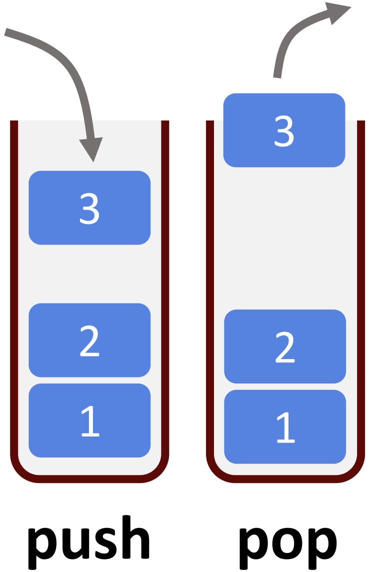
JC2002 Java Programming

Lecture 16: Call stack and recursion

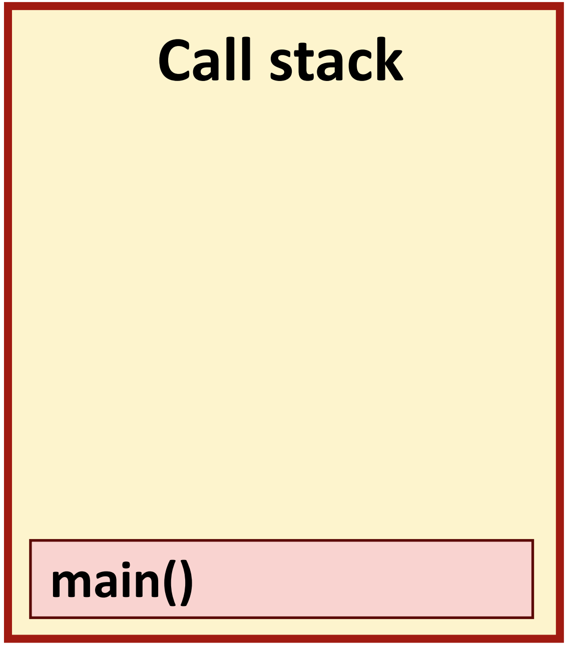
Call stack

- A ***stack*** is a data structure where data elements are piled on top of each other (like e.g., pile of cards).
  - The basic operations are *push*, which adds an element on the top, and *pop*, which removes the most recently added element from the top.
- In JVM, call stack is used to keep track of method calls in memory
  - When a method is called, the local variables and other data of the method are pushed in the call stack.
  - When the program returns from a method, the information is removed from the stack.

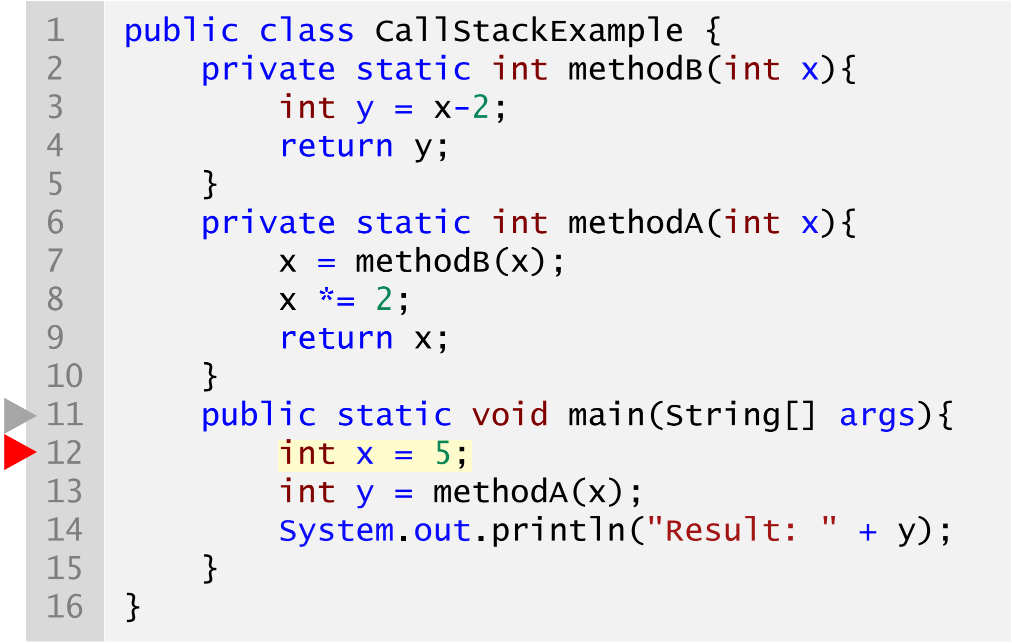


Understanding call stack (1)

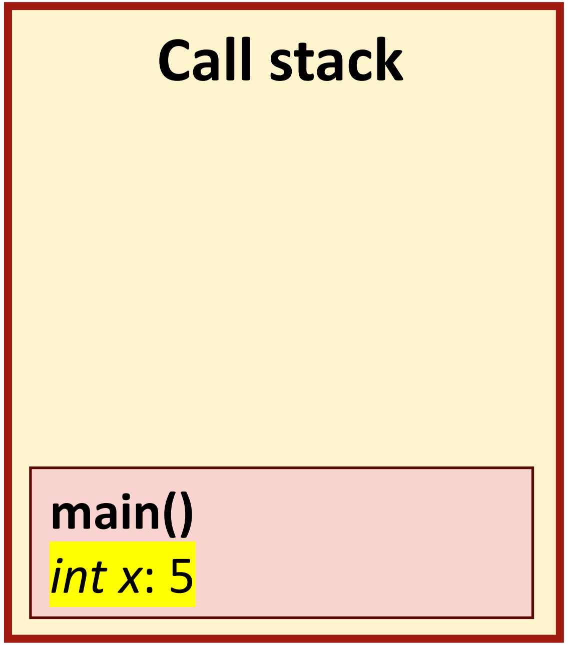
```java
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```



Understanding call stack (2)



