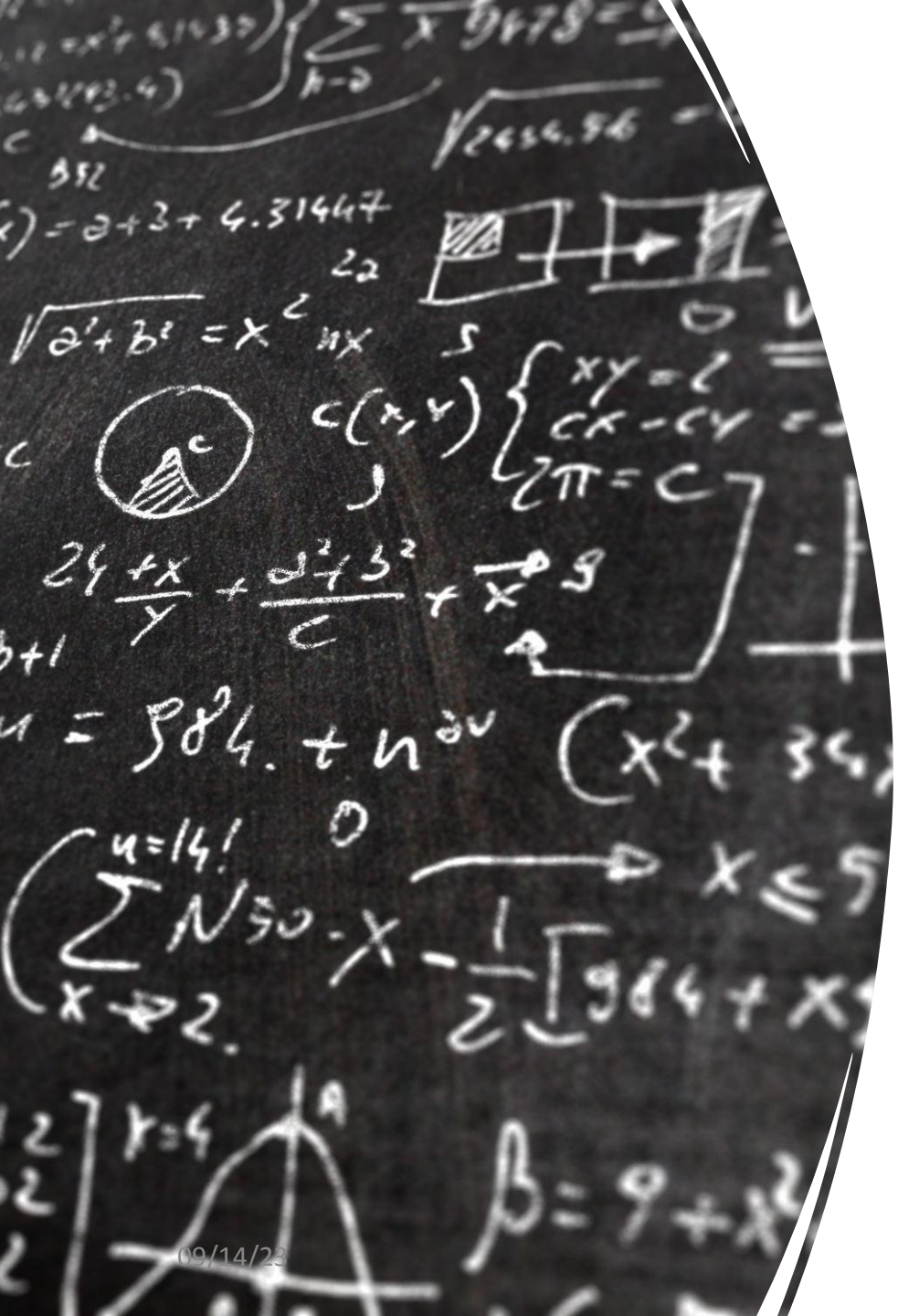


CS 113 – Computer Science I

Lecture 04 – Methods I

Thursday 02/1/2024

Announcements

- HW00 – due tonight
- HW01 - released Sunday

Github recitation next class

create a github account before next class

Methods:

- a set of instructions grouped together to perform a particular operation

- We've called a few methods in this class already:
  - `System.out.println();`
  - `Integer.parseInt();`
  - **what do these have in common? Parenthesis!**

Methods:

What's the purpose of a method?

