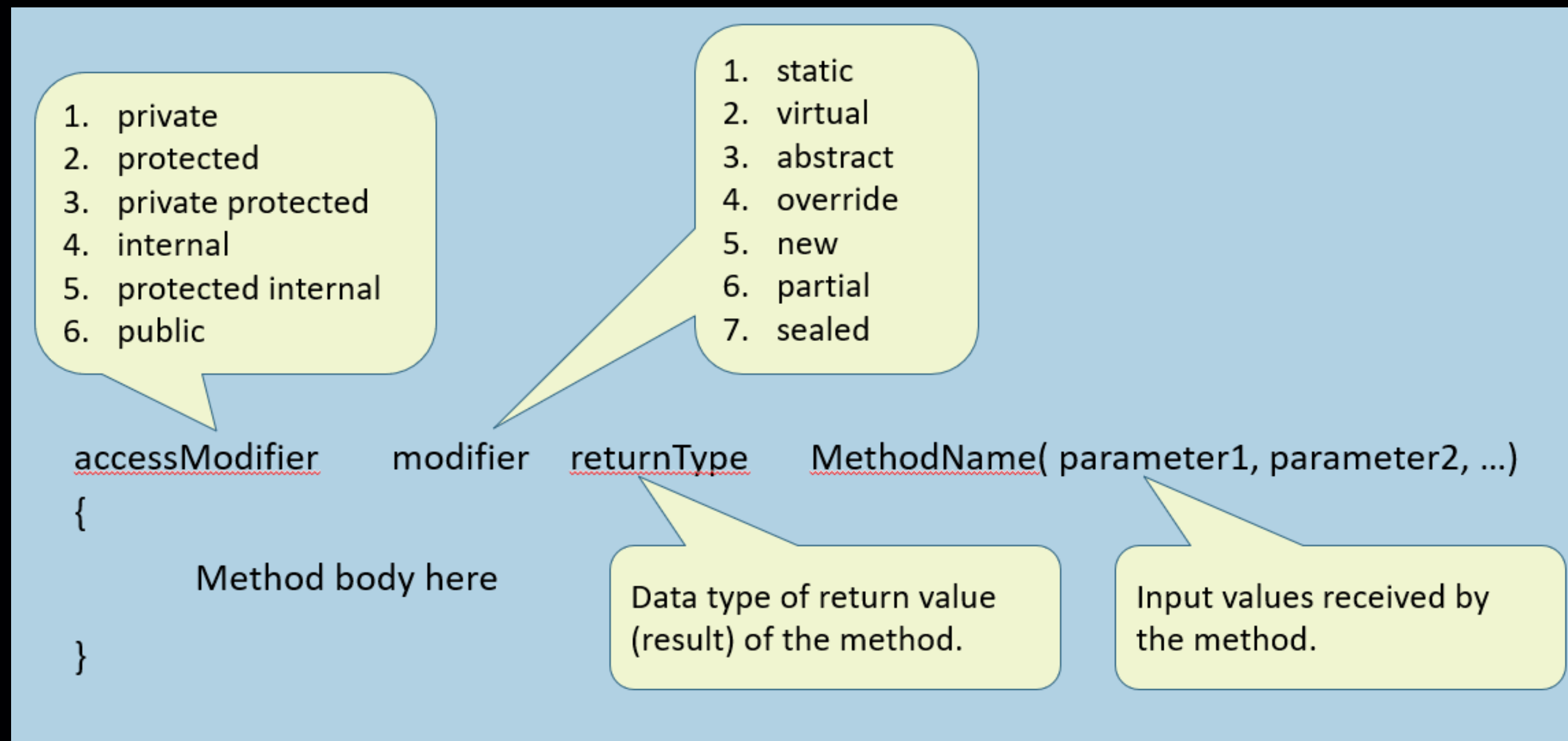


METHODS

- Method is a function (group of statements), to do some process based on fields.
- Methods are parts of the class.
- Methods can receive one or more input values as "parameters" and return a value as "return".



METHODS

Access Modifiers

Access Modifier (a.k.a. "Access Specifier" or "Visibility Modifier") specifies the accessibility of fields, where the fields can be accessible; they provide security for the fields.

Access Modifier	In the same class	In the child classes at the same assembly	In the other classes at the same assembly	Child classes at other assembly	Other classes at other assembly
private	Yes	No	No	No	No
protected	Yes	Yes	No	Yes	No
private protected	Yes	Yes	No	No	No
internal	Yes	Yes	Yes	No	No
protected internal	Yes	Yes	Yes	Yes	No
public	Yes	Yes	Yes	Yes	Yes

ENCAPSULATION

Encapsulation is a concept of:

