Methods

Forest Agostinelli
University of South Carolina

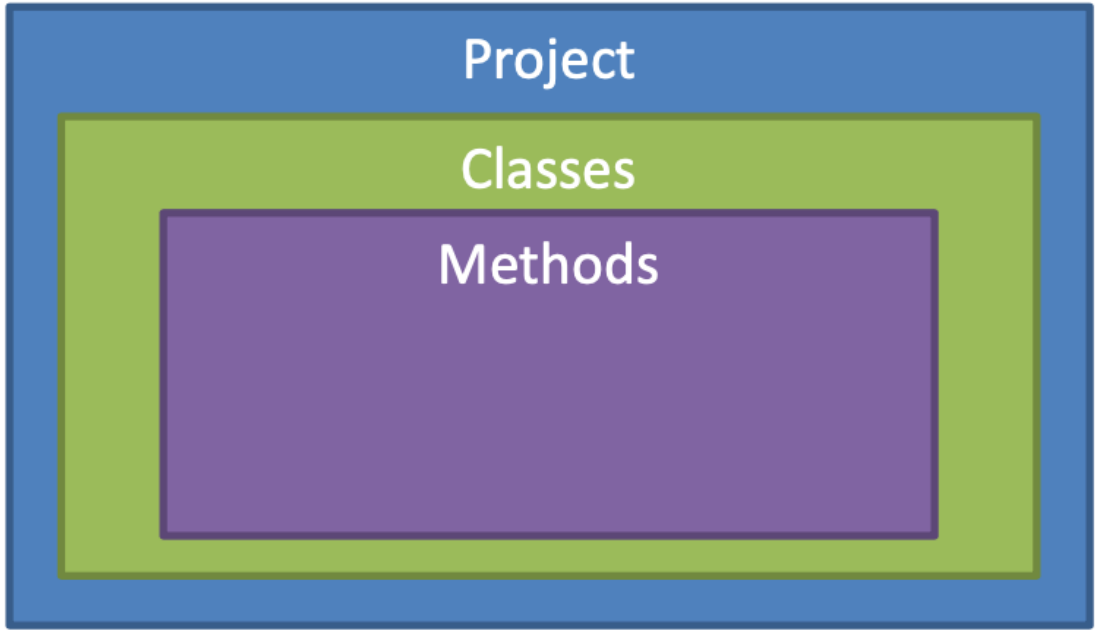
Outline

- Overview
- Call stack
- Example
- Algorithms for problem solving

Overview

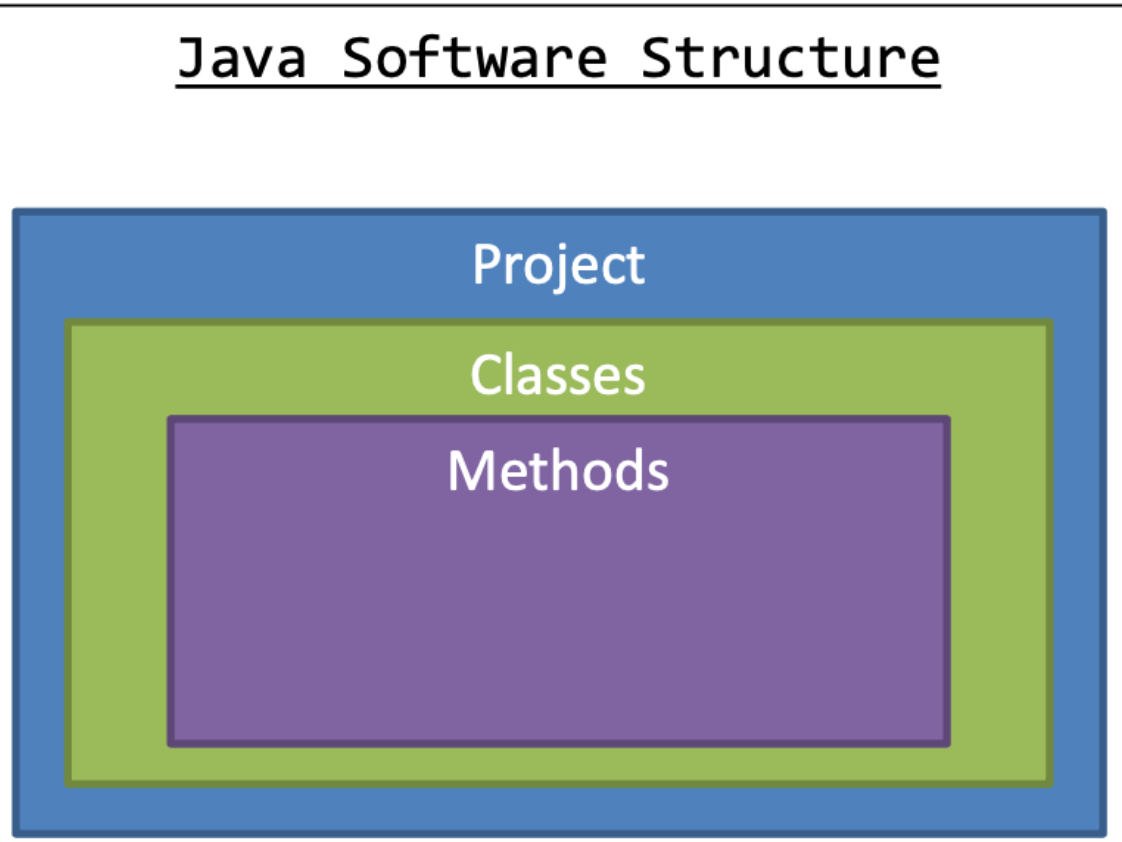
- Organized and structured code helps to:
  - Reuse parts of code, so you use less statements
  - Quickly find bugs or errors
  - Easily add or extend functionality
- Java Organizes Software
  - First in Projects
  - Then in Classes
  - Then in Methods

Java Software Structure



Overview

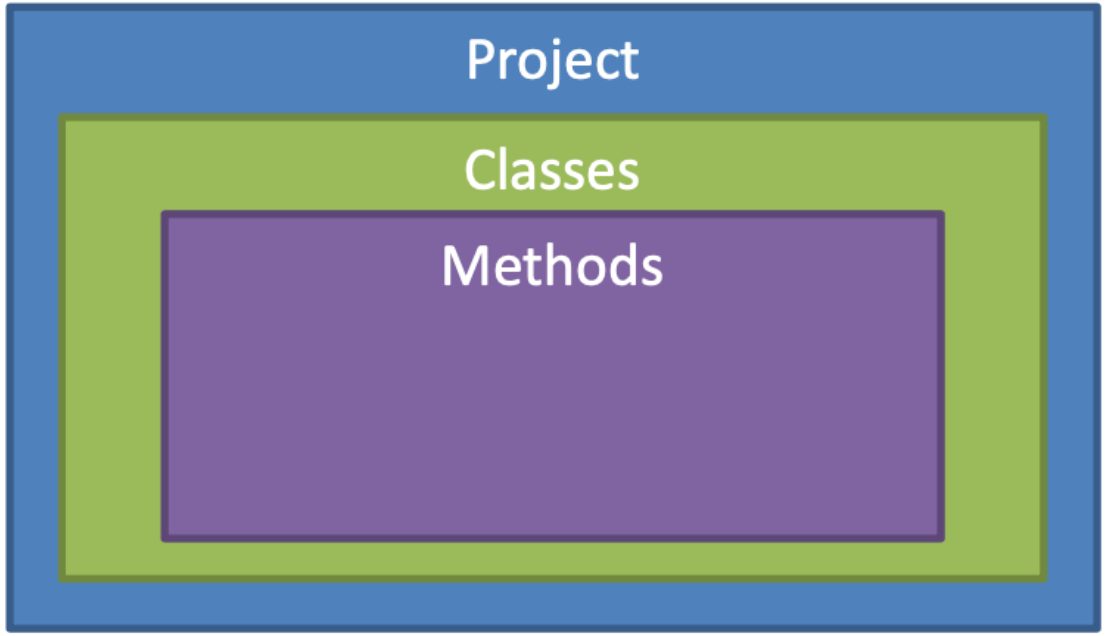
- Methods are where we write *functional* code
  - Declare and use “local”/ “temporary” / “method” variables
  - Branching Statements
  - Loops
- Using a method is referred to as “invoking” or “calling”
- We have only used the *main method* so far
  - Entry point of software
  - All functional code has been written inside of the main method
  - Called by the system



Overview

- Creating other methods organize code into *actions*
- Methods can be thought of as “verbs”
  - They act as actions or functionality in software
- Methods in Java must be written inside body of “classes”
  - Within the curly braces(“{}”) of a class
  - Methods cannot be defined inside of other methods only inside of classes

Java Software Structure



Overview

- Defining a simple method requires the following:
  - Scope: Where this method can be called
  - Return Type: What value does this method return
  - Identifier: The *callable* name of the method
  - Parameters: Arguments / information passed to the method
  - Body: Curly braces denoting the code that belongs to the method

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public void greetings()
{
        System.out.println("Hello World");
}
```

Overview

- Scope is where the method can be *called*
- The scope “public” indicates it can be called inside and outside of the class
- The scope “private” indicates it can only be called within the class and not outside
- There are other scopes, but we will focus on these

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public void greetings()
{
        System.out.println(“Hello World”);
}
```

Overview

- Return Type is a value that the method *returns* after it has completed
- This can be any data type
- The special type “void” means the method returns nothing
- Any return type that is **not void** must use the reserved word “return” followed by that type of data

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public void greetings()
{
      System.out.println(“Hello World”);
}
```

Overview

- Return Type is a value that the method *returns* after it has completed
- This can be any data type
- The special type “void” means the method returns nothing
- Any return type that is **not void** must use the reserved word “return” followed by that type of data

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public String getGreetingsString()
{
        return “Hello World”;
}
```

Overview

- Method Identifiers (“id”) follow the same rules as variables
- Identifiers may contain ONLY
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
- Identifiers CANNOT contain
  - Spaces of any kind
  - Digit as the First Character
  - Dots “.”
  - Asterisks “*”
  - Other types of special characters

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public String getGreetingsString()
{
        return “Hello World”;
}
```

Overview

- Method Identifiers ("id") follow the same rules as variables
- Identifiers are Case Sensitive
- Identifiers CANNOT be a **reserved word**
- Identifiers start with a Lowercase Character
- Multiword identifiers are "punctuated" using uppercase characters
- Methods should have meaningful identifiers
  - Clearly indicate what type of action(s) the method will perform
  - Use *verbiage* as the method's identifiers

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public String getGreetingsString()
{
        return "Hello World";
}
```

Overview

- Parameters allow information to be *given / passed* to the method from outside of it
- Placed inside the parenthesis
- Act as variables for the method
  - Requires a type and an identifier
- Multiple parameters require a comma "," separating them
  - All parameters require a type and an identifier
- The scope is only within the body of the method they are defined

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public double inchesToCentimeters(double inches)
{
        return inches * 2.54;
}
```

Overview

- Parameters allow information to be *given / passed* to the method from outside of it
- Placed inside the parenthesis
- Act as variables for the method
  - Requires a type and an identifier
- Multiple parameters require a comma "," separating them
  - All parameters require a type and an identifier
- The scope is only within the body of the method they are defined

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public boolean isGreaterThan(int a, int b)
{
        return a > b;
}
```

Overview

- The body of the method is where we place *functional* code
  - Declare and use “local”/ “temporary” / “method” variables
  - Branching Statements
  - Loops
- Variables declared inside of a method’s body cannot be used outside of that method

