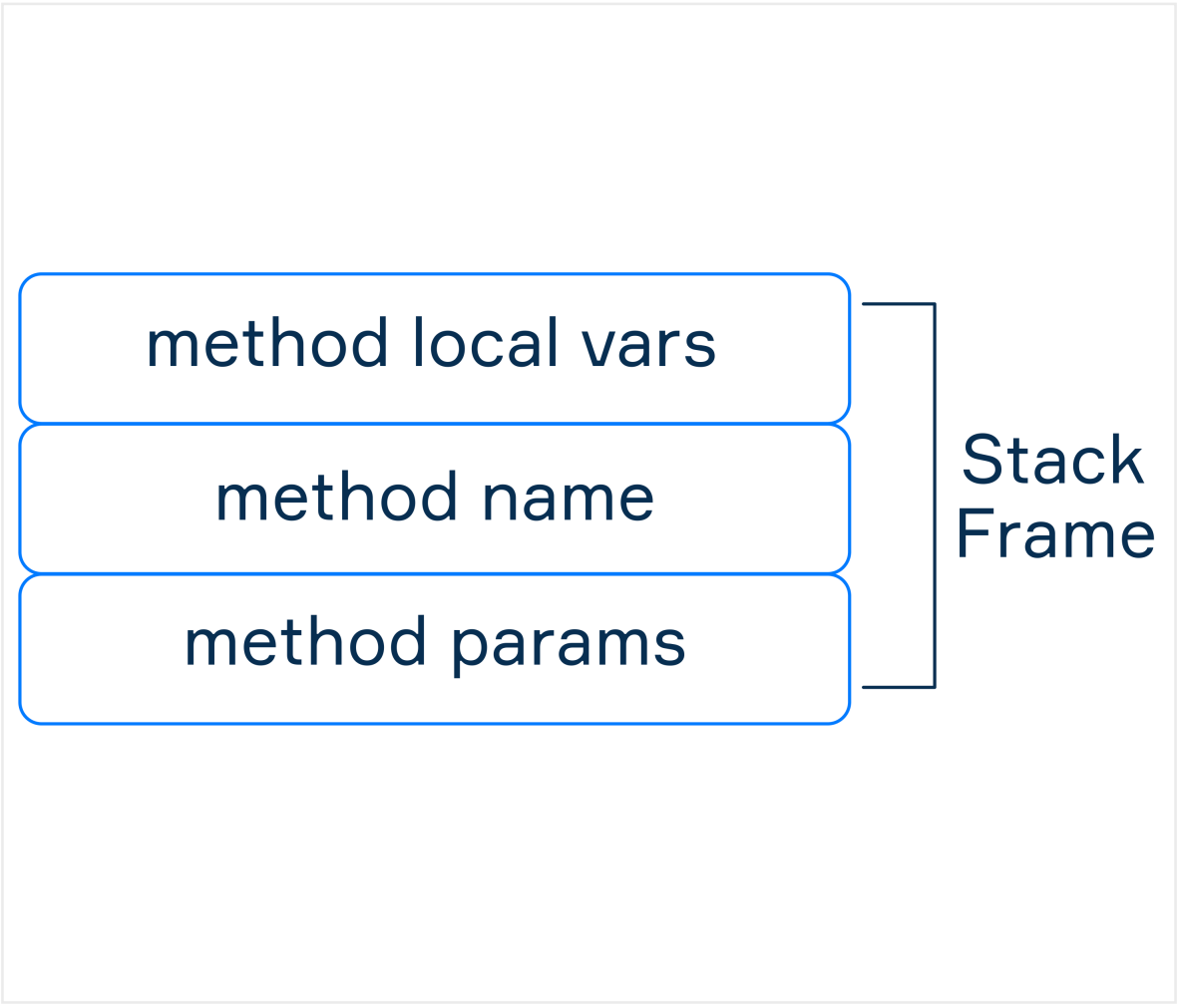
Call Stack

Call stack structure

JVM uses a **call stack** (or **execution stack**) to understand which method should be invoked next and to access information regarding the method. The call stack is composed of **stack frames** that store information about methods that have not yet terminated. The information includes the address of a method, parameters, local variables, intermediate computations, and some other data.



