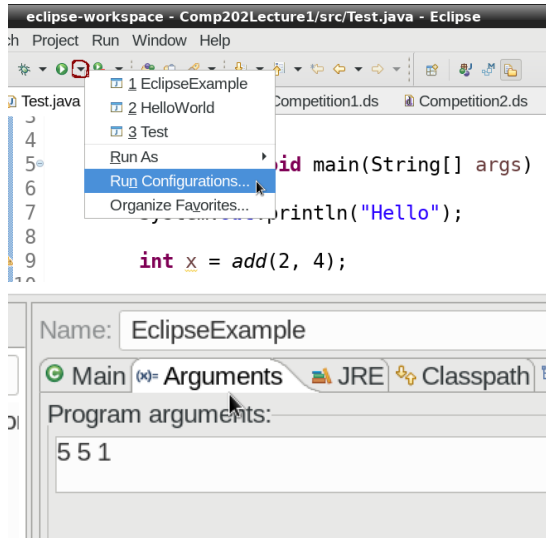


```
Interactions Console Compiler Output
Welcome to DrJava. Working directory
> run Calculator 9 4 4.5
Welcome to the Calculator program!
The first argument is: 9
The second argument is: 4
The third argument is: 4.5
```

- Note that the assignment uses methods to turn these String arguments into ints and doubles

Input Arguments in Eclipse

- You access the *Run Configuration* for your program in Eclipse
- Then place the arguments under the *Parameters* tab



- Let's look at the methods to turn Strings into numbers
- `int x = Integer.parseInt("123");`
- `double y = Double.parseDouble("56.4");`
- `boolean z = Boolean.parseBoolean("true");`
- An error is produced if the String doesn't match the type
- `Integer.parseInt("3.5")` doesn't work
- Neither does `Double.parseDouble("Hello")`

Parsing Strings

```
1 public class ParseArgs
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("The first argument is: " + args[0]);
6         System.out.println("The second argument is: " + args[1]);
7         |
8         int x = Integer.parseInt(args[0]);
9         System.out.println("The first argument is now an int: " + x);
10
11         double y = Double.parseDouble(args[1]);
12         System.out.println("The second argument is now a double: " + y);
13
14         boolean z = Boolean.parseBoolean(args[2]);
15         System.out.println("The third argument is now a boolean: " + z);
16     }
17 }
```

Section 2

Methods

- Methods are pieces of **reusable code**
- They take **parameters** as input
- They **return** a value as output
- They can contain any number of instructions within
- Therefore, a method is just an algorithm

Dividing Up Our Program

To make our program easier to understand and test, we break up the larger program into smaller algorithms

“I recommend that a novice trying to making something like the tart think of it not as one elaborate recipe but as a series of simpler preparations - a custard, a pastry, and a glaze. **Broken down into its component parts, any recipe will appear less intimidating and more manageable.**” -
Kitchen Wisdom

Here we are **calling** the sqrt method with the second instruction:

```
//create x
double x = 123;

//get the square root of x
//using the Math.sqrt method
double y = Math.sqrt(x);

//print out the answer
//gives 11.09
System.out.println(y);
```

We then store the **return** value of the method in a variable.

Math.random()

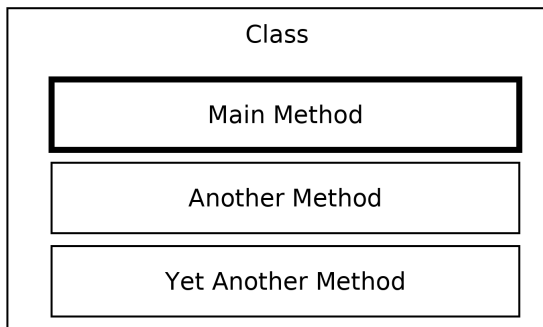
```
1 public class RandNums
2 {
3     public static void main(String[] args)
4     {
5         //get a number between 0 and 1
6         double x = Math.random();
7
8         //print out that number
9         //different every time
10        System.out.println("X is " + x + " this time.");
11        //X is 0.5114639381704913 this time.
12
13        //if you want a number between 0 and 100,
14        //multiply by 100
15        double y = x * 100;
16        System.out.println("Y is: " + y);
17        //Y is: 51.14639381704913
18
19        //note: x will never be 1, so y can't be 100
20    }
21 }
```

