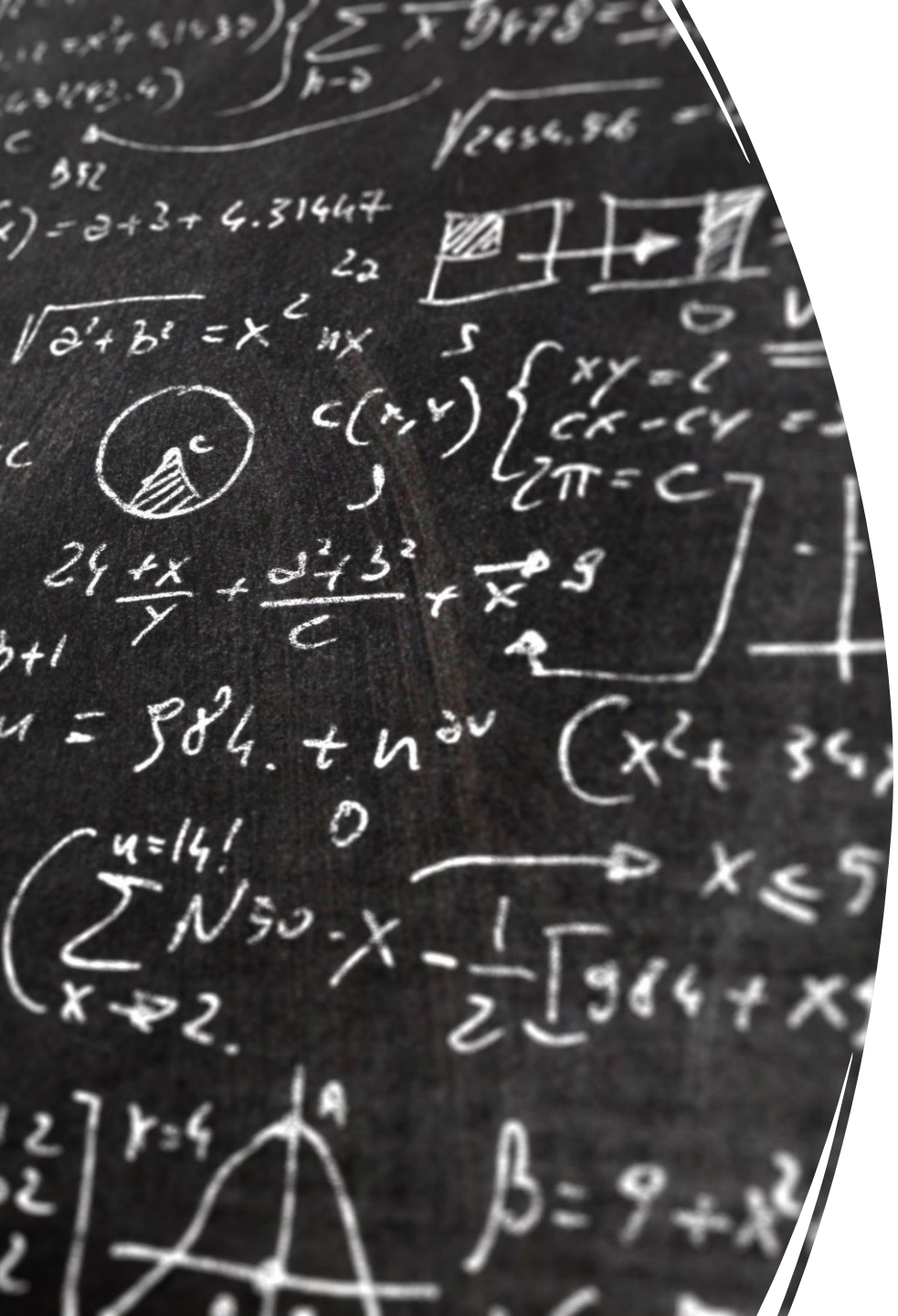


CS 113 – Computer Science I

Lecture 08 – Functions

Adam Poliak

09/29/2022

Announcements

- Assignment 03
  - Due tonight - Thursday 09/29

- Office hours:
  - Next week: cancelling Wednesday, will post updated time

Agenda

- Announcements
- Homework comment
- Functions

Common Homework Mistakes

- Incorrectly naming files

- Command line args vs console input

- Style (indentation)