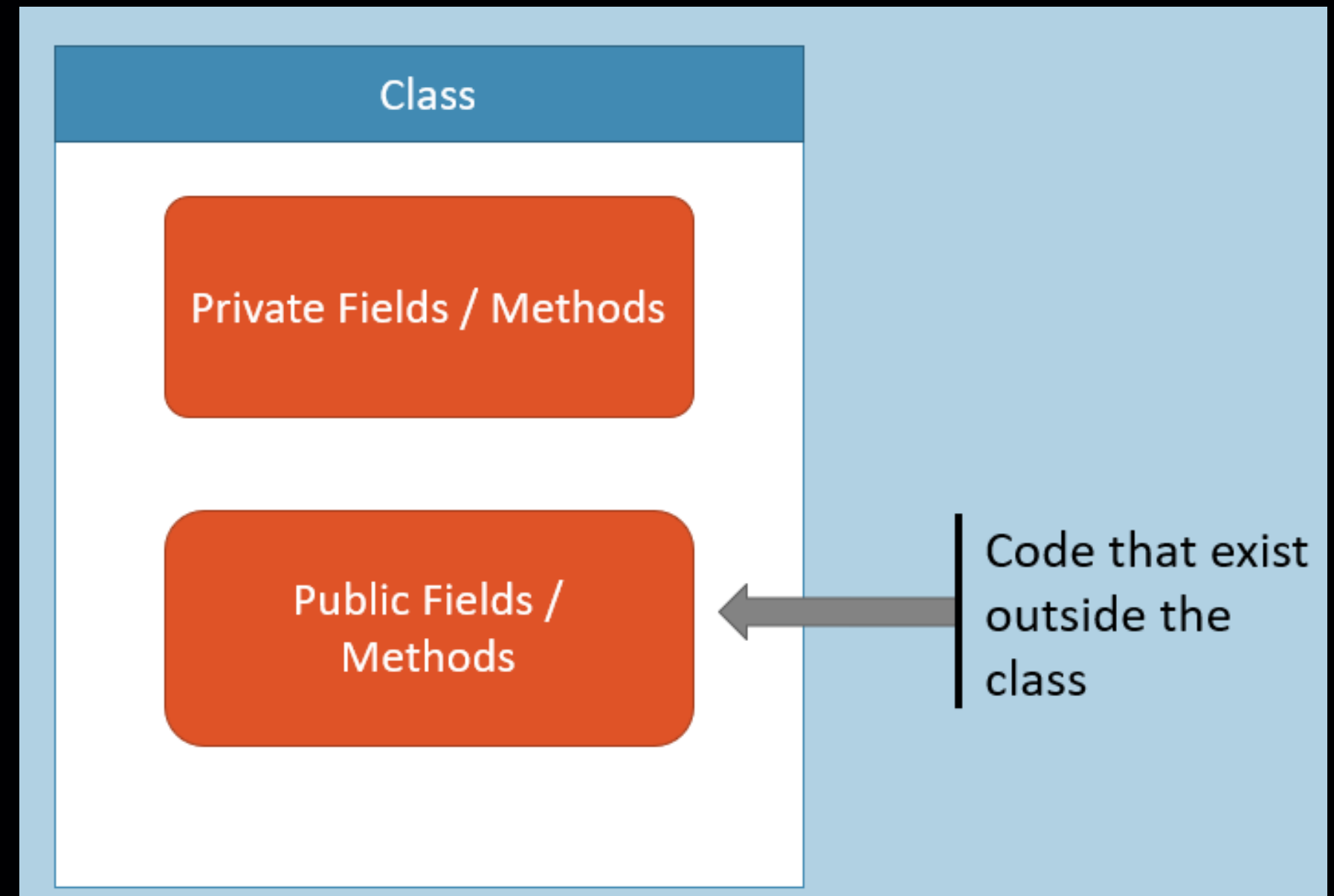
- Bundling the data (fields) and operations (methods) that manipulate the data together.
- Hides internal implementation details of an object and provides essential members to interact with them.

Benefits:

- Modularity
- Hiding implementation details
- Data Integrity

Implemented using:

- Private fields &
- Public properties or public methods



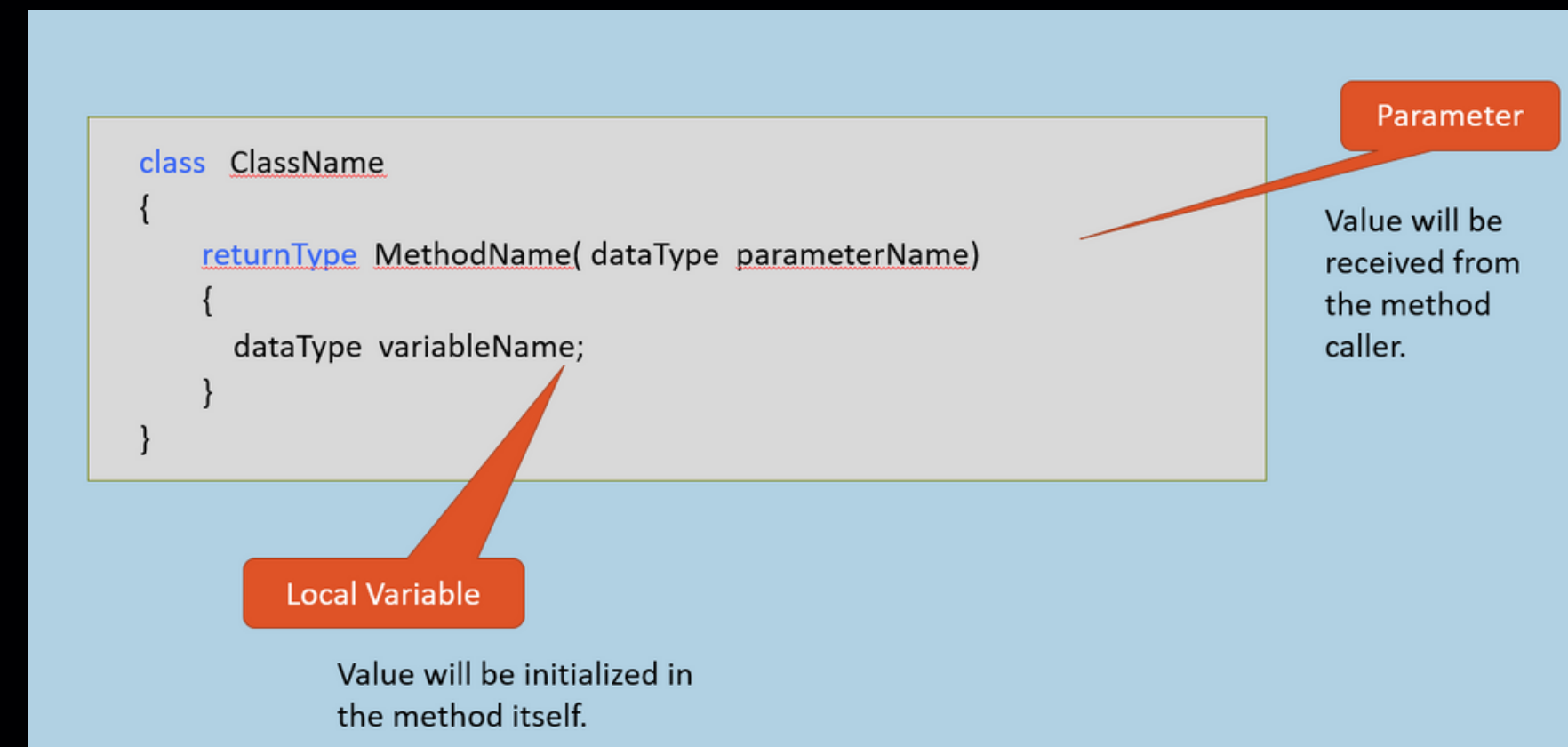
METHODS

Parameters:

- The variables that are being received from the "method caller" are called as "parameters".
- The parameters are stored in the Stack of the method.
- For every method call, a new stack will be created.