As a regular stack, the call stack follows the rule **Last In First Out (LIFO)**. It means stack frames are pushed at the top and move everything down. A new stack frame is added when the execution enters a method. And the stack frame is removed from the call stack if the execution of the method is done.

Stack frame example

```
public class Main {
    public static void main(String[] args) {
        int n = 99;
```

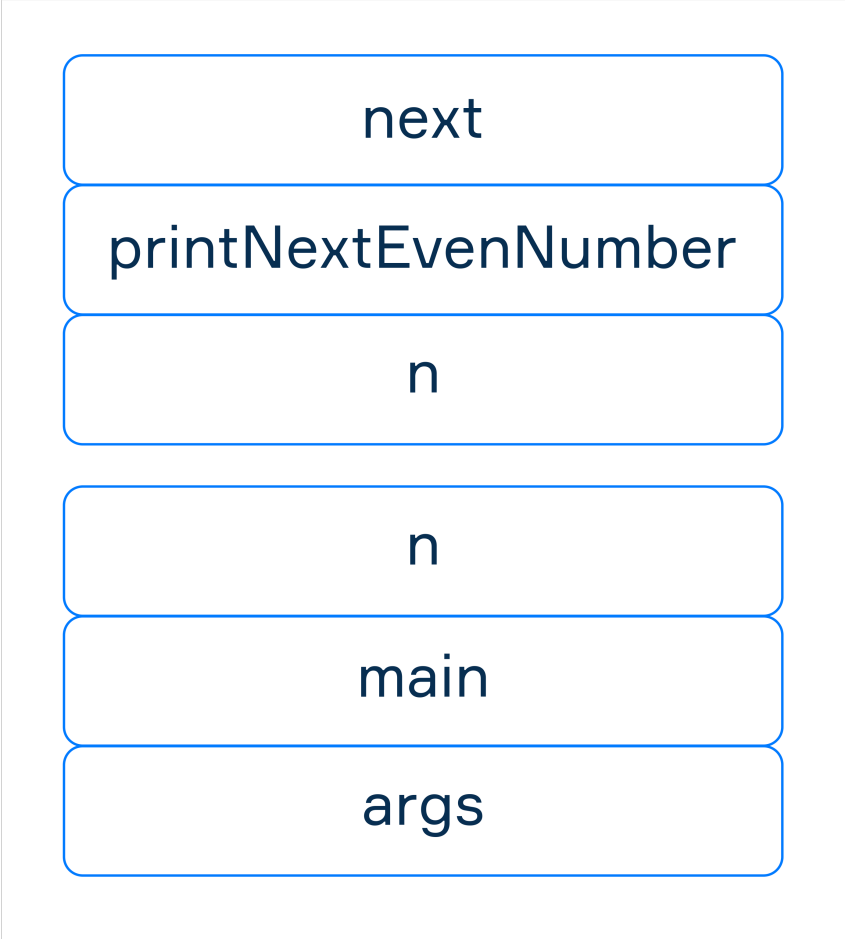
```
        printNextEvenNumber(n);
    }

    public static void printNextEvenNumber(int n) {
        int next = (n % 2 == 0) ? n + 2 : n + 1;
        System.out.println(next);
    }
}
```

The program declares two methods: main and printNextEvenNumber.
The first method to be invoked is main. Each time a method is invoked, a new stack frame is created. The stack frame for main is structured in the following way:

1. The method parameters (args) are pushed on the stack.
2. The method address (shown in the scheme as the method name — main) is added to the stack frame to keep a reference to where to return from the following method calls.
3. The local variables (n) are pushed on the stack.

