Using Built-in C# Data Types

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren | www.xpirit.com/gill

Agenda

- Understanding types in C#
- Using built-in data types
- Working with DateTime
- Converting between types
- Implicit typing

Understanding Types in C#

C# is a strongly typed language

Every variable has a type

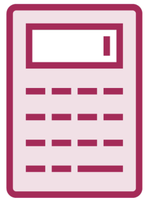
Used to store information

Expressions will return a value of a specified type

Using Data Types in C#

Size and location in memory

Data range



Supported operations

Return value of an expression

Data Types in C#

Predefined types

User-defined types

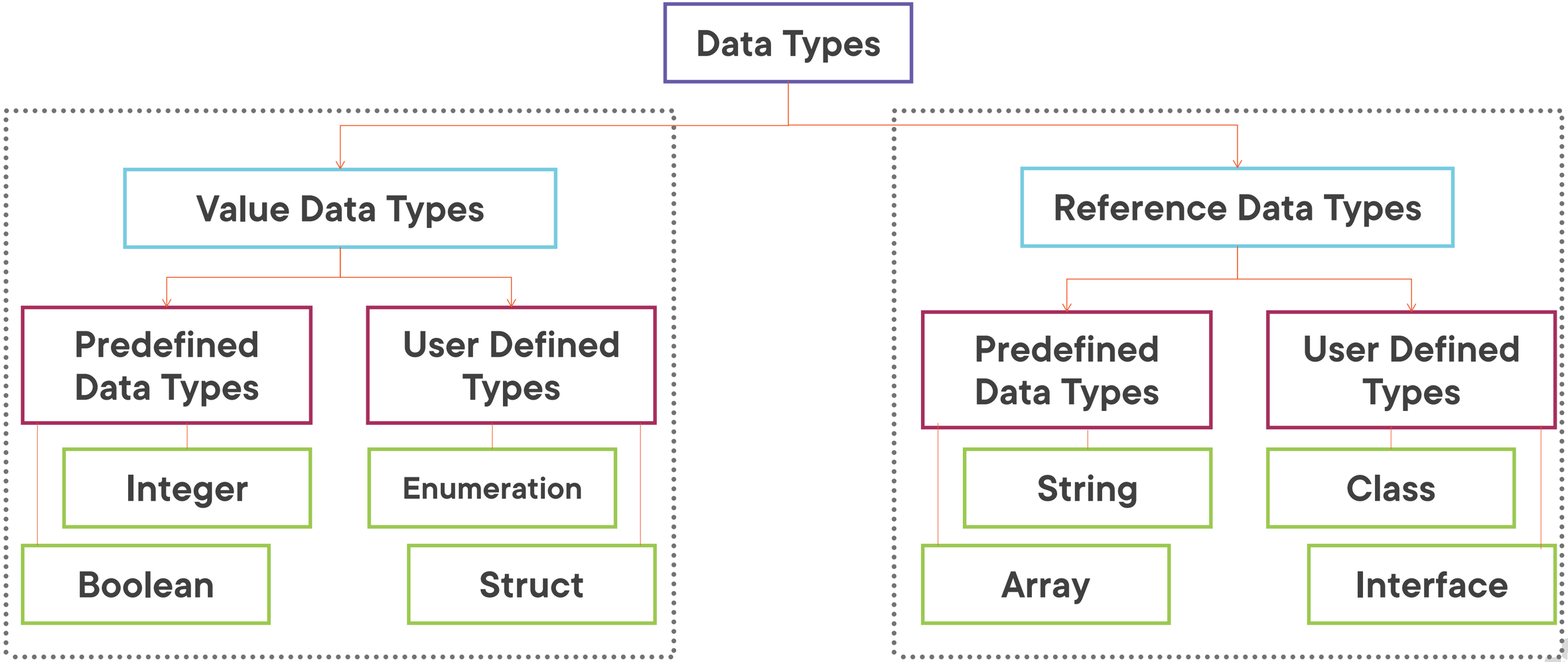
Predefined Data Types in C#

bool	int	float
double	decimal	char

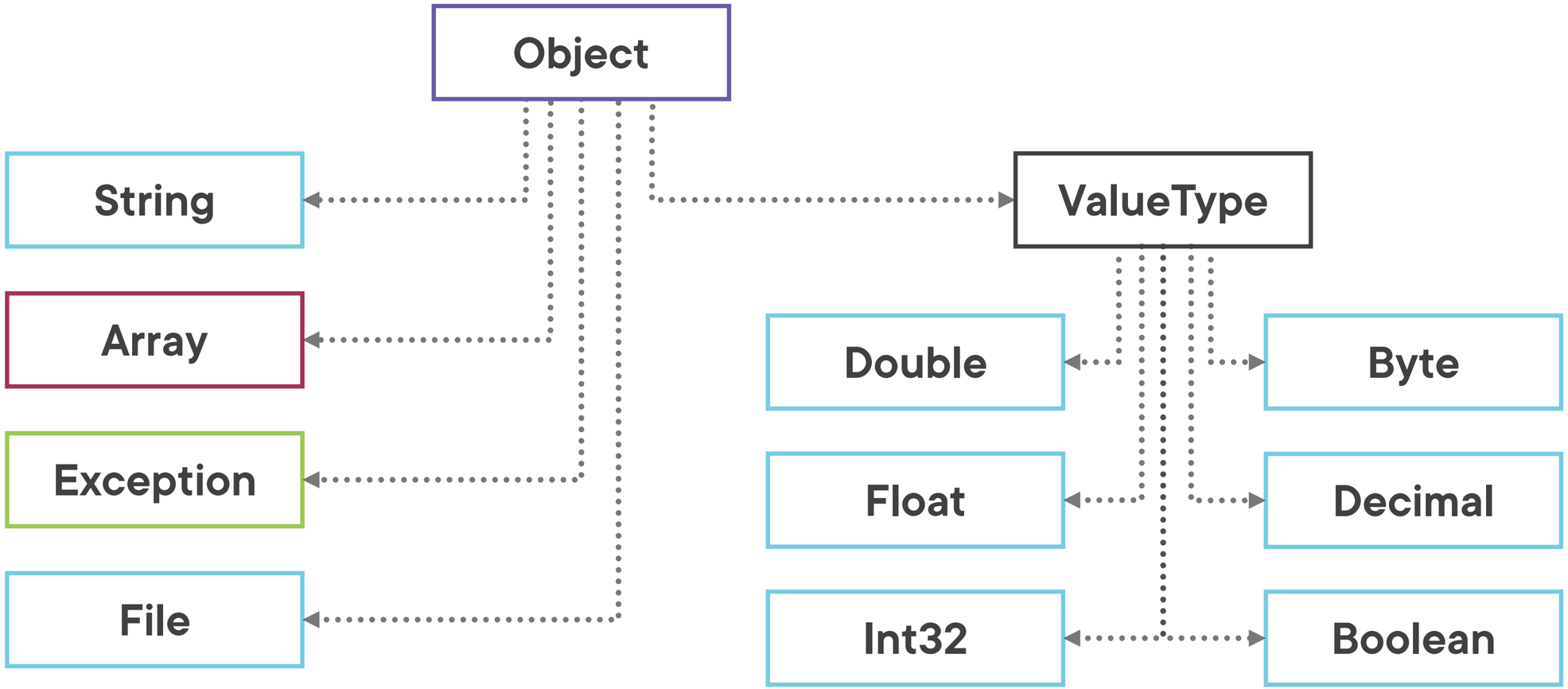
More Predefined Data Types

byte (sbyte)	short (ushort)
object	string

Value Types


Data Types
Value Data Types
Reference Data Types
Predefined Data Types
User Defined Types
Predefined Data Types
User Defined Types
Integer
Enumeration
String
Class
Boolean
Struct
Array
Interface