Let's rewrite LeapYears with a method!

What did we do here?

- organized computation in a unit which we can reuse

Methods:

Advantages of a method:

- Re-usability
- Abstraction!

  - Do you know how `System.out.println()` works? NO! and you don't need to in order to use it.

  - You can drive a car without needing to know how it works

Methods

Let’s go back to our code and look more closely at what we did....

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 within a class

Method signatures

```
public static int printNextYear(int year) {
   //....
}
```

- All methods have the following things:
  - access modifiers (public and static) ... ignore for now
  - Return Type - int, bool, String, **void**
  - Parameters - int year
  - Body - code between the brackets

More complex example

Write a method to “guess” a user’s age

1. Ask the user to enter a number between 1 and 10
2. Multiply that number by 2
3. Add 1766
4. Ask what year the user was born
5. Subtract the year they were born
6. Return the last two digits in your result.. that is the age

Write methods for step (4) and step (6) as well

Defining methods in Java: syntax

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

Calling methods in Java: syntax

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

parameters

```
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:

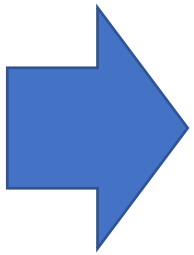
stack diagram: illustrates the execution of a program and value of variables within a program

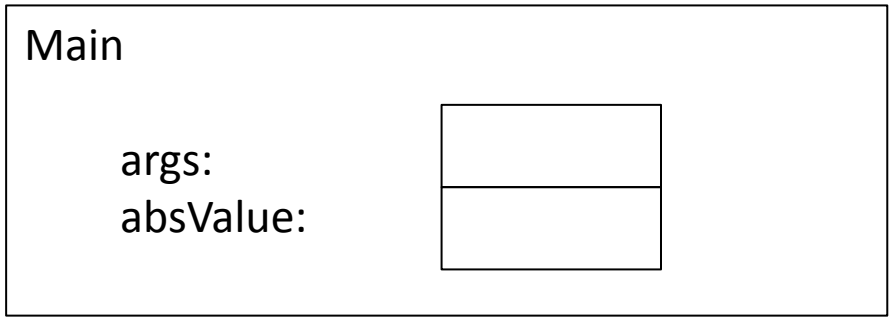
Exercise: Draw stack diagram

```
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

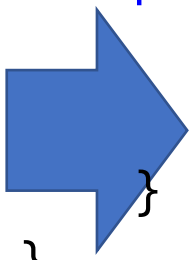
```
public class Neg {

    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);
    }
}
```

