Recursion

Interpretation
vs
Compilation

Interpretation vs. Compilation

- Interpretation
  - Code is read and executed one statement at a time
- Compilation
  - Entire program is translated into another language
  - The translated code is interpreted

Interpretation

- Python, JavaScript, etc. are interpreted languages

- If you have errors:
  - They'll commonly be *run-time* errors
  - Program crashes as it's running

- Program runs immediately when you run it

Compilation

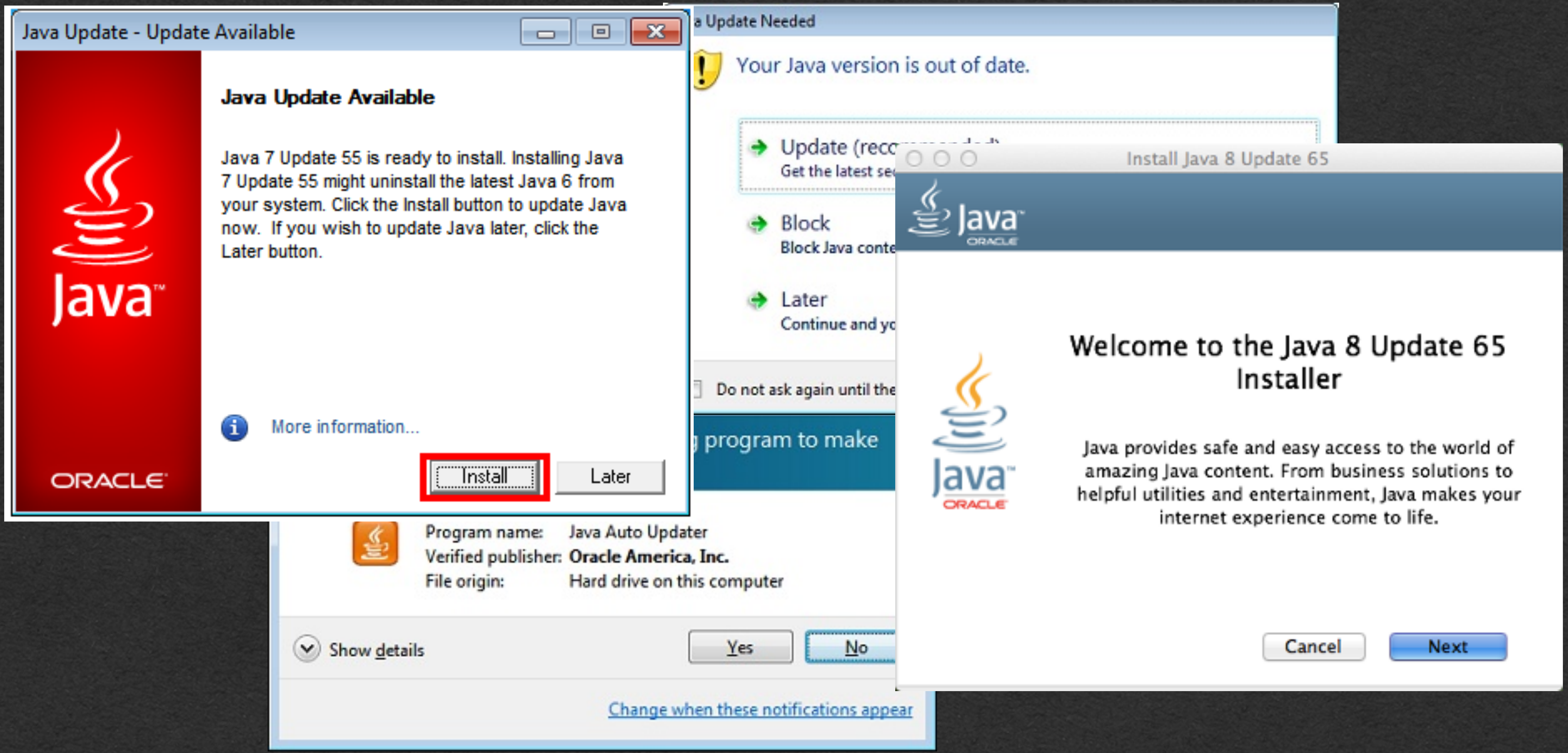
- Java, C, Scala, C++, etc. are compiled languages

- If you have errors:
    - They'll commonly be *compiler* errors
    - Compilers will check all syntax and types and alert us of any errors before they become run-time errors
    - Program fails to be converted into the target language and never runs

- Compilation takes time; Program does not run immediately

Compilation - Java

- Java compiles to Java Byte Code
- Executed by the Java Virtual Machine (JVM)
  - Installed on Billions of devices!



Java Update - Update Available

Java Update Available

Java 7 Update 55 is ready to install. Installing Java 7 Update 55 might uninstall the latest Java 6 from your system. Click the Install button to update Java now. If you wish to update Java later, click the Later button.

a Update Needed

Your Java version is out of date.

Block Java conte

Continue and yo

Do not ask again until the

Get the latest se

program to make

Program name: Java Auto Updater

Verified publisher: Oracle America, Inc.

File origin: Hard drive on this computer

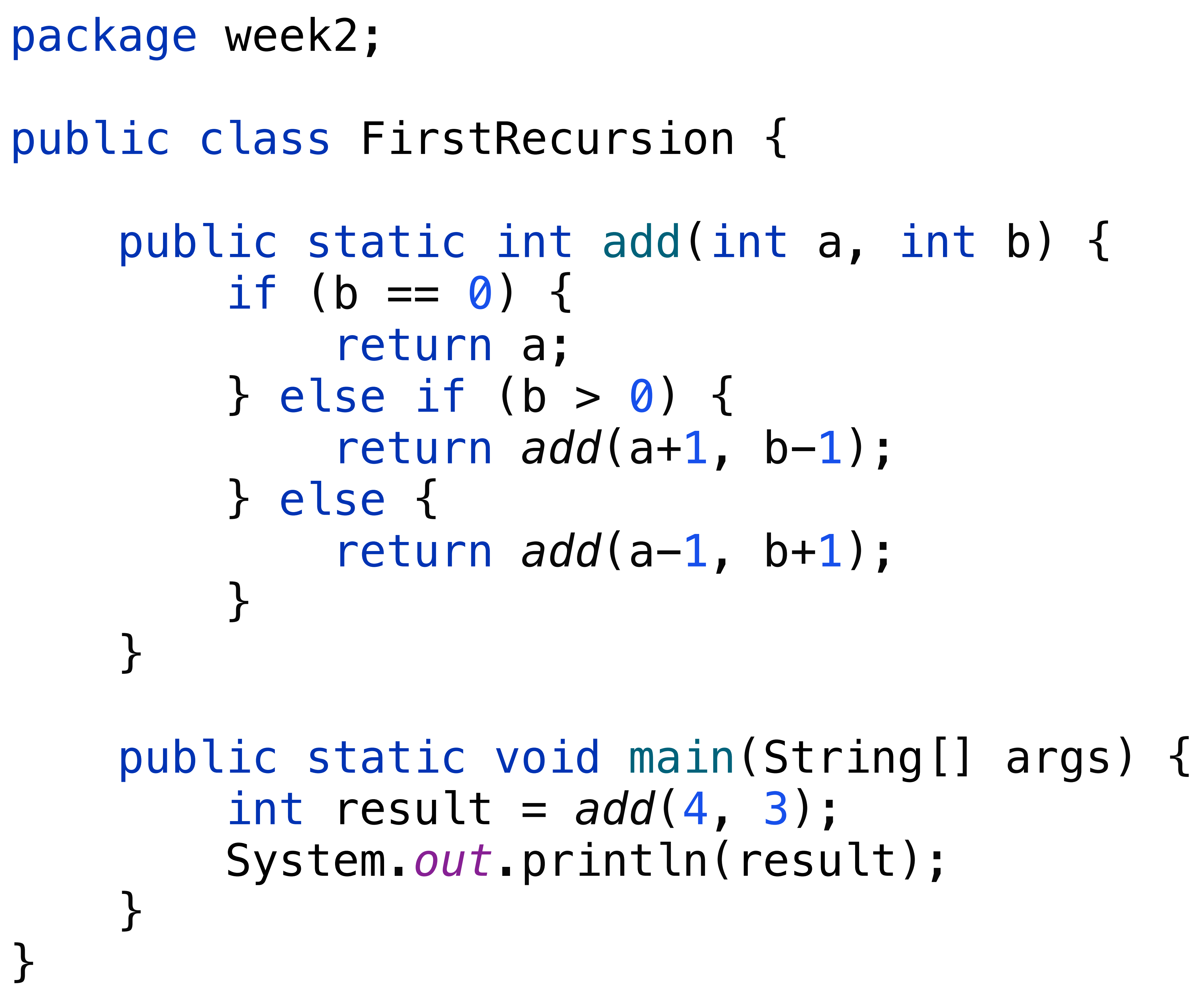
Install Java 8 Update 65

Welcome to the Java 8 Update 65 Installer

Java provides safe and easy access to the world of amazing Java content. From business solutions to helpful utilities and entertainment, Java makes your internet experience come to life.

Recursion

Recursion

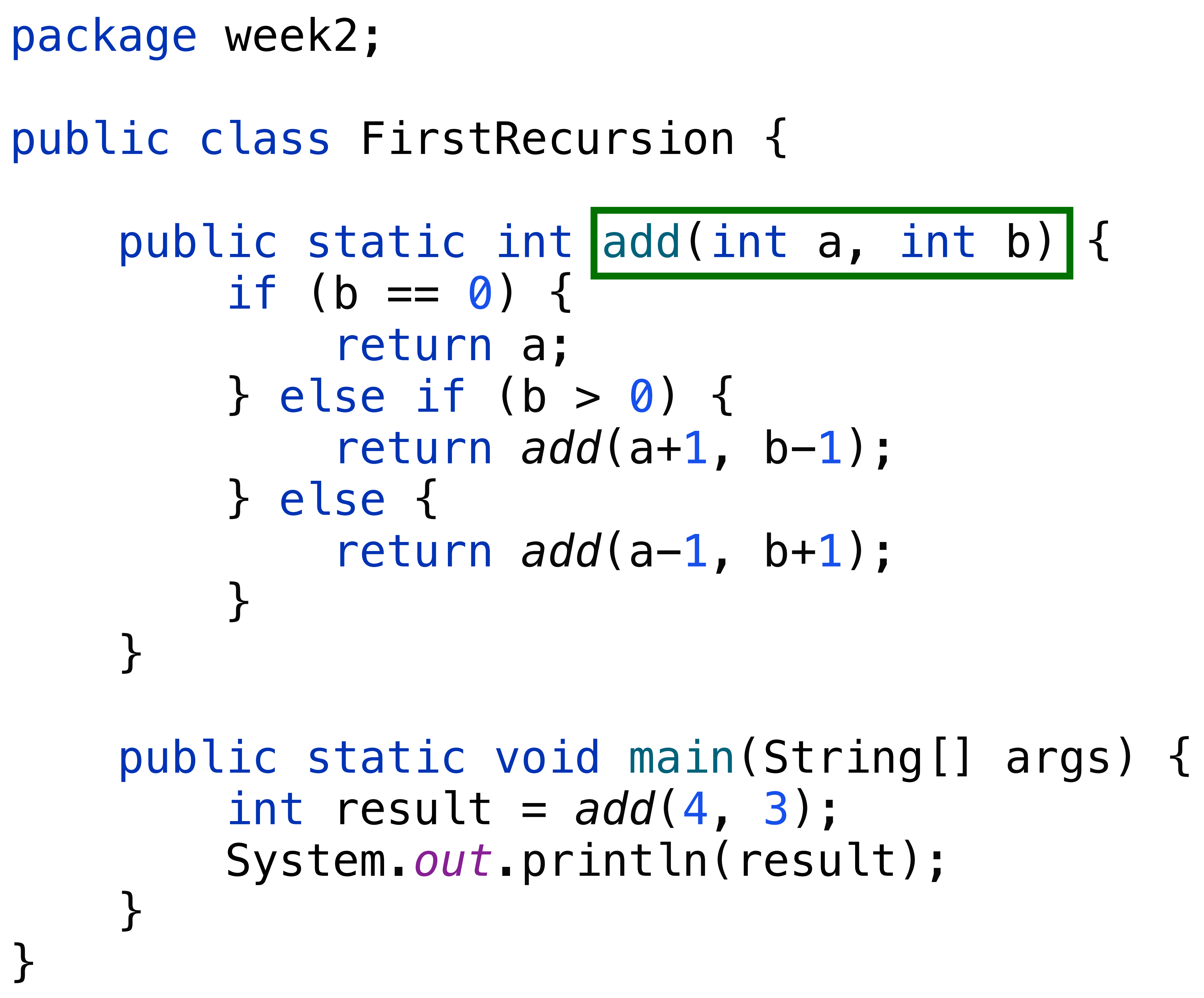


```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- Recursion:
  - When a method/function calls itself

- A recursive method is a method that calls itself

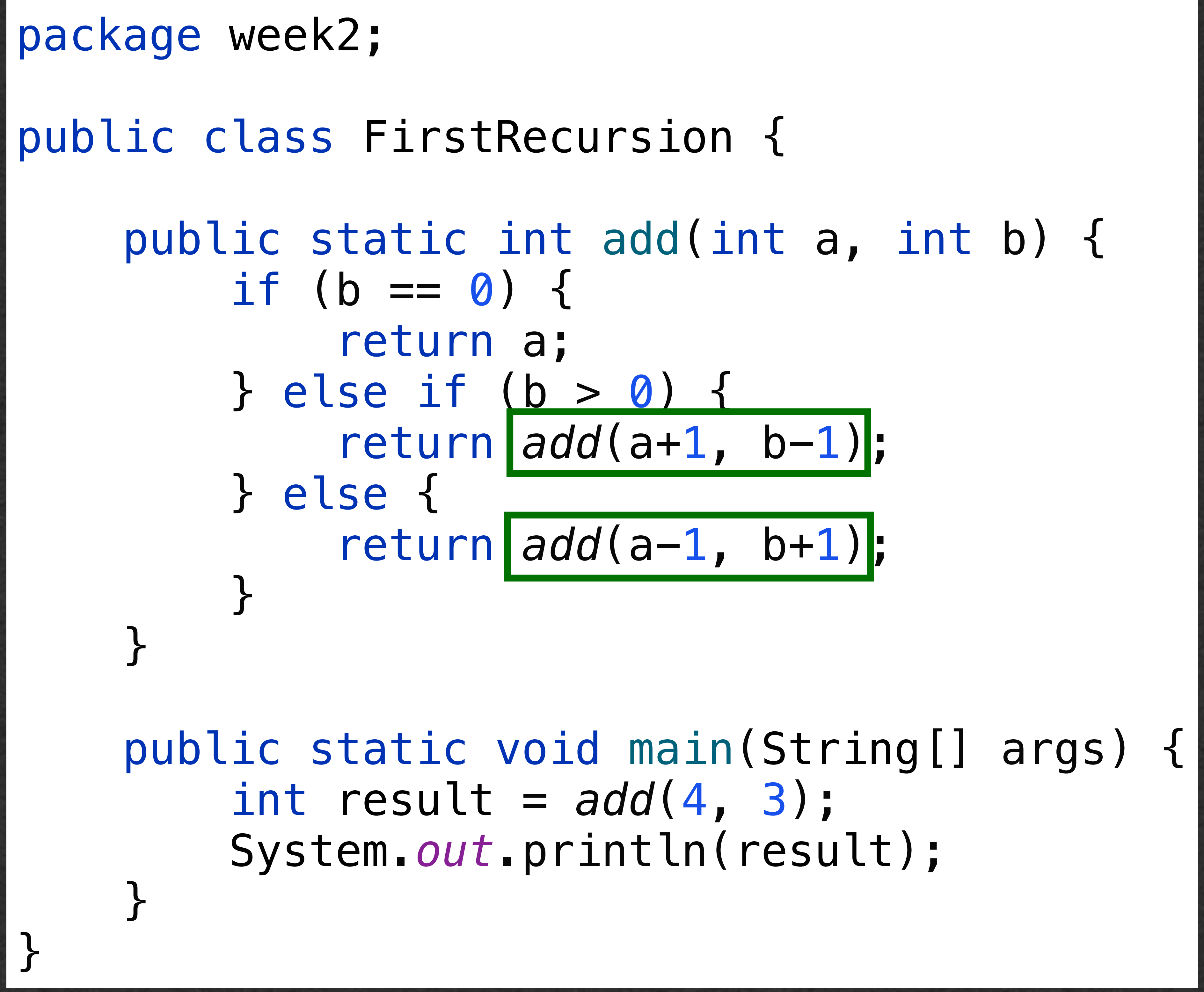
Recursion



```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- We are defining a method named **add**

