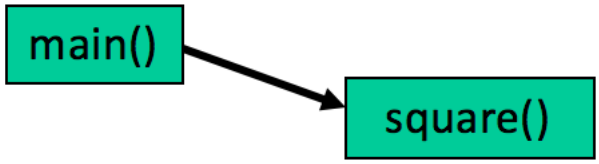
Week 22 Lecture

Recursion

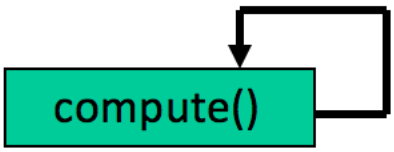
"In order to understand recursion, one must first understand recursion."

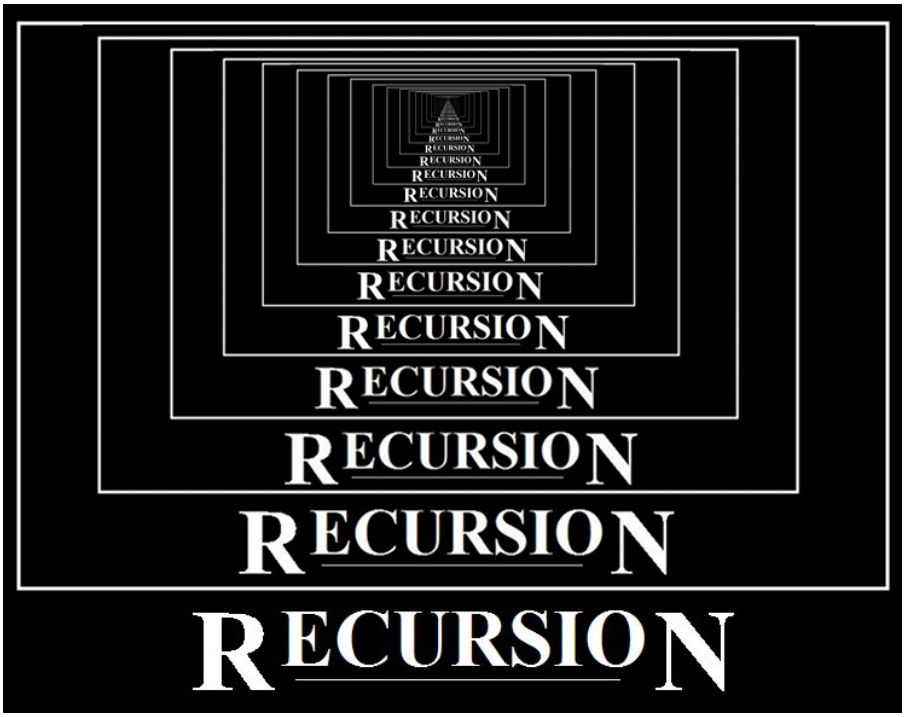
So far, we have seen methods that call other functions.

- For example, the main() method calls the square() function.



A method that calls itself is known as a recursive method. And, this technique is known as recursion.



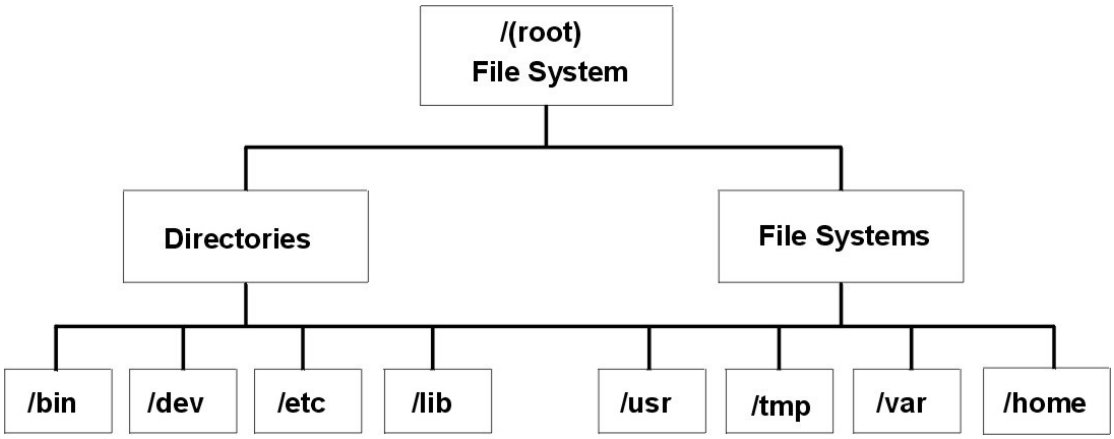


Why Recursion?

