CS 162FZ: Introduction to Computer Science II

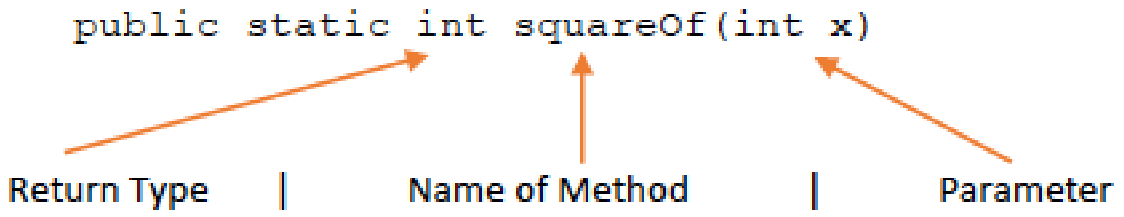
Lecture 06

Recursion I

Dr. Chun-Yang Zhang

Introduction

- We are familiar with creating and calling methods from other methods from Java
- Therefor we can come to the conlusion that a method can call *itself*
- Java and all programming languages can support this posibilty which is know as *recursion.*
- Lets revise methods:



```
public static int squareOf(int x)
```

Return Type | Name of Method | Parameter

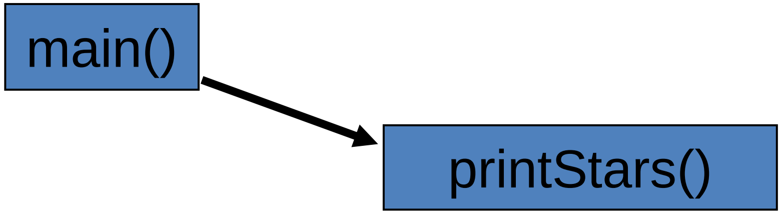
Example of a static Method calling another:

```
public class StaticMethodExample
{
        public static void main(String args[])
        {
        printStars(10);
        }

        public static void printStars(int n)
        {
        for(int i=0;i<n;i++)
        {
        System.out.print("*");
        }
        System.out.println("");
        }
}
```

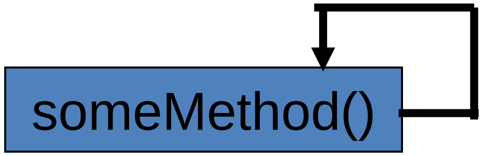
Introduction to Recursion

So far, we have seen methods that call other functions.

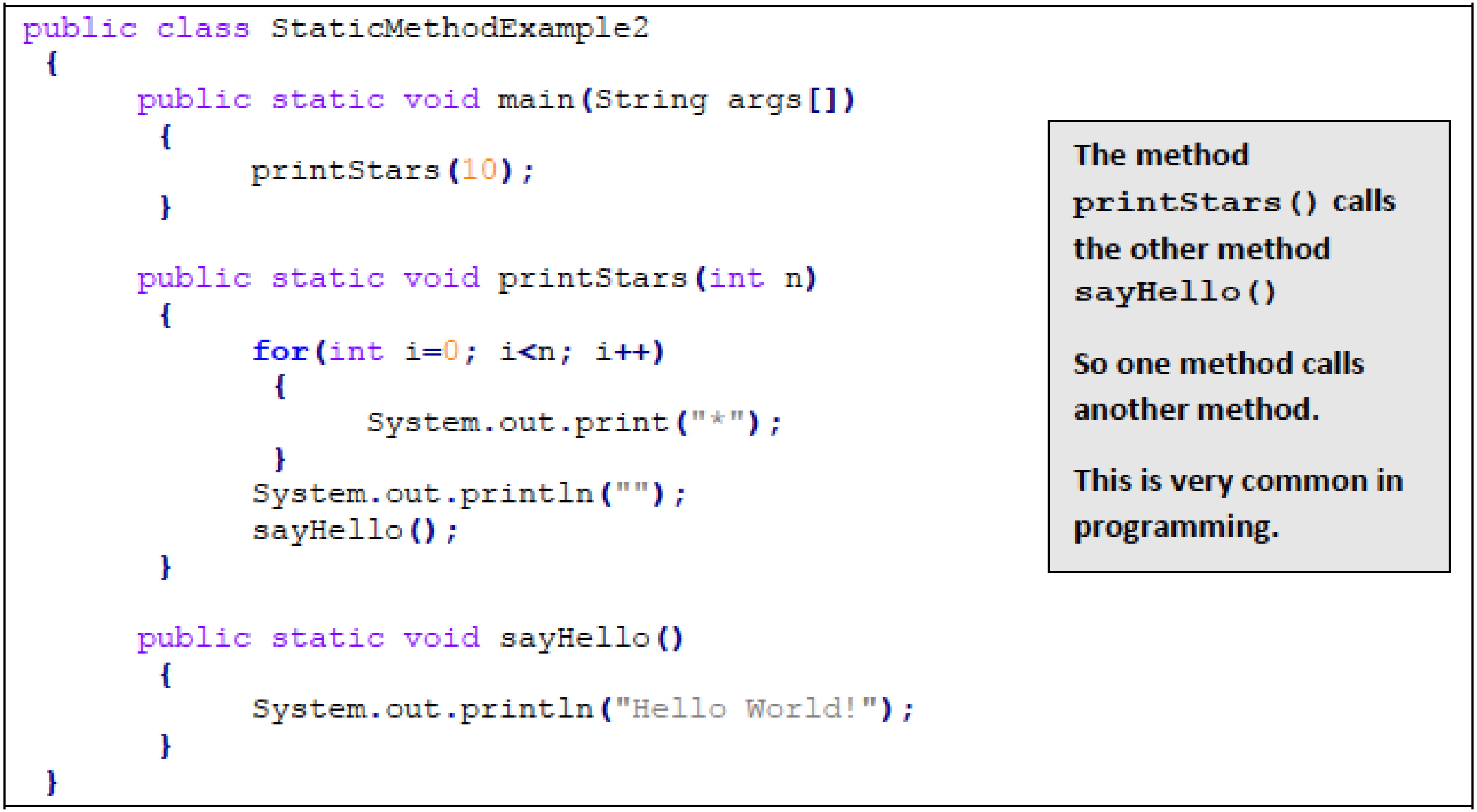
– For example, the `main()` method calls the printStars() function.



