

# COMP 110/L Lecture 5

Maryam Jalali

Slides adapted from Dr. Kyle Dewey

# Outlines

- **Methods**
  - **Defining methods**
  - **Calling methods**

# Methods

# Motivation to Methods

- Real world programs often are large and complex
- Easier to manage in smaller pieces, in the case of Java, methods
  - Example of a “divide and conquer” strategy
  - Each method solves one small part of the entire problem
- Java standard library methods (built-in)
  - Have already been using these: `println()`, `nextInt()`, `pow()`

# Reasons for Using Methods (Modularization)

- **Divide-and-Conquer:** Build Java programs from small, simple pieces.
- **Software Reusability:** Use existing methods as building blocks to create new Java programs.
- **Avoid repeating Code**
- **Easier to Debug:** Each method can be debugged separately.
- **Easier to Maintain:** Can make changes to a specific method rather than the whole Java program.

# Basic Idea of a Method

- Consider mathematical functions:

$$y = f(x), \text{ where } f(x) = ?$$

- Need some definition for  $f(x)$ 
  - defines the value of  $f(x)$  for any value of  $x$
- $f(x)$  requires an argument, or parameter,  $x$
- $f(x)$  produces a value that is assigned to  $y$ 
  - Can use this method with any legal value substituted for  $x$ 
    - e.g.  $y = f(5)$
- Java methods work the same way
  - Of course, we must follow the Java syntax rule