- 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.

When should I use recursion?

- Recursion is made for solving problems that can be broken down into smaller, repetitive problems.
- It is especially good for working on things that have many possible branches and are too complex for an iterative approach.
- One good example of this would be searching through a file system. You could start at the root folder, and then you could search through all the files and folders within that one. After that, you would enter each folder and search through each folder inside of that.



- Recursion works well for this type of structure because you can search multiple branching paths without having to include many different checks and conditions for every possibility.
- You might notice that the image above of the file system looks a lot like a tree structure. Trees and graphs are another time when recursion is the best and easiest way to do traversal.

Following concepts are used in recursion:

1. Base Case: Case for which method returns some definite value
2. Recursive Case: For which method calls itself
3. Every time when we call the method it should come closer to Base case

Example:

```
public class Factorial {

	public static void main(String[] args) {

		System.out.println(factorial(5));

	}

	public static int factorial(int n){
		// base case
		if(n<=1){
		    return 1;
		}else{
		// recursive case
		    return n * factorial(n-1);
		}
	}
}
```

5 * 4 * 3 * 2 * 1

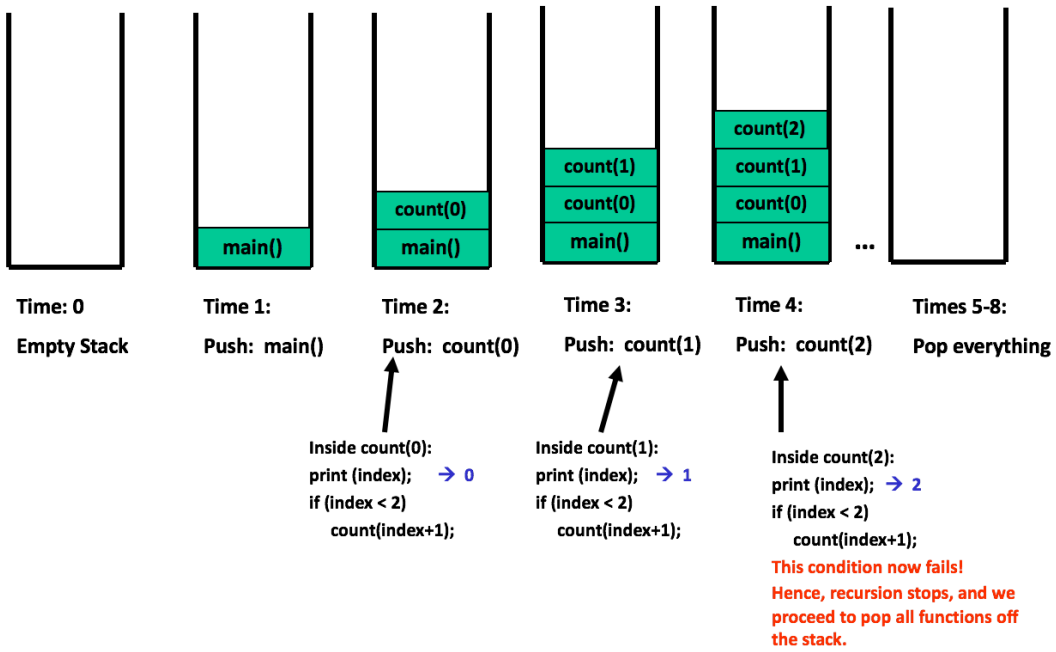
5! = 5 * 4 * 3* 2 * 1
= 5 * 4!
= 5 * 4 * 3!
= 5 * 4 * 3 * 2!
= 5 * 4 * 3 * 2 * 1

