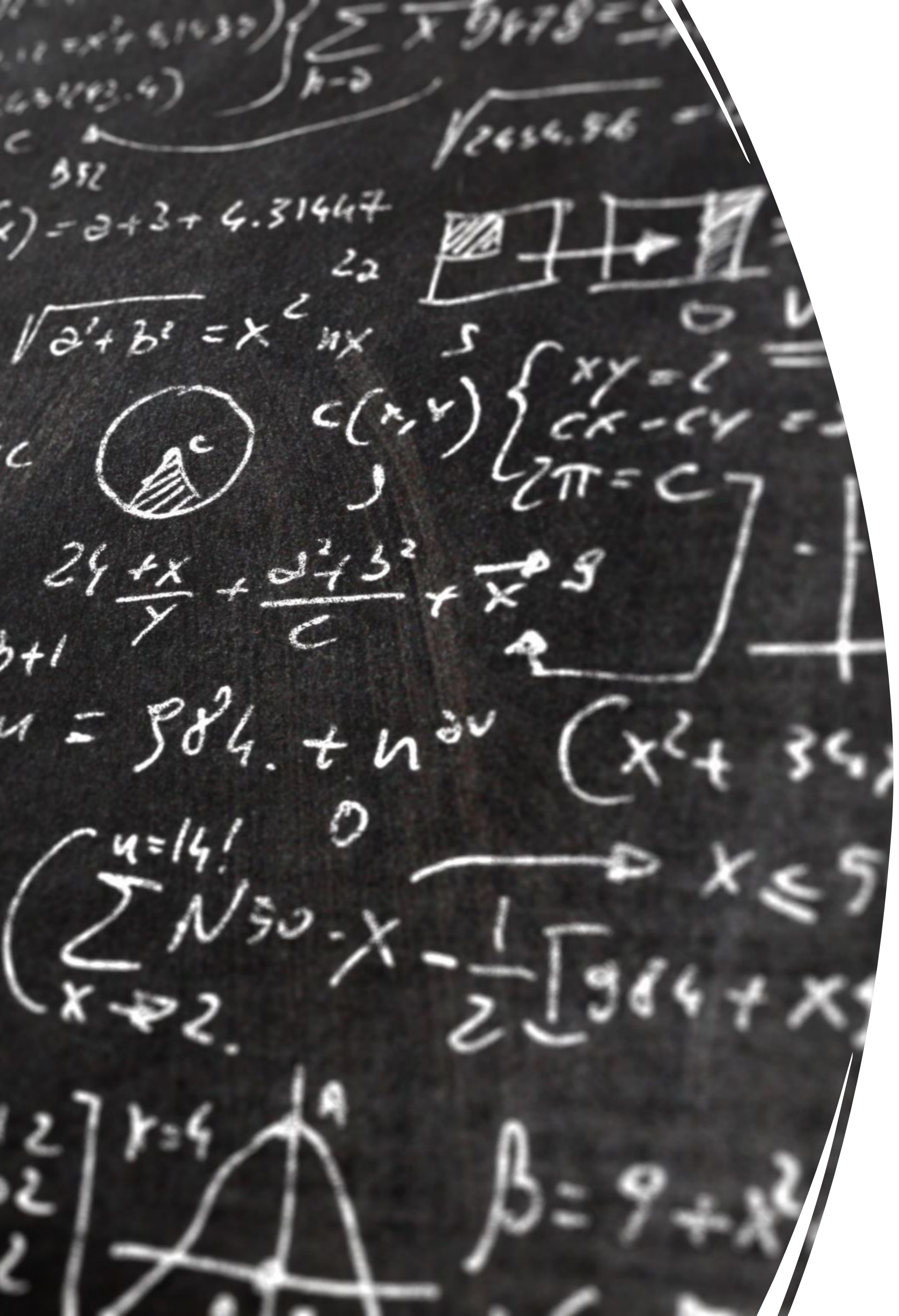


CS 113 – Computer Science I

Lecture 4 – Methods

Adam Poliak

01/26/2023

Announcements

- Assignment 00
  - Grades returned
- Assignment 01 – due 02/01
  - Released today
- Office hours:
  - TAs: almost finalized, will be posted on course website
  - Mine: 3:30 – 4:30pm today

