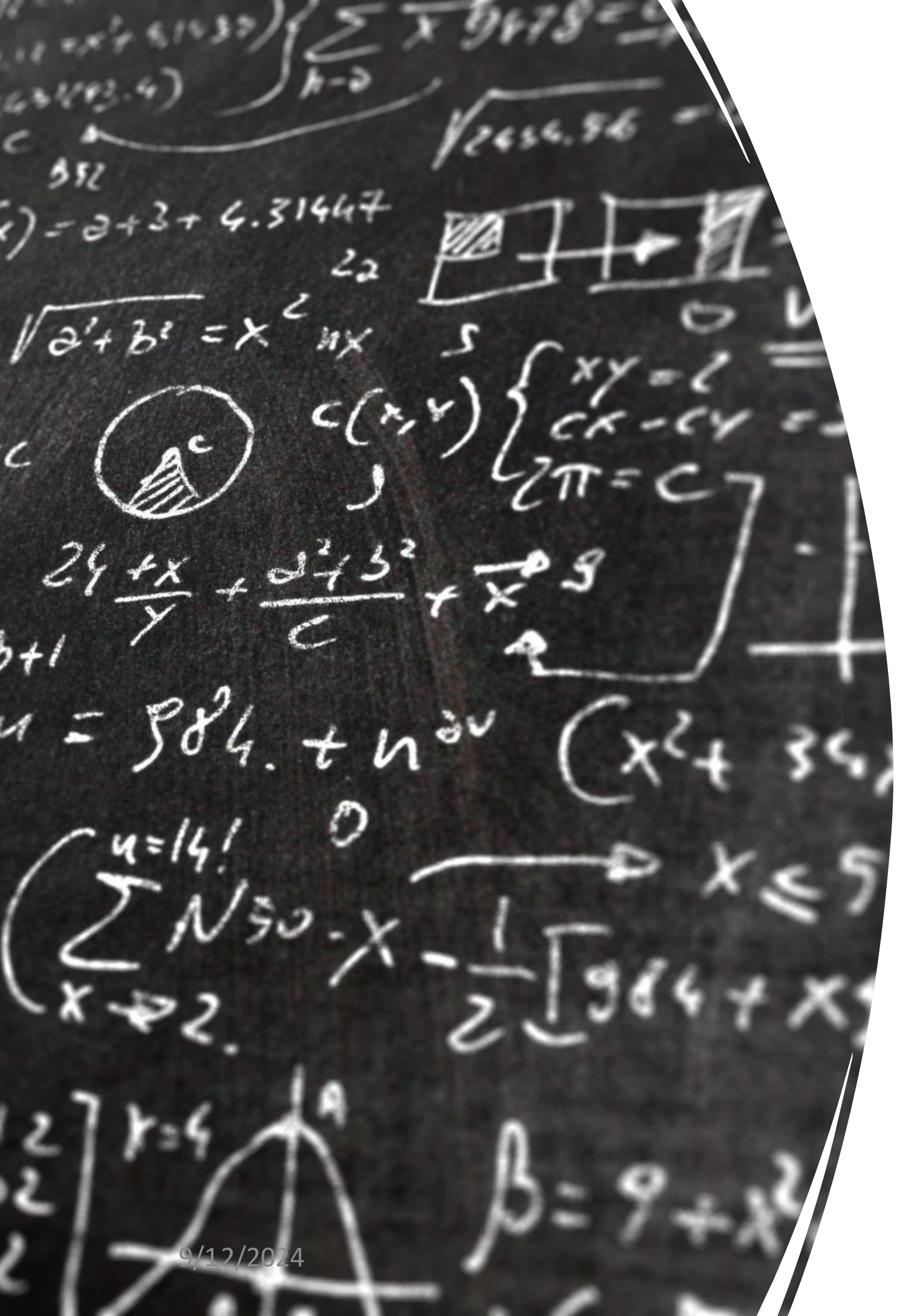


CS 113 – Computer Science I

Lecture 03 – Scanner, Methods I

Thursday 09/12/2023

Announcements

HW00 – due last night

- Regrade requests on Gradescope

HW01 – due this coming Monday night

HW02 – due next Monday

Office hours:

- Adams: Thursday 2:45-4:00pm
- TAs:
  - Sunday – Friday
  - 7:00pm-9:30pm Park 230

Outline

Review

Reading in data - Scanner

Methods

Converting Types (Strings & Numbers)

- Integer to String
    - int a = 23;
    - String numMajors = String.valueOf(a);

- String to integer
    - int x = Integer. parseInt(“40”);

- String to double
    - double a = Double.parseDouble(“40.11”);

Operators & Expressions

- Examples of operators:
  - +, -, /, *, %

- Expression
  - 55 + c

Operator

Operands

Order of operations

- 24 + 10 / 2;
- (24 + 10) / 2;

- Operations between floats and ints:
  - 1 / 3
  - 1 / 3.0

Exercise:

Expression	Value	Data Type
-4		
3.76		
”42.64”		
10 + 3.3		
9 – 5 * 1		
“hot” + “dog”		

String Operators (Textbook: 2.8)

What is the term for combining strings together?

- Concatenation

What is the concatenation operator?

- +

Reading in Data

Scanner class

Another way for reading in data

```
        Scanner sc = new Scanner(System.in);
```

System.in specifies we are reading from user input

What type is “sc”? Is it an int, double, or string?

It’s a Scanner type.