Exercise: Contains

Define a function, called contains

- Input: phrase (String)
- Input: c (char)
- Return (Boolean): return true if the phrase contains the character c; false otherwise

Example usage:

```
public static void main(String[] args) {
    boolean result = contains(“lolcats”, ‘a’);
    System.out.println(result); // should print true

    result = contains(“lolcats”, ‘’);
    System.out.println(result); // should print false
}
```

Functions

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

Inputs (aka *parameters* or *arguments*) generalize the function's execution

Outputs pass results of the function back to the caller

Functions in Java

Java defines two types of functions:

- static methods
- instance/member methods

Focus on **static methods** now

All functions should be contained inside a class

Anatomy of a function

- All functions have the following things:
  - Name
  - Arguments/parameters/inputs
  - Body
  - Return Type

```
public static void name(Arguments) {
    <BODY OF FUNCTION>
}
```

Defining functions in Java: syntax

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

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

Calling functions in Java: syntax

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

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);
}
```

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

```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 functions

- 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: Contains

`wget https://raw.githubusercontent.com/BrynMawr-CS113-F22/class-examples-poliak/main/week04/Contains.java`

Define a function, called contains

- Input: phrase (String)
- Input: c (char)
- Return (Boolean): return true if the phrase contains the character c; false otherwise

Example usage:

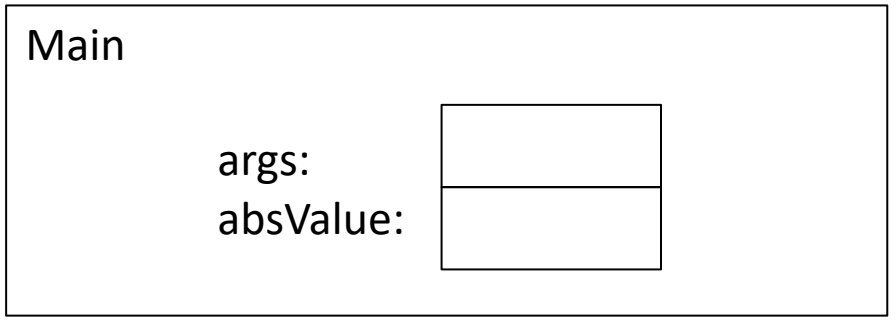
```
public static void main(String[] args) {
    boolean result = contains(“lolcats”, ‘a’);
    System.out.println(result); // should print true

    result = contains(“lolcats”, ‘’);
    System.out.println(result); // should print false
}
```

Exercise: Draw function 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);
    }
}
```

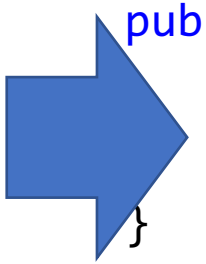
Exercise: Draw function stack

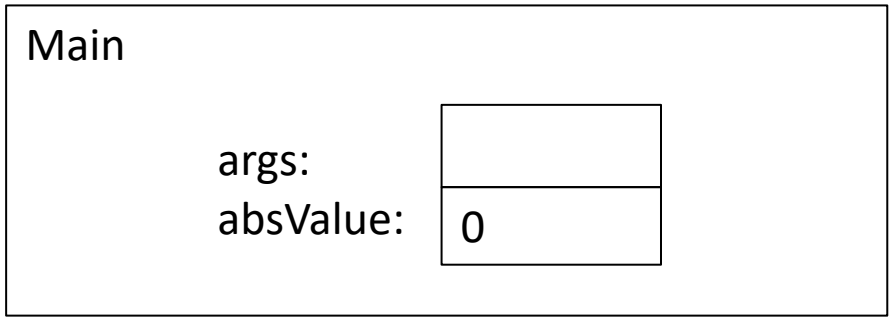
```
public class Abs {

