Java

IntelliJ, packages, types, methods

Hello World

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- The simplest Java example
  - .. it's still pretty complex
- Prints "Hello Java!" to the console
- Let's break this down

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- Package declaration
  - Matches the directory structure in the src (source) directory
  - This file is saved in the directory "src/week1"

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- Create a class
  - All Java code is defined in classes
  - We create a class named HelloWorld to contain our code

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- Filename matches the class name for that file
  - This file will be named "HelloWorld.java"
- Combining the filename and directory structure, the fulll path for this file is:
  - "src/week1/HelloWorld.java"

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- Main method
  - A method is a function defined within a class
  - The main method is a special method that controls the start of your program
  - Running a program is equivalent to calling the main method

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- static: the method can be called without extra steps
- void: the method does not return a value
- main: The main method must be named exactly "main"
- String[]: The main method takes Strings as a parameter

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- public: This code can be used from anywhere
- Mix the ingredients of public, static, void, main, String[] together and you have a main method in Java
- Confusing? This is why we don't use Java in CSE115

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- Or type main in IntelliJ and it creates a main method for you

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- We're finally ready to write some code!
  - System.out.println is a method that prints to the console
    - Or type sout in IntelliJ

Java - Hello World

```java
package week1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

- All Java statements must end with a semicolon
- Don't forget your semicolons

Variables and Methods

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(int val){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num2 = 2.4;
        num2 = multiplyByTwo(num2);
        System.out.println("new num2:"+num2);
        why(2);
    }

}
```

- This program defines several variables and methods in Java
- We'll focus on the Java specific details
- We'll step through the code starting in the main method

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- **Java is strongly typed!**
- Whenever you declare a variable, you must specify the type of that variable
- That variable can **only** store values of that type
- The variable "num" can only store a double
  - `num = "hello";` would cause an error

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- Declare a variable of type double
- Name the variable num
- Assign num the value 2.4

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- Next, we call a method named multiplyByTwo
- Let's talk about method declaration in Java!

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- Just like the main method, this method will be:
  - public: This method can be called from anywhere in the project
  - static: This method can be called without additional setup
- For the first few weeks, all of our methods will be public static methods

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- Next, we **must** specify the return type of the method
- This method has a return type of double so it **will** return a double
  - This is a contract you sign with Java saying you guarantee that you'll return a value of type of double
  - If you break this contract, Java will refuse to run your code

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- Since Java is strongly typed, you must specify the type of every parameter
- This method takes an input of type double
  - It can only be called with arguments of type double

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- All code between the braces are part of the body of the method
- Indentation does not matter (As opposed to Python)
- User "return" to return a value matching the return type of the method

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num: " + num);
        why();
    }

}
```

- Now we call the method we defined
- num is reassigned to the value 4.8
- print "new num: 4.8"

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- Some method don't return a value
- Give these methods a return type of "void"
- Methods that return void are called for their "side-effects" which is any functionality aside from the return value
  - This method has the side-effect of printing text

Java - Variables and Methods

```java
package week1;

public class VariablesFunctions {

    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }

    public static void why(){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        double num = 2.4;
        num = multiplyByTwo(num);
        System.out.println("new num:" + num);
        why();
    }

}
```

- When the end of the main method is reached
- The program ends

Types

Java - Types

```
package week1;

public class VariablesFunctions {

    public static void main(String[] args) {
        double num1 = 2.4;
        String str1 = "A string";
        int num2 = 6/4;
        boolean bool = true;
    }

}
```

- Java is strongly typed
- Let's talk about the common types we'll use to start the semester

Java - Types

```java
package week1;

public class VariablesFunctions {

    public static void main(String[] args) {
        double num1 = 2.4;
        String str1 = "A string";
        int num2 = 6/4;
        boolean bool = true;
    }

}
```

- double
  - A number that can have a decimal portion
  - We call this a floating point number
  - doubles cannot always be represented exactly in a computer (Foreshadow!)

Java - Types

```java
package week1;

public class VariablesFunctions {

    public static void main(String[] args) {
        double num1 = 2.4;
        String str1 = "A string";
        int num2 = 6/4;
        boolean bool = true;
    }

}
```

- String
  - A sequence of characters (char)
  - Note the capital S

Java - Types

```java
package week1;

public class VariablesFunctions {

    public static void main(String[] args) {
        double num1 = 2.4;
        String str1 = "A string";
        int num2 = 6/4;
        boolean bool = true;
    }

}
```

- int
  - A whole number
  - Can be negative
  - Dividing 2 ints will result in int!
    - Called integer division
    - 6/4 == 1
    - Result is always rounded down (Floor)
    - 99/100 == 0

Java - Types

```java
package week1;

public class VariablesFunctions {

    public static void main(String[] args) {
        double num1 = 2.4;
        String str1 = "A string";
        int num2 = 6/4;
        boolean bool = true;
    }

}
```

- boolean
  - true or false
  - Note the lowercase t and f

Memory Diagram!

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x=input*2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }

    public static void main(String[] args) {
        System.out.println("I can print!");
        int num=4;
        double num2=2.4;
        String str1="A string";
        int num3=6/4;
        boolean bool=true;
        System.out.println("I an a num3: "+num3);

        num2=multiplyByTwo(num2);
        System.out.println("new num2:"+num2);
        why(2);
    }
}
```

- Let's look at all the code together in one program
- We'll trace through this code in a memory diagram
  - [The same way you will on lab quizzes]

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

- We'll trace through this code using a memory diagram
- We keep track of everything stored in the computer's memory while the program runs