– A recursive method is a method **that calls itself.**



What is the output of the following program?



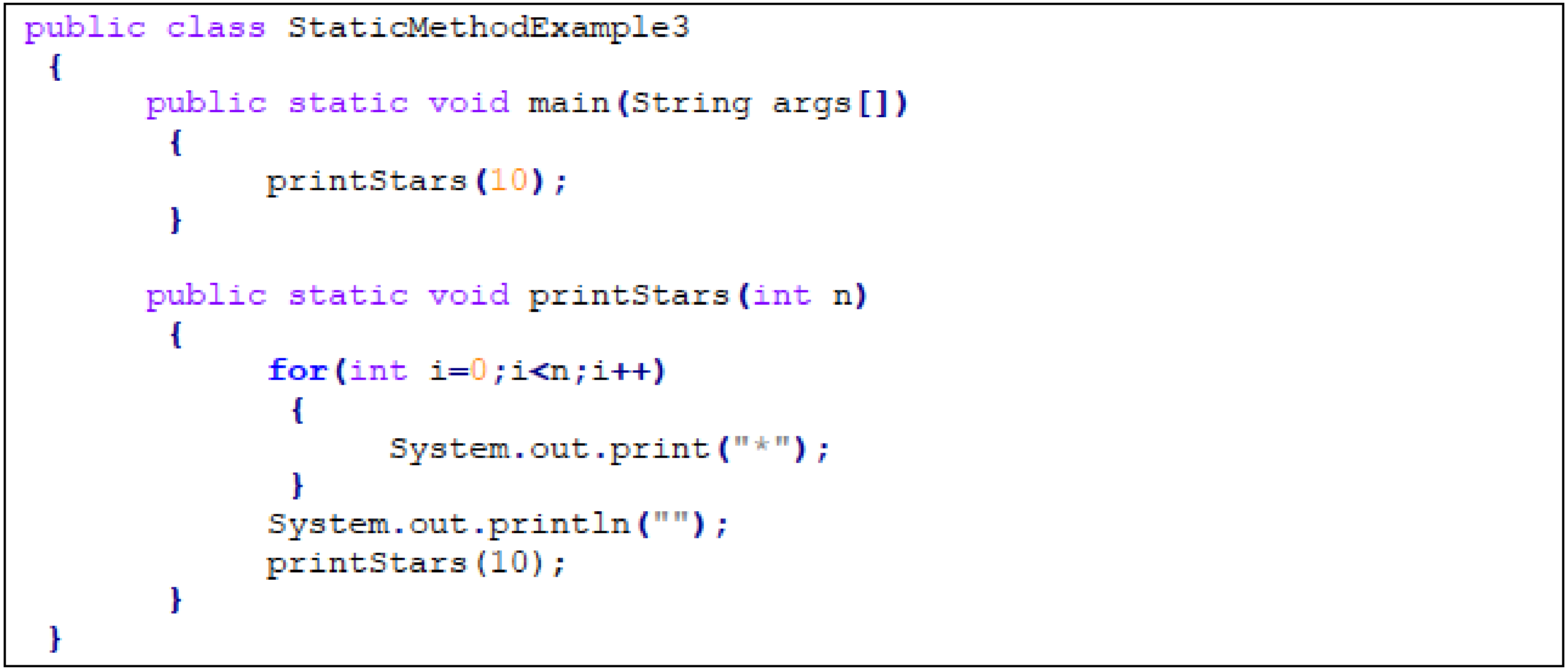
```
public class StaticMethodExample2
 {
      public static void main(String args[])
       {
            printStars(10);
       }

      public static void printStars(int n)
       {
            for(int i=0; i<n; i++)
             {
                  System.out.print("*");
             }
            System.out.println("");
            sayHello();
       }

      public static void sayHello()
       {
            System.out.println("Hello World!");
       }
 }
```