Using Scanner object

Javadocs:
https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

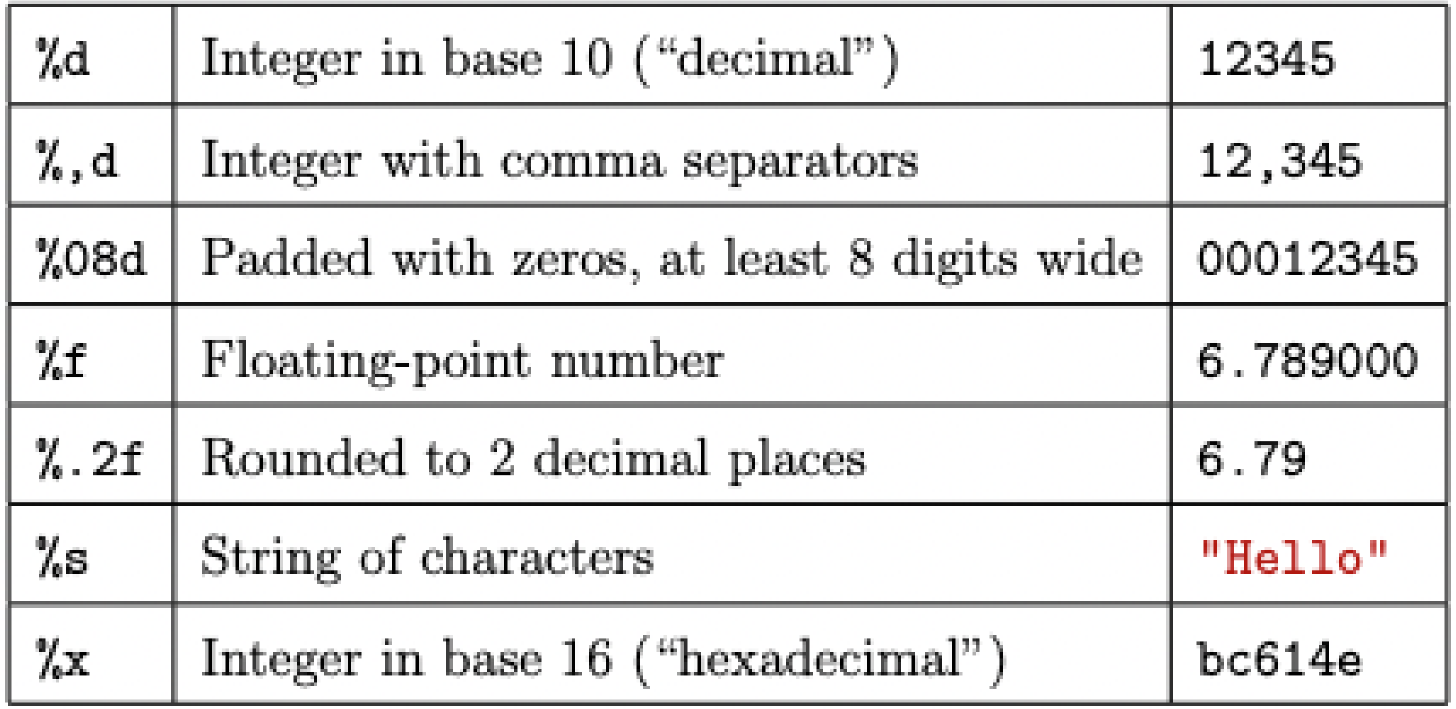
Reading in an integer:

```
        nextInt();
```

Reading in a string:

```
        nextLine();
```

Formatting Strings



%d	Integer in base 10 (“decimal”)	12345
%,d	Integer with comma separators	12,345
%08d	Padded with zeros, at least 8 digits wide	00012345
%f	Floating-point number	6.789000
%.2f	Rounded to 2 decimal places	6.79
%s	String of characters	"Hello"
%x	Integer in base 16 (“hexadecimal”)	bc614e

Table 3.1: Example format specifiers

Review

1.How do you print in Java?

2.How do you read input?

3.What does a declaration statement do?

4.What does an assignment statement do?

5.Give me an example of an illegal variable name.

6.Give me an example of an operator.

Demo

Demo 1: Ask user for a number, and return the square root

```
Math.sqrt(<number>);
```

Demo 2: Lets only print up to 2 decimal places

Demo 3: Lets round that answer to an integer

Math utilities

- Math.round(40.11);
- Math.cos(0);
- Math.sqrt(9);
- Math.random();

Examples of methods

Using methods

Abstraction:

allows us to use functionality without knowing how it works

Demo

Demo 1: Ask user for a number, and return the square root

```
Math.sqrt(<number>);
```

Lets round that answer to an integer

Lets now do this for 2 numbers

Lets now do this for 4 numbers

Lets now do this for 6 numbers

Creating Methods

Idea: Define re-useable portions of code

Analogy: machines with inputs and outputs

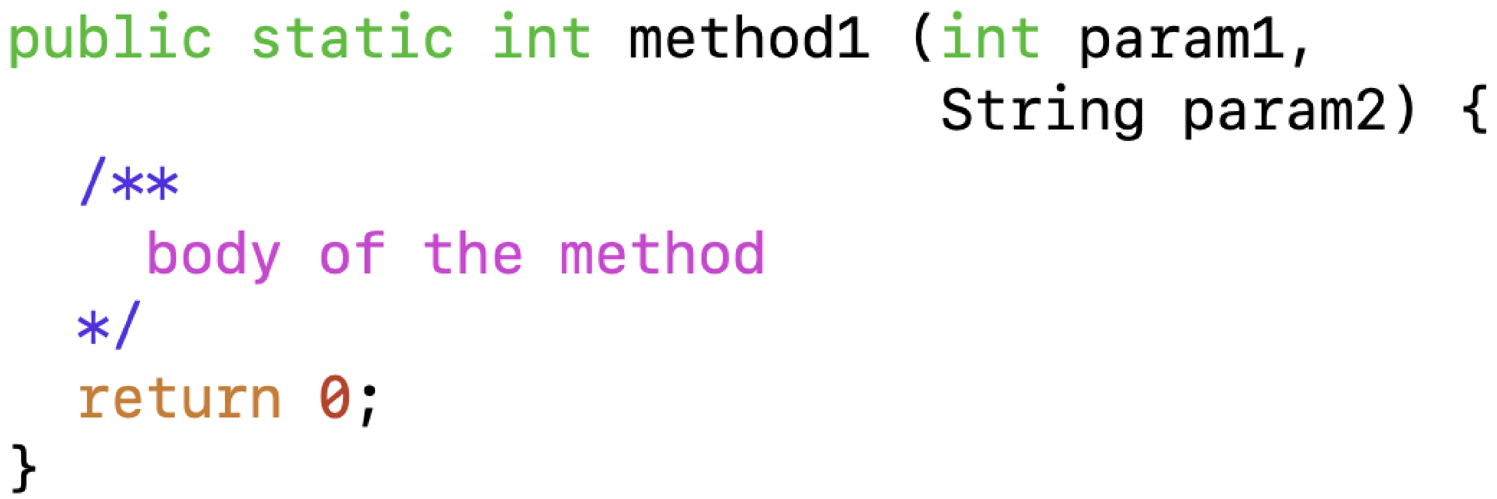
Two steps for programming with functions:

1. Define the function (name, inputs, outputs, implementation)
2. Call the function with inputs and wait for its output

All methods should be contained inside a class

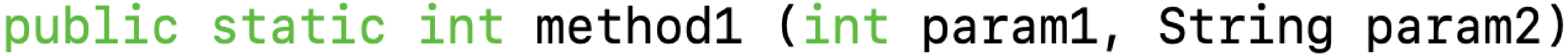
Anatomy of a method

- All methods have the following things:
  - Name
  - Parameter
  - Body
  - Return Type