# Motivation

**Input**

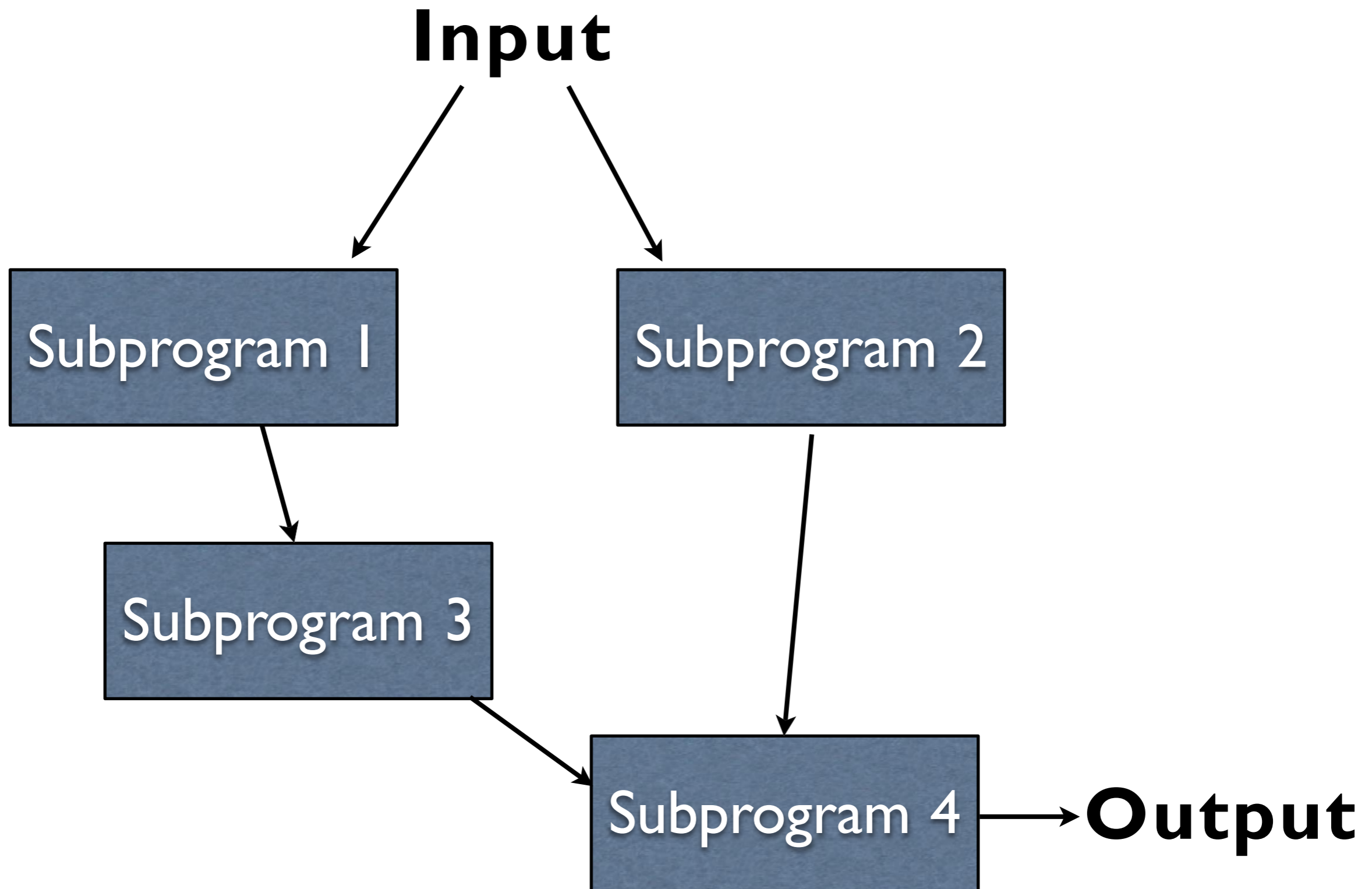


**Program**



**Output**

# Motivation





# Code Reuse

# Code Reuse

```
System.out.println(...)
```

# Code Reuse

```
System.out.println(...)  
    nextInt()
```

# Code Reuse

```
System.out.println(...)  
    nextInt()  
    nextLong()
```

# Code Reuse

```
System.out.println(...)  
    nextInt()  
    nextLong()  
    nextDouble()
```

# Code Reuse

```
System.out.println(...)  
    nextInt()  
    nextLong()  
    nextDouble()
```

**You have used all of these multiple times.**

# Code Reuse

```
System.out.println(...)  
    nextInt()  
    nextLong()  
    nextDouble()
```

You have used all of these multiple times.  
These are all *methods*.