Agenda

- Announcements
- Recap
- Methods

Recap

- “Terminal as a window in your own computer”
- Command line programs

- Variables:
    - Storing different types of data
    - Attributes of variables

- Input/Output in data
    - Output: System.out.print*
    - Input: Scanner

Agenda

- Announcements
- Recap
- Methods

Demo

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

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

Demo 2: 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

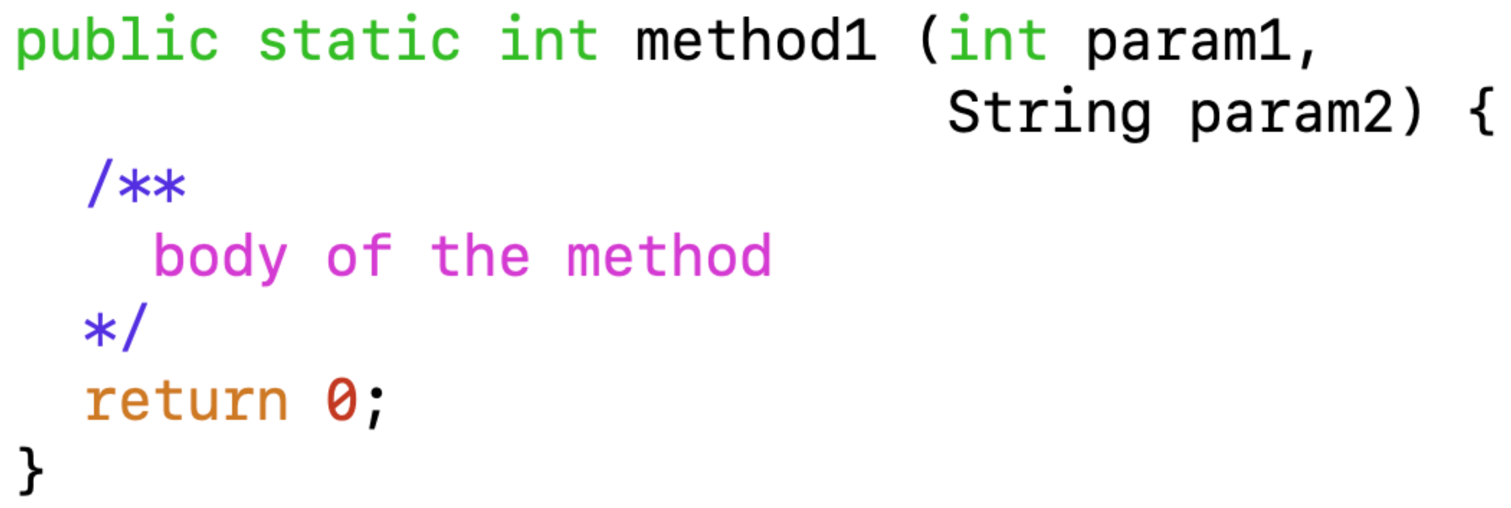
Two steps for programming with functions:

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

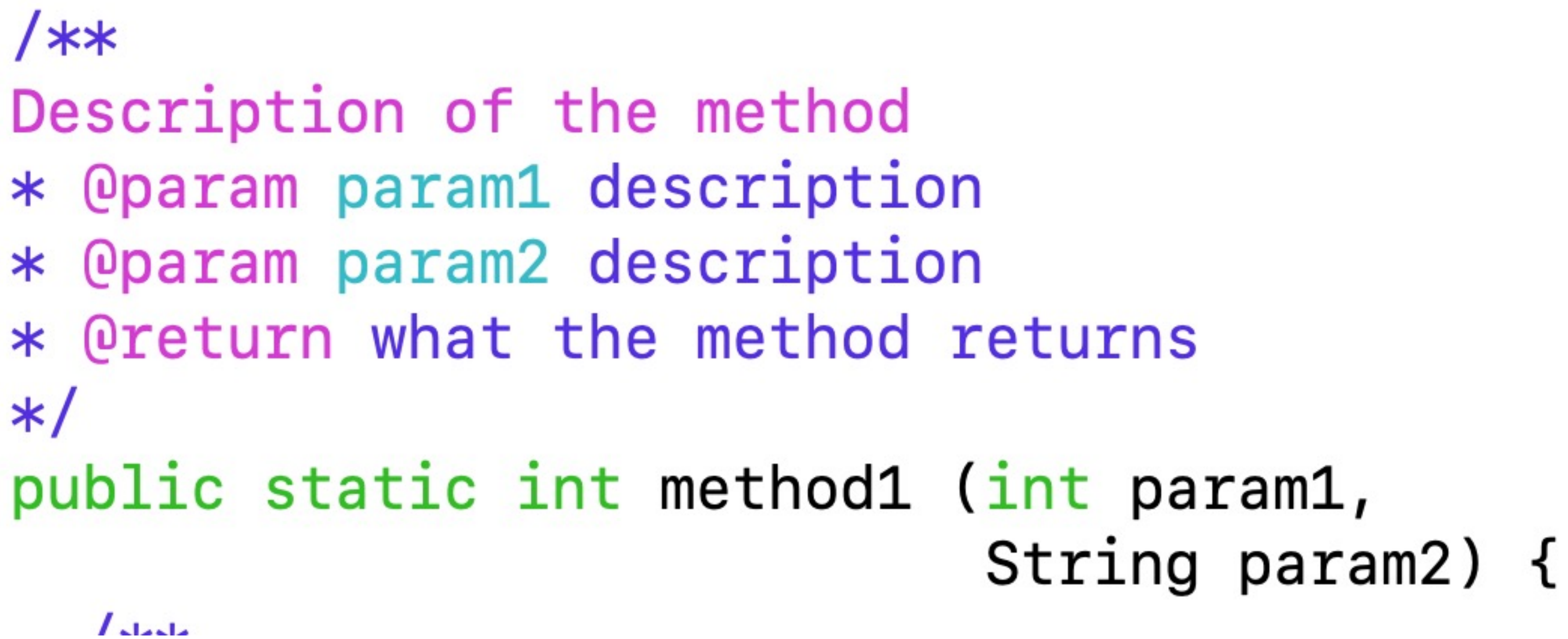


```
public static int method1 (int param1,
                           String param2) {
  /**
    body of the method
  */
  return 0;
}
```

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 methds 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 function: steps

1. When you encounter a function, pause!
2. Create a *frame* to hold the function's state
3. Copy argument values
4. Execute the function, 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 function's frame
7. Resume original function

```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);
}
```

Warning: don’t confuse printing with returning

```
// 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);
}
```

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

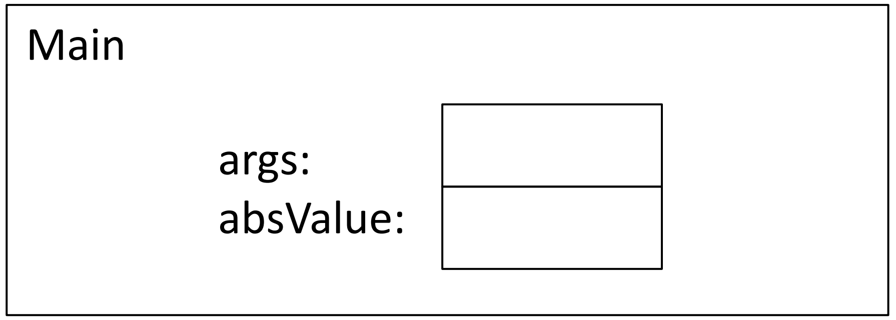
- Abstract details so user doesn’t need to worry about details