The method `printStars()` calls the other method `sayHello()`

So one method calls another method.

This is very common in programming.

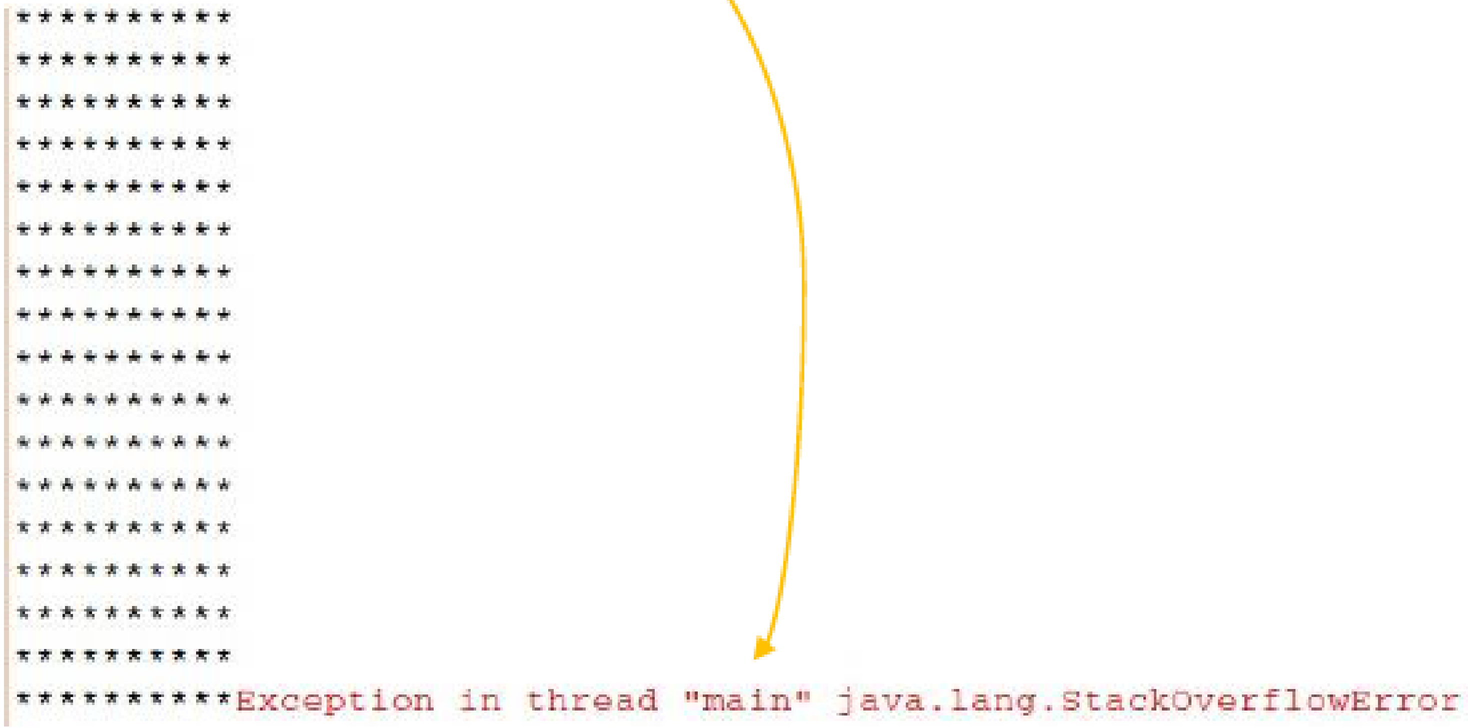
What is the output of the following program?



```java
public class StaticMethodExample3
 {
      public static void main(String args[])
       {
            printStars(10);
       }

      public static void printStars(int n)
       {
            for(int i=0;i<n;i++)
             {
                  System.out.print("*");
             }
            System.out.println("");
            printStars(10);
       }
 }
```

What is the output of the following program?