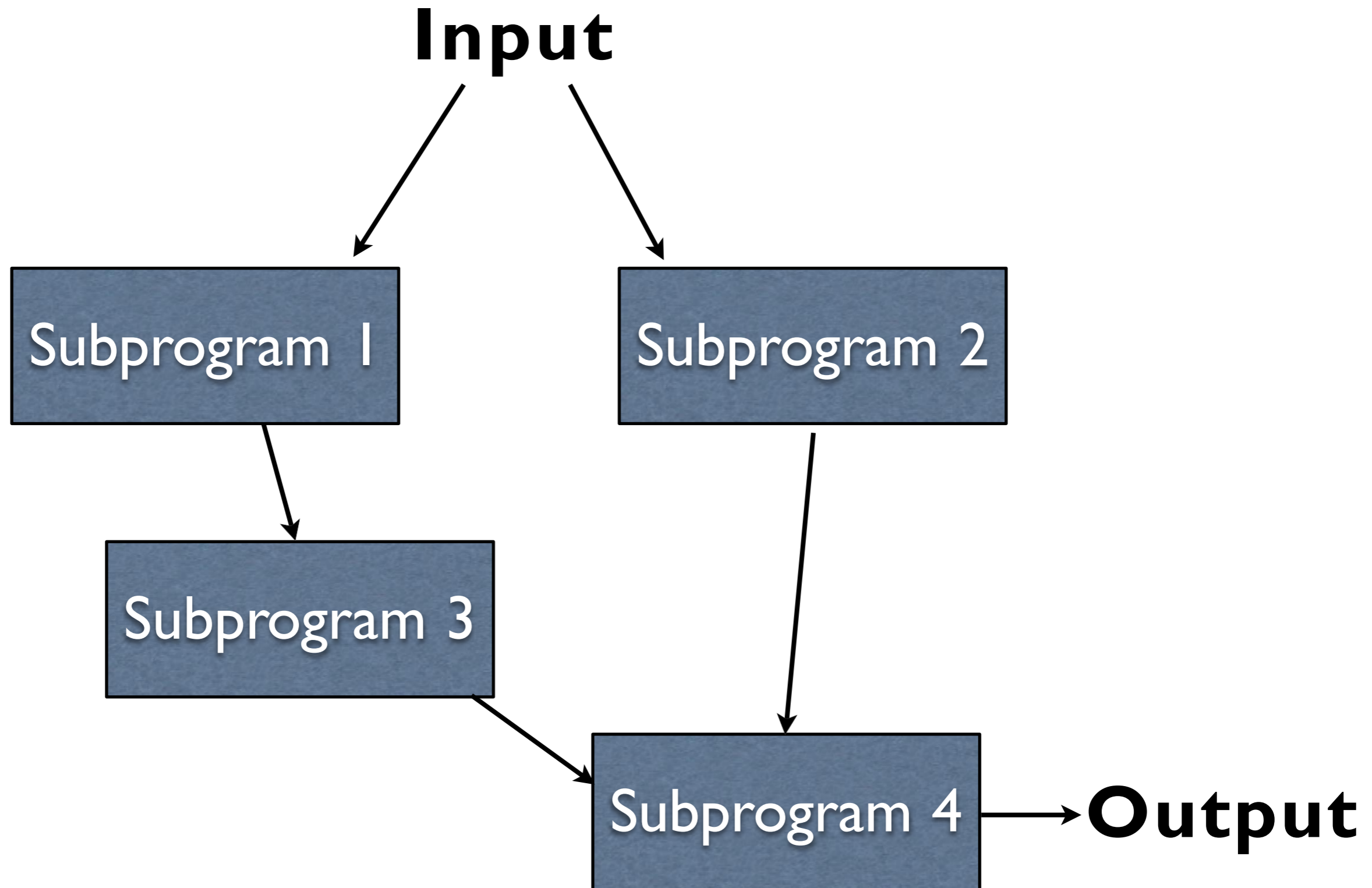
# Methods

Distinct subprograms.