    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 function stack

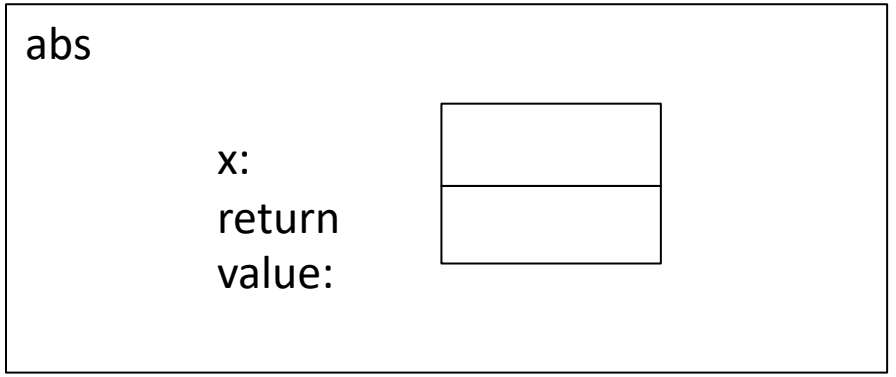
```
public class Abs {

    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);
    }
}
```

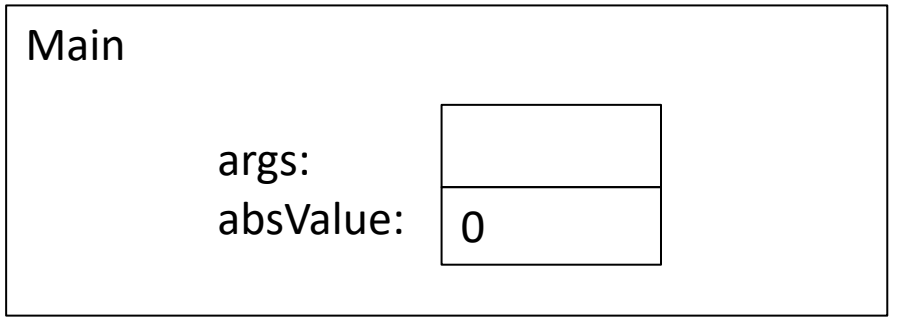




Exercise: Draw function stack

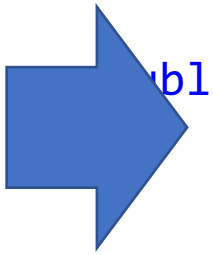
```
public class Abs {

    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);
    }
}
```

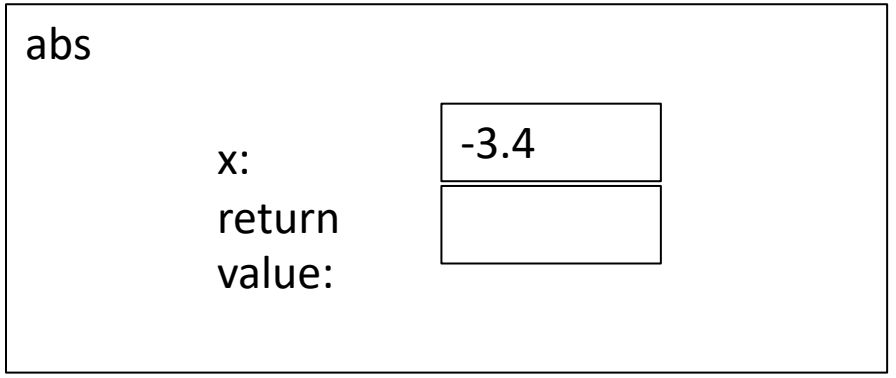


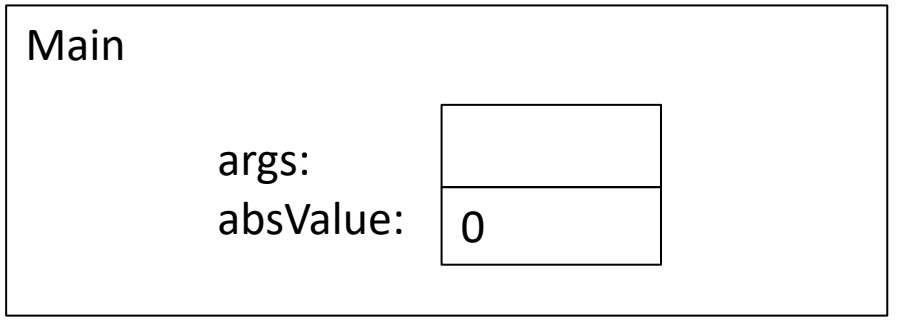


Exercise: Draw function stack

```
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);
    }
}
```