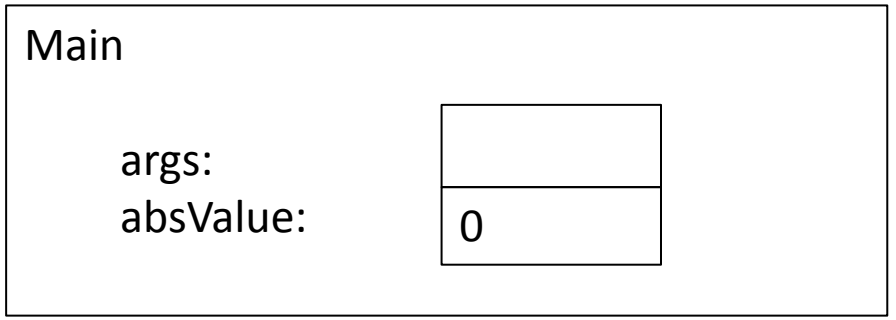




Exercise: Draw stack diagram

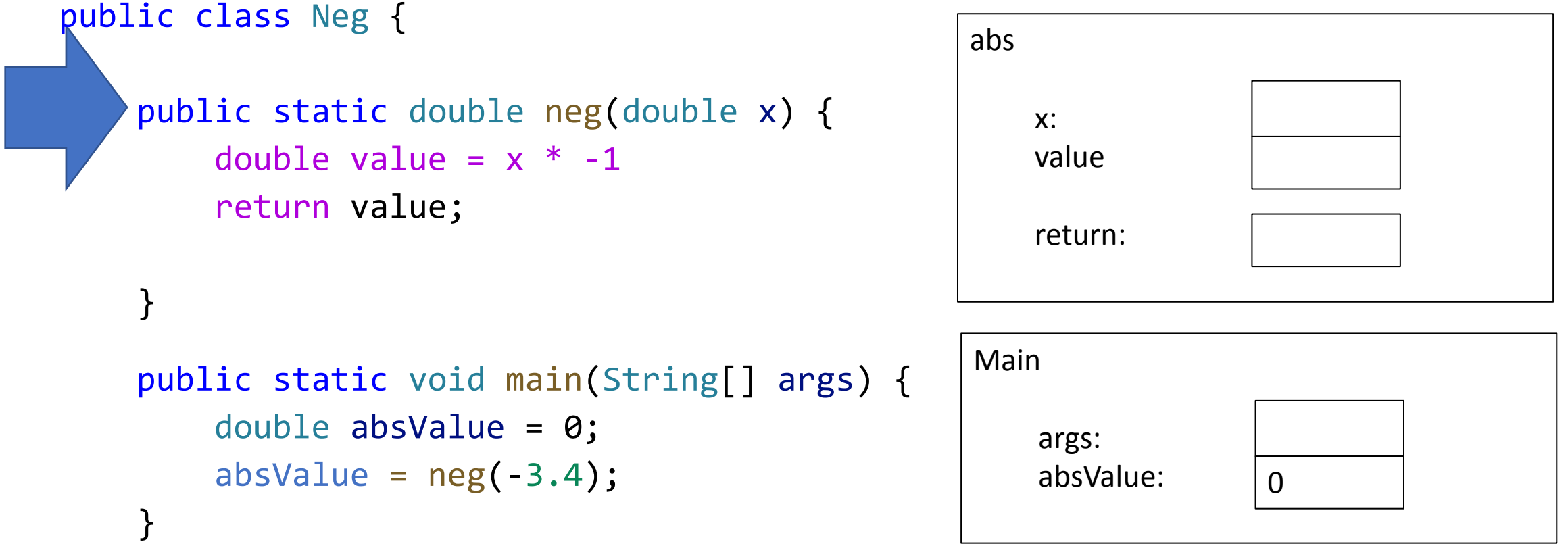
```
public class Neg {

    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);
    }
}
```





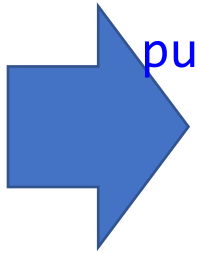
Exercise: Draw stack diagram

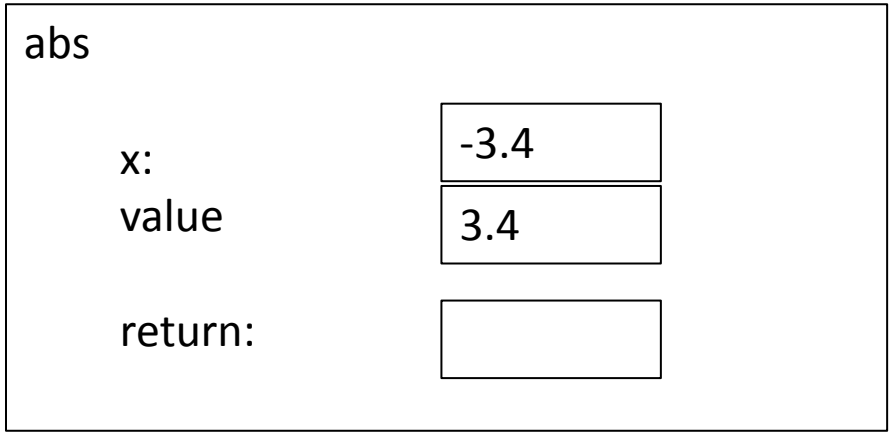
```
public class Neg {

    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);
    }
}
```