The program calls the `printStars()` method over and over again until there is not sufficient memory and the program crashes.

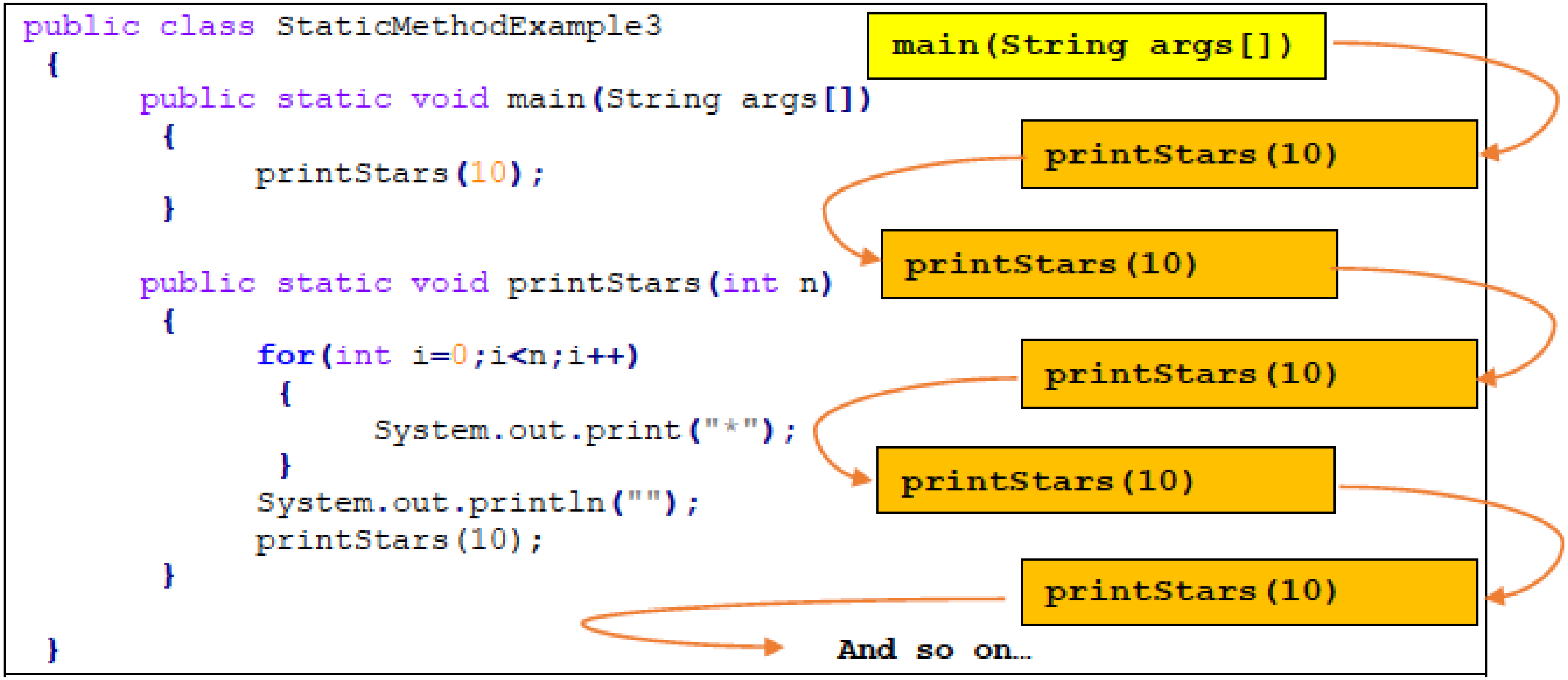


```
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********Exception in thread "main" java.lang.StackOverflowError
```

Recursion

- Concept of method calling itself over and over again is know as *recursion*
- Method keeps calling itself until some *stopping condition* is reached.
- It there is no stopping condition then the program will loop until the computer (Java Virtual Machine) runs out of memory (refuses to allocate more memory)

Recursion



```
public class StaticMethodExample3
 {
        public static void main(String args[])
         {
              printStars(10);
         }

        public static void printStars(int n)
         {
              for(int i=0;i<n;i++)
               {
                    System.out.print("*");
               }
              System.out.println("");
              printStars(10);
         }

 }
```

Recursion

- Recursion requires us to modify our thinking.
- We must stop thinking iteratively (for or while loops)
- While recursion may appear wasteful or even inefficient it is a very important concept in computer science and mathematics.