Local Variables:

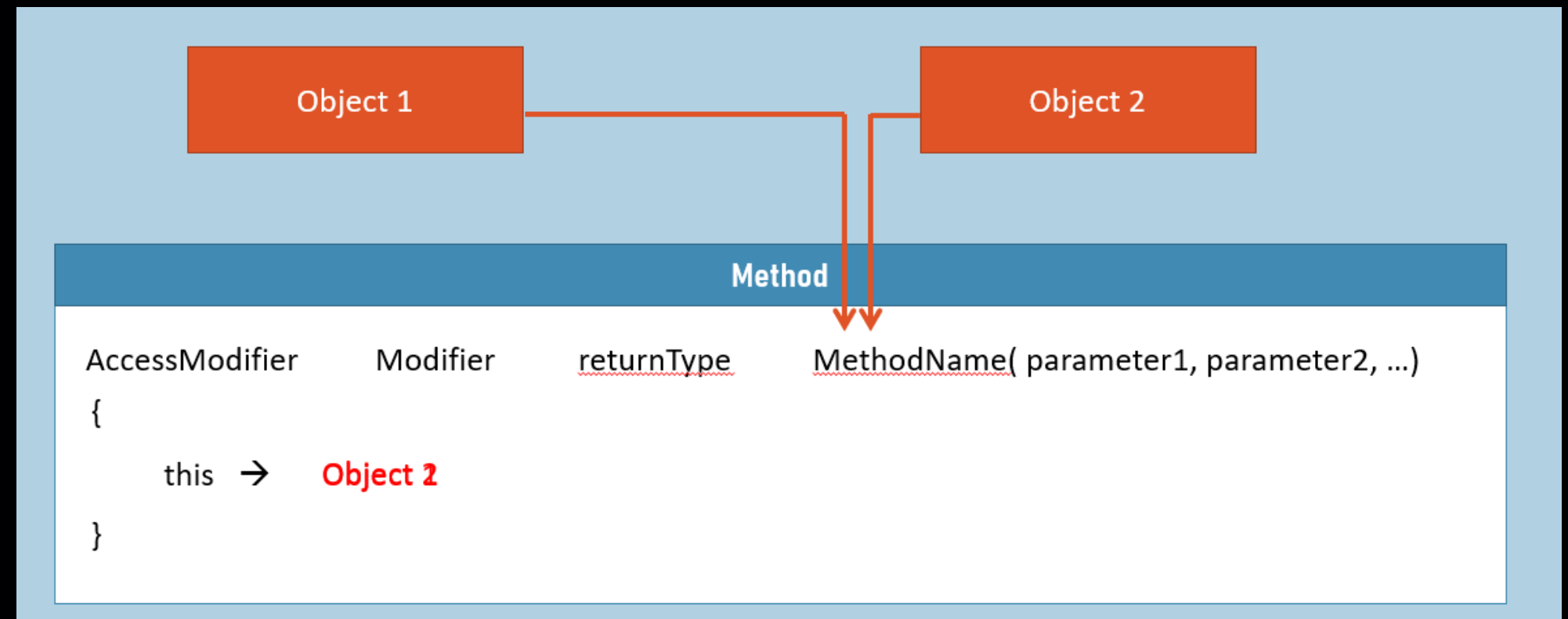
- The variables that are declared inside the method are called as "Local variables". Local variables can be used only within the same method.
- Local variables are stored in the same stack, just like parameters.
- The stack will be deleted at the end of method execution. So all local variables and parameters will be deleted.



METHODS

"this" keyword

- The "this" keyword refers to "current object", which method has invoked the method.
- The "this" keyword is available only within the "instance methods".



