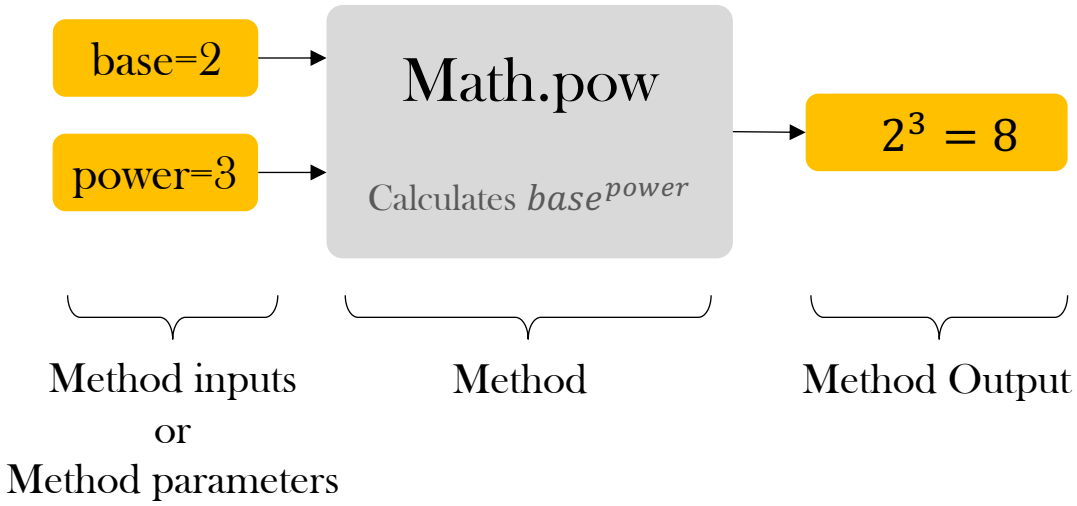
Methods

Methods

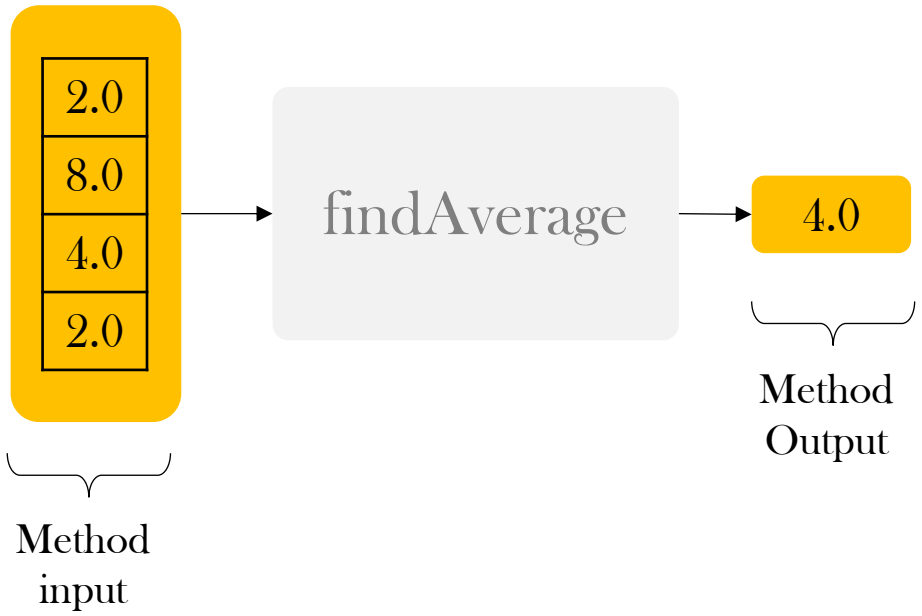
- A method is a collection of statements that are grouped together to perform an operation
- It has inputs and a single output




```
double base = 2.0;
int power = 3;

double result = Math.pow(base, power);

System.out.println("Result is: " + result);
```

Methods

- Example: Find the average of array elements



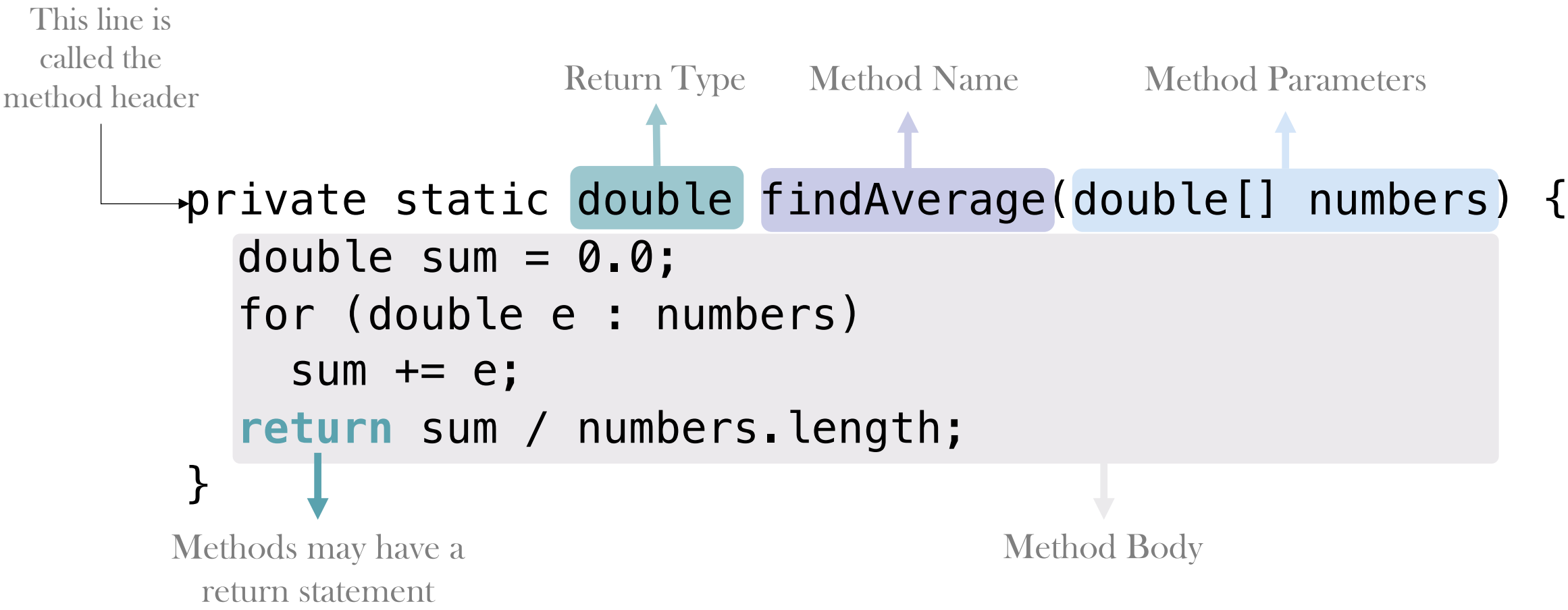

```
double[] numbers = {2.0, 8.0, 4.0, 2.0};

double result = findAverage(numbers);

System.out.println("Result is: " + result);
```

Methods

- What is the code inside the findAverage method?

```
public class AppMethods {
 public static void main(String[] args) {
  double[] numbers = {2.0, 8.0, 4.0, 2.0};
  double result = findAverage(numbers);
  System.out.println(result);
 }

 private static double findAverage(double[] numbers) {
  double sum = 0.0;
  for (double e : numbers)
   sum += e;
  return sum / numbers.length;
 }

}
```

Method Structure



Methods

- Method header is the first line in the method definition
- Method signature is the combination of the method name and the parameter list
- When a method is called, you pass a value to the parameter
  - This value is the actual parameter or argument
- A method may return a value
  - The return value type is the data type returned
- If a method does not return a value, use void as the return type
  - You do not need to write return statement for void methods

Method Calls

- When a method is called, argument values are passed to method parameters
- Method's return value is passed to the output variable

```
public class AppMethods {
 public static void main(String[] args) {
  int input1 = 5;
  int input2 = 8;
  // argument values are input1=5 and input2=8
  int o = max(input1,input2); // output variable is o
  System.out.println("Bigger number is: " + o);
 }
 // argument values are passed to a and b
 // a gets the value of input1
 // b gets the value of input2
 private static int max(int a, int b) {
  int result = a;
  if (b > a)
   result = b;
  return result;
 }
}
```

Method Calls

- When a method is called, argument values are passed to method parameters
- Method's return value is passed to the output variable

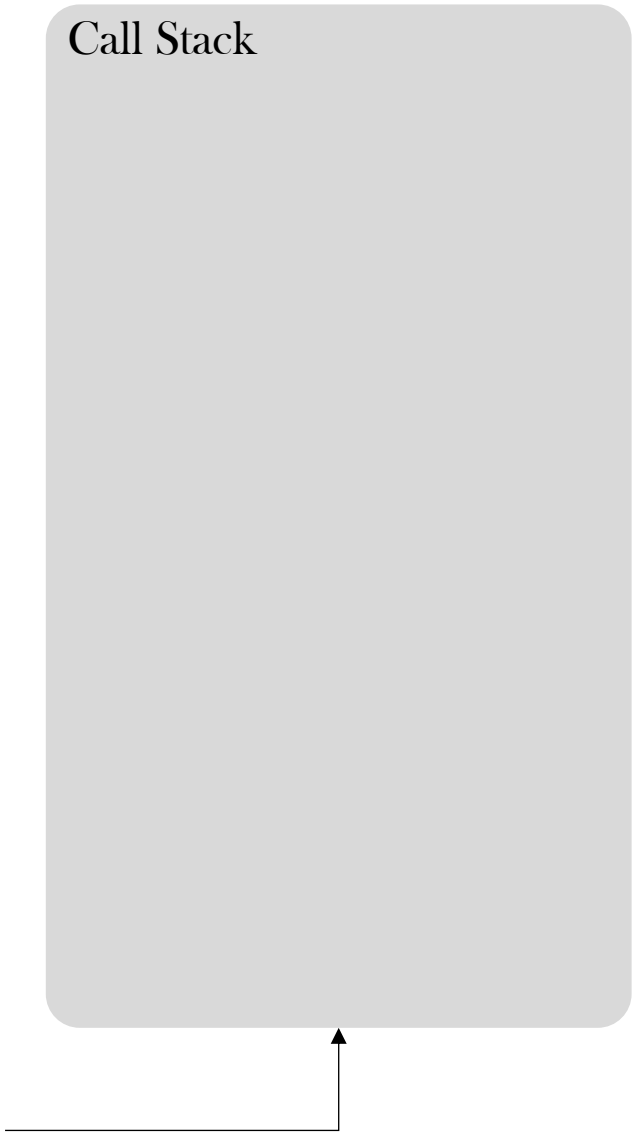
```
int input1 = 5;
int input2 = 8;

int o = max(input1,input2);

private static int max(int a, int b) {
 int result = a;
 if (b > a)
  result = b;
 return result;
}
```

Argument values

Method parameters

Method Calls

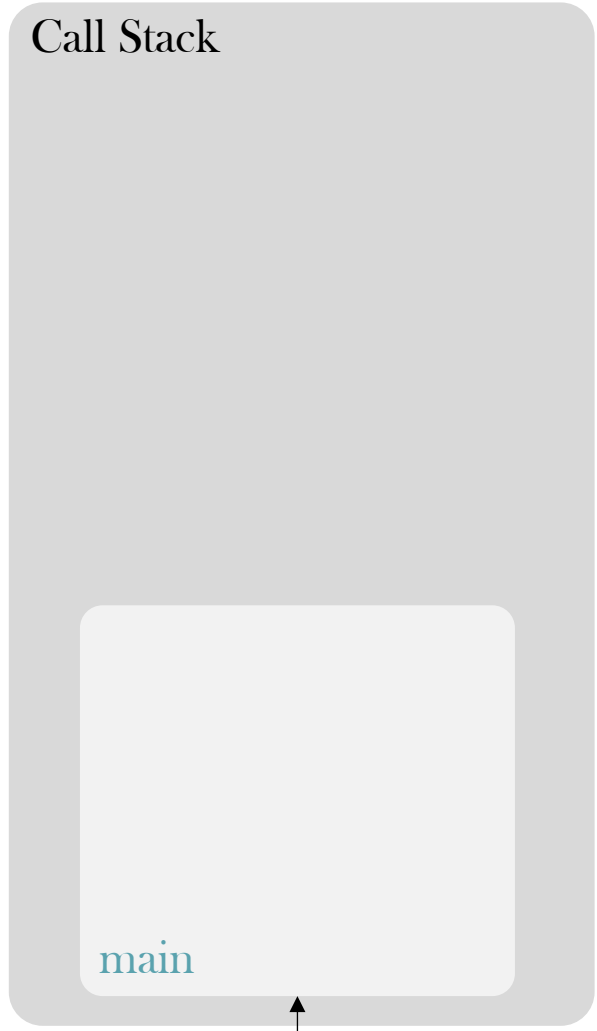
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



Call stack is a special region in memory used to run Java programs. Methods and variables are stored in call stack.

Method Calls

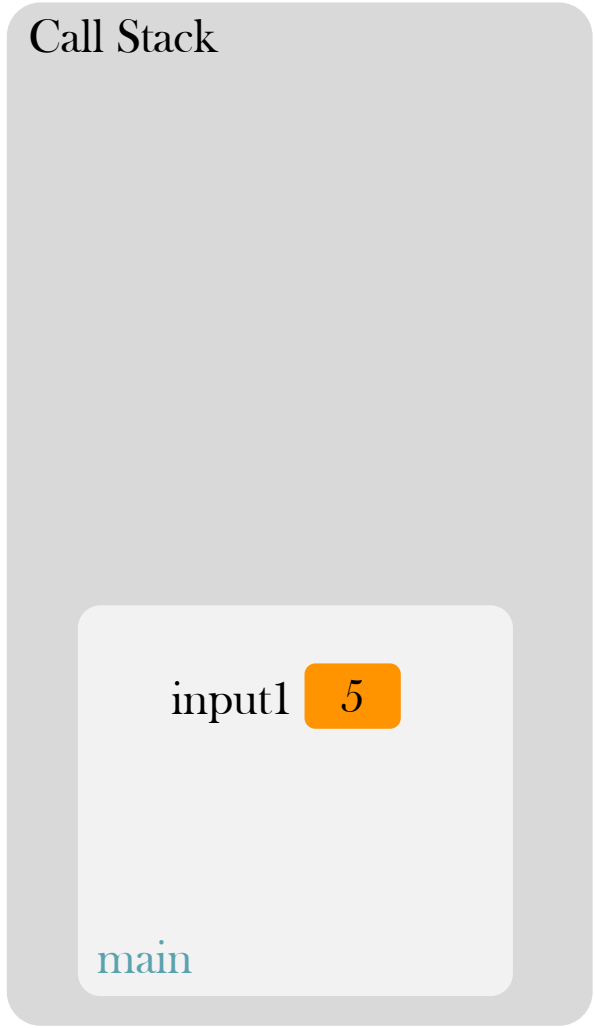
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



This box represents the activation record for the main function. Activation records are stored in call stack

Method Calls

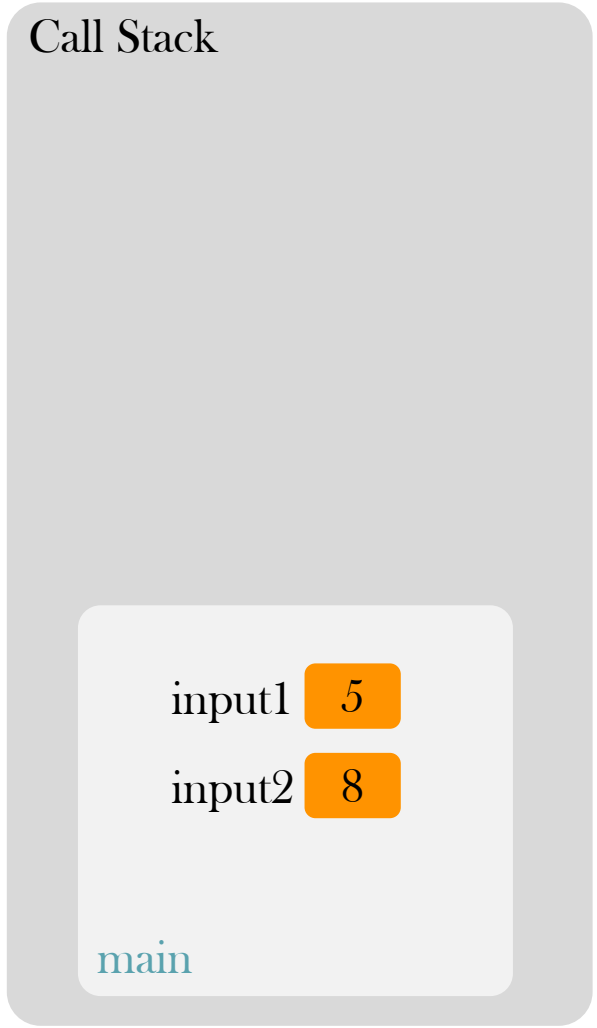
```
 1 public class AppMethods {
 2  public static void main(String[] args) {
 3   int input1 = 5;
 4   int input2 = 8;
 5   int o = max(input1,input2);
 6   System.out.println("Bigger number is: " + o);
 7  }
 8
 9  private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



input 1 variable is now 5

Method Calls

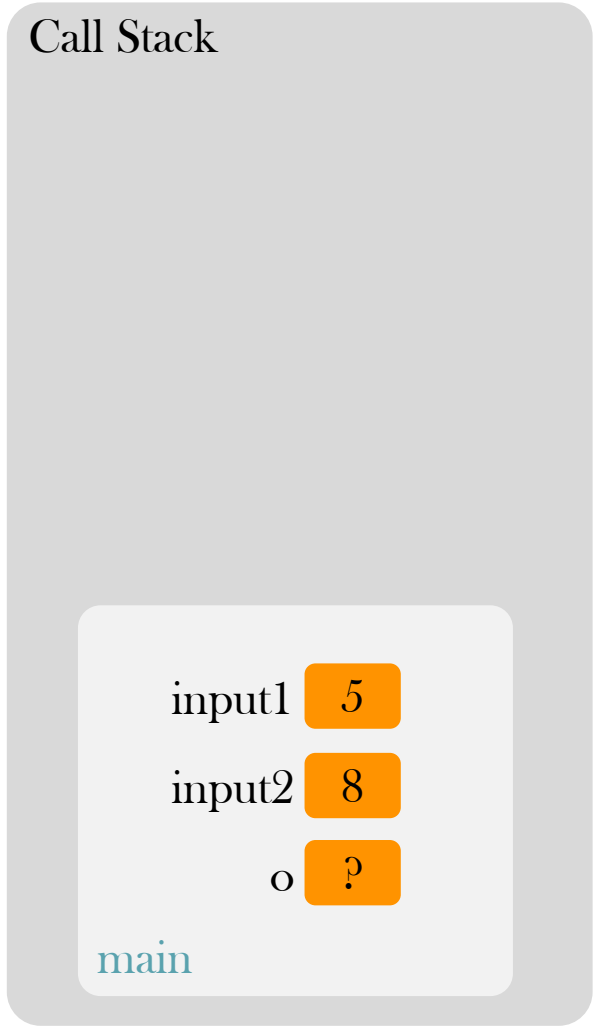
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



input 2 variable is now 8

Method Calls

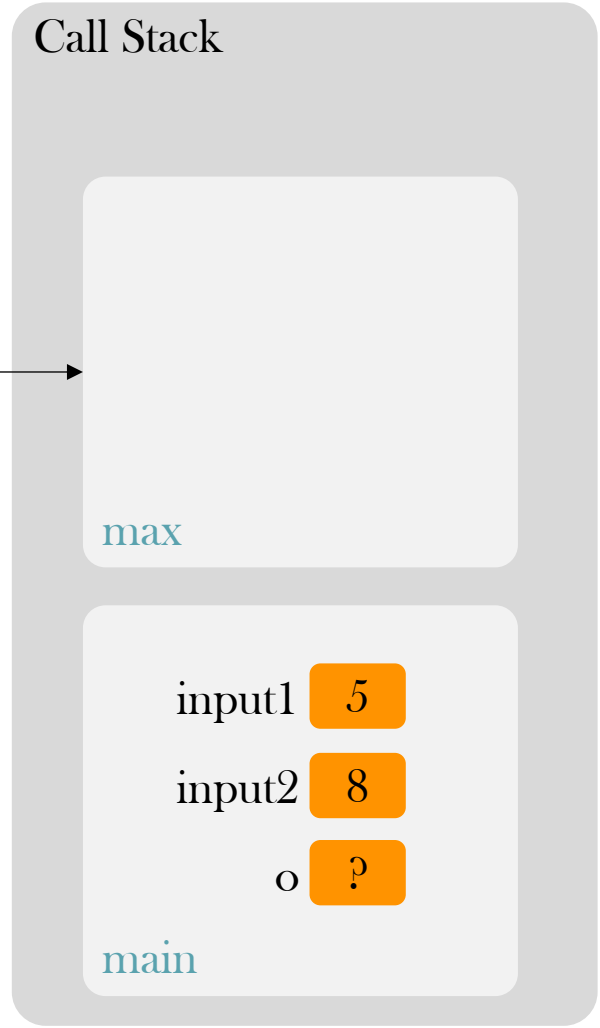
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



o variable waits for the output returned by the method

Method Calls

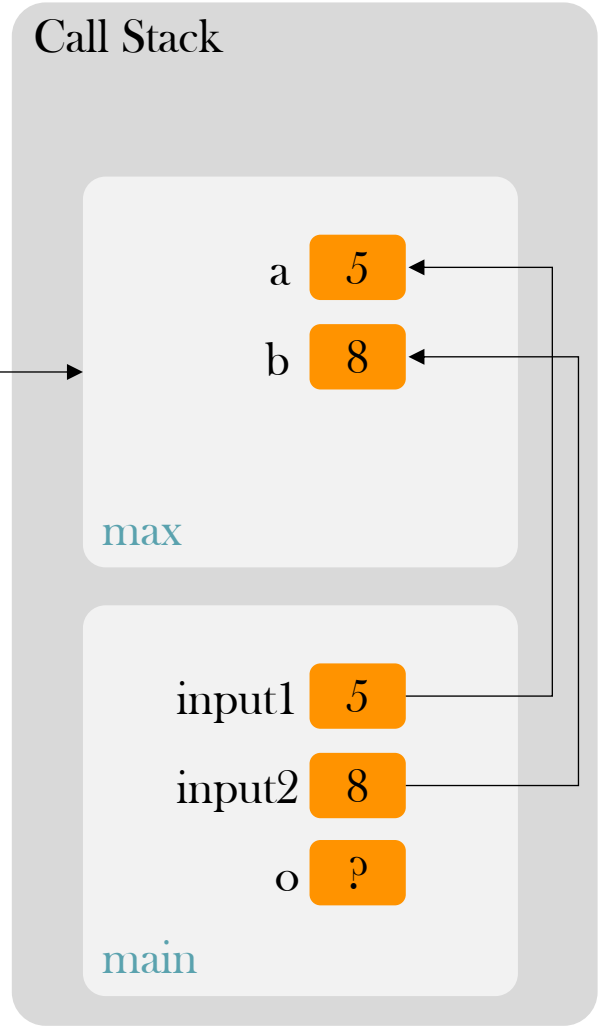
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



When a method is called, its activation record is created in the call stack

Method Calls

