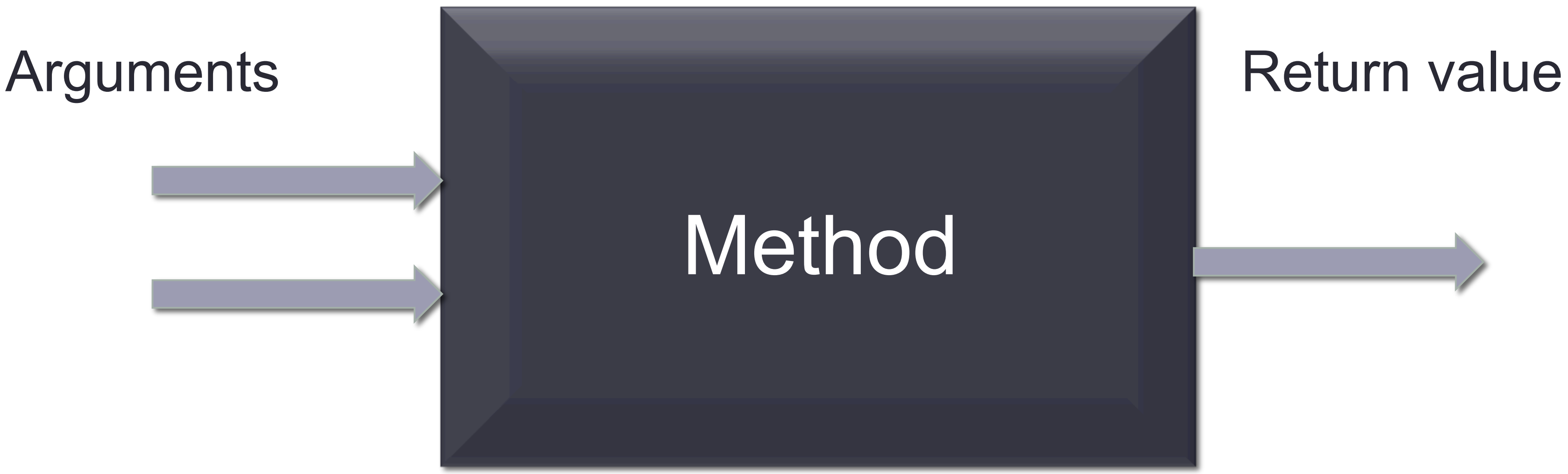
METHODS

CS302 – Introduction to Programming
University of Wisconsin – Madison
Lecture 15

By Matthew Bernstein – matthewb@cs.wisc.edu

Introducing Methods as Black Boxes

- A **method** is a section of code that carries out a particular task (example: add two numbers, sort a list, etc.)
- We can think of a method as a black box, that takes in some arguments, and returns a result



