Understanding Value Types and Reference Types

Gill Cleeren

CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview

Value types and reference types

Passing parameters to methods

Strings are reference types too

Creating custom types

- Enumeration
- Struct

Value Types and Reference Types

Types in .NET and C#

Value types

Reference types

Value Types

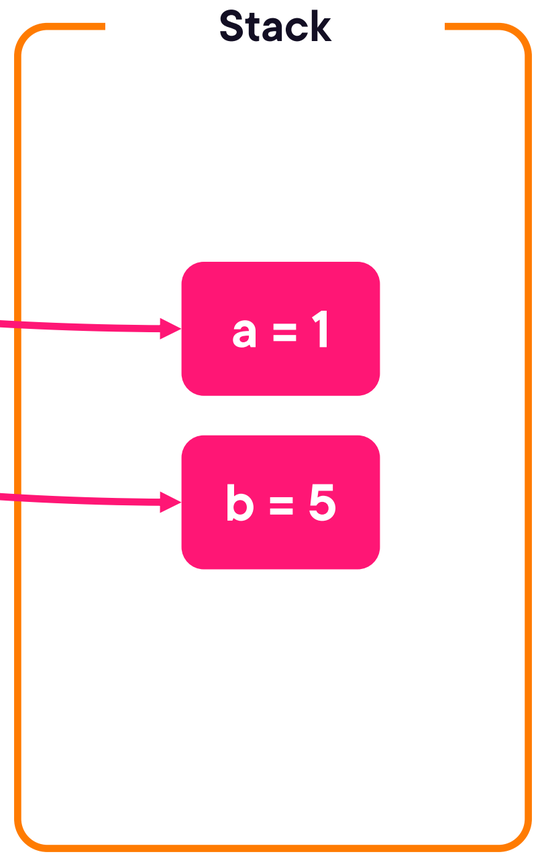
Int, float, double, char

Fixed size, allocated by compiler on stack

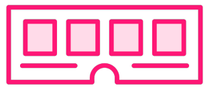
Value is copied to this memory location