```java
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```

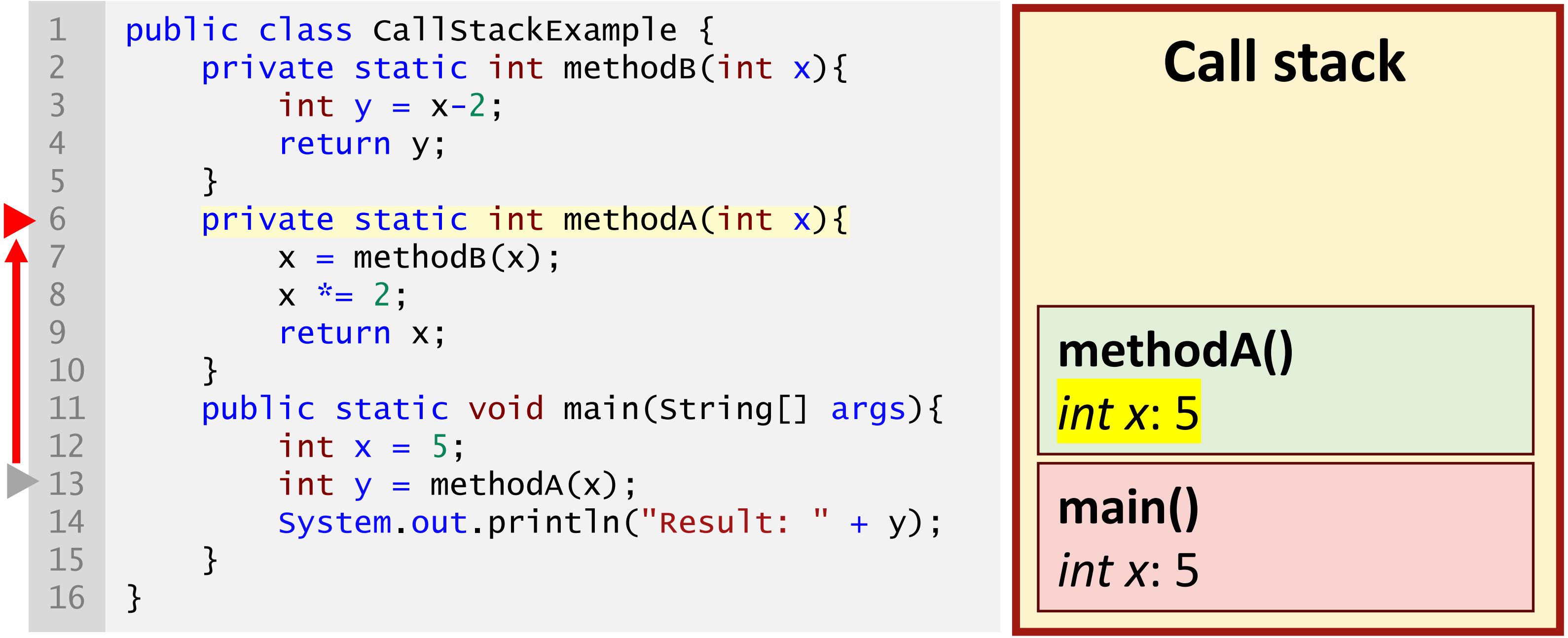


Call stack

main()

int x: 5

Understanding call stack (3)



```java
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```

Call stack

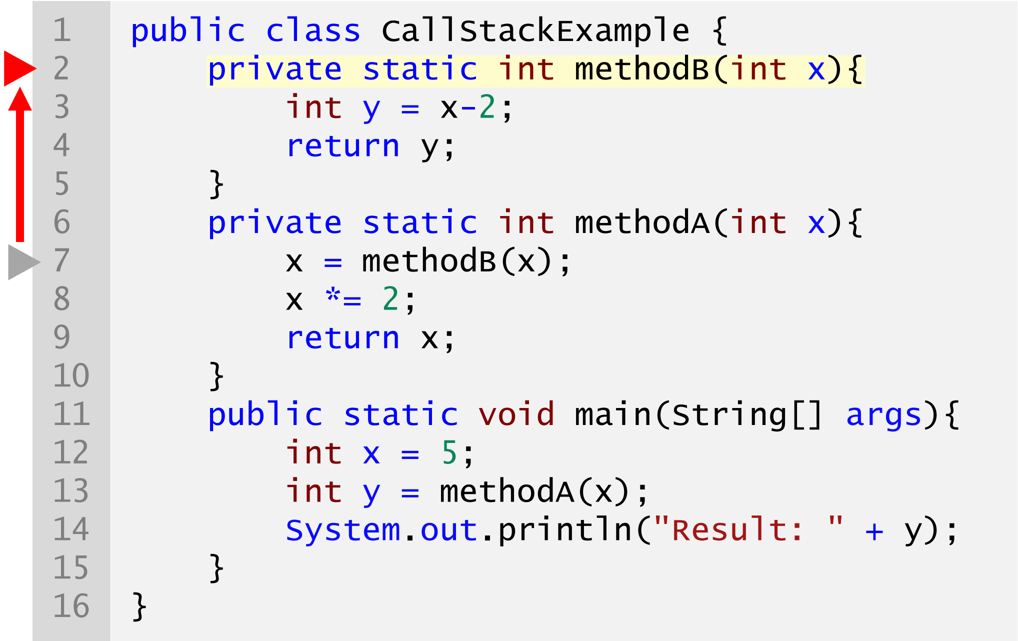
methodA()

int x: 5

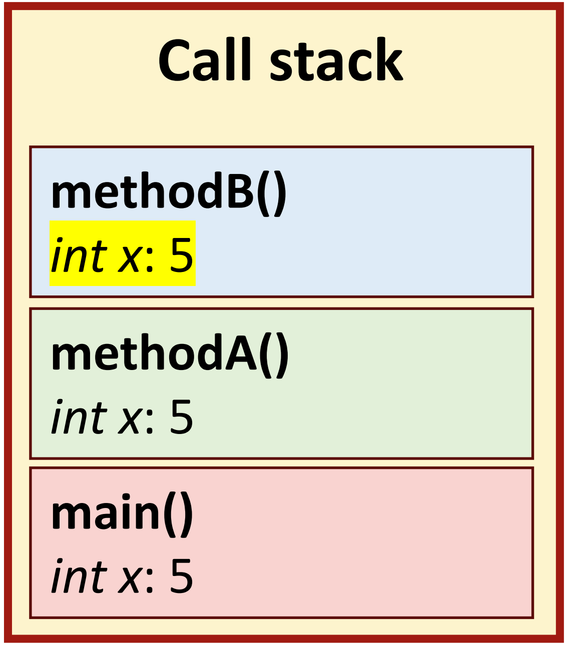
main()

int x: 5

Understanding call stack (4)