World´s Simplest Recursion Program

```
public class Recursion
{
        public static void ma
        {
                count(0);
                System.out.println();
        }

        public static void count (int index)
        {
                System.out.print(index);
                if (index < 2)
                        count(index+1);
        }
}
```

This program simply counts from 0-2:

012

This is where the recursion occurs. You can see that the count() function calls itself.

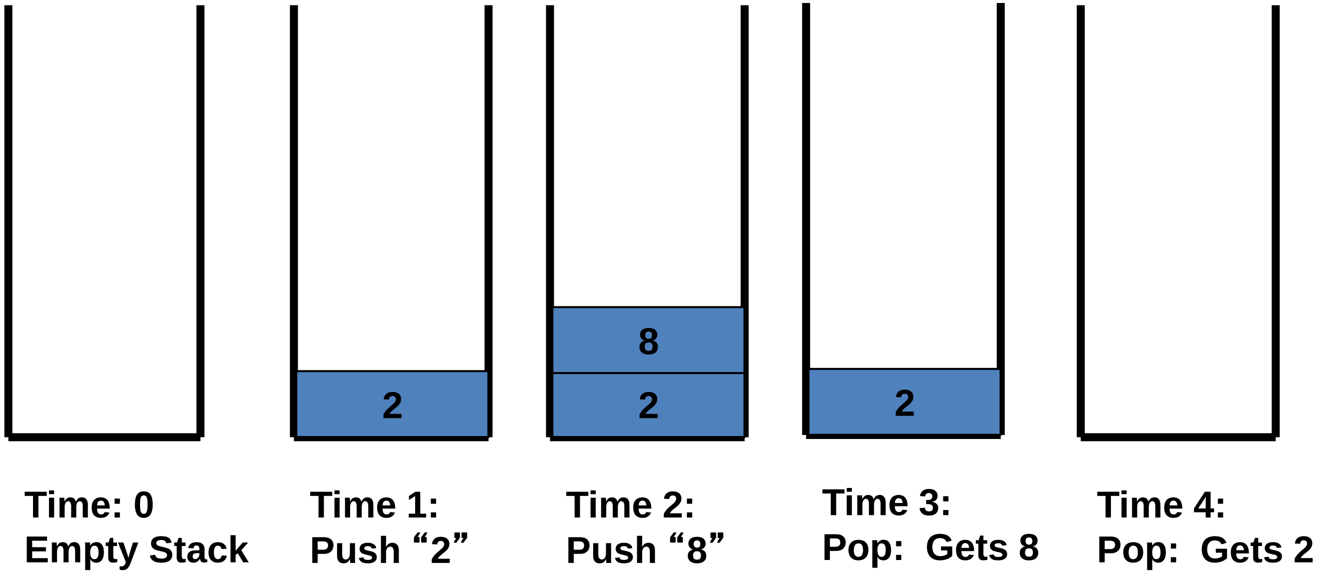
Visualizing Recursion

- To understand how recursion works, it helps to visualize what' s going on.
- To help visualize, we will use a common concept called the *Stack*.
- A stack basically operates like a container of trays in a cafeteria. It has only two operations:
  - Push: you can push something onto the stack.
  - Pop: you can pop something off the top of the stack.

Let' s see an example stack in action.

Stacks

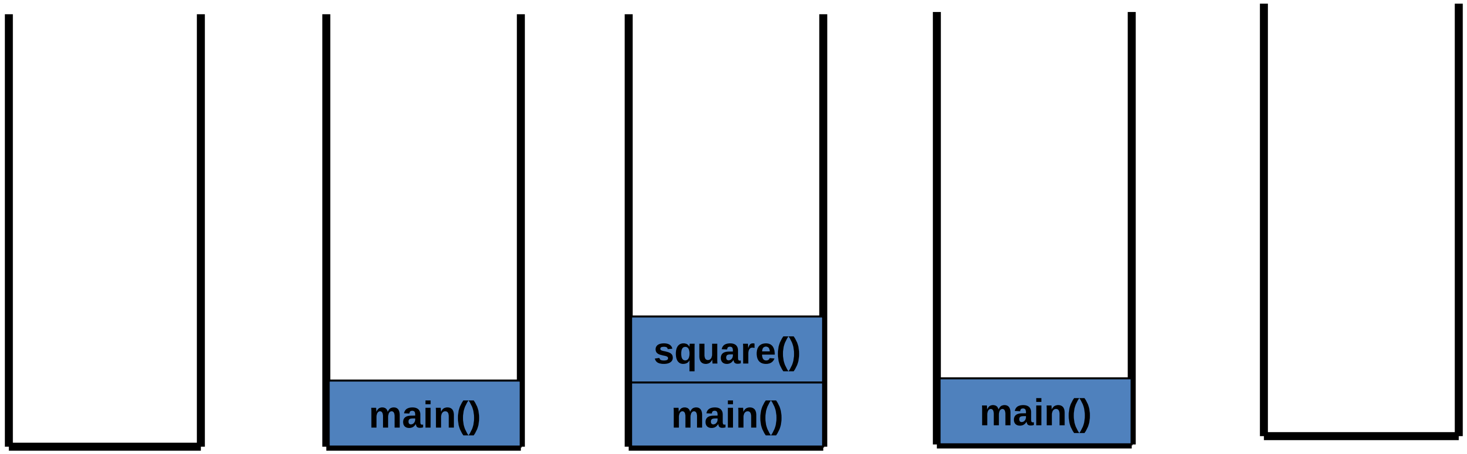
The diagram below shows a stack over time.
We perform two pushes and one pop.