Working with Value Types

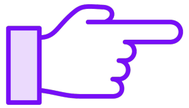
```
int a = 1;

int b = 5;
```



Reference Types



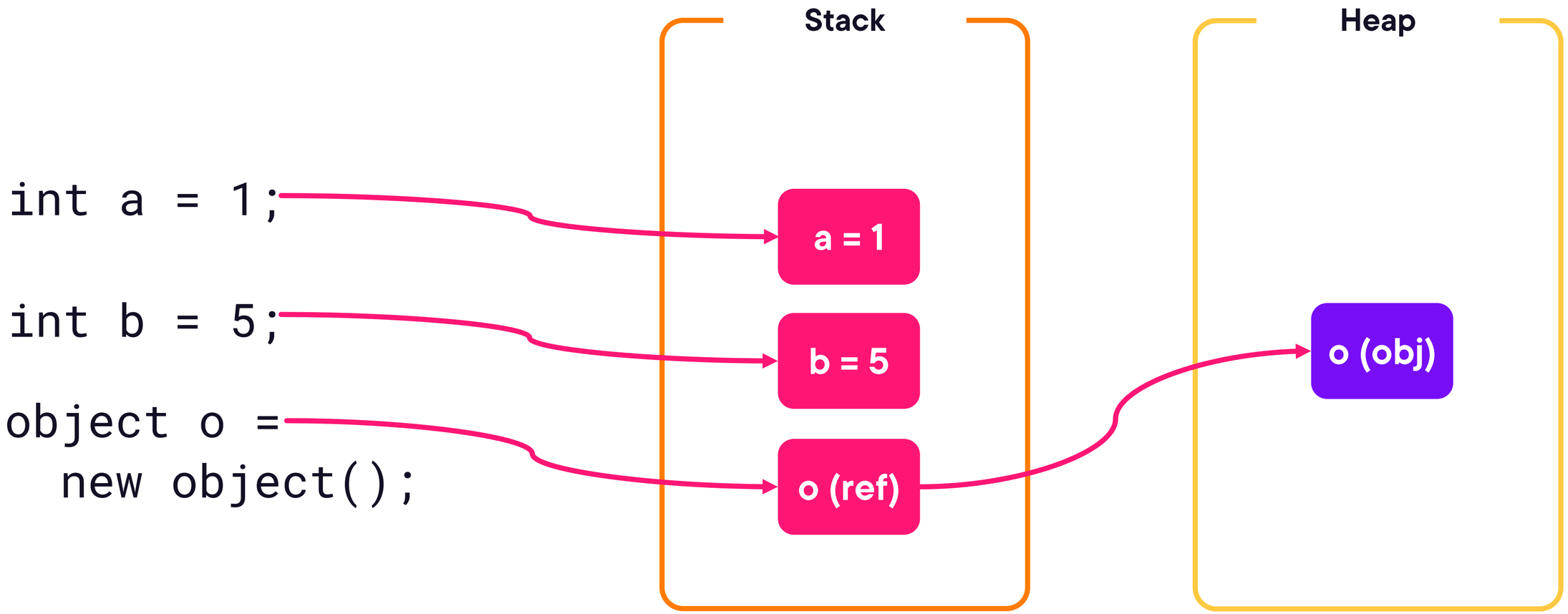
Allocated on heap



Stack contains just a pointer to the memory address

Classes are reference types

Working with Reference Types

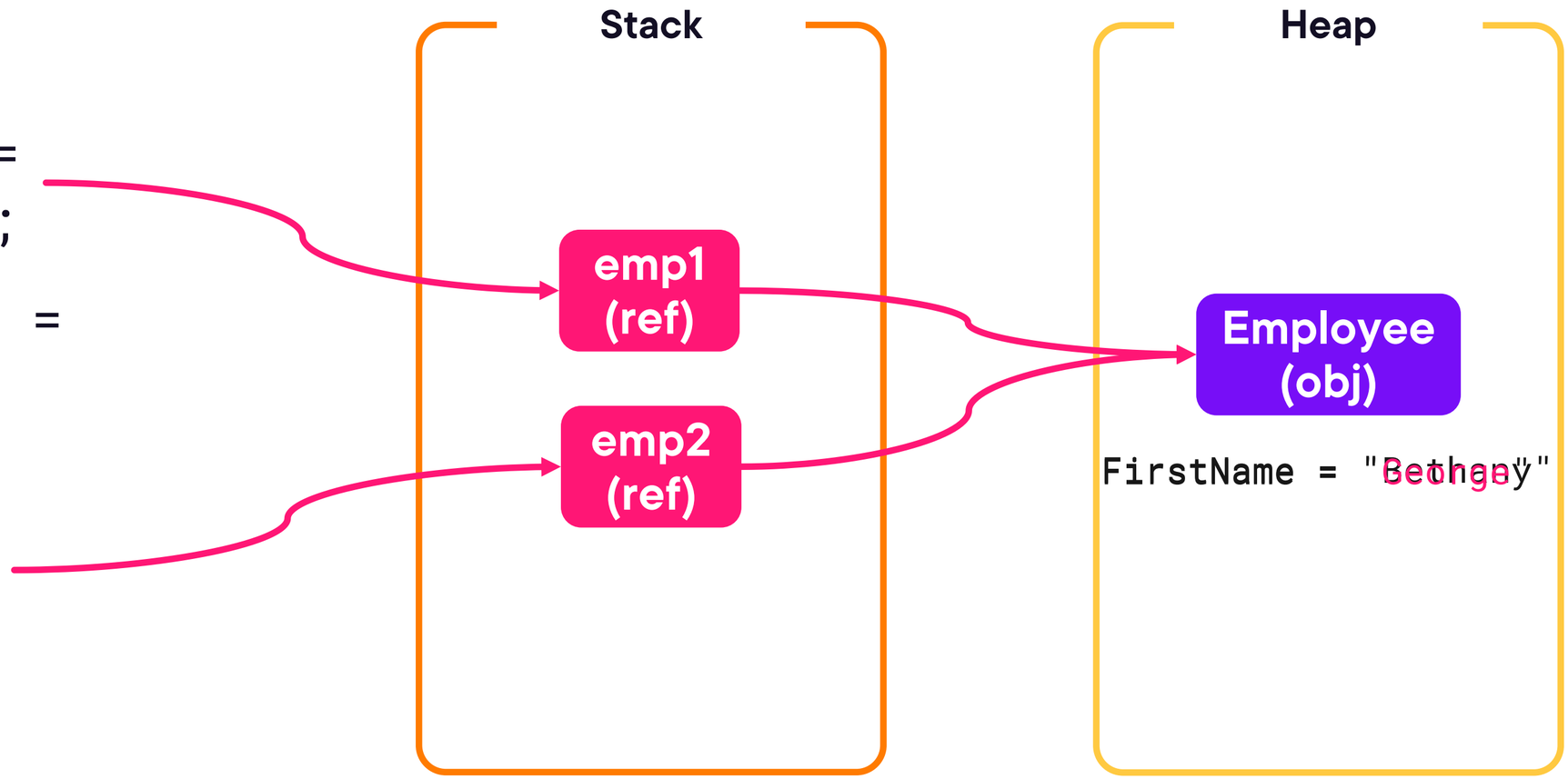


Understanding Classes Are Reference Types

```
Employee emp1 = new Employee();
emp1.firstName = "Bethany";

Employee emp2 = emp1;
emp2.firstName = "George";

string check = emp1.firstName;//check will be George
```