Defining a Method

```
<<scope>> <<return type>> <<id>> (<<parameters>>)
{
        <<Body of the method>>
}
```

Example

```
public boolean isGreaterThan(int a, int b)
{
      return a > b;
}
```

Overview

- Using a method is referred to as “invoking” or “calling” a method
- When a method is called the program *jumps* to that method and starts running the code
- Once that method has completed it *jumps back* from where it was called
- Calling a method from inside of the class where it was defined requires using its identifier and parameters

Defining a Method

```
<<id>>(<<parameters>>);
```

Example

```
public void printGreeting()
{
        String str = getGreetingString();
        System.out.println(str);
}
public String getGreetingString()
{
        return “Hello World”;
}
```

Overview

- Calling a method from outside of the class where it was defined requires:
  - The method to have the "public" scope
  - An *instance* of that class to be constructed
  - Using that instance followed by the dot (".") followed by the method's identifier and parameters
- Creating an instance of the class (object) requires declaring a variable and then constructing it by using "new" followed by the class type and parenthesis
  - This is how Scanner has worked
  - Calling a method from an object that has not been constructed will cause a run-time error called a *NullPointerException/NullReferenceException*

Defining a Method

```
//Create an instance of the class
<<class type>> <<id>> = new <<class type>>();
<<id>>.<<method id>>(<<parameters>>);
```

Example

```
public class GreetingsProgram
{
	public static void main(String[] args)
	{
		GreetingsProgram g = new GreetingsProgram();
		g.printGreetings();//Calls the method
	}
	…
}
```

Overview

- This is how we would call a method from the main method
  - Cannot directly call a method from the main method without creating an instance of the class (object)
  - We will discuss why in a future lecture

Defining a Method

```
//Create an instance of the class
<<class type>> <<id>> = new <<class type>>();
<<id>>.<<method id>>(<<parameters>>);
```

Example

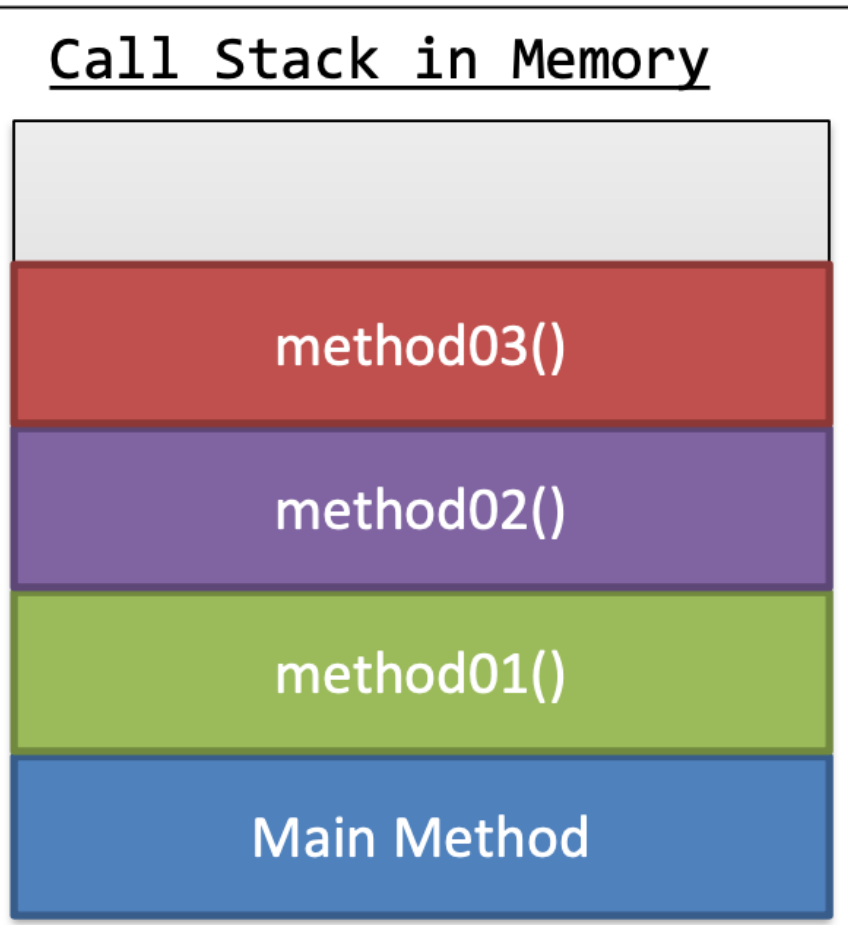
```
public class GreetingsProgram
{
	public static void main(String[] args)
	{
		GreetingsProgram g = new GreetingsProgram();
		g.printGreetings();//Calls the method
	}
	…
}
```

Outline

- Overview
- Call stack
- Example
- Algorithms for problem solving

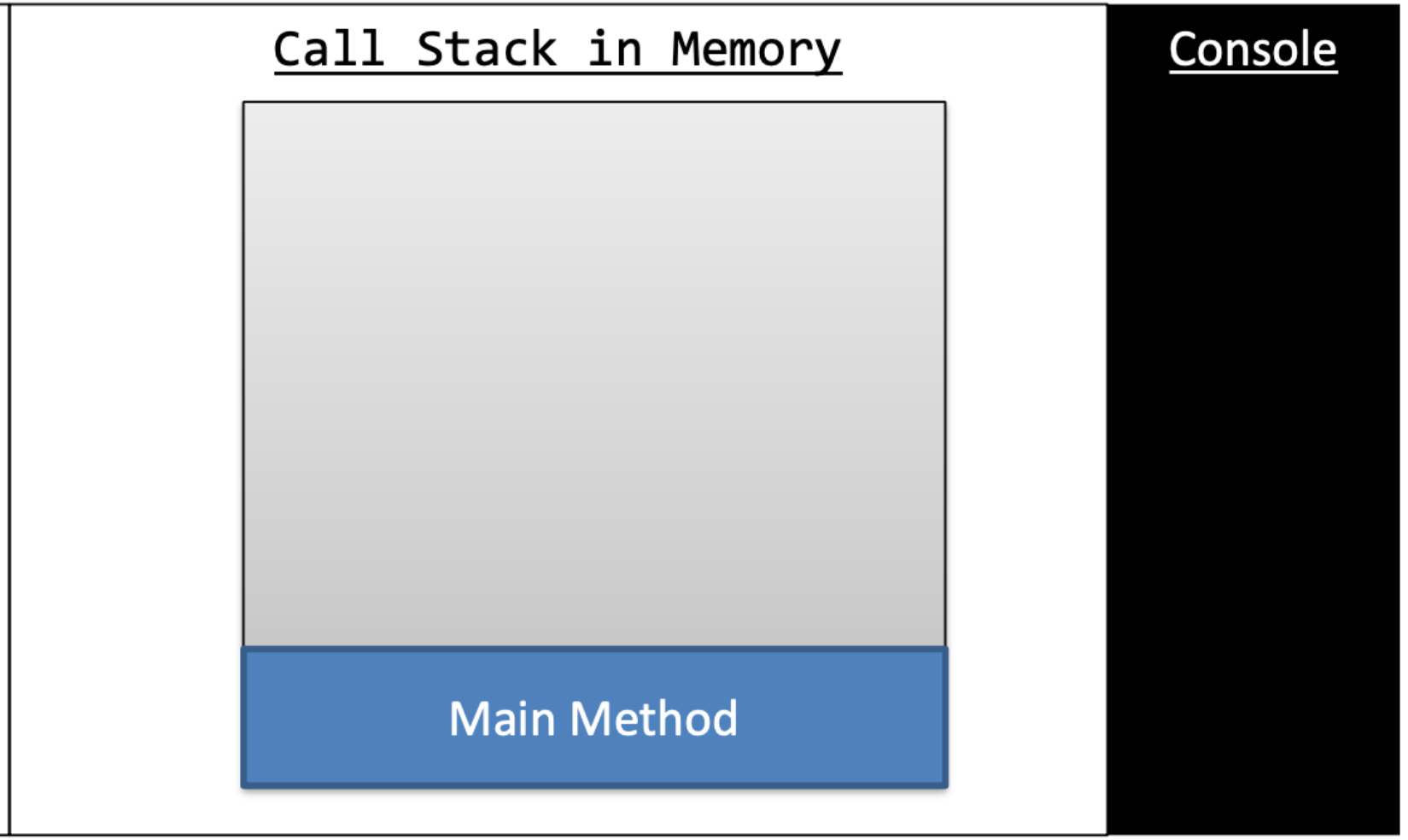
Call Stack

- Programs have different sections of memory
  - Stack / Call Stack
  - Heap
  - Data (Global)
  - Text
- When a method is called it is *pushed* onto the call stack
- When a method completes it is *popped* off of the call stack



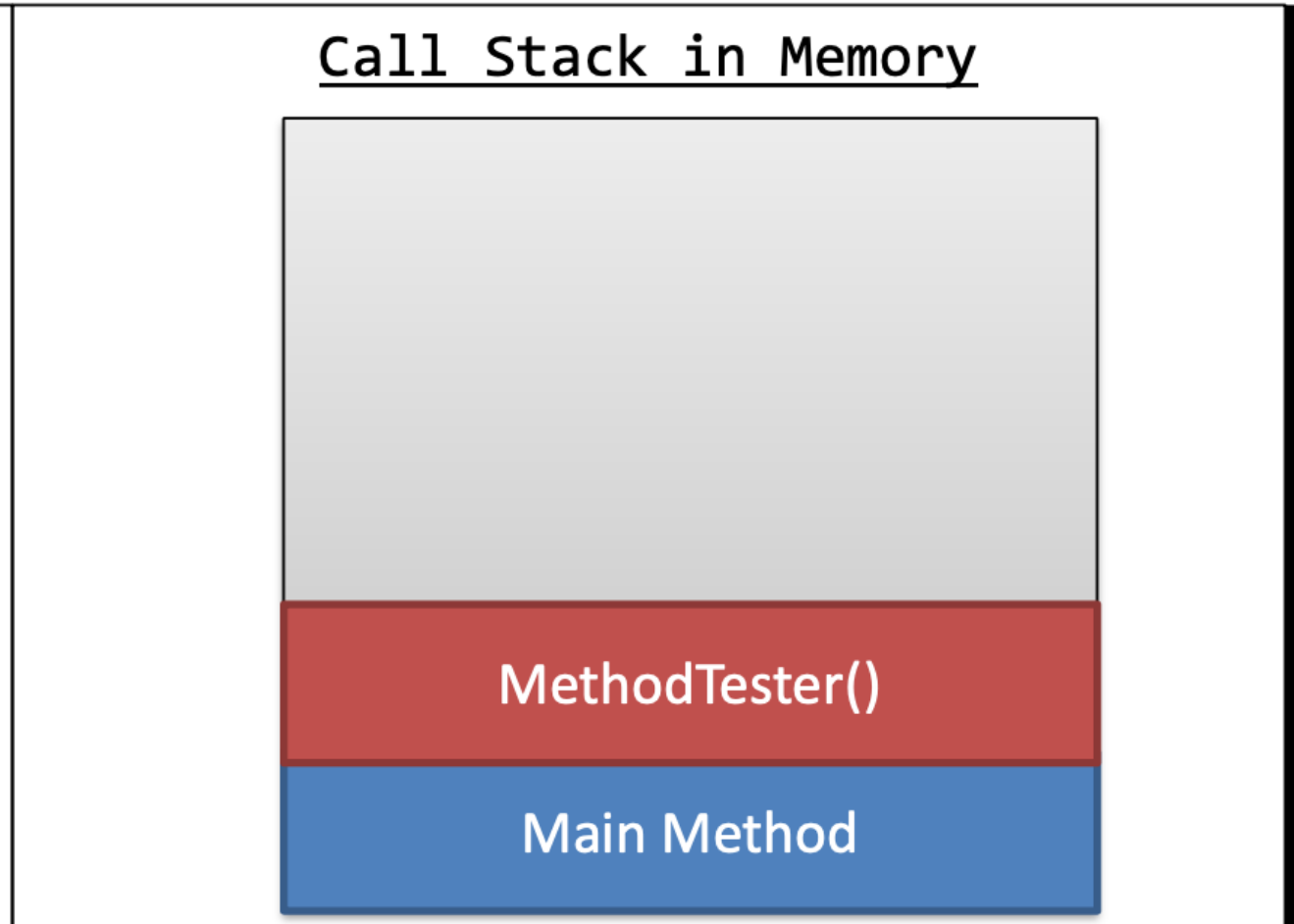
Call Stack

```
public class MethodTester {
	public static void main(String[] args)
	{
		MethodTester m = new MethodTester();
		m.start();
	}
	public void start()
	{
		System.out.println("Start");
		printOne();
	}
	public void printOne()
	{
		System.out.println("One");
		printTwo();
	}
	public void printTwo()
	{
		System.out.println("Two");
		printThree();
	}
	public void printThree()
	{
		System.out.println("Three");
	}
}
```