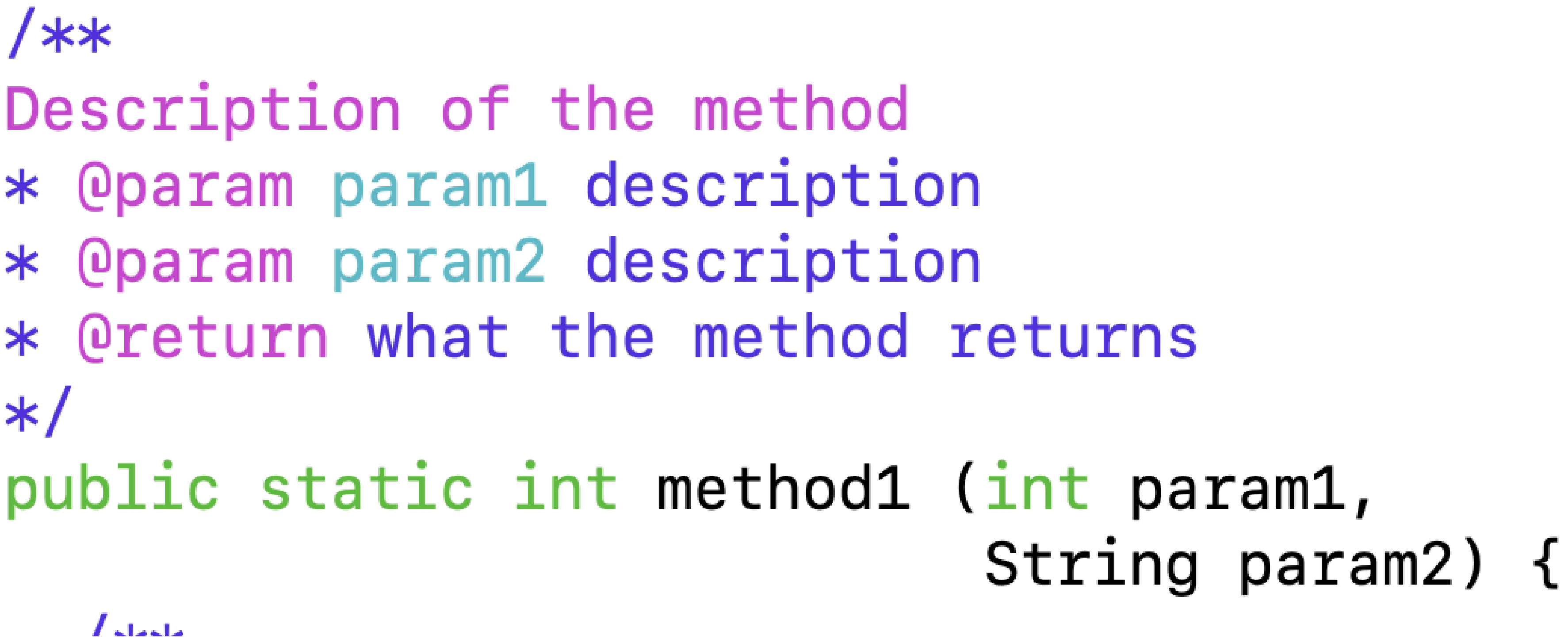
```
public static int method1 (int param1,
                             String param2) {
  /**
    body of the method
  */
  return 0;
}
```

Method signature



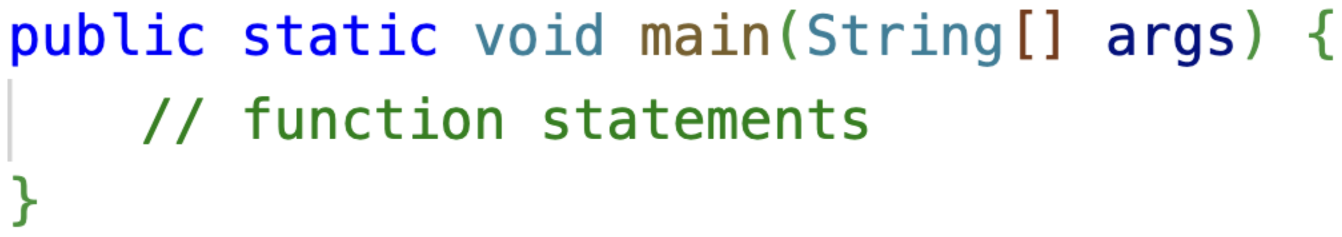
```
public static int method1 (int param1, String param2)
```

Method documentation

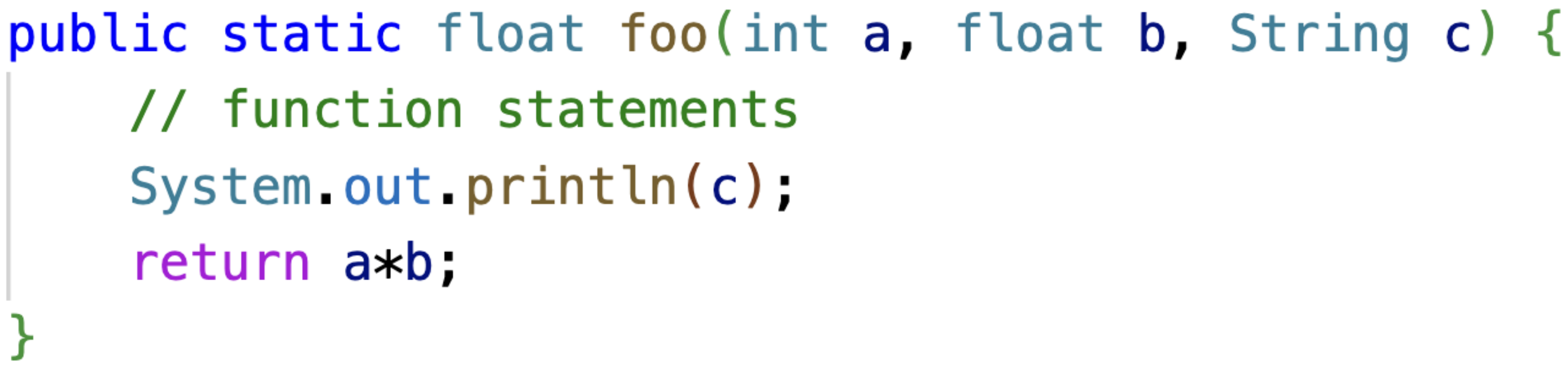


```
/**
Description of the method
* @param param1 description
* @param param2 description
* @return what the method returns
*/
public static int method1 (int param1,
                             String param2) {
   /**
```