Section 3

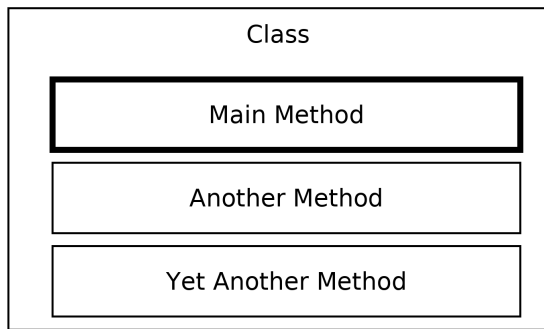
Methods Overview

Structure of a Program

- In our programs, we will have *classes* which contain many *methods*
- Within each of these methods, there are instructions which *may* or *may not* be executed by Java
- Usually there is a *main method* in a class
- The class usually contains many other methods that we write

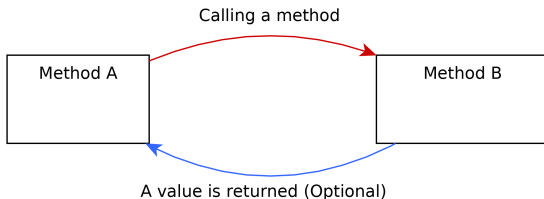


Structure of a Program Notes



Important Points:

- The main method can go above or below the other methods - it doesn't matter
- When the program runs, Java executes instructions starting with the *main method*
- The instructions in other methods are **not executed** unless the method is called



Important Points:

- We can call the method with or without passing parameters
- The method does some calculations
- A value may or may not be returned
 - Note that printing something **does not** count as returning a value
 - The `return` keyword returns a value

Four Parts of a Method's Description

The four parts of a method's description are:

- The **return type**
- The **method name**
- The **parameters** the method accepts
- A brief **description** or purpose
 - To be placed in comments to let others know what this method does

To describe `Math.sqrt()`

- **Return type** - double
- **Method name** - sqrt
- **Parameters** (what the method accepts) - One double value
- **Description/Purpose** - “Returns the square root.”

The four parts turn into the first line of the method:

- Method name - addNumbers
- Input - Two integers
- Output - An integer
- Purpose - To add the two input parameters.

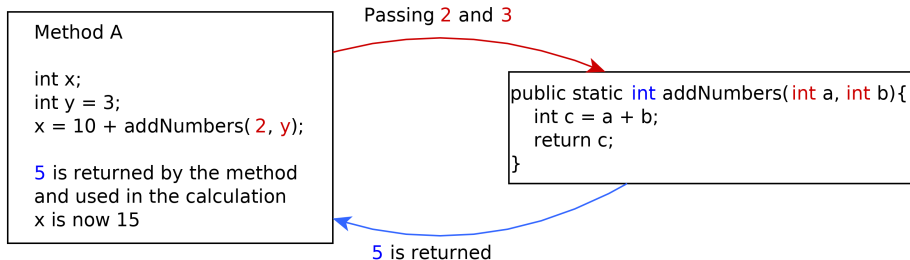
```
public int addNumbers(int a, int b)
```

This first line is called the **method header**
Always always figure out the method header first!

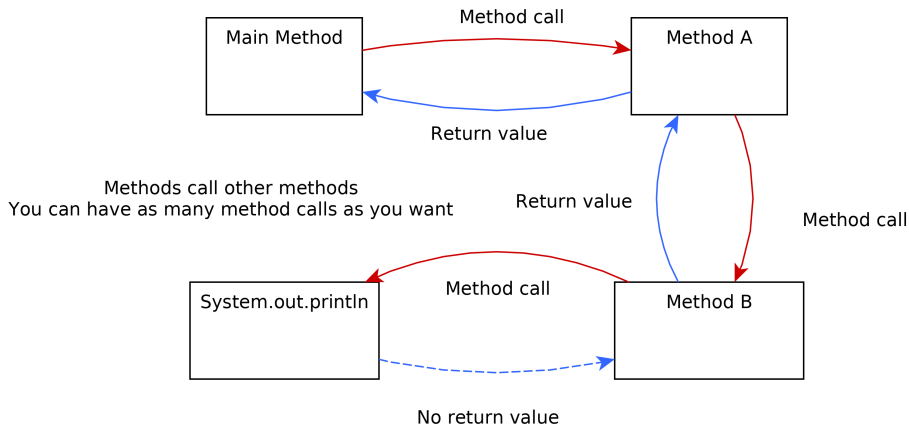

```
public int addNumbers(int a, int b)
{
    int c = a + b;
    return c;
}
```

