# Reference and value types

# Agenda

A strongly typed language

Types overview

Value types

The stack

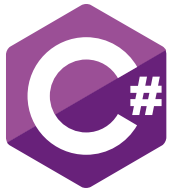
Reference types

The heap

Built-in types

Custom types

# A strongly typed language



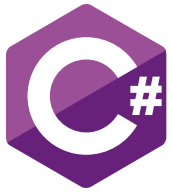
# Strong typing

Every variable has a type

The type of a variable determines where and how it is stored

Types are checked at compile time

# Types overview



Reference and value types

# C# types

## Value types

Built-in value types

Enumerations

Structures

## Reference types

Built-in reference types

Classes

Records

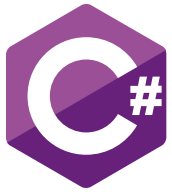
Interfaces

Arrays

Delegates

Strings

# Value types



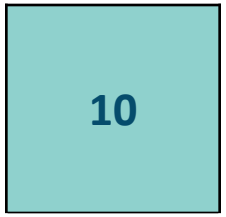
# Value types

Directly contains their value

Stored on the stack

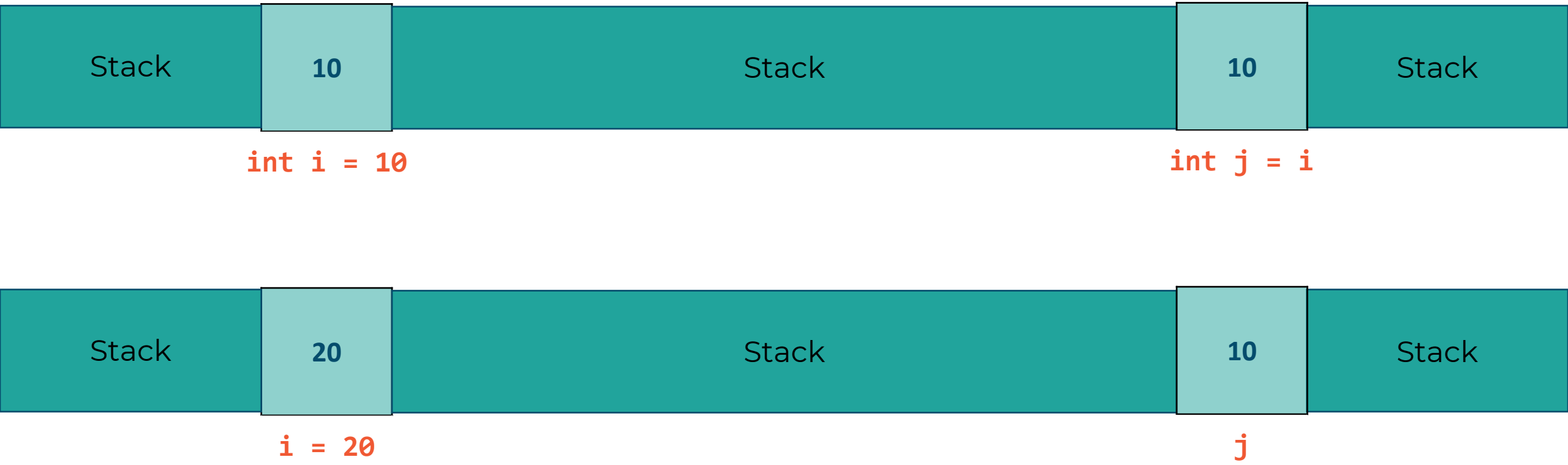
Cannot be null

# Value type




**int i = 10**

# Value type copy



# Value types

Built-in types except string and object

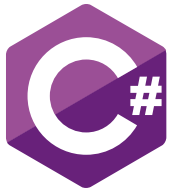
Structures

Enumerations

# Demo

Value types

# What is the stack ?



# The stack

LIFO

Memory that is responsible for keeping track of the context of the execution (parameters, method calls, variables)