Defining methods in Java: syntax

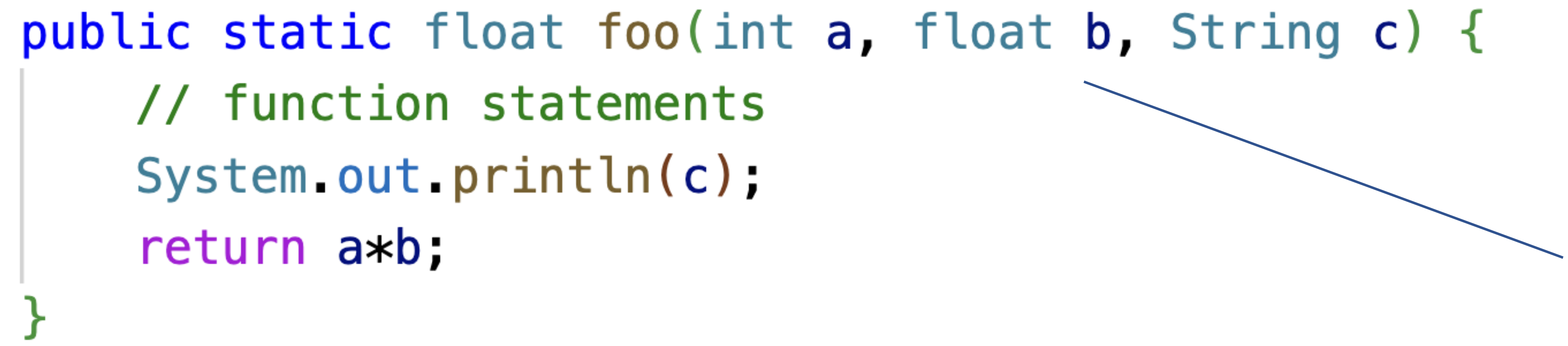


```java
public static void main(String[] args) {
    // function statements
}
```



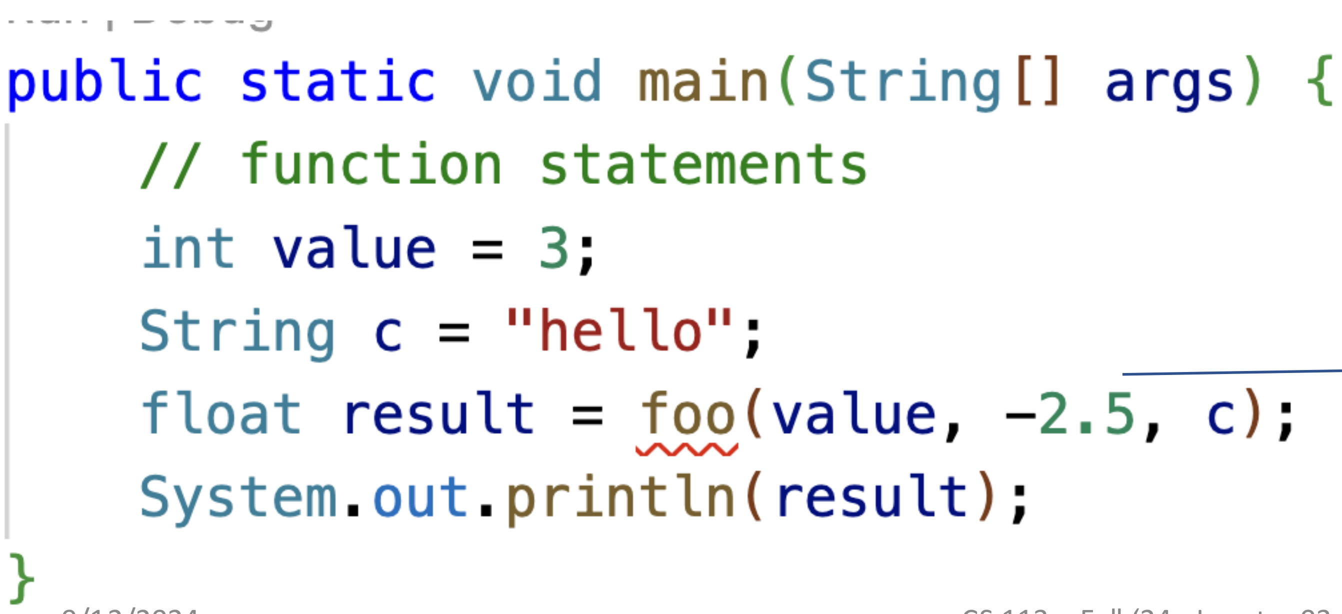
```java
public static float foo(int a, float b, String c) {
    // function statements
    System.out.println(c);
    return a*b;
}
```

Calling methods in Java: syntax



```java
public static float foo(int a, float b, String c) {
    // function statements
    System.out.println(c);
    return a*b;
}
```

parameters



```java
Run | Debug
public static void main(String[] args) {
    // function statements
    int value = 3;
    String c = "hello";
    float result = foo(value, -2.5, c);
    System.out.println(result);
}
```

arguments

Executing a method: steps

1. When you encounter a method, pause!
2. Create a *frame* to hold the method state
3. Copy argument values
4. Execute the method, line by line. Continue until
    1. you hit a return statement
    2. you run out of statements
5. Send back return value (can be nothing if function is *void*)
6. Delete the method’s frame
7. Resume original function