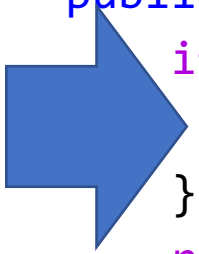


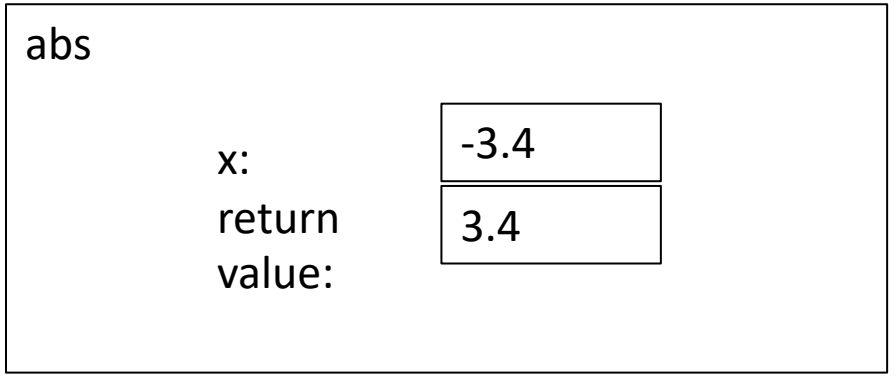


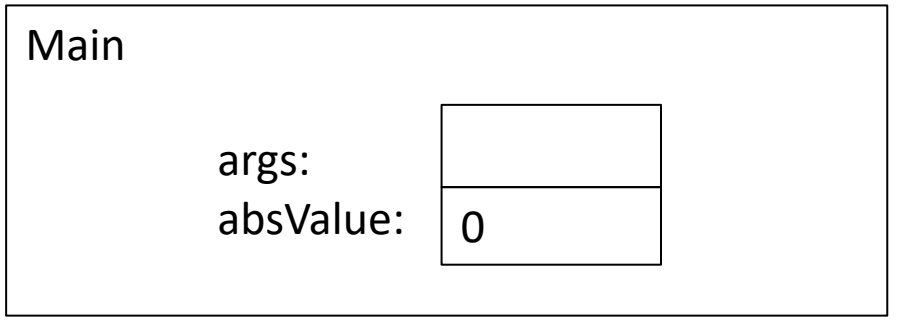


Exercise: Draw function stack

```
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);
    }
}
```