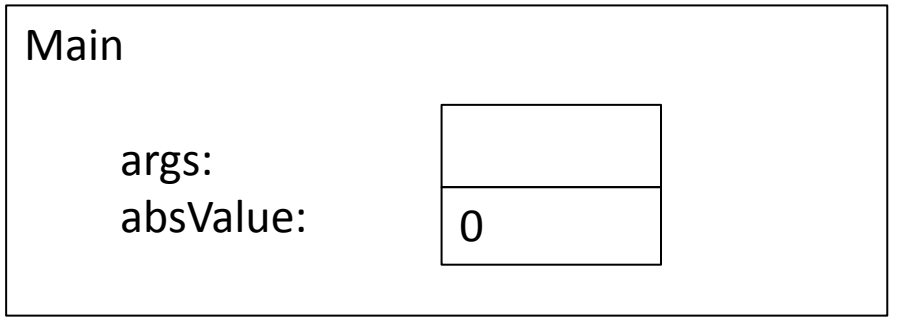


Exercise: Draw stack diagram

```
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);
    }
}
```

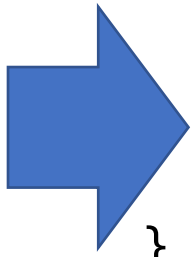


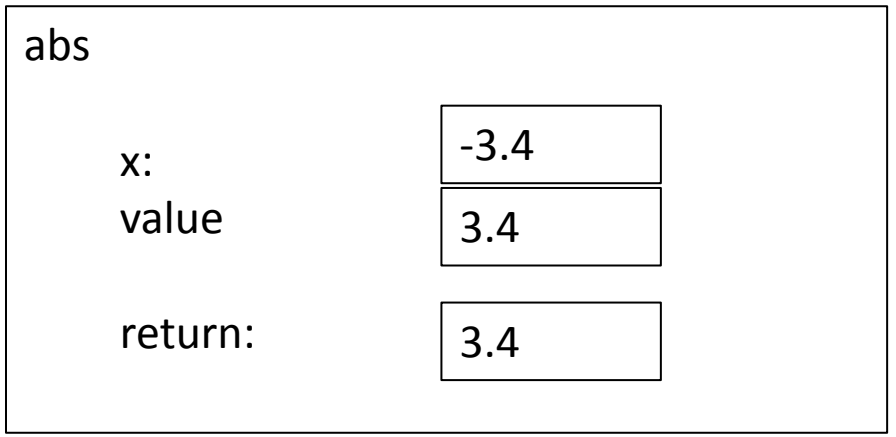


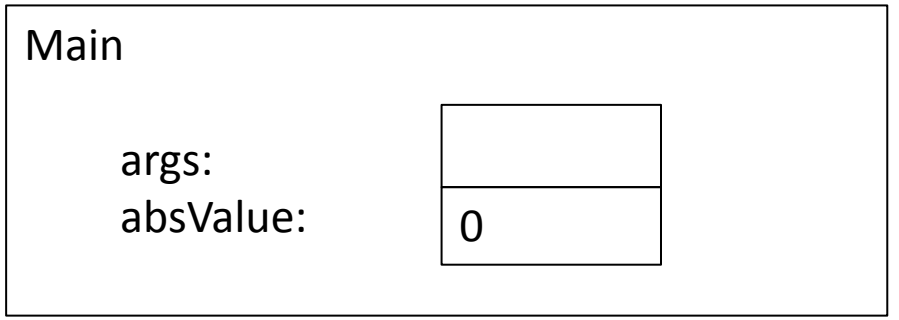


Exercise: Draw stack diagram

```
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);
    }
}
```