METHODS

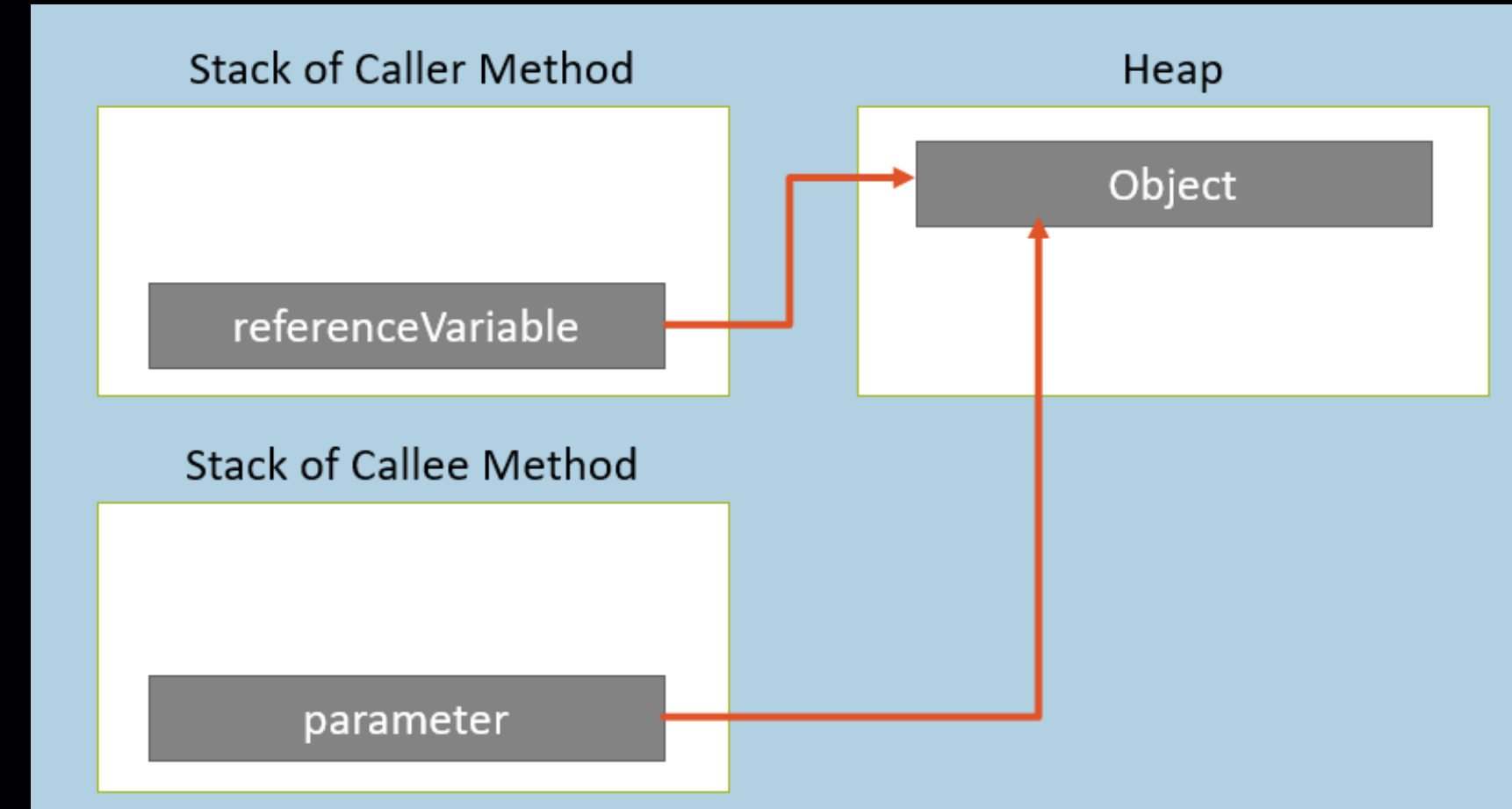
Instance Methods (vs) Static Methods

Topic	Instance Fields	Static Fields
Association	Associated with Objects	Associated with class.
Manipulates	Manipulates instance fields	Manipulates static fields.
Declaration	Declared without "static" keyword. Syntax: returnType methodName() { }	Declared with "static" keyword. Syntax: static returnType methodName() { }
Accessible with	Accessible with object (through reference variable).	Accessible with class name only (not with object).
Can access (fields)	Can access both instance fields and static fields	Can't access instance fields; but can access static fields only.
Can access (methods)	Can access both instance methods and static methods.	Can't access instance methods; but can access static methods only.
"this" keyword:	Can use "this" keyword, as there must be "current object" to call instance method.	Can't use "this" keyword, as there is NO "current object" while calling instance methods.

METHODS

Reference Variables as Arguments

- If you pass "reference variable" as argument, the reference (address) of object will be passed to the method.
- The parameter's data type will be the class name.
- If you make any changes to object in the method, the same will be affected automatically in the caller method, as you are accessing the same object.



METHODS

Default Arguments

- Default value of the parameter of a method.
- If you don't pass value to the parameter, the default value gets assigned to the parameter.
- To void bothering to pass value to the parameter; instead, take some default value into the parameter automatically, if the method caller has not supplied value to the parameter.

METHODS

Named Arguments

- Supply value to the parameter, based on parameter name.
- Syntax: **parametername: value**
- You can change order of parameters, while passing arguments.
- Parameter names are expressive (understandable) at method-calling time.
- Calling a method:
MethodName(ParameterName : value,
ParameterName : value);

METHODS

Method Overloading

- Writing multiple methods with the same name in the same class, with different parameters.
- Caller would have several options, while calling a method.
- Difference between parameters of all the methods that have same name, is MUST.

```
1 | MethodName( )  
2 | MethodName( int )  
3 | MethodName( string )  
4 | MethodName( int, string )  
5 | MethodName( string , int)  
6 | MethodName( string , string, int)  
7 | etc.
```