Stacks and Methods

- When you run a program, the computer creates a stack for you.
- Each time you invoke a method, the method is placed on top of the stack.
- When the method returns or exits, the method is popped off the stack.
- The diagram on the next page shows a sample stack for a simple Java program.
- Let pretend we are calling a method int `square(int x)` which returns the square of `x` i.e. x=2 so we return 4

Stacks and Methods



Time: 0
Empty Stack

Time 1:
Push: main()

Time 2:
Push: square()

Time 3:
Pop: square()
returns a value.
method exits.

Time 4:
Pop: main()
returns a value.
method exits.

Stacks and Recursion

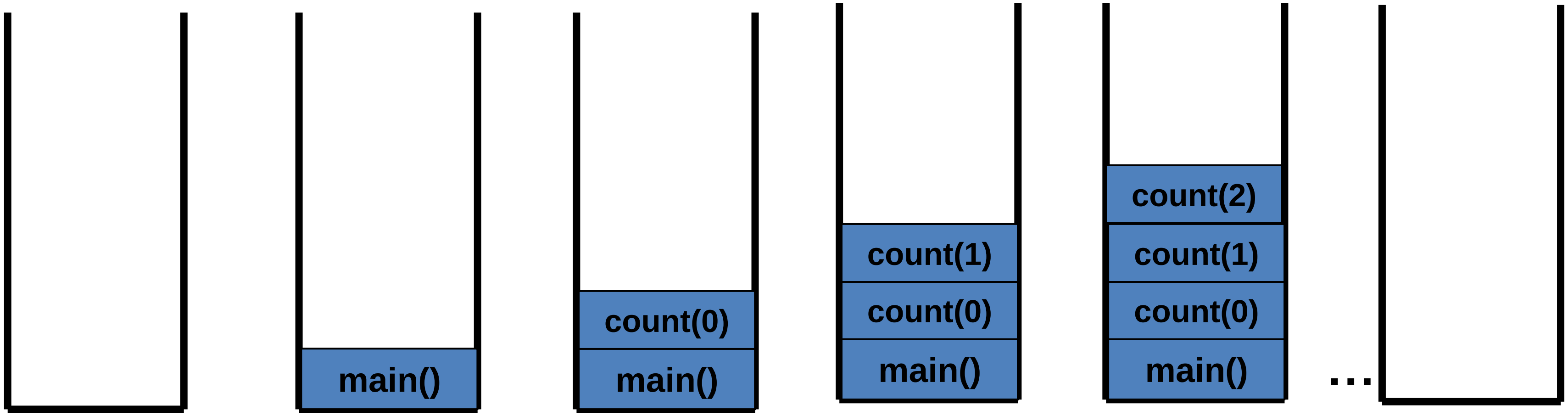
- Each time a method is called, you *push* the method on the stack.
- Each time the method returns or exits, you *pop* the method off the stack.
- If a method calls itself recursively, you just *push* another copy of the method onto the stack.
- We therefore have a simple way to visualize how recursion really works.

Back to the Simple Recursion Program

Here' s the code again. Now, that we understand stacks, we can visualize the recursion.

```
public class Recursion1V0
{
      public static void main (String args[])
      {
            count(0);
            System.out.println();
      }

      public static void count (int index)
      {
            System.out.print(index);
            if (index < 2)
                  count(index+1);
      }
}
```

Stacks and Recursion in Action



Time: 0
Empty Stack

Time 1:
Push: main()

Time 2:
Push: count(0)

Time 3:
Push: count(1)

Time 4:
Push: count(2)

Times 5-8:
Pop everything

Inside count(0):
print (index);
→ 0
if (index < 2)

count(index+1);

Inside count(1):
print (index);
→ 1
if (index < 2)

count(index+1);

Inside count(2):
print (index); → 2
if (index < 2)
count(index+1);

This condition now fails!Hence, recursion stops, and we proceed to pop all functions off the stack.