```
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```



Call stack

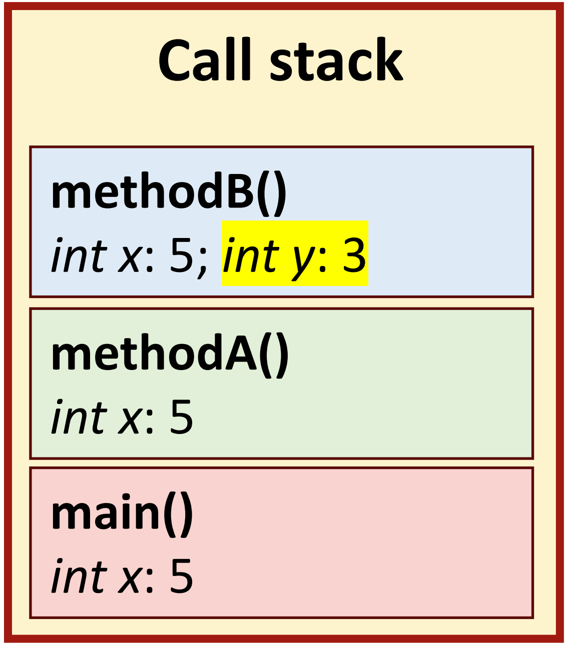
methodB()
int x: 5

methodA()
int x: 5

main()
int x: 5

Understanding call stack (5)

```java
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```



Call stack

methodB()

int x: 5; *int y*: 3

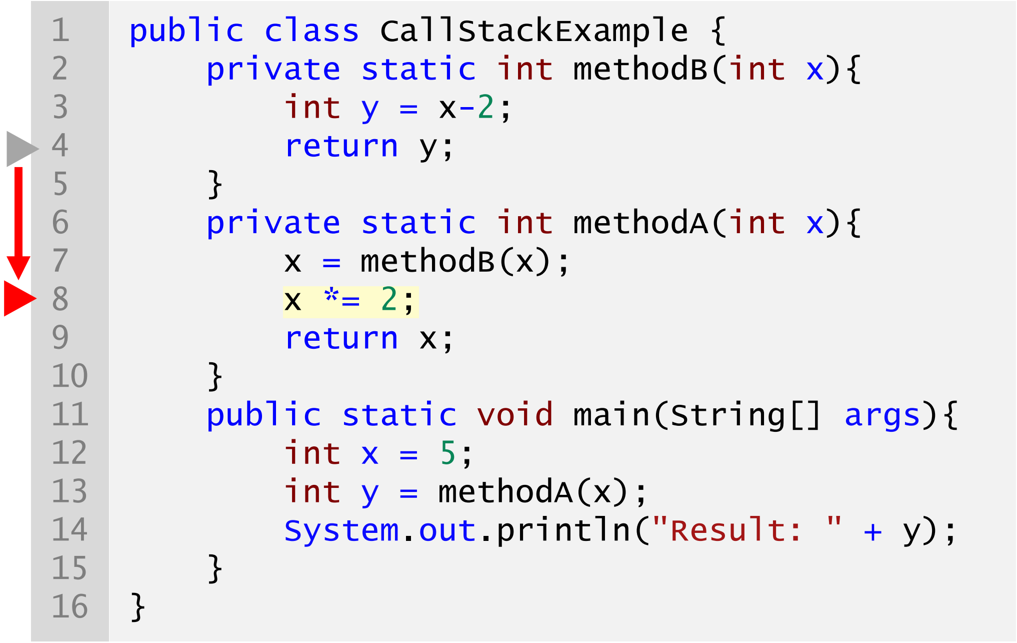
methodA()

int x: 5

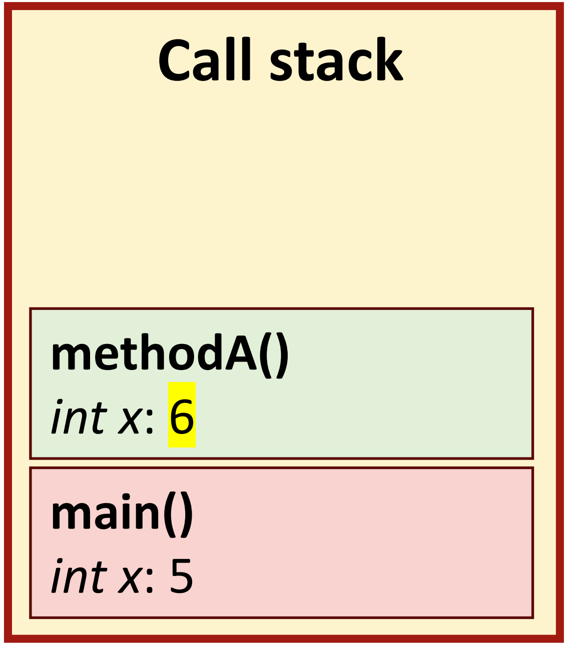
main()

int x: 5

Understanding call stack (6)



```java
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```



Call stack

methodA()

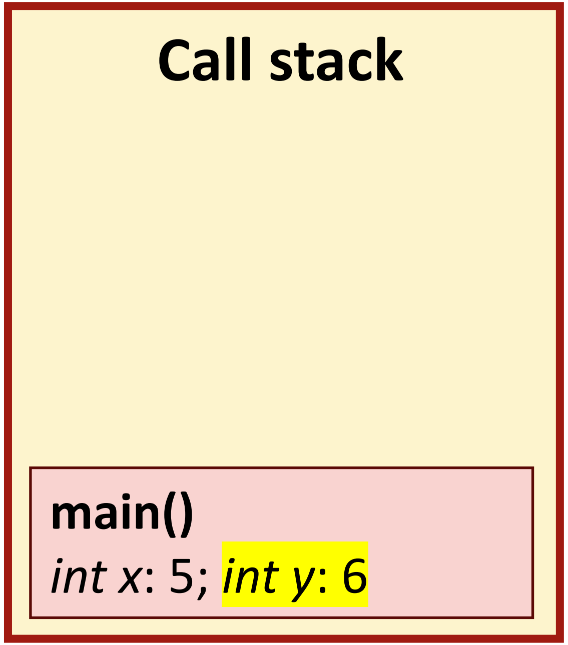
int x: 6

main()

int x: 5

Understanding call stack (7)

```java
public class CallStackExample {
    private static int methodB(int x){
        int y = x-2;
        return y;
    }
    private static int methodA(int x){
        x = methodB(x);
        x *= 2;
        return x;
    }
    public static void main(String[] args){
        int x = 5;
        int y = methodA(x);
        System.out.println("Result: " + y);
    }
}
```



Call stack

main()