METHODS

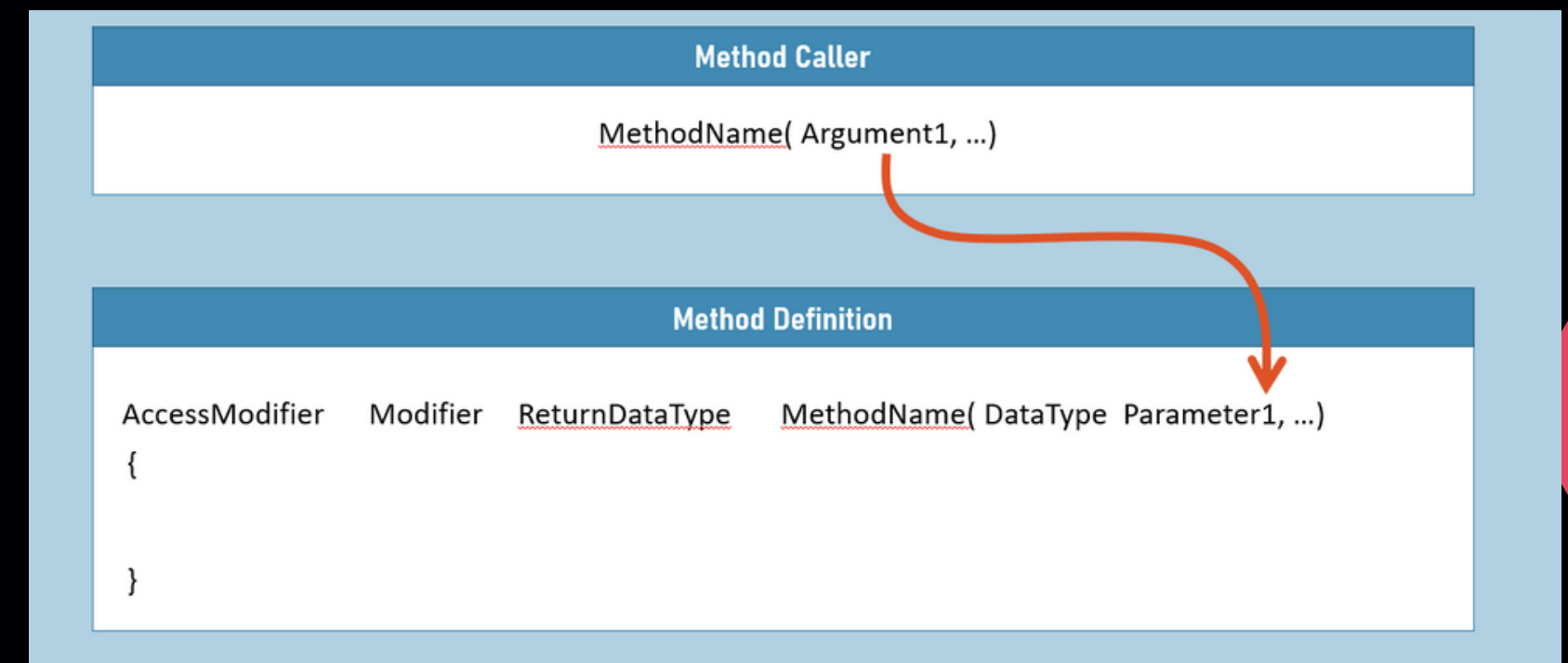
Parameter Modifiers

Specifies how the parameter receives a value.

1. Default [No keyword]
2. ref
3. out
4. in
5. params

Parameter Modifiers (default)

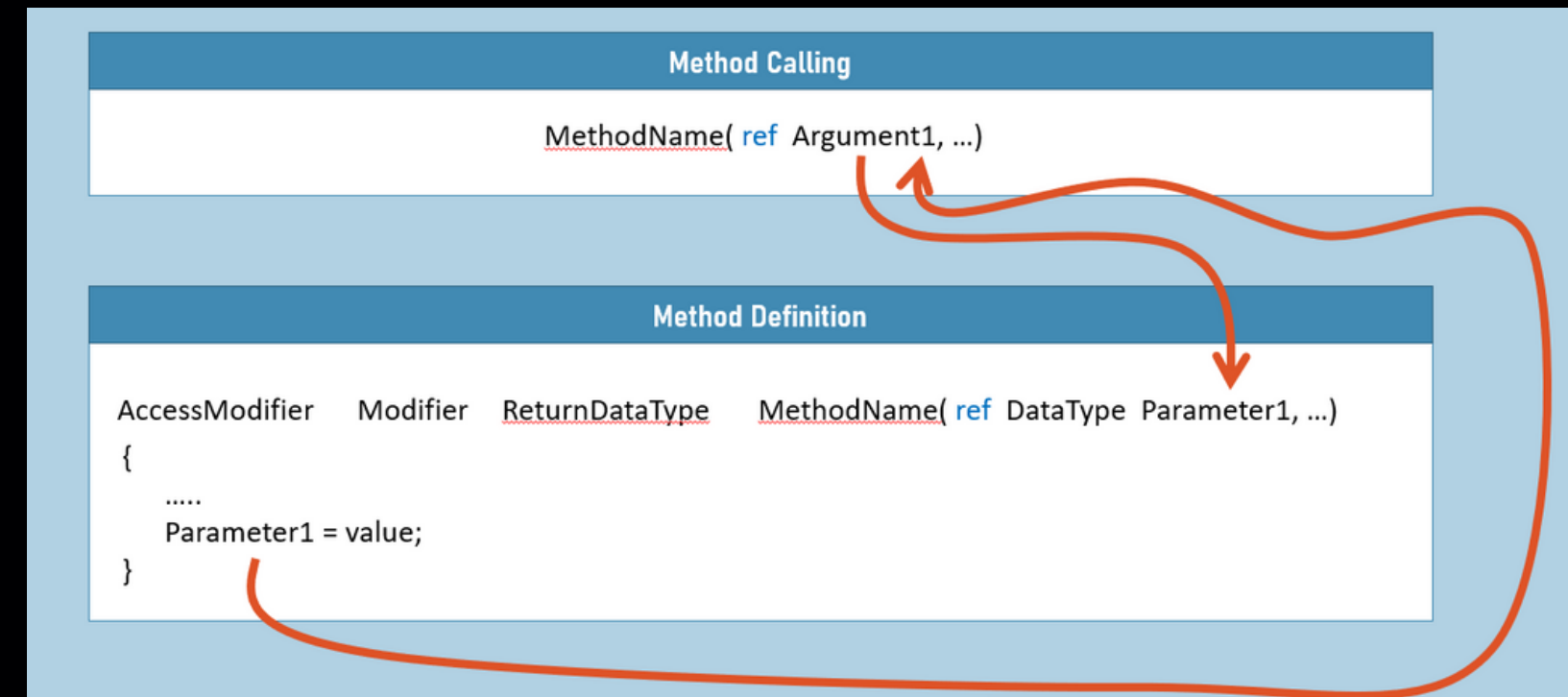
The "Argument" will be assigned into the "Parameter" but not reverse.