Exercise: Draw stack diagram

```java
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(-3.4);
    }
}
```

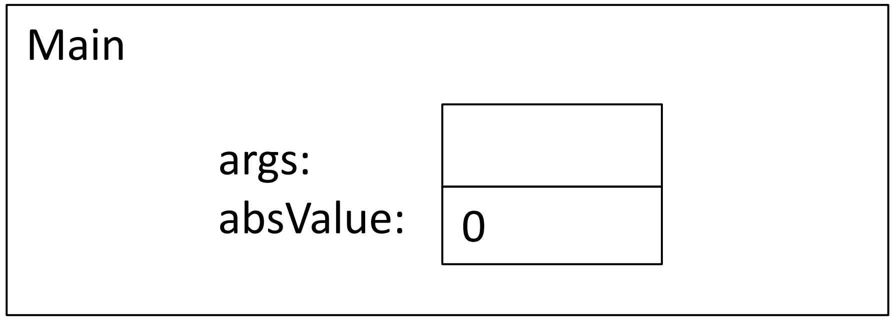
Exercise: Draw stack stack

```
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(-3.4);
    }
}
```



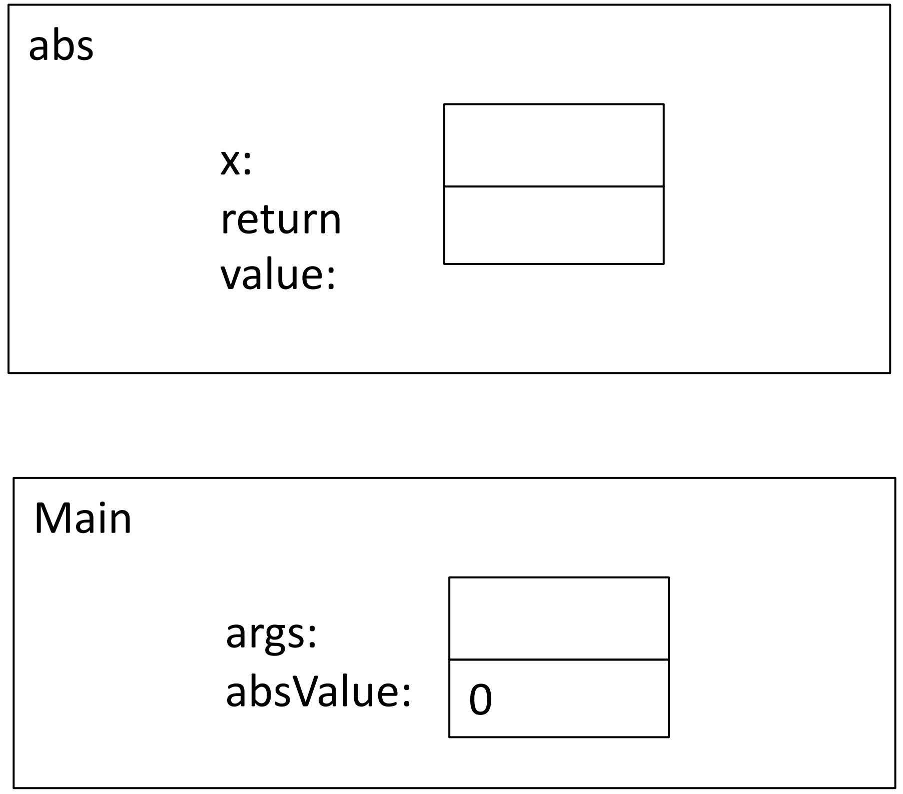
Exercise: Draw stack diagram

```
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(-3.4);
    }
}
```