```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```

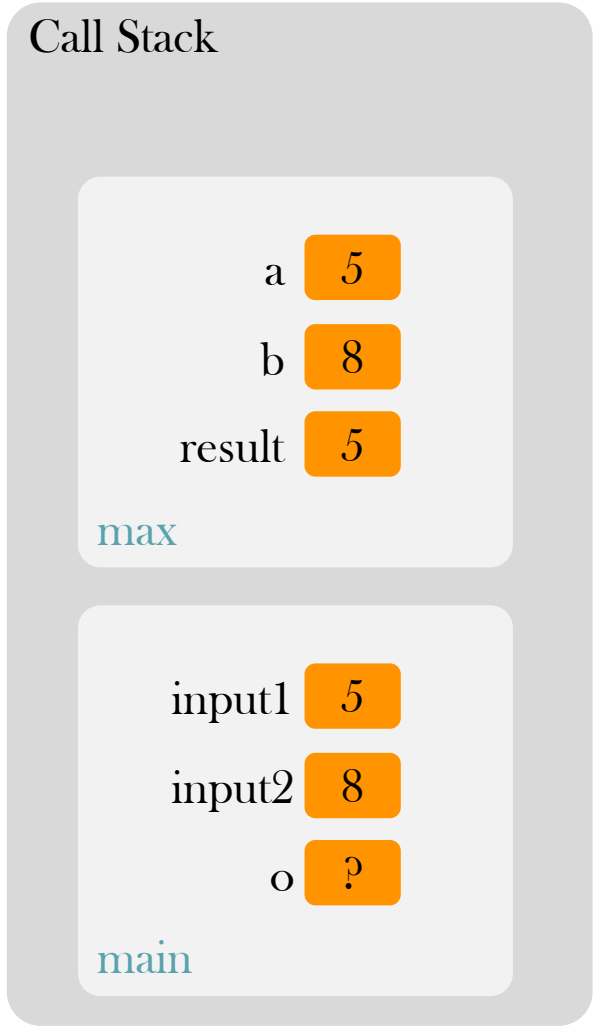


When a method is called, its activation record is created in the call stack

Argument values (input 1 and input2) are assigned to method parameters a and b

Method Calls

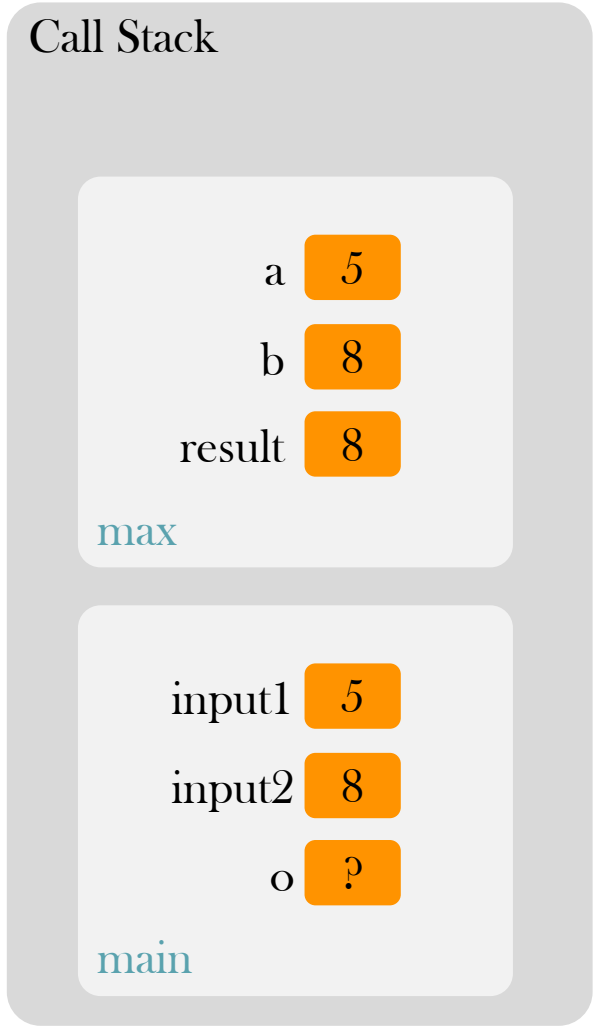
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



result variable is now 5

Method Calls

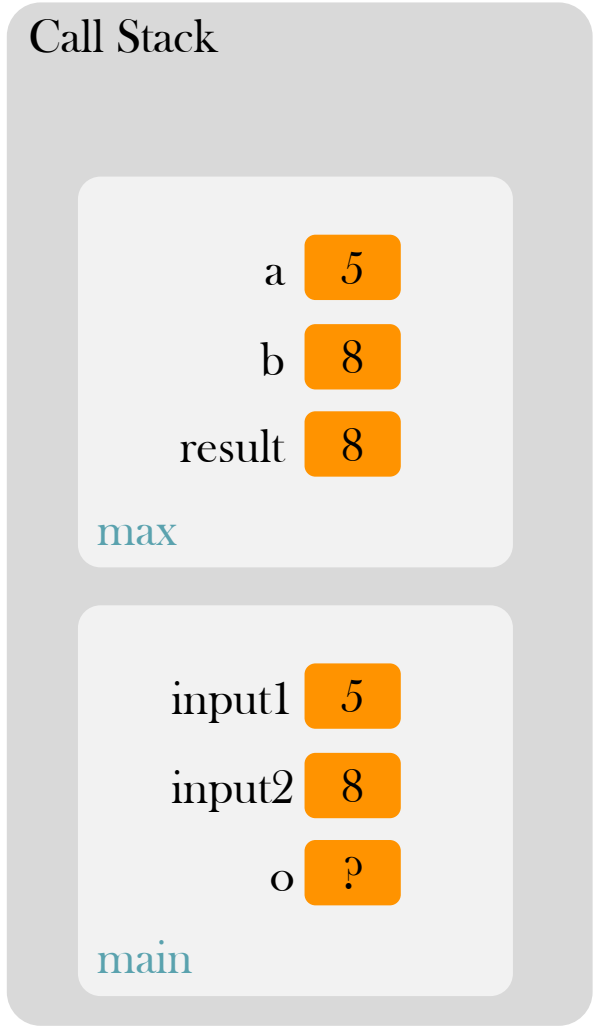
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12     result = b;
13   return result;
14  }
15 }
```



result variable is now 8

Method Calls

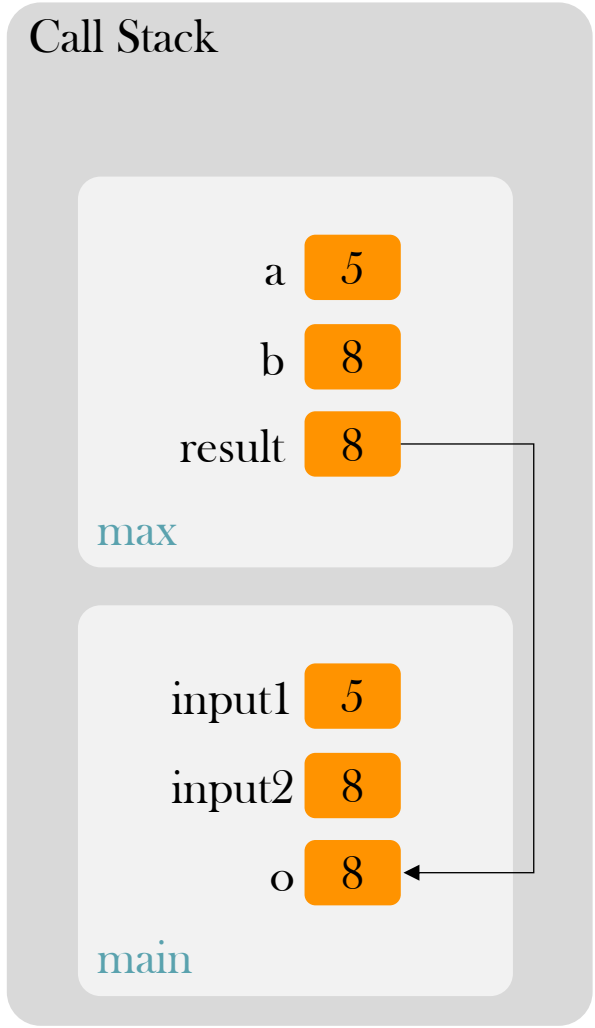
```
 1 public class AppMethods {
 2  public static void main(String[] args) {
 3   int input1 = 5;
 4   int input2 = 8;
 5   int o = max(input1,input2);
 6   System.out.println("Bigger number is: " + o);
 7  }
 8
 9  private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



Method will return 8

Method Calls

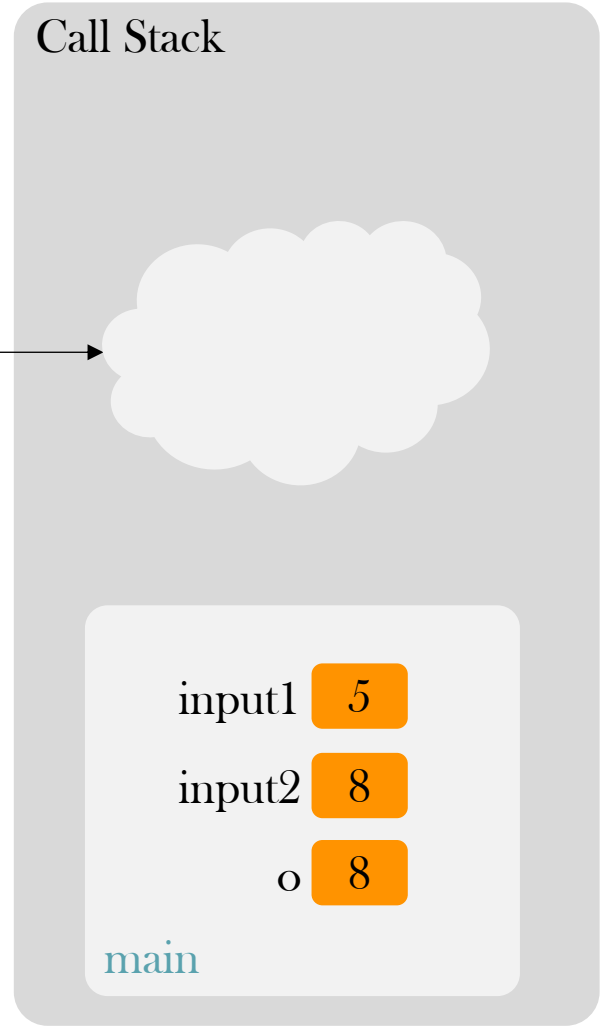
```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



Method's return value is assigned to variable o

Method Calls

```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```

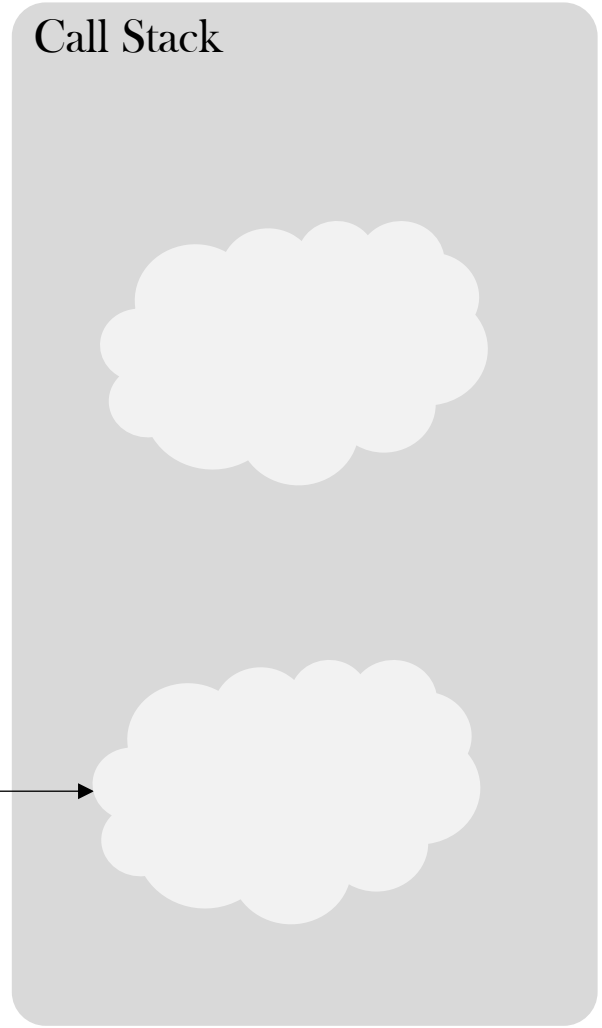


When a method call is completed, its activation record is removed from the call stack:
max's activation record is deleted from the memory

Print 8

Method Calls

```
1  public class AppMethods {
2   public static void main(String[] args) {
3    int input1 = 5;
4    int input2 = 8;
5    int o = max(input1,input2);
6    System.out.println("Bigger number is: " + o);
7   }
8
9   private static int max(int a, int b) {
10   int result = a;
11   if (b > a)
12    result = b;
13   return result;
14  }
15 }
```



When a method call is completed, its activation record is removed from the call stack:
main's activation record is deleted from the memory

main method is finished

Call Stack Example

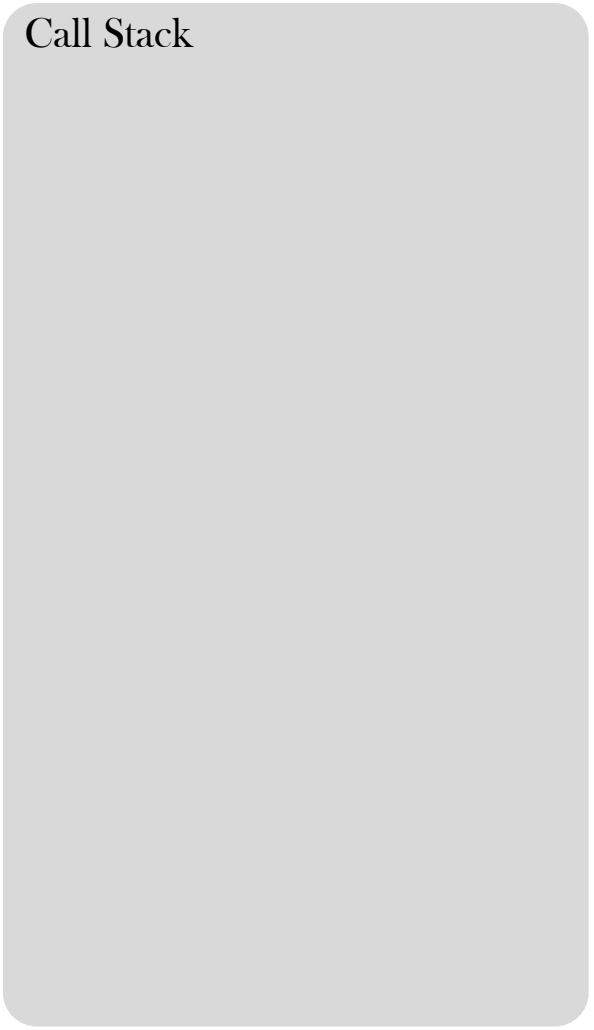
- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```



Initially, call stack is empty

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

main method is called

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

a is assigned to 3.0

Call Stack Example

- What is the output of the following program?