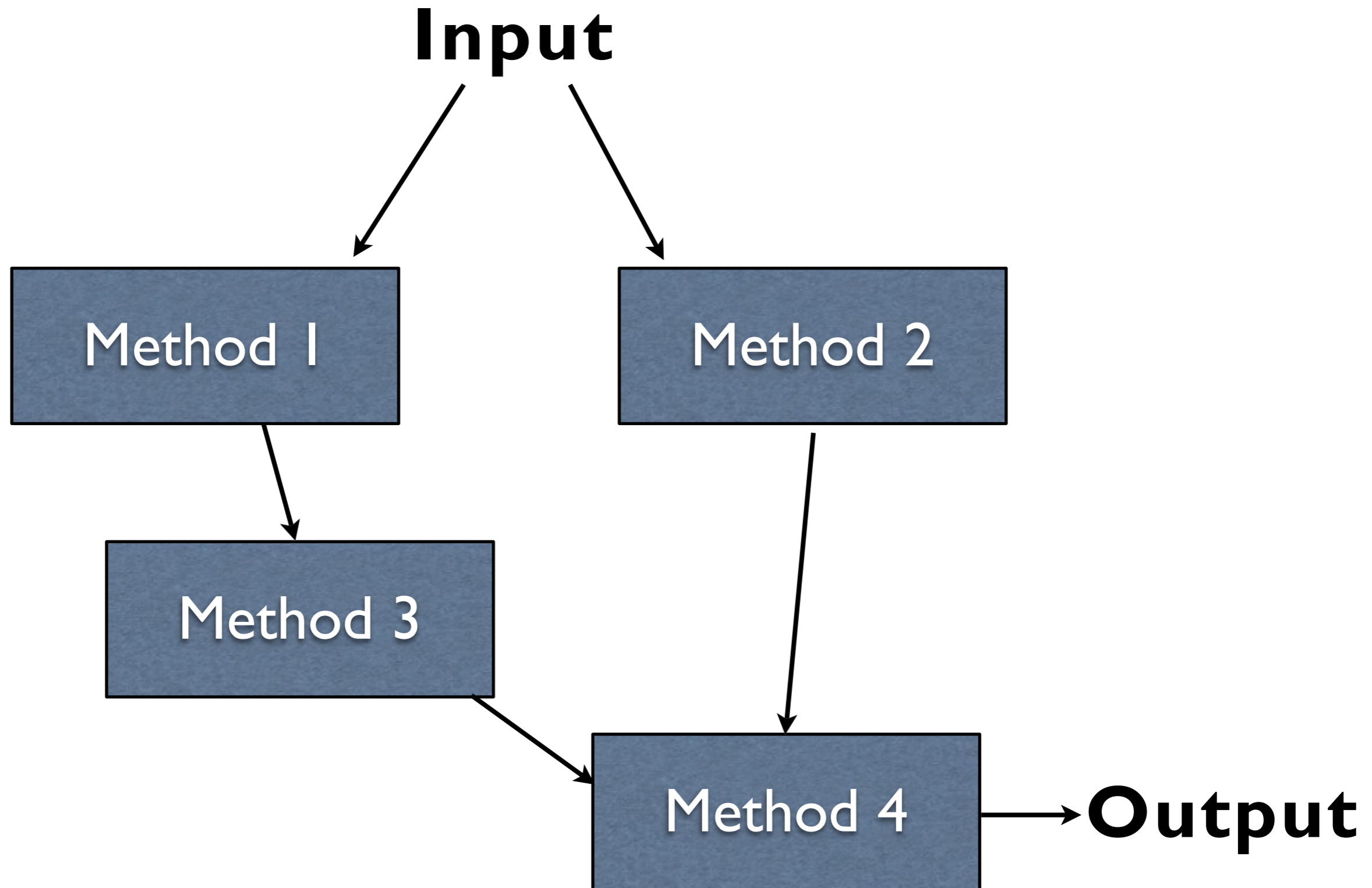
# Methods

Distinct subprograms.



# Methods

Distinct subprograms.



# Method Terminology

- We can *define* a method
  - Make it available to the rest of the program
- We can *call* a method
  - Execute the subprogram

# Elements of a Java Method

## **Method Definition:**

- 1- Declares the “signature” of the method  
Consists of return data type, method name, input parameters, Java operations
- 2- Reusable source code that can be called whenever needed.
- 3 - A sequence of instructions (code) that is packaged into a unit that can be reused.

## **Method Call:**

1. Actually makes **use** of the method
- 2. Real values** are specified for arguments

# Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.

# Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



# Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



```
System.out.println("Hello");
```

# Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



```
System.out.println("Hello");
```

One input, no outputs (cannot assign to a variable).



# Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



```
System.out.println("Hello");
```

One input, no outputs (cannot assign to a variable).

```
Math.pow(2, 3);
```

# Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



```
System.out.println("Hello");
```

One input, no outputs (cannot assign to a variable).

```
Math.pow(2, 3);
```

Two inputs, one output.

```
inputScanner.nextInt();
```

```
inputScanner.nextInt();
```

**No inputs, one output.**

```
inputScanner.nextInt();
```

**No inputs, one output.**

---

```
System.out.print("Goodbye");
```

```
inputScanner.nextInt();
```

**No inputs, one output.**

---

```
System.out.print("Goodbye");
```

**One input, no outputs (cannot assign to a variable)**

```
inputScanner.nextInt();
```

**No inputs, one output.**

```
System.out.print("Goodbye");
```

**One input, no outputs (cannot assign to a variable)**

```
inputScanner.nextLong();
```

```
inputScanner.nextInt();
```