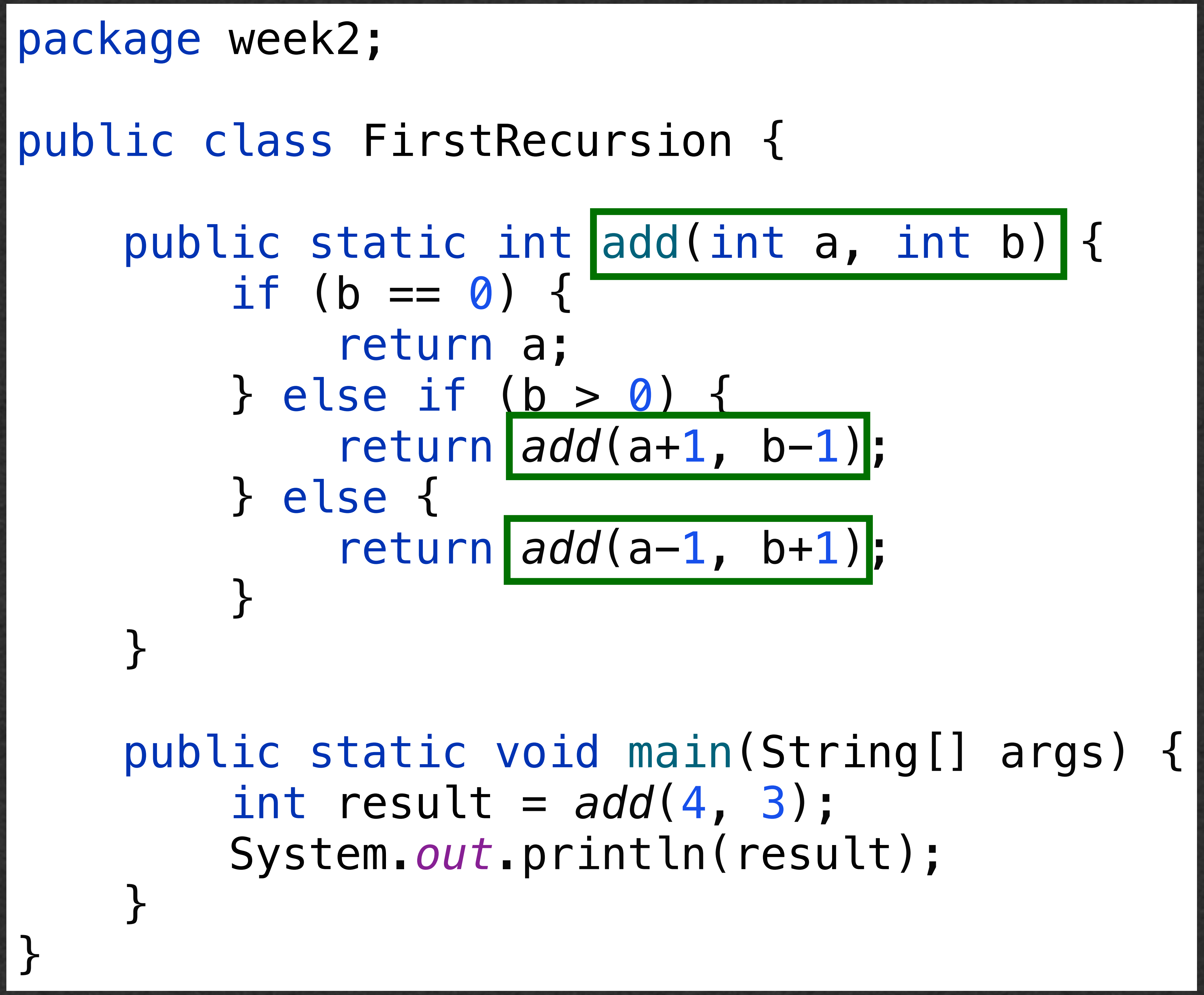
Recursion



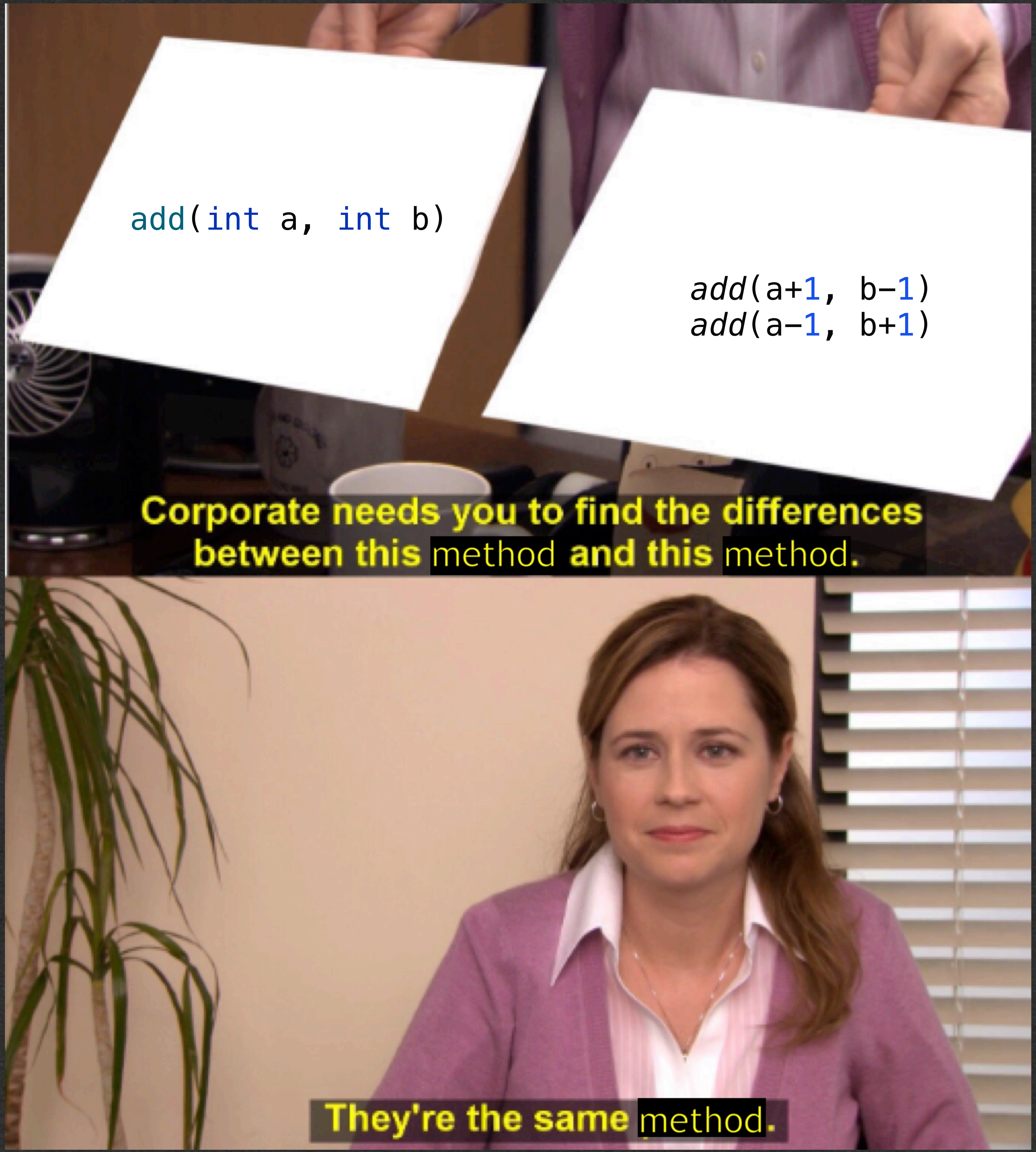
```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- We are defining a method named **add**

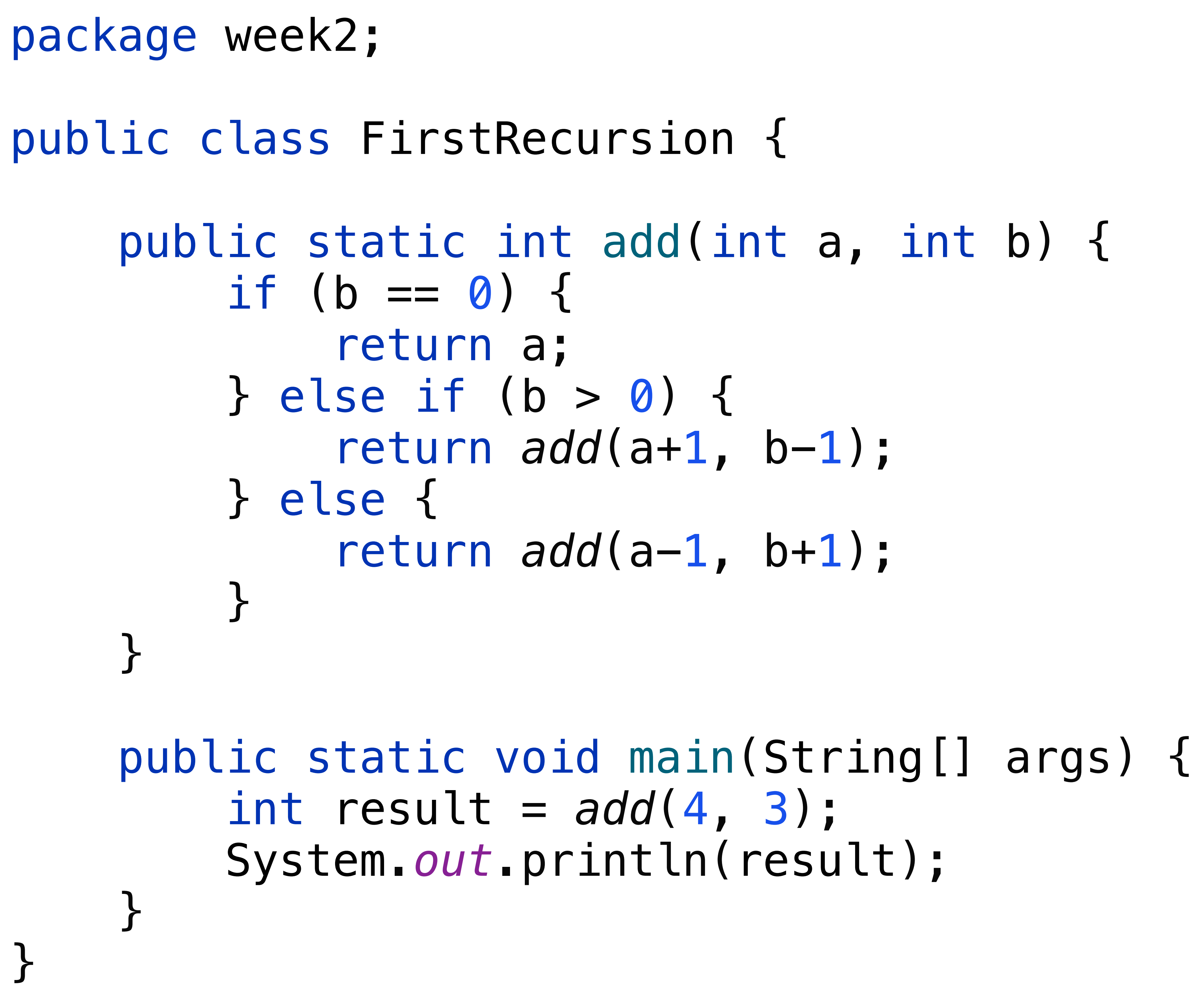
- And we call a method named **add**



```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```


add(int a, int b)
add(a+1, b-1)
add(a-1, b+1)
Corporate needs you to find the differences between this method and this method.
They're the same method.

Recursion

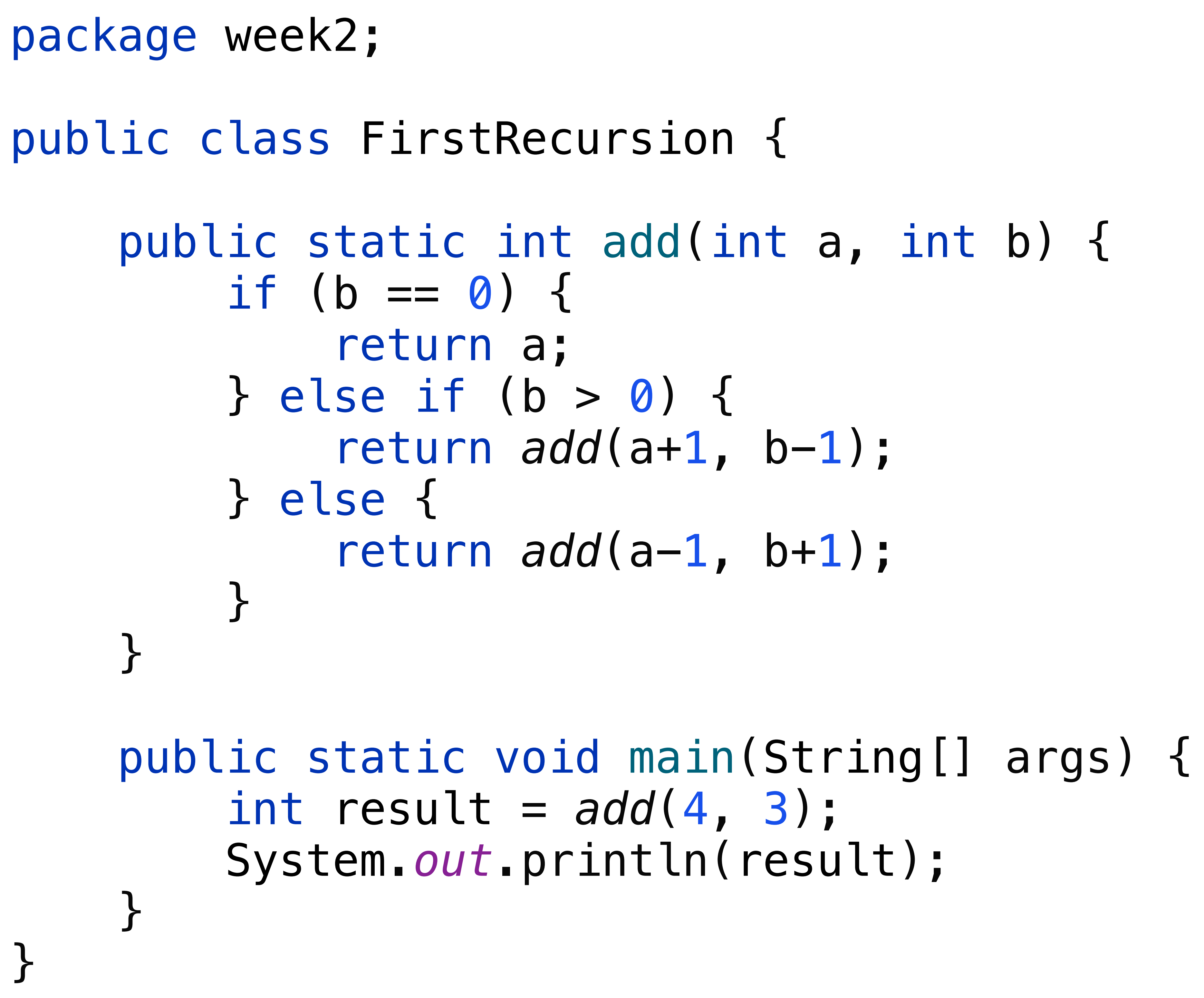


```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- This **add** method calls itself!

- This is a recursive method.

Recursion



```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- Let's see how this work with a...

Memory Diagram!

