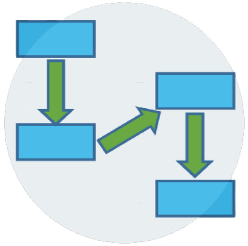
Recursion



Tushar B. Kute,
http://tusharkute.com

Recursion

- In Java, a method that calls itself is known as a recursive method. And, this process is known as recursion.
- A physical world example would be to place two parallel mirrors facing each other.
- Any object in between them would be reflected recursively.

How it works?

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

static void recurse() {
    ... .. ...
    recurse()
    ... .. ...
}
```

Normal Method Call

Recursive Call

Applications

- Recursion is often used to solve problems that can be naturally divided into subproblems, such as:
  - Traversing a tree or graph
  - Searching for an element in a sorted list
  - Sorting a list
  - Computing the factorial of a number
  - Reversing a string

Recursion

- To write a recursive function, you need to identify two things:
  - The base case: This is the simplest form of the problem that can be solved directly.
  - The recursive step: This is how the function breaks down the problem into smaller subproblems and then calls itself to solve those subproblems.

Recursion

- Recursion can be a powerful tool for solving problems, but it can also be difficult to understand and debug.
- It is important to carefully design recursive functions and to test them thoroughly before using them in production.

Recursion

- Here are some tips for writing recursive functions:
  - Make sure the function has a base case.
  - Make sure the recursive step reduces the problem to a smaller version of the same problem.
  - Avoid infinite recursion.
  - Test the function thoroughly before using it in production.

Base Condition

- The base condition in recursion is the simplest form of the problem that can be solved directly.
- It is the stopping point for the recursion, and it is what prevents the function from recursively calling itself forever.
- For example, the base condition for the factorial function is n == 0. This is because the factorial of 0 is simply 1, and there is no need to recursively call the function to solve this problem.

Base Condition

- Here are some tips for writing base conditions in recursive functions:
  - Make sure the base condition is the simplest form of the problem that can be solved directly.
  - Make sure the base condition is always reachable.
  - Test the base condition thoroughly to make sure it works as expected.

Direct Recursion

- Direct and indirect recursion in Java are two types of recursion where a function calls itself.
- Direct recursion occurs when a function directly calls itself. For example, the following function to calculate the factorial of a number uses direct recursion:

```
public static int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Indirect Recursion

- Indirect recursion occurs when a function calls another function, which then calls the original function directly or indirectly.
- For example, the following two functions use indirect recursion to reverse a string:

Indirect Recursion

```
public static String reverse(String str)
{
    if (str.length() == 0) {
        return "";
    } else {
        return reverse(str.substring(1))+str.charAt(0);
    }
}
public static String reverseHelper(String str) {
    return reverse(str);
}
```

Direct vs. Indirect Recursion

Feature	Direct recursion	Indirect recursion
Definition	A function calls itself directly.	A function calls another function, which then calls the original function directly or indirectly.
Example	The factorial function	The function to reverse a string
Use cases	Simple problems	Complex problems, such as implementing complex data structures

Memory Allocation

- When a function is called in Java, a stack frame is allocated on the stack.
- The stack frame is a region of memory that stores the local variables and parameters of the function. When the function returns, the stack frame is deallocated.
- In recursive functions, a new stack frame is allocated for each recursive call.
- This means that the memory usage of a recursive function can grow exponentially with the number of recursive calls.

Memory Allocation

- To avoid this, it is important to design recursive functions carefully.
- One way to do this is to use a base case to stop the recursion as soon as possible. Another way to reduce memory usage is to use tail recursion.
- Tail recursion is a type of recursion where the recursive call is the last thing the function does.
- This means that the stack frame for the current recursive call can be deallocated before the stack frame for the previous recursive call is returned.

Memory Allocation

- Here is an example of a recursive function that is not tail recursive:

```
public static int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Memory Allocation

- This function will allocate a new stack frame for each recursive call.
- Here is an example of a tail recursive function to calculate the factorial of a number:

```
public static int factorialTailRecursive(int n, int
accumulator) {
    if (n == 0) {
        return accumulator;
    } else {
        return factorialTailRecursive(n - 1, n *
accumulator);
    }
}
```

Memory Allocation

- Here are some tips for reducing memory usage in recursive functions:
  - Use a base case to stop the recursion as soon as possible.
  - Use tail recursion whenever possible.
  - Avoid using global variables in recursive functions.
  - Pass as few arguments to recursive functions as possible.
  - Use tail call optimization (TCO), if available on your compiler.

Pros: Recursion

- Elegance: Recursive functions can be very concise and elegant, especially for problems that can be naturally divided into subproblems.
- Expressiveness: Recursion can be used to express complex algorithms in a clear and concise way.
- Modularity: Recursive functions can be used to implement complex algorithms in a modular way, making them easier to understand and maintain.
- Efficiency: Tail recursive functions can be very efficient, and some compilers can optimize them to use the same stack frame for all recursive calls.

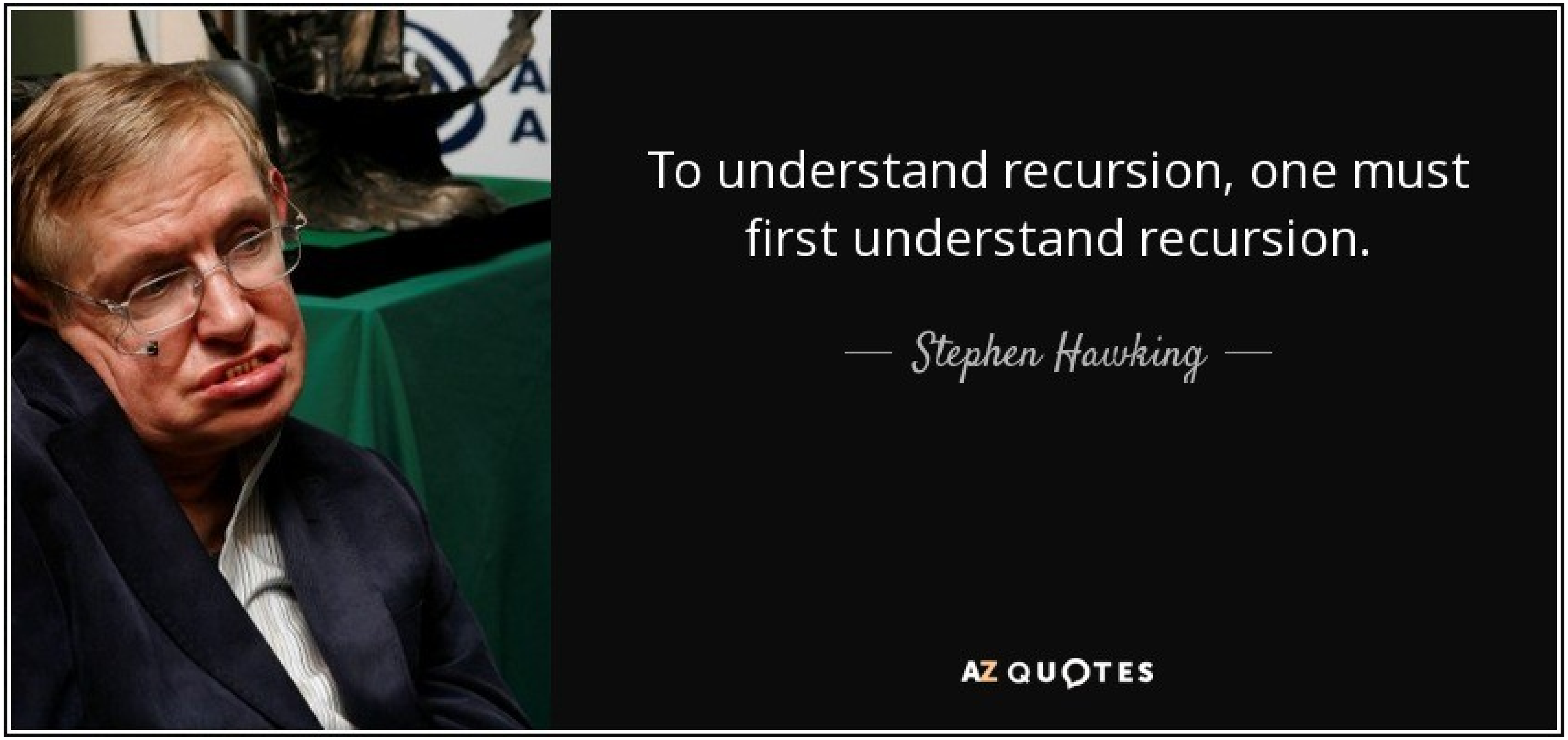
Cons: Recursion

- Memory usage: Recursive functions can use a lot of memory, especially if they are not tail recursive.
- Complexity: Recursive functions can be difficult to understand and debug, especially for complex problems.
- Stack overflows: Recursive functions can cause stack overflows if the recursion depth is too large.

Function complexity

- The function complexity during recursion depends on the following factors:
  - The number of recursive calls: The more times the function calls itself, the more complex the function will be.
  - The complexity of the recursive calls: The complexity of the recursive calls also contributes to the overall complexity of the function.
  - The complexity of the base case: The complexity of the base case is the complexity of the simplest form of the problem that can be solved directly.

Conclusion



To understand recursion, one must first understand recursion.
— Stephen Hawking —
AZ QUOTES

Thank you

This presentation is created using LibreOffice Impress 7.4.1.2, can be used freely as per GNU General Public License

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com