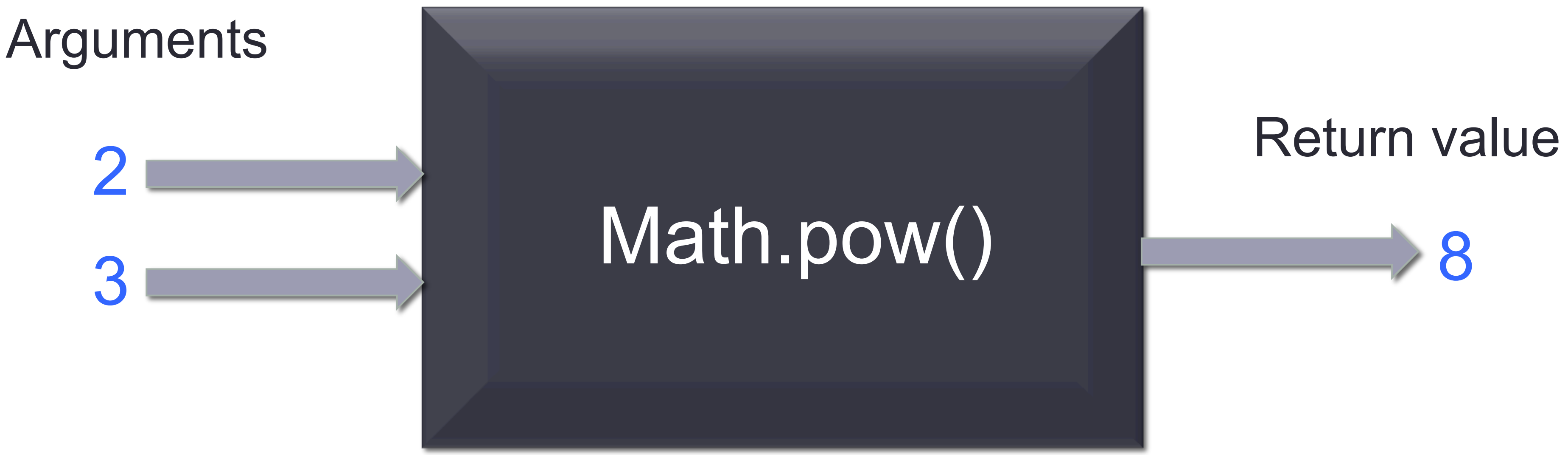
Example of a Method

- Remember the Math.pow() method. It accepts two numbers as its arguments:
  - The first number is a value we want to raise to a certain power
  - The second number is the power we want to raise our value to
- Example:

```
Math.pow(2, 3); // Computes 2^3 which equals 8
```

Math.pow() as a Black Box

```
Math.pow(2, 3);
```



Defining Methods (i.e. Defining the Black Box)

- As a programmer using some method, you don’t need to know what’s going on inside the black box. All you need to know is three pieces of information:
  - **The method’s name**
  - **The method’s parameters** (i.e. what arguments it accepts)
  - **The type of data that the method returns**
- So when we build our own methods, we have to define these pieces of information for our method, and then implement the black box

Defining Methods

```
public static int addTwoInts(int valA, int valB)
{
        int sum = valA+ valB;
        return sum;
}
```

Let's Break it Down

don't worry about what "public" and "static" mean. Just know that, for now, all of our methods will be preceded by these two reserved words.