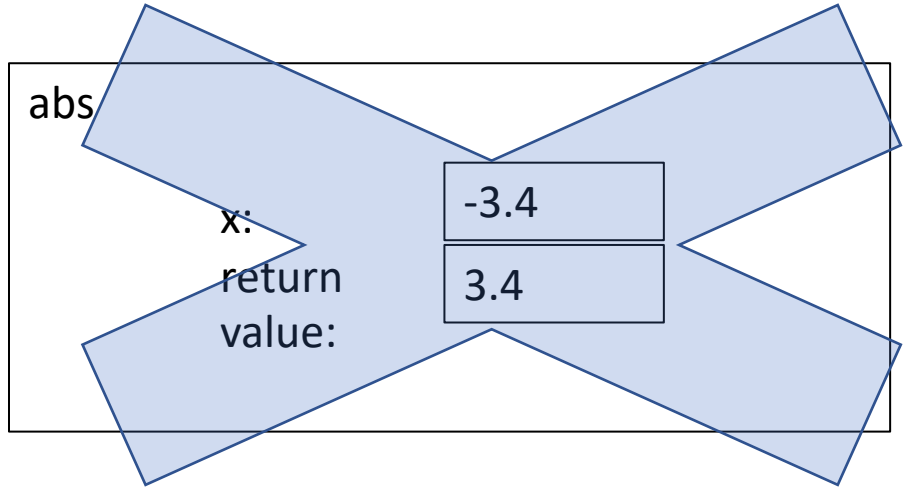


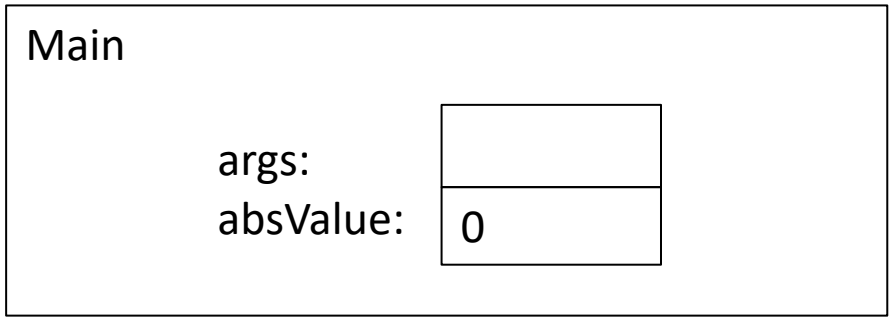




Exercise: Draw function stack

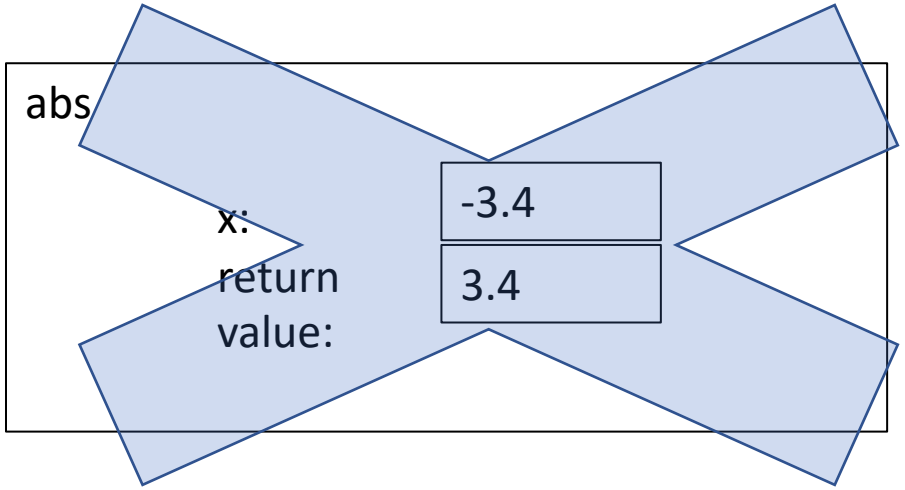
```
public class Abs {

    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);
    }
}
```

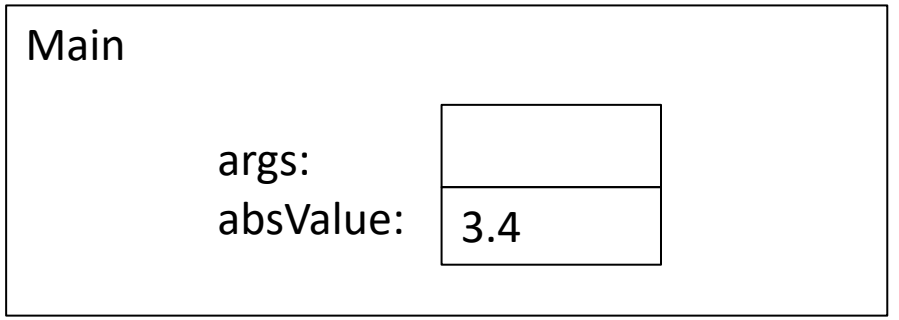




Exercise: Draw function stack

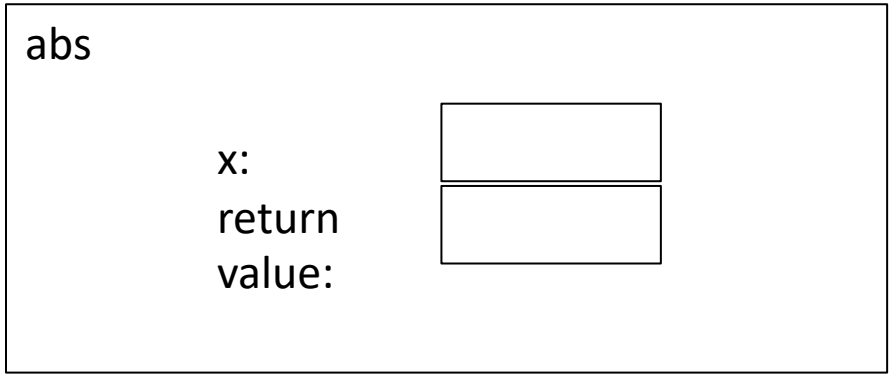
```
public class Abs {

    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);

}
```