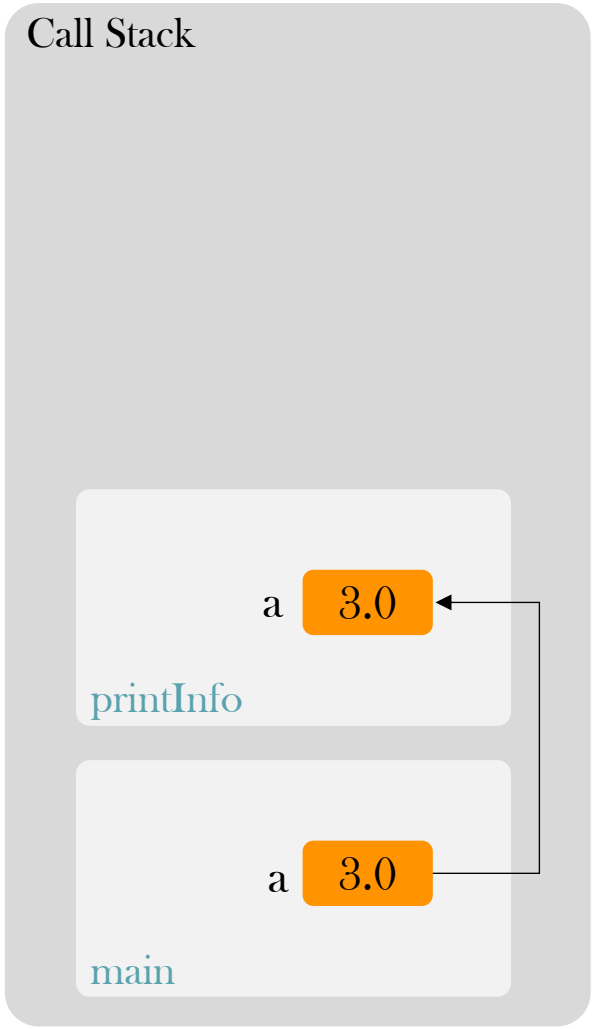
```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

printInfo is called

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```



a variable in printInfo gets 3.0

Call Stack Example

- What is the output of the following program?

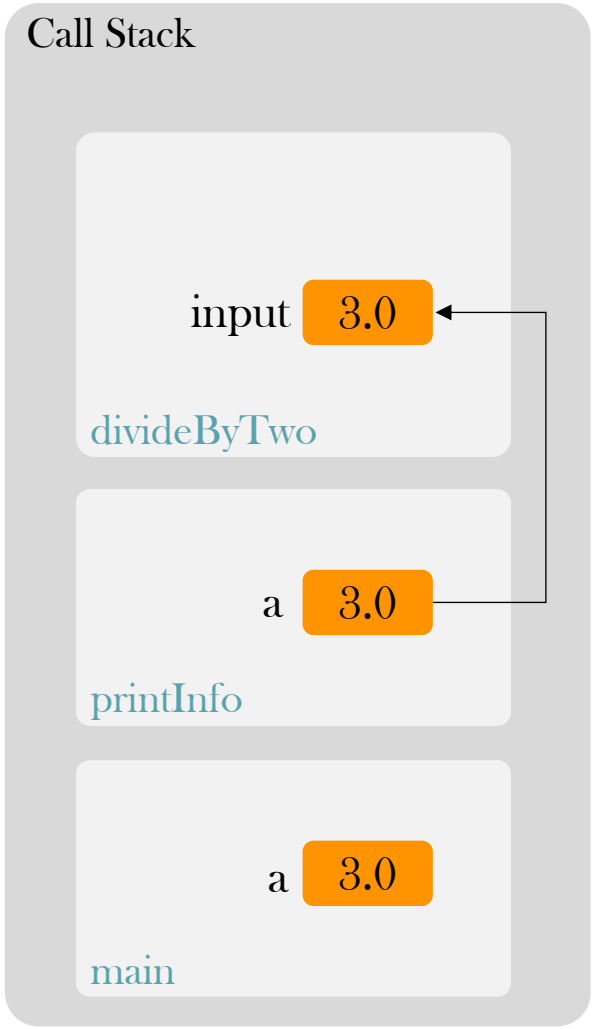
```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

divideByTwo is called

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

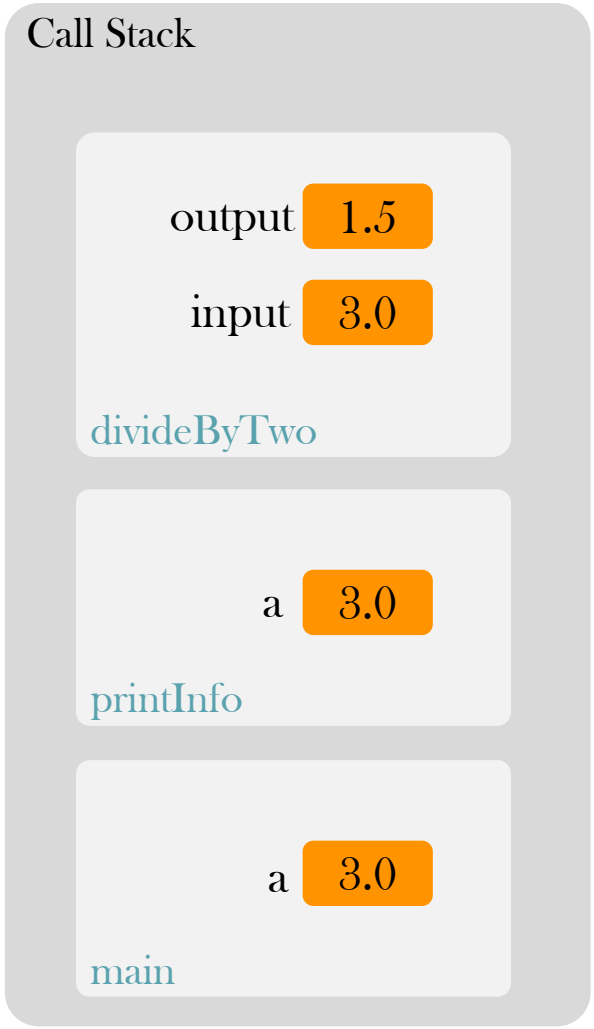


input variable gets 3.0

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

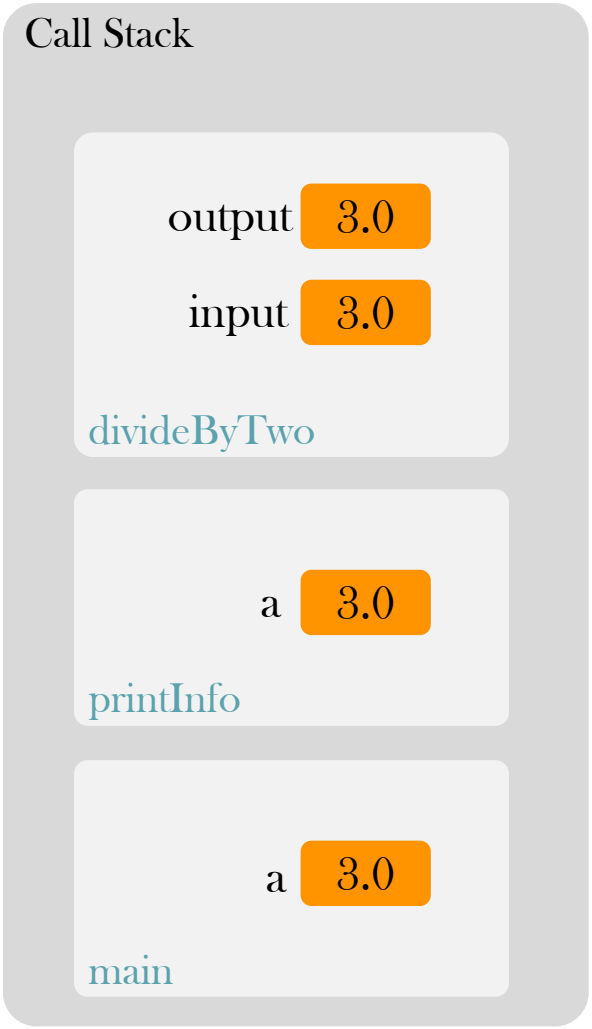


Output variable is now 1.5

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

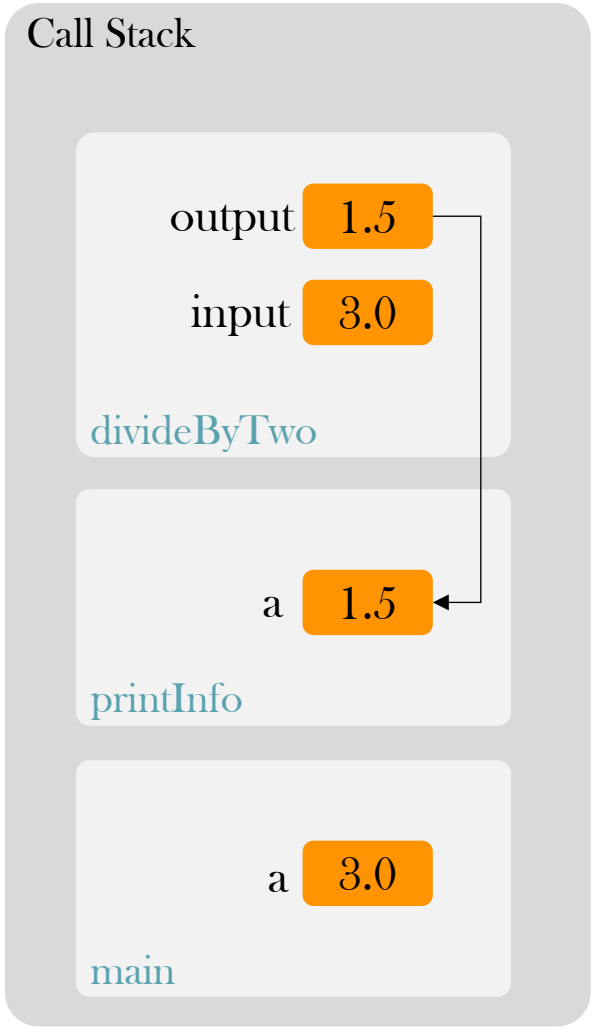


divideByTwo will return 1.5

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```



a in printInfo gets 1.5

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

divideByTwo method is finished and deleted from call stack

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

1.5 is printed

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

printInfo method is finished and deleted from call stack

Call Stack Example

- What is the output of the following program?

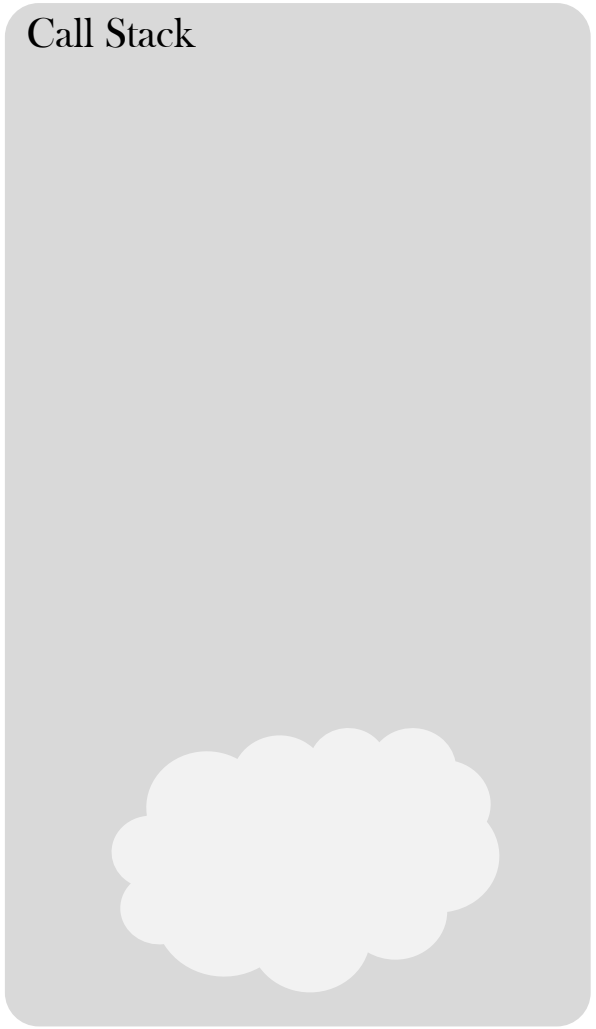
```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

3.0 is printed

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

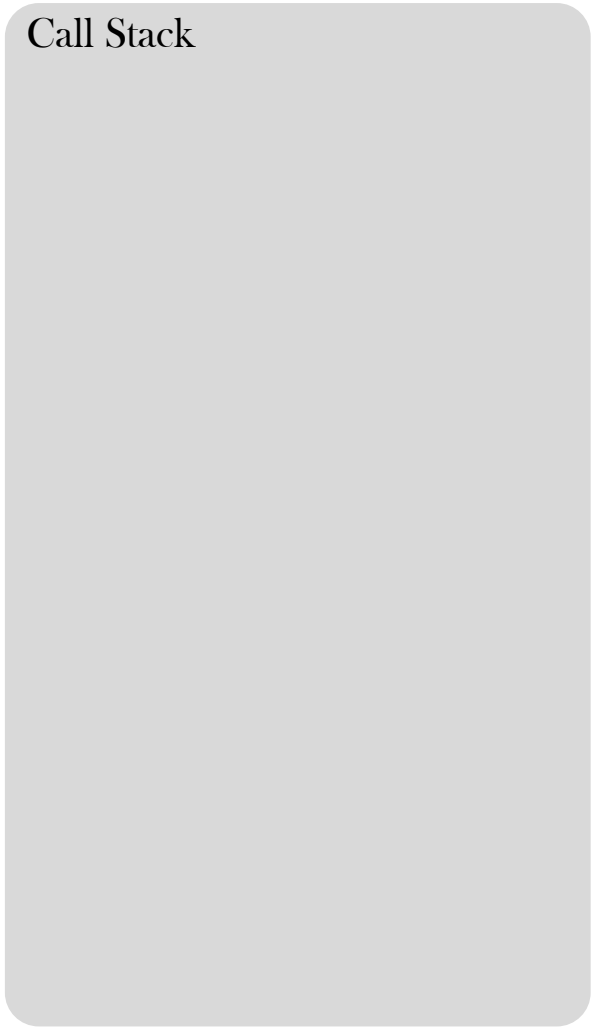


main method is finished and deleted from call stack

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```



Program is finished. Call stack is empty

Call Stack Example

- What is the output of the following program?

```
1  public class AppCallStack {
2
3   public static void main(String[] args) {
4    double a = 3.0;
5    printInfo(a);
6    System.out.println("a in main: " + a);
7   }
8
9   private static void printInfo(double a) {
10   a = divideByTwo(a);
11   System.out.println("a in printInfo: " + a);
12  }
13
14  private static double divideByTwo(double input) {
15   double output = input/2.0;
16   return output;
17  }
18 }
```

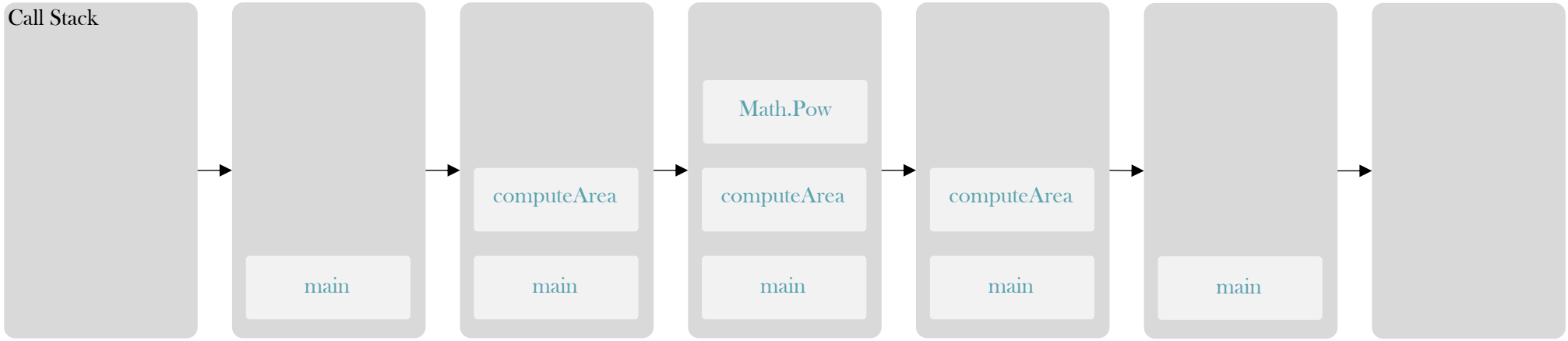
Answer

Program output

```
a in printInfo: 1.5
a in main: 3.0
```

Activation Records and Call Stack

- Each time a method is invoked, the system creates an activation record that stores parameters and variables for the method and places the activation record in an area of memory known as a call stack
  - When a method calls another method, a new activation record is created for the new method
  - When a method finishes its work and returns to its caller, its activation record is removed from the call stack
- Example: Assume `main` method calls `computeArea` and computeArea calls `Math.Pow`
  - Activation records in the call stack will be like this:



Passing arguments by value

- The arguments are passed by value to parameters when invoking a method
- Variables in the caller method is not affected by changes made to the parameters inside the method
- Pass by value is only applicable for primitive data types such as int, long, float, double, boolean

```
public class Increment {
  public static void main(String[] args) {
    int x = 1;
    System.out.println("Before the call, x is " + x);
    increment(x);
    System.out.println("after the call, x is " + x);
  }
  public static void increment(int n) {
    n++;
    System.out.println("n inside the method is " + n);
  }
}
```

Program output

```
Before the call, x is 1
n inside the method is 2
After the call, x is 1
```

Call Stack and Heap

- Call stacks of Java codes provided can be analyzed from: http://www.pythontutor.com/

- Press “Start Visualizing Code Now” link
- Choose Java 8 from the list
- Paste your code into the box
- Press “Visualize Execution”
- See the next slide for a sample screenshot

```
public class App {

  public static void main(String[] args) {
    int[] x = {4, 66, 98};
    System.out.println(Arrays.toString(x));
    change(x);
    System.out.println(Arrays.toString(x));
  }

  private static void change(int[] x) {
    x[0] = -1;
    x[1] = x[1] * 2;
    x[2] = x[0];
  }
}
```

Void Methods

Methods that do not return anything

Void Return Type

- Methods with void return type do not return a value

```
public class AppMethods {
 public static void main(String[] args) {
  double[] numbers = {2.0, 8.0, 4.0, 2.0};
  printArray(numbers); // call the method to print numbers
 }
 /**
  * Prints the input array, line by line (This is a Javadoc style comment)
  * @param numbers Input array
  */
 private static void printArray(double[] numbers) {
  for (double e: numbers)
   System.out.println(e);
  // note that there is no return statement in this method
 }
}
```

Void Return Type

- In some cases, you can write `return;` statement to stop the execution of a void method