What Just Happened?

```
Employee emp1 =
new Employee();

emp1.FirstName =
"Bethany";


Employee emp2
= emp1;

emp2.FirstName
= "George";
```



Demo

Working with value types

Classes are reference types

Passing Data to Methods

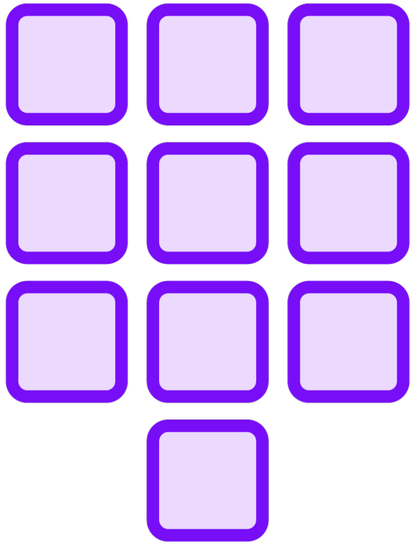
Passing Parameters

By value

Default if nothing else is specified

By reference

Require use of ref keyword on parameters



Passing parameters by value

- Default way of passing parameters
- A copy is created for the method
- Value in caller stays the same

Passing Parameters by Value

```
int a = 33;
int b = 44;

AddTwoNumbers(a, b);
```

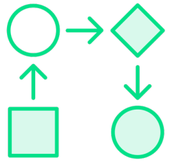
33 44

```
public int AddTwoNumbers(int a, int b)
{
        b += 10;
        int sum = a + b;
        return sum;
}
```

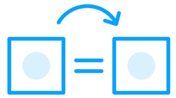
Passing Parameters by Reference

A reference to the value is passed

No copy is made



Changes made in method affect original values



ref keyword is used

Passing Parameters by Reference

```
int a = 33;
int b = 44;

AddTwoNumbers(a, ref b);
```

```
public int AddTwoNumbers(int a, ref int b)
{
        b += 10;
        int sum = a + b;
        return sum;
}
```

Demo

Passing parameters by value

Using ref to pass parameters by reference

```
public static int AddTwoNumbers(int a, out int b, out int c)
{
    b = 10;
    int sum = a + b;
    c = sum / 10;
    return sum;
}
```

Using the out Keyword

Out values don’t need to be initialized
Multiple values can be returned

Demo

Using the out keyword

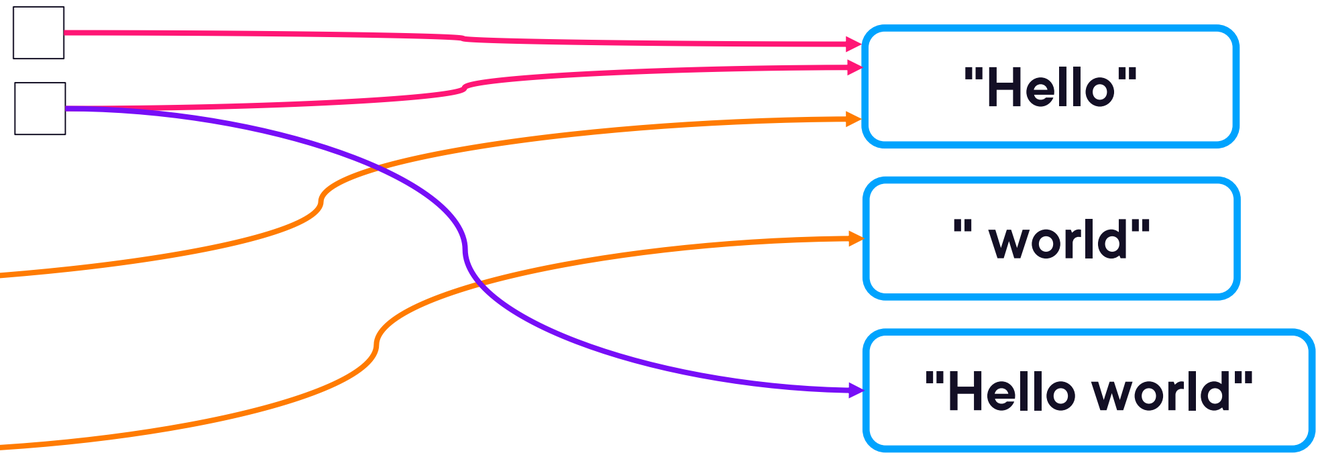
Strings Are Reference Types Too

```
string a = "Hello";
string b;
b = a;
b += " world";
Console.WriteLine(a);//Output: Hello
```