Output:

```
120
```

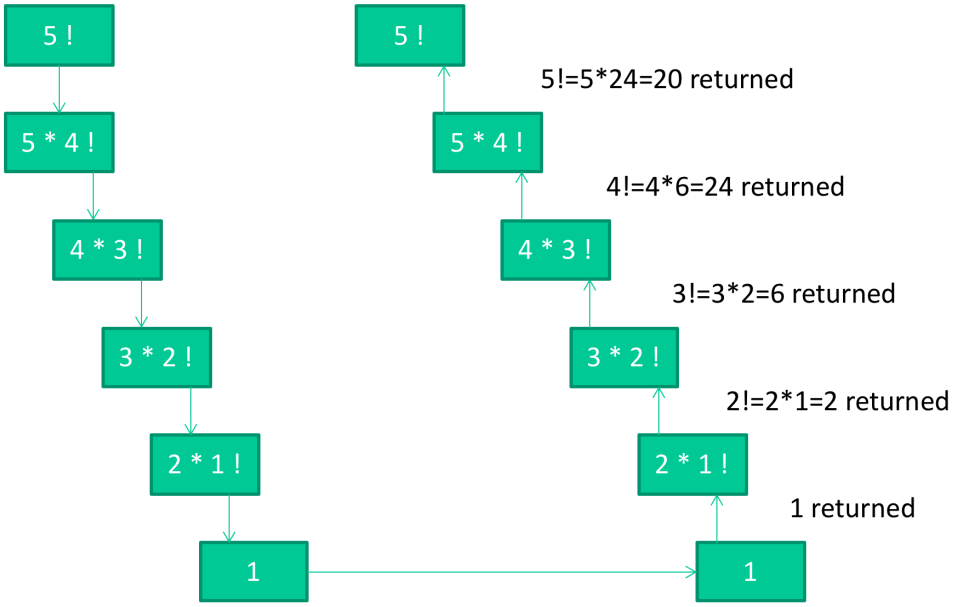
Stacks and Recursion

- 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.



Call Stack

```
factorial(5)
  factorial(4)
    factorial(3)
      factorial(2)
        factorial(1)
          return 1
        return 2*1 = 2
      return 3*2 = 6
    return 4*6 = 24
  return 5*24 = 120
```


5 !
5 * 4 !
4 * 3 !
3 * 2 !
2 * 1 !
1
5 !
5!=5*24=20 returned
5 * 4 !
4!=4*6=24 returned
4 * 3 !
3!=3*2=6 returned
3 * 2 !
2!=2*1=2 returned
2 * 1 !
1 returned
1

Recursion Vs Iteration

- Compared the two processes, we can find that they seem almost same, especially in term of mathematical function.
- On the other hand, when we consider the running processes of the two programs, they evolve quite differently.
- In the iterative case, the program variables provide a complete description of the state. If we stopped the computation in the middle, to resume it only need to supply the computer with all variables. However, in the recursive process, information is maintained by the computer, therefore "hidden" to the program.
- Generally speaking, yes recursion use more memory than iteration. This is because of the extensive use of the call stack.
- Recursion is generally used because of the fact that it is simpler to implement, and it is usually more ‘elegant’ than iterative solutions.
- Remember that anything that’s done in recursion can also be done iteratively, but with recursion there is generally a performance drawback.

Should we always use recursion?

Recursion seems really useful! Maybe we should use it all the time?

Well, like anything, recursion is best in doses. Recursion is a useful tool, but it can increase memory usage.

So, let’s go back to the factorial call stack image from above. Every time we add a new call to the stack, we are increasing the amount of memory that we are using.

```
public class RecursionTest {

	public static void main(String[] args) {
		findResult(5);
	}

	public static void findResult(int n){
	     if (n>0){
		findResult (n-1);
		System.out.print(n + " ");
	      }
	 }
}
```

What will be the output?

```
main
  findResult(5)
    findResult (4)
      findResult (3)
        findResult (2)
          findResult (1)
            findResult (0)
          print 1
        print 2
      print 3
    print 4
 print 5
```

1 2 3 4 5

StringBuilder

StringBuilder objects are like String objects, except that they can be modified. Hence Java StringBuilder class is also used to create mutable (modifiable) string object.

Example:

String text = “Hello”;

1.
text = “Hello World”;

2.
text = text + “ World”

```
StringBuilder strBuilder = new StringBuilder("Hello");     String text = “Hello”;
strBuilder.append("World");          text = text + “World”;
System.out.println(strBuilder);
strBuilder.append(" Test");          text = text + “ Test”;
System.out.println(strBuilder);
```

Output

```
HelloWorld
HelloWorld Test
```