METHODS

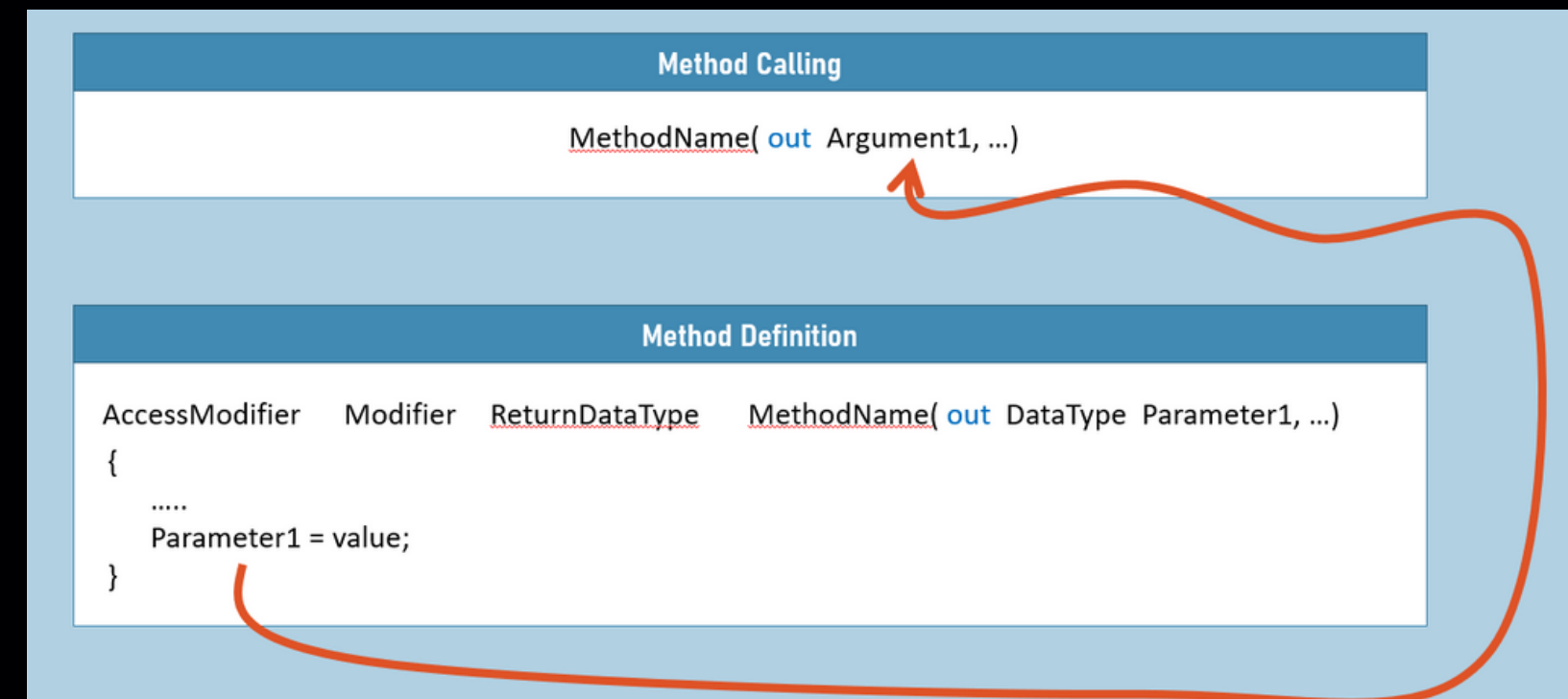
Parameter Modifiers (ref)

- The "Argument" will be assigned into the "Parameter" and vice versa.
- The Argument must be a variable and must be pre-initialized.



Parameter Modifiers (out)

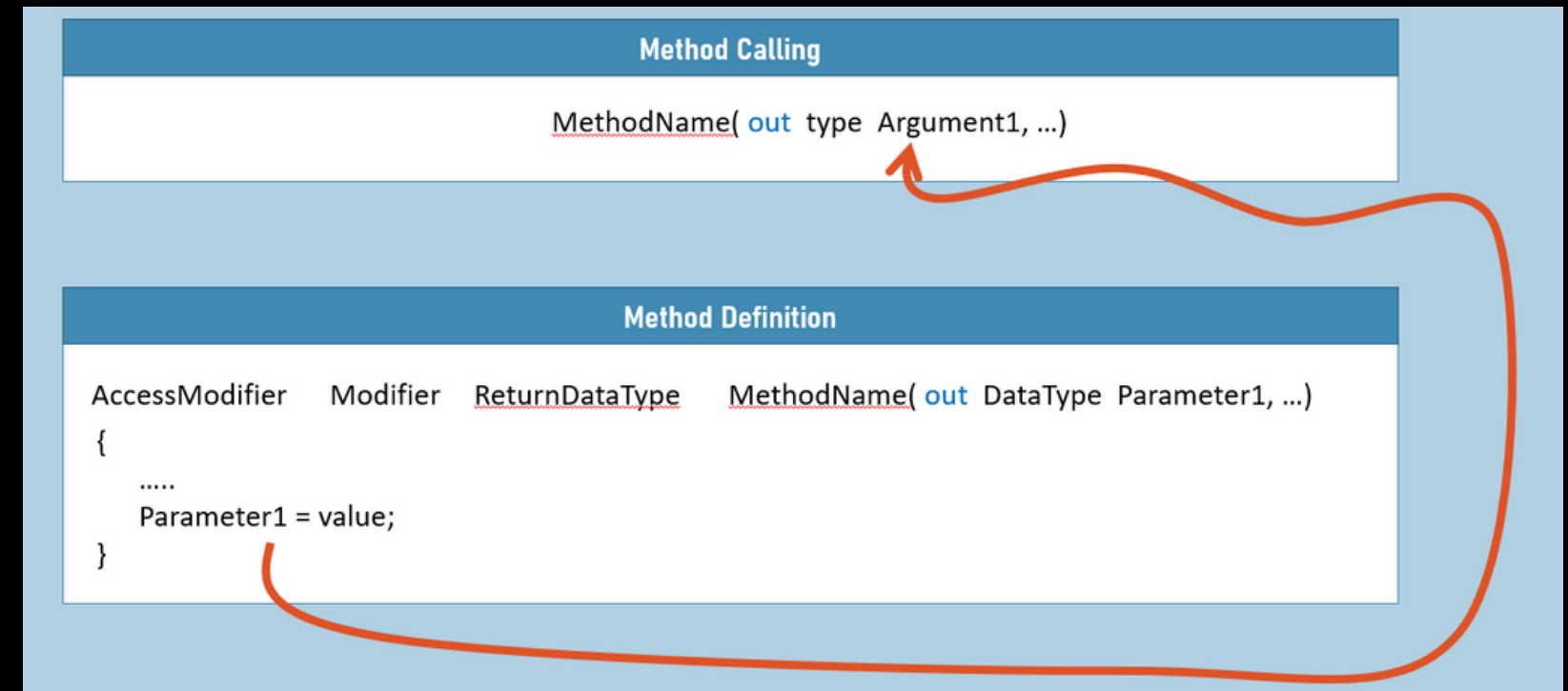
- The "Argument" will not be assigned into the "Parameter" but only reverse.
- The Argument must be a variable; The Argument can be un-initialized.