**No inputs, one output.**

```
System.out.print("Goodbye");
```

**One input, no outputs (cannot assign to a variable)**

```
inputScanner.nextLong();
```

**No inputs, one output.**



```
inputScanner.nextInt();
```

**No inputs, one output.**

```
System.out.print("Goodbye");
```

**One input, no outputs (cannot assign to a variable)**

```
inputScanner.nextLong();
```

**No inputs, one output.**

```
inputScanner.nextDouble();
```

```
inputScanner.nextInt();
```

**No inputs, one output.**

```
System.out.print("Goodbye");
```

**One input, no outputs (cannot assign to a variable)**

```
inputScanner.nextLong();
```

**No inputs, one output.**

```
inputScanner.nextDouble();
```

**No inputs, one output.**

# Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from

# Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from

Method 1

Method 2

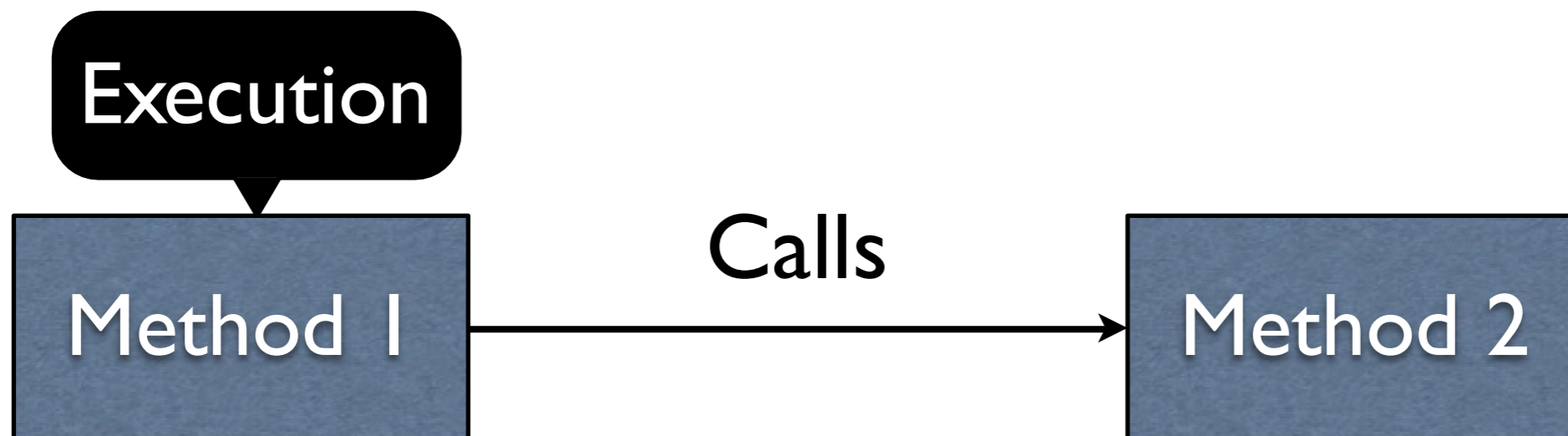
# Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



# Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



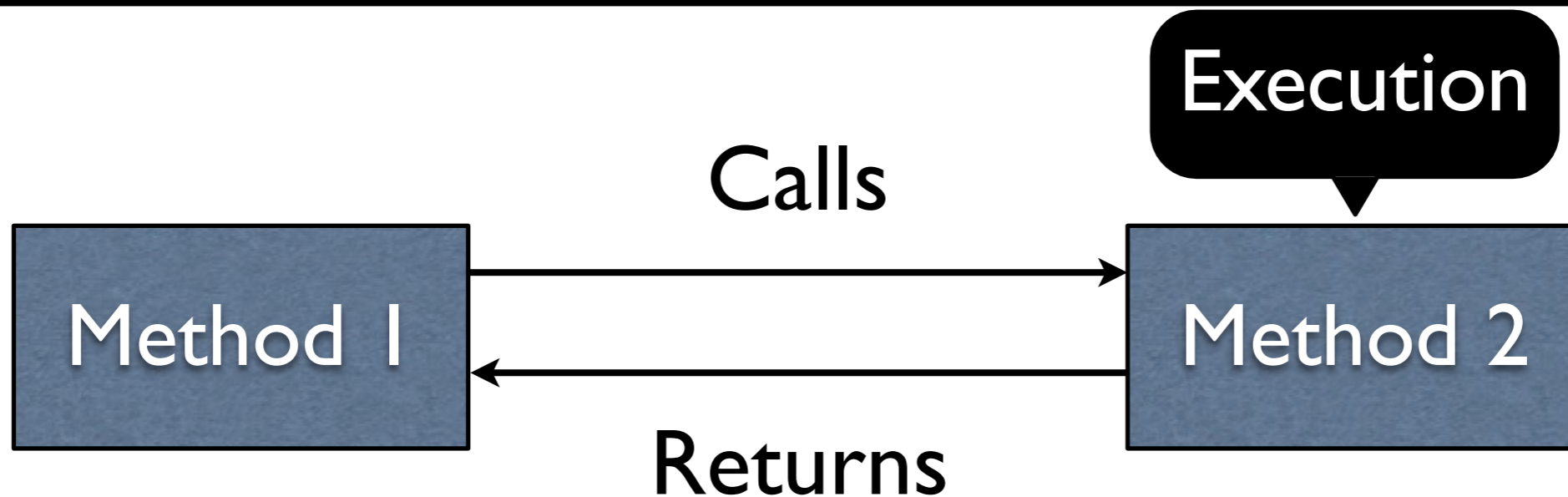
# Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