```
private static void printGrade(double grade) {

 if ((grade < 0) || (grade > 100)) {
  System.out.println("Invalid grade");
  return;
 }

 if (grade >= 50) {
  System.out.println("Pass");
 }
 else if (grade < 50) {
  System.out.println("Fail");
 }
}
```

Array Parameters in Methods

Methods with Array Parameters

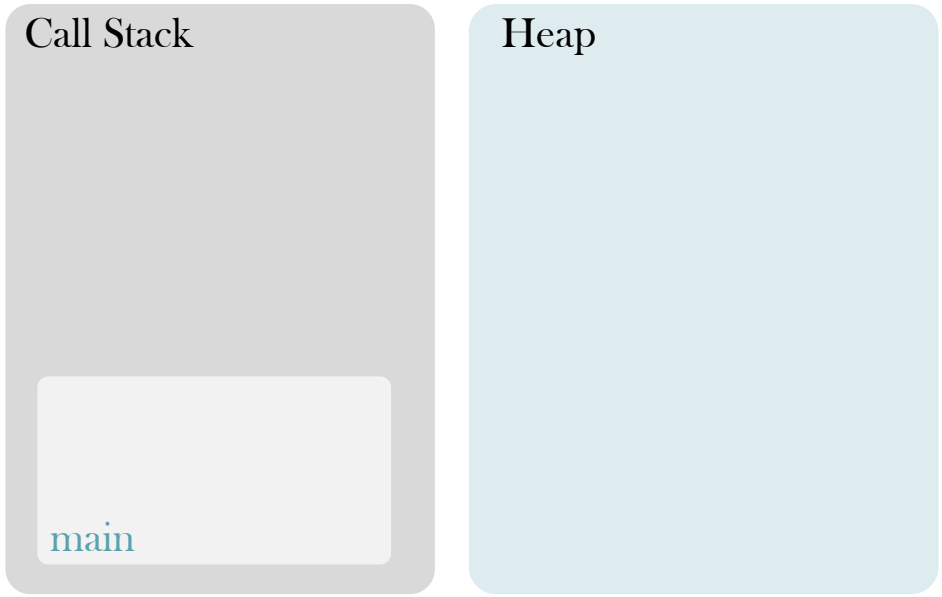
- Methods can accept arrays as inputs
- Example: printArray method prints array elements line by line

```
public class AppPrintArray {

 public static void main(String[] args) {
  String[] names = {"A","B","C","D"};
  printArray(names);
 }
 private static void printArray(String[] inputArray) {
  for (String e : inputArray)
   System.out.println(e);
 }
}
```

Methods with Array Parameters

- Let's see call stack and heap for the program given below

```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```

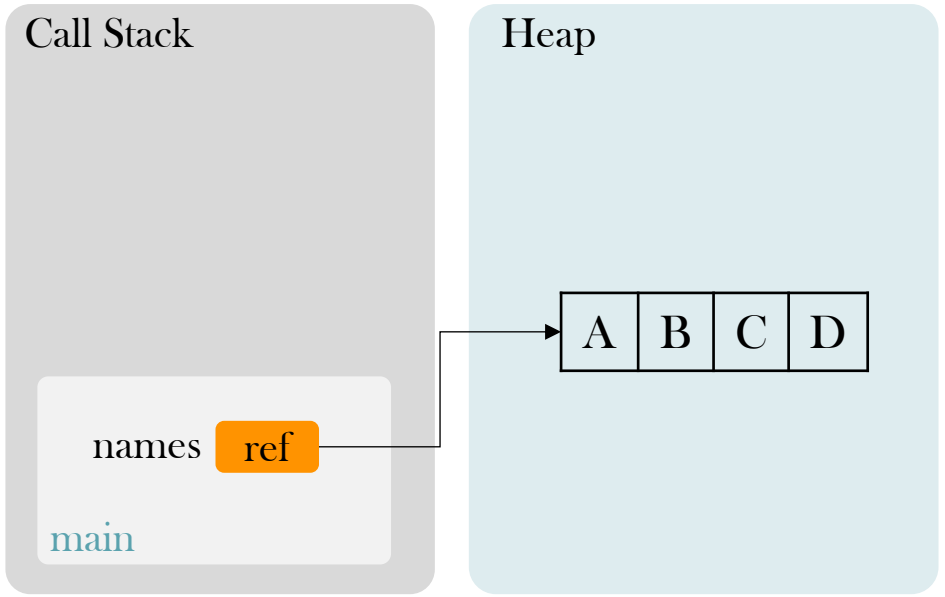


main method starts

Methods with Array Parameters

- Let's see call stack and heap for the program given below

```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```

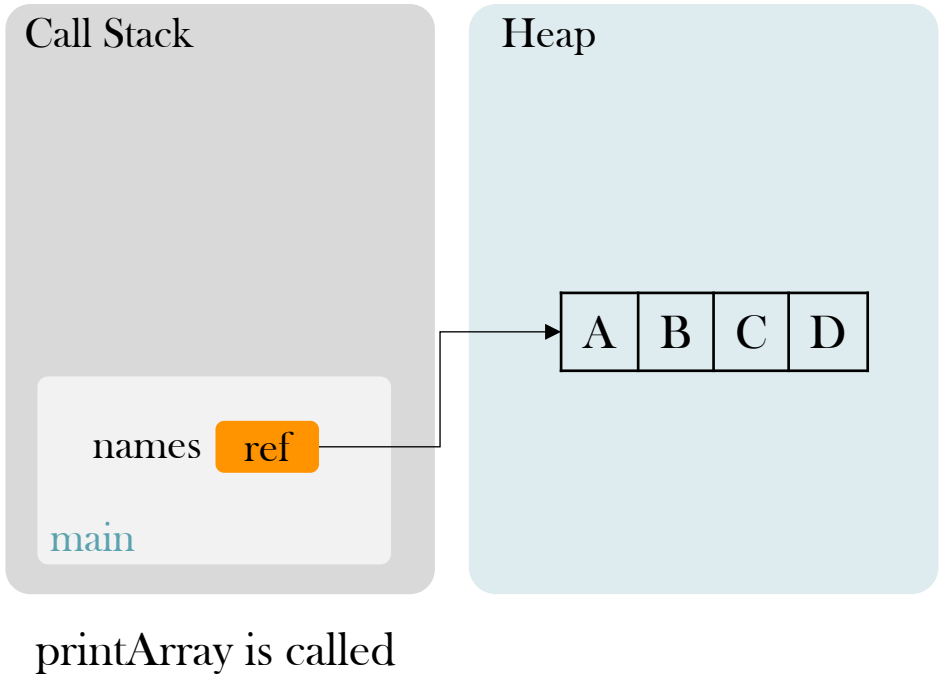


`names` is an array reference variable and it is located in the call stack.
However actual array values are created in a special memory region, called heap

Methods with Array Parameters

- Let's see call stack and heap for the program given below

```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```

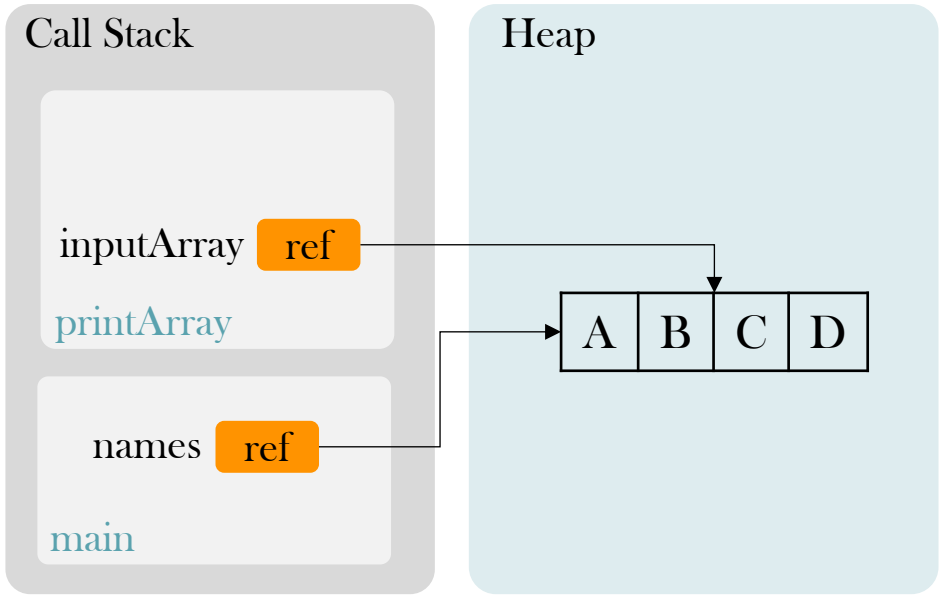


printArray is called

Methods with Array Parameters

- Let's see call stack and heap for the program given below

```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```

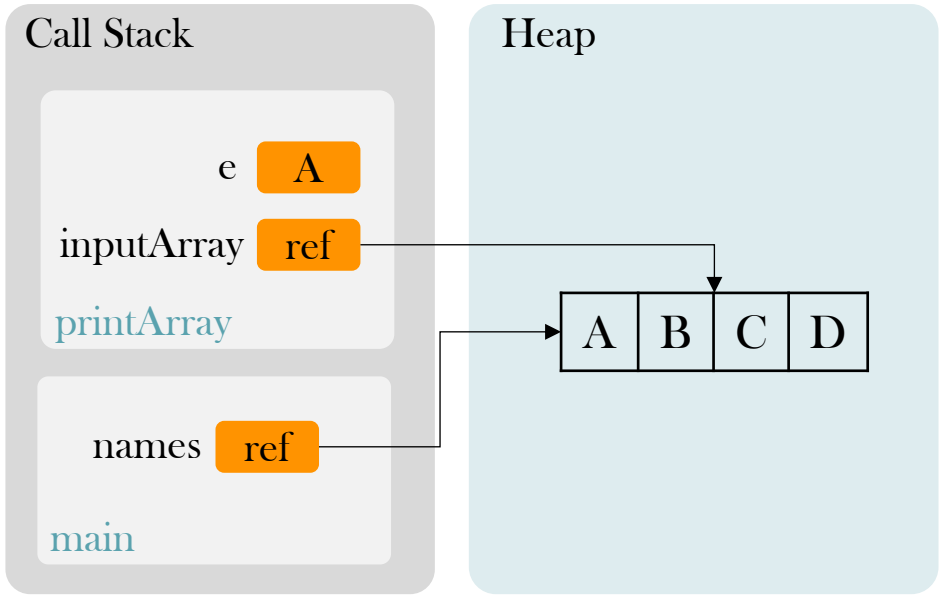


Parameter `inputArray` gets the value of the input array argument, which is ref. Therefore, `inputArray` and `names` arrays point to the same array in the heap

Methods with Array Parameters

- Let's see call stack and heap for the program given below

```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```

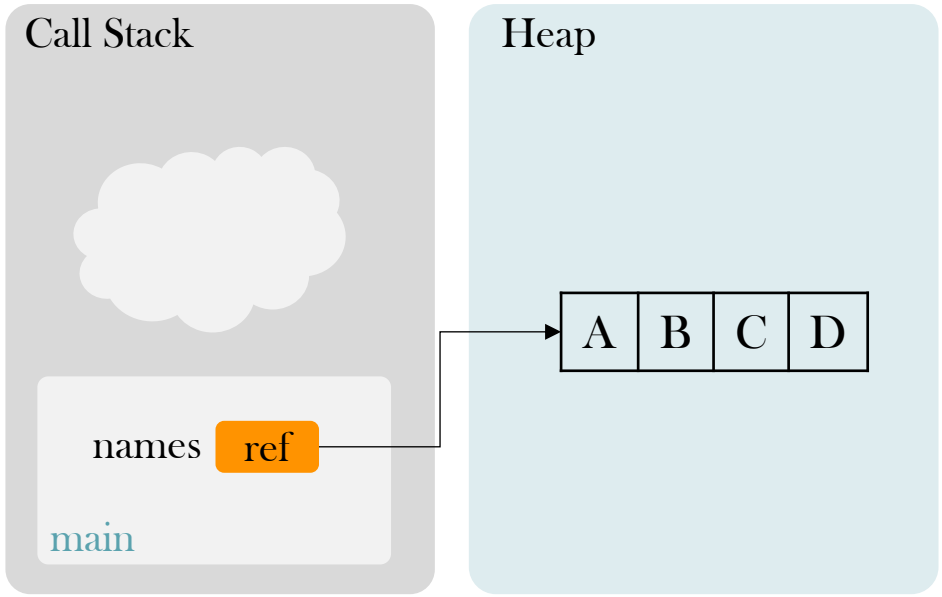


In the foreach loop, variable e is created. It takes the value of "A" for the first iteration

Methods with Array Parameters

- Let's see call stack and heap for the program given below

```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```

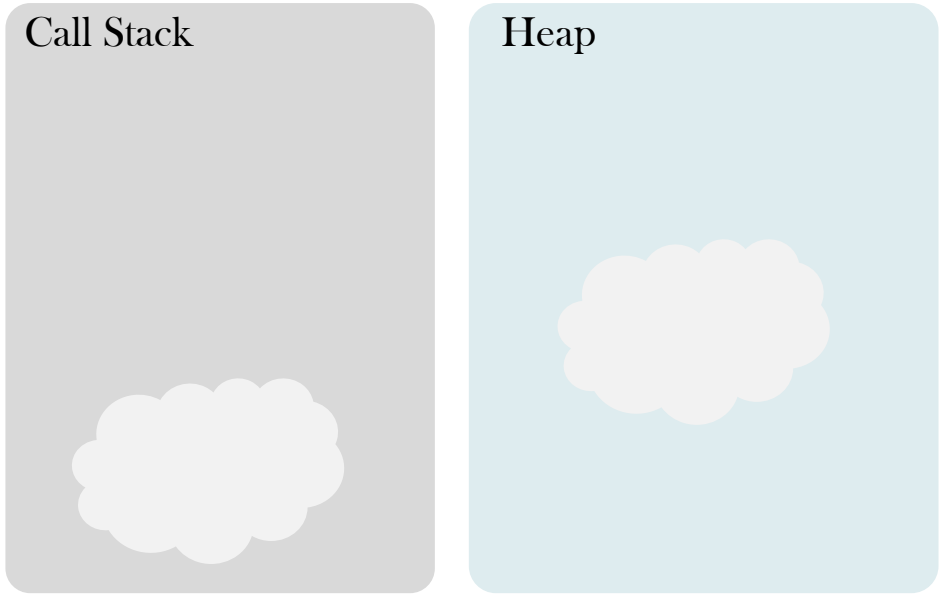


When printArray method call is done, its activation record is deleted from the call stack

Methods with Array Parameters

- Let's see call stack and heap for the program given below

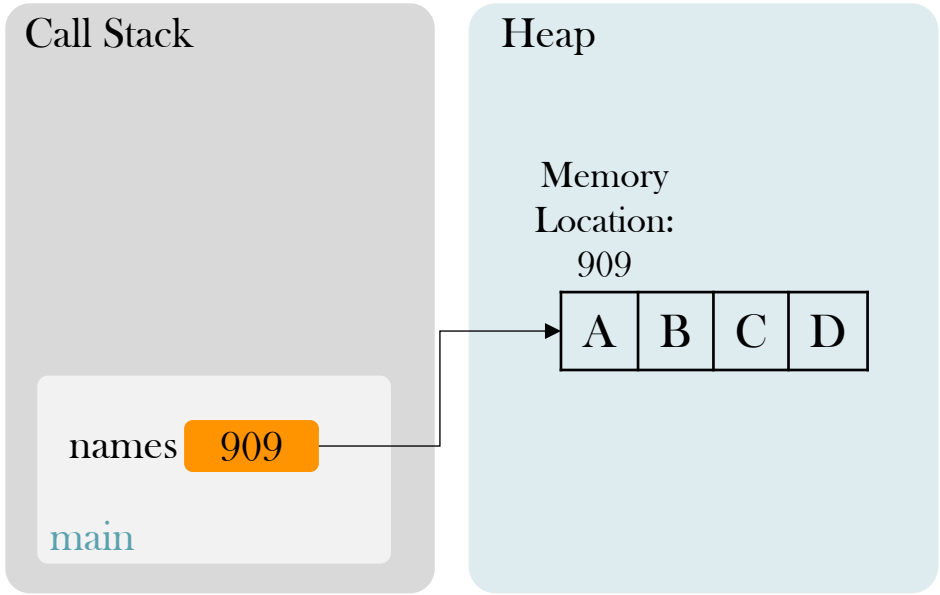
```
public class AppPrintArray {
  public static void main(String[] args) {
    String[] names = {"A","B","C","D"};
    printArray(names);
  }
  private static void printArray(String[] inputArray) {
    for (String e : inputArray)
      System.out.println(e);
  }
}
```



When main is finished, its activation record is deleted from the call stack, together with the names array.

Array is a Reference Variable

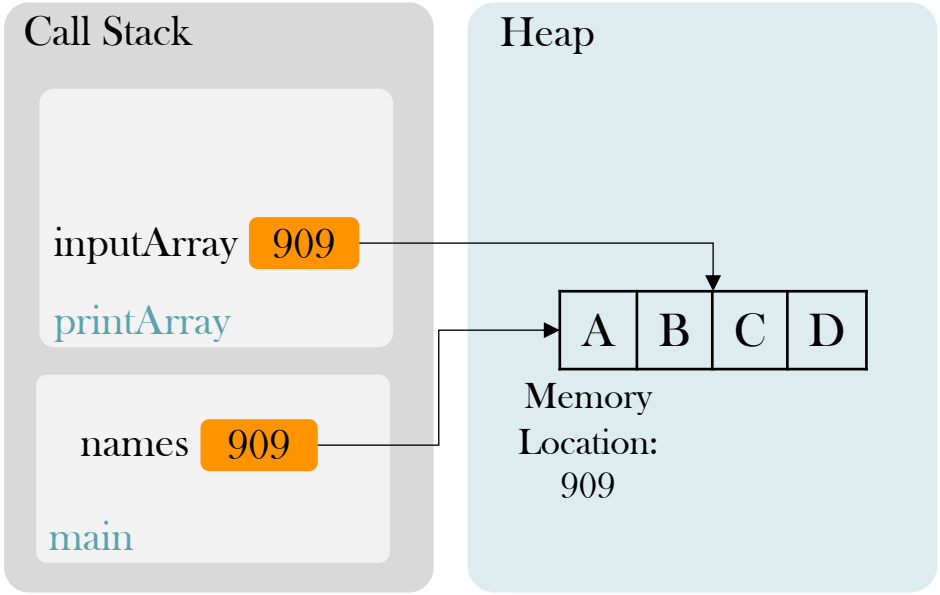
- Array variables are reference variables: their type is reference type
  - Remember the primitive types: int, double, float, long, boolean etc.
- If a variable is a reference type, its contents are stored in heap, not in call stack
  - In call stack, only the refence is stored. Reference is the memory location of the actual content in the heap



Names is an array reference variable.
It stores the memory location of the array in heap

Array is a Reference Variable

- When a method is called with an array argument, array parameter gets the reference value, i.e., the memory address
- This is called pass by reference
- In the example below, both names and inputArray variables have the same memory address, i.e., they point to the same array in the heap



names is the array argument
inputArray is the array parameter in printArray method
When main calls printArray, memory location of names is passed to the inputArray variable

Pass by Value vs Pass by Reference

- Java uses pass by value to pass arguments to a method
  - There are important differences between passing a value of variables of primitive data types and passing arrays
- For a parameter of a primitive type value, the actual value is passed
  - Changing the value of the local parameter inside the method does not affect the value of the variable outside the method
- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method
  - Any changes to the array that occur inside the method body will affect the original array that was passed as the argument

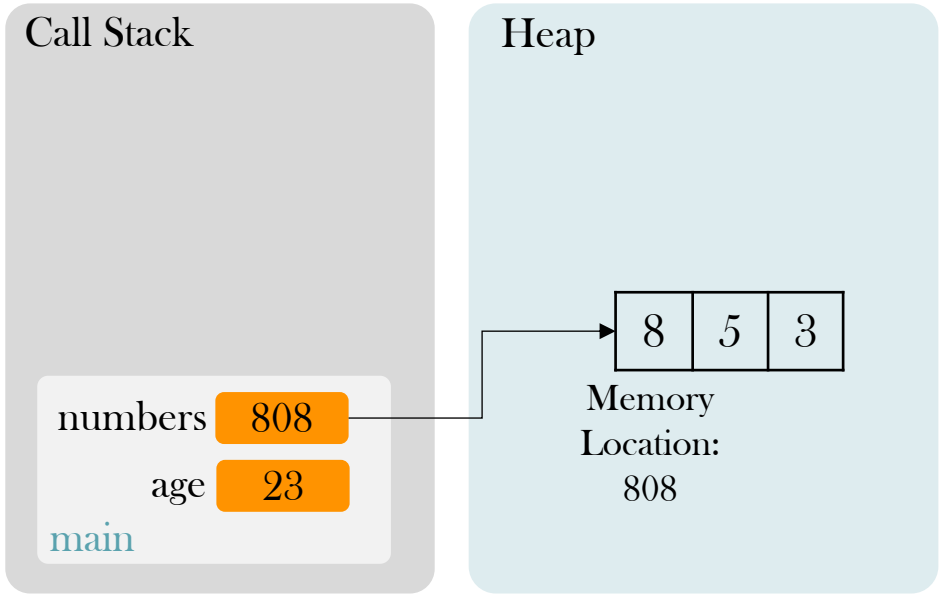
Pass by Value vs Pass by Reference

- What is the output of the program?