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

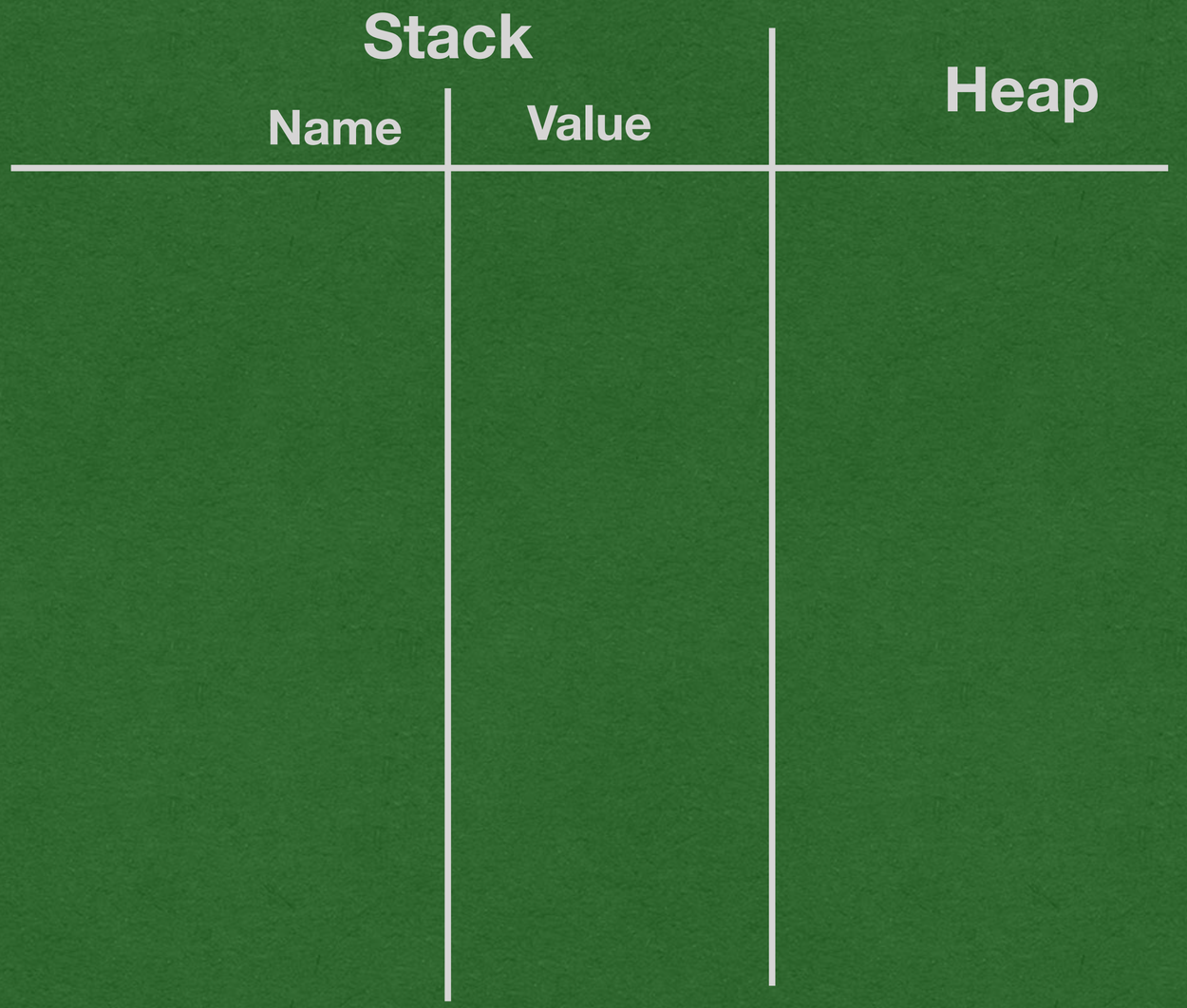
Name	Value

- We setup space for the "stack" memory which stored variable
- We separate the stack into two column for the name and value of each variable