Getting a value or writing a value to the stack is fast

# Value type/Stack

**Main method**

```
static void Main(string[] args)
{
    int i = 0;
    bool b = true;
    Method1();
}
```

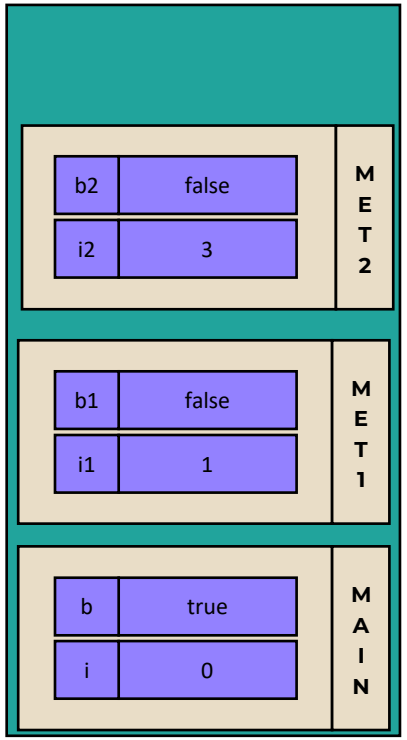
**Method 1**

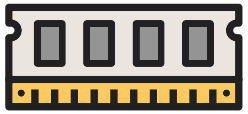
```
static void Method1()
{
    int i1 = 1;
    bool b1 = false;
    Method2();
}
```

**Method 2**

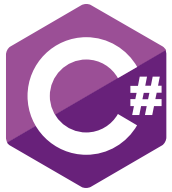
```
static void Method2()
{
    int i2 = 3;
    bool b2 = false;
}
```