Let’s Append to an Existing String

Strings Are Reference Types

```
string a;
string b;

a = "Hello";
b = a;
b += " world";
```




```
Console.WriteLine("a = {0}", a);//a = Hello
Console.WriteLine("b = {0}", b);//b = Hello world
```

Strings are immutable.

String immutability can have a performance impact!

Loop actions or many concatenate actions can cause high memory use!

```
StringBuilder stringBuilder = new StringBuilder();

stringBuilder.Append("Employee list");

stringBuilder.AppendLine("Bethany Smith");

stringBuilder.AppendLine("George Jones");

stringBuilder.AppendLine("Gill Cleeren");

string list = stringBuilder.ToString();
```

Introducing the StringBuilder Class

Demo

Understanding string immutability

Using the StringBuilder

Working with Custom Types

Types in .NET

Class

Enumeration

Struct

Interface

Delegate

Types in .NET and C#

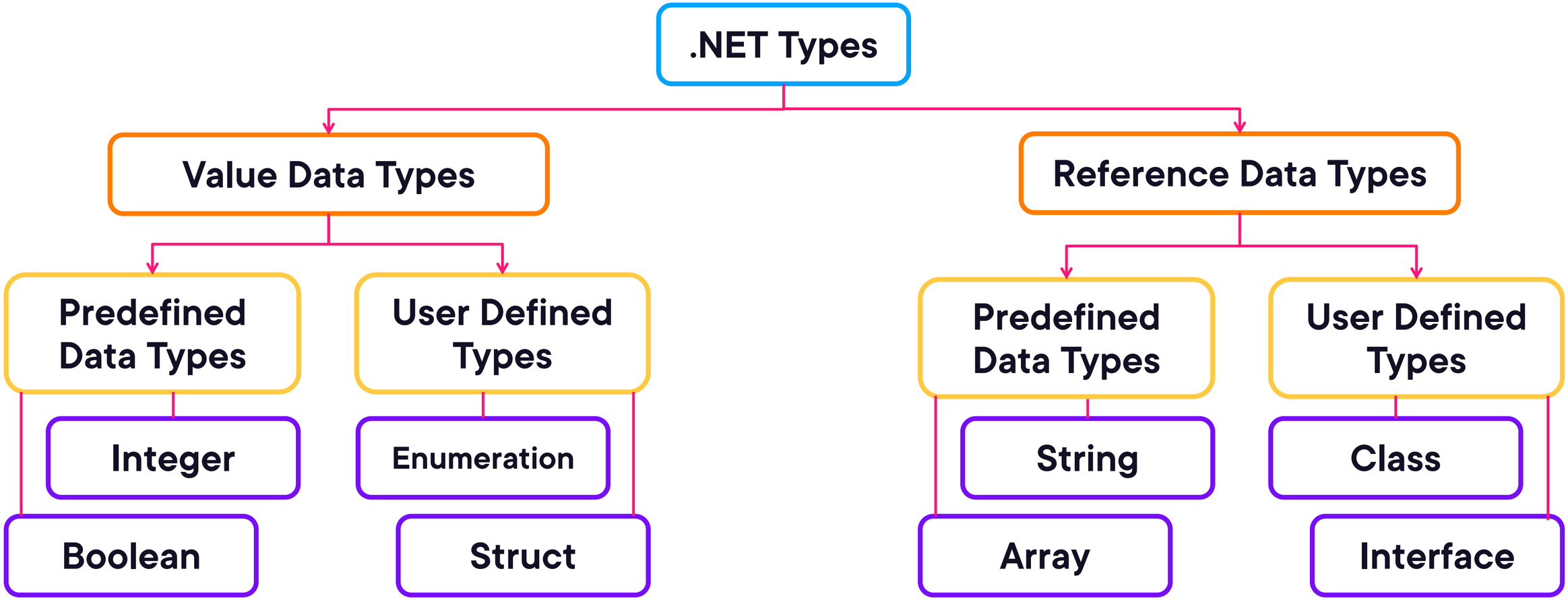
Value types

Enumerations & Structs