Exercise: Draw stack diagram

```java
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```

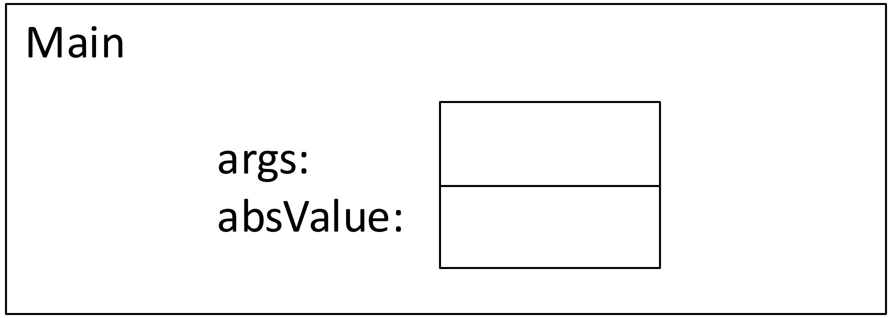
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



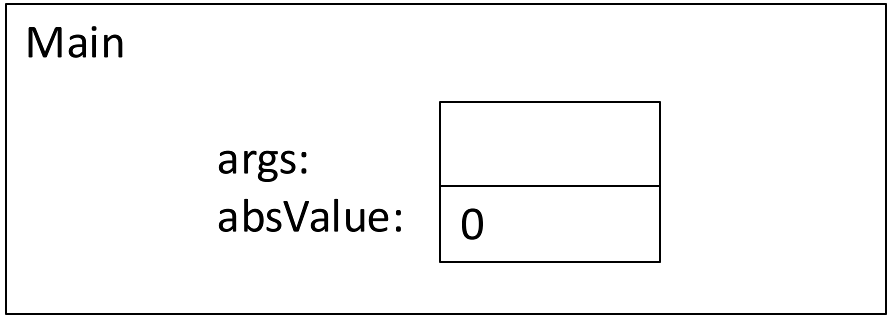
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



Main

args:

absValue: 0

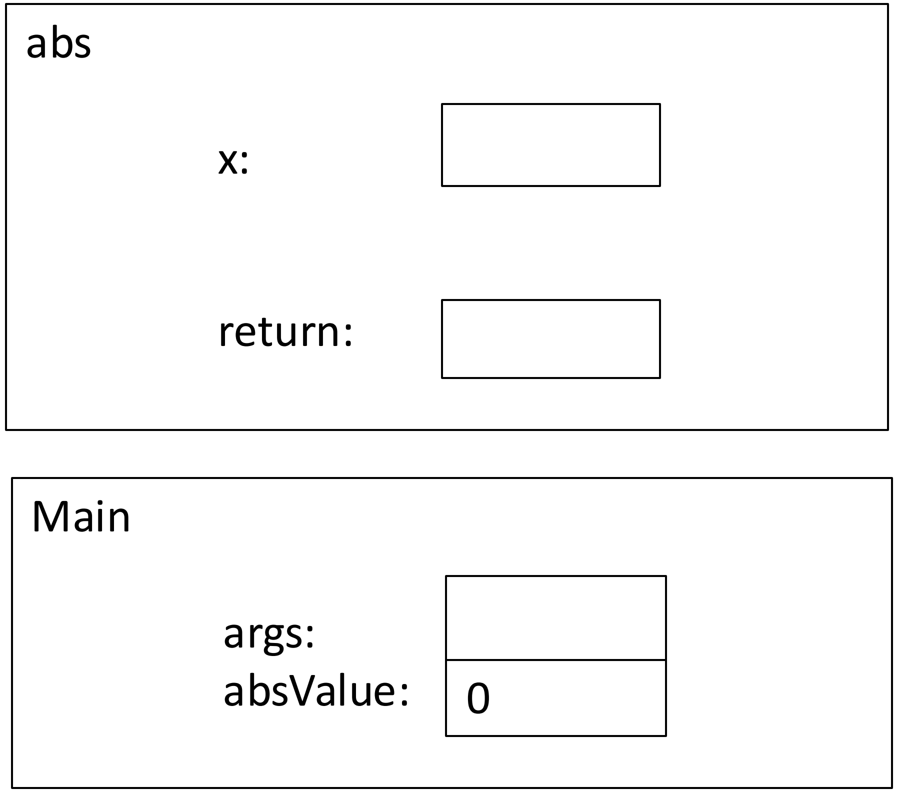
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x:

return:

Main

args:

absValue: 0

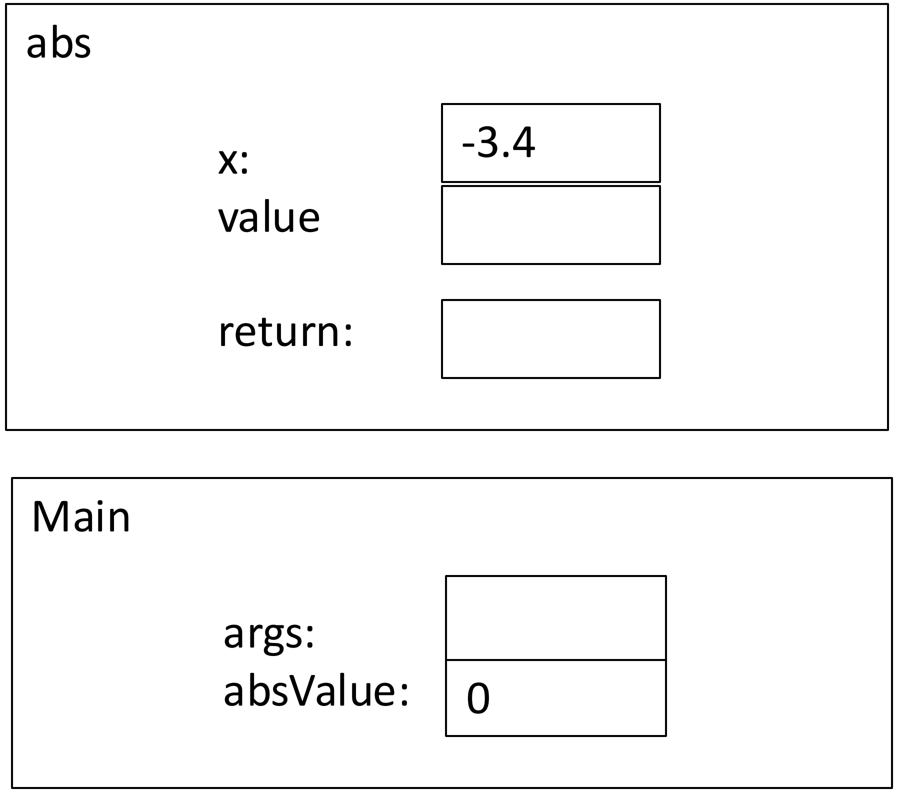
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x: -3.4

value

return:

Main

args:

absValue: 0

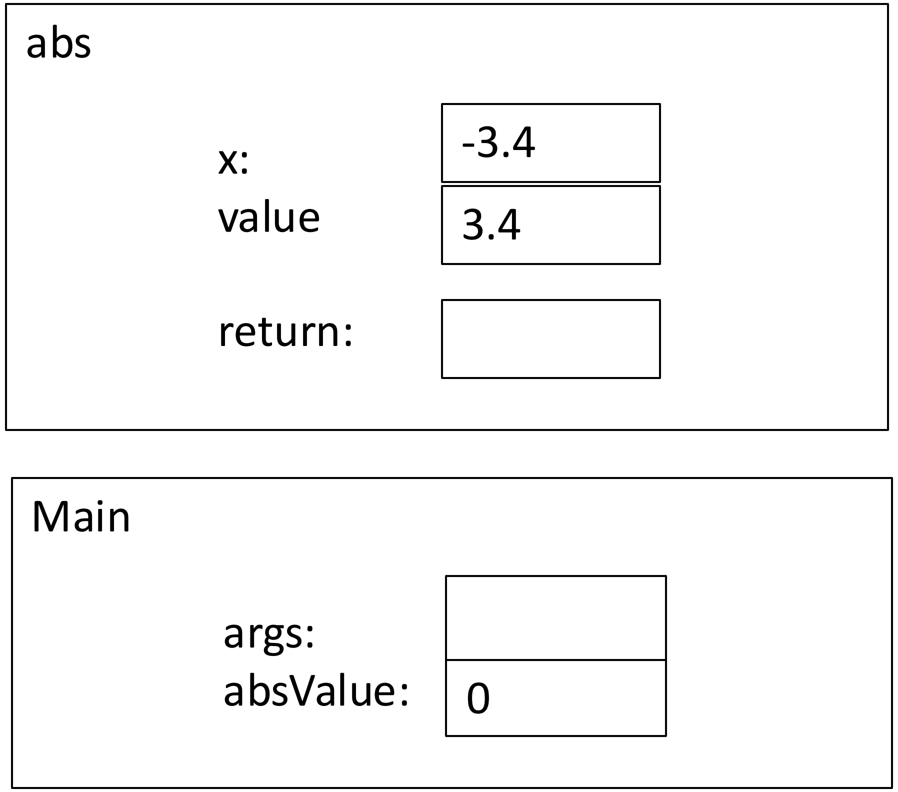