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

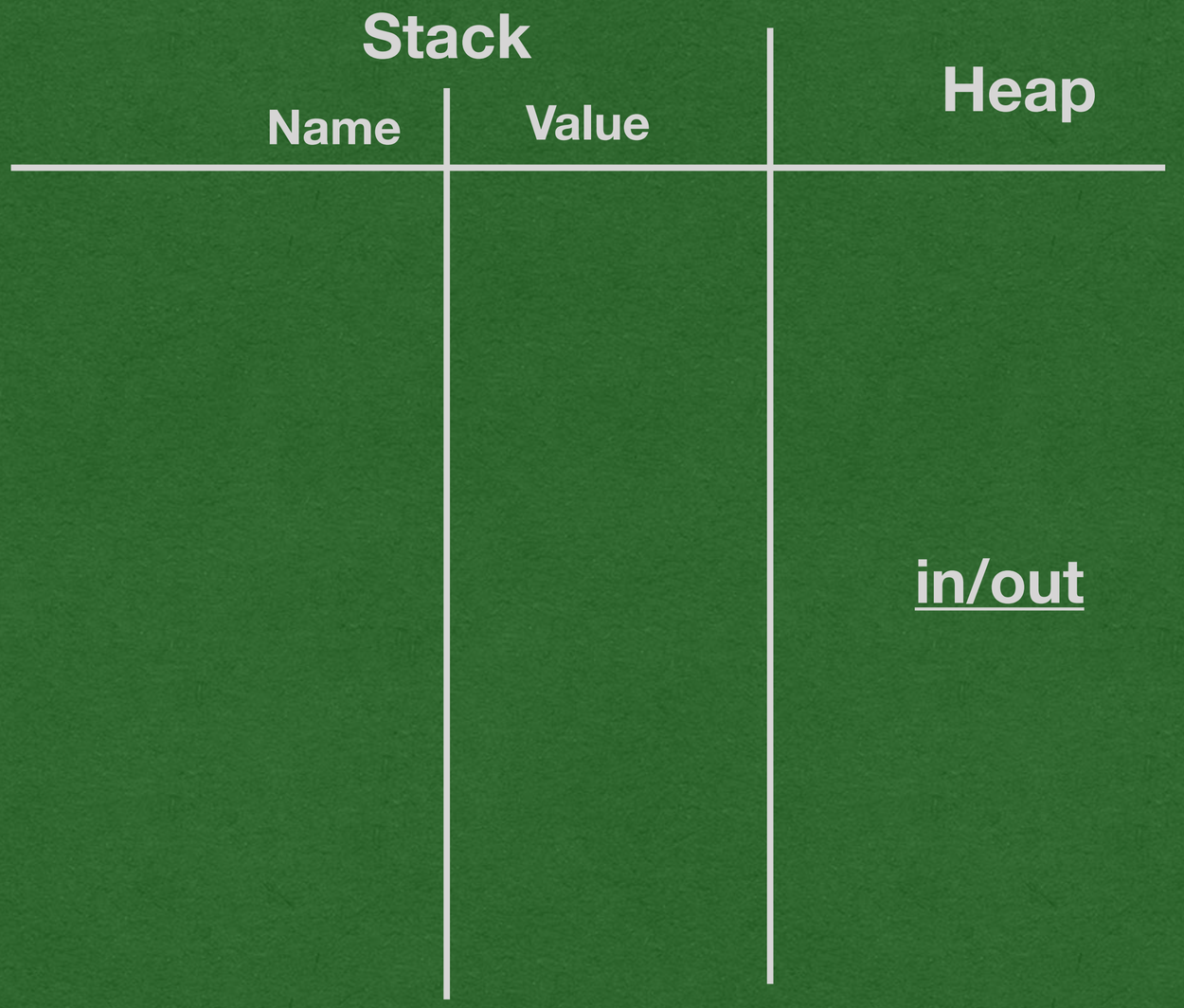


- Heap Memory: A wonderful topic.. for another day

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

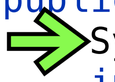


- Add an in/out section wherever you have room
- This is where you put everything that's printed to the console

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```



Stack		Heap
Name	Value	
		in/out
		I can print!

- Now we're ready to start tracing through the code and see how this program works in memory
- First, we print "I can print!"

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

Name	Value
num	4

Heap

in/out

I can print!

- When variables are declared, add then to the stack with their value
- We don't add the variable type in our memory diagrams

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

Name	Value
num	4
num2	2.4
str1	"A string"

Heap

in/out

I can print!

- Add the next 2 variables to the stack
- Newer variables are added to the bottom of the stack

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

Name	Value
num	4
num2	2.4
str1	"A string"
num3	1

Heap

in/out

I can print!

- Don't forget about integer division
- When dividing 2 ints, the result will not have a decimal portion

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

Name	Value
num	4
num2	2.4
str1	"A string"
num3	1
bool	true

Heap

in/out

I can print!
I am num3: 1

- Declare another variable
- Print to the console again

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

Name	Value
num	4
num2	2.4
str1	"A string"
num3	1
bool	true

Heap

in/out

I can print!
I am num3: 1

- When a method is called, a "stack frame" is added to the stack
- A stack frame is an isolated part of the stack just for the method call

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

Stack

Name	Value
num	4
num2	2.4
str1	"A string"
num3	1
bool	true

Heap

in/out

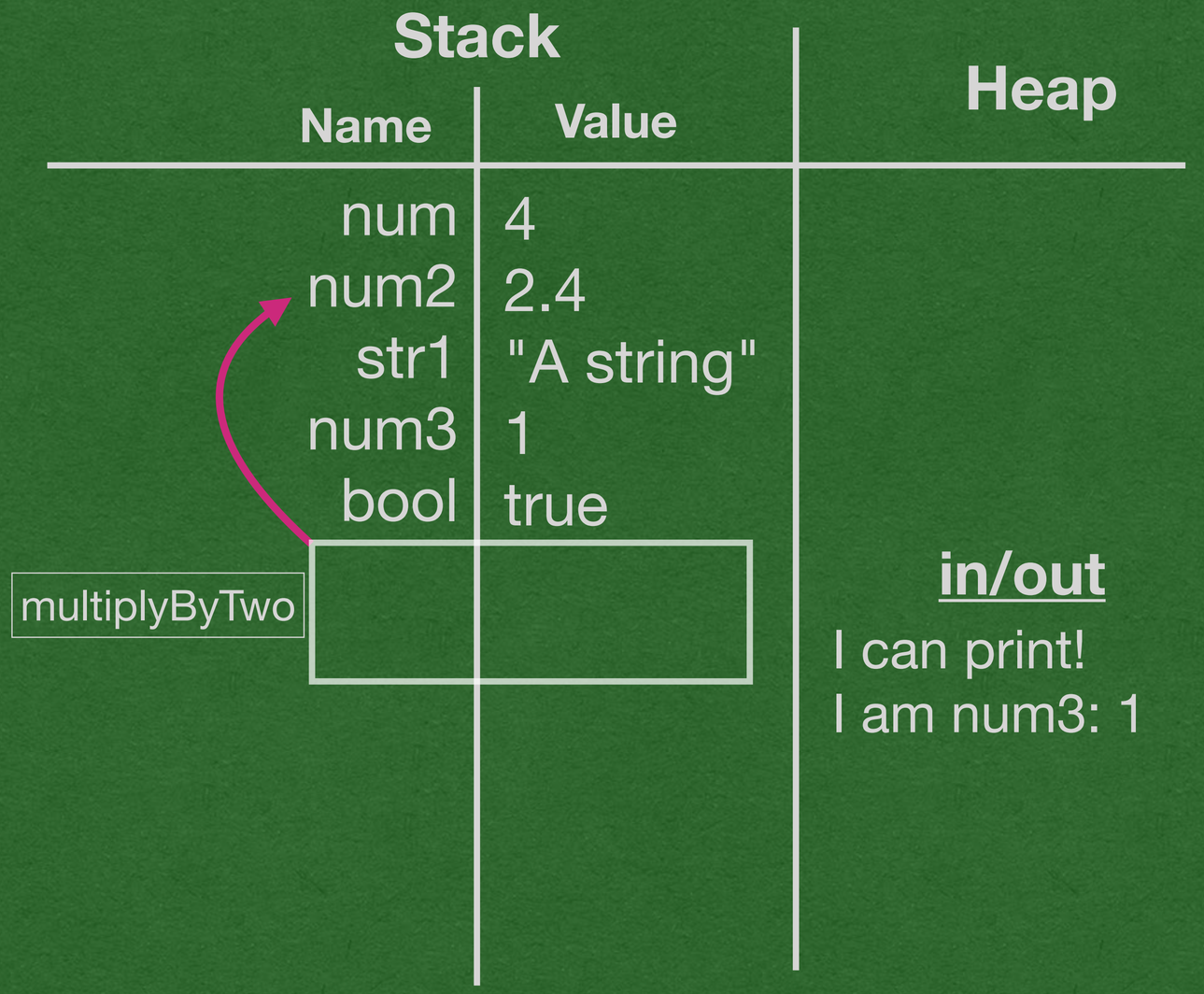
I can print!
I am num3: 1

- We show stack frames as solid boxes
- You cannot break out of this part of the stack until the method returns
- Variables on the stack that are not inside the stack frame cannot be accessed