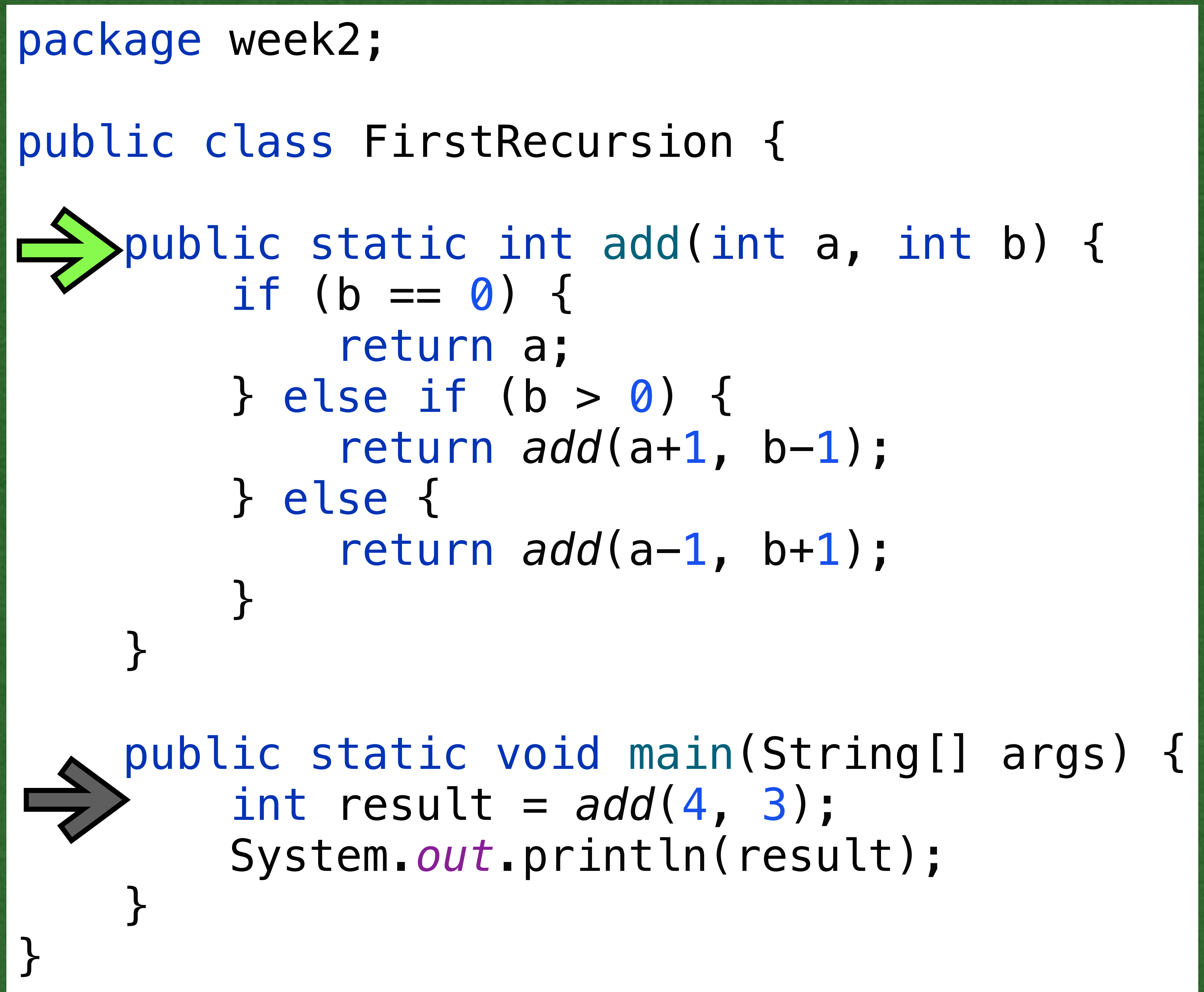
Stack
Heap
IO
Global Variables
Create Global Variable
Stack Frames
Create Heap Object
Create IO Line

Name	Value
main	
result	7 Cross out
add	
a	4 Cross out
b	3 Cross out
add	
a	5 Cross out
b	2 Cross out
add	
a	6 Cross out
b	1 Cross out
add	
a	7 Cross out
b	0 Cross out

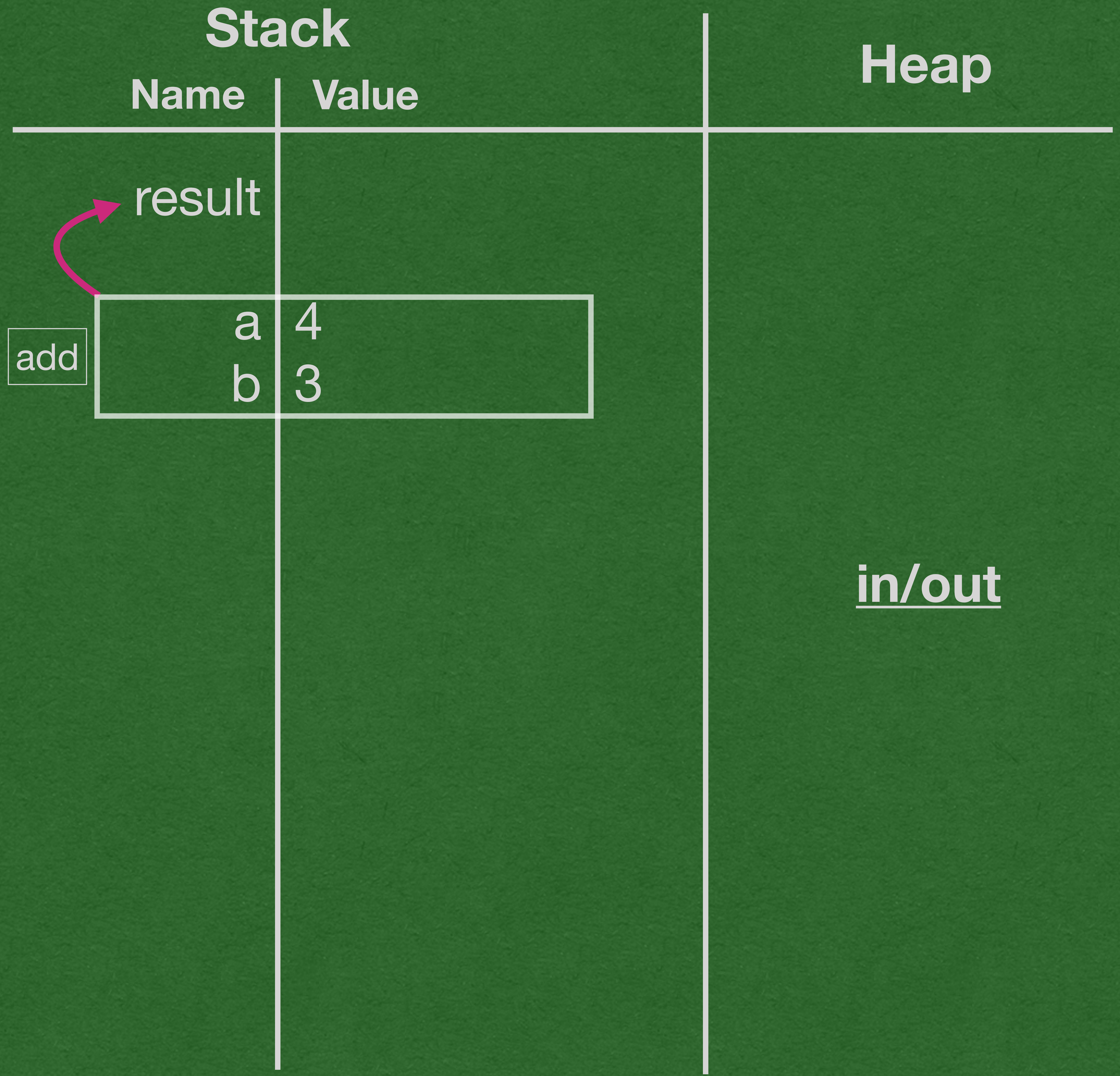
IO:

```
7
```

```java
package week3;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

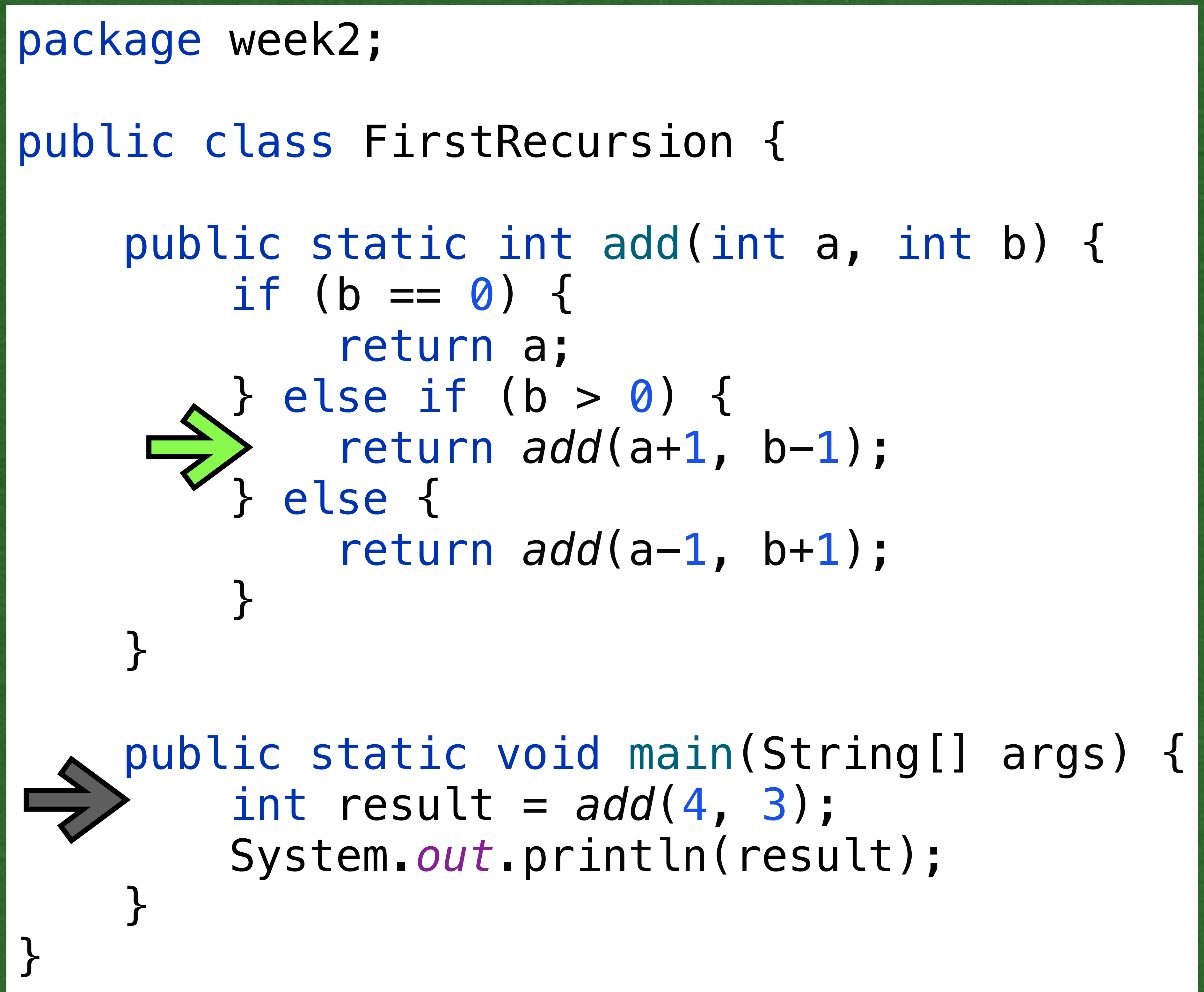


```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

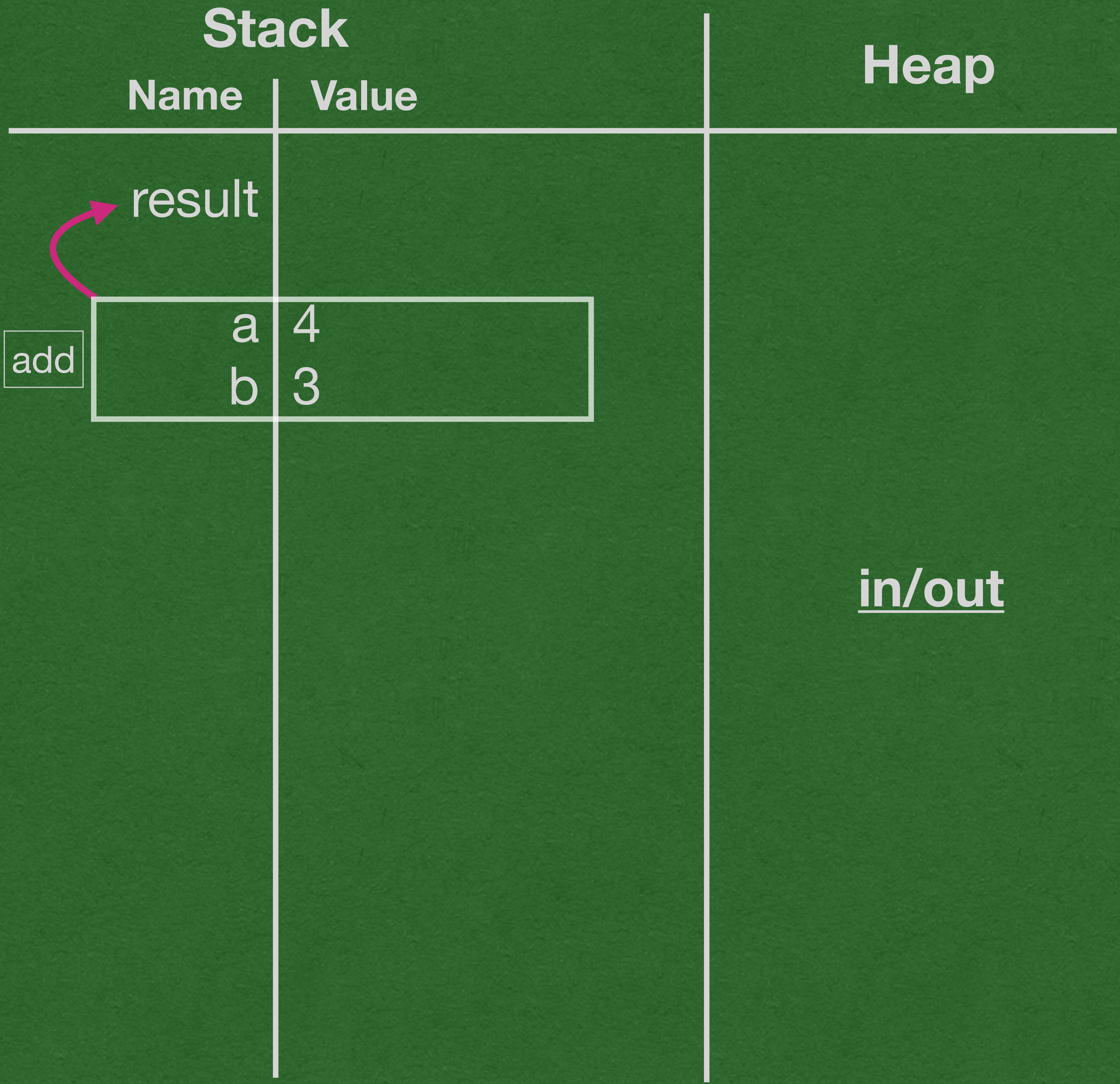


Name	Value
result	
a	4
b	3

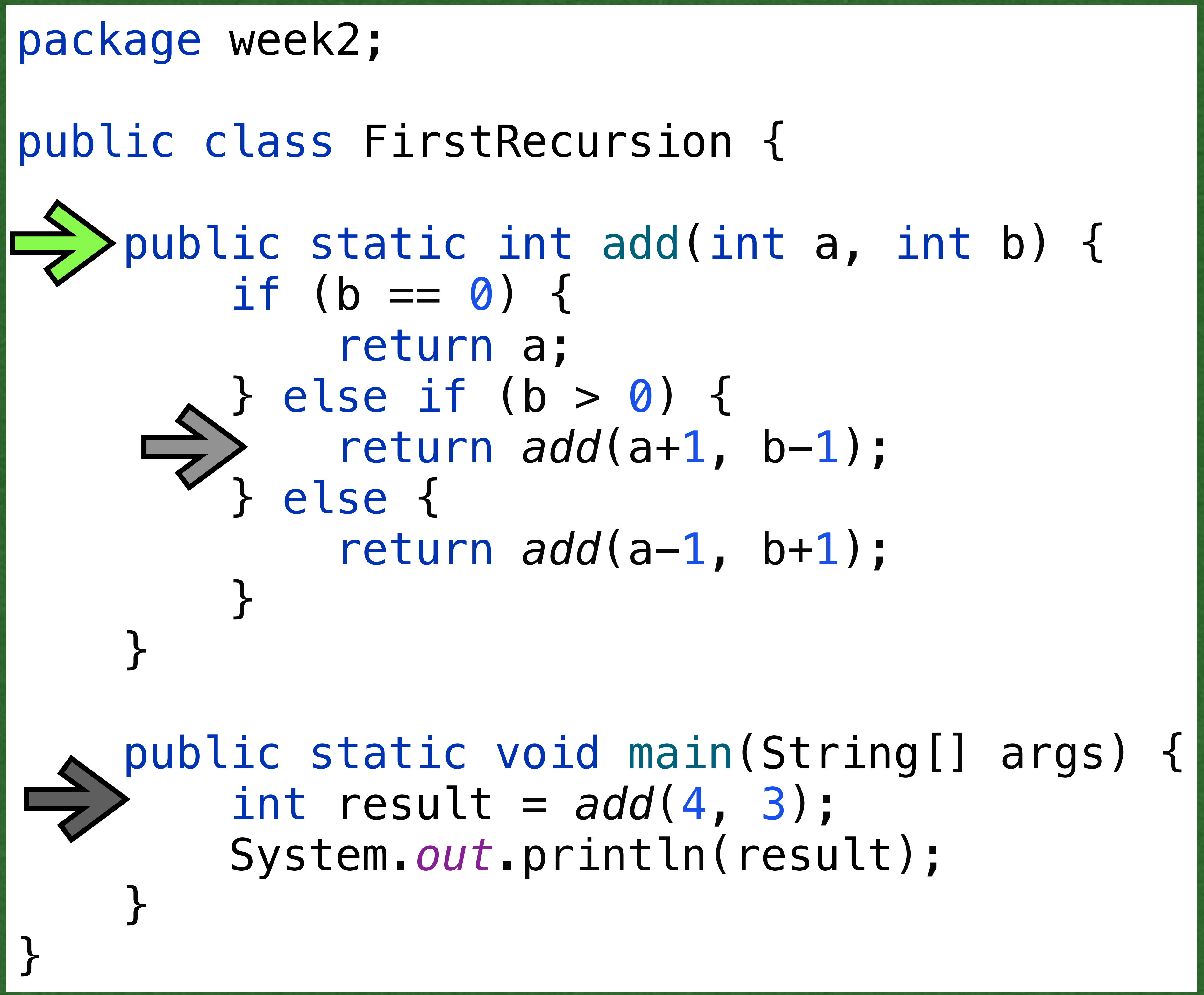
- The diagram starts with a method call
- Add a stack frame on the stack with the parameters of the method call