METHODS

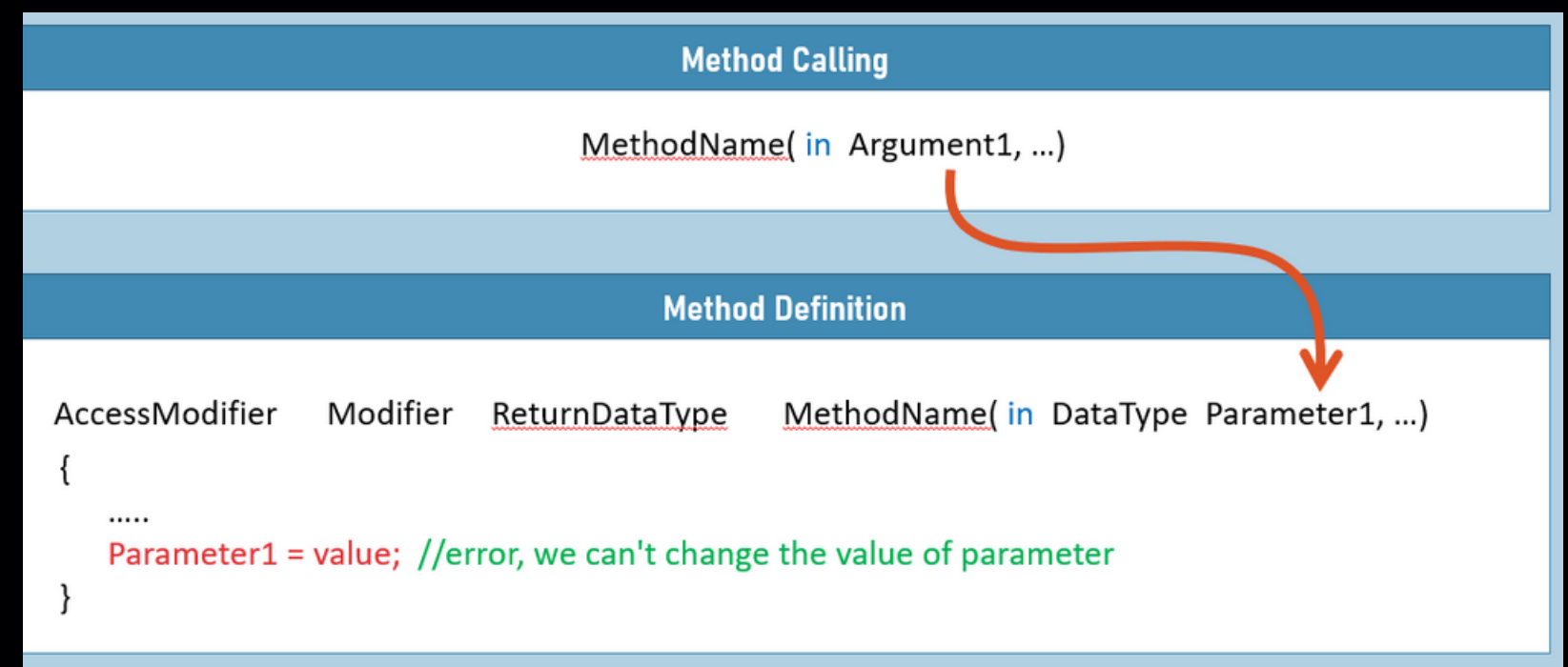
'out' variable declaration

- You can declare out variable directly while calling the method with 'out' parameter.
- New feature in C# 7.0.



Parameter Modifiers (in)

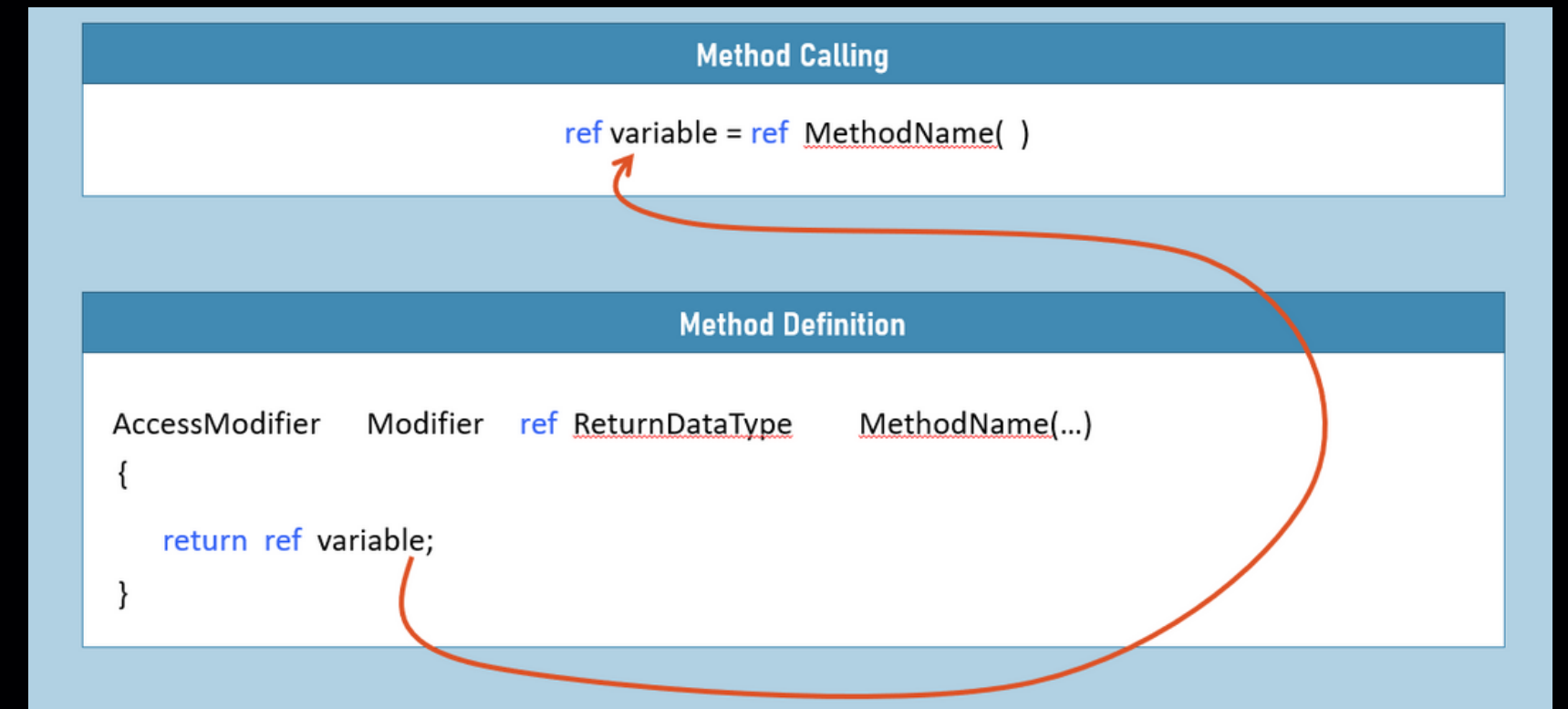
- The "Argument" will be assigned into the "Parameter", but the parameter becomes read-only.
- We can't modify the value of parameter in the method; if you try to change, compile-time error will be shown.
- New feature of C# 7.2.