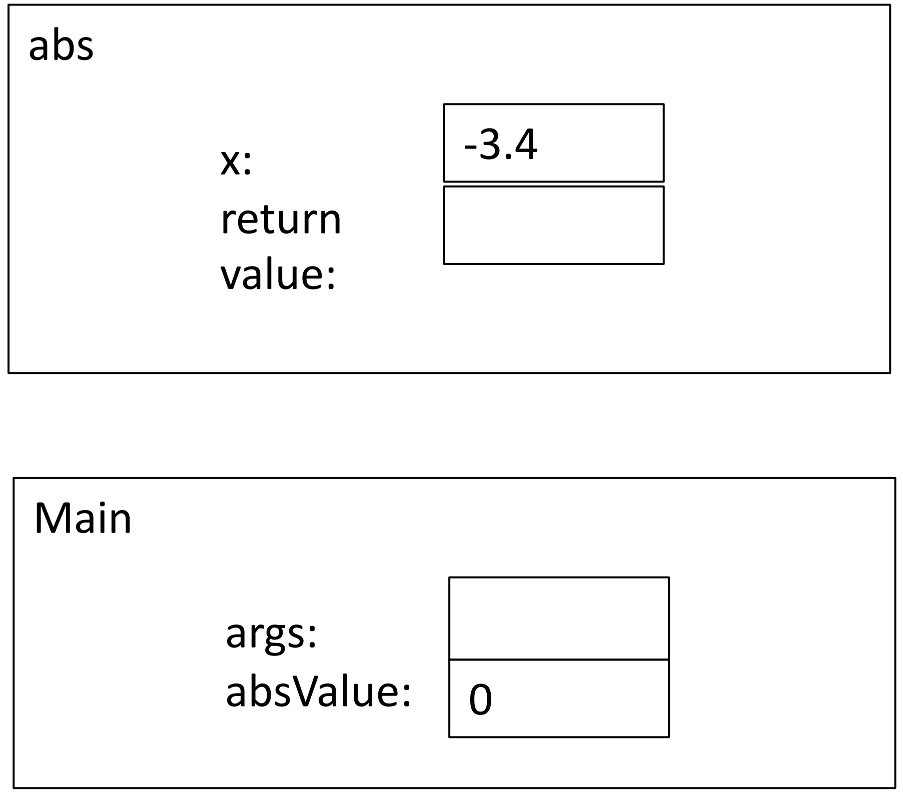
Exercise: Draw stack diagram

```java
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(-3.4);
    }
}
```