Stack



# Reference types



# Reference types

A reference to their value is stored on the stack

The object data is stored on the heap

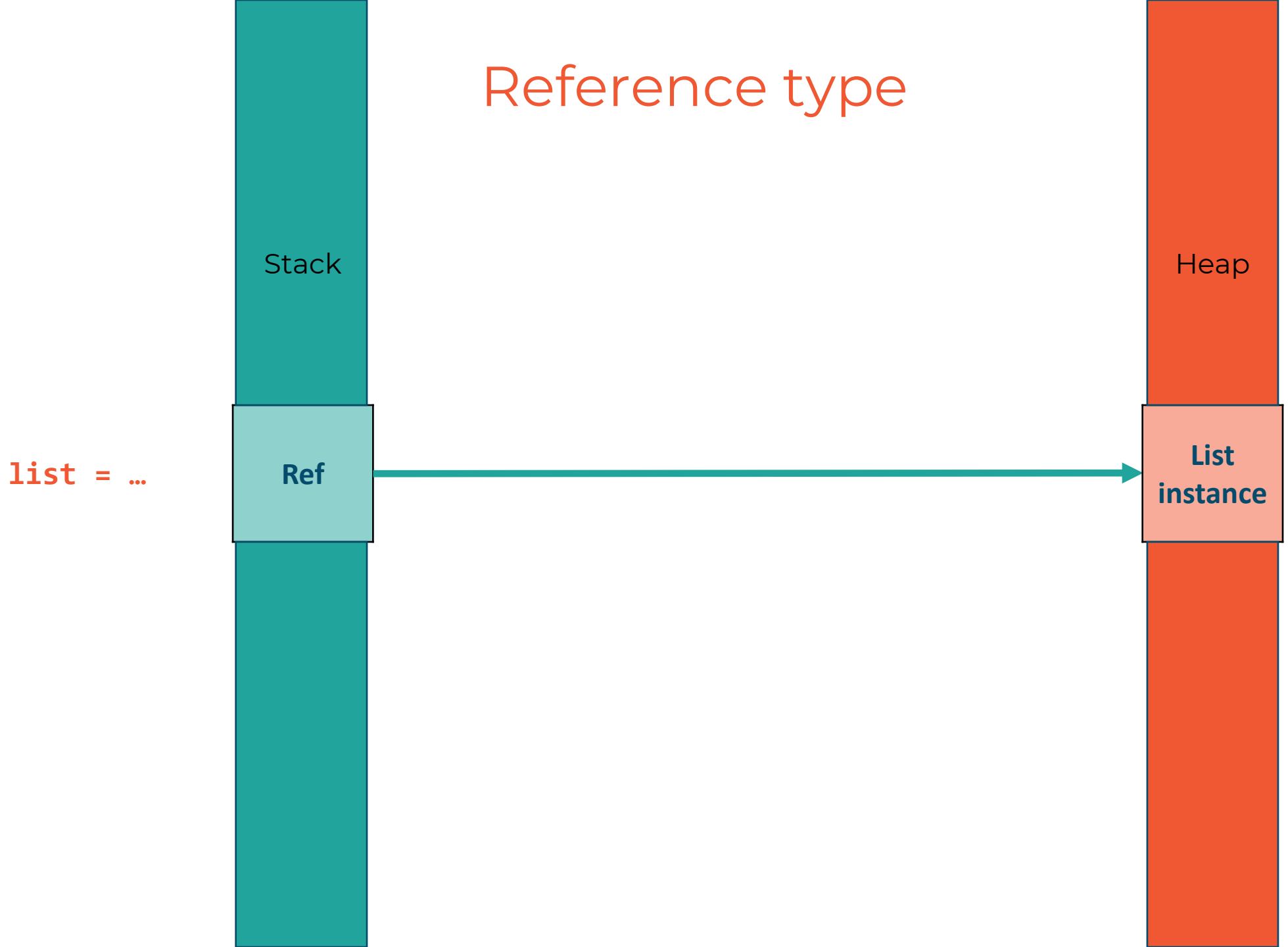
Can be null

# Null reference

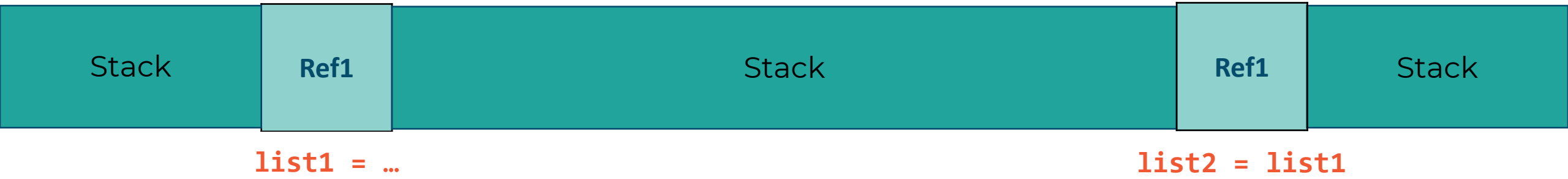
A null reference is a reference that does not refer to any object

It is the default value for reference type variables

# Reference type



# Reference type copy




`list1 = …`

`list2 = list1`


`List1.Add(…)`

`list2`

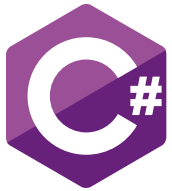
# Reference types

- Class
- Record
- Interface
- Array
- Delegate
- String
- Object

# Demo

Reference types

# What is the heap ?



# The heap

Large section of memory

Objects are persistent

Memory is freed up by the Garbage Collector

Getting a value or writing a value to the heap is slower but the heap is not limited in memory