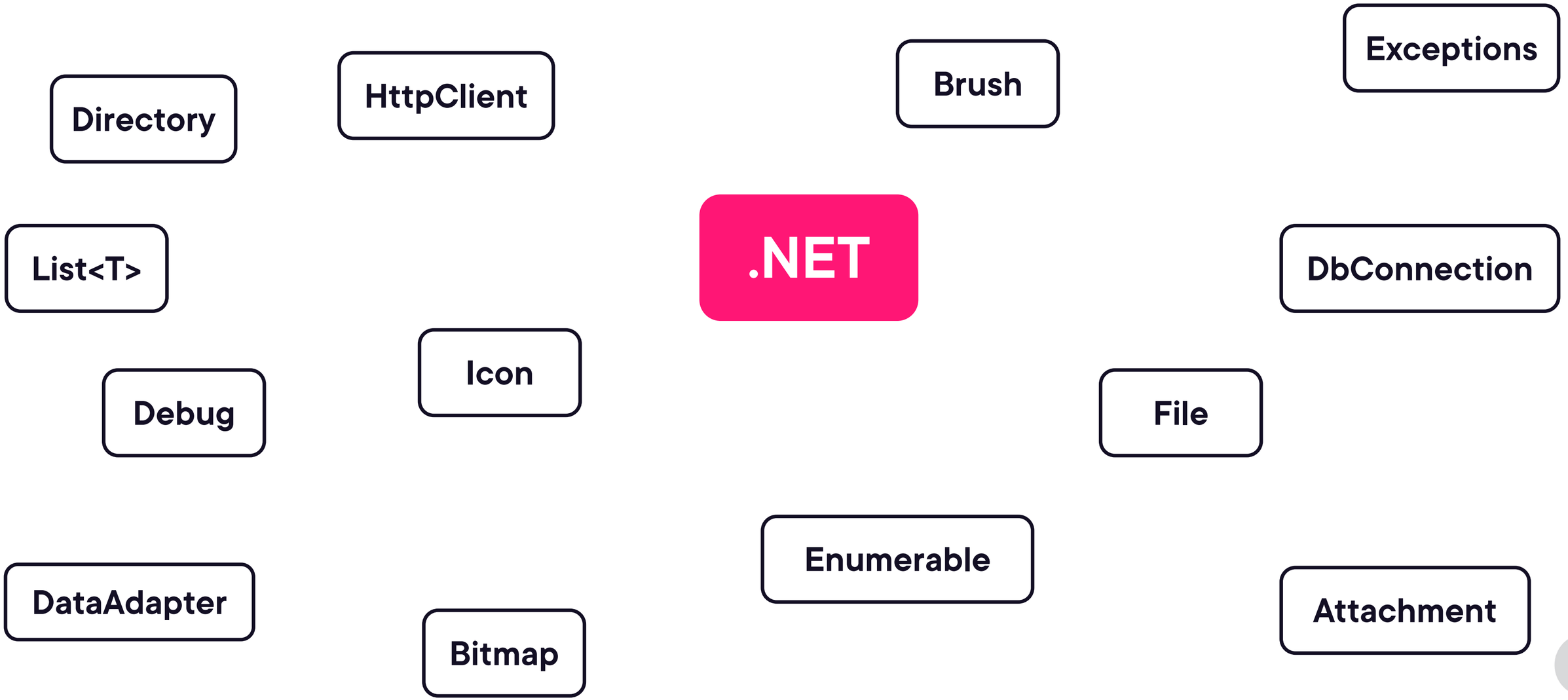
Reference types

Classes, Interfaces & Delegates

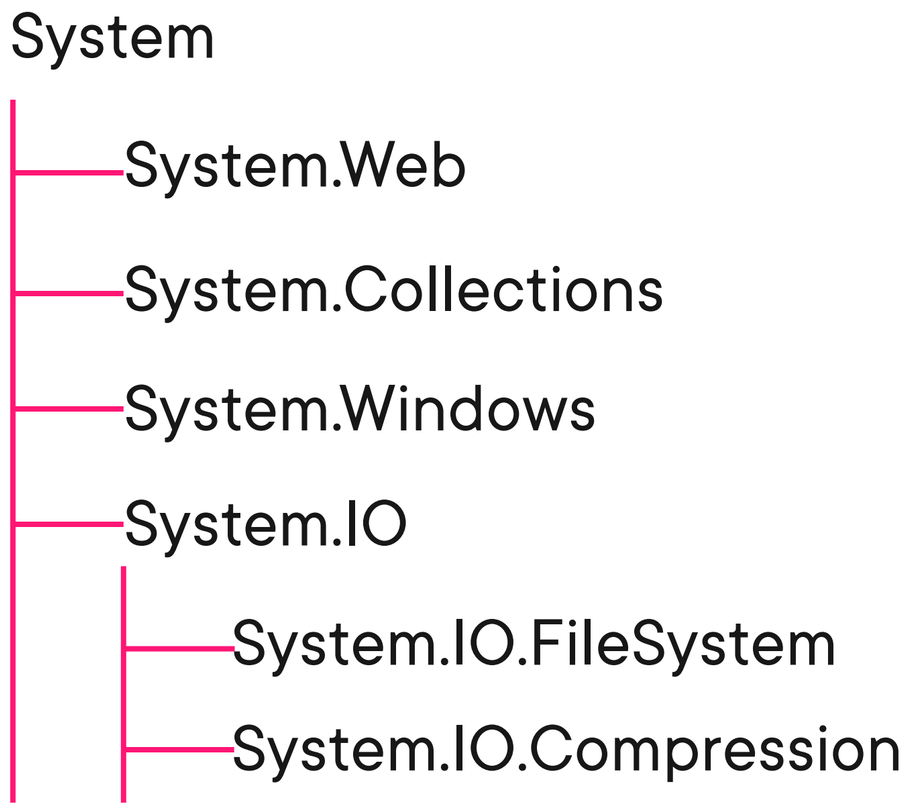
Types in .NET and C#



Custom Types in .NET



Organizing Types in Namespaces



```
using System;
using System.IO;
FileInfo fileInfo1
   = new FileInfo(@"D:\MyFile.txt"); //available through System.IO namespace
```

The using Keyword

A using statement only brings the types within the specified namespace, not the ones in nested namespaces

Global Usings

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Demo

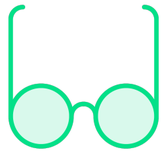
Browsing for existing types

Using a custom type

Understanding global using statements

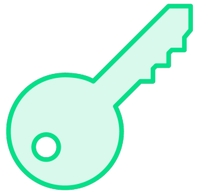
Creating Enumerations

Using an Enumeration in C#



Named constants for improved readability

Value type



Uses enum keyword

```
enum EmployeeType
{
     Sales,
     Manager,
     Research,
     StoreManager
}
```

Creating an Enumeration