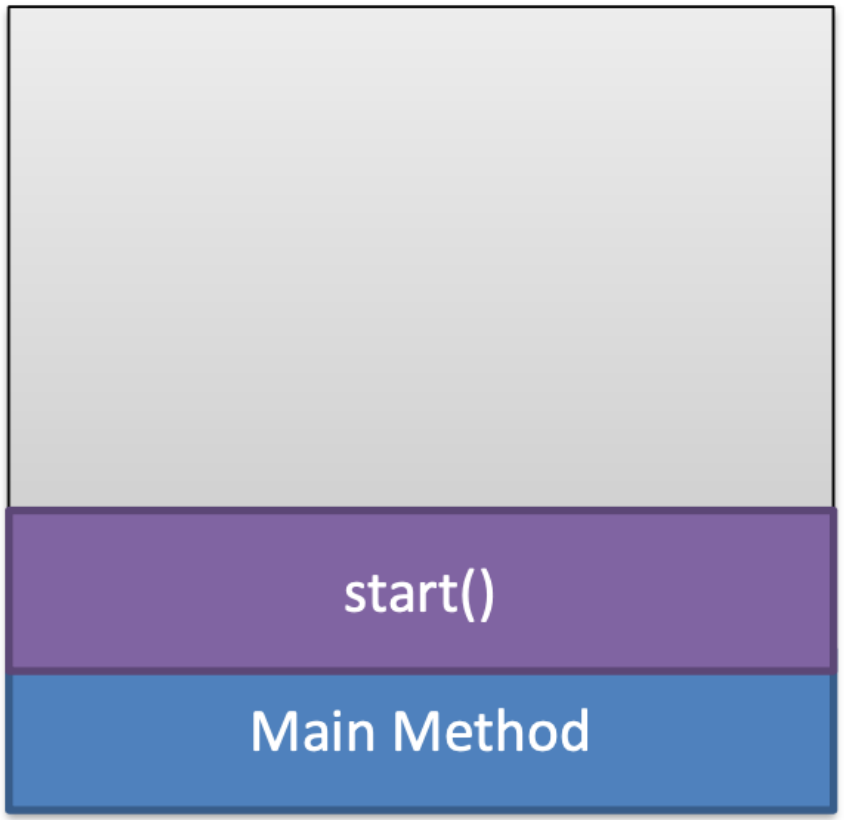
Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Call Stack