# Calling Methods

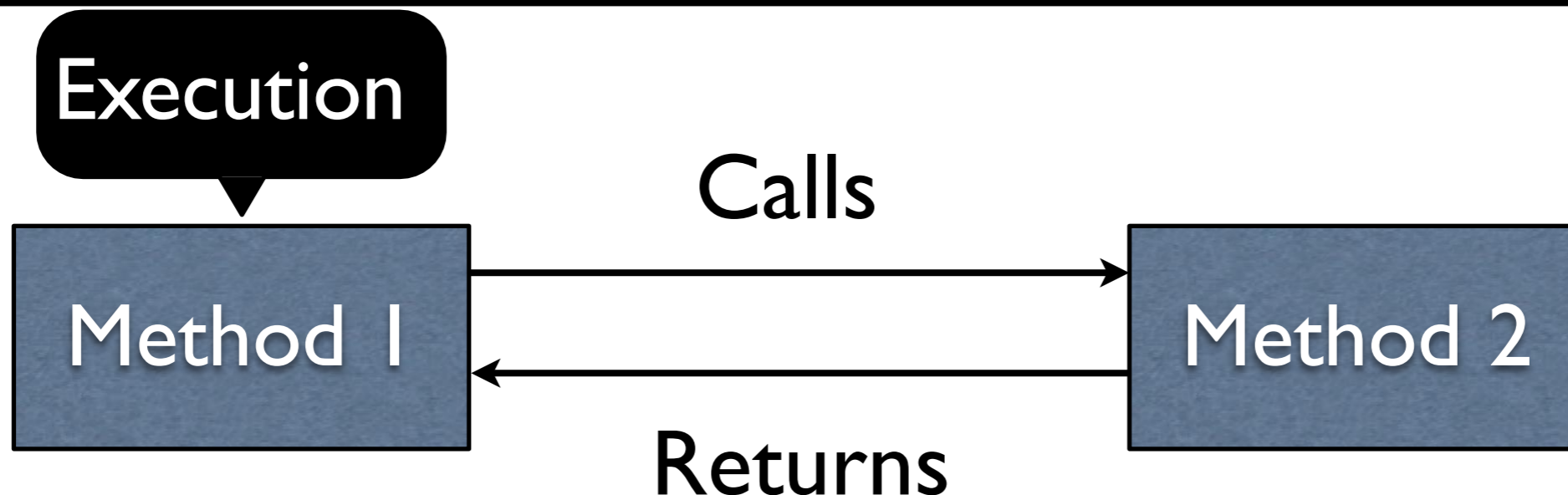
- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from





# Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



```
class Main {  
    public static void main(String[] args) {  
        ... ..  
        myFunction();  
        ... ..  
    }  
  
    private static void myFunction() {  
        // function body  
        ... ..  
        ... ..  
    }  
}
```

1

2

3

# Defining a Method

Easiest to see with real code.

**Example:**

```
Return42.java
```

# Method Parameters

Parameters are *passed* on a call,  
copying their values into the called method.

# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

---

```
public static int foo(int x) {  
    return x + 1;  
}
```

# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

```
public static int foo(int x) {  
    return x + 1;  
}
```

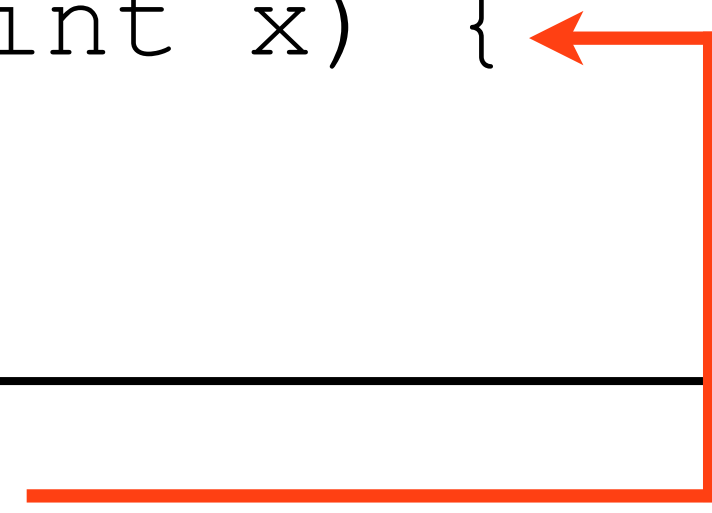
```
int a = foo(7);
```

# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

```
public static int foo(int x) {  
    return x + 1;  
}
```

```
int a = foo(7);
```