Recursion, Variation 1

What will the following program do?

```
public class Recursion1V1
{
      public static void main (String args[])
      {
            count(3);
            System.out.println();
      }

      public static void count (int index)
      {
            System.out.print(index);
            if (index < 2)
                  count(index+1);
      }
}
```

Recursion, Variation 2

What will the following program do?

```
public class Recursion1V2
{
        public static void main (String args[])
        {
                count(0);
                System.out.println();
        }

        public static void count (int index)
        {
                if (index < 2)
                        count(index+1);
                System.out.print(index);
        }
}
```

Note that the print statement has been moved to the end of the method.

Recursion

- In computer science, some problems are more easily solved by using recursive functions.
- If you go on to take a computer science algorithms course, you will see lots of examples of this.
- For example:
  - Traversing through a directory or file system.
  - Traversing through a tree of search results.
- For today, we will focus on the basic structure of using recursive methods.

Two Types of Recursion

- **Direct recursion:** a method contains a reference or call to itself directly (like in the **printStars()** example)
- **Indirect Recursion** a method calls another method that eventually calls the original method e.g. `method_a()` calls `method_b()` and `then method_b()` calls `method_a()`.

How Recursion works?

- A recursive computation solves a problem by using the solution of the same problem, but with simpler values. We call this the **recursive step.**
- For recursion to terminate or stop there must also be a special case for the simplest values. We call this the **base case** (or **anchor case** or **stopping condition**).
- The **base case** is the case in which the method value is specified for one or more known values of the input parameters.

How Recursion works?

- A **recursive step** (or **inductive step**) is the step in which the action to be taken for the current value of the parameter is defined in terms of previously defined values.
- In order to perform recursion we have to consider the following two perspectives:
  - 1. How can the simplest instance of the problem be solved? *(Base case*)
  - 2. Given a more complicated instance of the problem, how can it be made more like the simplest instance? i.e. how can it be brought *closer to the simplest instance of the problem (make it like the base case)*?

Palindrome Example

- Let’s say we want to test if a String is a palindrome.
- A palindrome is a string of text that is the same read forwards or backwards.
- Another way to think of it is as a string whose first half is a mirror image of its second half.
- Two examples of palindromes are: **DEED NAVAN**

You have already written a java program to check if a string is a palindrome using iteration now lets try it using a recursive approach.

Three Steps to Recursive Success

Reduction, - making the problem smaller

•We could check to see if the first and last characters are the same. In the case of NAVAN, the first and last characters are the same. So let’s remove them.

•We are left with the string AVA.

•Again we can see that the first and last characters are the same so we remove them.

•We are left with the string V.

•So now we can say that a word is a palindrome if:

1. The first and last characters are the same, and
2. The word obtained after removing these characters is also a palindrome.

Three Steps to Recursive Success

Base Cases - handling simplest values. The key is to find solutions to the simplest inputs (base case).

Case 1: Strings with no characters (Empty String).

- This is a palindrome.

Case 2: Strings with 1 character.

- This is a palindrome.

Case 3: Strings with two (or more characters).

- Follow our reduction step (i.e. check first and last characters for a match and if there is a match remove the first and last characters and (rinse and ☺) repeat).

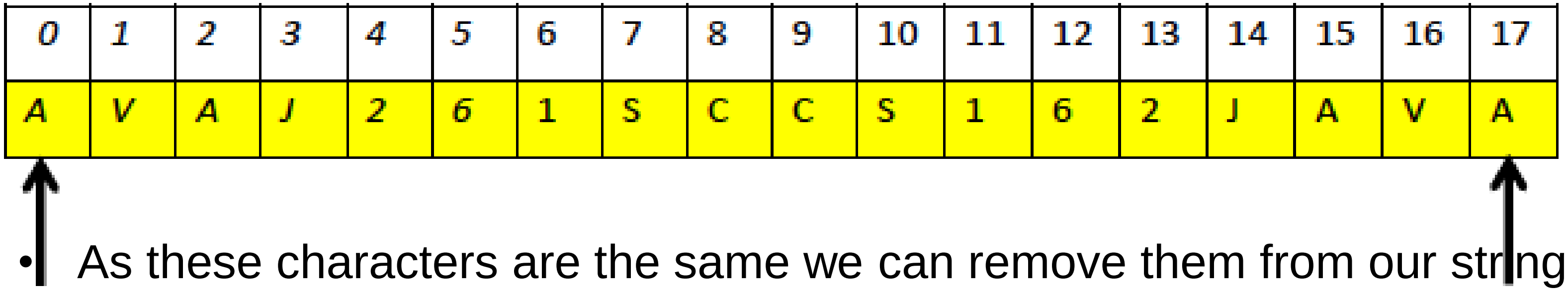
Three Steps to Recursive Success

Implement - combining base cases and reduction step.