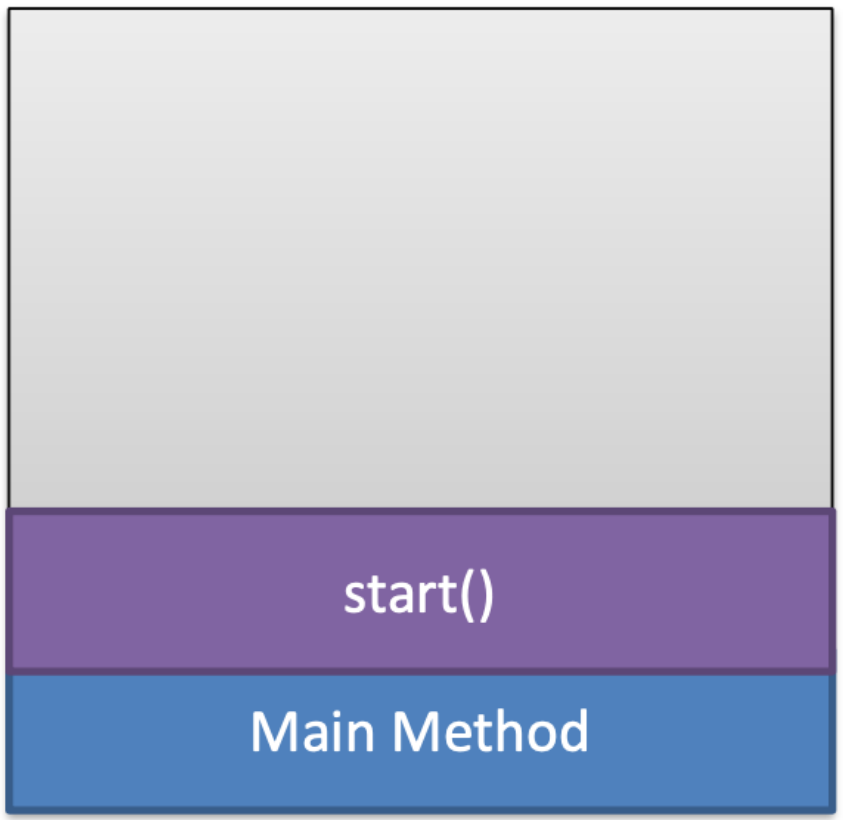
```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Console

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

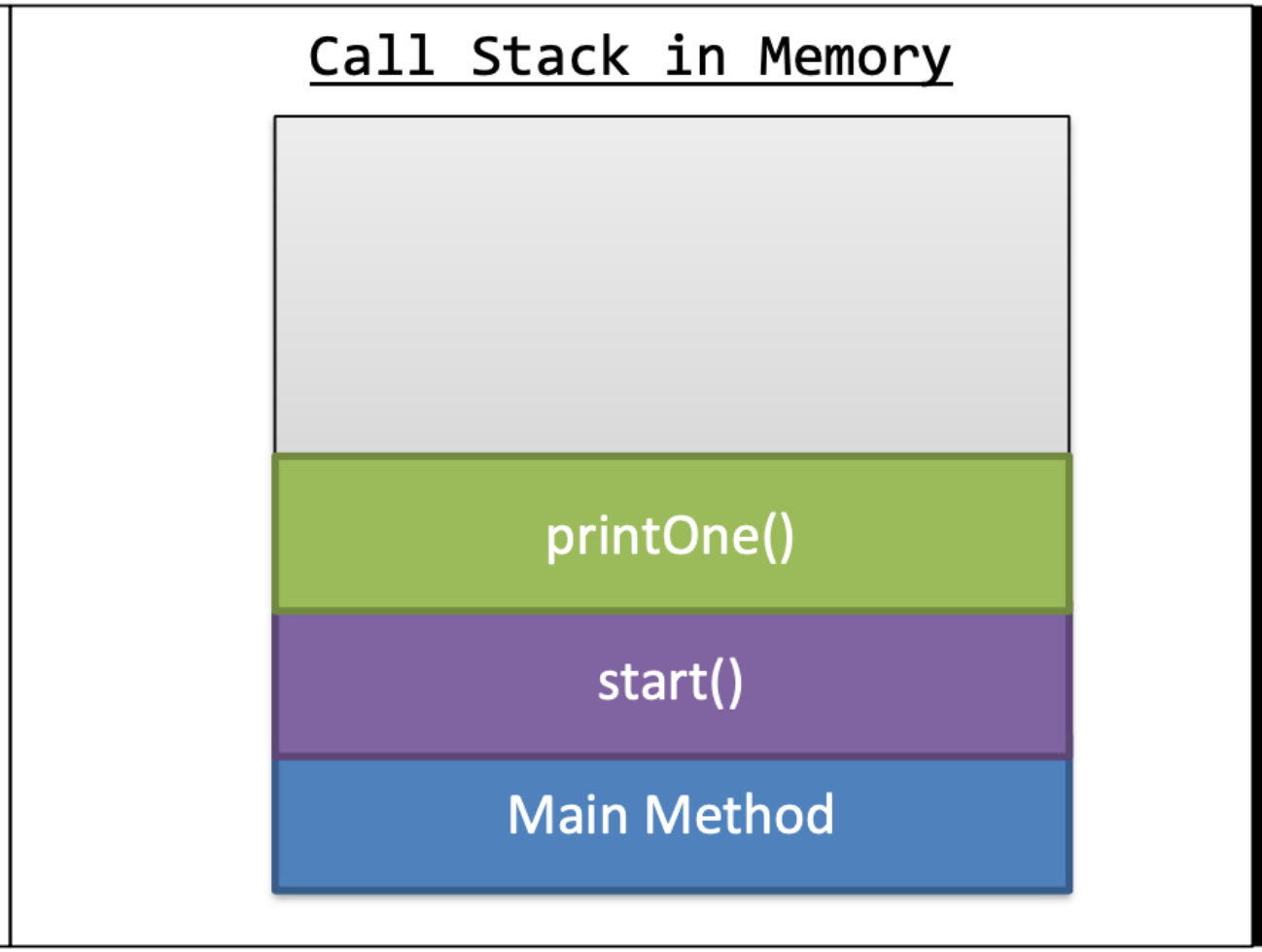


Console

Start

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

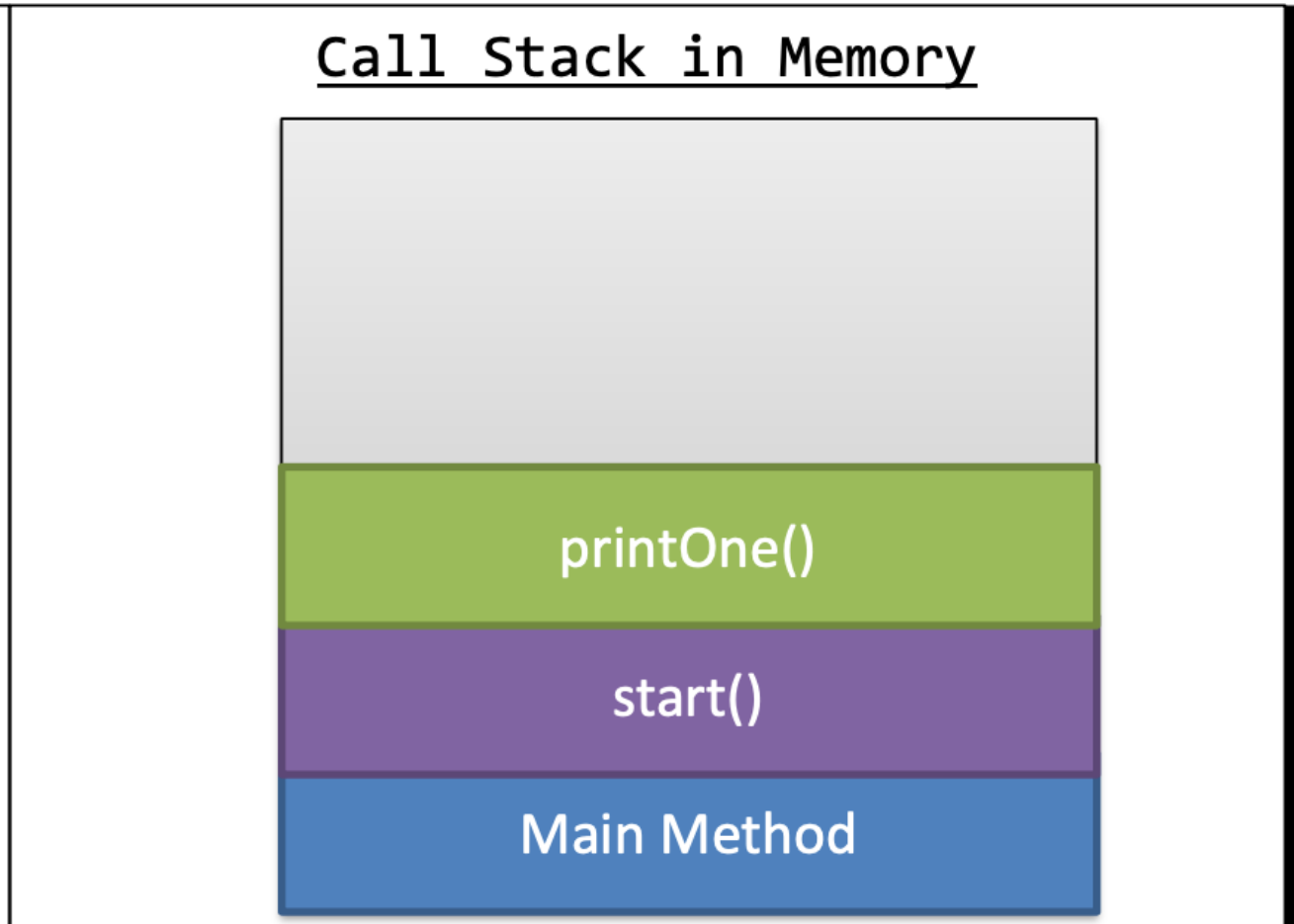


Console

Start

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Console

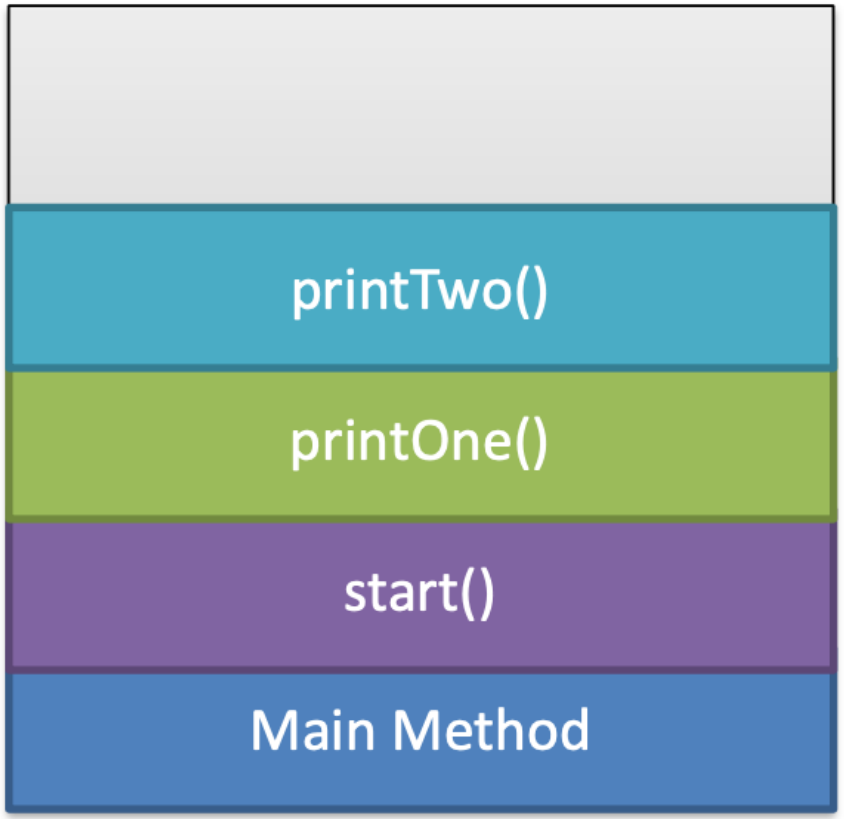
Start

One

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



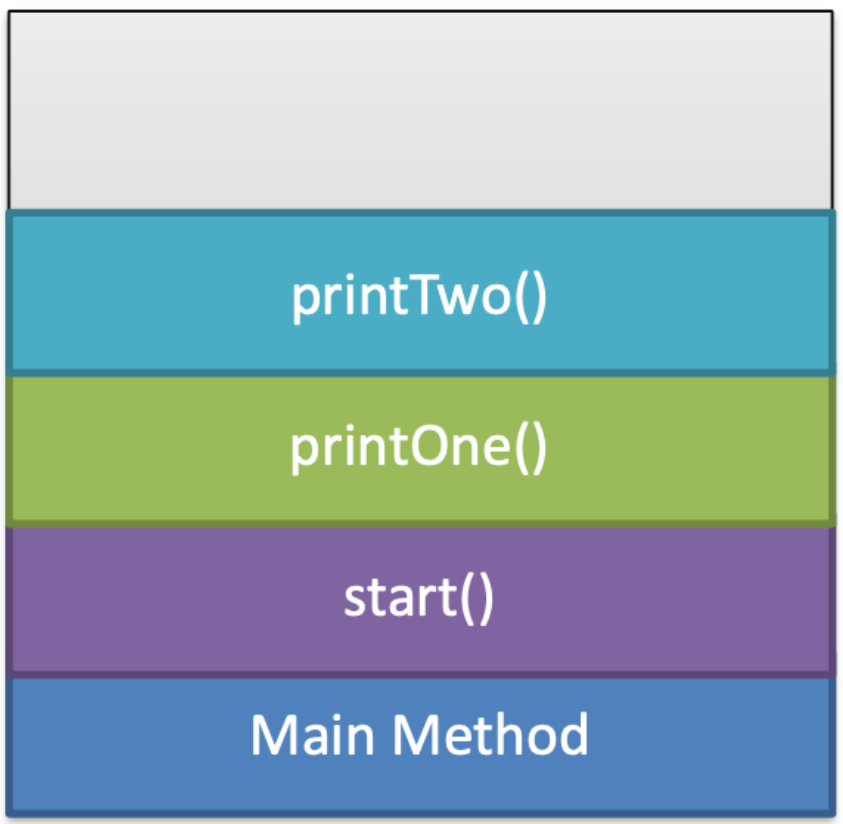
Console

Start

One

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Console

Start
One
Two

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Console

Start
One
Two

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory

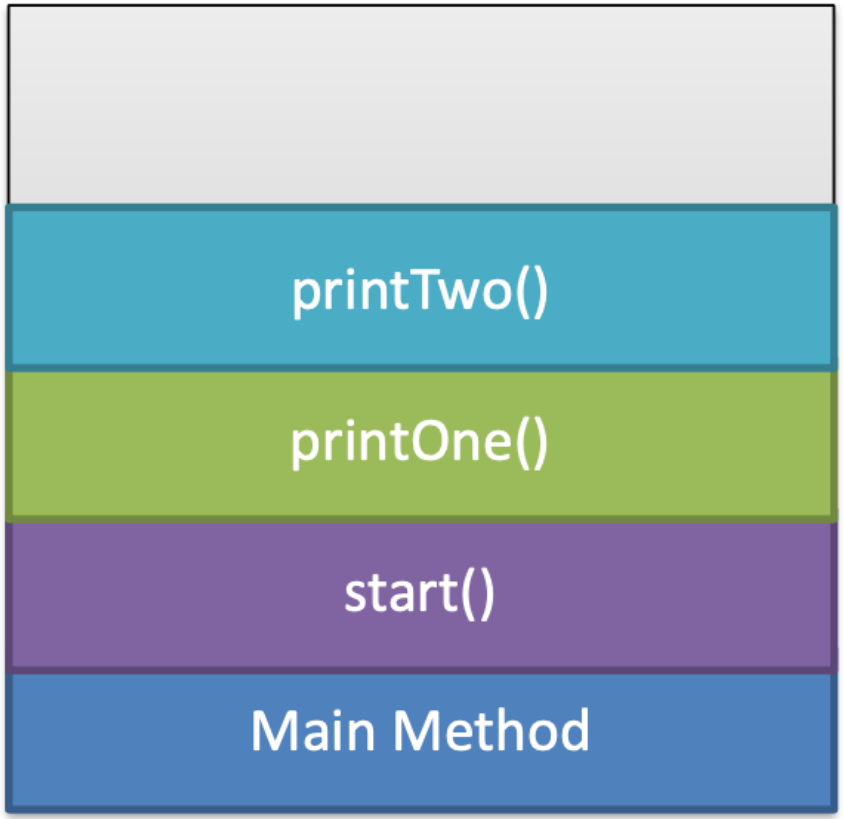
Console

Start
One
Two
Three