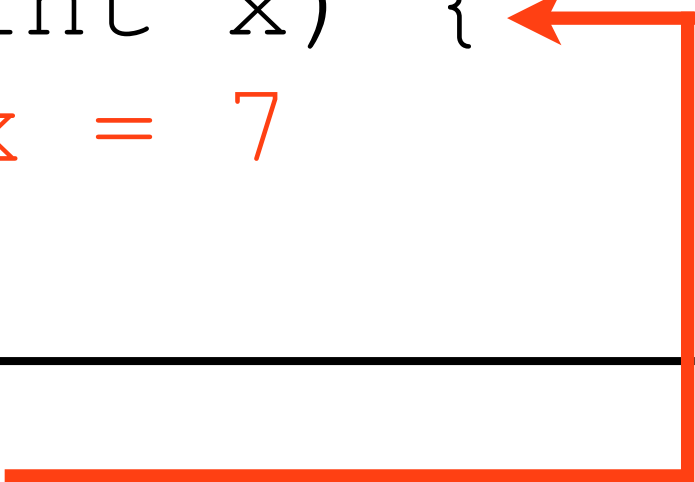


# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

```
public static int foo(int x) {  
    return x + 1;           x = 7  
}
```

```
int a = foo(7);
```





# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

```
public static int foo(int x) {  
    return x + 1;           x = 7  
}
```

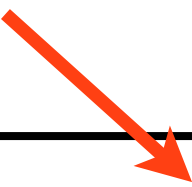
```
int a = foo(7);
```

# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

```
public static int foo(int x) {  
    return x + 1;           x = 7  
}
```

```
int a = foo(7);
```



# Method Parameters

Parameters are *passed* on a call, copying their values into the called method.

```
public static int foo(int x) {  
    return x + 1;           x = 7  
}
```

```
int a = foo( );  
       7   8
```

**Example:**

ReturnParameter.java

**Example:**

`MultParameters1.java`

**Example:**

`MultParameters2.java`

# Example:

`MultParameters3.java`

# Method Definition

## General Form

---

```
public static  
returnType  
methodName(parameter_list) {  
    ...  
    return expression;  
}
```



# Method Definition

## General Form

---

```
public static
```

Magic

```
returnType
```

```
methodName (parameter_list) {
```

```
    ...
```

```
    return expression;
```

```
}
```

# Method Definition

## General Form

```
public static
```

Magic

```
returnType
```

Type of value produced

```
methodName (parameter_list) {
```

```
...
```

```
return expression;
```

```
}
```

# Method Definition

## General Form

```
public static  
returnType  
methodName (parameter_list) {  
    ...  
    return expression;  
}
```

Magic

Type of value produced

Name given to  
method; same naming  
rules as variables

# Method Definition

## General Form

```
public static  
returnType  
methodName (parameter_list) {  
    ...  
    return expression;  
}
```

Magic

Type of value produced

Inputs to  
method  
(int x)

Name given to  
method; same naming  
rules as variables

# Method Definition

## General Form

```
public static  
returnType  
methodName (parameter_list) {  
    ...  
    return expression;  
}
```

Magic

Type of value produced

Inputs to  
method  
(int x)

Method ends  
here, evaluates  
expression, and  
produces its result

Name given to  
method; same naming  
rules as variables

# Methods which Produce no Values

Methods which produce no values  
have a `void` return type

**Example:**

```
ReturnNothing.java
```

# Aside: Expressions vs. Statements

- Expressions return values (e.g.,  $1 + 2$ )
- Statements do not return values (e.g., `System.out.println("Hello")`)
- Statements are separated with semicolon (;)

```
System.out.println("Hello");  
System.out.println("Goodbye");
```

# `main` Method

`main` is just another method.

`main` serves as the entry point to your program.



# main Method

`main` is just another method.

`main` serves as the entry point to your program.

---

```
public static  
void  
main (String[] args) {  
    ...  
}
```