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x: -3.4

value 3.4

return:

Main

args:

absValue: 0

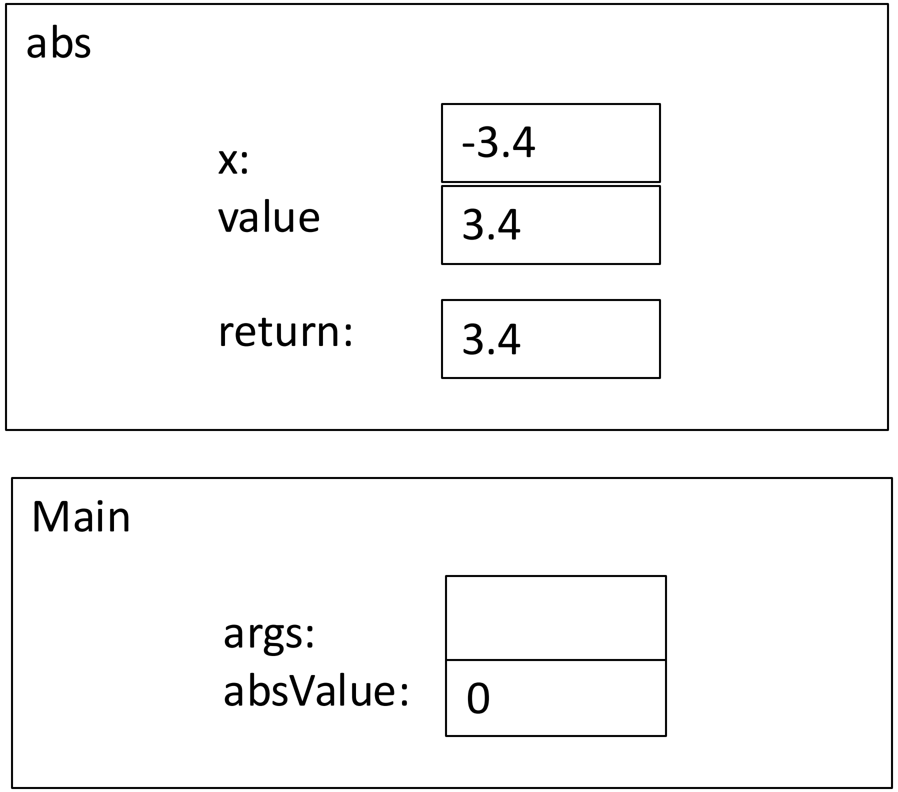
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x: -3.4

value 3.4

return: 3.4

Main

args:

absValue: 0

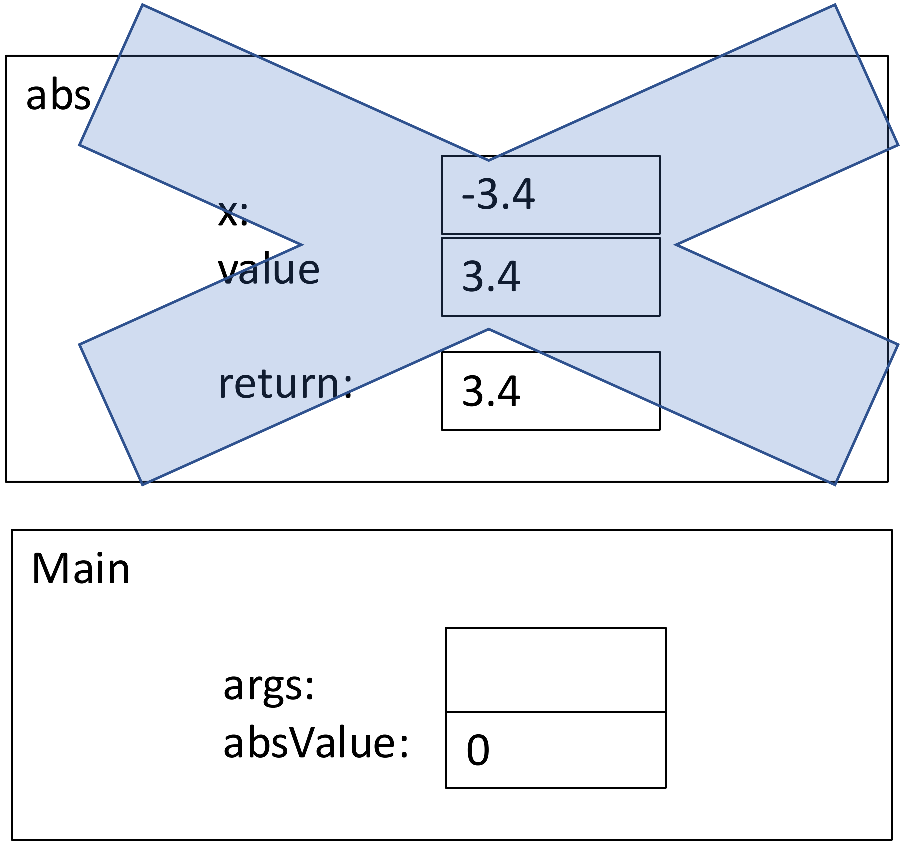
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x: -3.4

value 3.4

return: 3.4

Main

args:

absValue: 0

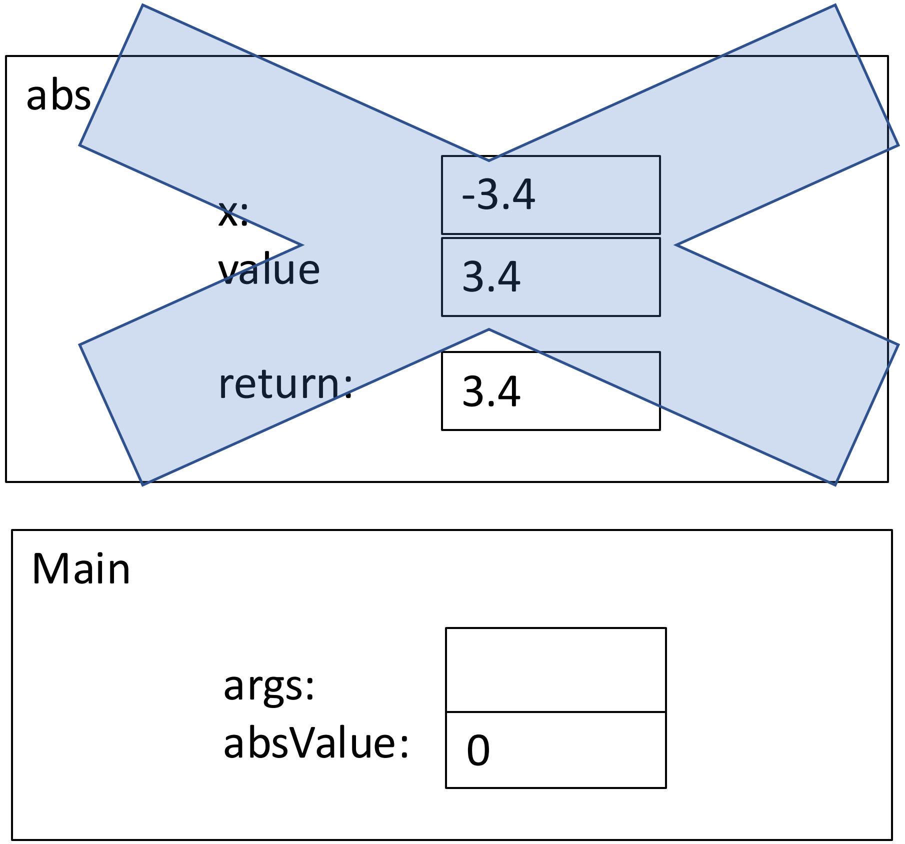
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x: -3.4

value 3.4

return: 3.4

Main

args:

absValue: 0

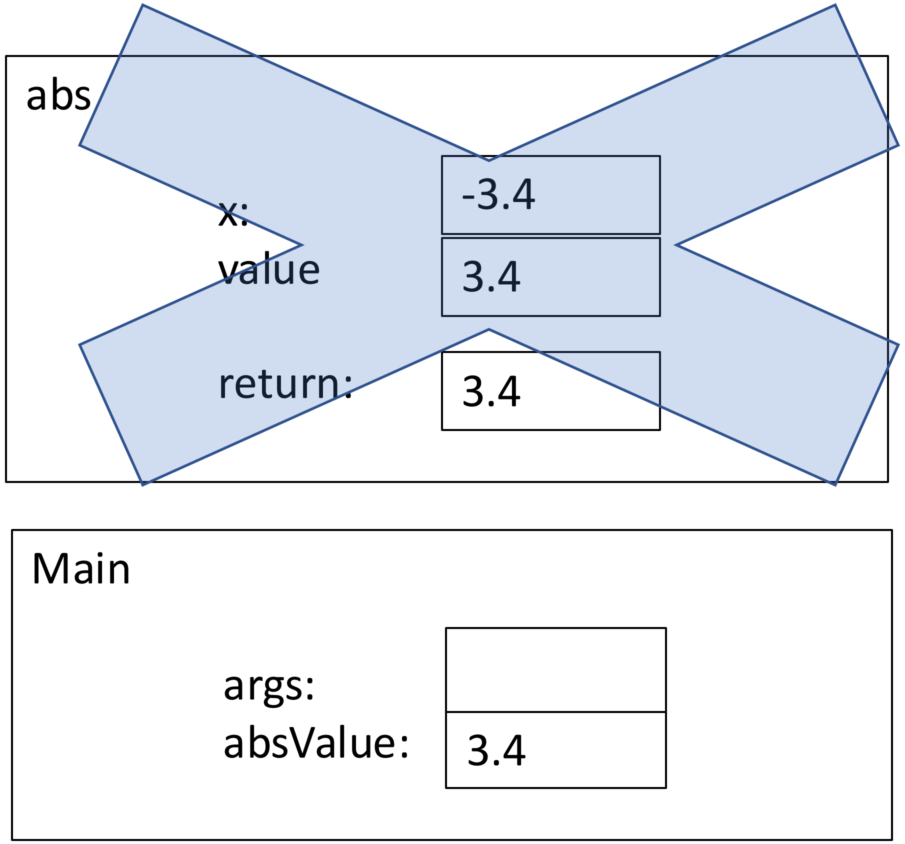
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(-3.4);
    }
}
```