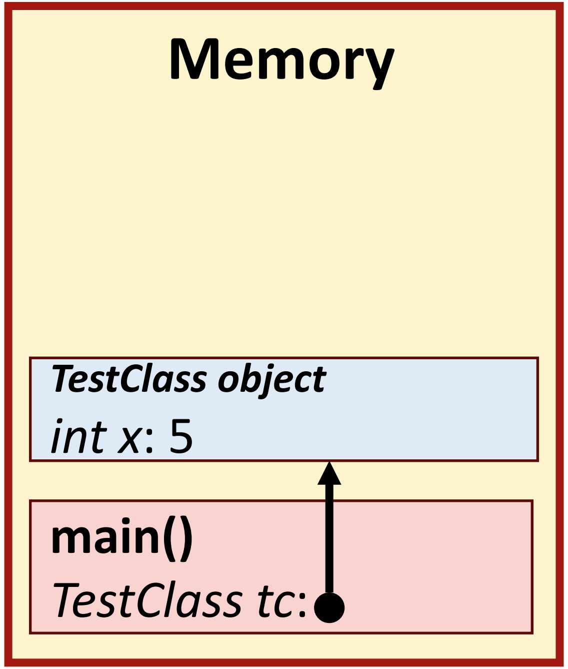
int x: 5; *int y*: 6

Using shadowed class attributes

- Method parameters are visible only within the method scope
- If a class attribute and a method parameter share the same name, Java will use the method parameter
- If you need to access the class attribute, use keyword **`this`**

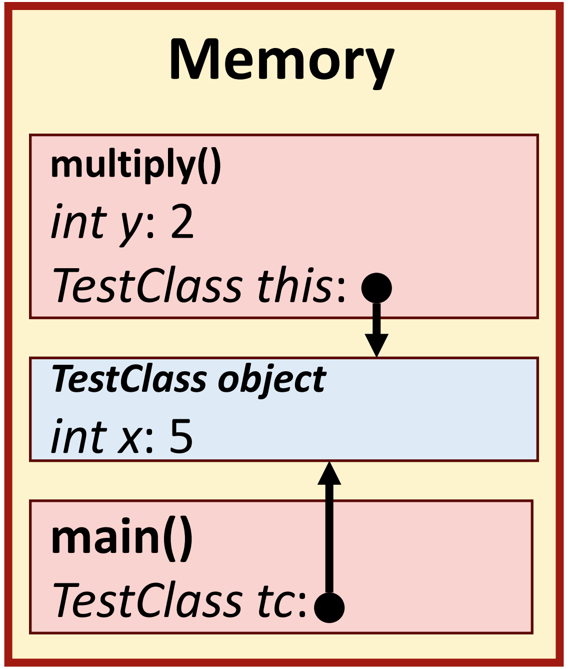
Using keyword this: instantiate class

```java
public class TestClass {
    private int x = 5;
    public TestClass() {}
    private int multiply(int y) {
        y = y * x;
        return y;
    }
    private void add(int x) {
        this.x += x;
        return;
    }
    public static void main(String[] args) {
        TestClass tc = new TestClass();
        int x = tc.multiply(2);
        tc.add(x);
        System.out.println("Result: " + tc.x);
    }
}
```



Using keyword this: instantiate class

```java
public class TestClass {
    private int x = 5;
    public TestClass() {}
    private int multiply(int y) {
        y = y * x;
        return y;
    }
    private void add(int x) {
        this.x += x;
        return;
    }
    public static void main(String[] args) {
        TestClass tc = new TestClass();
        int x = tc.multiply(2);
        tc.add(x);
        System.out.println("Result: " + tc.x);
    }
}
```



Memory

multiply()

int y: 2

TestClass this:

TestClass object

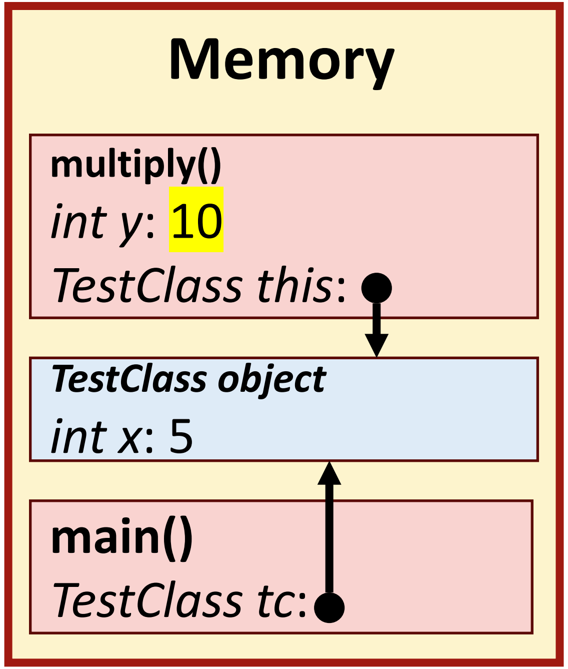
int x: 5

main()

TestClass tc:

Using keyword this: instantiate class

```
public class TestClass {
    private int x = 5;
    public TestClass() {}
    private int multiply(int y) {
        y = y * x;
        return y;
    }
    private void add(int x) {
        this.x += x;
        return;
    }
    public static void main(String[] args) {
        TestClass tc = new TestClass();
        int x = tc.multiply(2);
        tc.add(x);
        System.out.println("Result: " + tc.x);
    }
}
```



Memory

multiply()

int y: 10

TestClass this:

TestClass object

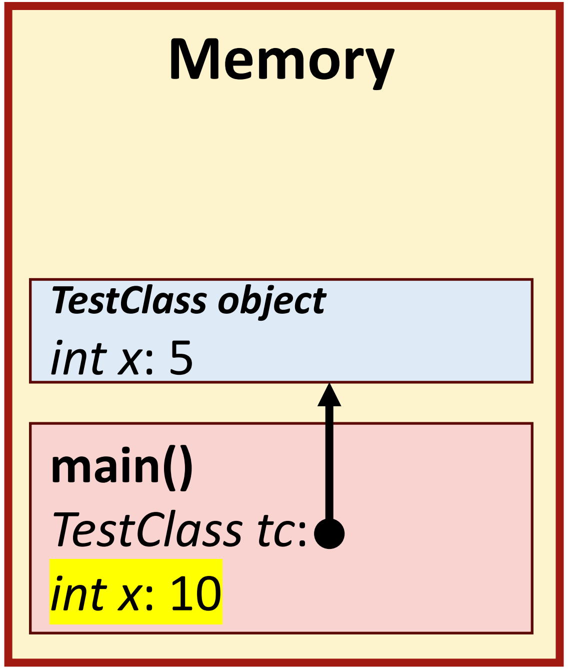
int x: 5

main()

TestClass tc:

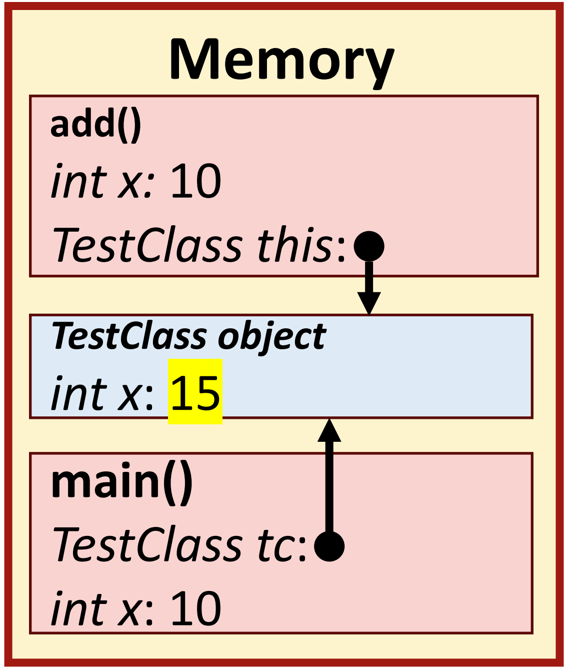
Using keyword this: instantiate class

```java
public class TestClass {
    private int x = 5;
    public TestClass() {}
    private int multiply(int y) {
        y = y * x;
        return y;
    }
    private void add(int x) {
        this.x += x;
        return;
    }
    public static void main(String[] args) {
        TestClass tc = new TestClass();
        int x = tc.multiply(2);
        tc.add(x);
        System.out.println("Result: " + tc.x);
    }
}
```



Using keyword this: instantiate class

```java
public class TestClass {
    private int x = 5;
    public TestClass() {}
    private int multiply(int y) {
        y = y * x;
        return y;
    }
    private void add(int x) {
        this.x += x;
        return;
    }
    public static void main(String[] args) {
        TestClass tc = new TestClass();
        int x = tc.multiply(2);
        tc.add(x);
        System.out.println("Result: " + tc.x);
    }
}
```



Memory

add()

int x: 10

TestClass this:

TestClass object

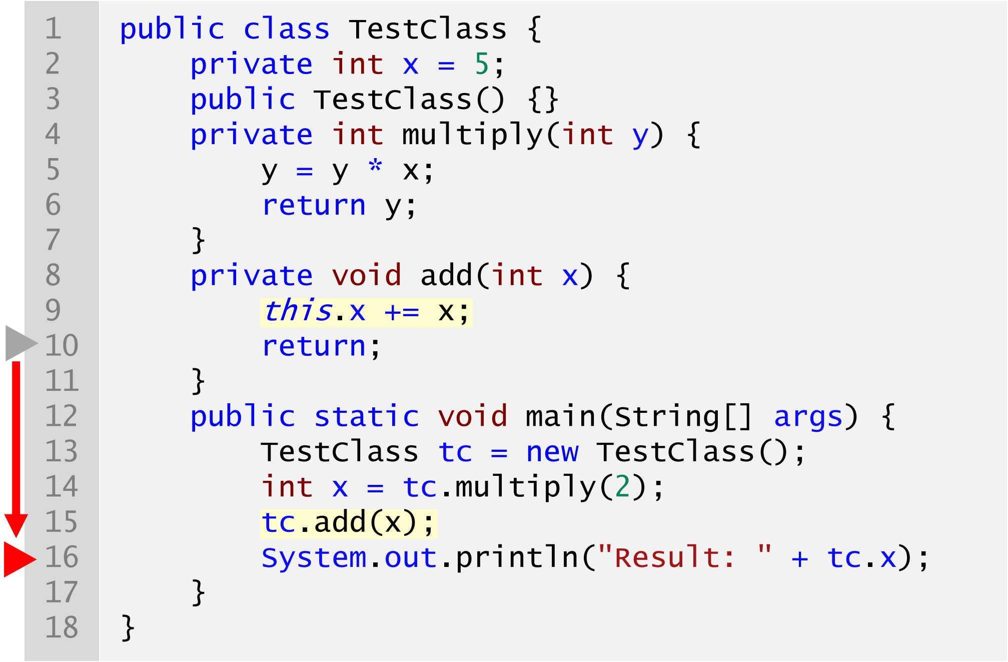
int x: 15

main()

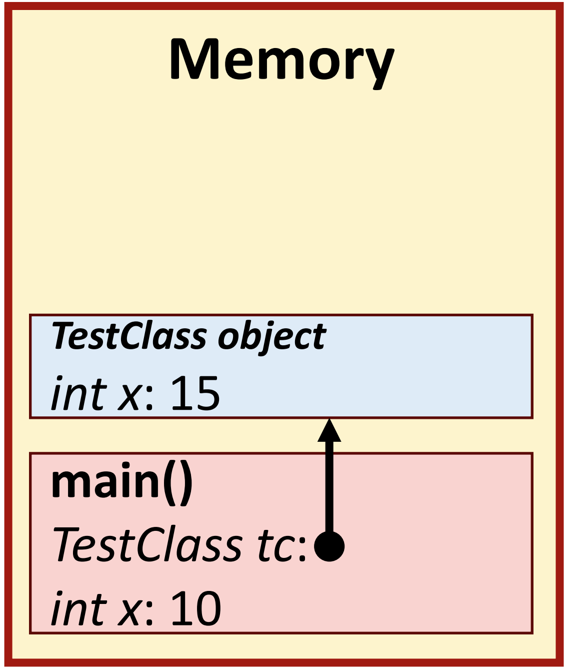
TestClass tc:

int x: 10

Using keyword this: instantiate class



```java
public class TestClass {
    private int x = 5;
    public TestClass() {}
    private int multiply(int y) {
        y = y * x;
        return y;
    }
    private void add(int x) {
        this.x += x;
        return;
    }
    public static void main(String[] args) {
        TestClass tc = new TestClass();
        int x = tc.multiply(2);
        tc.add(x);
        System.out.println("Result: " + tc.x);
    }
}
```



Memory

TestClass object

int x: 15

main()

TestClass tc:

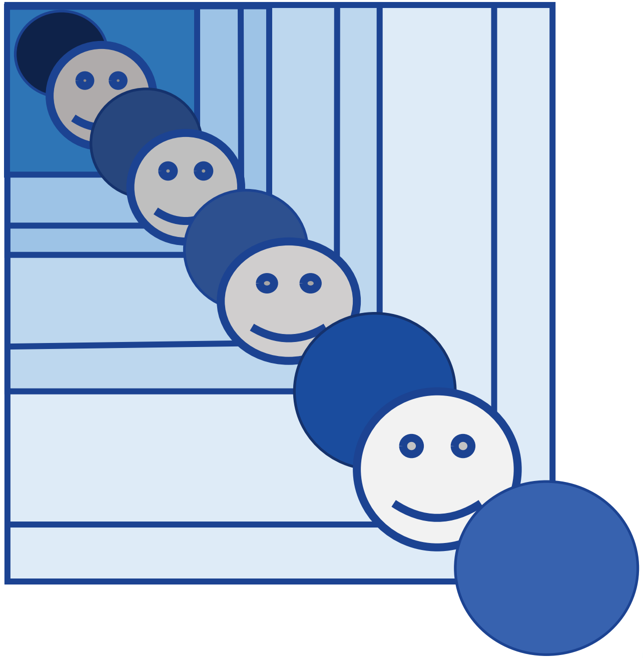
int x: 10

Recursion

- A recursive method returns a call to itself until a base case is reached

```
methodX(param) {
    if(..) {
        methodX(newParam);
    }
    ..
}
```