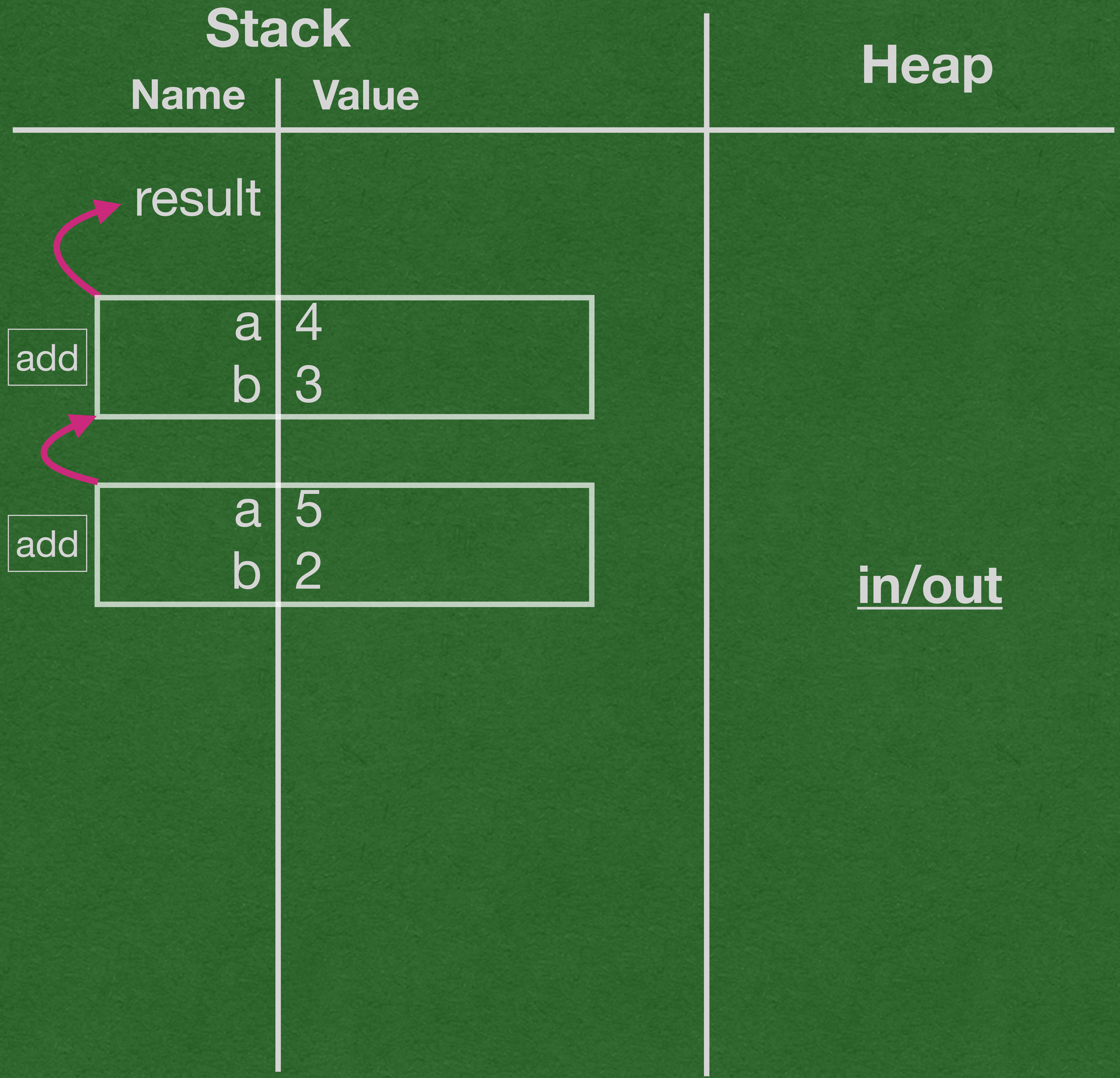
```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



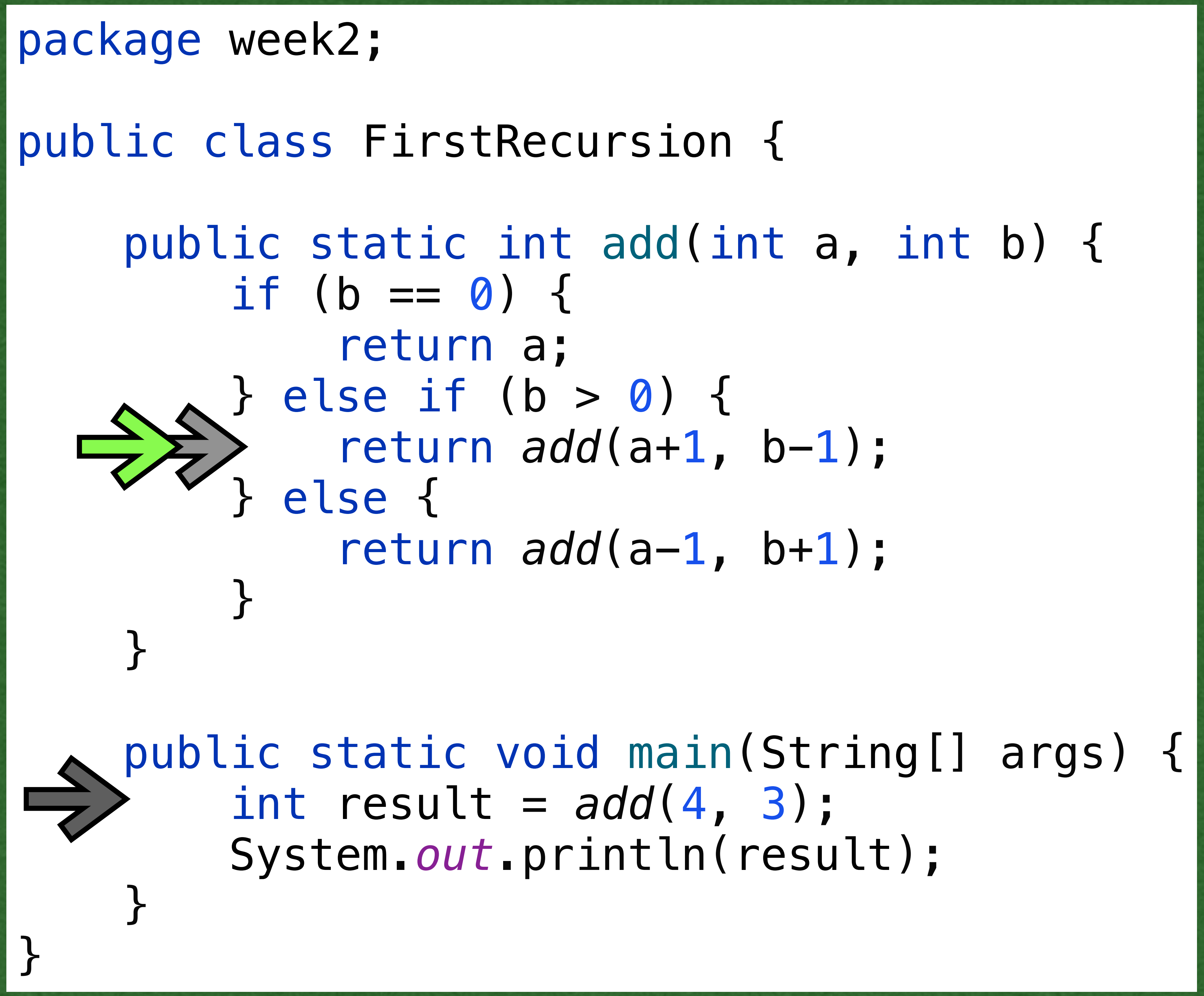
- We reach the recursive call
- The trick:
  - There is none. Treat this as any other method call



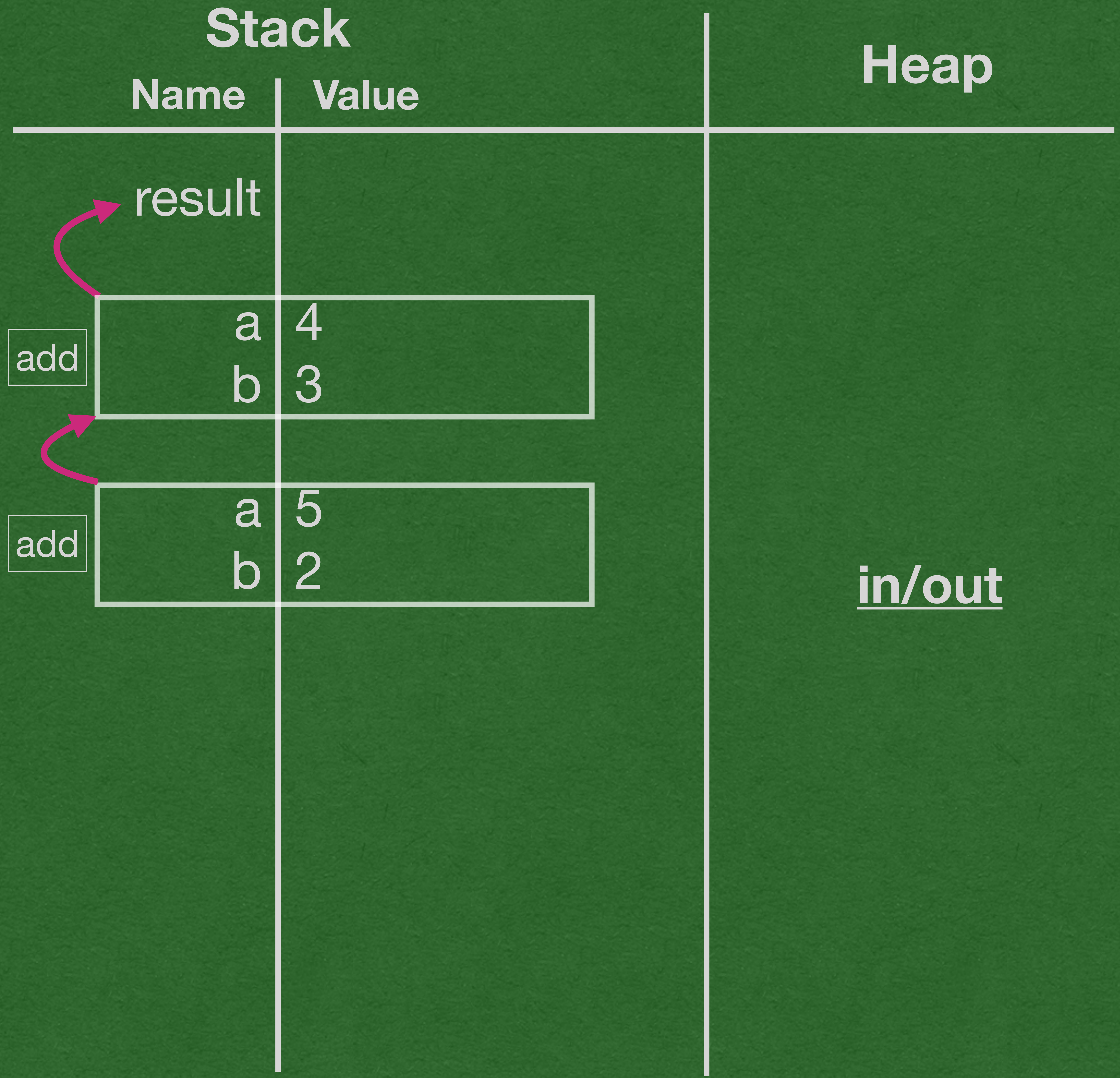
```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



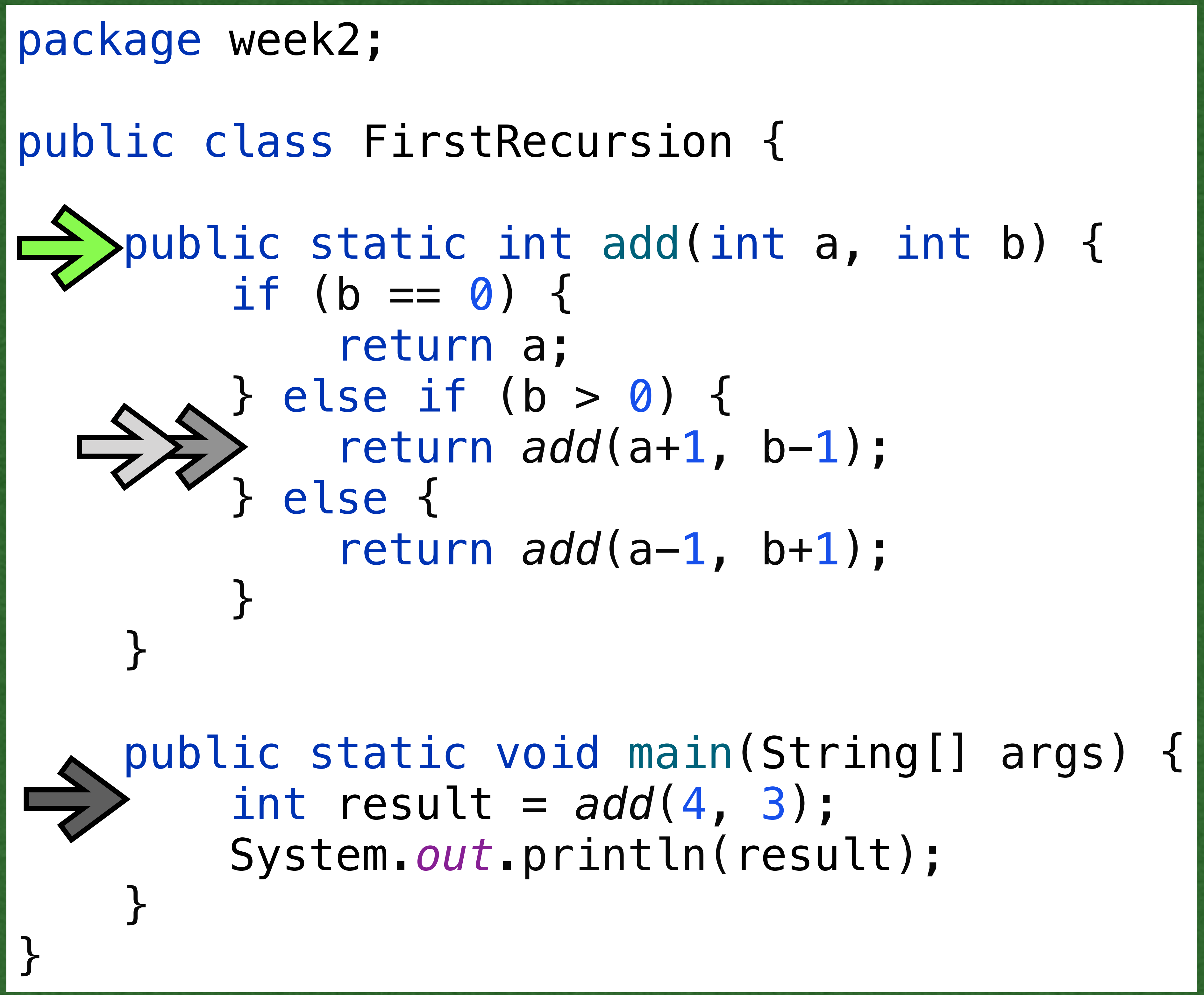
- Add the stack frame a parameters to the stack just like any other method call
- The return arrow points to the stack frame that called the method