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



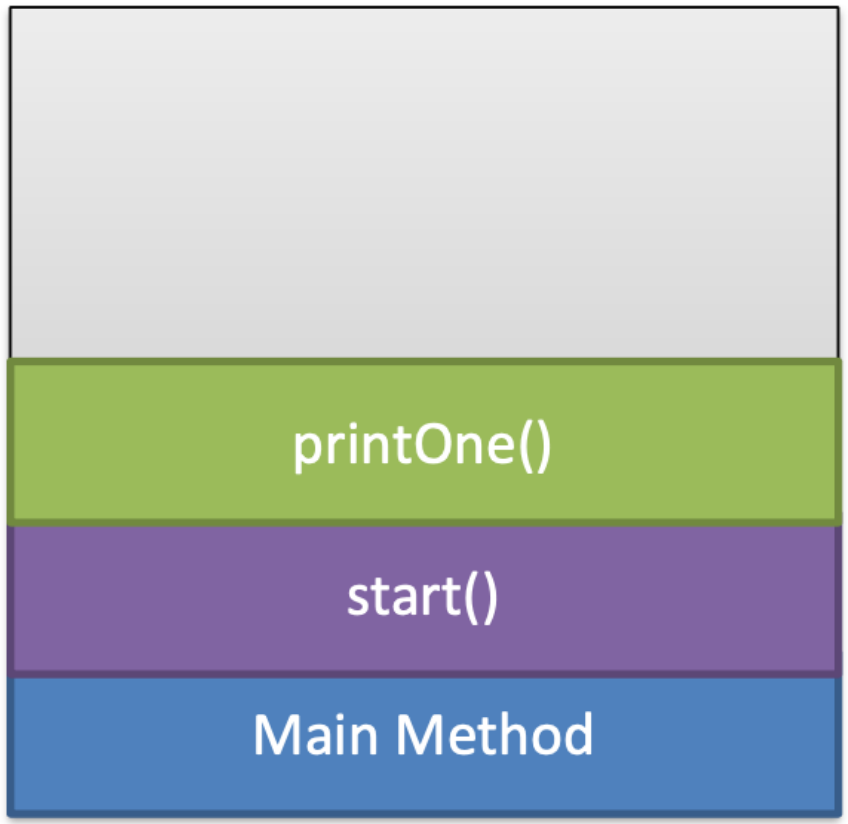
Console

Start
One
Two
Three

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory

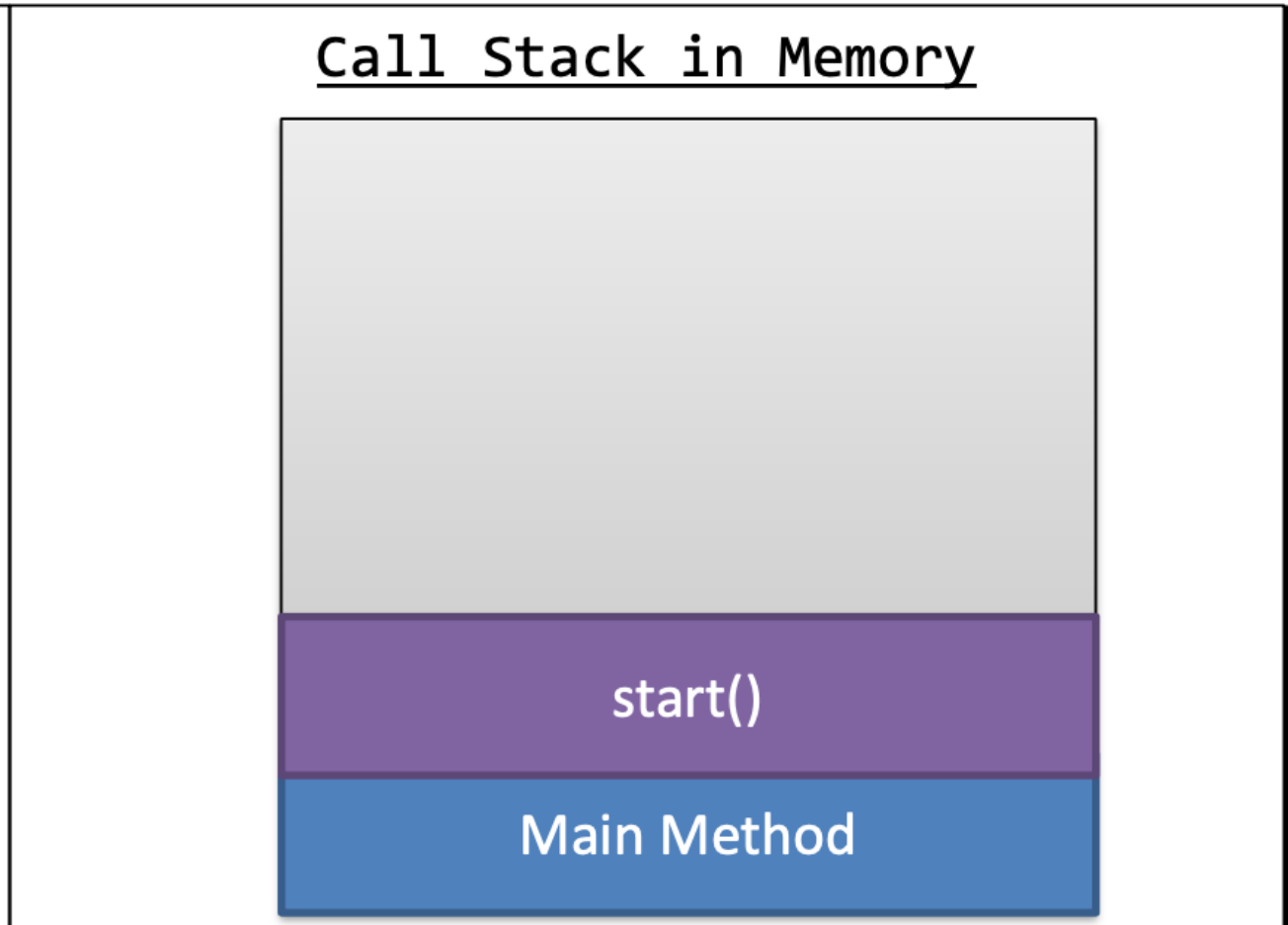


Console

Start
One
Two
Three

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

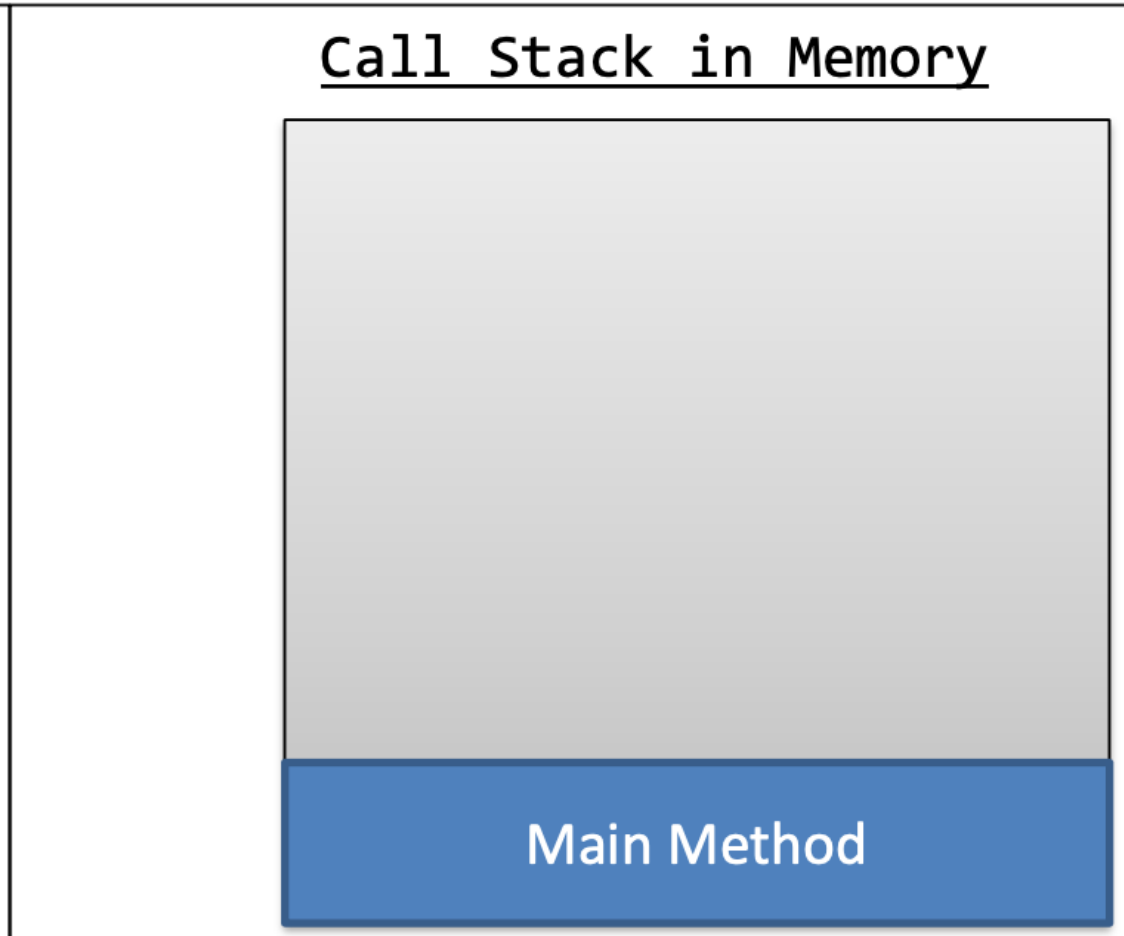


Console

Start
One
Two
Three

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                System.out.println("Start");
                printOne();
        }
        public void printOne()
        {
                System.out.println("One");
                printTwo();
        }
        public void printTwo()
        {
                System.out.println("Two");
                printThree();
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Console

Start

One

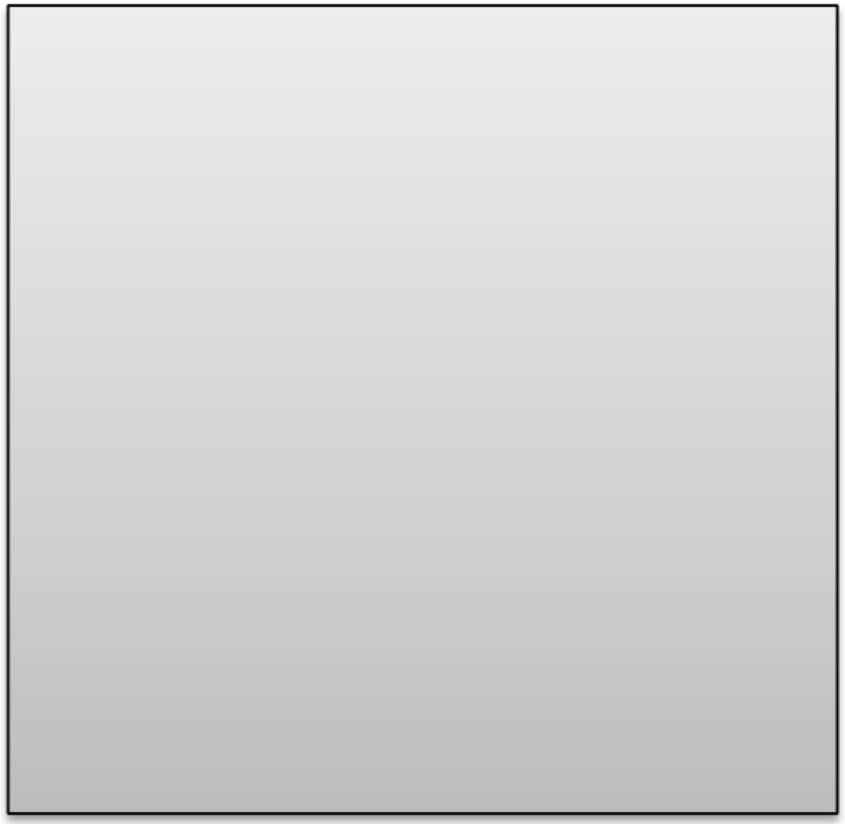
Two

Three

Call Stack