The data type being returned by the method

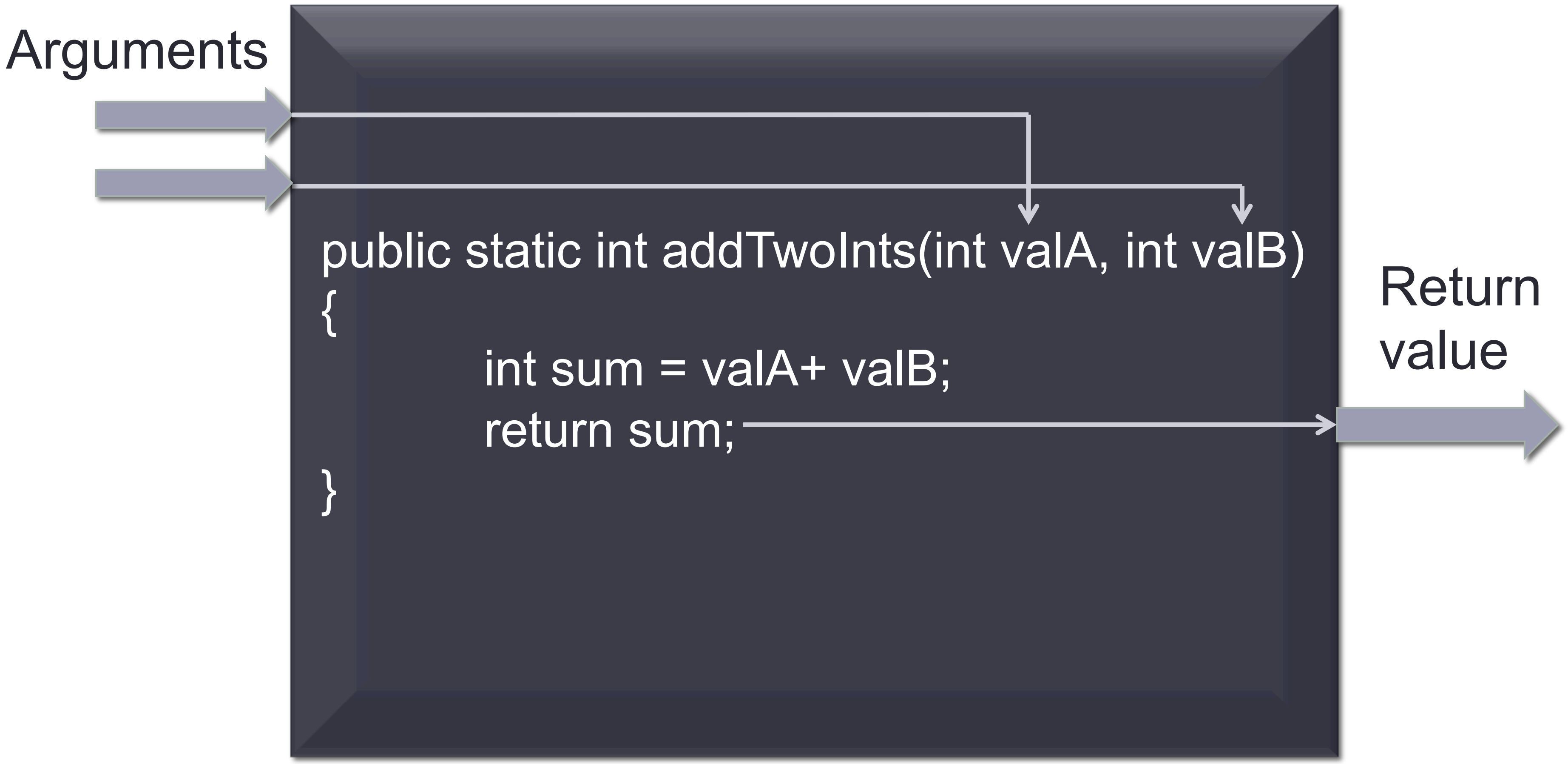
The method's name is "addTwoInts"

The two parameters to this method are two int values which are set to the variables "valA" and "valB"

```
public static int addTwoInts(int valA, int valB)
{
        int sum = valA+ valB;
        return sum;
}
```

Here, we return execution to the code that called the method. The caller also gets the value in the "sum" variable.

The Black Box Again



```
public static int addTwoInts(int valA, int valB)
{
        int sum = valA+ valB;
        return sum;
}
```

Class Examples

-- Let’s go through examples of implementing methods--

Vocabulary

- The line that you use to declare your method is called the method’s **header**.
- The variables that a method creates when it is called and executed are called **parameters**
- The values passed to a method (that are stored in the method’s parameters) are called **arguments**

Header ↘

Parameters ↙ ↘

```
public static int addTwoInts(int valA, int valB)
{
        …
}
```

Naming Methods

- Methods are named using camel case
- The name should describe the task they do
- Examples:

```
    printArray
    computeRadius
    findSurfaceBoundary
```

The **return** statement

- So far, we've seen examples where we return a variable from a method.

- If we have a method called:

```
public static double calculateMean(double[] values)
{
        …
        return result;
}
```

- The code that called the method will be able to assign the value of the returned variable:

```
double[] myValues = {3.24, 4.5, 13.5};
double mean = calculateMean(myValues);
```

A return statement can return an expression

- A return statement need not return a variable. It can also return the result of an expression provided this result is of the data type this method is supposed to return:

```
public static double valueCubed(double value)
{
    return sideLength * sideLength * sideLength;
}
```

Multiple **return** statements

- We can write a method with multiple return statements as long as are sure that every path of execution will lead to a return statement:

```
public static double cubeVolume(double sideLength)
{
        if (sidelength >= 0)
        {
                return sideLength * sideLength * sideLength;
        }
        else
        {
              return 0;
        }
}
```