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

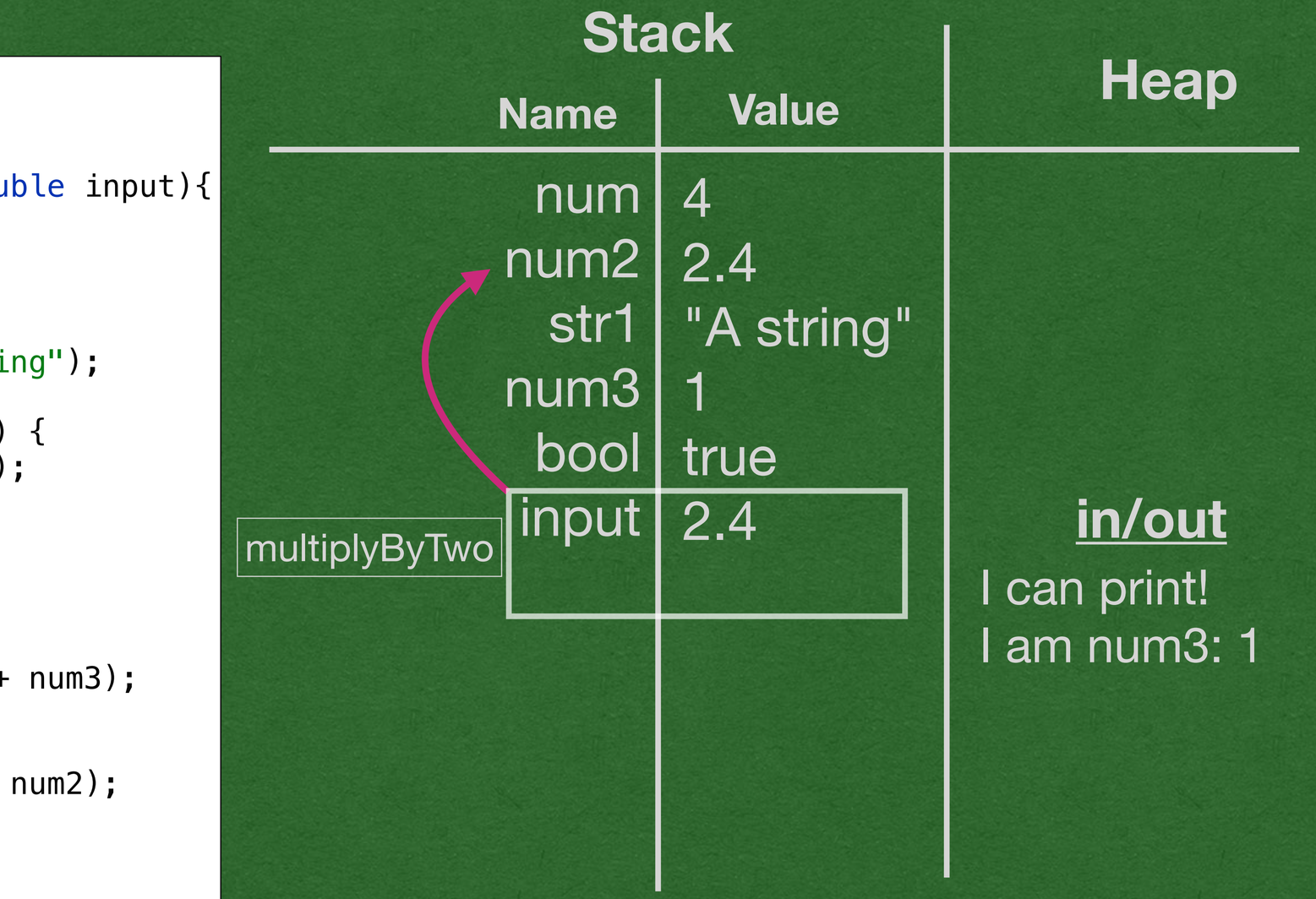


- We add the name of the method called next to the stack frame
- Add an arrow to denote where the return value will be assigned