- The return statement says which variable to output
- We can only output one variable, and the type must match the header

Method Call Example



Chained Method Calls



- Note that `System.out.println` doesn't return a value
 - Return type is **void**
- The method call graph can get complicated
- Methods can call themselves

Calling a Method Repeatedly

- Just like calling other methods, we can call the addNumbers method repeatedly
- This will perform the calculation twice:

```
1 public class ANExample{
2
3     public static void main(String[] args){
4         int x = addNumbers(1, 45);
5         System.out.println(x); //46
6
7         int y = addNumbers(56, 239);
8         System.out.println(y); //295
9     }
10
11     public static int addNumbers(int a, int b){
12         return a + b;
13     }
14 }
```

Execution Location

Java executes instructions in the main method and the addNumbers method

- We say that **control** is switching back and forth
 - The main method is executed first
 - The addNumbers method is called
 - The calculation is performed and a value is returned
 - x is printed
 - The addNumbers method is called
 - The calculation is performed and a value is returned
 - y is printed

■ **Always step-by-step!**

```
1 public class ANExample{
2
3     public static void main(String[] args){
4         int x = addNumbers(1, 45);
5         System.out.println(x); //46
6
7         int y = addNumbers(56, 239);
8         System.out.println(y); //295
9     }
10
11     public static int addNumbers(int a, int b){
12         return a + b;
13     }
14 }
```

Calling with Wrong Types

- What happens if we call the method with the wrong types?

```
int x = addNumbers(56, 34);  
int y = addNumbers("blah", 123);  
}  
  
public static int addNumbers(int a, int b)  
{  
    int c = a + b;  
    return c;  
}
```

- The parameters passed has to match the types in the method header
- The method header says it accepts two `int` values
- Java will complain if we try to pass a `String`

Section 4

Methods Example

- We're just about at the point where we can create a program for ordering pizzas

This program will have three parts:

- The `pricePerPizza` method will tell us the price per pizza
- The main method will have the number of pizzas to order and will figure out the total cost
- The `printTotal` method will print the number of pizzas and the total price

- The pricePerPizza method will tell us the price per pizza
- Right now, that will just be \$14.95
- So it accepts nothing and returns a double

```
public static double pricePerPizza()  
{  
    return 14.95;  
}
```

- The printTotal method will take the number of pizzas and the total cost
- So it accepts the number of pizzas as an int, and the cost as a double
- This method returns nothing, so it is void

```
public static void printTotal(int number, double totalCost){  
    System.out.println("Your total for " + number  
                        + " pizzas is: $" + totalCost);  
}
```

- The main method will get the price per pizza and will perform the total cost calculation

```
public static void main(String[] args){  
    int numPizzasOrdered = 100;  
    double ppp = pricePerPizza();  
  
    //calculate the total cost  
    double totalCost = numPizzasOrdered * ppp;  
  
    printTotal(numPizzasOrdered, totalCost);  
}
```

```
1 public class PizzaPrice
2 {
3     public static void main(String[] args){
4         int numPizzasOrdered = 100;
5         double ppp = pricePerPizza();
6
7         //calculate the total cost
8         double totalCost = numPizzasOrdered * ppp;
9
10        printTotal(numPizzasOrdered, totalCost);
11    }
12
13    public static double pricePerPizza(){
14        return 14.95;
15    }
16
17    public static void printTotal(int number, double totalCost){
18        System.out.println("Your total for " + number
19                            + " pizzas is: $" + totalCost);
20    }
21 }
```

Section 5

If Statements

- Up until now, our programs haven't been able to make decisions
- Based on the value of variables, we might want to do different things
- For example, if the temperature is below 0, the program says it's freezing
- Or if someone orders more than 10 pizzas, they get a discount
- We decide which instructions to run with an **if statement**

Pizza Discount

- Let's change our pricePerPizza method to give a \$5 discount if ten or more pizzas are ordered
- Note that we have to change the method header

```
public static double pricePerPizza(int numPizzas){  
  
    //start the price off at 14.95  
    double price = 14.95;  
  
    //if ten or more pizzas are being ordered  
    if (numPizzas >= 10){  
        //subtract $5 from the price  
        price = price - 5;  
    }  
  
    //return the price per pizza  
    return price;  
}
```