```
public class MethodTester {
	public static void main(String[] args)
	{
		MethodTester m = new MethodTester();
		m.start();
	}
	public void start()
	{
		System.out.println("Start");
		printOne();
	}
	public void printOne()
	{
		System.out.println("One");
		printTwo();
	}
	public void printTwo()
	{
		System.out.println("Two");
		printThree();
	}
	public void printThree()
	{
		System.out.println("Three");
	}
}
```

Call Stack in Memory



Console

Start

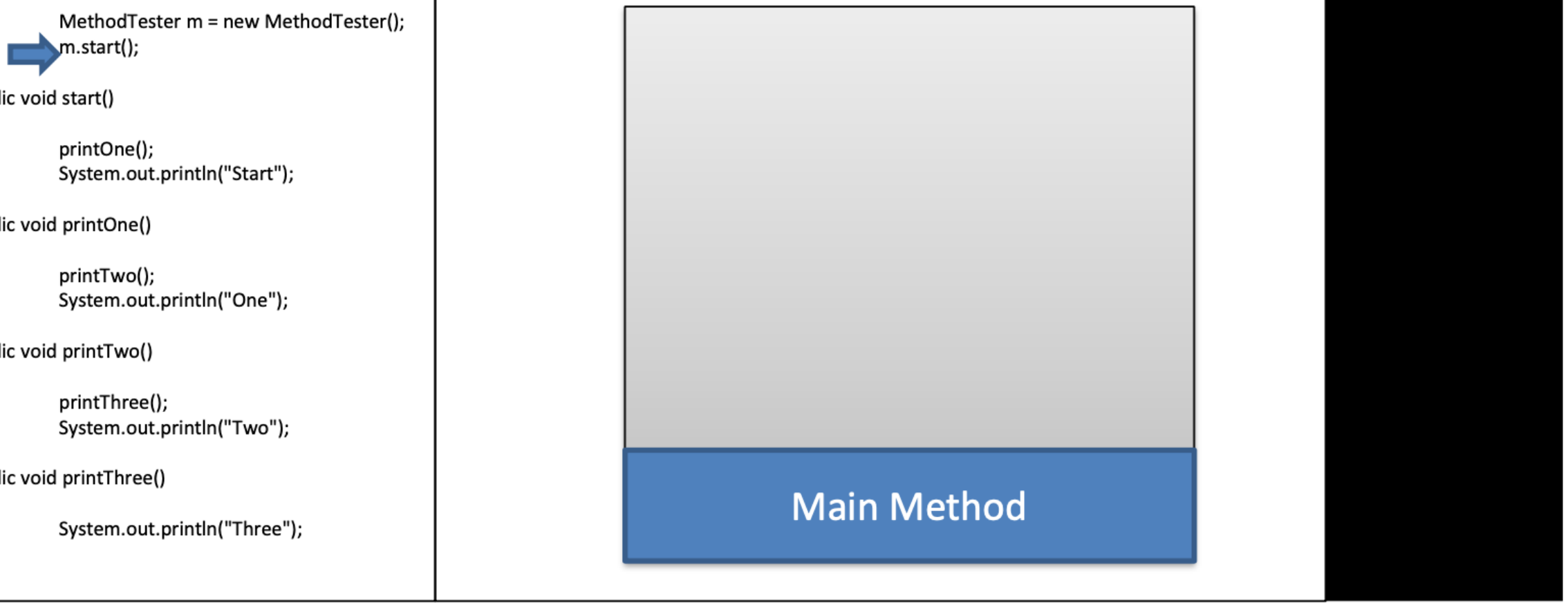
One

Two

Three

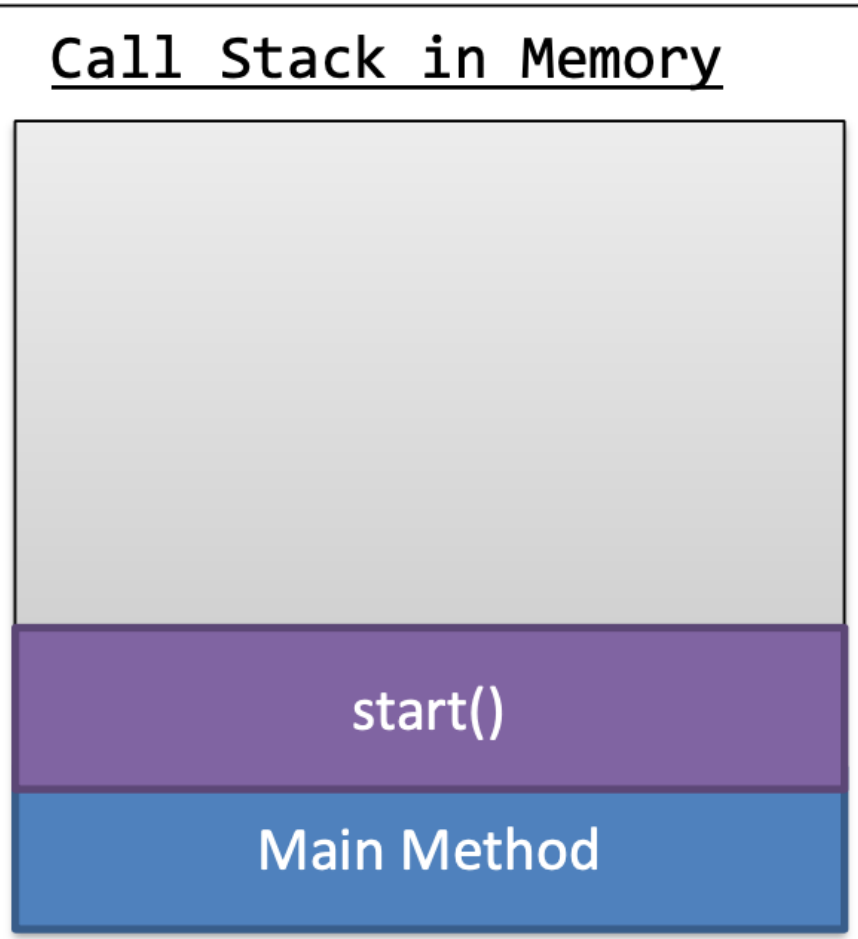
Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Call Stack

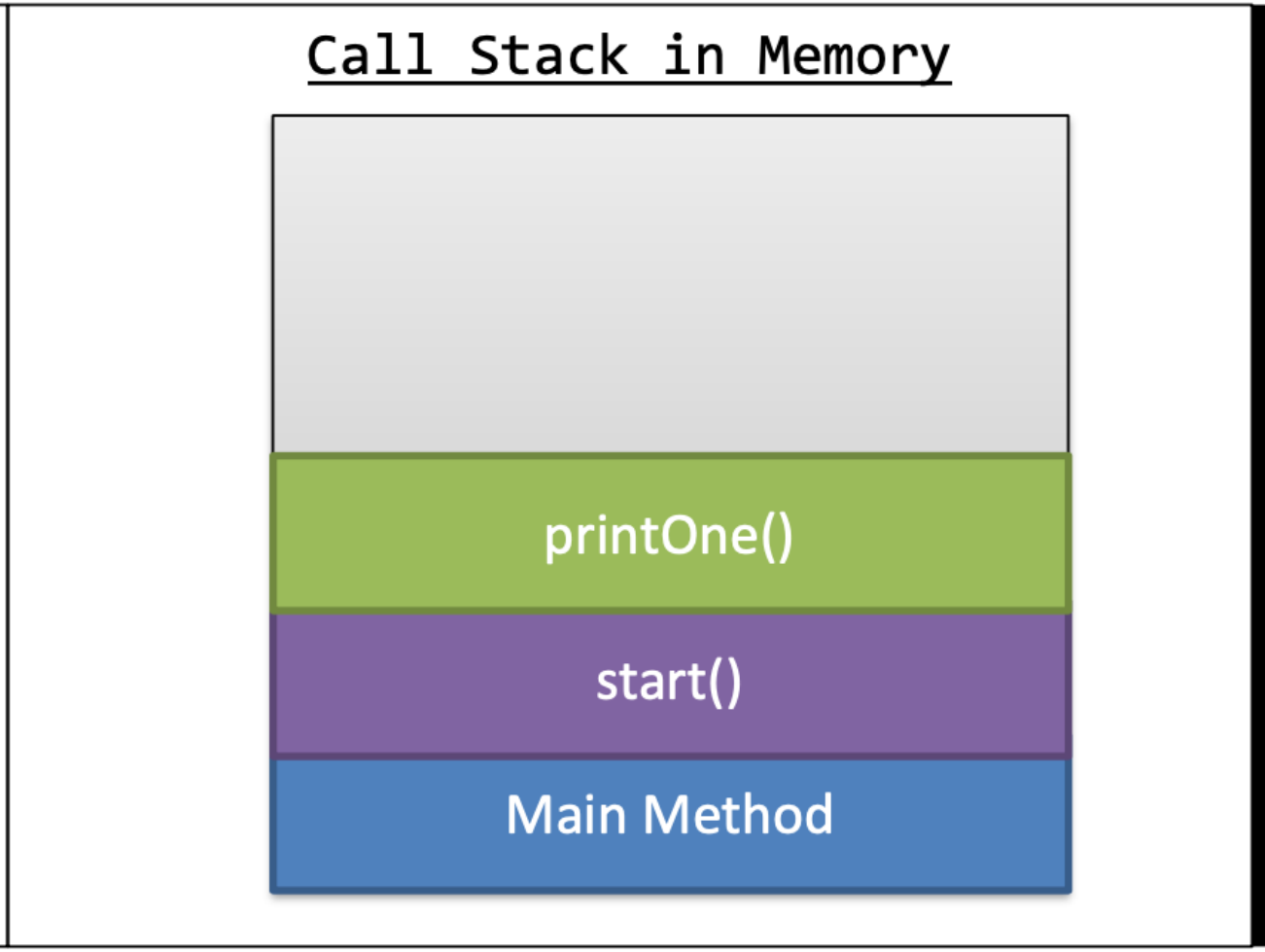
```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Console