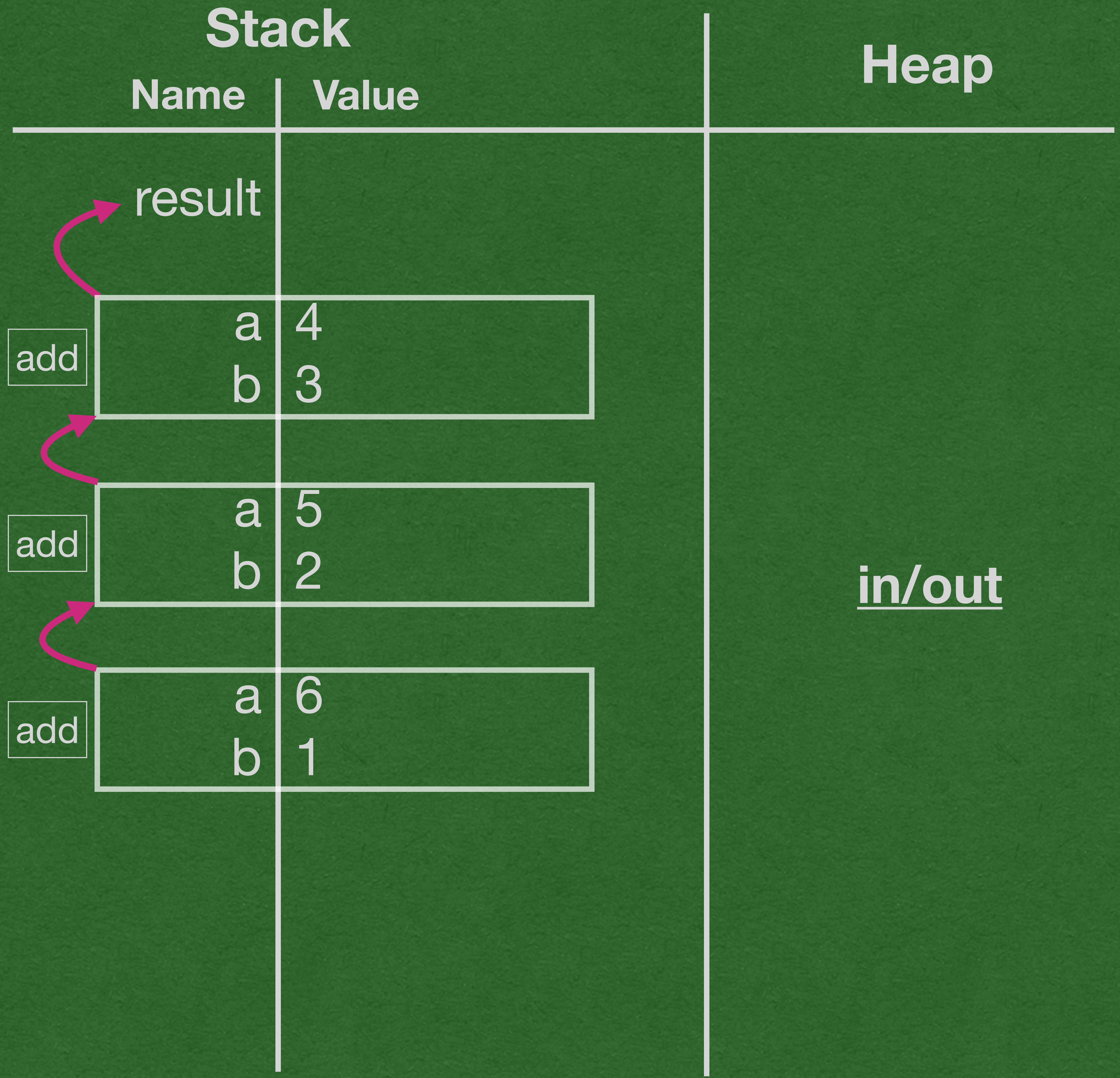
```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



- We get to the next recursive call
- Do it again!



```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

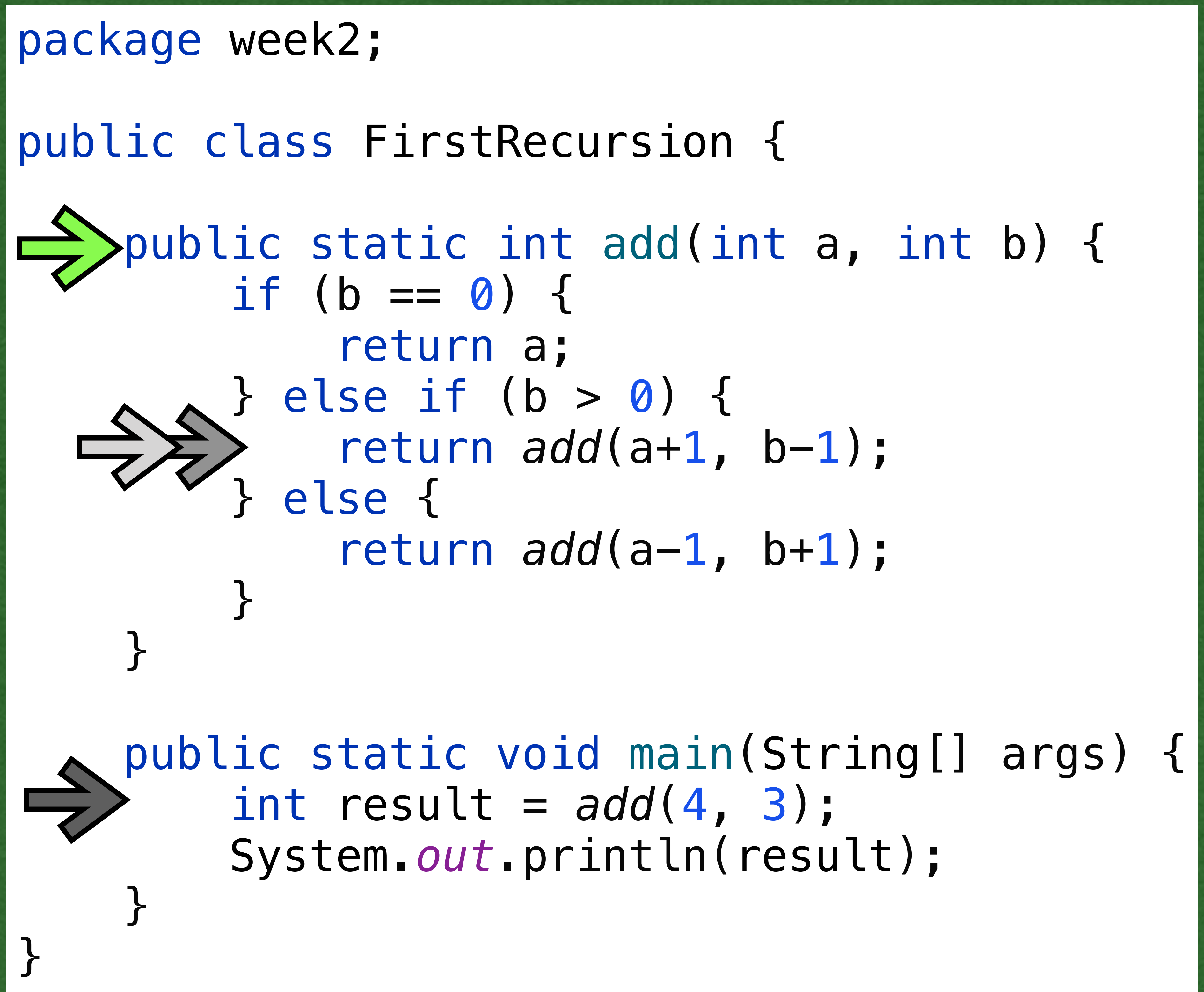


Stack

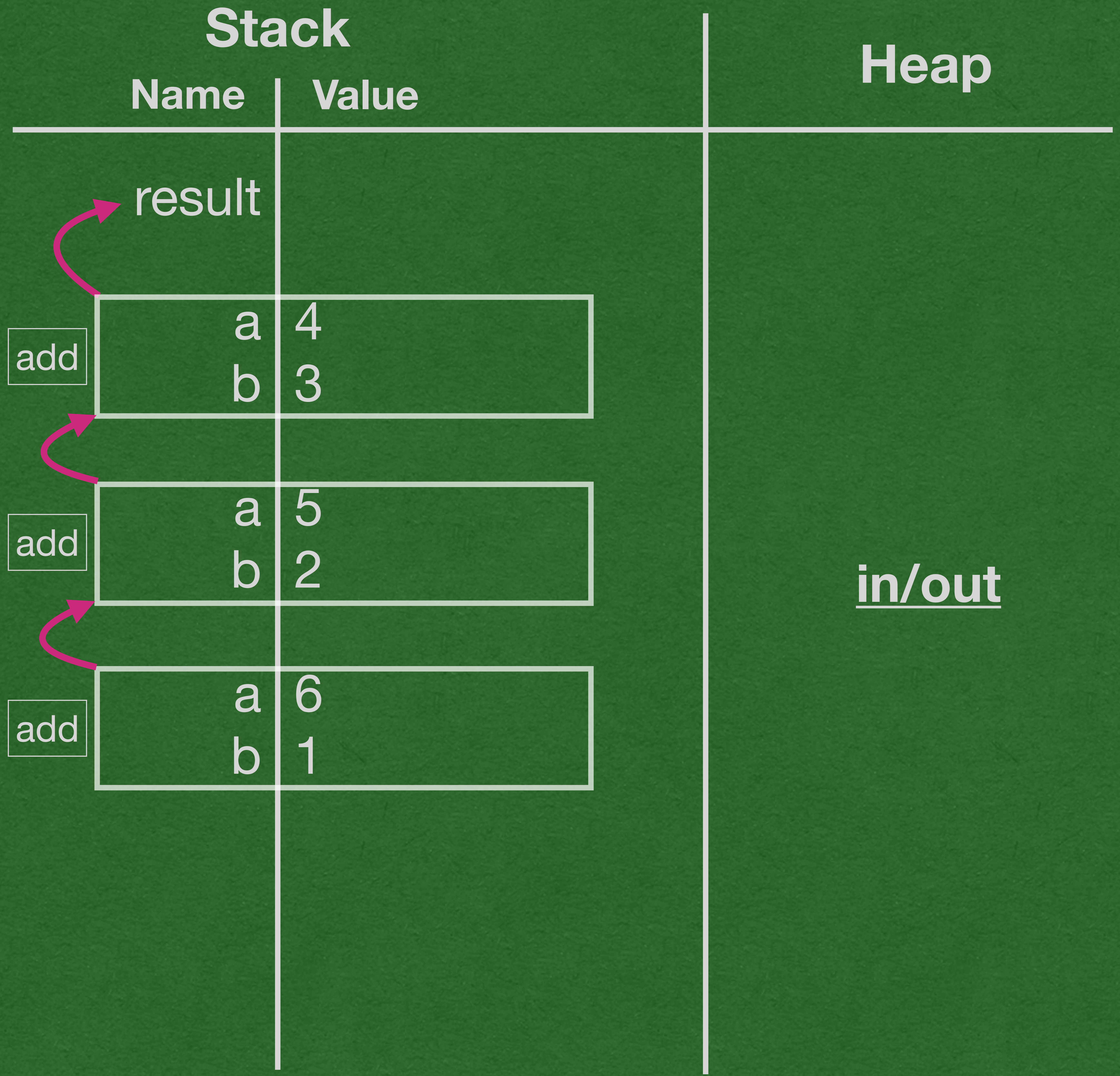
Heap

Name	Value
result	
a	4
b	3
a	5
b	2
a	6
b	1

- We have 3 frames on the stack (plus the main stack frame)
- Only the frame on the top of the stack is active



```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



Stack

Heap

Name | Value

result

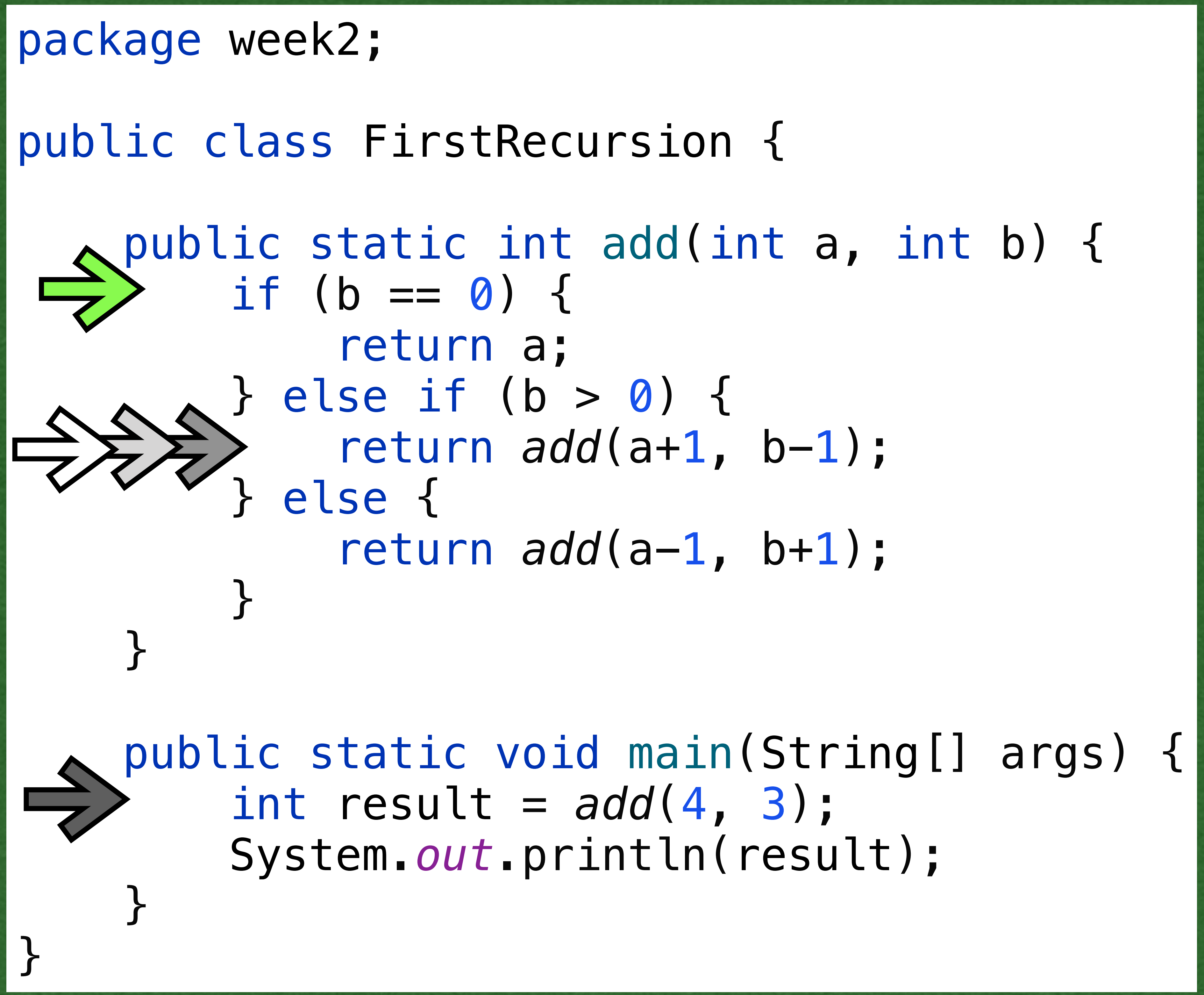
add
a 4
b 3

add
a 5
b 2

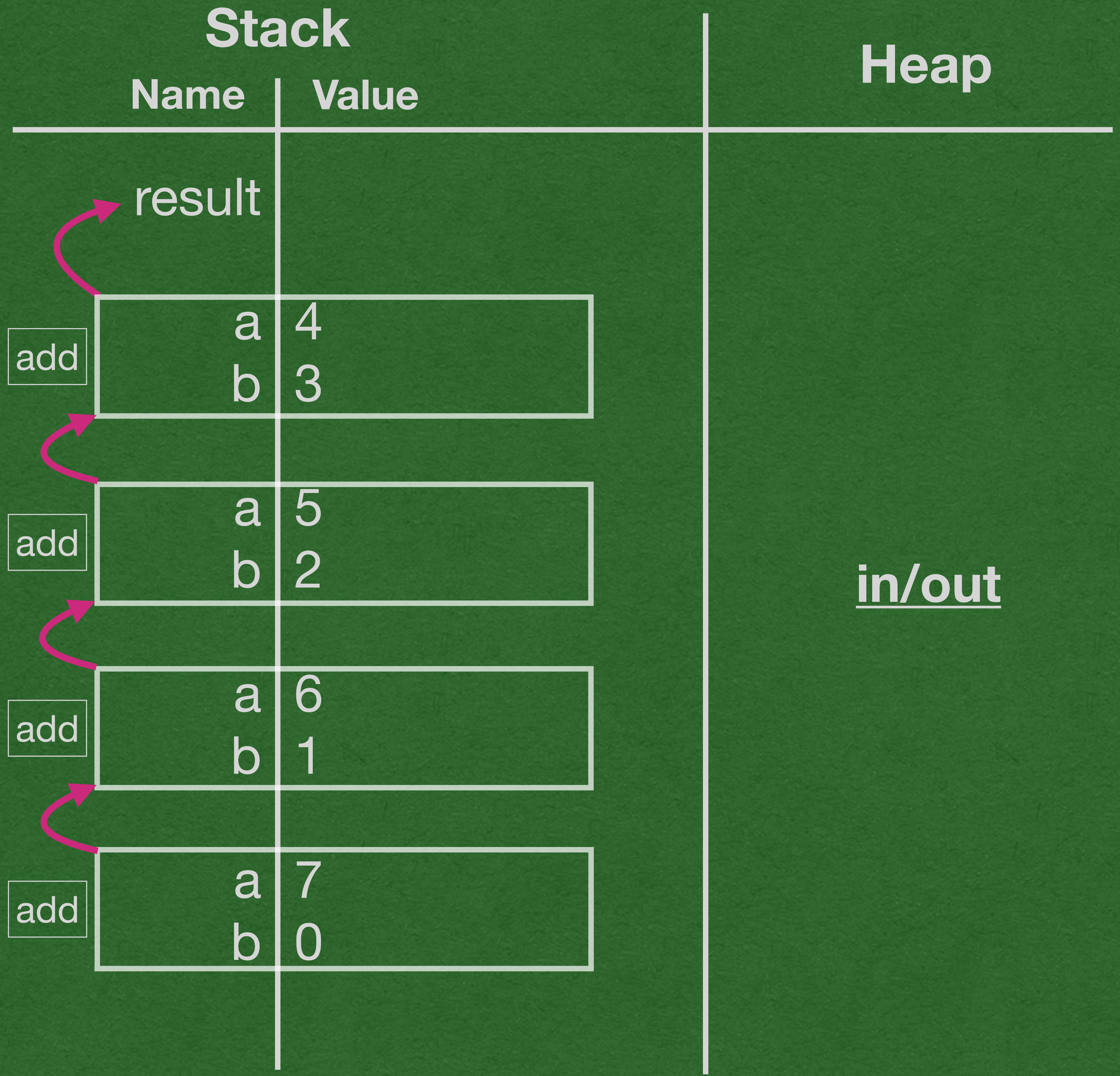
add
a 6
b 1

in/out

- The other stack frames are waiting until they are back on top of the stack



