Let's break down a simple function like \*\*addition\*\* to understand how a stack frame works with its local variables.

### Code Example:

Here’s a simple Java program that calls a function to add two numbers:

```java

public class StackExample {

// Addition function

public static int add(int a, int b) {

int sum = a + b; // Local variable 'sum'

return sum;

}

public static void main(String[] args) {

int x = 10; // Local variable 'x'

int y = 20; // Local variable 'y'

int result = add(x, y); // Call to add(x, y)

System.out.println("Sum: " + result);

}

}

```

### What Happens in the Stack During Function Call:

#### 1. \*\*When `main()` is Executed\*\*:

- The JVM creates a \*\*stack frame\*\* for the `main()` method. This frame contains local variables `x`, `y`, and `result`.

- \*\*Stack after `main()` starts\*\*:

```

+-------------------+

| main() |

|-------------------|

| x = 10 |

| y = 20 |

| result = ? |

+-------------------+

```

#### 2. \*\*When `add(x, y)` is Called\*\*:

- When `add(x, y)` is called, a new \*\*stack frame\*\* is created for the `add()` function.

- The parameters `a` and `b` are initialized with the values `x` (10) and `y` (20), respectively.

- The local variable `sum` is declared and computed as `sum = a + b`.

- \*\*Stack after calling `add(10, 20)`\*\*:

```

+-------------------+

| add(a = 10, b = 20)|

|-------------------|

| sum = 30 |

+-------------------+

| main() |

|-------------------|

| x = 10 |

| y = 20 |

| result = ? |

+-------------------+

```

#### 3. \*\*Returning from `add()`\*\*:

- After computing `sum = a + b`, the function returns `30` (the value of `sum`).

- The stack frame for `add()` is \*\*popped off\*\* the call stack, and control returns to `main()`.

- \*\*Stack after returning from `add(10, 20)`\*\*:

```

+-------------------+

| main() |

|-------------------|

| x = 10 |

| y = 20 |

| result = 30 | <-- result now holds the return value of `add()`

+-------------------+

```

#### 4. \*\*Final State in `main()`\*\*:

- The `main()` method resumes, with `result` now set to `30`. It prints the result and then the program ends, with the `main()` frame being removed from the stack.

- \*\*Final Stack (after program execution ends)\*\*:

```

(empty stack)

```

### Detailed Stack Frame for `add()` Function:

Let's zoom into the \*\*stack frame\*\* for the `add()` function call. It looks something like this:

- \*\*Function\*\*: `add(int a, int b)`

- \*\*Local Variables\*\*:

- `a = 10` (value passed from `x` in `main()`)

- `b = 20` (value passed from `y` in `main()`)

- `sum = 30` (computed as `a + b`)

Here is how this looks in the stack frame:

```

+-------------------+

| add(a = 10, b = 20)|

|-------------------|

| sum = 30 |

+-------------------+

```

### How the Stack Works:

1. \*\*Function Call\*\*:

- When a function is called (like `add(x, y)`), the program pauses the execution of the current function (`main()` in this case) and creates a new stack frame for `add()`.

2. \*\*Local Variables in the Stack Frame\*\*:

- Inside the new stack frame for `add()`, the local variables `a`, `b`, and `sum` are created.

- The values of `x` and `y` are passed to the parameters `a` and `b`.

- The local variable `sum` is computed and stored in the same frame.

3. \*\*Returning from the Function\*\*:

- After `add()` finishes its computation, the stack frame for `add()` is popped off the stack.

- Control returns to `main()`, and the value returned by `add()` is assigned to the variable `result`.

4. \*\*Stack Unwinding\*\*:

- Once the result is printed and `main()` finishes, the stack frame for `main()` is popped off the stack, and the stack becomes empty again.

### Summary:

- Each function call gets its own \*\*stack frame\*\*, which stores local variables and function parameters.

- When the function returns, its stack frame is popped off, and the program resumes from where it left off in the calling function.

- This process of pushing and popping frames happens dynamically, allowing for the execution of nested or recursive function calls.