- Now that we have our base cases and reduction step, it’s time to combine them to implement our solution.
- We write an `if` statement which will include the base case and reduction step. Additionally, **if there is a termination condition other than the base case** then that needs to be considered as well.

Palindrome Revisited

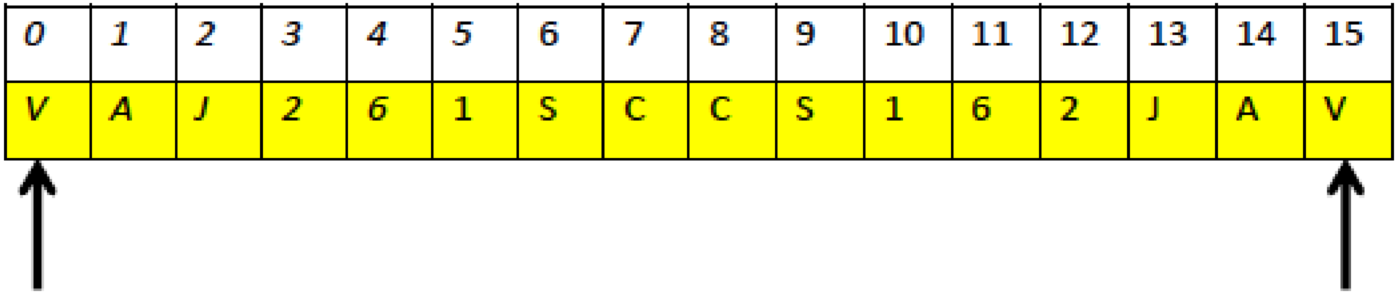
- Let us look at an example using the string: “AVAJ261SCCS162JAVA”.
- Using the technique described above let us start by comparing the first and the last character.
- We can see that the first character at position 0 of the string is “A” and the last character at position 17 is “A”.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	V	A	J	2	6	1	S	C	C	S	1	6	2	J	A	V	A

- As these characters are the same we can remove them from our string.
- We now repeat the process of comparing the first and the last characters.
- This time we are comparing the characters at position 0 and position 15 of our new string.
- Both of these positions contain the character “V”.

Palindrome Revisited



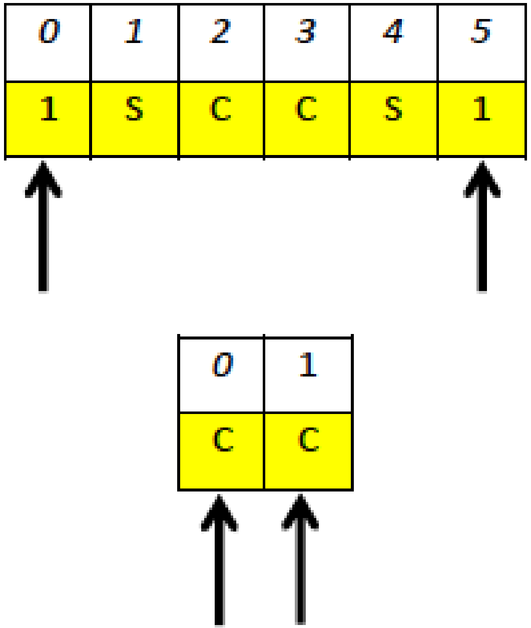
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	A	J	2	6	1	S	C	C	S	1	6	2	J	A	V

- We keep repeating this pattern of checking the first and last character of the string and if they are equal, we remove them from our string, as the string is a potential palindrome.

- An intermediate step in our string reduction and the final string to check are:

Palindrome Revisited

- An intermediate step in our string reduction and the final string to check are:



0	1	2	3	4	5
1	S	C	C	S	1

0	1
C	C

Palindrome Revisited

- We can see that using the string: “AVAJ261SCCS162JAVA” and following our pattern we reduce our string to “CC”.

- Again we compare the first and last characters and remove these from our string as they are equal.

- We have gone through all the characters of the string and our result is an empty string.

- We have completed the recursive process of checking the first and last characters until we have arrived at an empty string which is our ending condition.

- This means that our string “AVAJ261SCCS162JAVA” is a palindrome.

- Let us now look at implementing a recursive solution in Java to check if a sting is a palindrome.