Avoiding Multiple Return Statements

- Some programmers dislike the use of multiple return statements. You can avoid multiple return statements by storing the method result in a variable that you return in the last statement

Avoiding Multiple Return Statements

```
public static double cubeVolume(double sideLength)
 {
        double volume;

        if (sidelength >= 0)
        {
              volume = sideLength * sideLength * sideLength;
        }
        else
        {
             volume = 0;
        }

        return volume;
 }
```

Common Errors

- Your program will get a compile time error if you make any of the following mistakes:
  - Your method is supposed to return a value, but you do not have a return statement.
  - Your method returns a different data type than what is specified in the method’s signature.
  - If your method is supposed to return a value, but not all paths of execution lead to a return statement

Methods Without Return Values

- Not all methods need to have a return value
- If you would like to define a method without a return value, then you must precede the method name with the data type **void**

```
public static void printArray(double[] values)
{
        for (double val : values)
        {
                System.out.print(val + “ “);
        }
}
```

Scope

- Each variable has a **scope** that defines what parts of your program have access to that variable
- Variables declared inside of a method have scope within that method *only*. You cannot access such a variable from a different method
- Variables whose scope consists of an entire method are called **local variables**
- Variables declared inside of a code block (specified by curly brackets), that variable can only be accessed inside of that that block.
- Examples of code blocks include loops

```
public static int numValuesGreaterThan(double[] array, double value)
{
        int count = 0; // count is a local variable and can only be
                       // inside this method accessed
        for (double element: array)
        {
                // element's scope is inside this loop. It cannot be
                // accessed outside of this loop
                if (element > value)
                {
                        count++;
                }
        }
        return count;
}
```

More Details on Scope

- You can have two variables of the same name and data type as long as their scope's do not overlap
- -- See in-class example --

Examples Using Methods

- Let’s make our MeanMedianMode program more modular using methods
- Let’s create a method for getting user input in order to reduce redundant code in our SheepMaster program

Example

- Let’s break our Bubble Sort method in our MeanMedianMode.java program.
  What does the output look like?

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
	at MeanMedianMode_Methods.bubbleSort(MeanMedianMode_Methods.java:168)
	at MeanMedianMode_Methods.calculateMedian(MeanMedianMode_Methods.java:118)
	at MeanMedianMode_Methods.main(MeanMedianMode_Methods.java:31)
```

We see the trace of method calls that led to the failure in the bubbleSort method.

Method Stubs

- From Horstmann page 224:
- When writing large programs, it is not always feasible to implement and test all methods at once. You often need to test a method that calls another, but the other method hasn't yet been implemented. You can **temporarily** replace the missing method with a **stub**.
- A stub is a method that returns a simple value that is sufficient for testing another method.

Example of a method stub