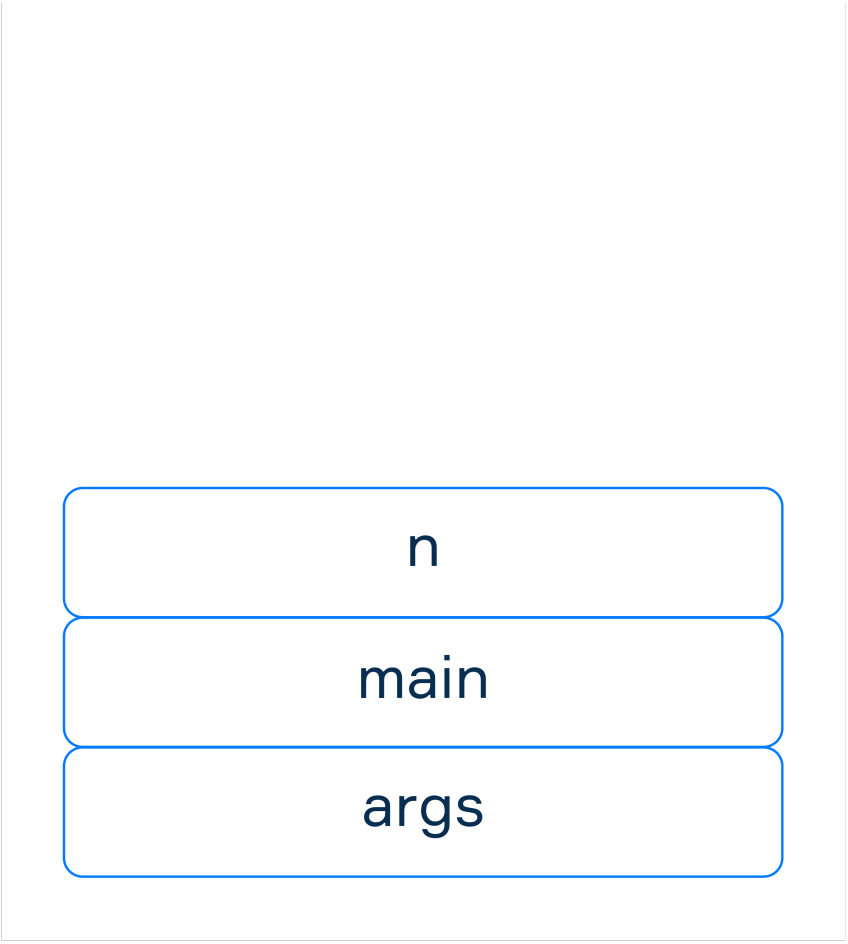
Actually, the stack stores just a reference to the args array since all reference types are stored in heap memory. But, the stack stores the actual value of n (which is 99 in our example).

Stack and methods execution
The next method to be invoked is printNextEvenNumber. As always, a new stack frame is created. The method parameters (n), address (printNextEvenNumber for simplicity), and local variables (next) are added to the new stack frame.



Note, both frames have variables named n, but these variables are not the same since they belong to different methods.

Next, the program executes the method at the top of the call stack (printNextEvenNumber). After the execution, the current frame printNextEvenNumber is removed from the call stack and the previous frame main continues the execution.



The standard method println works in a similar way as the methods we have defined — the new stack frame is created and when println finishes its work, the printNextEvenNumber continues the execution.

Any Java program works almost in this way. When the stack is empty, the execution stops.

Stack overflow

The number of possible method invocations depends on the amount of memory allocated for the stack. When your stack contains too many stack frames, it can be overflowed. It leads to the StackOverflowError that will stop the execution. The stack size can be set with the -Xss command line switch like:

java YourProgramName -Xss256k

But we recommend you to be careful with it and read some articles on the Internet before modifying the default stack size. Also, sometimes the StackOverflowError points to incorrect recursion calls in your program.