```
public class AppPassValueReference {
 public static void main(String[] args) {

  int age = 23; // primitive type
  int[] numbers = {8,5,3}; // array is a reference type
  System.out.println("Before: Age=" + age + ", Numbers=" + Arrays.toString(numbers));
  modifyValues(age, numbers);
  System.out.println("After : Age=" + age + ", Numbers=" + Arrays.toString(numbers));

 }

 private static void modifyValues(int age, int[] b) {
  age = 2;
  b[0] = 2;
 }
}
```

Step by Step Execution

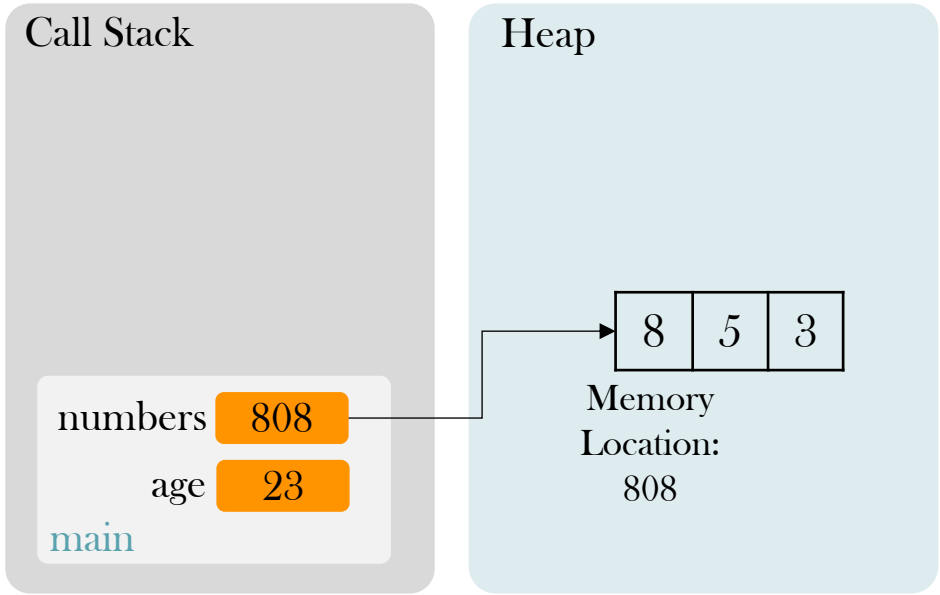
```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



`age` and `numbers` variables are created

Step by Step Execution

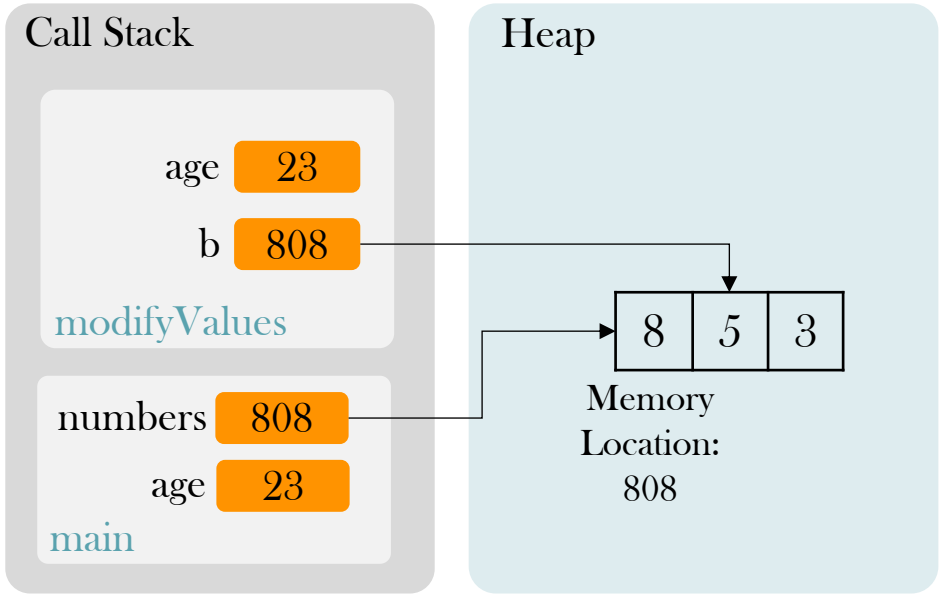
```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



`modifyValues` is called

Step by Step Execution

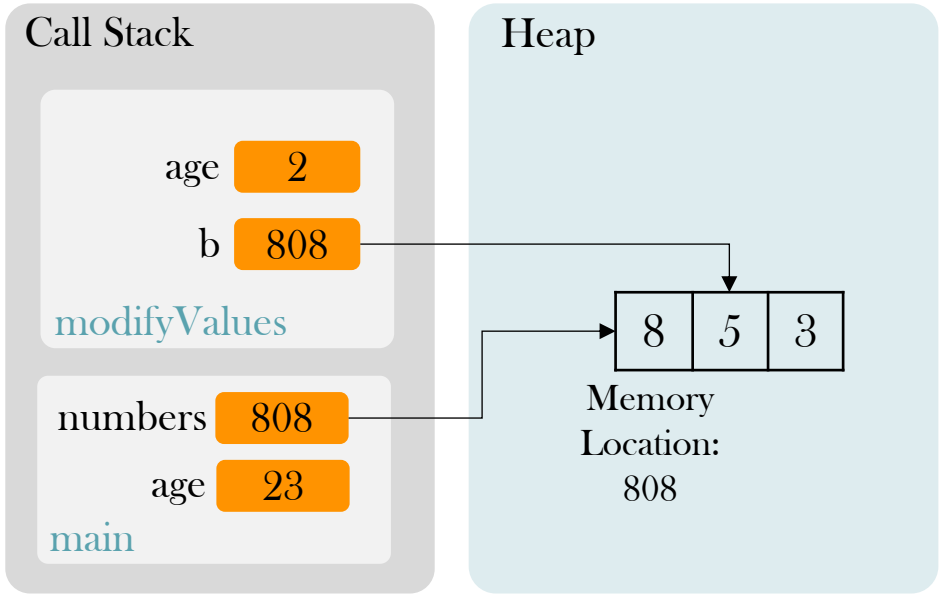
```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



`age` gets 23. `b` get the memory address 808 of the `numbers` variable

Step by Step Execution

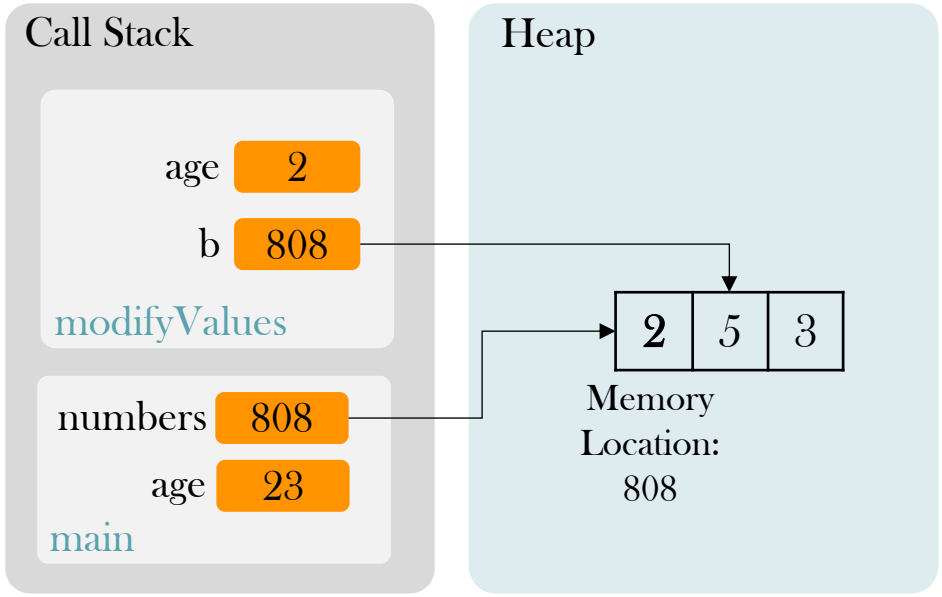
```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



age gets 2

Step by Step Execution

```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



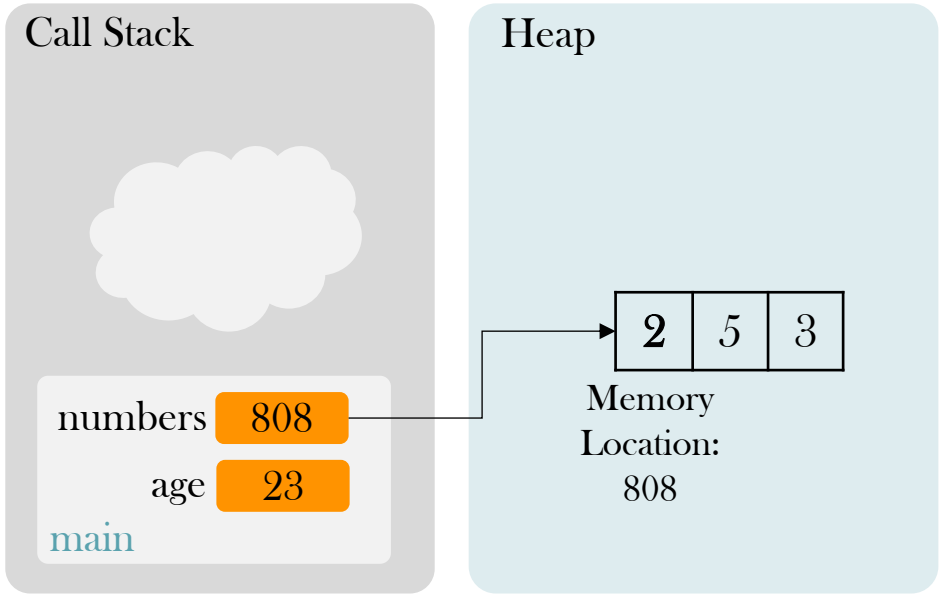
`b[0]` gets 2.

Note that `numbers` array contents are also changed.

`age` in main is not affected.

Step by Step Execution

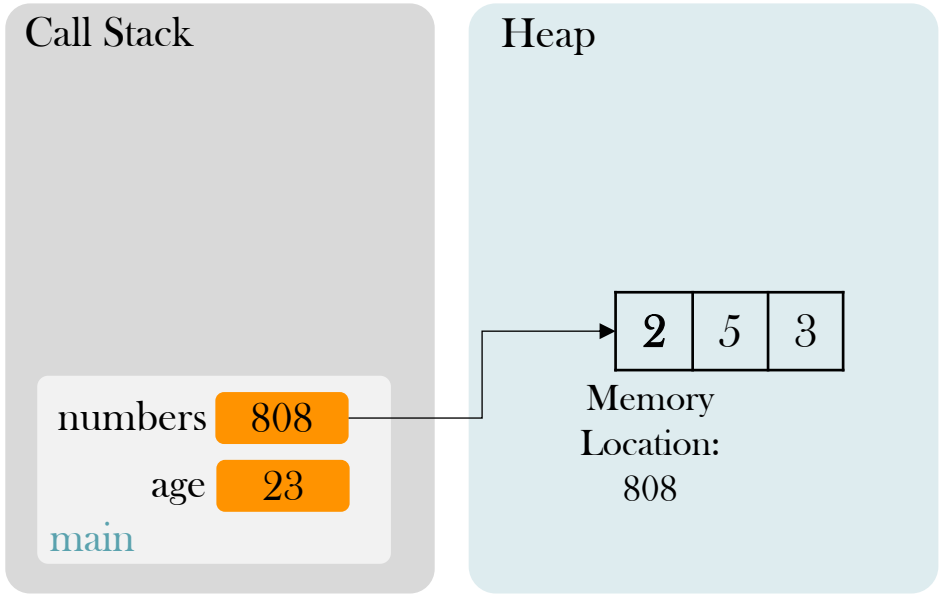
```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



`modifyValues` method exits. Its activation record in the call stack is deleted.

Step by Step Execution

```
public class AppPassValueReference {
  public static void main(String[] args) {
    int age = 23;
    int[] numbers = {8,5,3};
    modifyValues(age, numbers);
  }
  private static void modifyValues(int age, int[] b) {
    age = 2;
    b[0] = 2;
  }
}
```



In main, `numbers` array is changed but `age` is still 23.

Pass by Value vs Pass by Reference

- Output of the program is given below

```
public class AppPassValueReference {
 public static void main(String[] args) {
  int age = 23; // primitive type
  int[] numbers = {8,5,3}; // array is a reference type
  System.out.println("Before: Age=" + age + ", Numbers=" + Arrays.toString(numbers));
  modifyValues(age, numbers);
  System.out.println("After : Age=" + age + ", Numbers=" + Arrays.toString(numbers));
 }
 private static void modifyValues(int age, int[] b) {
  age = 2;
  b[0] = 2;
 }
}
```

Program output

```
Before: Age=23, Numbers=[8, 5, 3]
After : Age=23, Numbers=[2, 5, 3]
```

Pass by Value vs Pass by Reference

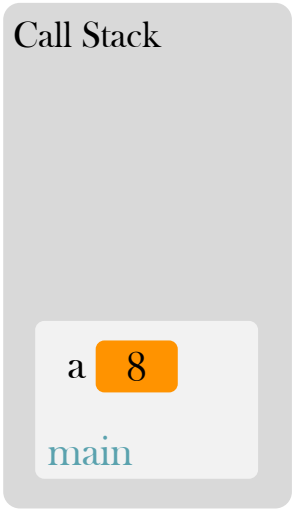
- Methods can modify array inputs since arrays are reference type
- Methods can not modify primitive type inputs such as int, float, double, long, boolean etc.
  - If you want to modify a primitive type variable, method should return a value and that variable should be assigned to the method output value

```
public class App {

 public static void main(String[] args) {
  int a = 8;
  a = modify(a);
 }

 private static int modify(int value) {
  return value+1;
 }
}
```

Pass by Value vs Pass by Reference

- Methods can modify array inputs since arrays are reference type
- Methods can not modify primitive type inputs such as int, float, double, long, boolean etc.
  - If you want to modify a primitive type variable, method should return a value and that variable should be assigned to the method output value

```
public class App {
  public static void main(String[] args) {
    int a = 8;
    a = modify(a);
  }
  private static int modify(int value) {
    return value+1;
  }
}
```



a is 8

Pass by Value vs Pass by Reference

- Methods can modify array inputs since arrays are reference type
- Methods can not modify primitive type inputs such as int, float, double, long, boolean etc.
  - If you want to modify a primitive type variable, method should return a value and that variable should be assigned to the method output value

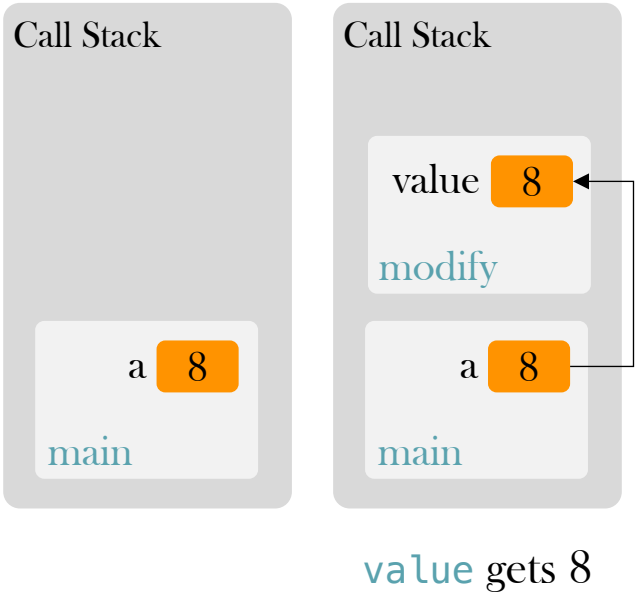
```
public class App {
  public static void main(String[] args) {
    int a = 8;
    a = modify(a);
  }
  private static int modify(int value) {
    return value+1;
  }
}
```

`modify` method will be called

Pass by Value vs Pass by Reference

- Methods can modify array inputs since arrays are reference type
- Methods can not modify primitive type inputs such as int, float, double, long, boolean etc.
  - If you want to modify a primitive type variable, method should return a value and that variable should be assigned to the method output value

```
public class App {
  public static void main(String[] args) {
    int a = 8;
    a = modify(a);
  }
  private static int modify(int value) {
    return value+1;
  }
}
```

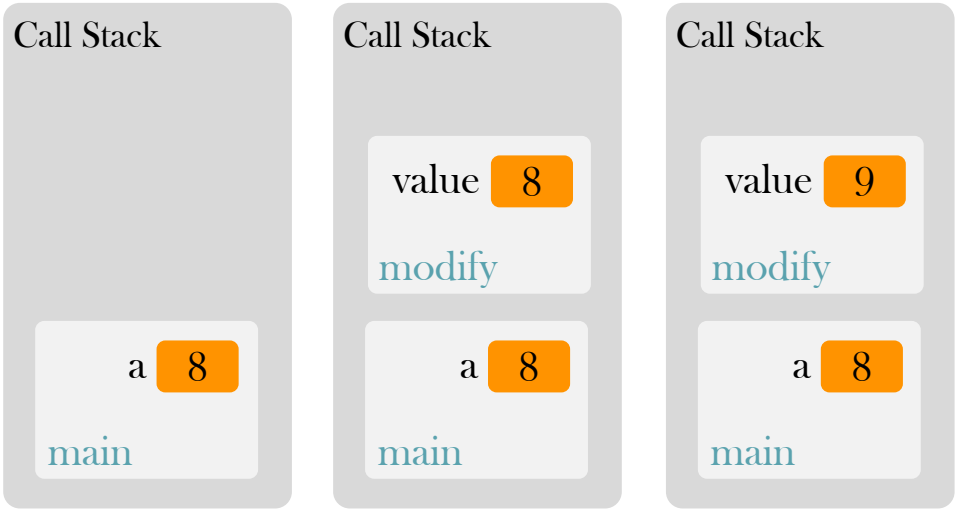


`value` gets 8

Pass by Value vs Pass by Reference

- Methods can modify array inputs since arrays are reference type
- Methods can not modify primitive type inputs such as int, float, double, long, boolean etc.
  - If you want to modify a primitive type variable, method should return a value and that variable should be assigned to the method output value