```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

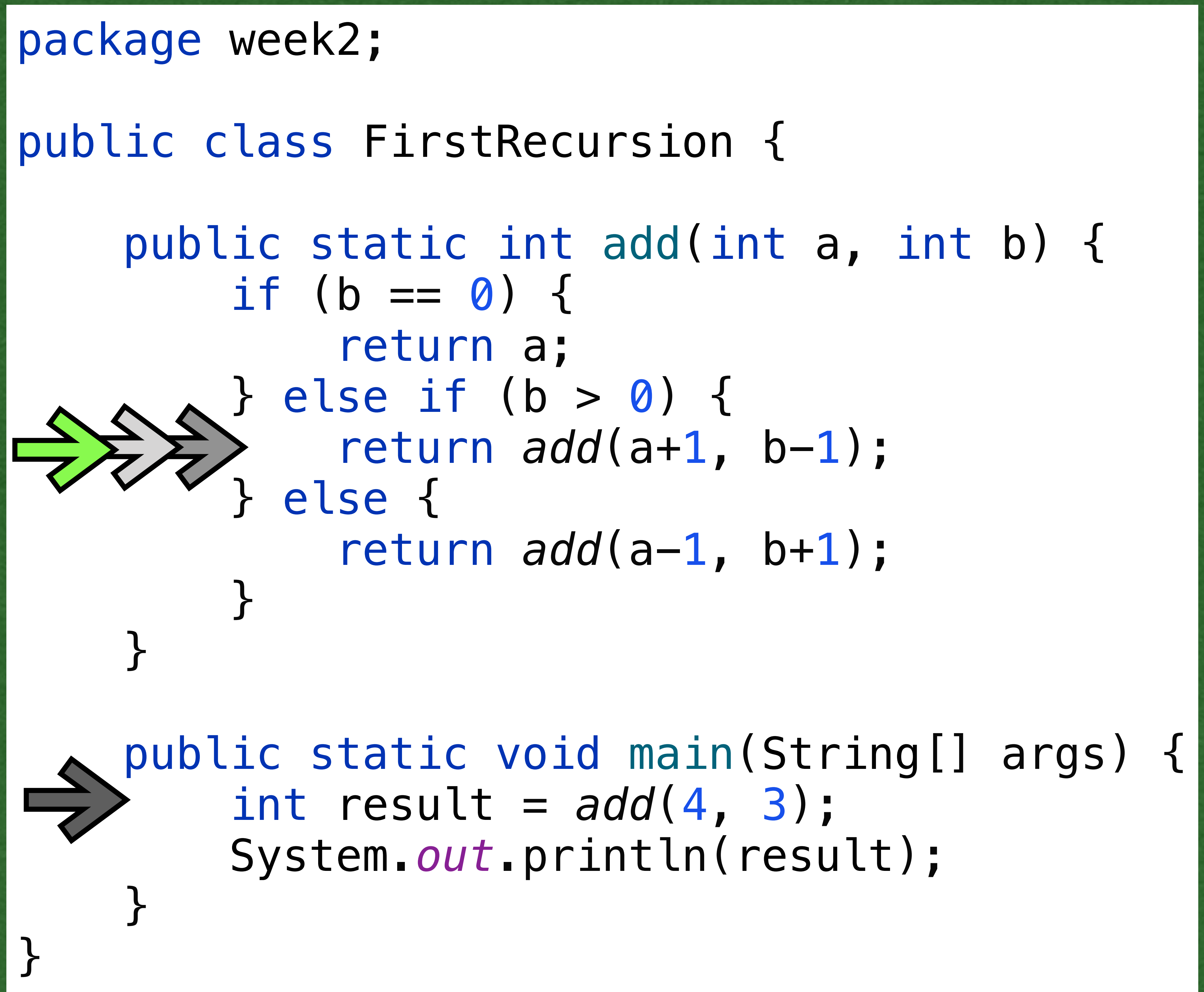


Stack

Heap

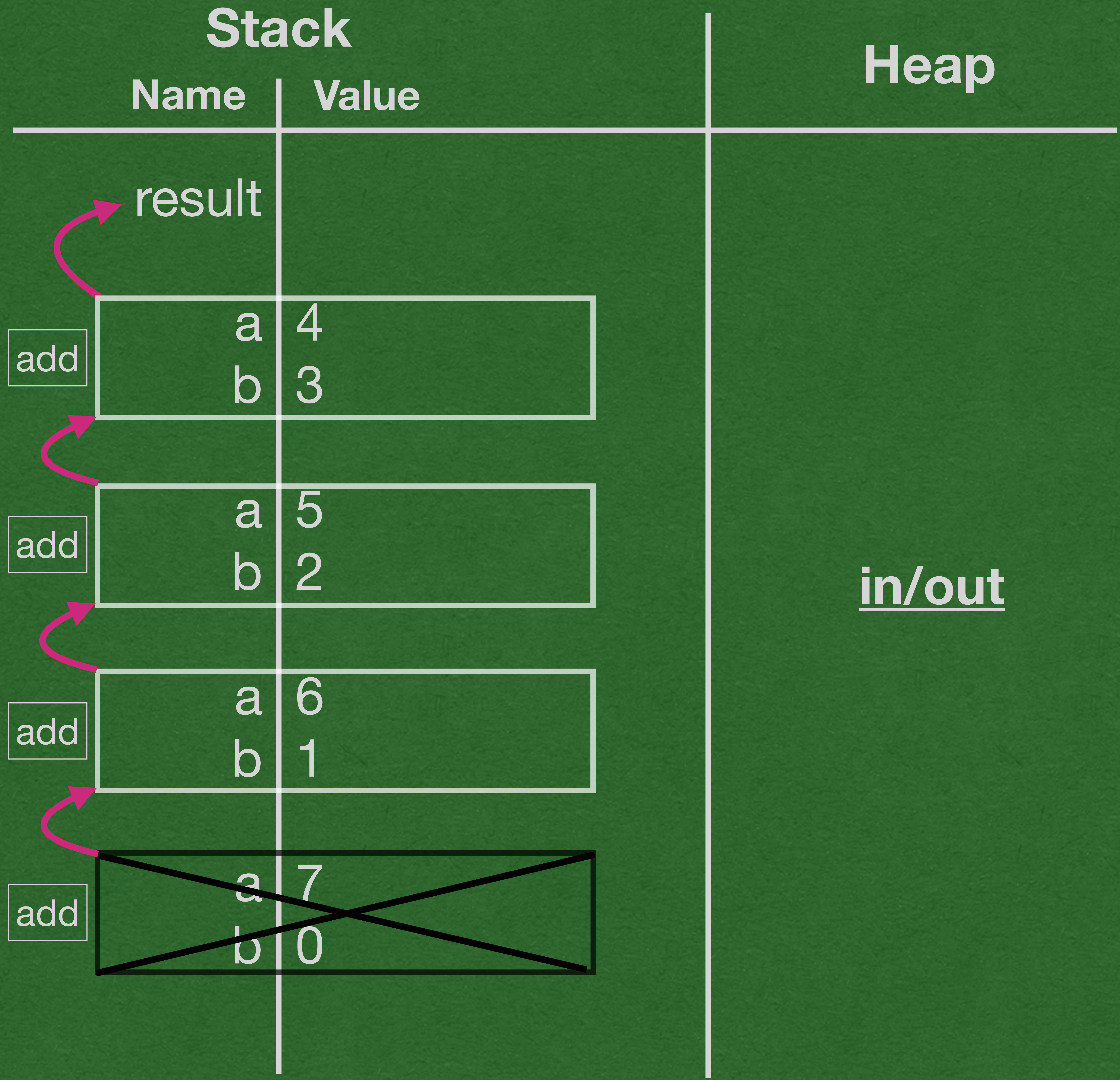
	Name	Value
	result	
add	a	4
	b	3
add	a	5
	b	2
add	a	6
	b	1
add	a	7
	b	0

- Call the method again
- This time, the first condition is true



```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- This stack frame returns 7
- The frame is removed from the stack and the next frame regains control

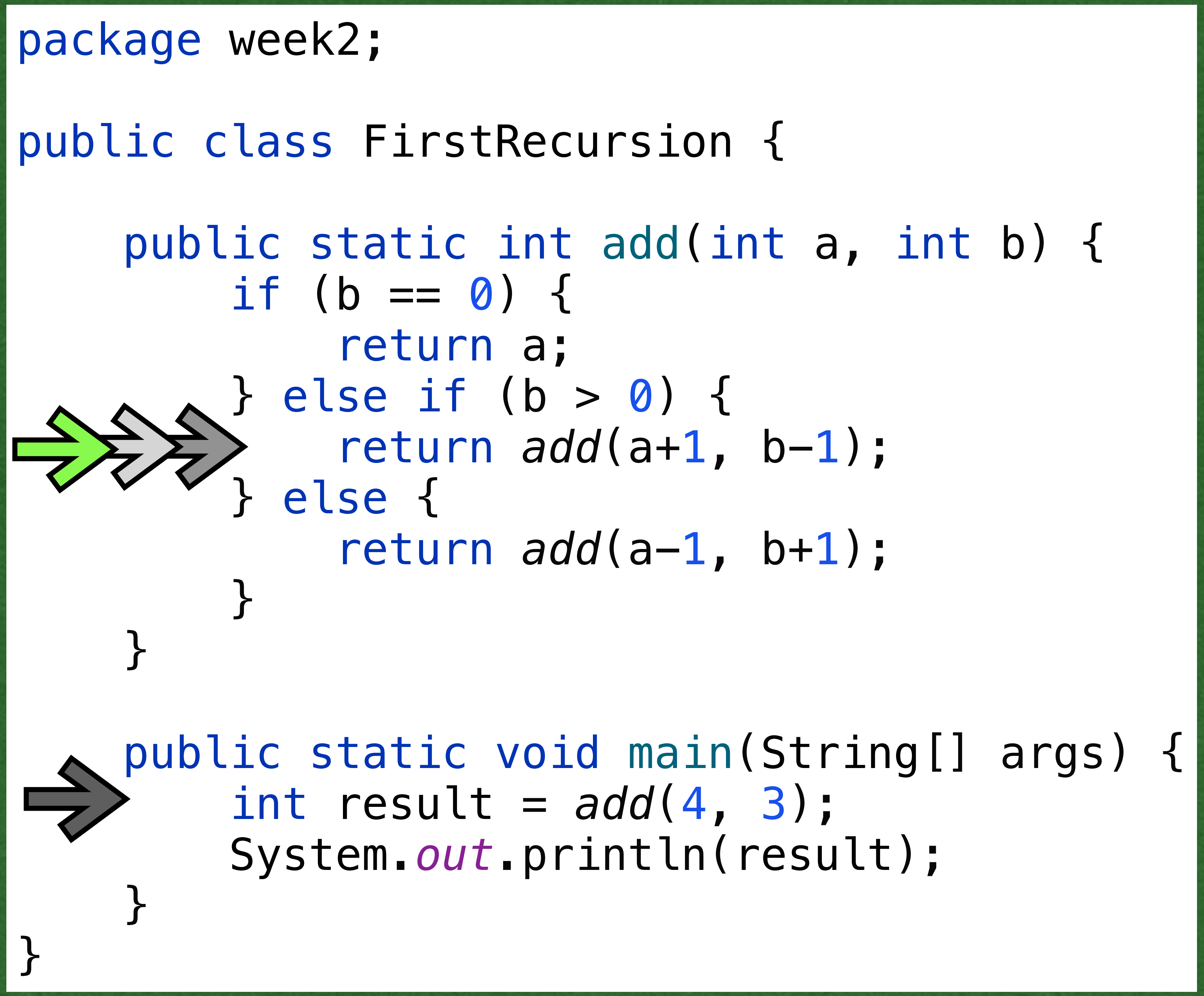


Stack

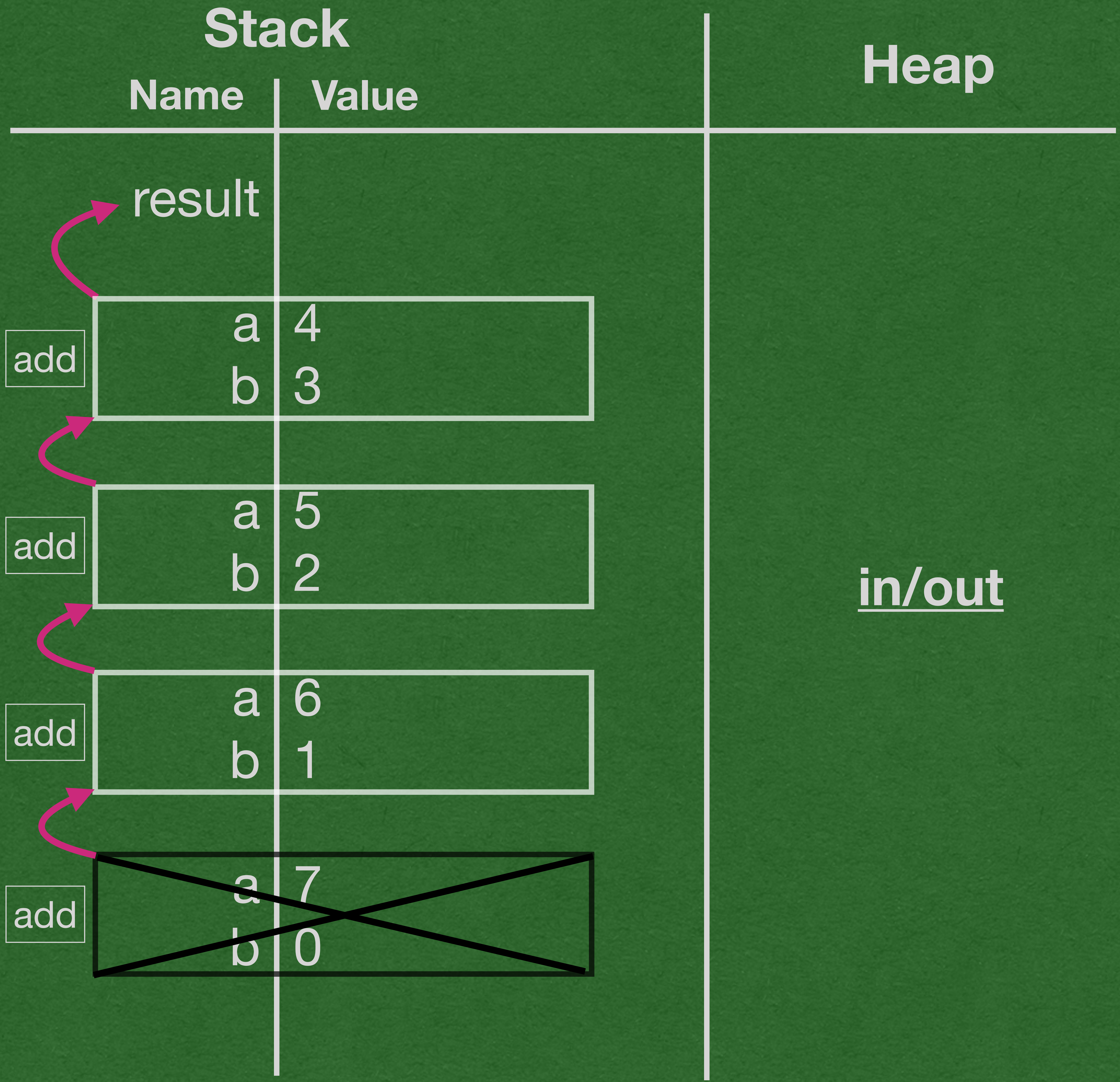
Heap

	Name	Value
	result	
add	a	4
	b	3
add	a	5
	b	2
add	a	6
	b	1
add	a	7
	b	0

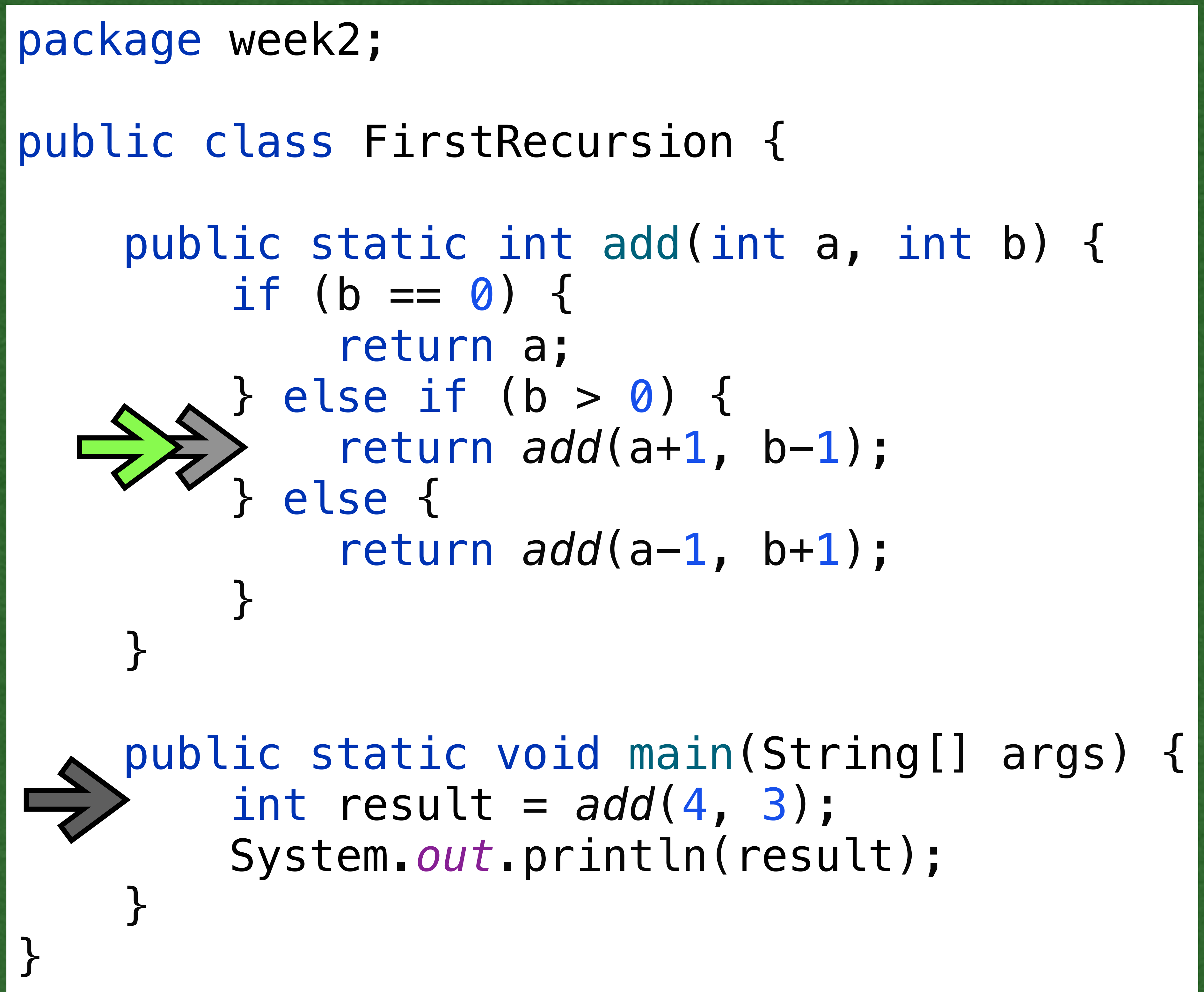
in/out