```
enum EmployeeType
{
     Sales,//0
     Manager,//1
     Research,//2
     StoreManager//3
}
```

Default Values for Enumerations

```
Console.WriteLine(EmployeeType.Sales);
```

Accessing Enum Values

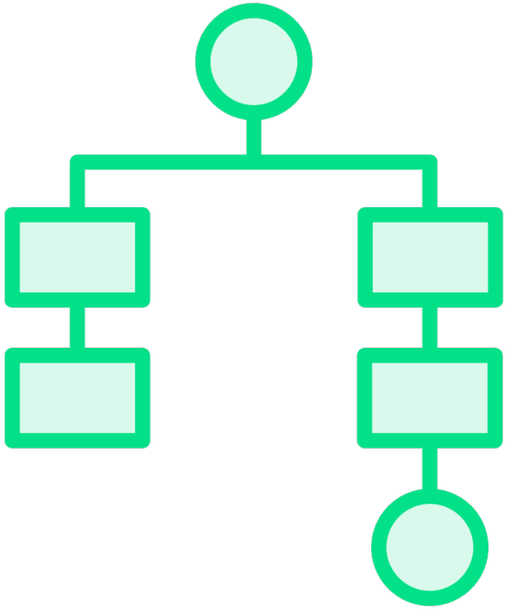
Demo

Creating an enumeration

Using the enumeration

Accessing the values

Working with Struct



Creating a Struct

- Represents a custom data structure
- Value type
- Can be new'ed
- Can contain methods and other members

Adding a Method to the Struct

```
struct WorkTask
{
    public string description;
    public int hours;

    public void PerformWorkTask()
    {
        //Code to perform work
    }
}
```

Demo

Creating a struct

Using the struct

Summary

Types can either be value types or reference types

.NET itself contains many types

Classes are the most commonly used way to create custom reference types

Enums and structs are custom value types

Up Next:

Doing more with custom types