METHODS

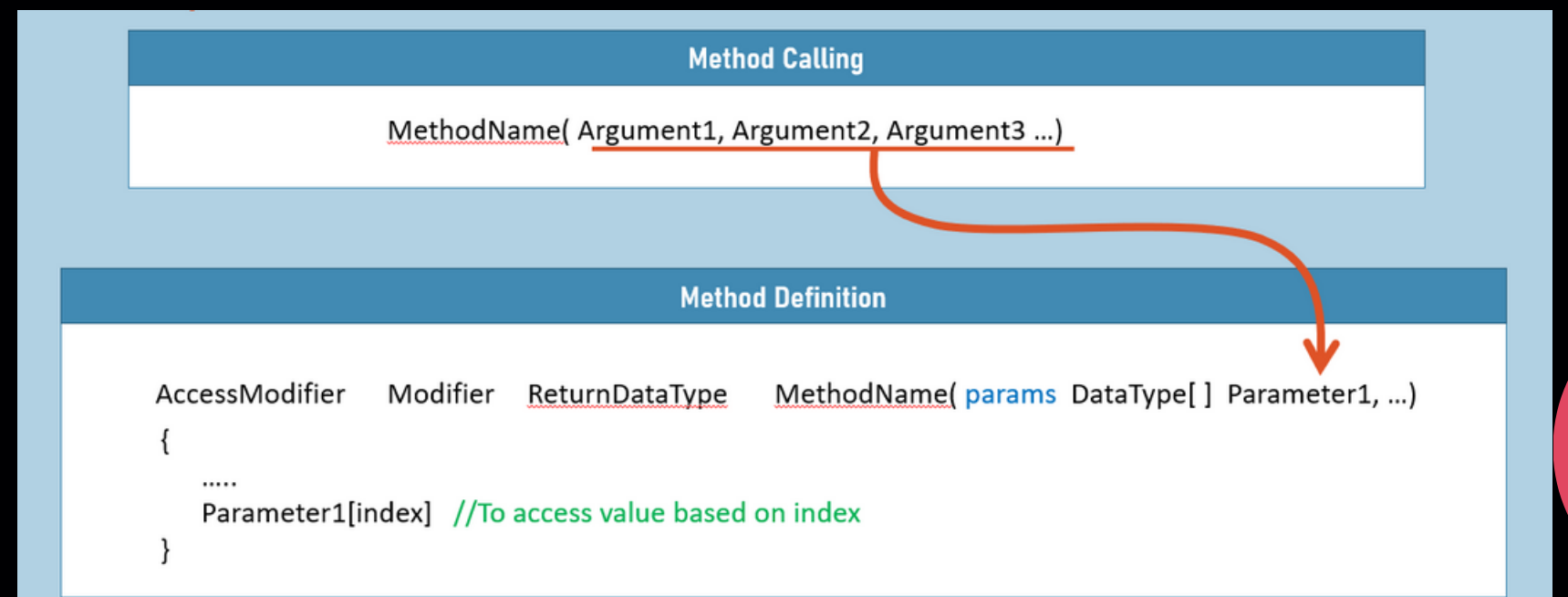
ref returns

- The reference of return variable will be assigned to receiving variable.
- New feature in C# 7.3.



Parameter Modifiers (params)

- All the set of arguments will be at-a-time received as an array into the parameter.
- The "params" parameter modifier can be used only for the last parameter of the method; and can be used only once for one method.



METHODS

Local Functions

- "Local functions" are functions, to do some small process, which is written inside a method.
- Local functions are not part of the class; they can't be called directly through reference variable.
- Local functions don't support "access modifiers" and "modifiers".
- Local functions support parameters, return.

Local Function [Inside a method]

```
public void MethodName( param1, param2, ... )  
{  
    LocalFunctionName( ); //Calling the Local Function  
    ReturnDataType LocalFunctionName( param1, param2, ... )  
    {  
        //Local Function Body Here  
    }  
}
```

METHODS

Static Local Functions

- "Static Local functions" are functions, same as normal "Local Functions".
- Only the difference is, static local functions can't access local variables or parameters of containing method.
- This is to avoid accidental access of local variables or parameters of containing method, inside the local function.

Local Function [Inside a method]

```
public void MethodName( param1, param2, ... )  
{  
    LocalFunctionName( ); //Calling the Local Function  
      
    static ReturnDataType LocalFunctionName( param1, param2, ... )  
    {  
        //We can't access local variables or parameters of containing method.  
    }  
}
```


METHODS

Key Points to Remember

- Method is a part of class, that contains collection of statements to do some process.
- Access modifiers of Methods: private, protected, private protected, internal, protected internal, public.
- Modifiers of Methods: static, virtual, abstract, override, new, partial, sealed
- For each method call, a new stack will be created; all local variables and parameters of the method will be stored in that stack; will be deleted automatically at the end of method execution.
- In instance methods, the 'this' keyword refers to 'current object, based on which the method is called'.
- Instance methods can access & manipulate instance fields & static fields; Static methods can access only static fields.
- But static method can create an object for the class; then access instance fields through that object.
- Using named arguments , you can change order of parameters while calling the method.
- Method Overloading is 'writing multiple methods with same name in the same class with different set of parameters'.
- The 'ref' parameter is used to receive value into the method and also return some value back to the method caller; The 'out' parameter is only used to return value back to the method caller; but not for receiving value into the method.