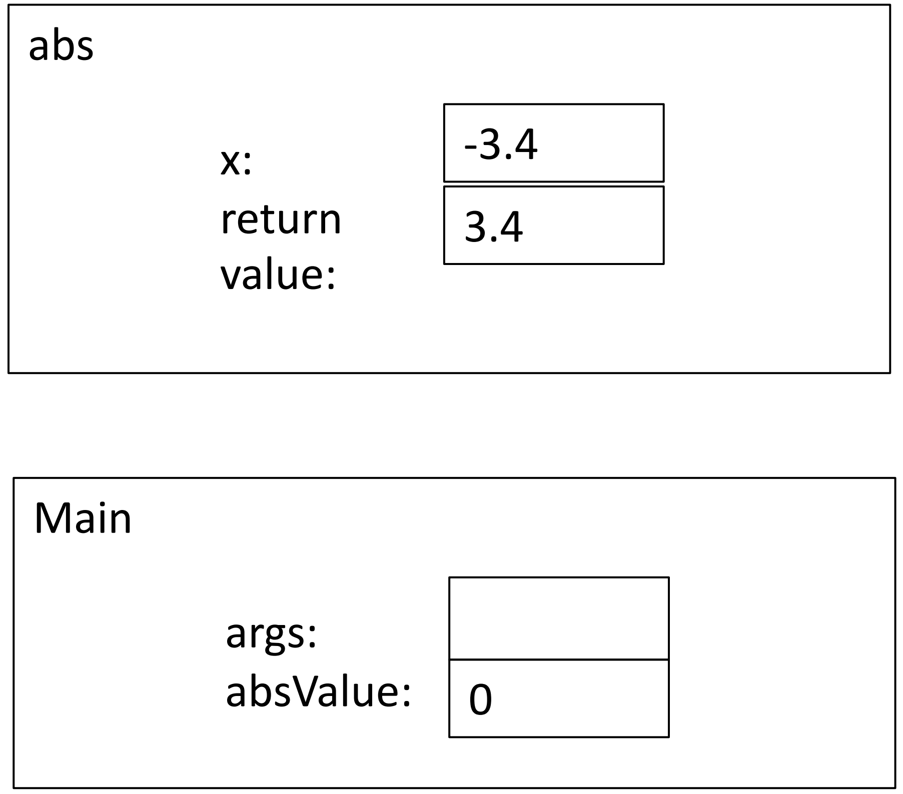
Exercise: Draw stack diagram

```
public class Abs {

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

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



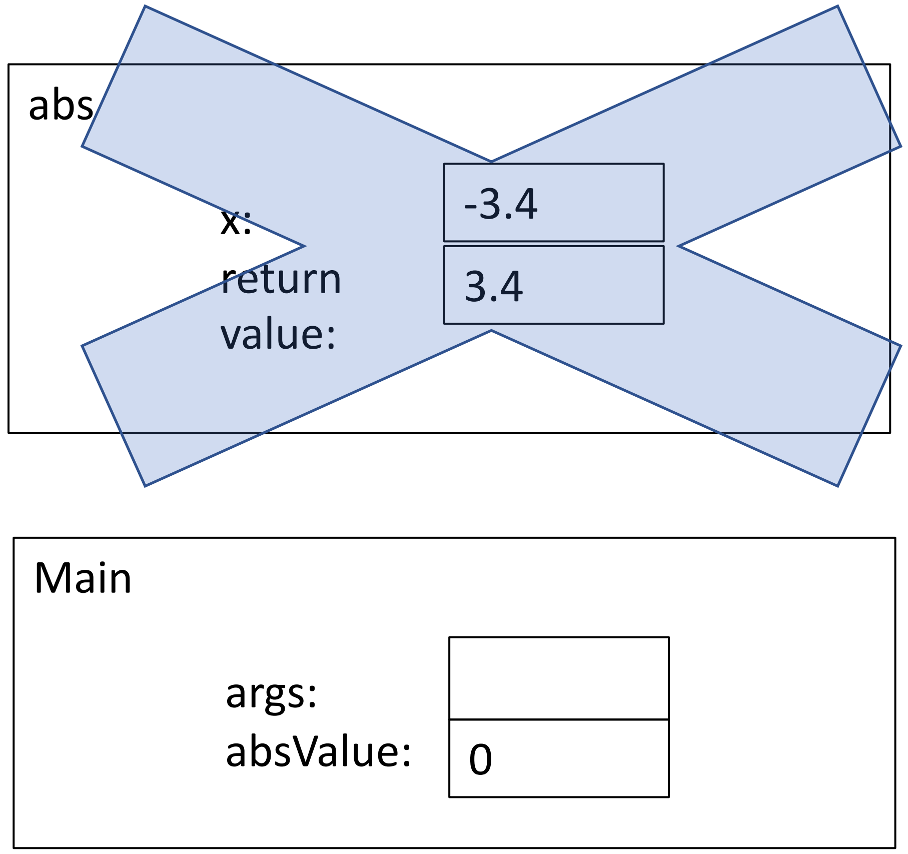
Exercise: Draw stack diagram