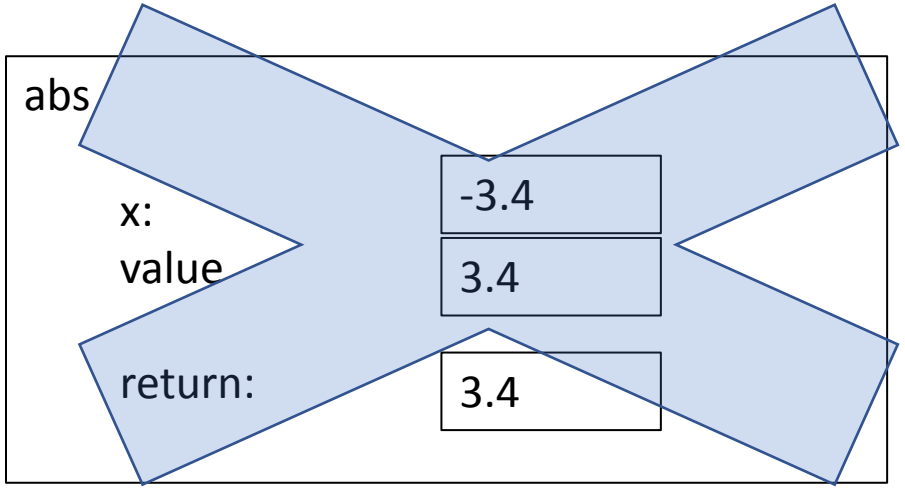


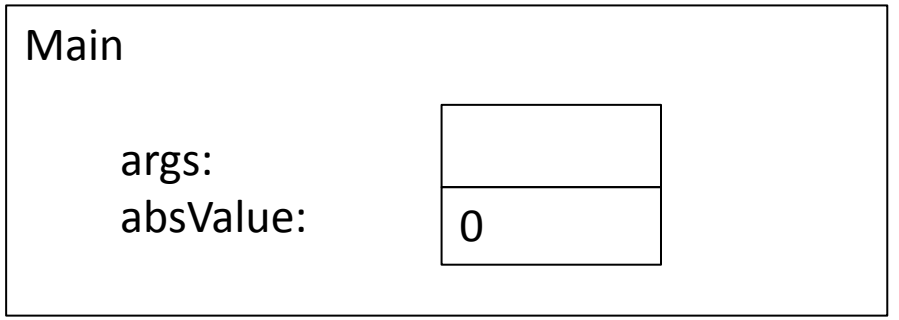




Exercise: Draw stack diagram

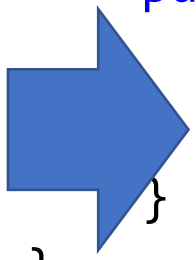
```
public class Neg {

    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);
    }
}
```

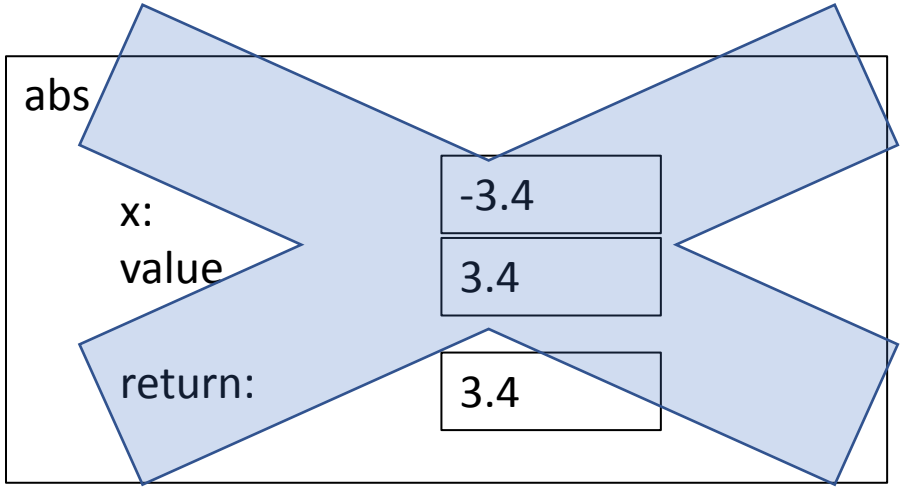


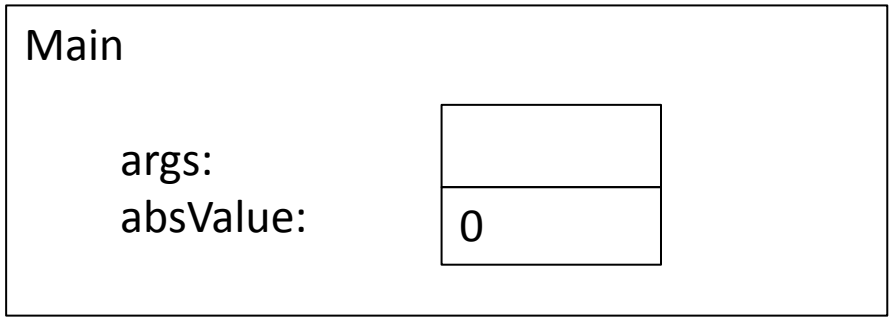


Exercise: Draw stack diagram

```
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);
    }
}
```