```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

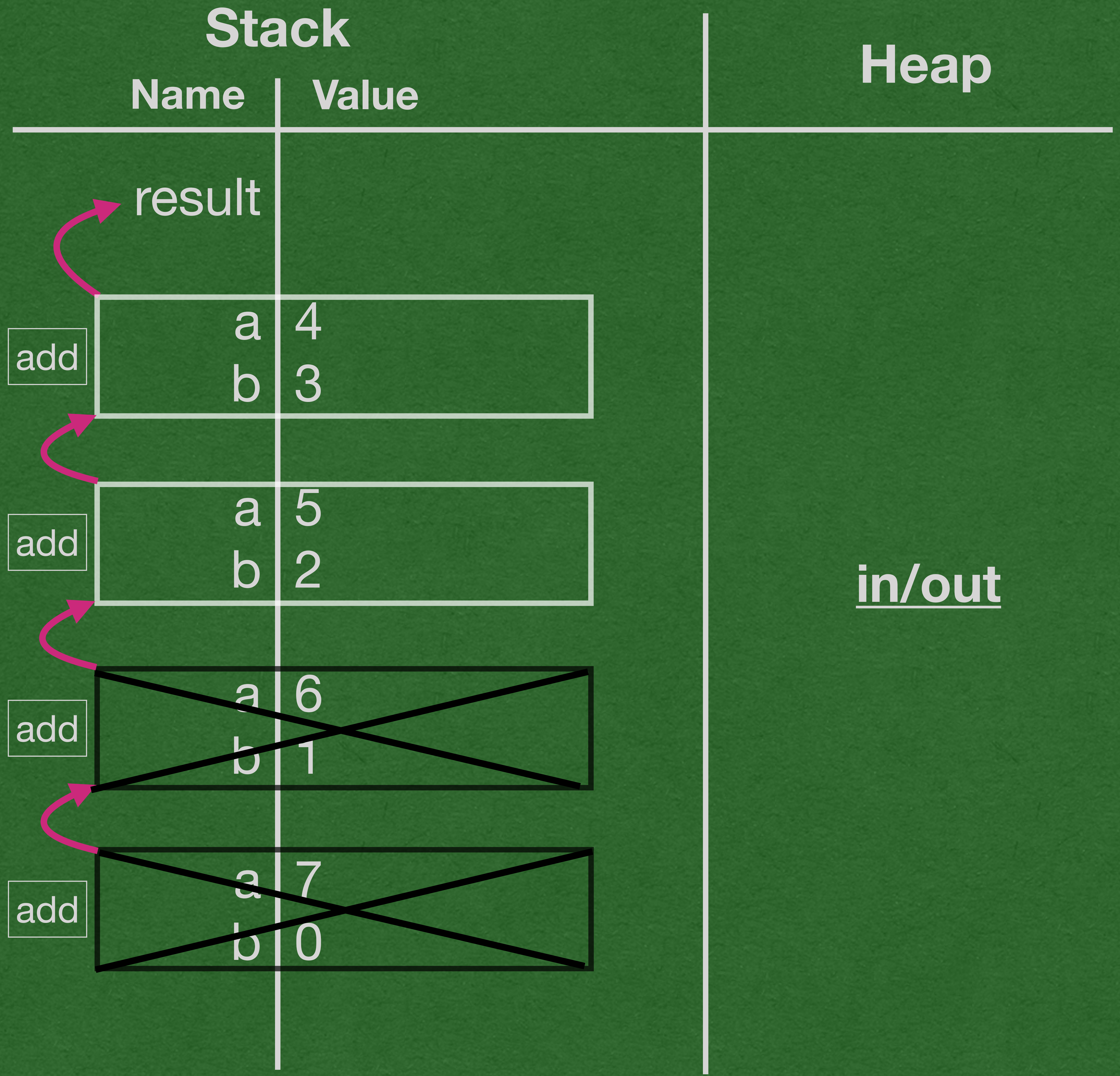


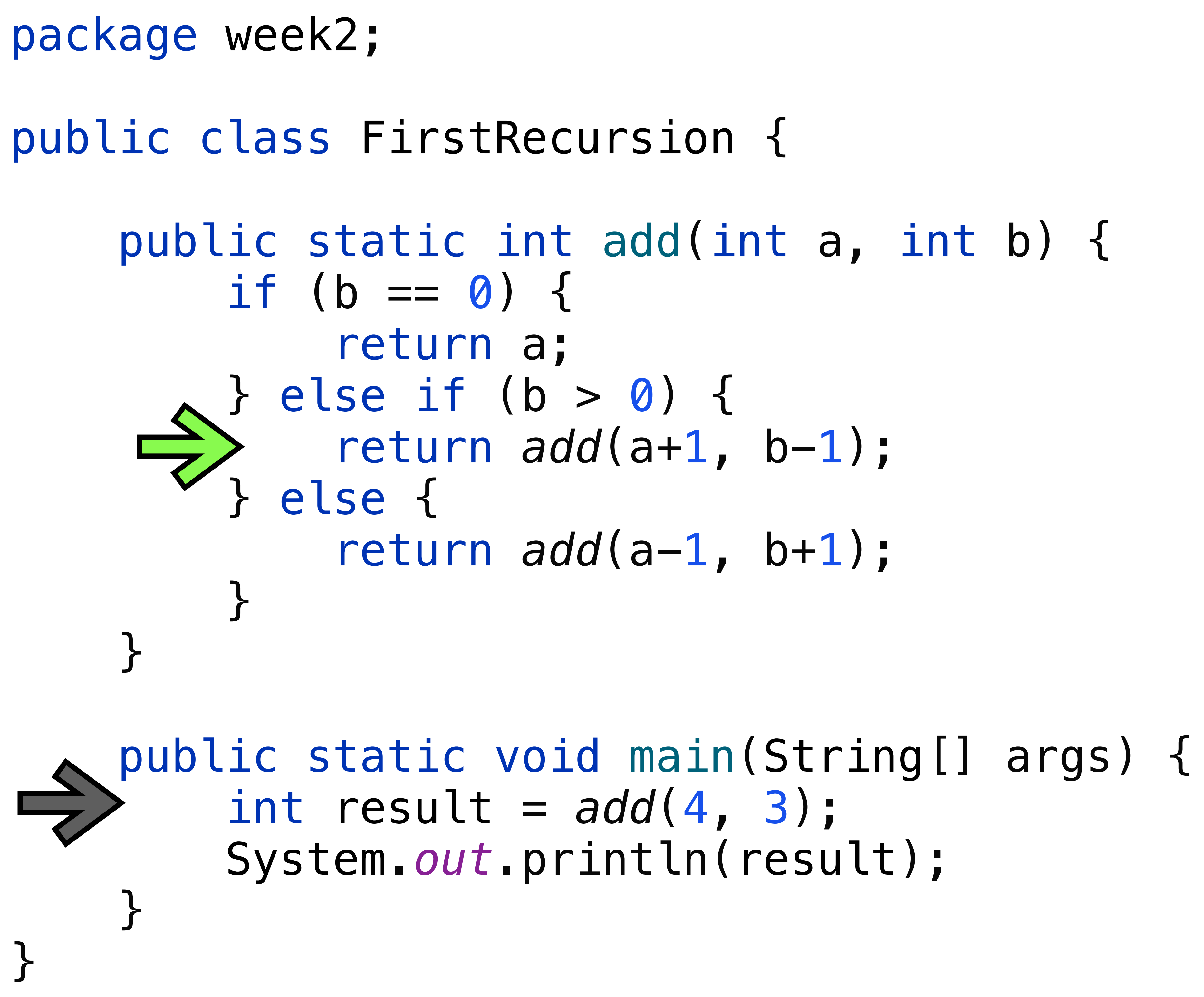
- This frame called add(6, 1)
  - The method call evaluated to 7