Pizza Discount

- We have added an **if statement** to the method
- If what's inside the brackets (the **if condition** evaluates to true)
- Then the code inside the **if block** will be executed

```
public static double pricePerPizza(int numPizzas){  
  
    //start the price off at 14.95  
    double price = 14.95;  
  
    //if ten or more pizzas are being ordered  
    if (numPizzas >= 10){  
        //subtract $5 from the price  
        price = price - 5;  
    }  
  
    //return the price per pizza  
    return price;  
}
```



```
public static double pricePerPizza(int numPizzas){  
    //start the price off at 14.95  
    double price = 14.95;  
  
    //if ten or more pizzas are being ordered  
    if (numPizzas >= 10){  
        //subtract $5 from the price  
        price = price - 5;  
    }  
  
    //return the price per pizza  
    return price;  
}
```

- The instruction(s) within the if block only executes if the if condition is true
- So the program can take two different paths
 - *price* could be 14.95 or 9.95 depending on the test

Section 6

Writing Proper Code

- The goal of programming is to write correct code
- But it's also extremely important to write readable code
- This section will talk about the difference between good code and bad code
- TAs can take marks off for code that does not meet these guidelines
 - Don't worry, we'll be very forgiving for the first assignment

The golden rule is that your code should be obviously correct and understandable

Multi-step Calculations

Breaking up your calculations and using lots of variables can keep your code readable

```
int value = 5;

//get the remainder mod 2
int remainder = value % 2;

//check if this remainder is 1
//if so, then the number was odd
boolean isOdd = (remainder == 1);

//print out whether the value is odd or not
System.out.println(value + " is odd: " + isOdd);
```

versus

```
System.out.println(value + " is odd: " + (value % 2 == 1));
```

```
int value = 5;

//get the remainder mod 2
int remainder = value % 2;

//check if this remainder is 1
//if so, then the number was odd
boolean isOdd = (remainder == 1);

//print out whether the value is odd or not
System.out.println(value + " is odd: " + isOdd);
```

- Comments are absolutely critical to understanding what your program is trying to do
- You should have a line of comments for every 1-2 lines of code
- Good guideline for if a line of code needs a comment:
 - Is it obvious what this line of code does?
 - The first line is straightforward and we can't write a useful comment here

Java has a few rules for naming your variables

- Can't start with numbers
- Can only contain letters and numbers
- No keywords (public, static, void, etc.)

Variable Names

And there are some best practices for names:

- Should be short

- Not okay:

- `thisIsAVariableThatStoresAnIntAndWillStoreARemainder`

- Shouldn't be **too** short

- Hard to read code that has a ton of one-letter variable names

- Capitalization should be camelCase

- The first word is lowercase, all others are uppercase

- Example: `isOdd`, `nameOfSchool`

- Should be descriptive and unique

- The better your variables are named, the fewer comments you need

- No random words for variables, such as `kittens`

Bad Naming Example

One student back in my TA days had code that looked like this:

```
int kittens = 145;
int kitties = kittens * 3;
int cats = kitties % 2;
String kats = " is odd: ";
boolean kittyKats = (cats == 1);
System.out.println((kittens + kittens + kittens) + kats + kittyKats);
```

...The TAs will take marks off for this...

Indentation

- To keep code organized, programmers **indent** their code
- Between the braces, we tab or space the code in

```
1 public class SayHelloClass
2 {
3     public static void main(String[] args)
4     {
5         sayHello();
6     }
7
8     //This method prints hello world.
9     public static void sayHello()
10    {
11        System.out.println("Hello World");
12    }
13 }
```

- Note how the code within the class and the main method is moved over a bit each time
- Dr. Java can help you indent your code - 'Edit → Indent' command

There are two ways to put your starting braces in your code

On the same line:

```
public static int addNumbers(int a, int b){  
    int c = a + b;  
    return c;  
}
```

On the next line:

```
public static int addNumbers(int a, int b)  
{  
    int c = a + b;  
    return c;  
}
```

Doesn't matter - just be consistent!

Section 7

Scope and Other Details

Random Number in a Range

- Let's create a helpful method for creating random numbers
- The method will create random numbers between a *min* value and a *max* value

```
public static double getRandNum(int min, int max){  
    //how big is the range between the numbers?  
    int range = max - min;  
  
    //get a random value in this range  
    double rand = Math.random() * range;  
  
    //add it to the min number  
    double finalNum = rand + min;  
  
    //return the random number  
    return finalNum;  
}
```

Rand Range Explanation

```
public static double getRandNum(int min, int max){  
    //how big is the range between the numbers?  
    int range = max - min;  
  
    //get a random value in this range  
    double rand = Math.random() * range;  
  
    //add it to the min number  
    double finalNum = rand + min;  
  
    //return the random number  
    return finalNum;  
}
```