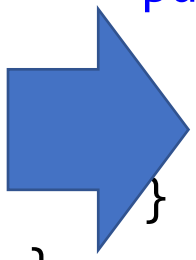


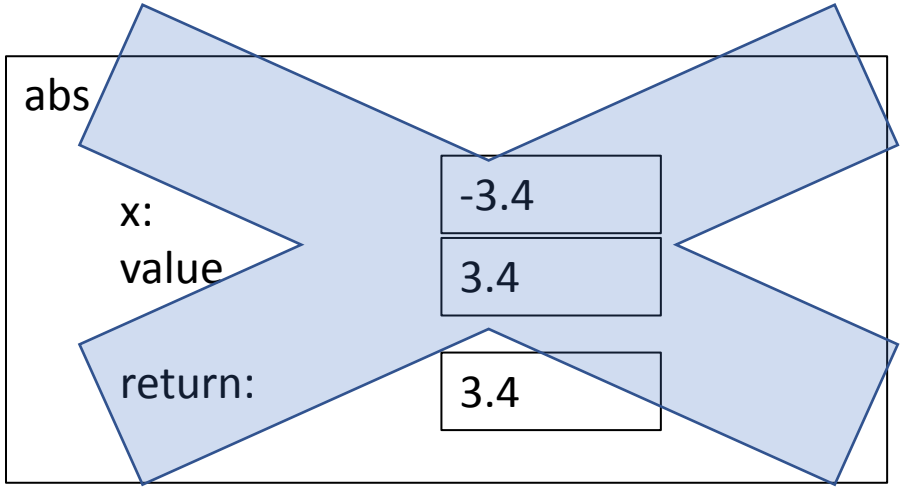


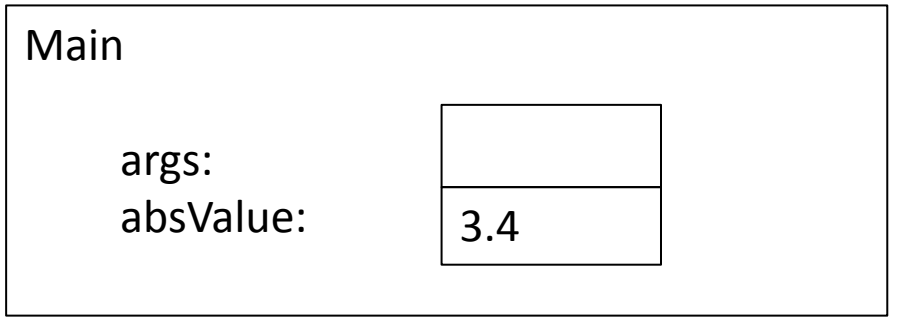


Exercise: Draw stack diagram

```
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);
    }
}
```