Call Stack

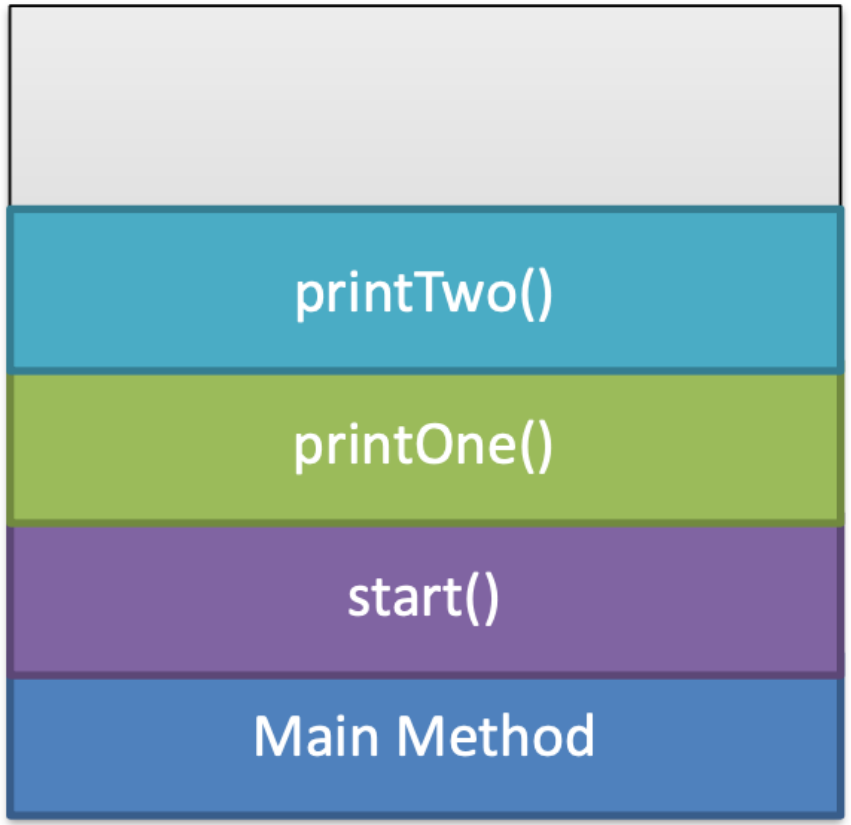
```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```



Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



Console

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory

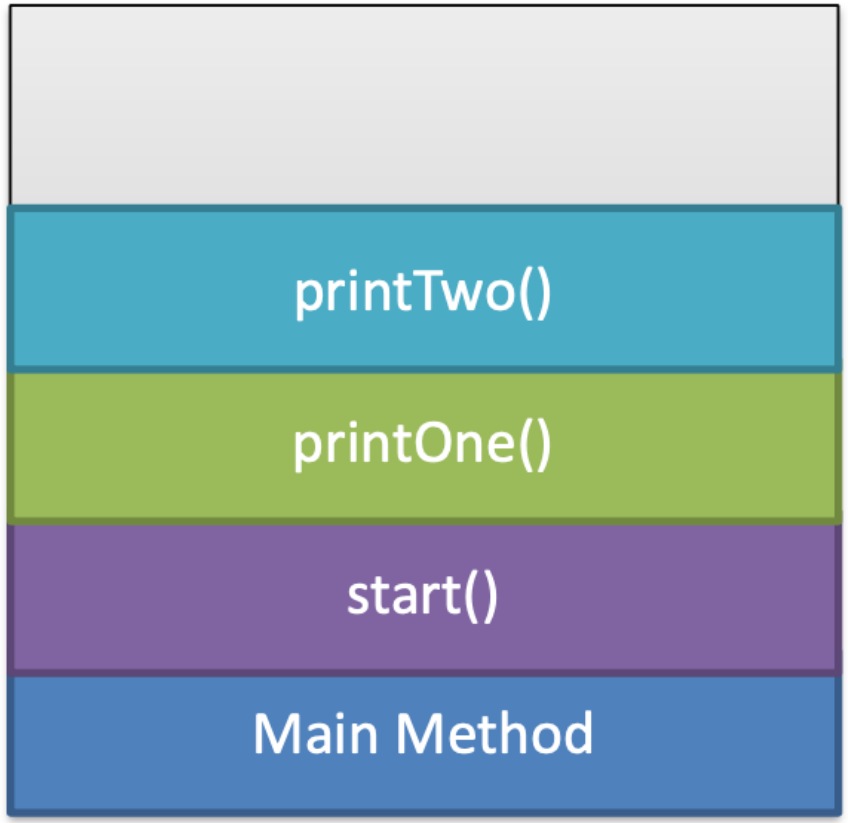
Console

Three

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



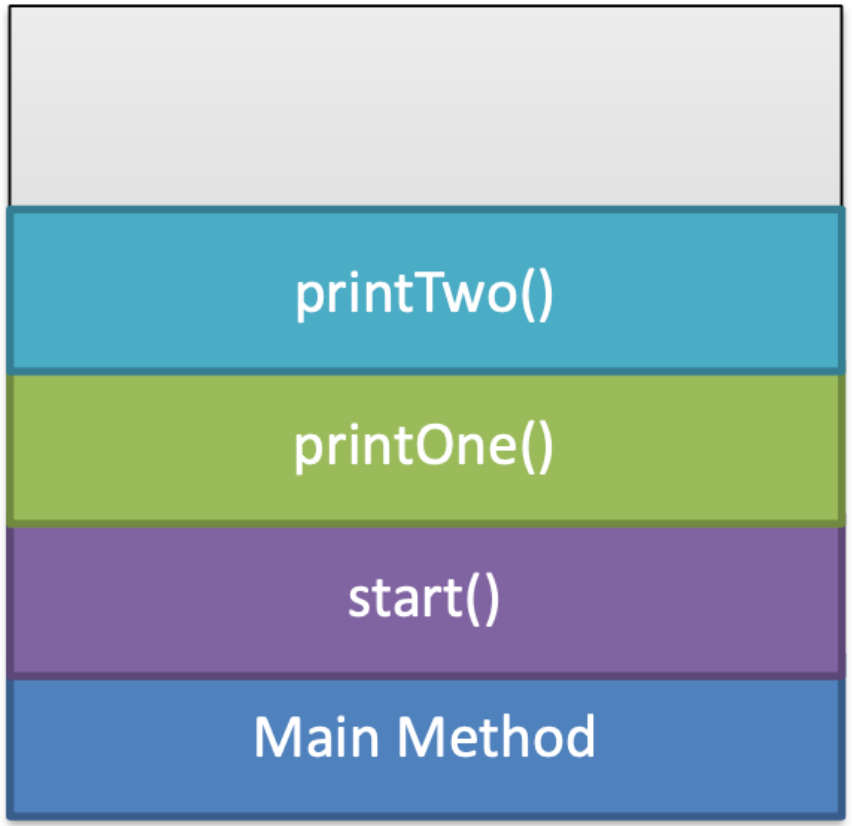
Console

Three

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



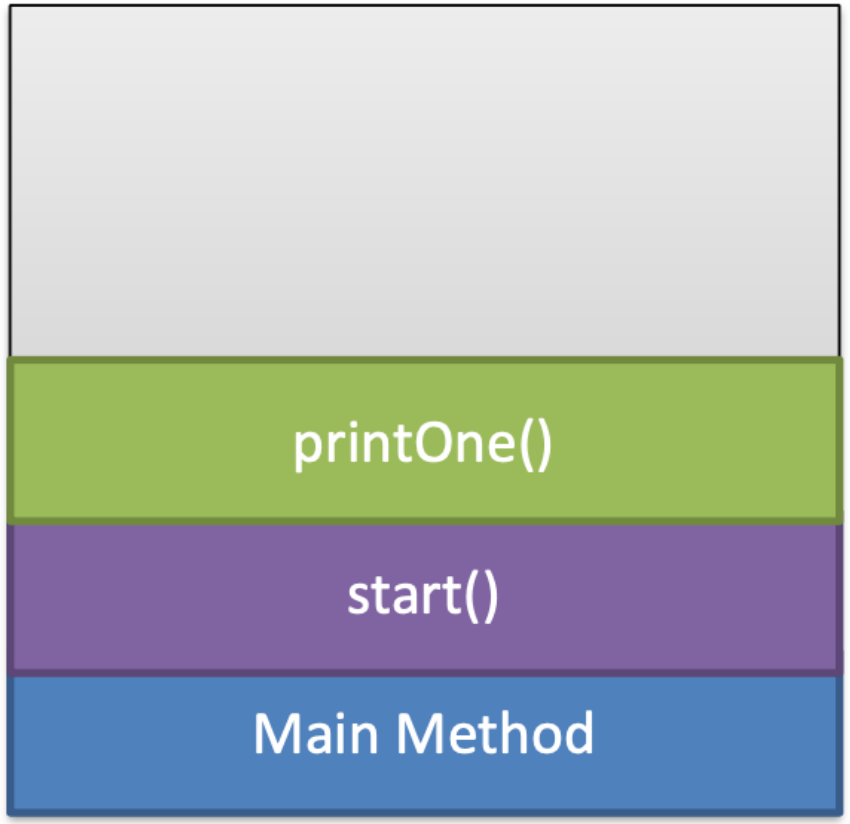
Console

Three
Two

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



Console

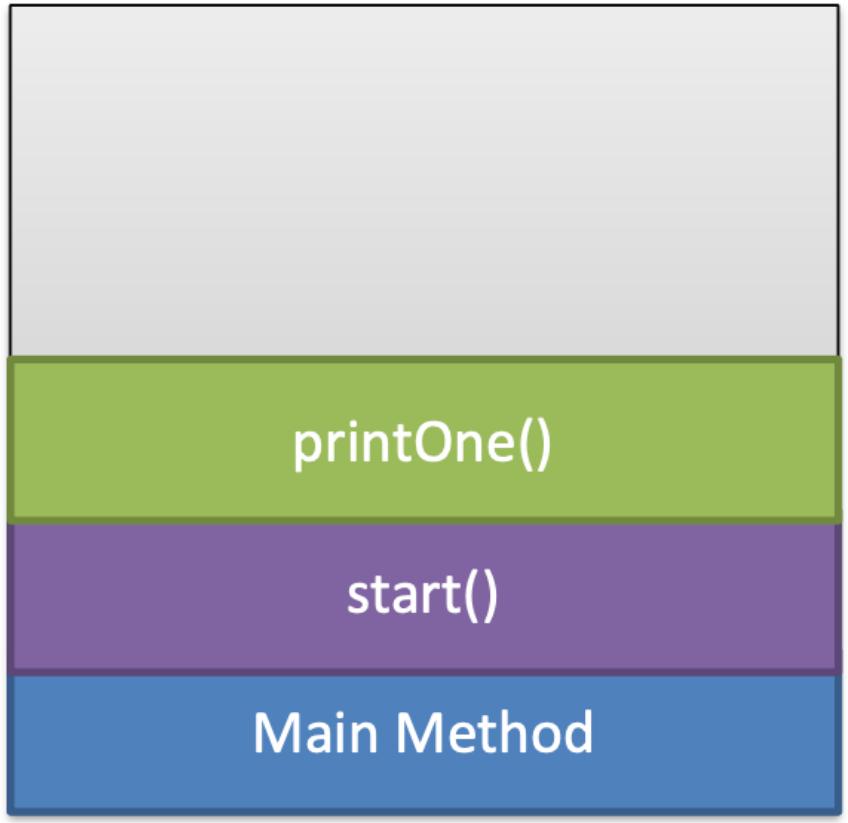
Three

Two

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



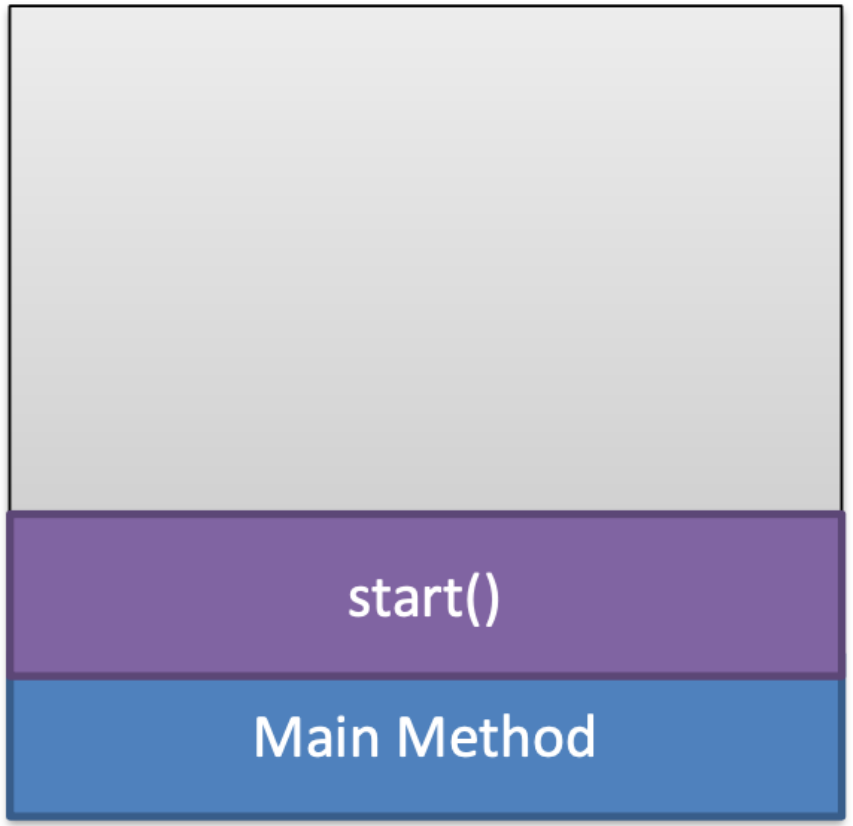
Console

Three
Two
One

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

Call Stack in Memory



Console

Three
Two
One

Call Stack