- Example: *min* is 75 and *max* is 100
- We want a number between 75 and 100
- The range is 25, so we get a random number from 0 to 25 (not including 25)
- Then we add that random number to 75 to get the final result

We can look at four different kinds of methods:

- No input and no return value
- Input but no return value
- No input, but a return value
- Input and a return value

I'll show example methods and how to call each one

No Input/No Return Value

- No input arguments, and no return value
- May be useful for printing a welcome message
- No return value means this method is void

```
//notice that the return type is void
//and the method accepts no parameters
public static void sayHello()
{
    int a = 4 * 6;
    System.out.println("Top of the morning!");
    System.out.println("This is my favourite number: " + a);
}

public static void main(String[] args)
{
    //calling the method
    sayHello();
}
```

Input/No Return Value

- Has input parameters, but no return value
- Could be used to print out a total
- No return value means this method is void

```
//notice that the return type is void
//and the method accepts a double parameter
public static void printTotal(double total)
{
    System.out.println("Your bill is for : $" + total);
}

public static void main(String[] args)
{
    //calling the method with a double variable
    double total = 198.13;
    printTotal(total);

    //could also do
    printTotal(3847.23);
}
```


No Input/Return Value

- Has no input parameters, but has a return value
- Less commonly seen
- Make sure to put in the return type

```
//notice that the return type is double
//but the method accepts no parameters
public static double getRandomGrade()
{
    //Math.random returns a random double between 0 and 1
    double rand = Math.random();

    //the return value will be between 0 and 100
    return rand * 100;
}

public static void main(String[] args)
{
    //calling the method
    double grade = getRandomGrade();
    System.out.println("Your grade is now: " + grade);
}
```

Inputs/Return Value

- Has input parameters and a return value
- This is the most common kind of method

```
//notice the int return type and
//the two int parameters
public static int multiplyNumbers(int a, int b)
{
    int result = a * b;
    return result;
}

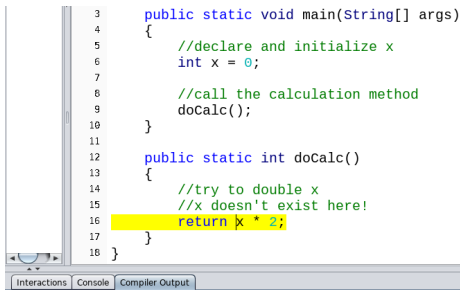
public static void main(String[] args)
{
    int x = 3;
    //calling the method
    //passing the parameters
    //storing the result
    int y = multiplyNumbers(x, 45);
    System.out.println("Y is: " + y);
}
```

```
public int addNumbers(int a, int b)
```

Why is it `int a` and `int b`?

- Need to give the parameters names so you can refer to them in the method's instructions.
- These variable names have **no connection** to variables in other methods
- `a` and `b` are assigned values when the method is called
- The `a` and `b` variables exist only for that method

- Variables only exist in the method they are declared in
 - We call this the variable's **scope**



```
3 public static void main(String[] args)
4 {
5     //declare and initialize x
6     int x = 0;
7
8     //call the calculation method
9     doCalc();
10 }
11
12 public static int doCalc()
13 {
14     //try to double x
15     //x doesn't exist here!
16     return x * 2;
17 }
18 }
```

The screenshot shows a code editor with a Java program. The program has two methods: `main` and `doCalc`. In `main`, a variable `x` is declared and initialized to 0. In `doCalc`, the code attempts to use `x` in a calculation, but this results in a compiler error because `x` is not in scope within `doCalc`. The error message at the bottom states: "1 error found: File: /home/dcx/Dropbox/COMP 202/Lecture 4 - More Meth Error: cannot find symbol symbol: variable x".

- We can't access the `x` variable from another method

The only way for values to travel from one method to another is:

- By passing the value as a parameter
- By returning a value
- There's a third way, but it'll come later in the course

This means that there can be two variables named the same thing in two different methods

They will be totally separate and will not share a value

```
public static void main(String[] args){  
    int x = 10;  
    //prints out 10  
    System.out.println("Main x is: " + x);  
}  
  
public static void otherMethod(){  
    int x = 36;  
    //prints out 36  
    //the int x's are not connected in any way!  
    System.out.println("otherMethod x is: " + x);  
}
```

Remember:

Variables declared in one method cannot be accessed from another method!

Values must be passed as parameters or returned

- Note that variables are not passed
- It's a value that is passed
- This will be important later