```
/**
*    Turns a digit into its English name.
*    @param digit an integer between 1 and 9
*    @return the name of digit (“one”,….,”nine”)
*/
public static String digitName(int digit)
{
          return “mumble”
}
```

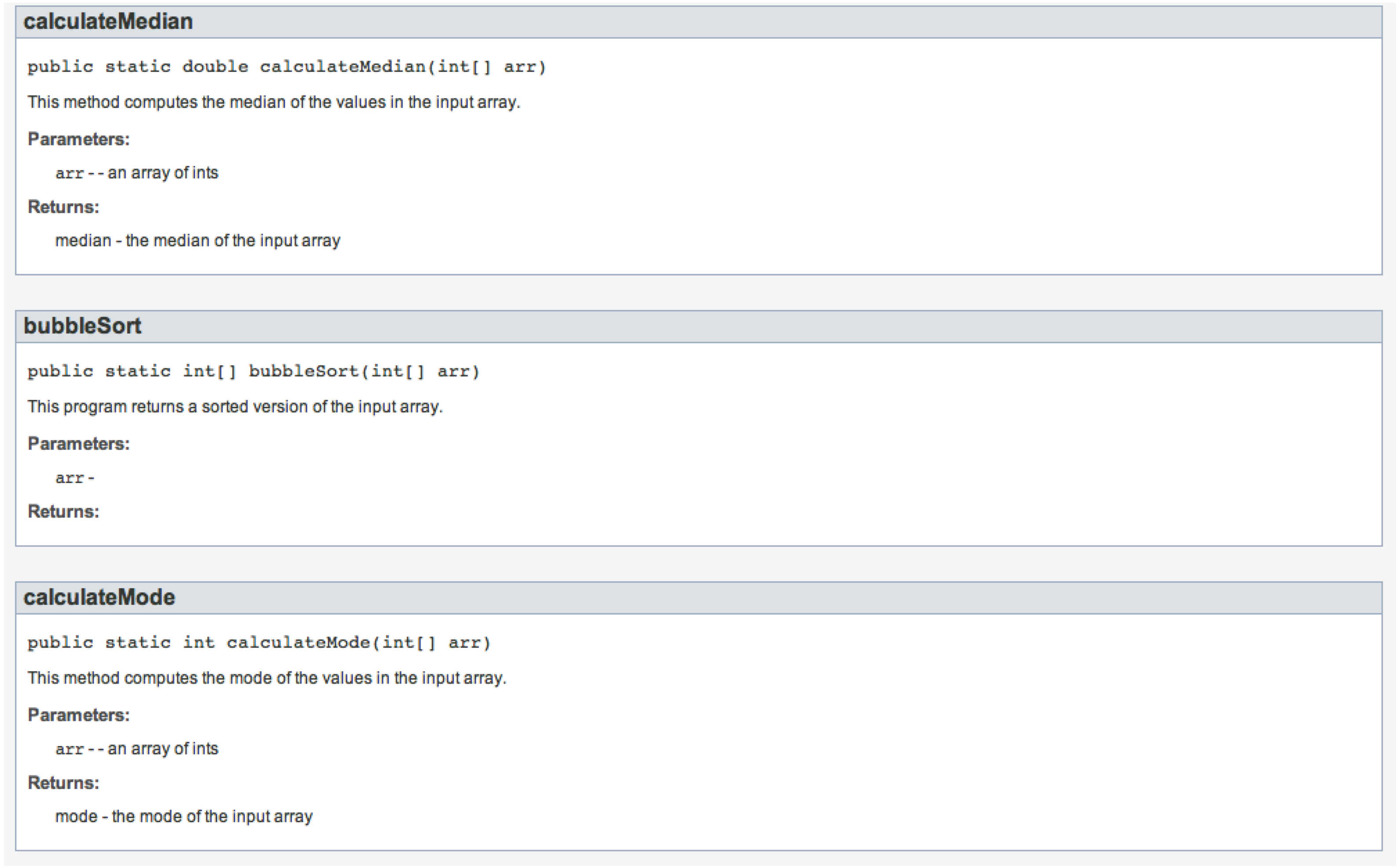
Javadocs

- In Eclipse, the blue comments are used for generating Javadocs.
- Javadocs are sort of like an instruction manual for using your code.
- You create a bunch of .html files which you can use in a website
- Those blue comments you see in Eclipse are what the Javadoc tool uses to generate the documents
- You should only use these blue comments if you want the text inside to be translated to Javadocs

See this website for more information:

- http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

Example of Javadocs



calculateMedian

```
public static double calculateMedian(int[] arr)
```

This method computes the median of the values in the input array.

Parameters:

`arr` - - an array of ints

Returns:

median - the median of the input array

bubbleSort

```
public static int[] bubbleSort(int[] arr)
```

This program returns a sorted version of the input array.

Parameters:

`arr` -

Returns:

calculateMode

```
public static int calculateMode(int[] arr)
```

This method computes the mode of the values in the input array.

Parameters:

`arr` - - an array of ints

Returns:

mode - the mode of the input array

Cool CS Link of the Day

- How Pseudorandom Number generators work:
- http://www.youtube.com/watch?v=itaMNuWLzJo