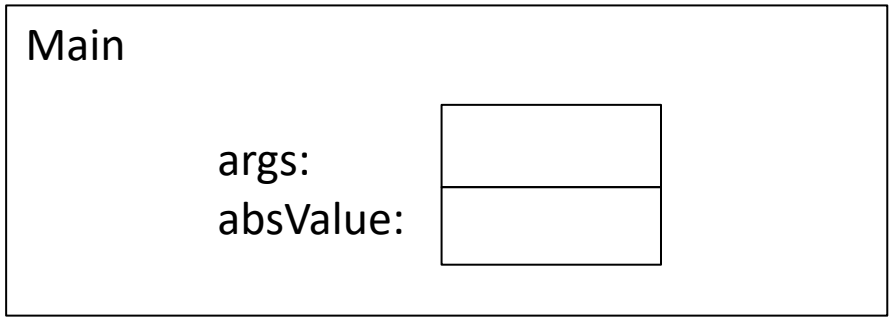




Exercise: Draw function stack

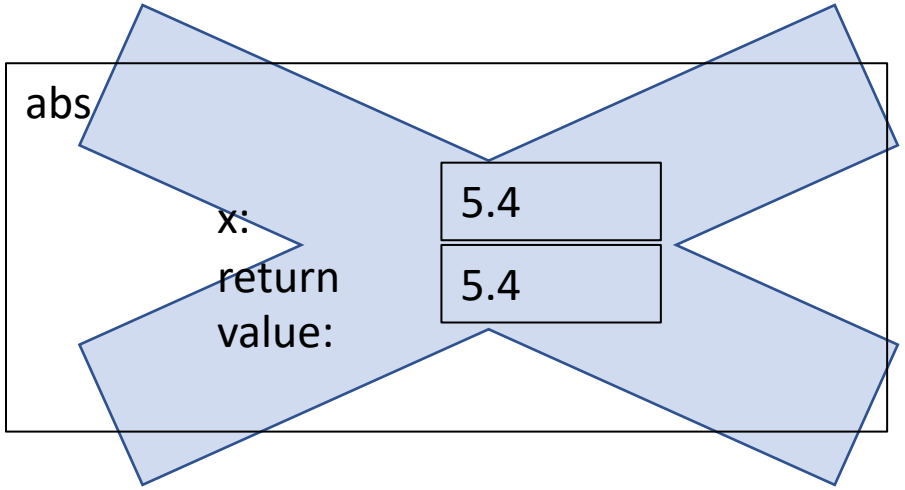
```
public class Abs {

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

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

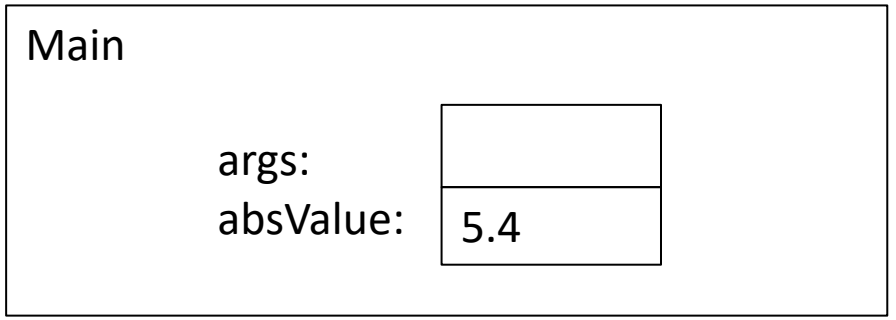




Exercise: Draw function stack

```
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(5.4);
    }
}
```





Function specifications

Idea: “contract” between the function user and the function 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 function makes that last after the function returns (e.g. printing to the console is a side effect)

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

Function 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 function specifications?

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

Function: 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