```
public class App {
  public static void main(String[] args) {
    int a = 8;
    a = modify(a);
  }
  private static int modify(int value) {
    return value+1;
  }
}
```

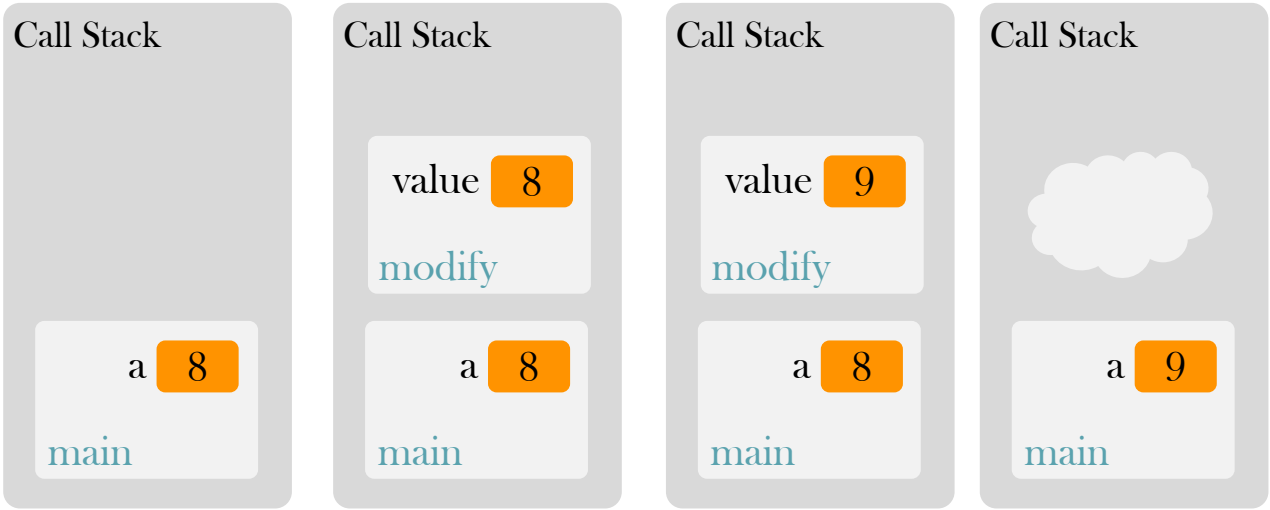


`value` gets 9

Pass by Value vs Pass by Reference

- Methods can modify array inputs since arrays are reference type
- Methods can not modify primitive type inputs such as int, float, double, long, boolean etc.
  - If you want to modify a primitive type variable, method should return a value and that variable should be assigned to the method output value

```
public class App {
  public static void main(String[] args) {
    int a = 8;
    a = modify(a);
  }
  private static int modify(int value) {
    return value+1;
  }
}
```



Method return value 9 is assigned to a

Returning Arrays from Methods

Methods Returning Array

- Write a method which generates an array of length 10, filled with value given as argument

```
public static void main(String[] args) {
 int[] output = generateArray(5);
 System.out.println("Output array is " + Arrays.toString(output));
}
```

Methods Returning Array

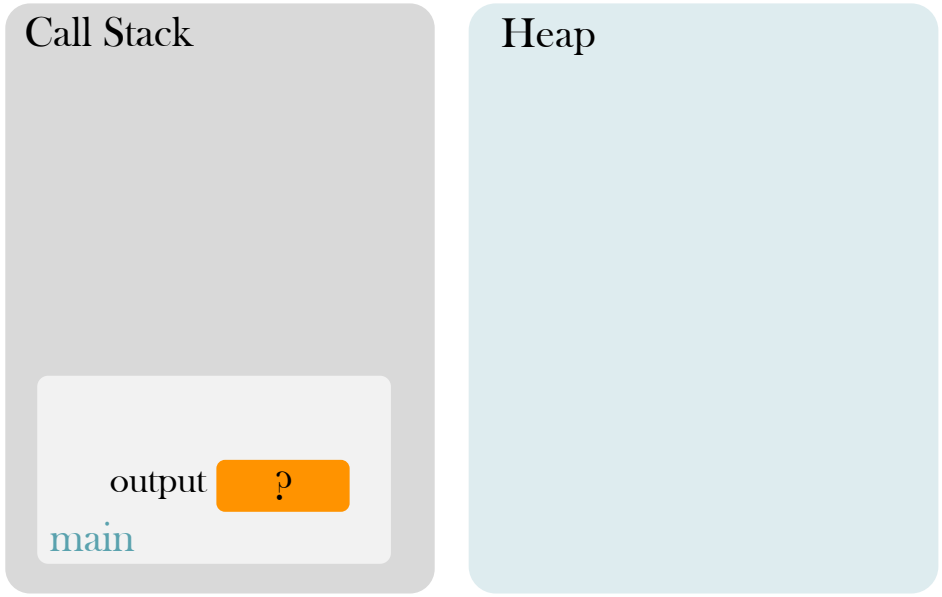
- Write a method which generates an array of length 4, filled with value given as argument

```
public static void main(String[] args) {
 int[] output = generateArray(5);
 System.out.println("Output array is " + Arrays.toString(output));
}
```

```
private static int[] generateArray(int inputValue) {

 int[] result = new int[4];              // create an empty array of size 10
 for (int i = 0; i < result.length; i++)
  result[i] = inputValue;                // fill the array
 return result;                          // return the result array

}
```

Step by Step Execution

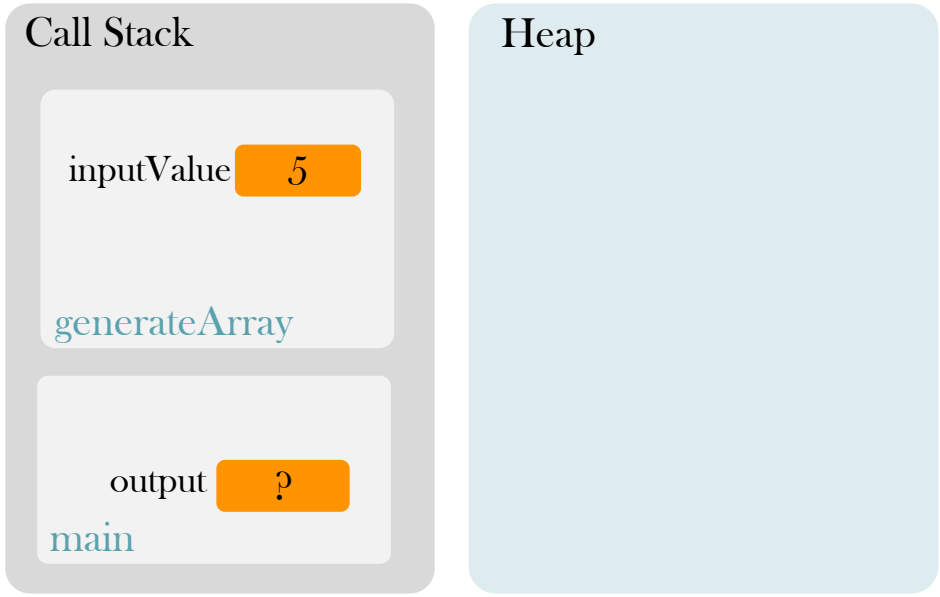
```
public static void main(String[] args) {
  int[] output = generateArray(5);
}
private static int[] generateArray(int inputValue) {
  int[] result = new int[4];
  for (int i = 0; i < result.length; i++)
    result[i] = inputValue;
  return result;
}
```



Before the method is called, `output` array is created.
It does not contain any value yet.

Step by Step Execution

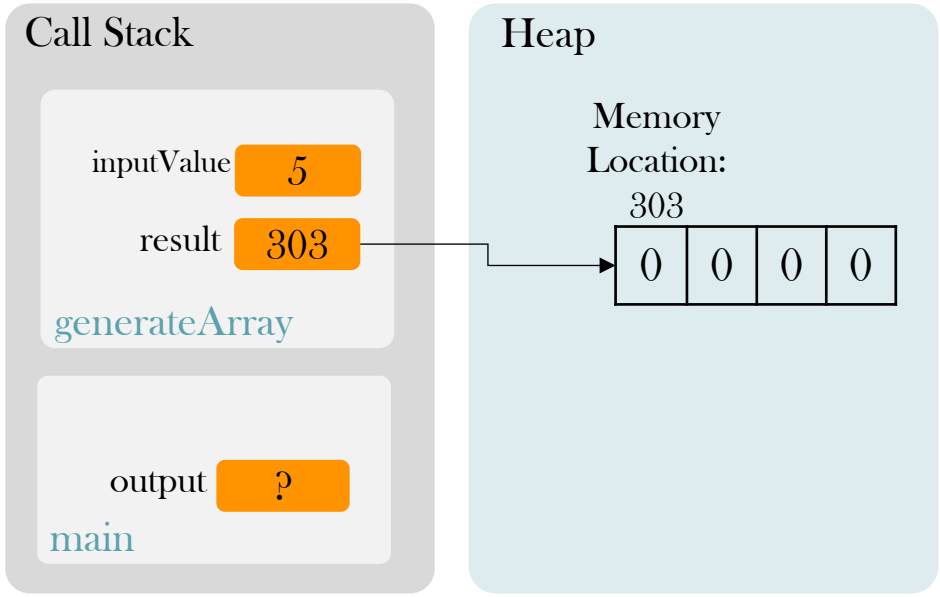
```
public static void main(String[] args) {
  int[] output = generateArray(5);
}
private static int[] generateArray(int inputValue) {
  int[] result = new int[4];
  for (int i = 0; i < result.length; i++)
    result[i] = inputValue;
  return result;
}
```



`inputValue` parameter gets 5

Step by Step Execution

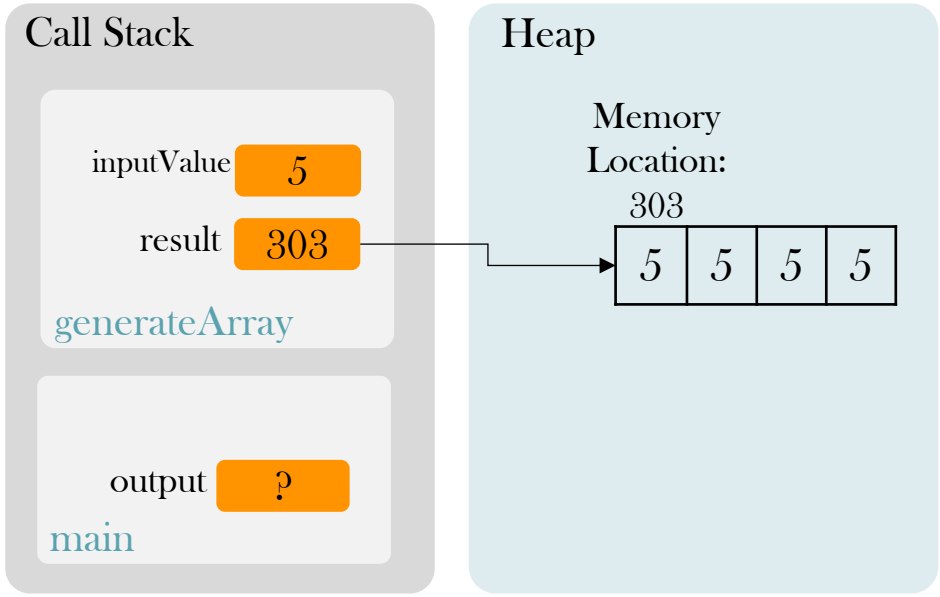
```
public static void main(String[] args) {
  int[] output = generateArray(5);
}
private static int[] generateArray(int inputValue) {
  int[] result = new int[4];
  for (int i = 0; i < result.length; i++)
    result[i] = inputValue;
  return result;
}
```



`result` array is created in heap, with zero values

Step by Step Execution

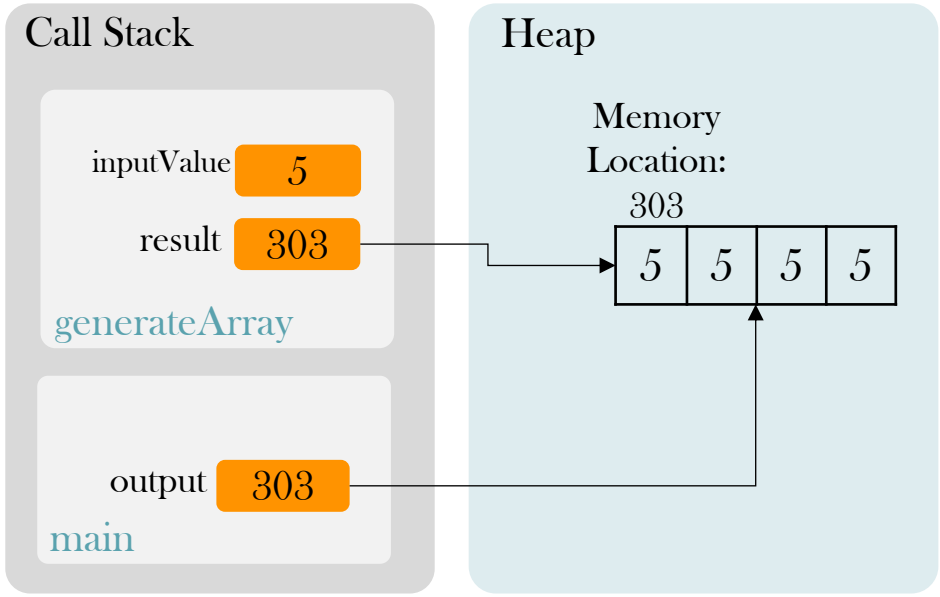
```
public static void main(String[] args) {
  int[] output = generateArray(5);
}
private static int[] generateArray(int inputValue) {
  int[] result = new int[4];
  for (int i = 0; i < result.length; i++)
    result[i] = inputValue;
  return result;
}
```



Array is filled with 5

Step by Step Execution

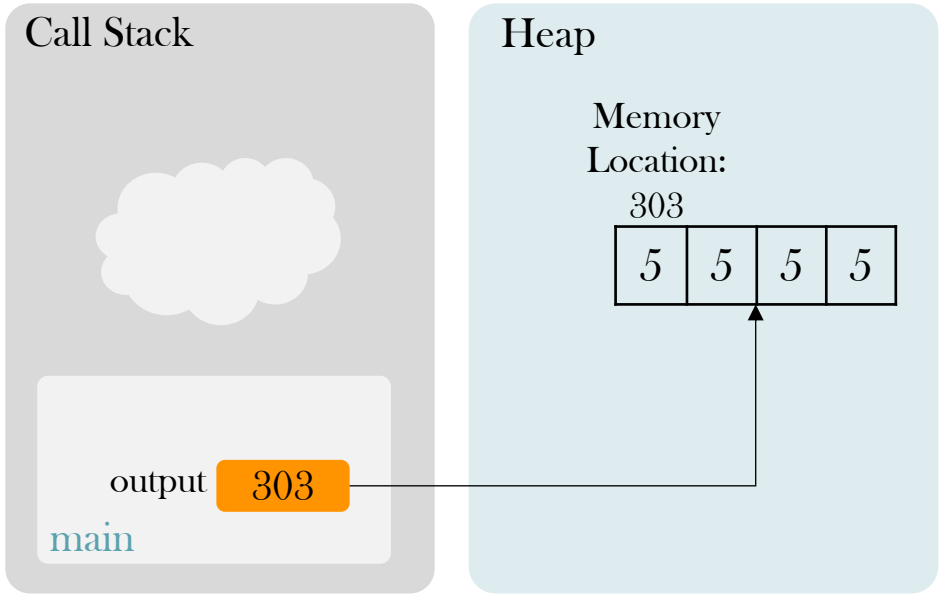
```
public static void main(String[] args) {
  int[] output = generateArray(5);
}
private static int[] generateArray(int inputValue) {
  int[] result = new int[4];
  for (int i = 0; i < result.length; i++)
    result[i] = inputValue;
  return result;
}
```



Return statement assigns `output` array to the `result` array

Step by Step Execution

```
public static void main(String[] args) {
  int[] output = generateArray(5);
}
private static int[] generateArray(int inputValue) {
  int[] result = new int[4];
  for (int i = 0; i < result.length; i++)
    result[i] = inputValue;
  return result;
}
```



Method exits. Its activation record is removed from the call stack.

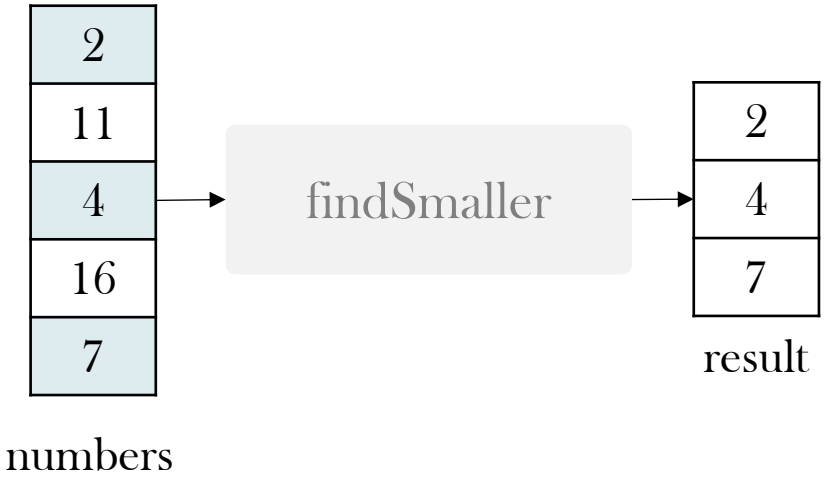
Methods Returning Array

- If a method returns an array, array should be created inside the method, and should be returned

```
private static int[] generateArray(int inputValue) {

 int[] result = new int[4];            // create an empty array of size 10
 for (int i = 0; i < result.length; i++)
  result[i] = inputValue;
 return result;                        // return the result array

}
```

Example: Method Returning Array

- Write a method which accepts an integet input array as a parameter and returns the elements which are less than 10 as an output array




```
int[] numbers = {2,11,4,16,7};
int[] result = findSmaller(numbers);
```

Solution Algorithm

- First, count the number of elements that are less than 10, say `counter`
- Create an empty array of size `counter`
- Traverse the input array from start to finish, if the current element is less than 10, put the value to output array

Source Code – Part 1

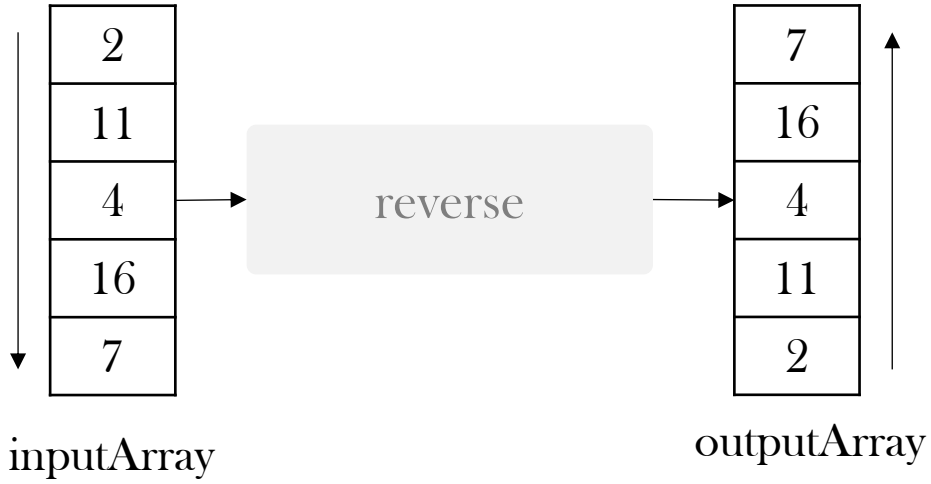
```
import java.util.Arrays;

public class AppFindSmaller {

  public static void main(String[] args) {
    int[] numbers = {2,11,4,16,7};
    int[] result = findSmaller(numbers);
    System.out.println(Arrays.toString(result));
  }