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

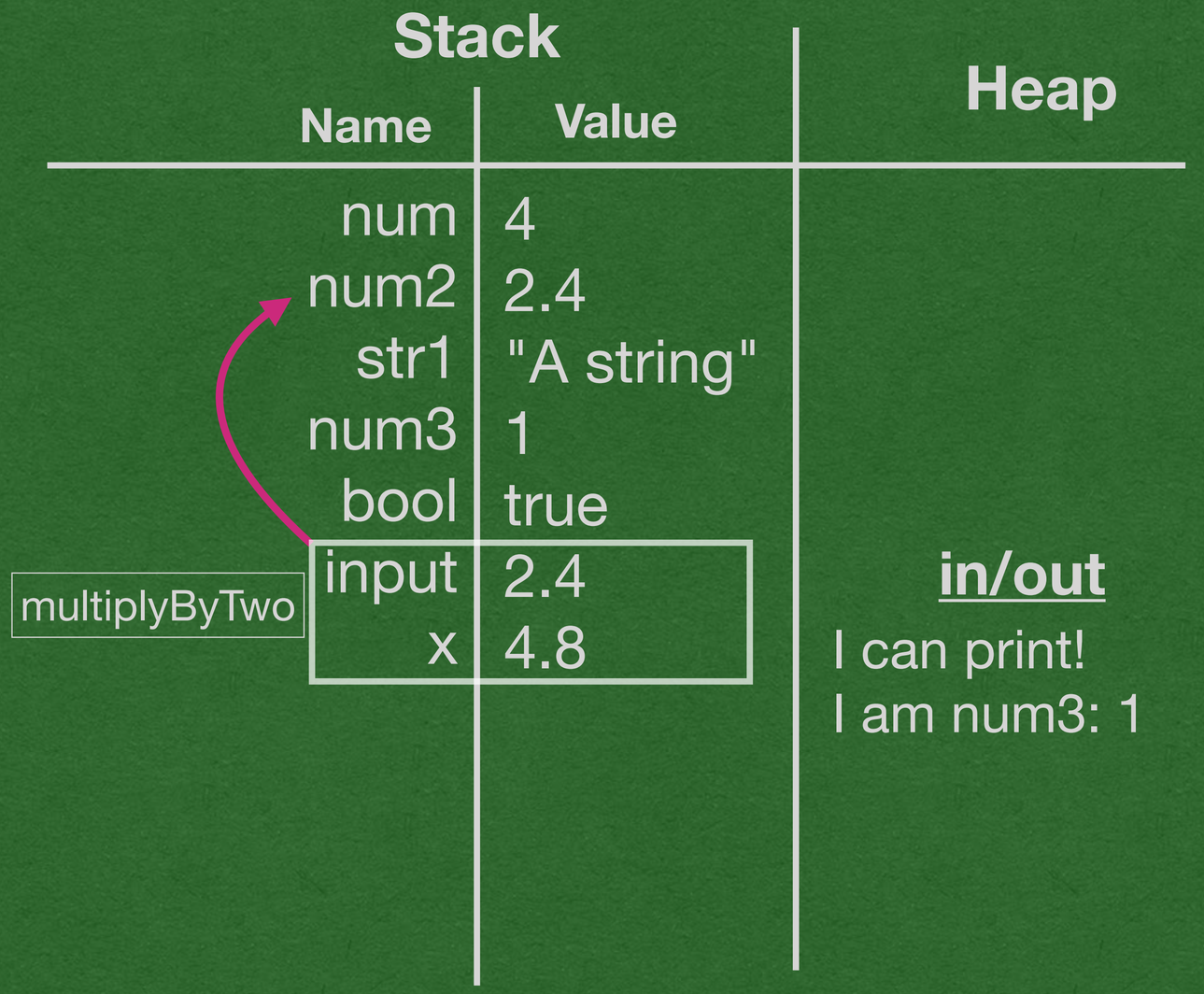


- Finally, we add each parameter (only 1 in this example) to the stack and assign them the values of the arguments
- We're now ready to run the code of the method using only the variable inside the stack frame