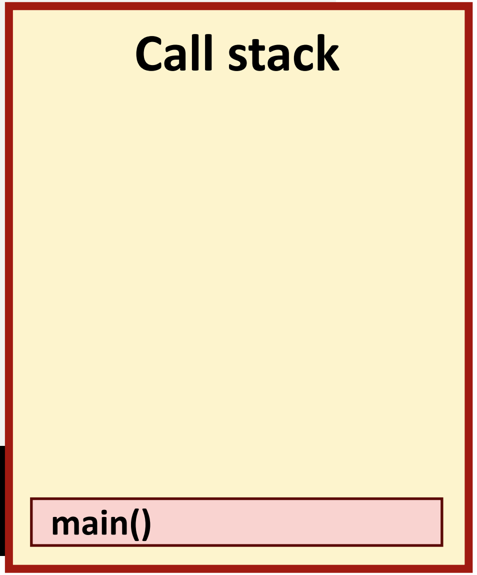
- Method attributes stored in call stack: deep recursion can lead to excessive memory consumption



Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

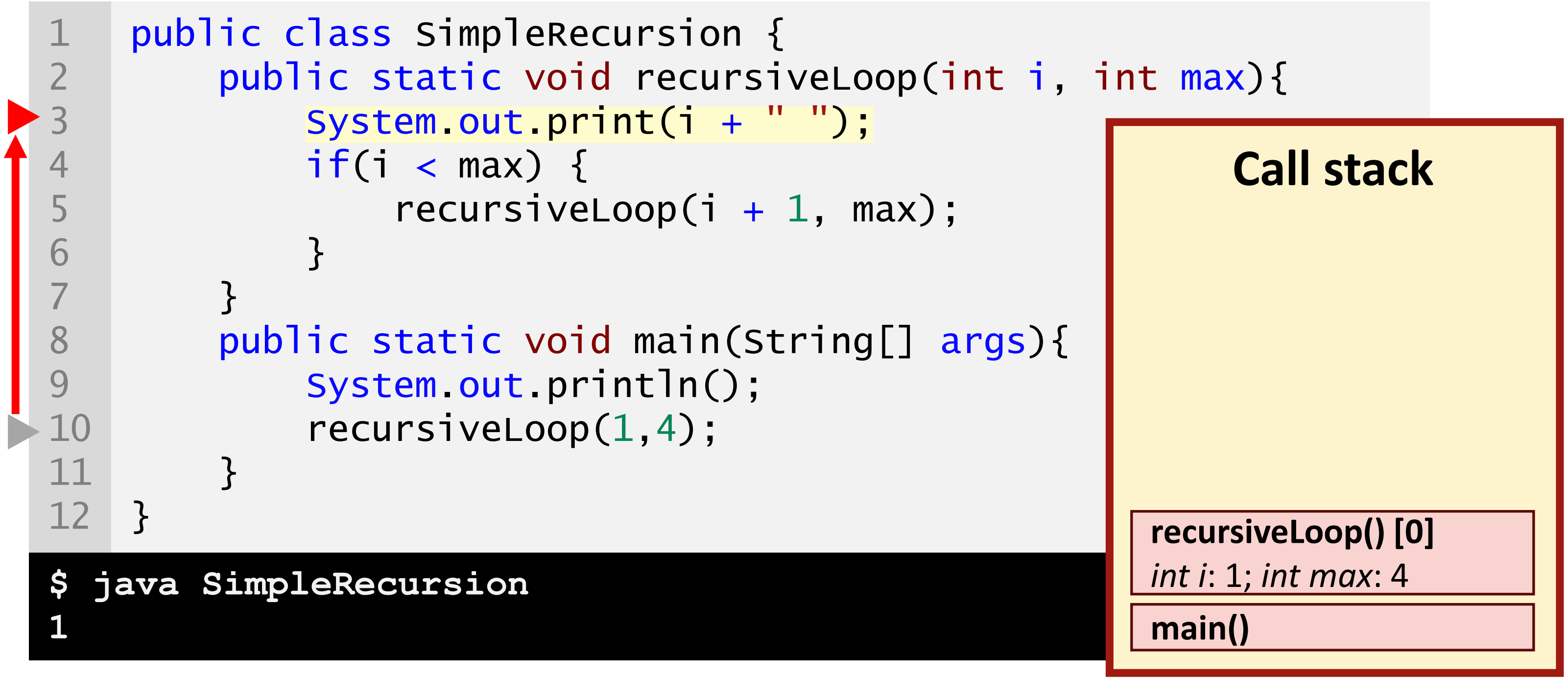
```
$ java SimpleRecursion
```



Call stack

main()

Simple recursion example



```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1
```

Call stack

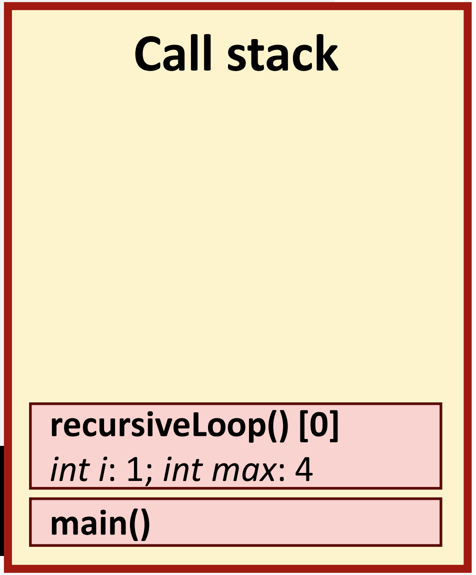
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1
```



Call stack

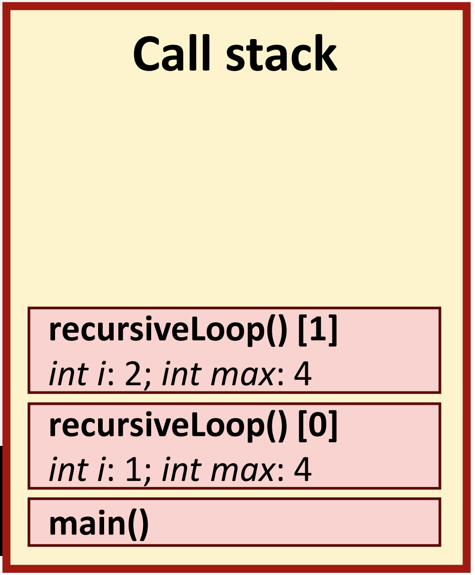
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2
```