   // code continues

}
```

Source Code – Part 2

```
public class AppFindSmaller {
  // code continues from here
  /**
   * Finds the values that are less than 10 in the input array
   * and returns smaller values in an array (This is a Javadoc style comment)
   *
   * @param numbers Input array
   * @return Array containing smaller values
   */
  private static int[] findSmaller(int[] numbers) {
    int counter = 0; // counts the number of smaller elements
    for (int e : numbers) // foreach loop
      if (e < 10)
        counter++;

    // create the output array
    int[] output = new int[counter];

    // place smaller numbers into the output array
    int index = 0;
    for (int e : numbers)
      if (e < 10)
        output[index++] = e;

    return output;
  }
}
```

Example: Method Returning Array

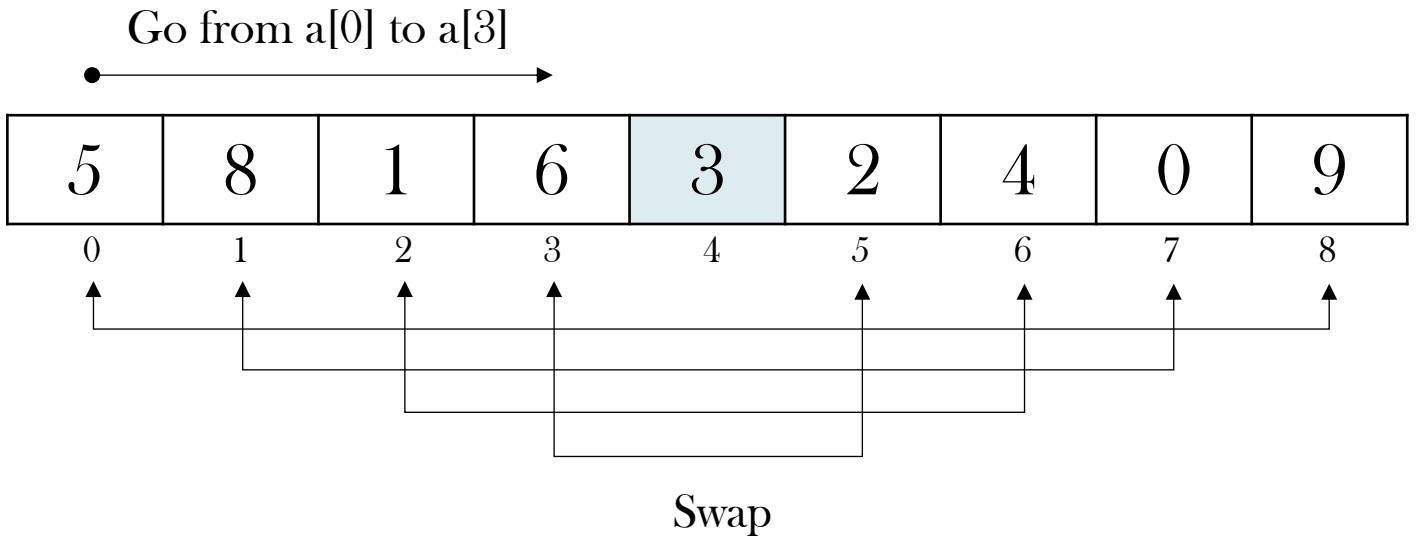
- Write a method which outputs reversed version of an input array




```
int[] inputArray = {2,11,4,16,7};
int[] outputArray = reverse(inputArray);
```

Solution Algorithm

- Create a new empty array of the same size
- Copy all element to the new array.
  - We do not want to modify the original input array
- Traverse the input array from start until the middle of the array
  - Swap a[i] with a[length-i]



Source Code

```
public class AppReverse {
 public static void main(String[] args) {
  int[] a = {5,8,1,6,3,2,4,0};
  int[] output = reverse(a);
 }
 /**
  * Returns the reversed version of an array
  * @param inputArray Input array
  * @return Reversed array
  */
 private static int[] reverse(int[] inputArray) {

  int[] result = new int[inputArray.length];   // create the new array
  int middle = inputArray.length/2;            // compute middle index
  for (int i = 0; i < middle; i++) {
   result[i] = inputArray[inputArray.length-i-1]; // swap elements
   result[inputArray.length-i-1] = inputArray[i];
  }
  return result; // return the result array
 }
}
```

Reverse Array Version 2

- Previous program creates a reversed version of an input array and does not change the original array
- This time we want a method which modifies the original input array and reverses it
- New `reverseNew` method does not need to return anything. Its return type is void
- Sample method is shown below

```
int[] inputArray = {2,11,4,16,7};
reverseNew(inputArray);
```

Source Code

```
public class AppReverse2 {
 public static void main(String[] args) {
  int[] a = {5,8,1,6,3,2,4,0,9};
  reverseNew(a); // new reverse method. Modifies the input array
 }

 /**
  * Reverses an array
  * @param b Input array
  */
 private static void reverseNew(int[] b) {
  int temp;
  for (int i = 0; i < (b.length/2); i++) {
    // swap elements
   temp = b[i];
   b[i] = b[b.length-i-1];
   b[b.length-i-1] = temp;
  }
 }
}
```

Comparison of Two Solutions

- We have written two versions of the reverse method
- First solution returns a new array and does not modifies the original array
  - Its return type is an array
- Second solution modifies the original array and its return type is void
  - It does not need to return an array

First solution

```
int[] inputArray = {2,11,4,16,7};
int[] outputArray = reverse(inputArray);
```

Second solution

```
int[] inputArray = {2,11,4,16,7};
reverseNew(inputArray);
```

Method Overloading

Method Overloading

- Two or more methods can have the same name but different parameter lists
- This is referred to as method overloading
- The Java compiler determines which method to use based on the method signature

```
public static int max(int num1, int num2) { /** Return the max of two int values */
  if (num1 > num2)
    return num1;
  else
    return num2;
}
public static double max(double num1, double num2) { /** Find the max of two double values */
  if (num1 > num2)
    return num1;
  else
    return num2;
}
public static double max(double num1, double num2, double num3) { /** Return the max of three double values */
  return max(max(num1, num2), num3);
}
```

Variable Length Arguments

Variable Length Argument Lists

- You can pass a variable number of arguments of the same type to a method

```
public class VarArgsDemo {
  public static void main(String[] args) {
    printMax(34, 3, 3, 2, 56.5);      // printMax method can process variable number of arguments
    printMax(4,6);                    // printMax method can process variable number of arguments
    printMax(new double[]{1, 2, 3});  // You can give an array as an argument
  }
  public static void printMax(double... numbers) {
    if (numbers.length == 0) {
      System.out.println("No argument passed");
      return;
    }
    double result = numbers[0]; // method treats the numbers parameter as an array
    for (int i = 1; i < numbers.length; i++)
      if (numbers[i] > result)
        result = numbers[i];
    System.out.println("The max value is " + result);
  }
}
```

Variable Length Argument Lists

- You can pass a variable number of arguments of the same type to a method
- Only one variable-length parameter may be specified in a method, and this parameter must be the last parameter
- Java treats a variable-length parameter as an array
  - You can pass an array or a variable number of arguments to a variable-length parameter
  - When invoking a method with a variable number of arguments, Java creates an array and passes the arguments to it

Scope of Variables

Scope of Variables

- The scope of a variable is the part of the program where the variable can be referenced
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable

- What is the output of the following code?

```
public static void main(String[] args) {
  int a = 2;
  if (a < 10)
  {
    int b = 4;
    System.out.println("a is less than 10");
  }
  System.out.println(a);
  System.out.println(b);
}
```

Scope of Variables

Scope of a

```
public static void main(String[] args) {
  int a = 2;
  if (a < 10)
  {
    int b = 4;
    System.out.println("a is less than 10");
  }
  System.out.println(a);
  System.out.println(b);
}
```

Scope of b

```
public static void main(String[] args) {
  int a = 2;
  if (a < 10)
  {
    int b = 4;
    System.out.println("a is less than 10");
  }
  System.out.println(a);
  System.out.println(b); // b is not defined here
}
```

The scope of local variable b is from the line it is defined until the end of the if block.

Try to access b in println statement gives compile error

Scope of Variables

- What is the output of the following code?

```
int sum = 0;
for (int i = 1; i <= 4; i++)
{
  sum = sum + i;
}
System.out.println("Sum of " + i + " numbers is: " + sum);
```

Scope of Variables

```
int sum = 0;
for (int i = 1; i <= 4; i++)
{
  sum = sum + i;
}
System.out.println("Sum of " + i + " numbers is: " + sum);
```

Scope of i is the for loop. It is not defined outside of the for loop.
Try to access it in the println statement gives compile error.

A variable declared in the initial-action part of a for-loop header has its scope in the entire loop.

Scope of Variables

- A variable declared in the initial-action part of a for-loop header has its scope in the entire loop
- However, a variable declared inside a for-loop body has its scope limited in the loop body from its declaration to the end of the block that contains the variable

```
for (int i = 0; i < 10; i++)
{
  System.out.println(i);

  double b = 5;
  System.out.println(b);
  System.out.println(b * 2);
  System.out.println(b * 3);
}
```

Scope of i

```
for (int i = 0; i < 10; i++)
{
  System.out.println(i);

  double b = 5;
  System.out.println(b);
  System.out.println(b * 2);
  System.out.println(b * 3);
}
```

Scope of b

Scope of Variables

- A variable defined inside a method is referred to as a local variable
- A local variable defined in a method can not be accessed from another method

- What is the output of the following code?

```
public class AppScope1 {
  public static void main(String[] args) {
    int a = 4;
    int b = 7;
    printValues();
  }

  private static void printValues() {
    System.out.println("a is " + a + ", b is : " + b);
  }
}
```

Scope of Variables

```
public class AppScope1 {
  public static void main(String[] args)
  {
    int a = 4;
    int b = 7;
    printValues();
  }

  private static void printValues() {
    System.out.println("a is " + a + ", b is : " + b);
  }
}
```

Scope of a and b variables are in the main method. They can not be accessed from another method.

Println statement in `printValues` method gives compile error

Scope of Variables

- A local variable must be declared and assigned a value before it can be used

- What is the output of the following code?

```
public static void main(String[] args) {
  int value;
  System.out.println("Value is " + value);
}
```

Scope of Variables

```
public static void main(String[] args) {
  int value;
  System.out.println("Value is " + value);
}
```

`value` variable is not initialized. Code gives compile error

Scope of Variables

- What is the output of the following code?

```
public static void main(String[] args) {
  int[] a;
  System.out.println(a);
}
```

Scope of Variables

```
public static void main(String[] args) {
  int[] a;
  System.out.println(a);
}
```

Array `a` is not initialized.
Code gives compile error.
To fix it, you can write: `int[] a = null;`

Scope of Variables

- A method parameter is a local variable: The scope of a method parameter covers the entire method

- What is the output of the following code? `a` is a method parameter

```
public static void main(String[] args) {
  int val = 4;
  int result = computeSquare(val);
}
private static int computeSquare(int a) {
  System.out.println("input is: " + a);
  System.out.println("square of the input is : " + a * a);

  return a * a;
}
```

Scope of Variables

- A method parameter is a local variable: The scope of a method parameter covers the entire method

- What is the output of the following code? `a` is a method parameter

```
public static void main(String[] args) {
  int val = 4;
  int result = computeSquare(val);
}
private static int computeSquare(int a) {
  System.out.println("input is: " + a);
  System.out.println("square of the input is : " + a * a);

  return a * a;
}
```

Scope of a

Program output

```
input is: 4
square of the input is : 16
```

Scope of variables

- You can declare a local variable with the same name in different blocks in a method, but you cannot declare a local variable twice in the same block or in nested blocks.

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

  // It is fine to declare i in
  // two nonnested blocks

  for (int i = 1; i < 10; i++)
    System.out.println(i);

  for (int i = 1; i < 10; i++)
    System.out.println(i*100);
}
```

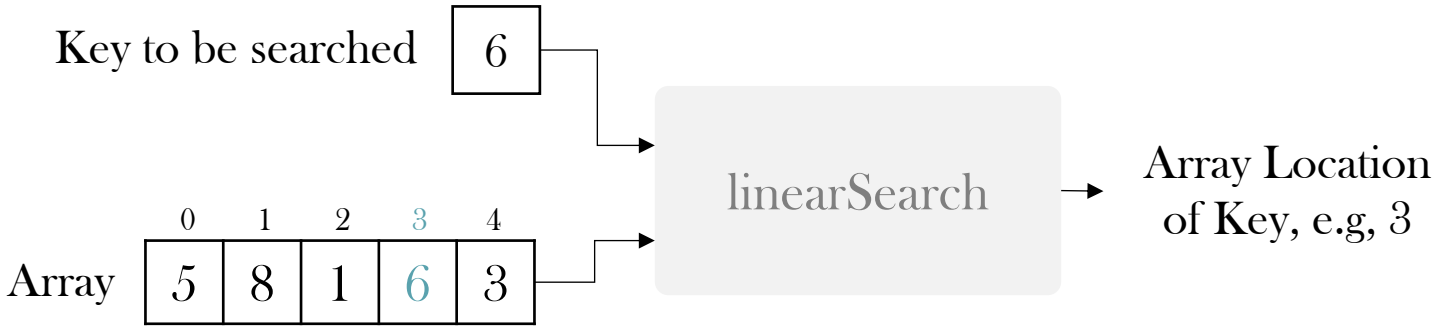
```
private static void method() {

  // It is wrong to declare i in two
  // nested blocks

  int i = 1;
  for (int i = 0; i < 10; i++) {
    System.out.println(i);
  }
}
```

Method Example

Linear Search

Linear Search

- Write a method which accepts an array and a key to be searched, and returns the array index of the key, if found. If the key is not in the array, method should return -1
  - Assume that no duplicate values are present in the input array




```
int location = linearSearch(numbersArray, 6);
```

Solution Algorithm

- Compare the key element sequentially with each element in the array until the key matches an element in the array, or the array is exhausted without a match being found
  - If a match is made, return the index of the element in the array
  - If no match is found, return -1

Source Code

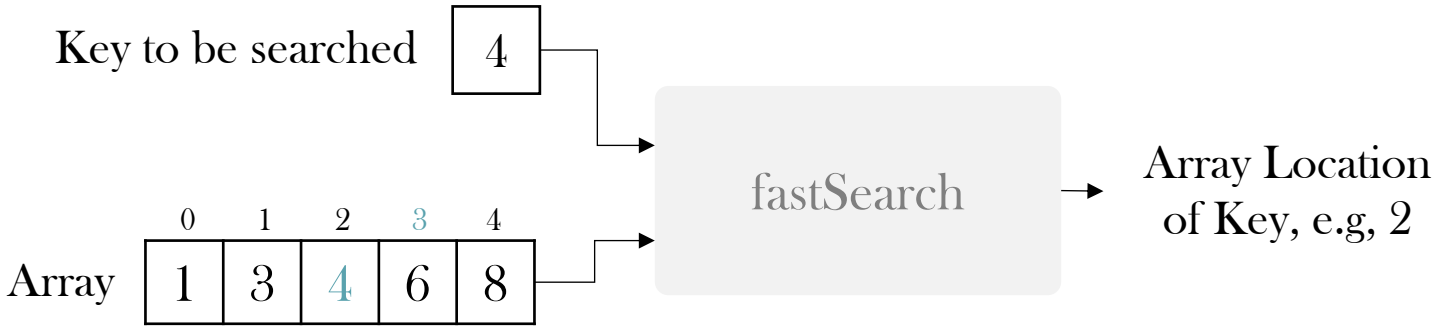
```
public class App {

  public static void main(String[] args) {
    int[] list = {8,7,4,3,2,1,9};
    int key = 1;
    int location = linearSearch(list, key);
    System.out.println("Location of " + key + " is: " + location);
  }

  /**
   * Search linearly the key in array. If key found, return the array index. If key is not found, return -1
   * @param list Input array
   * @param key Key element to be searched
   * @return Array index location of the key if found. -1 if key is not in the array
   */
  private static int linearSearch(int[] list, int key) {

    // search sequentially using a for loop
    for (int i = 0; i < list.length; i++)
      if (list[i] == key)
        return i; // key found: return the index

    return -1; // key is not found, return -1
  }
}
```

Method Example

Fast Search for Sorted Arrays

Fast Search for Sorted Arrays

- Write a method which searches a key in a sorted array (in increasing order) and returns its position if found. Returns -1 if the key is not found
- If the array is sorted, we can use a faster search algorithm than the linear search



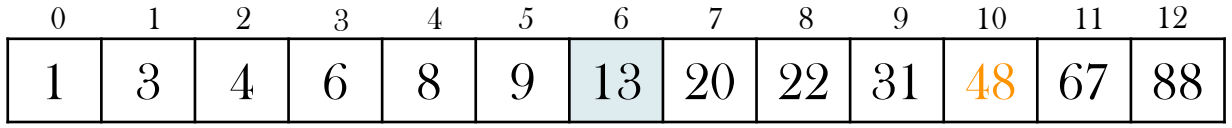

```
int location = fastSearch(numbersArray, 4);
```

Solution Algorithm

- Check the middle value of the array
  - If the key is equal to the middle array element, return the index
  - If the key is smaller than the middle array value, search the left subarray
  - If the key is larger than the middle array value, search the right subarray
- Stop and return -1, if the subarray to be searched is empty

Steps of the Algorithm

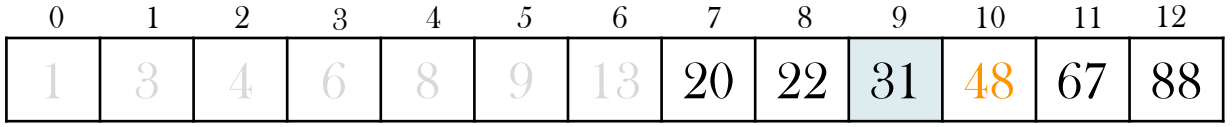
Step 1.



Key = 48

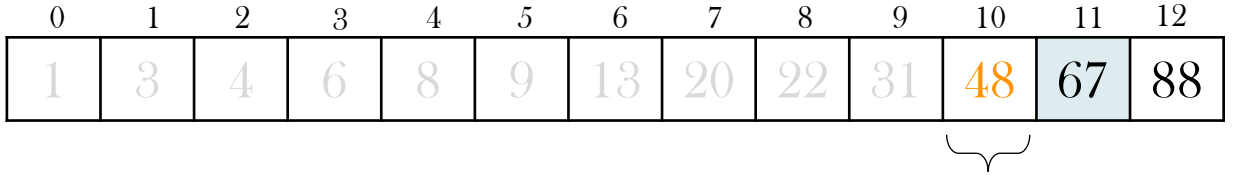
48 is greater than middle element 13.
Search the right part of the array

Step 2.



48 is greater than middle element 31.
Search the right part of the array

Step 3.