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

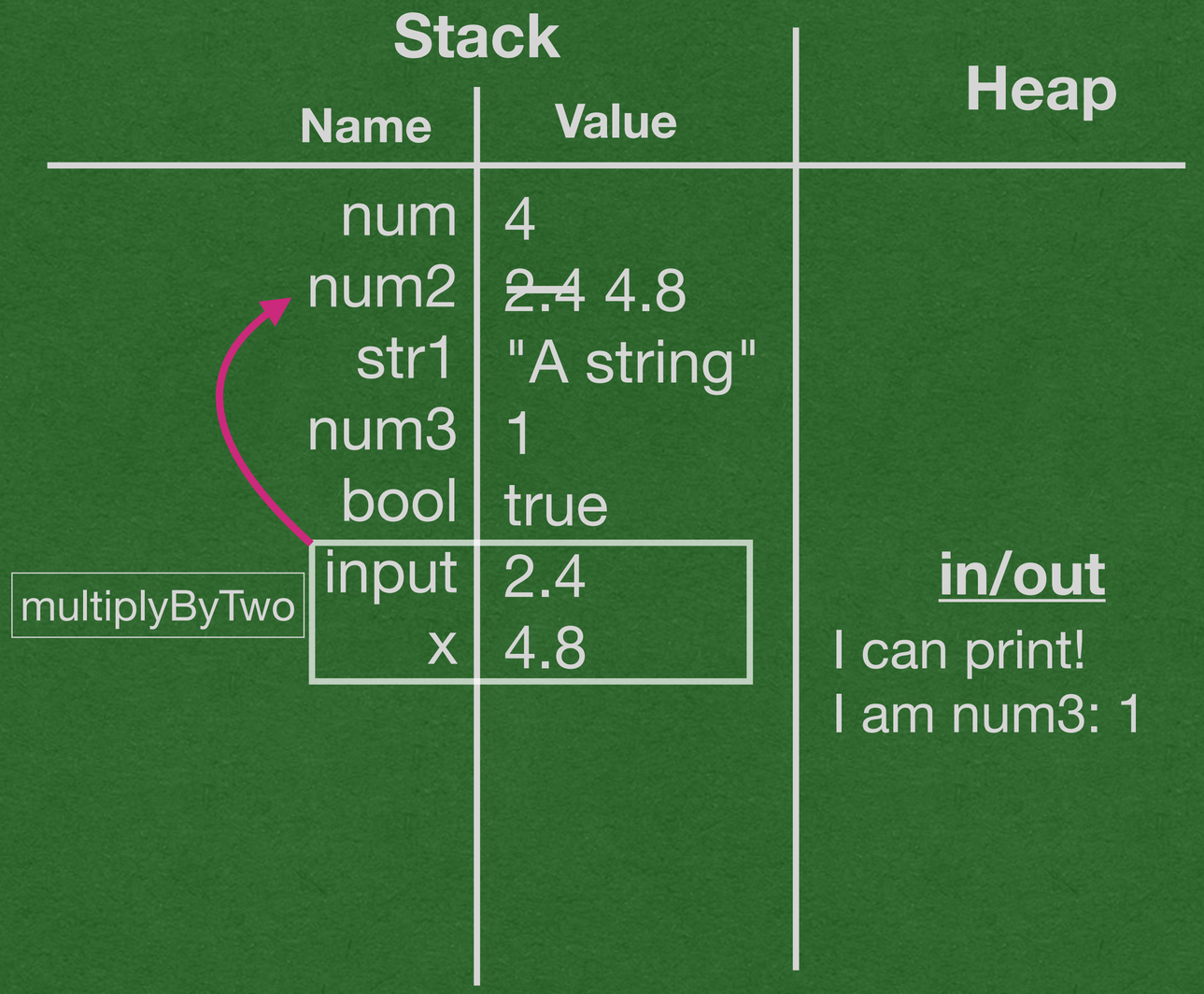


- New variables are declared inside the stack frame

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

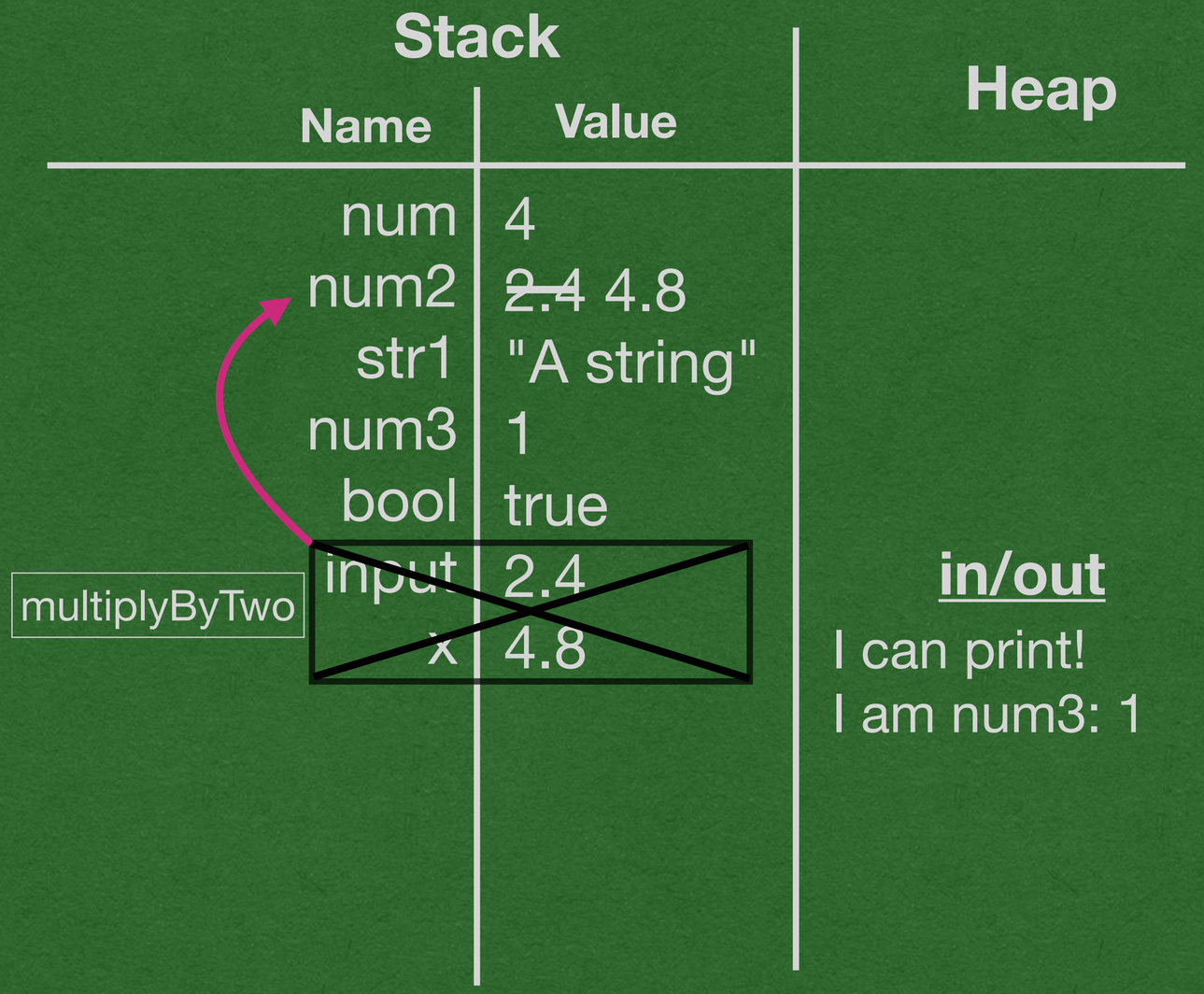


- When a value is returned, assign it to the value following the return arrow
- To update a value, cross out the old value and write the new value