Call stack

recursiveLoop() [1]
int i: 2; *int max*: 4

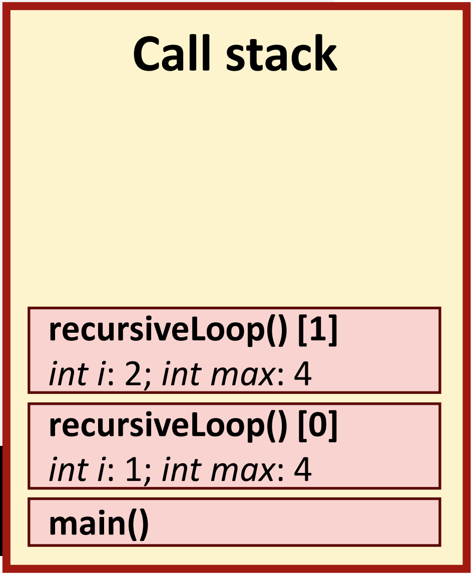
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2
```



Call stack

recursiveLoop() [1]
int i: 2; *int max*: 4

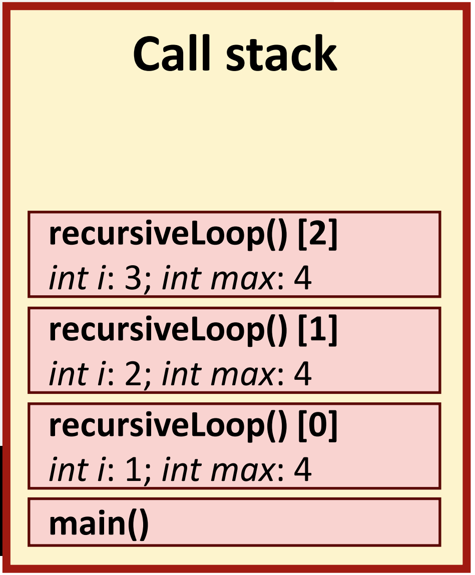
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3
```



Call stack

recursiveLoop() [2]
int i: 3; *int max*: 4

recursiveLoop() [1]
int i: 2; *int max*: 4

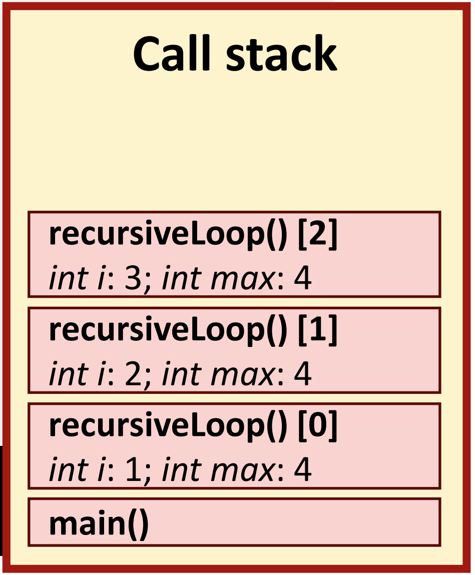
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3
```



Call stack

recursiveLoop() [2]
int i: 3; *int max*: 4

recursiveLoop() [1]
int i: 2; *int max*: 4

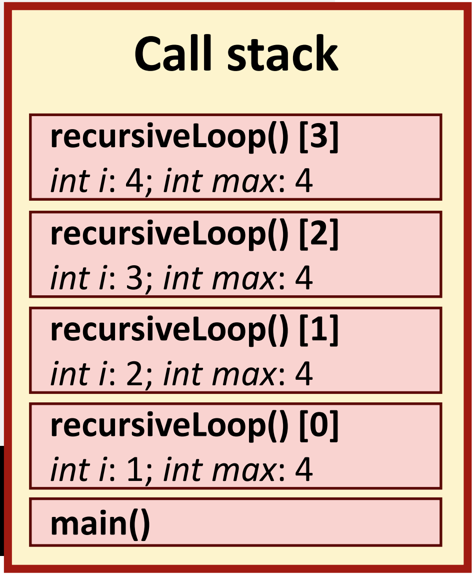
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3 4
```



Call stack

recursiveLoop() [3]
int i: 4; *int max*: 4

recursiveLoop() [2]
int i: 3; *int max*: 4

recursiveLoop() [1]
int i: 2; *int max*: 4

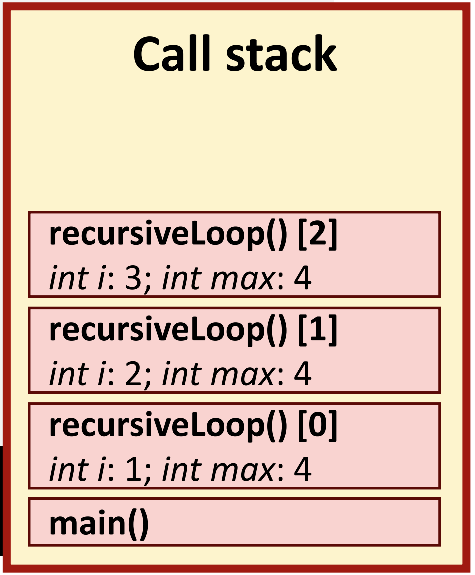
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3 4
```



Call stack

recursiveLoop() [2]
int i: 3; *int max*: 4

recursiveLoop() [1]
int i: 2; *int max*: 4

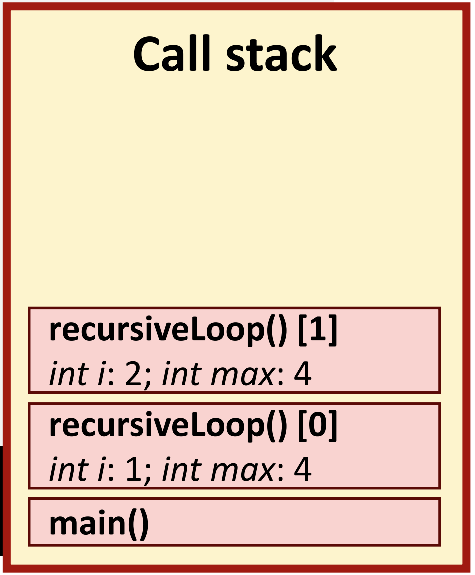
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3 4
```