```
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(-3.4);
    }
}
```



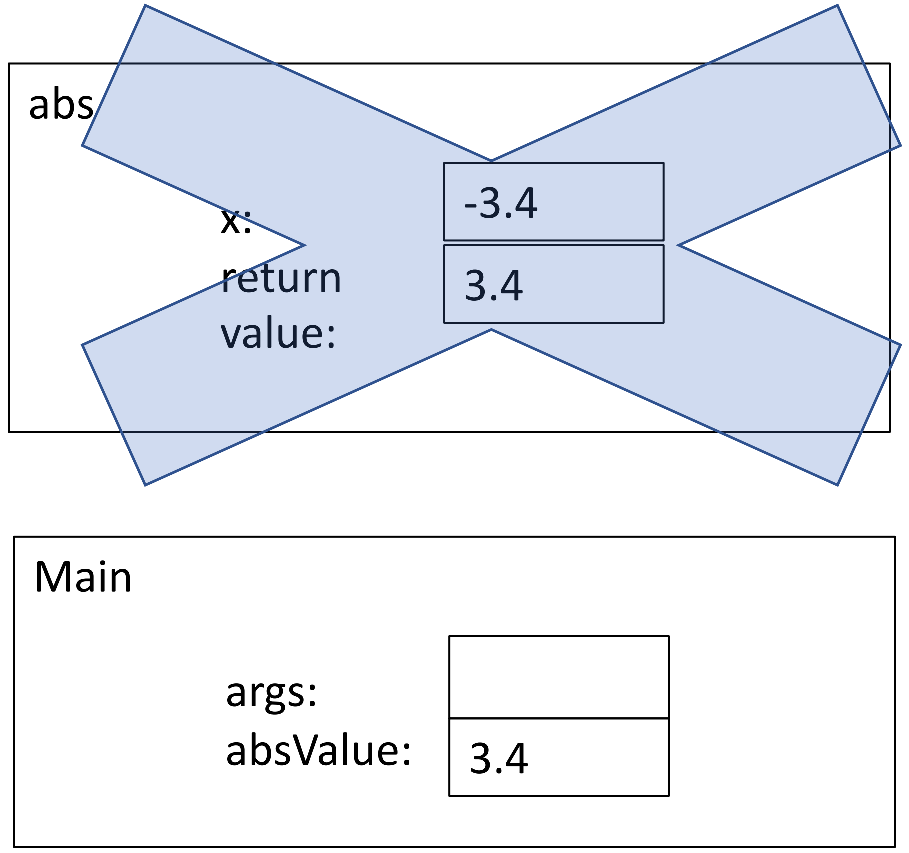
Exercise: Draw stack diagram

```
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(-3.4);
    }
}
```



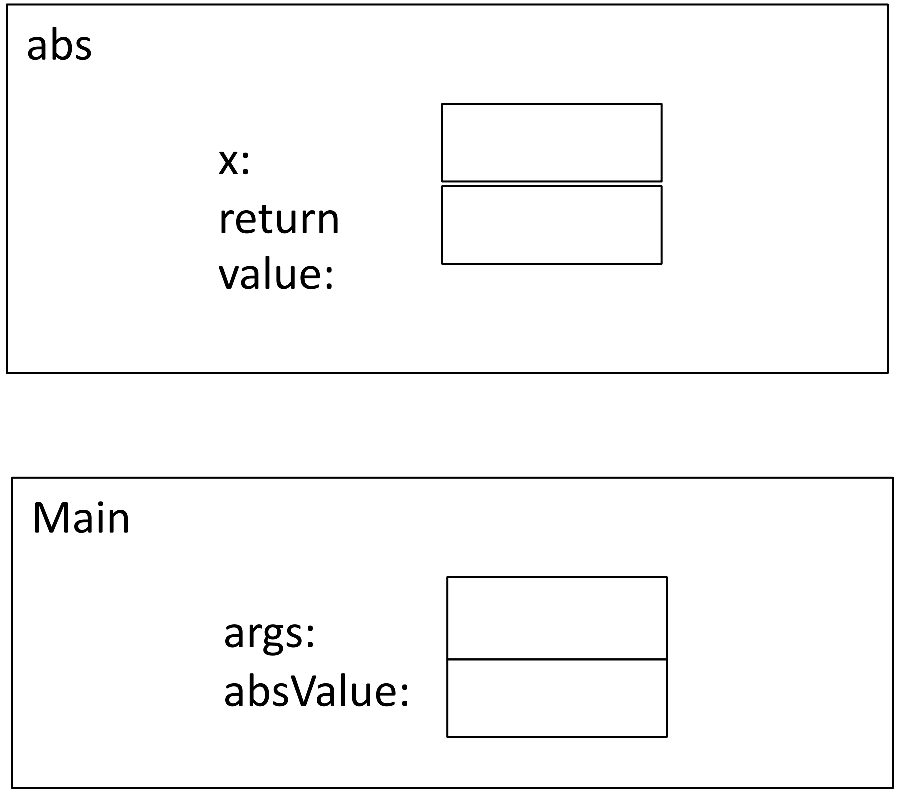
Exercise: Draw stack diagram