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

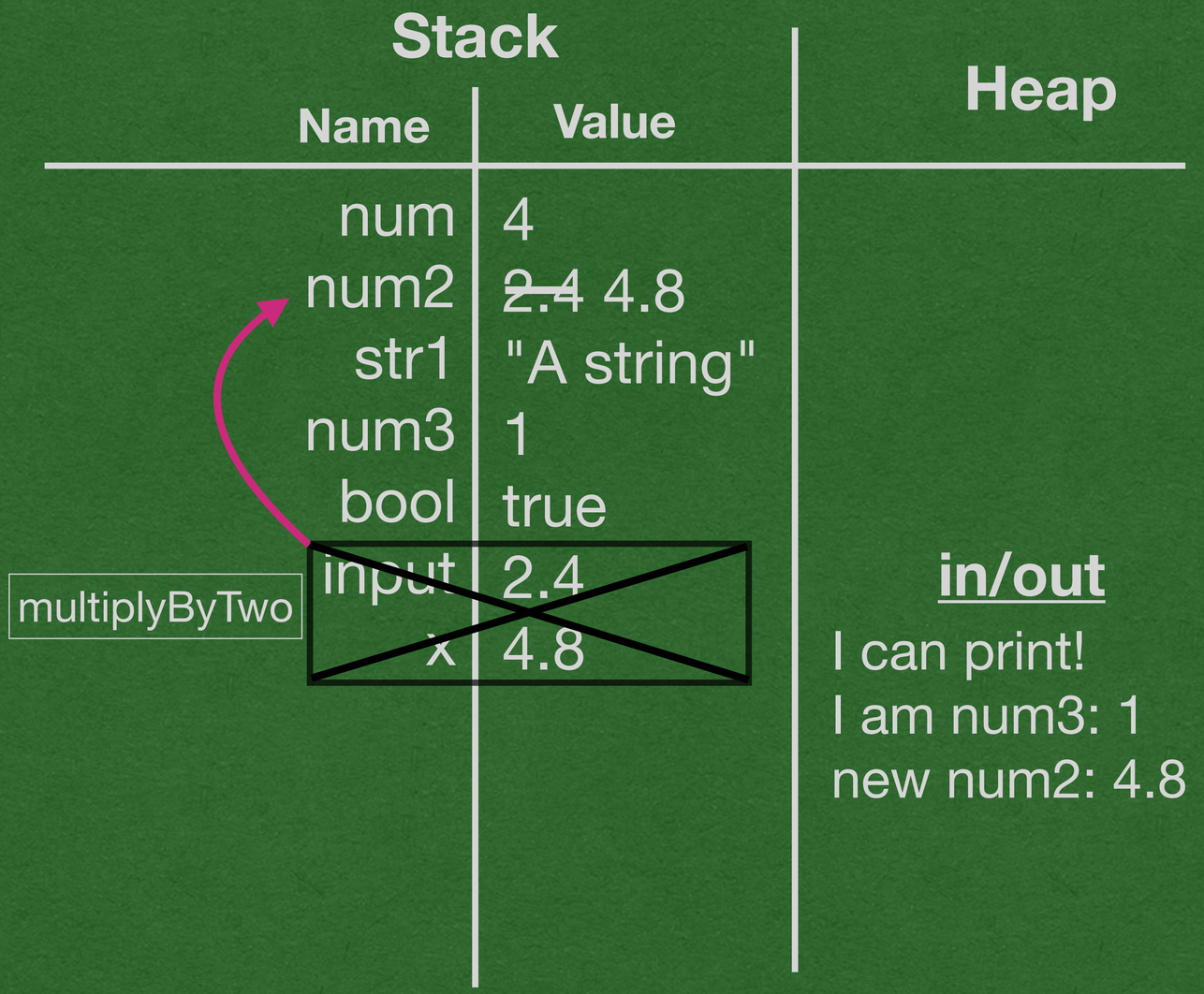


- Cross out the entire stack frame when the method returns
- The variables in the stack frame are erased from memory