```
public class MethodTester {
        public static void main(String[] args)
        {
                MethodTester m = new MethodTester();
                m.start();
        }
        public void start()
        {
                printOne();
                System.out.println("Start");
        }
        public void printOne()
        {
                printTwo();
                System.out.println("One");
        }
        public void printTwo()
        {
                printThree();
                System.out.println("Two");
        }
        public void printThree()
        {
                System.out.println("Three");
        }
}
```

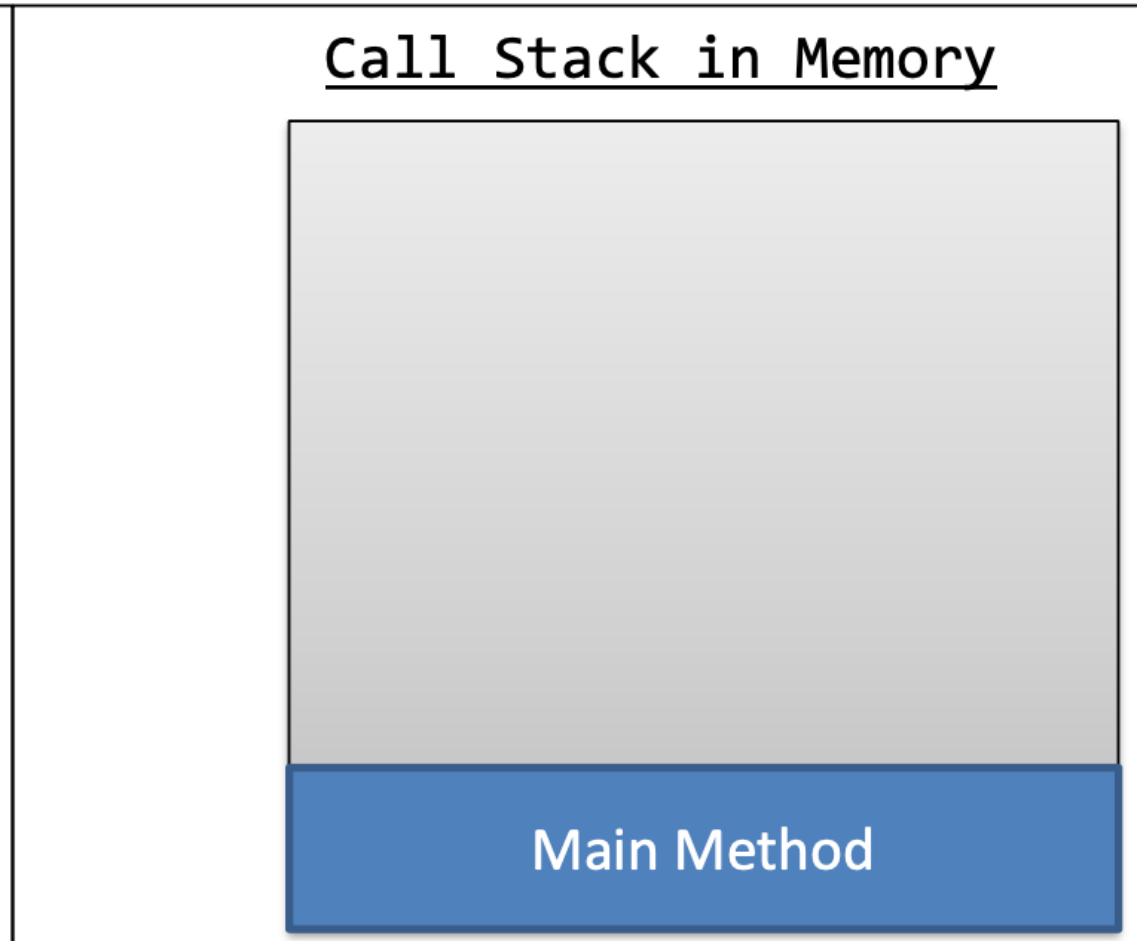
Call Stack in Memory

Console

Three
Two
One
Start

Call Stack

```
public class MethodTester {
	public static void main(String[] args)
	{
		MethodTester m = new MethodTester();
		m.start();
	}
	public void start()
	{
		printOne();
		System.out.println("Start");
	}
	public void printOne()
	{
		printTwo();
		System.out.println("One");
	}
	public void printTwo()
	{
		printThree();
		System.out.println("Two");
	}
	public void printThree()
	{
		System.out.println("Three");
	}
}
```



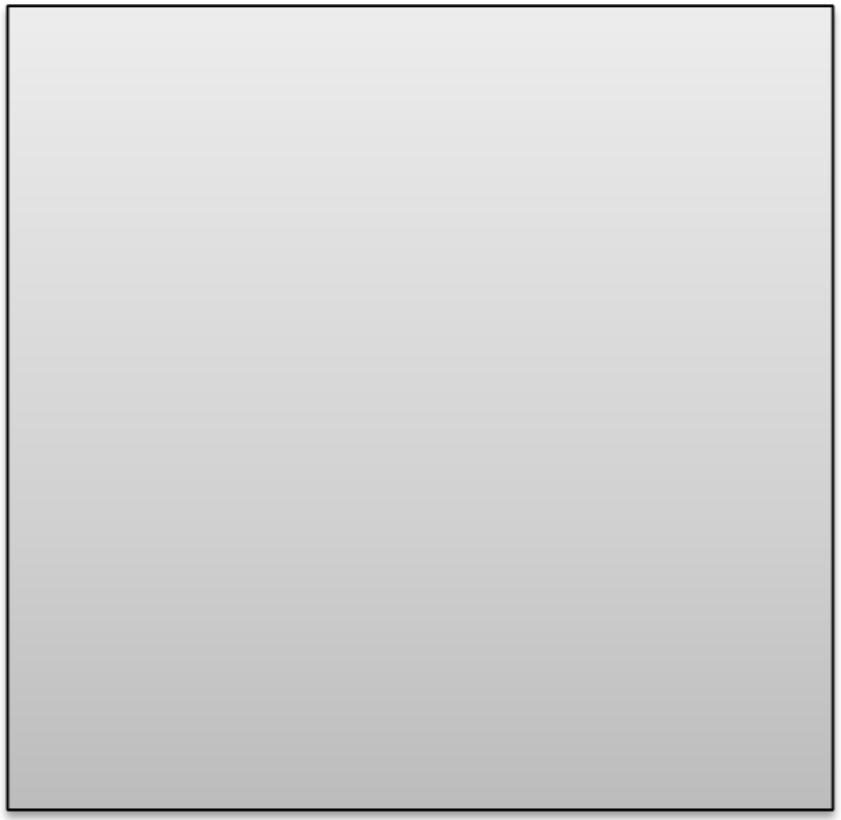
Console

Three
Two
One
Start

Call Stack

```
public class MethodTester {
	public static void main(String[] args)
	{
		MethodTester m = new MethodTester();
		m.start();
	}
	public void start()
	{
		printOne();
		System.out.println("Start");
	}
	public void printOne()
	{
		printTwo();
		System.out.println("One");
	}
	public void printTwo()
	{
		printThree();
		System.out.println("Two");
	}
	public void printThree()
	{
		System.out.println("Three");
	}
}
```

Call Stack in Memory



Console

Three
Two
One
Start

Quick Quiz

- What happens?

```
public class Example {

    public static void main(String[] args)
    {
        Example the = new Example();
        the.foo();
    }
    public void foo()
    {
        foo();
    }

}
```

Outline

- Overview
- Call stack
- Example
- Algorithms for problem solving

Example

```java
/*
 * Written by JJ Shepherd
 */
import java.util.Scanner;
public class MeasureConverter {

    public static final String IN = "INCHES";
    public static final String FT = "FEET";
    public static final String CM = "CENTIMETERS";

    public static void main(String[] args) {
        MeasureConverter m = new MeasureConverter();
        m.start();
    }
    public void start()
    {
        Scanner keyboard = new Scanner(System.in);
        printGreetings();
        boolean quit = false;
        while(!quit)
        {
            printOptions();
            String unit1 = keyboard.nextLine();
            String unit2 = keyboard.nextLine();
            if(!isValidUnit(unit1) || !isValidUnit(unit2))
            {
                System.out.println("One of the units were invalid. Try again");
                continue;
            }
            printInput(unit1,unit2);
            double value = keyboard.nextDouble();
            keyboard.nextLine();
            double result = 0.0;
            if(unit1.equalsIgnoreCase(IN) && unit2.equalsIgnoreCase(FT))
            {
                result = inToFt(value);
            }
            else if(unit1.equalsIgnoreCase(IN) && unit2.equalsIgnoreCase(CM))
            {
                result = inToCm(value);
            }
            else if(unit1.equalsIgnoreCase(CM) && unit2.equalsIgnoreCase(IN))
            {
                result = cmToIn(value);
            }
            else if(unit1.equalsIgnoreCase(CM) && unit2.equalsIgnoreCase(FT))
            {
                result = cmToFt(value);
            }
            else if(unit1.equalsIgnoreCase(FT) && unit2.equalsIgnoreCase(IN))
            {
                result = ftToIn(value);
            }
            else if(unit1.equalsIgnoreCase(FT) && unit2.equalsIgnoreCase(CM))
            {
                result = ftToCm(value);
            }
```

Example

```
            else
            {
                result = value;
            }
            printResults(unit1,unit2,result);
            System.out.println("Press Enter to keep converting units or enter \"quit\" to
quit");
            quit = keyboard.nextLine().equalsIgnoreCase("quit");
        }
        System.out.println("Goodbye!");
    }
    public void printGreetings()
    {
        System.out.println("Welcome to the units converter!");
    }
    public void printOptions()
    {
        System.out.println("Enter the type of units followed by the second type.\nUnits can be
either \""+IN+"\", \""+FT+"\", or \""+CM+"\"");
    }
    public void printResults(String u1, String u2, double result)
    {
        System.out.println("There are "+result+" "+u2+" in "+u1);
    }
    public boolean isValidUnit(String input)
    {
        return input.equalsIgnoreCase(IN) || input.equalsIgnoreCase(CM) ||
input.equalsIgnoreCase(FT);
    }
    public void printInput(String u1, String u2)
    {
        System.out.println("Enter "+u1+" and I'll determine the number of "+u2);
    }
    public double inToFt(double in)
    {
        return in / 12.0;
    }
    public double inToCm(double in)
    {
        return in * 2.54;
    }
    public double cmToIn(double cm)
    {
        return cm * 0.393701;
    }
    public double cmToFt(double cm)
    {
        return cm * 0.0328084;
    }
    public double ftToIn(double ft)
    {
        return ft * 12.0;
    }
    public double ftToCm(double ft)
    {
        return ft * 30.48;
    }

}
```

Outline

- Overview
- Call stack
- Example
- Algorithms for problem solving

Algorithms for Problem Solving

- Donald Knuth conjectured that, starting with the number 4, a sequence of square root, floor, and factorial operations can create any desired positive integer
- How can we reach 5 from 4 using only these operations?

Algorithms for Problem Solving

- Donald Knuth conjectured that, starting with the number 4, a sequence of square root, floor, and factorial operations can create any desired positive integer
- How can we reach 5 from 4 using only these operations?

- Can we create an algorithm that will still work if we changed the operations?
  - Logarithms
  - Exponents
- Can we create an algorithm that will still work if we changed the space of possible values as well as the operations?
  - Imaginary numbers
  - Vectors
  - Other objects

What Methods Should We Implement?