# Reference type/Heap

**Main method**

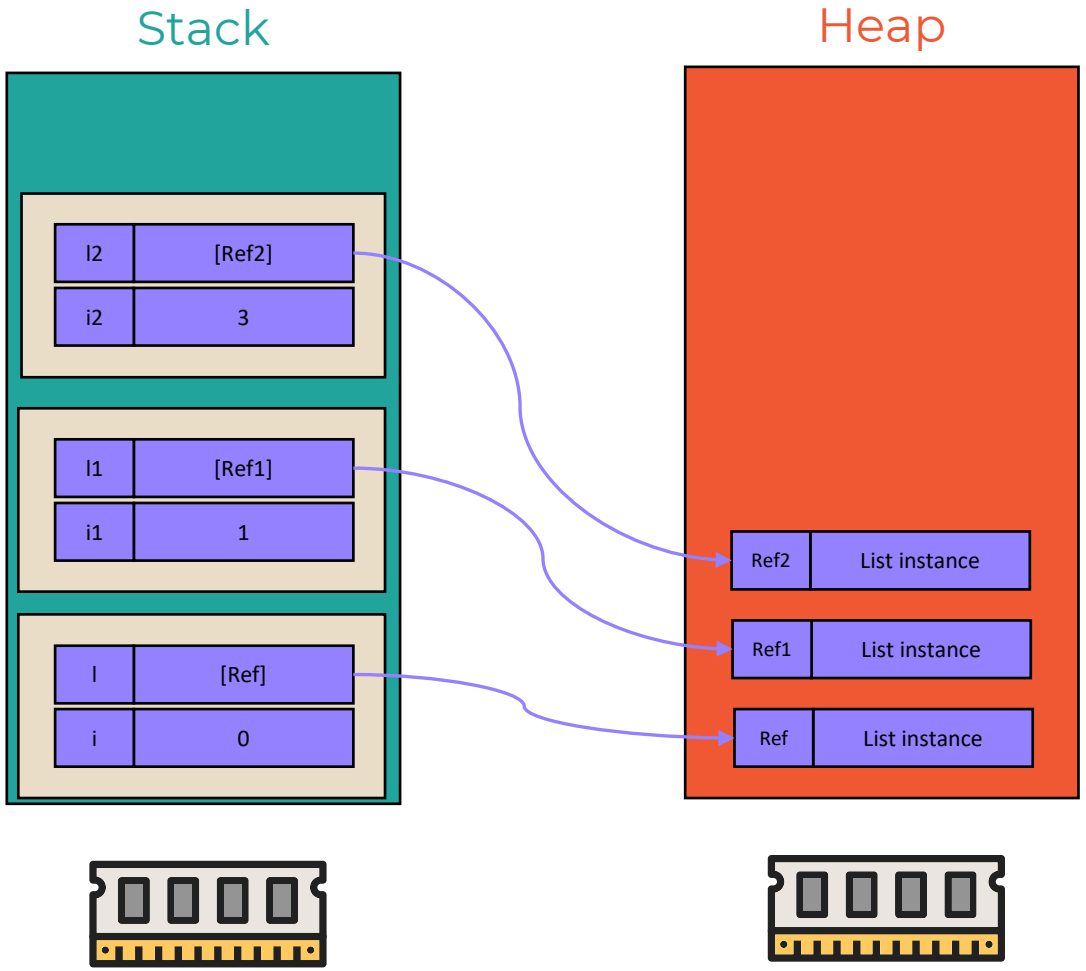
```
static void Main(string[] args)
{
    int i = 0;
    var l = new List<string>();
    Method1();
}
```

**Method 1**

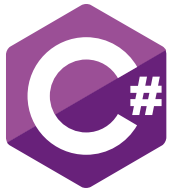
```
static void Method1()
{
    int i1 = 1;
    var l1 = new List<string>();
    Method2();
}
```

**Method 2**

```
static void Method2()
{
    int i2 = 3;
    var l2 = new List<string>();
}
```



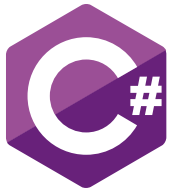
# Built-in types



# Built-in types

- Boolean (`bool`)
- Integers (`sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`)
- Floating-point types (`float`, `double`, `decimal`)
- Character (`char`)
- Strings of characters (`string`)
- Object (`object`)

# Custom types



# Custom types

- Enumerations
- Structures
- Interfaces
- Classes
- Records
- Delegates

# Summary

C# is strongly typed, every variable has a type

There are two categories of types : value types and reference types

Value types contain their value and are stored on the stack

Reference types contain a reference to the memory address where the instance is stored on the heap

A reference type can be null but not a value type