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

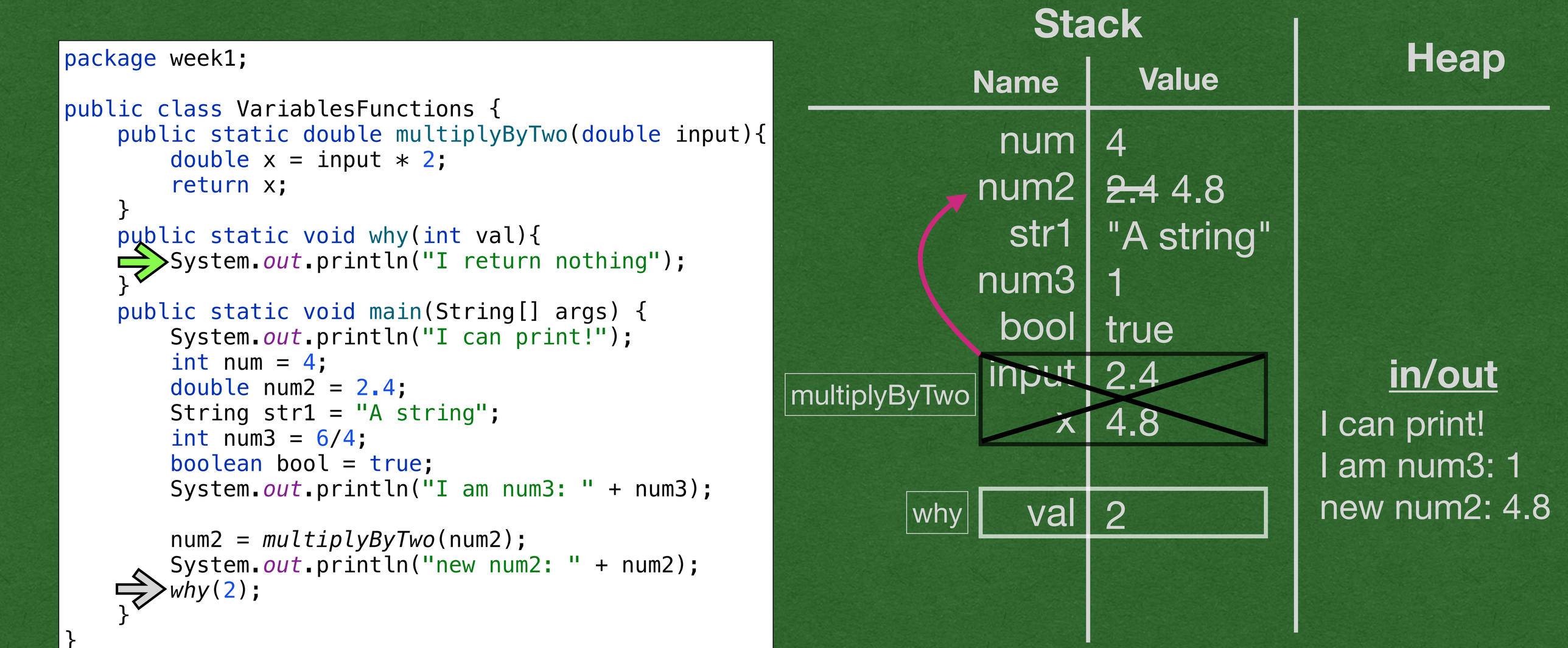


- Control returns to the main method
- Continue with the program

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

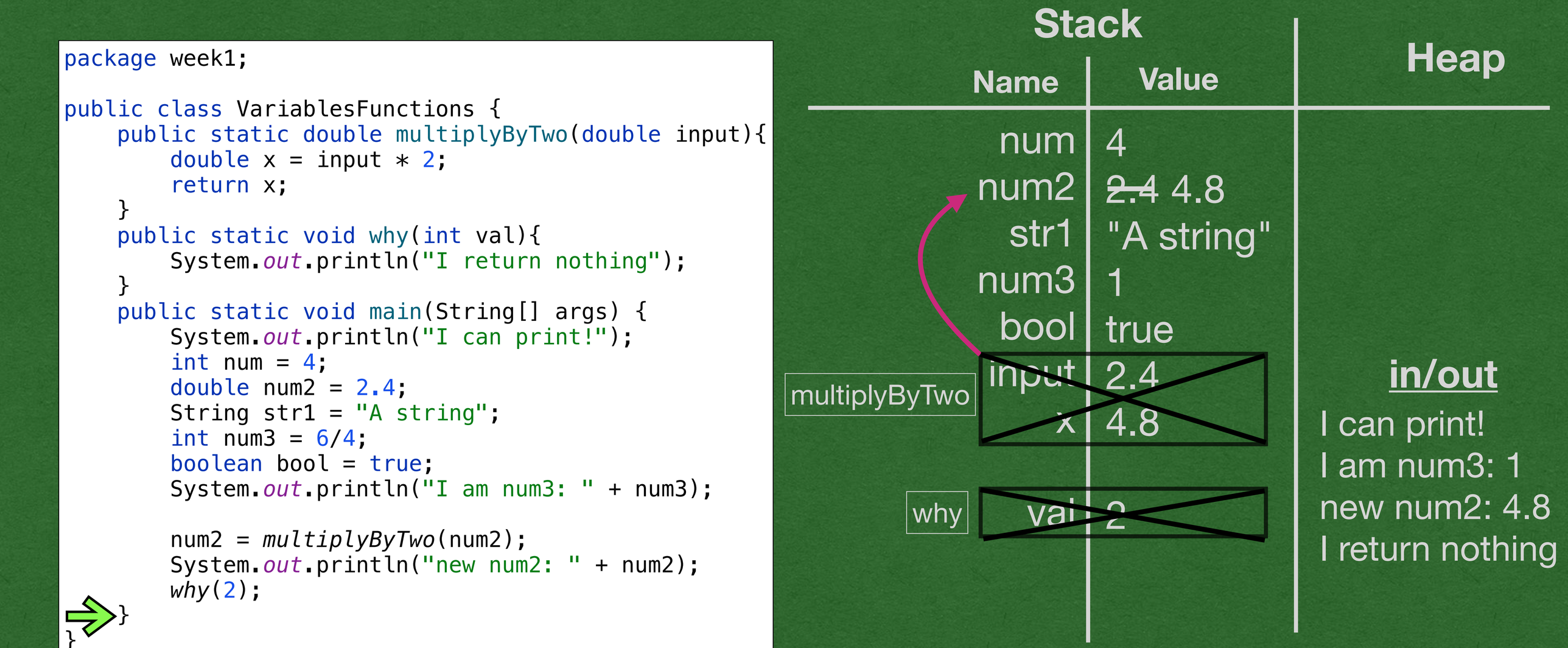


- We reach another method call
- This method returns void so there's no return arrow

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```

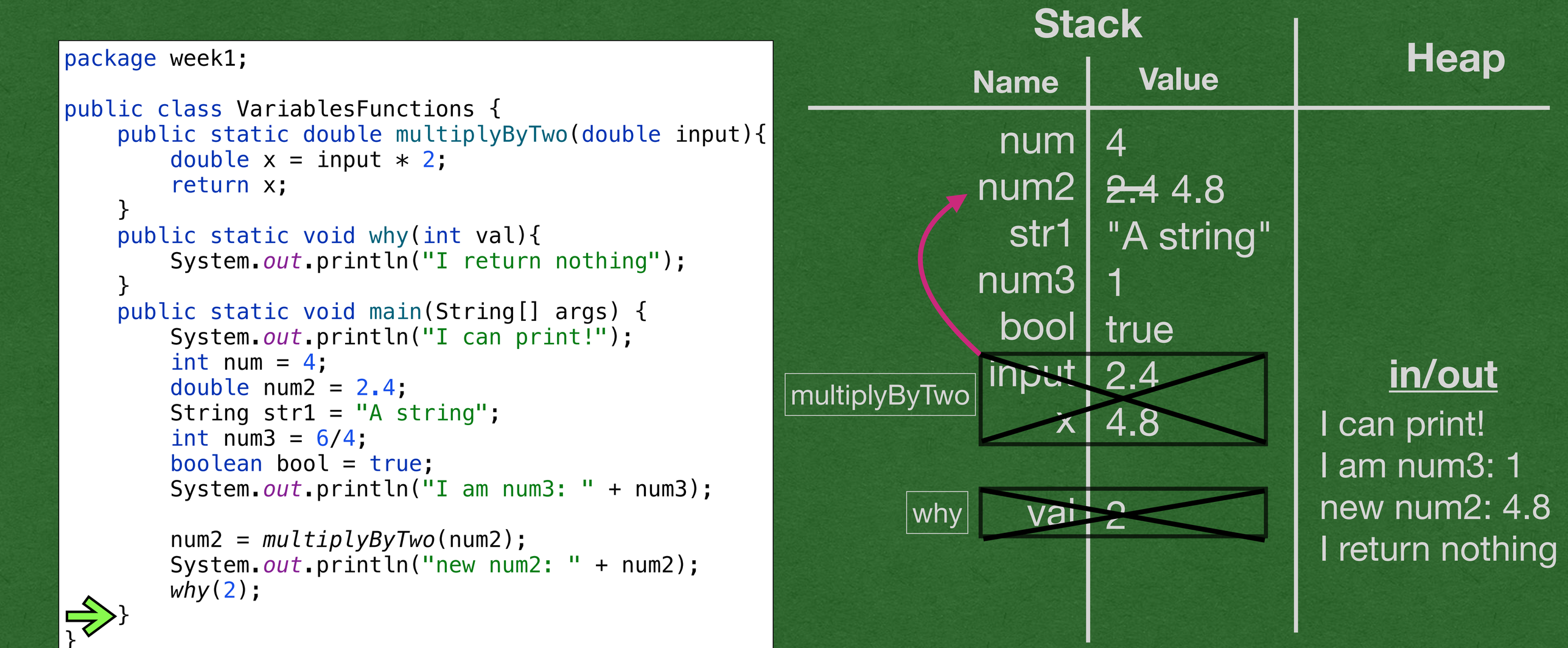


- The method prints, then exits
- Cross out the stack frame

```java
package week1;

public class VariablesFunctions {
    public static double multiplyByTwo(double input){
        double x = input * 2;
        return x;
    }
    public static void why(int val){
        System.out.println("I return nothing");
    }
    public static void main(String[] args) {
        System.out.println("I can print!");
        int num = 4;
        double num2 = 2.4;
        String str1 = "A string";
        int num3 = 6/4;
        boolean bool = true;
        System.out.println("I am num3: " + num3);

        num2 = multiplyByTwo(num2);
        System.out.println("new num2: " + num2);
        why(2);
    }
}
```



- We reach the end of the program
- We now have a record of everything that happened in memory while the program ran