```
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(-3.4);

}
```



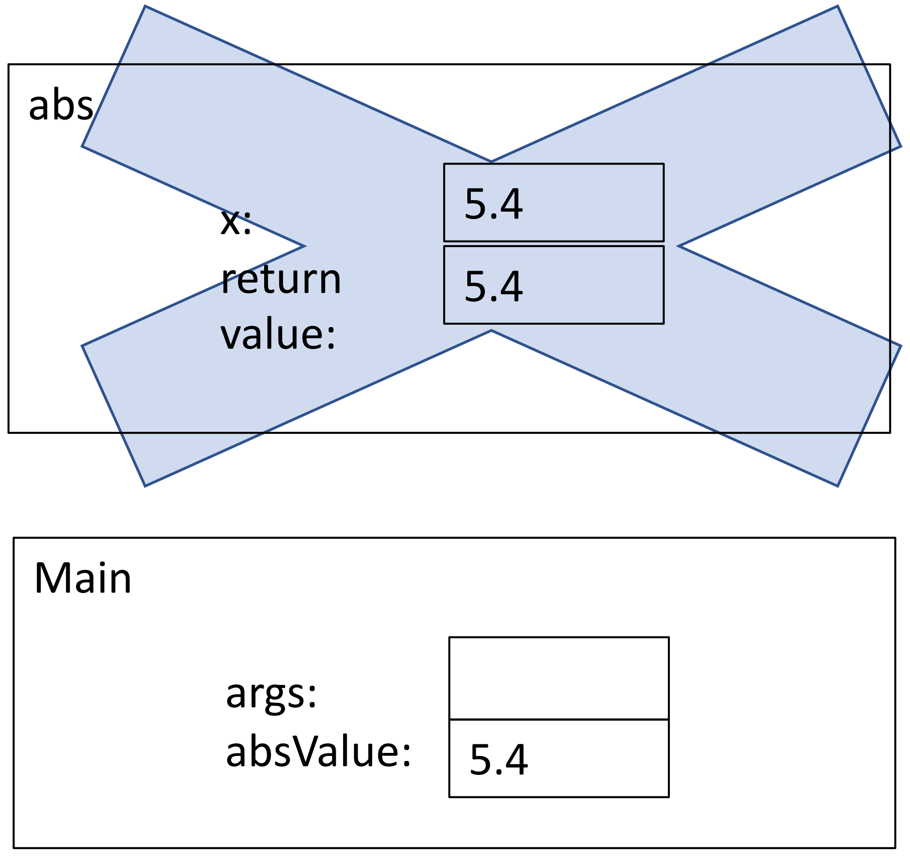
Exercise: Draw stack diagram

```
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);
    }
}
```



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);
    }
}
```



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

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
```