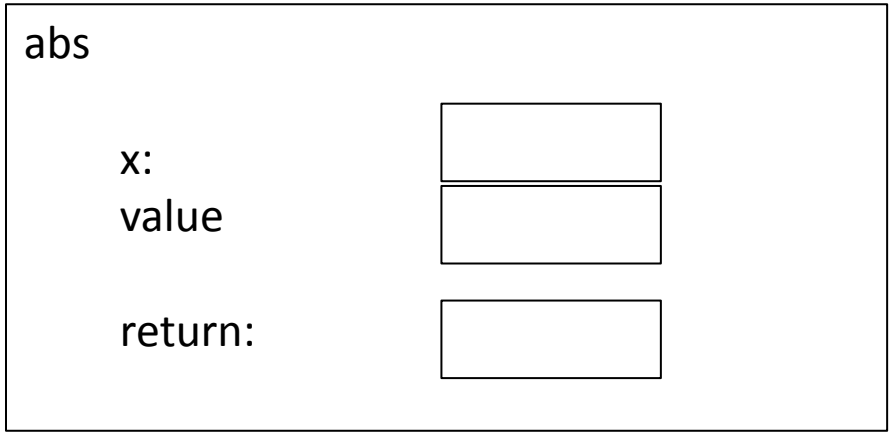


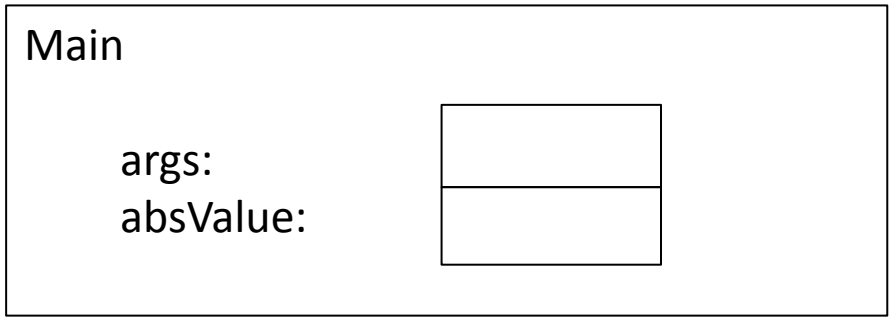




Exercise: Draw stack diagram

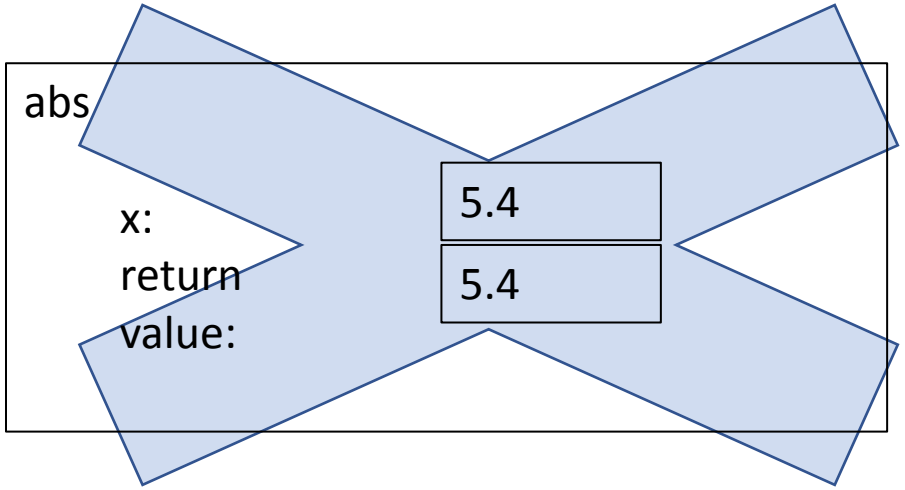
```
public class Neg {

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

    }

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

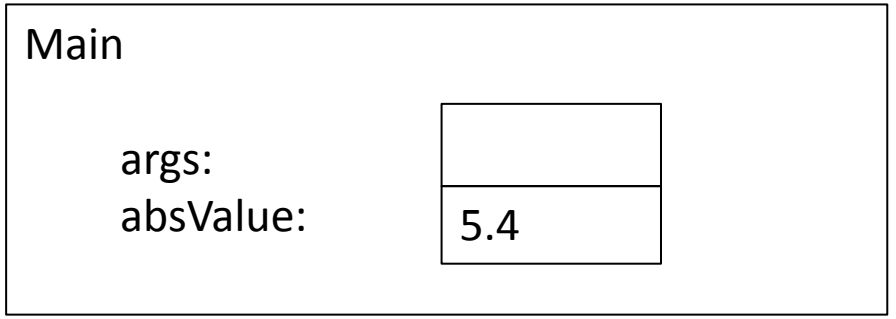




Exercise: Draw stack diagram

```
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);
    }
}
```





Stack Diagram for GuessYear

What is different here?

```
int area(double width, double height) {
    return width * height;
}
```

```
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

Benefits of methods

- Split large problems into small problems

  Easier to maintain code/cleaner code
  - Only need to fix mistakes once

- Implement once, re-use in different programs

- Abstract details so user doesn’t need to worry about details

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) {
    /**
```

Method specifications

"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