Call stack

recursiveLoop() [1]
int i: 2; *int max*: 4

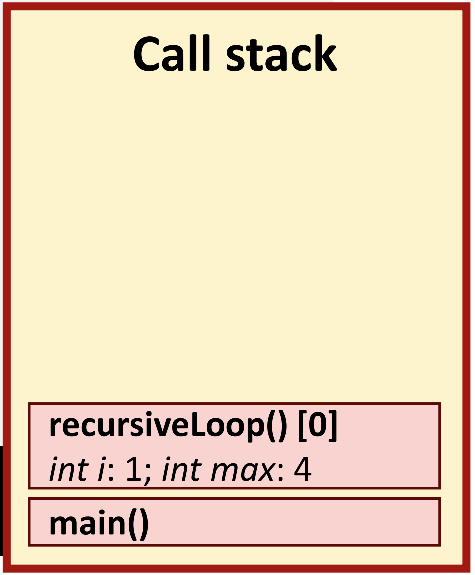
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3 4
```



Call stack

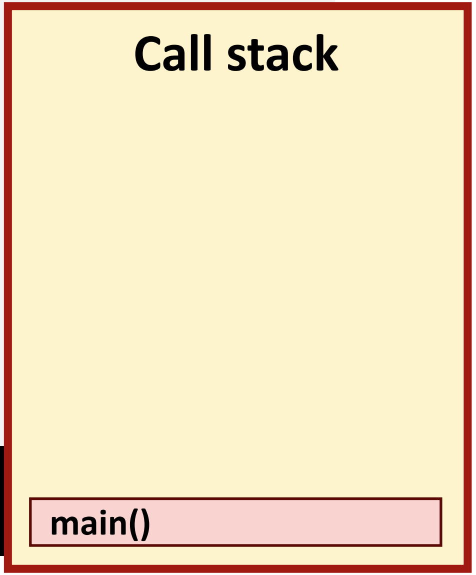
recursiveLoop() [0]
int i: 1; *int max*: 4

main()

Simple recursion example

```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

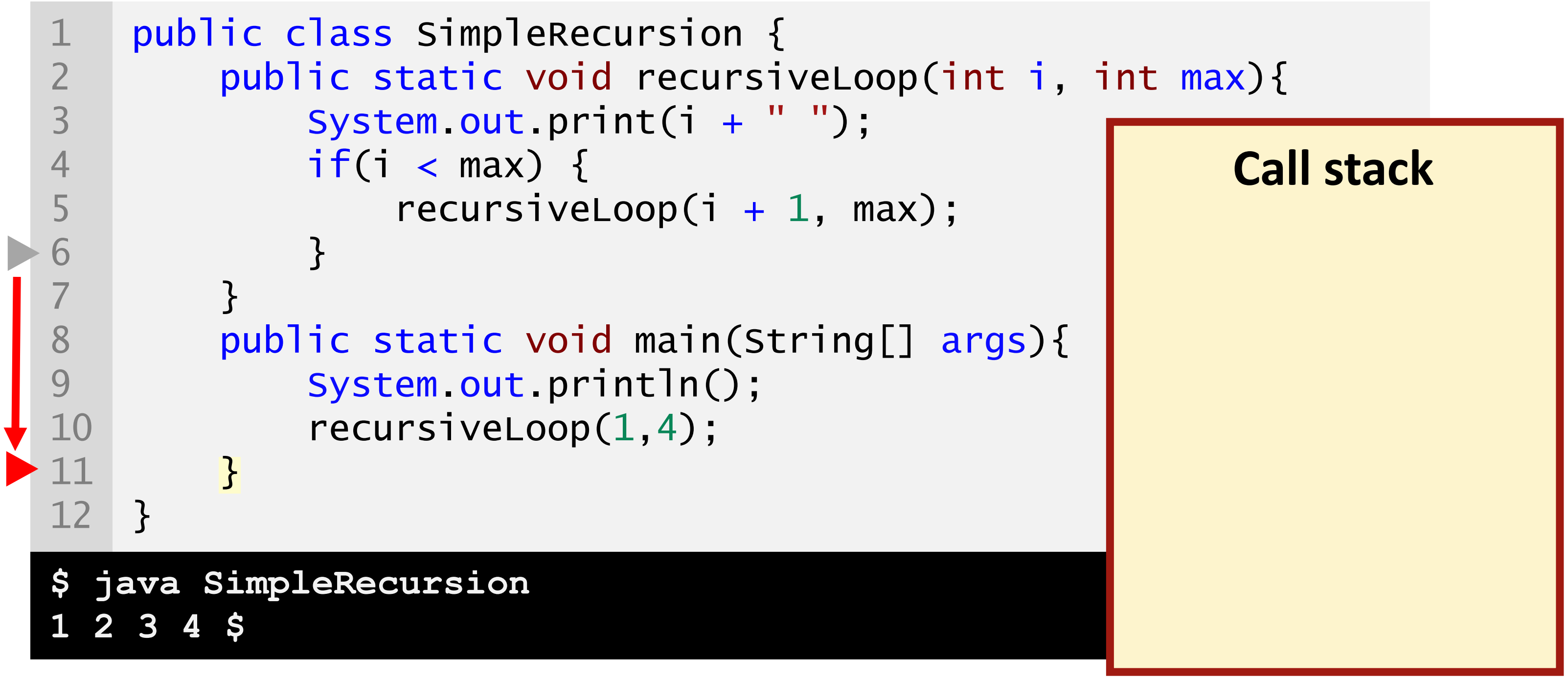
```
$ java SimpleRecursion
1 2 3 4
```



Call stack

main()

Simple recursion example



```java
public class SimpleRecursion {
    public static void recursiveLoop(int i, int max){
        System.out.print(i + " ");
        if(i < max) {
            recursiveLoop(i + 1, max);
        }
    }
    public static void main(String[] args){
        System.out.println();
        recursiveLoop(1,4);
    }
}
```

```
$ java SimpleRecursion
1 2 3 4 $
```

Call stack

Caveats of recursion

- Mistakes like omitting the base case or writing the recursion step incorrectly can cause *infinite recursion*
- Recursive programs may result in exponential method calls
- Each recursive method can be re-written using loops
- Use recursive methods only if the problem is naturally recursive (i.e., to improve understanding) or there are performance benefits, or you need to impress in a job interview ☺

Fibonacci number example

- Fibonacci sequence: Each number is a sum of two preceding numbers, starting from 0 and 1
- Generating a Fibonacci sequence is a naturally recursive problem

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	…
0	1	1	2	3	5	8	13	21	34	55	…

Fibonacci by recursion example

```java
public class FibonacciRecursion {
    public static int fibonacci(int f1,int f2,int cnt) {
        if(cnt == 2) {
            return f1+f2;
        } else {
            return fibonacci(f2, f1+f2, cnt-1);
        }
    }
    public static void main(String[] args) {
        System.out.println("Result: " + fibonacci(0,1,19));
    }
}
```

```
$ java FibonacciRecursion
Result: 4181
```

Questions, comments?