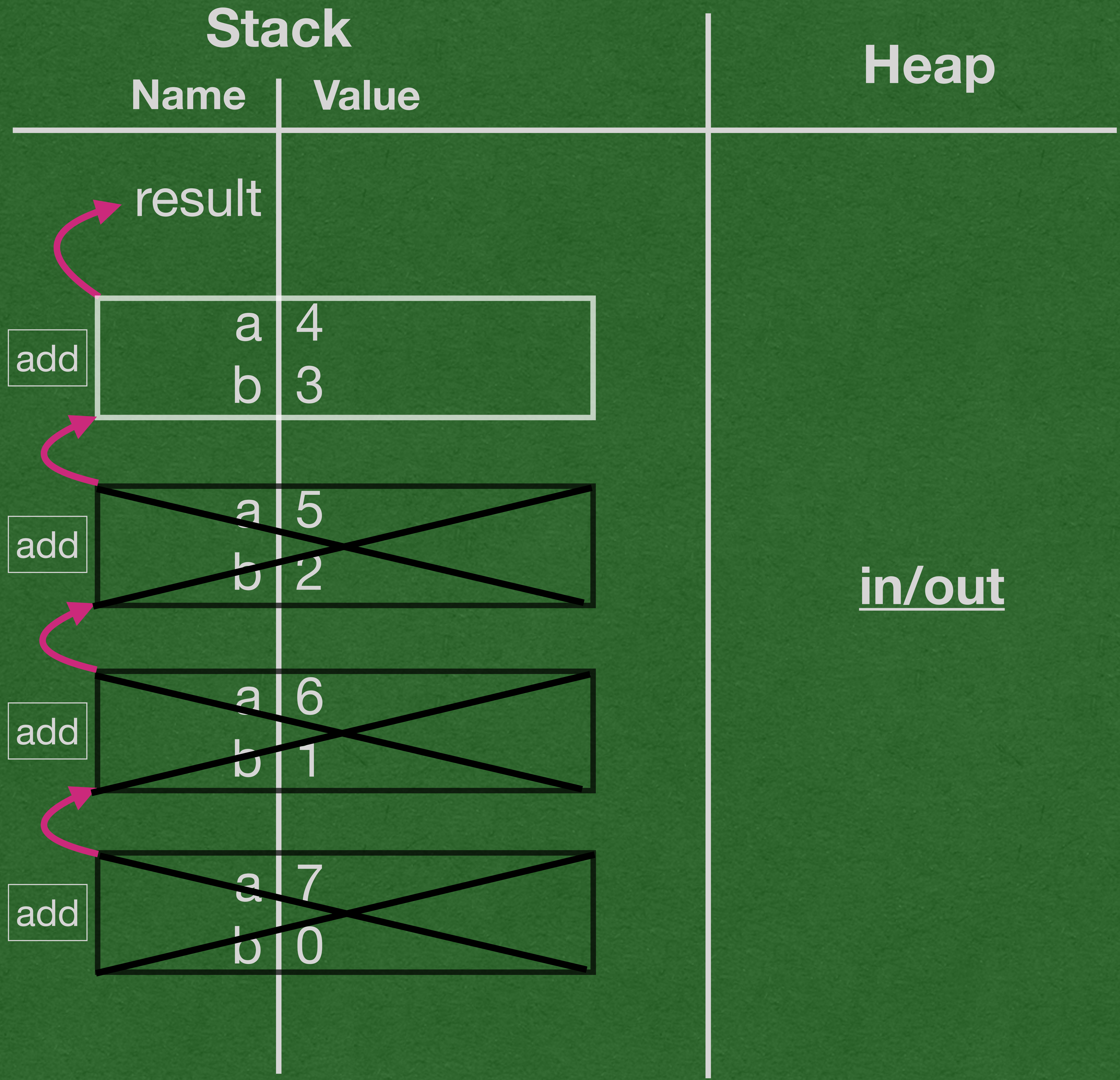
```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- Returns the value 7
- This frame is removed from the stack
- Control goes to the next frame on the stack

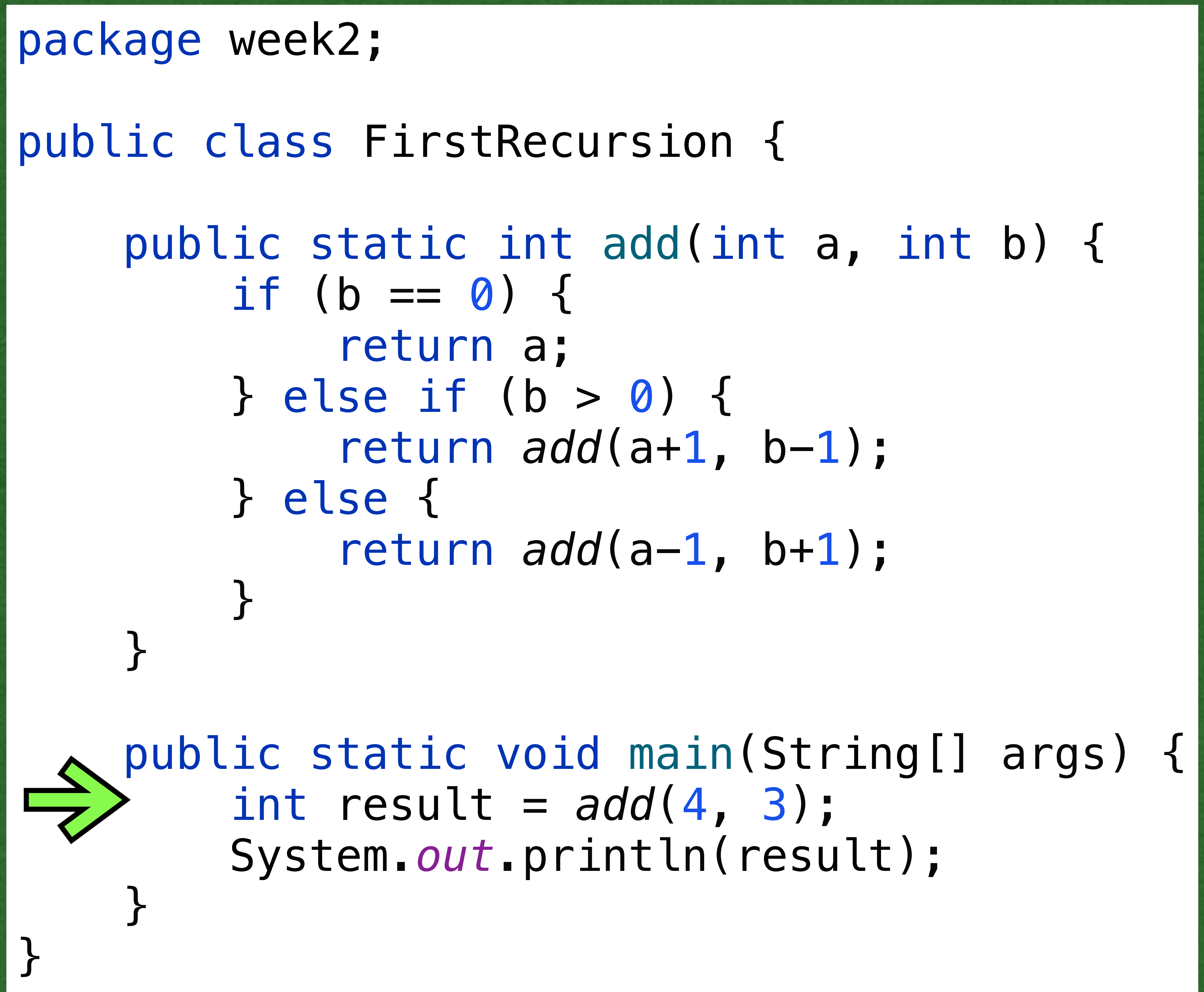




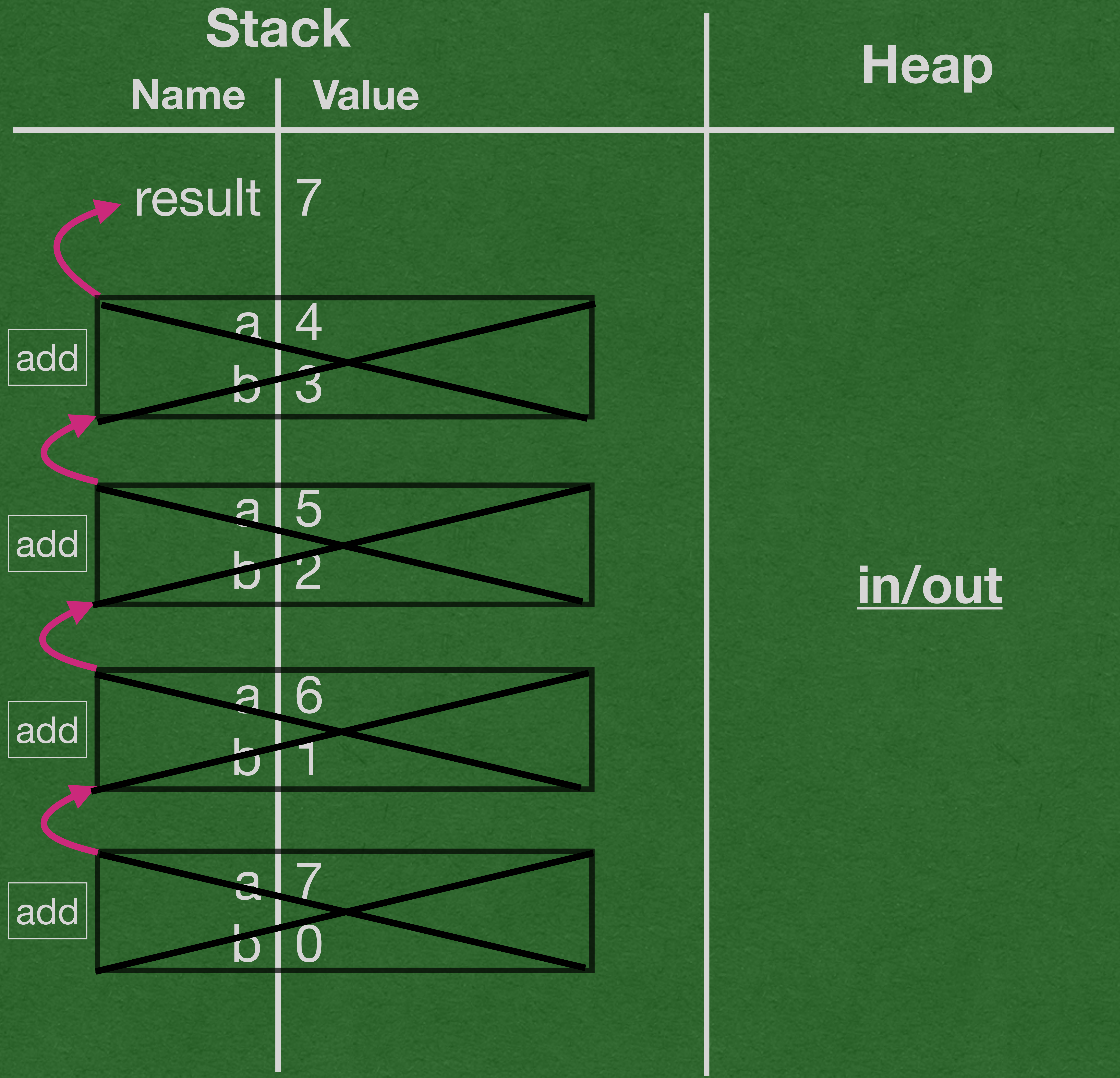
```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



- The next frame returns the value 7 that it received from the previous frame



```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



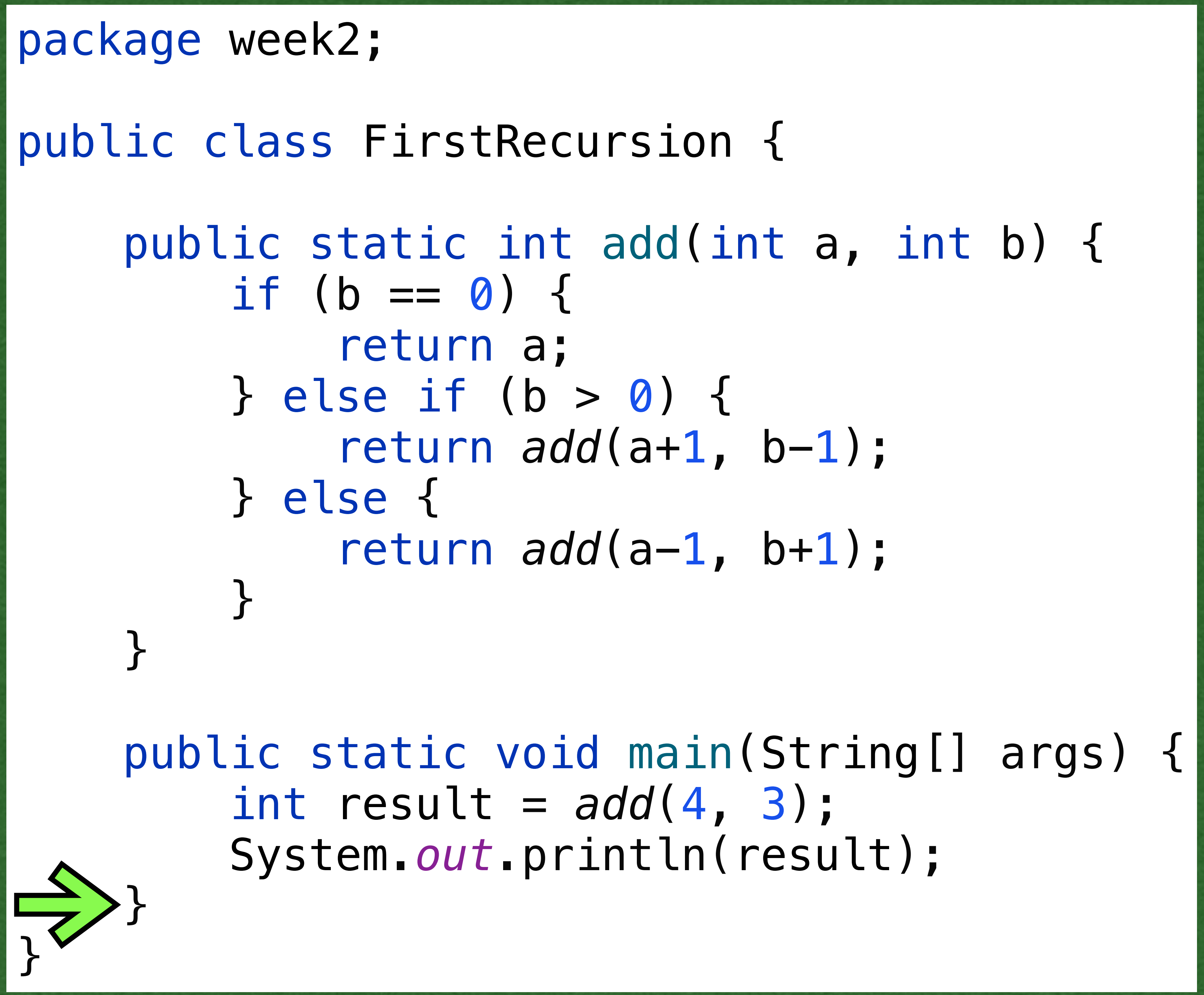
Stack

Name	Value
result	7
add a	4
b	3
add a	5
b	2
add a	6
b	1
add a	7
b	0

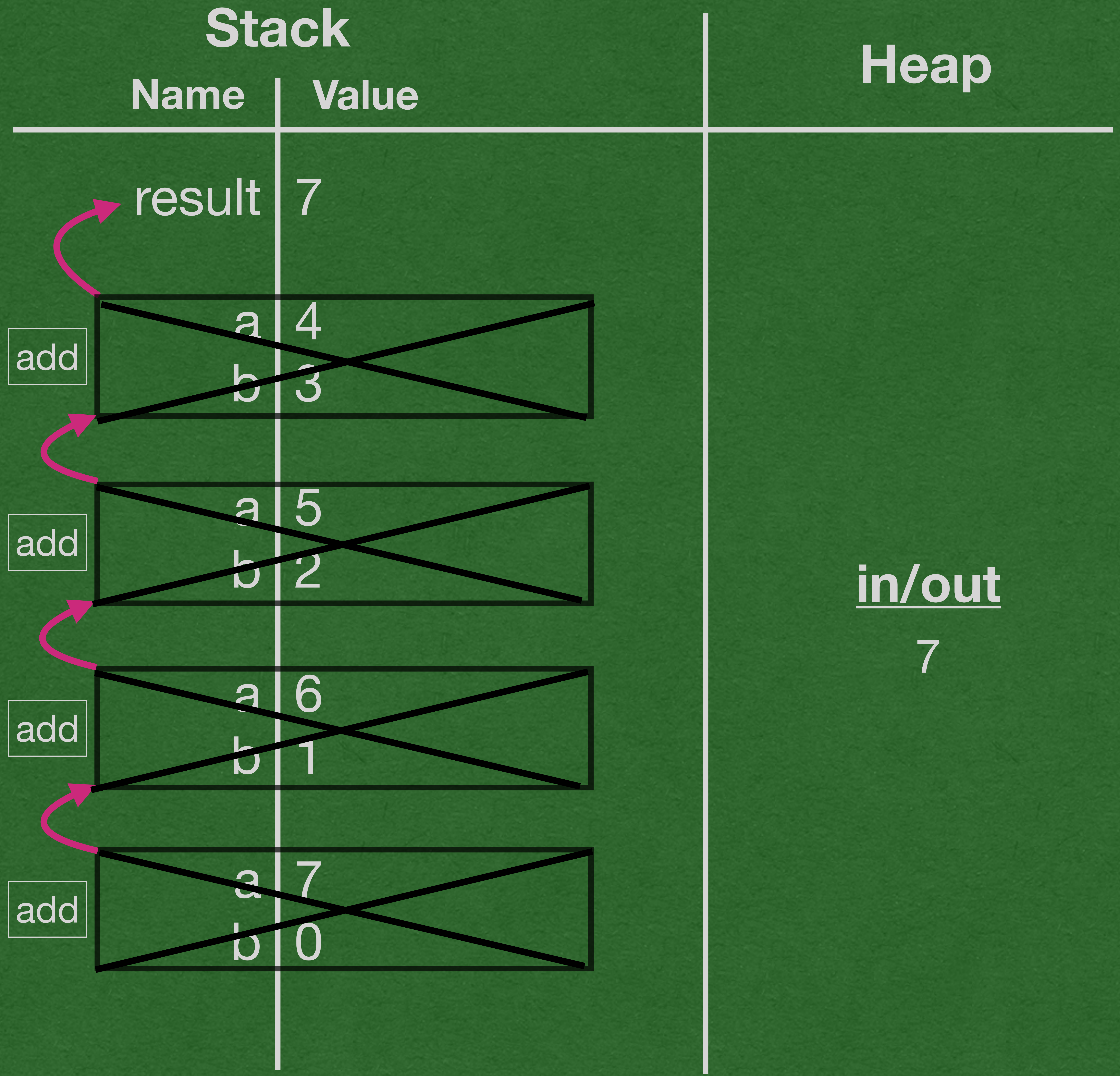
Heap

in/out

- Return 7 to the main stack frame



```
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



- Print 7 to the screen
- End the program

Stack
Heap
IO
Global Variables
Create Global Variable
Stack Frames
Create Heap Object
Create IO Line
Cross out

IO: 7

Name	Value
main	
result	7
add	
a	4
b	3
add	
a	5
b	2
add	
a	6
b	1
add	
a	7
b	0

```
package week3;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}

```

Stack
Heap
IO
Global Variables
Create Global Variable
Stack Frames
Create Heap Object
Create IO Line
Cross out

Name	Value
main	
result	7
add	
a	4
b	3
add	
a	5
b	2
add	
a	6
b	1
add	
a	7
b	0

7 ×

```java
package week3;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```