abs

x: -3.4

value 3.4

return: 3.4

Main

args:

absValue: 3.4

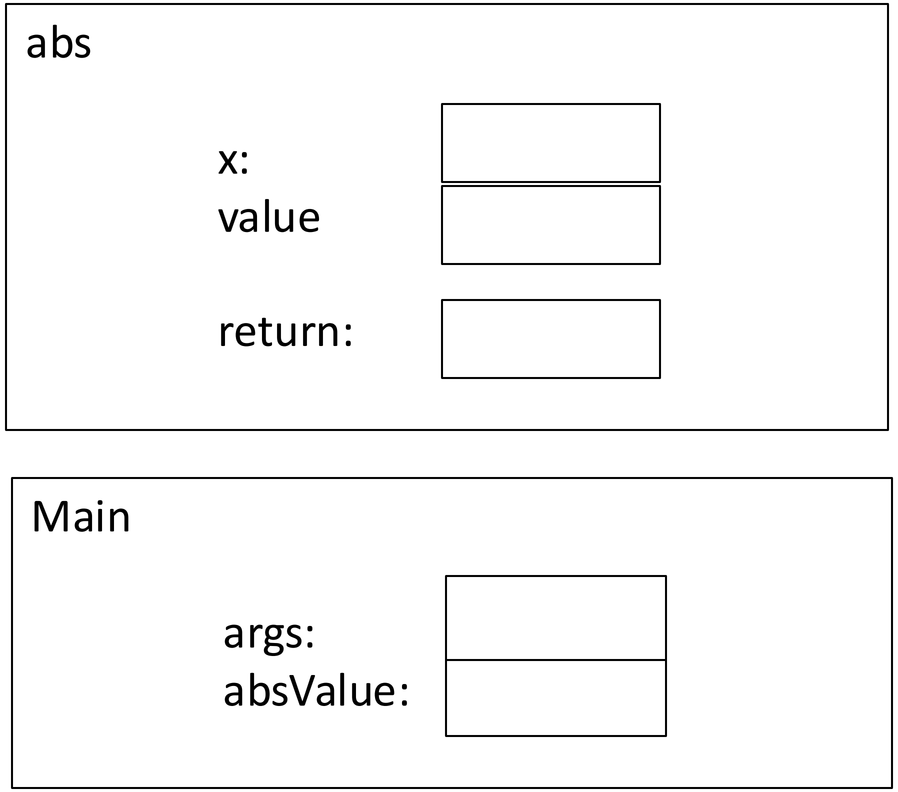
Exercise: Draw stack diagram

```
public class Neg {

    public static double neg(double x) {
        double value = x * -1
        return value;

    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = neg(5.4);
    }
}
```



abs

x:

value

return:

Main

args:

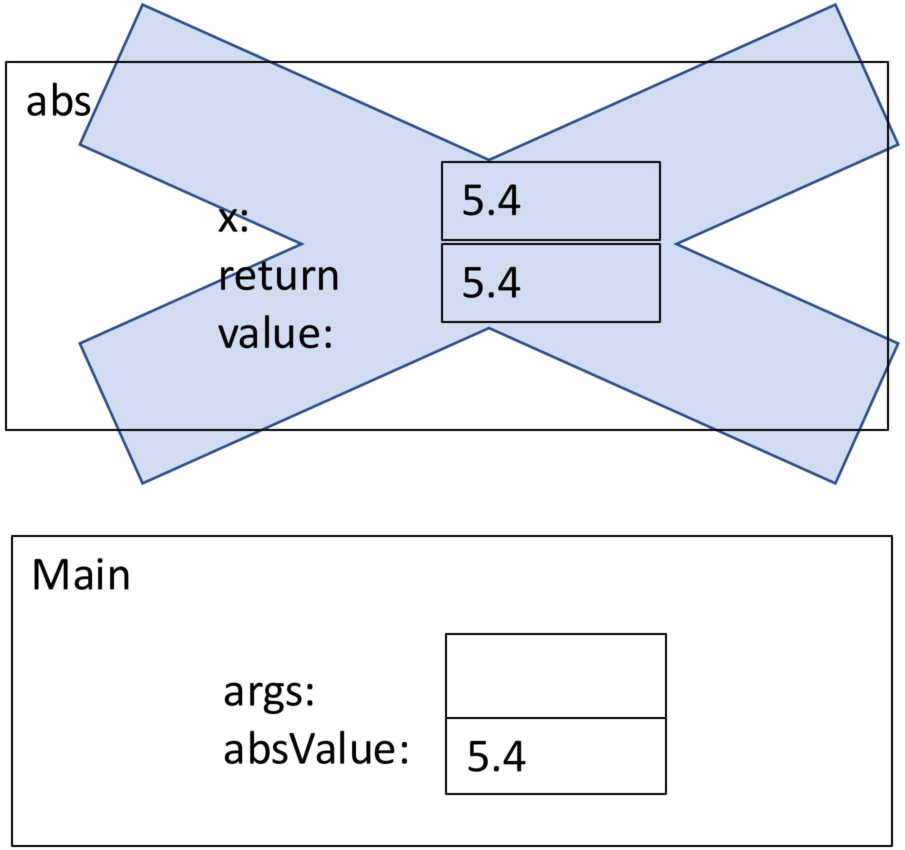
absValue:

Exercise: Draw stack diagram

```
public class Abs {

    public static double abs(double x) {
        if (x < 0) {
            return -x;
        }
        return x;
    }

    public static void main(String[] args) {
        double absValue = 0;
        absValue = abs(5.4);
    }
}
```



abs

x: 5.4

return value: 5.4

Main

args:

absValue: 5.4

What is different here?

```
// Function: area
// Description: computes the area of a rectangle
// Input: width (double)
// Input: height (double)
// returns (double), the area as width * height
// side effects: none
public static double area(double width, double height) {
    return width * height;
}
```

```
// Function: area
// Description: computes the area of a rectangle
// Input: width (double)
// Input: height (double)
// returns (none)
// Side effect: prints the area to the console
public static void area(double width, double height) {
     double a = width * height;
     System.out.println("Area is "+ a);
}
```

Warning: don’t confuse printing with returning

```java
// Function: area
// Description: computes the area of a rectangle
// Input: width (double)
// Input: height (double)
// returns (double), the area as width * height
// side effects: none
public static double area(double width, double height) {
    return width * height;
}
```

```java
// Function: area
// Description: computes the area of a rectangle
// Input: width (double)
// Input: height (double)
// returns (none)
// Side effect: prints the area to the console
public static void area(double width, double height) {
     double a = width * height;
     System.out.println("Area is "+ a);
}
```

Benefits of methods

- Split large problems into small problems
- Easier to maintain code/cleaner code
  - Only need to fix mistakes
  - DRY: Don’t repeat yourself
- Implement once, re-use in different programs
- Abstract details so user doesn’t need to worry about details

Method: IsInteger

```
$ java CheckInput
Enter an integer: aplle
That is not an integer!!
Enter an integer: 0.0
That is not an integer!!
Enter an integer: 0-3
That is not an integer!!
Enter an integer: -4
You entered: -4

$ java CheckInput
Enter an integer:
That is not an integer!!
Enter an integer: 498756.0
That is not an integer!!
Enter an integer: 498756
You entered: 498756
```

Method specifications

Idea: “contract” between the function user and the method implementation

Inputs and their types

Return type

Description of how function behaves, including special cases and side effects

A **side effect** refers to changes the method makes that last after the method returns (e.g. printing to the console is a side effect)

The **method signature** includes just the inputs and outputs of the function

Method Specifications

```
/**
 * Returns a random real number from a Gaussian distribution with
 *  mean &mu and standard deviation &sigma
 *
 * @param mu the mean
 * @param sigma the std
 * @ return a real number distributed according to the Gaussian distribution
 * /
public static double gaussian(double mu, double sigma) {
        return mu + sigma * gaussian();
}
```

Why have method specifications?

- Make the behavior of function clear
- Enable user to use function without having to look at the implementation

Unit testing

Verify that method is implemented correctly

Call the method with different inputs and check the results

In a library, we can use the main method to test methods

Top down design

1. Identify features of the program
    1. List them out!
2. Identify verbs and nouns in feature list
    1. Verbs: functions
    2. Nouns: objects/variables
3. Sketch major steps – how features should fit together
    1. Algorithm!
4. Write program skeleton
    1. Include function **stubs** (placeholders for our functions)
    2. Function **stub:** empty function with parameters and return type
5. Implement and test function stubs one at a time