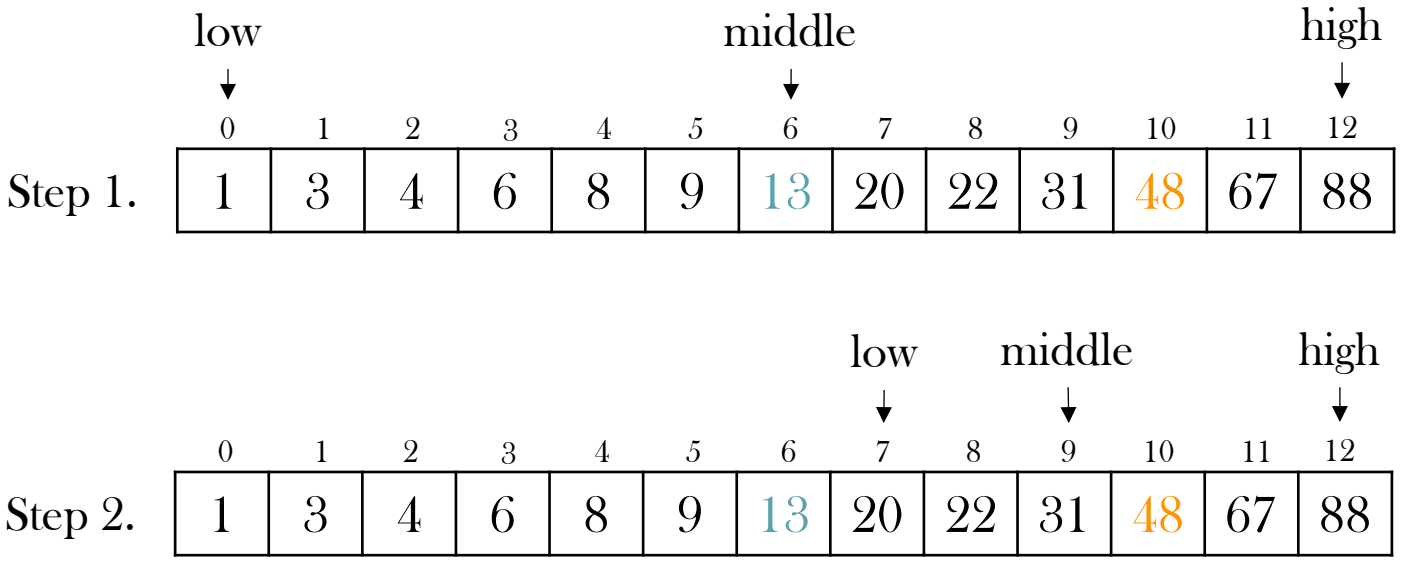


48 is less than middle element 67.
Search the left part of the array

→ Algorithm finds 48 at location [10] and stops.

Implementation Details

- Use two indexes: low and high
  - Low is the leftmost position of the subarray
  - High is the rightmost position of the subarray
- Compute middle position: (low+high)/2
- Stop the algorithm when low >= high



Source Code

```
/**
 * Performs fast search for key in a sorted array.
 * @param a Sorted array in increasing order
 * @param key Key to be searched
 * @return Location of the key in array. -1 if key is not found.
 */
private static int fastSearch(int[] a, int key) {
  int low = 0;
  int high = a.length-1;
  while (high >= low) {
    int middle = (high + low) / 2;
    if (a[middle] == key)
      return middle;
    else if (key > a[middle])
      low = middle + 1;
    else
      high = middle - 1;
  }
  return -1;
}
```

Method Example

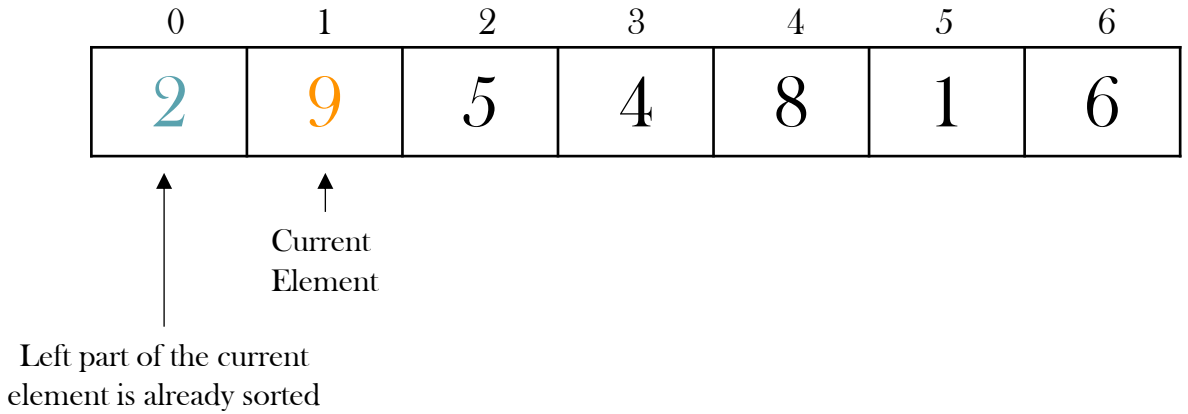
Sorting An Array

Sort an Array In Increasing Order

- Our algorithm sorts a list of values by repeatedly inserting a new element into a sorted sublist until the whole array is sorted
- Pseudocode of the algorithm

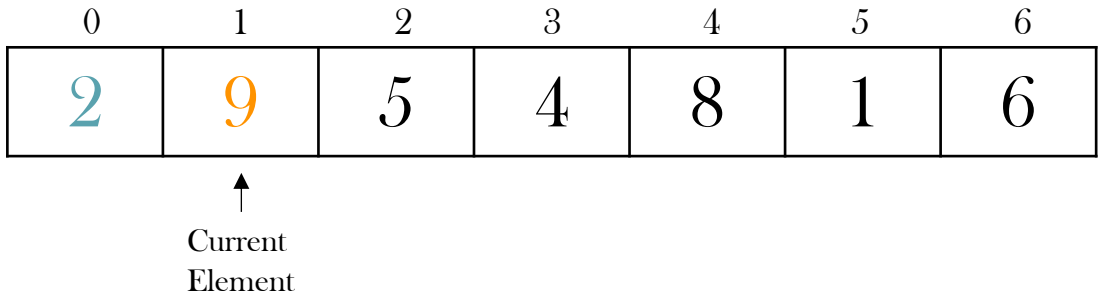
```
for each array element, starting from 2nd position, do
  insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted
```

Steps

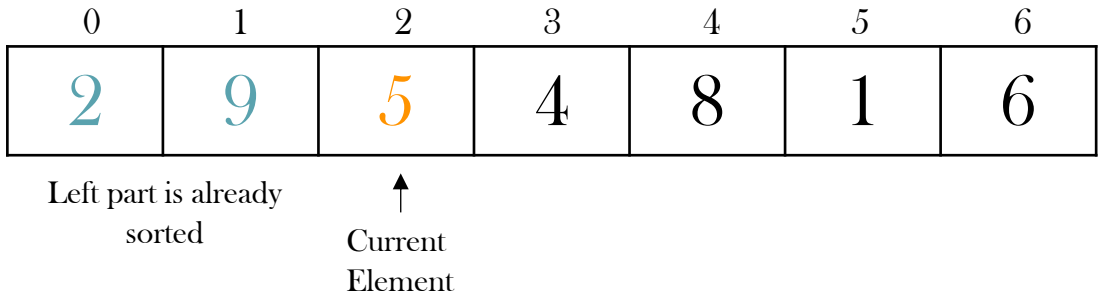


Step 1. Try to insert 9 into the left part. In this case, 2 is less than 9, do nothing

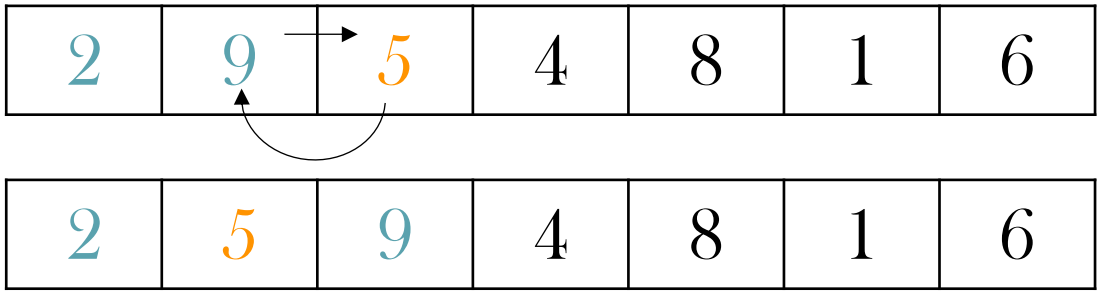
Steps



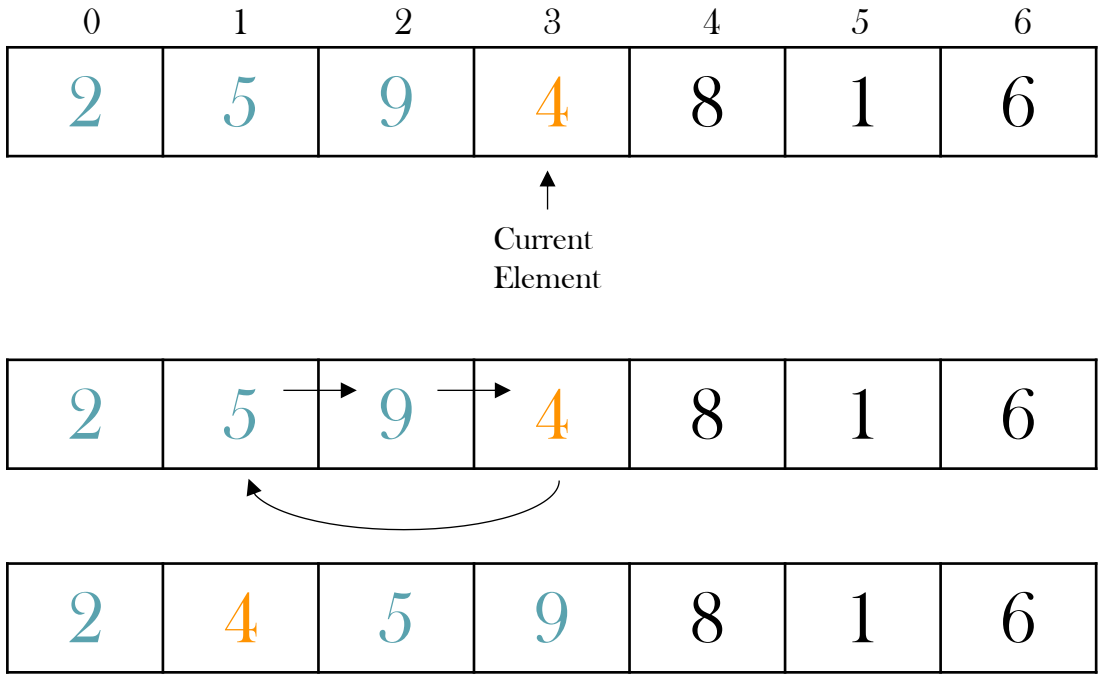
Step 1. Try to insert 9 into the left part. In this case, 2 is less than 9, do nothing



Step 2. Insert 5 into the left part. In this case, 5 should go to a[1] and 9 should move to the right into the position a[2]

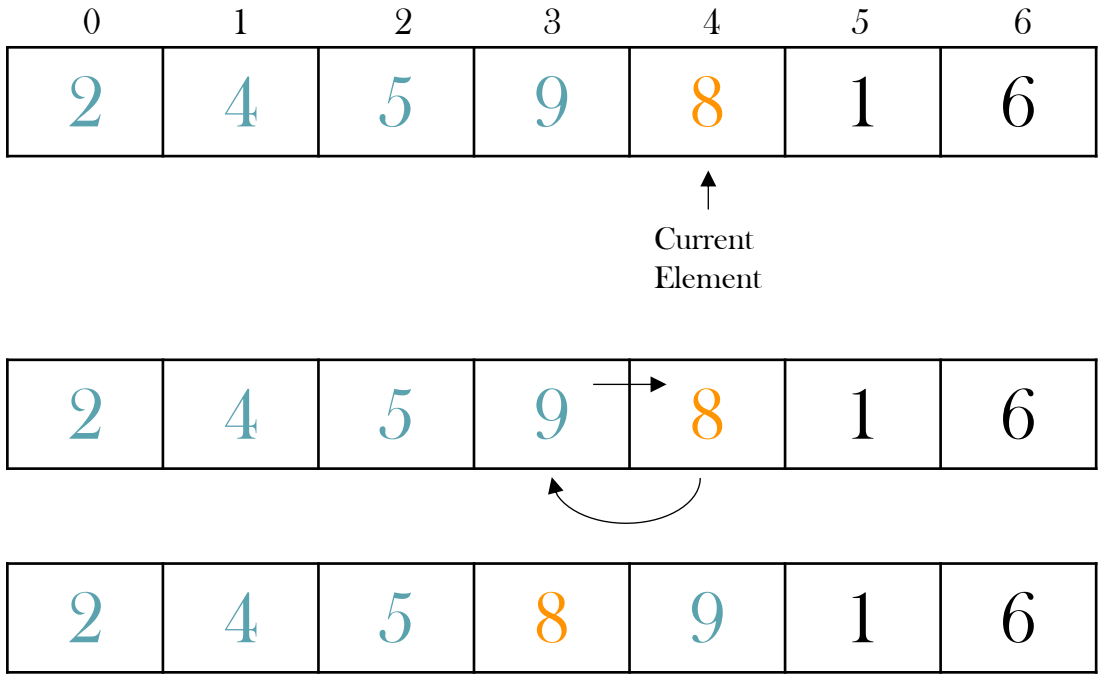


Steps



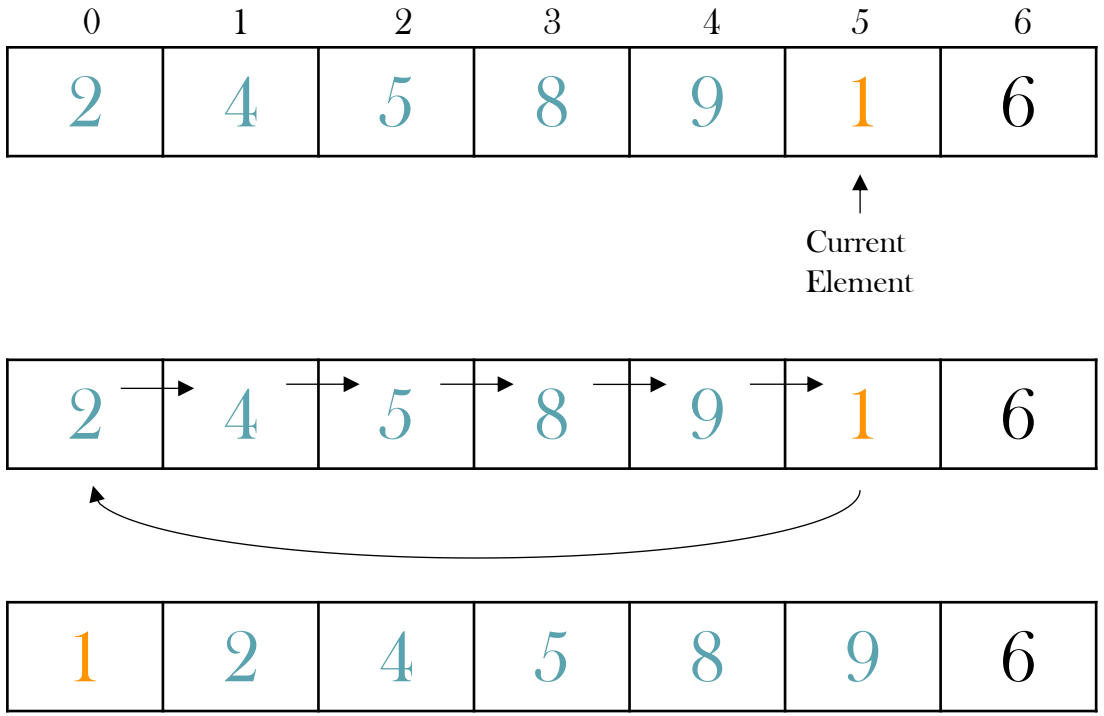
Step 3. Insert 4 into the left part. Move all elements greater than 4 one position to the right and place 4 into a[1]

Steps



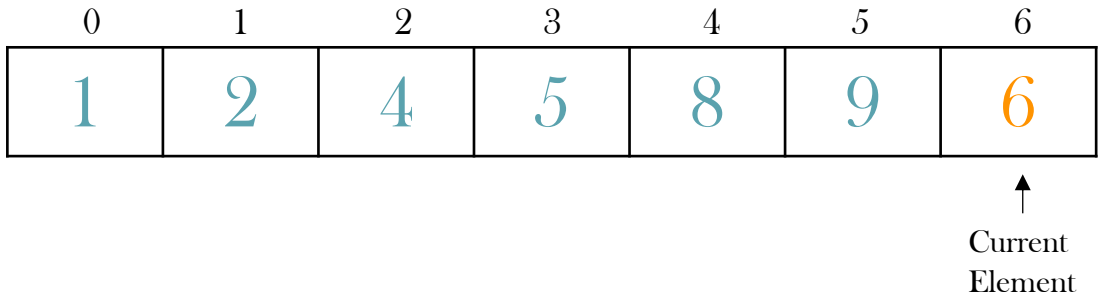
Step 4. Insert 8 into the left part. Move all elements greater than 8 one position to the right and place 8 into a[3]

Steps

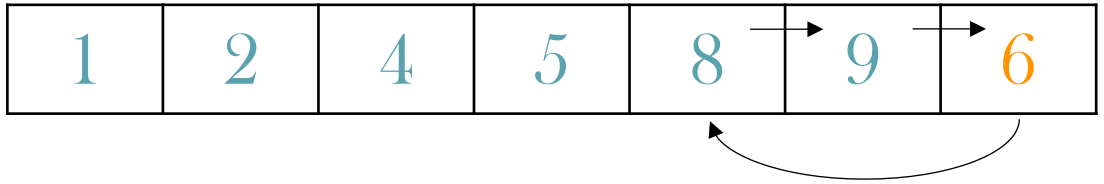


Step 5. Insert 1 into the left part. Move all elements greater than 1 one position to the right and place 1 into a[0]

Steps



Step 6. Insert 6 into the left part. Move all elements greater than 6 one position to the right and place 6 into a[4]



Algorithm stops. Array is sorted in increasing order.

Source Code

```
public class App {

  public static void main(String[] args) {
    int[] a = {2,9,5,4,8,1,6};
    sortAlgorithm(a);
    System.out.println(Arrays.toString(a));
  }
  // code continues
}
```

Source Code

```
/**
 * Sorts integers in increasing order
 * @param a Input array to be sorted
 */
private static void sortAlgorithm(int[] a) {

  // Insert a[i] into a sorted sublist a[0..i-1]
  for (int i = 1; i < a.length; i++) {
    int currentElement = a[i];
    int j;
    // Traverse from (i-1)th location towards the beginning of the array
    for (j = i-1; j >= 0; j--)
      // If jth element is greater than currentElement, move the jth element one position to the right
      if (a[j] > currentElement)
        a[j+1] = a[j];
      else
        // Else, if jth element is smaller than the current element, exit from the loop
        break;
    a[j+1] = currentElement; // store current element to (j+1)th location
  }
}
```