Peeking at the Full Hierarchy in C#



Looking at the Backing Types

Primitive type	Backing Type
int	Int32
float	Float
decimal	Decimal
bool	Boolean
byte	Byte

Using Built-in Data Types

Creating an Integer Value

```
int a = 2;
int b = a + 3;
```

Expression

Creating a Boolean Value

```
bool c = true;
```

C# Types Lead to Type Safety

```
int c = 3;
c = true;
```

Demo

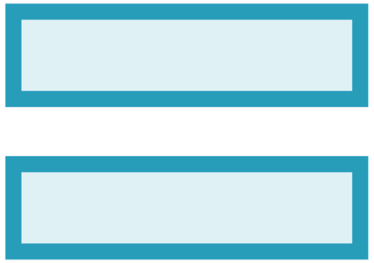
Working with primitive types

```
int a, b, c;

a = 3;

b = 10;

c = a++;

b = a + b * c;
```

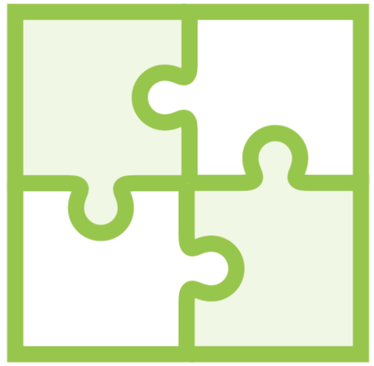
Expressions in C#

Operators in C#

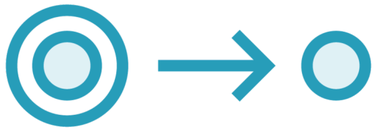
Arithmetic operators



Equality operators



Logical operators



Assignment operators

Using Arithmetic Operators

Operator	Example
+	a + b
-	a – 3
*	a * b * c
/	a / 10
++	a++
--	b--

Using Equality Operators

Operator	Example
==	if(a == b)
!=	if(a != b)
> or <	if(a > 10)
>= or <=	if(a <= 5)

Using Assignment Operators

Operator	Example
=	a = b + c
+=	a += 3
-=	a -= b

Demo

Using operators in C#

Default values for types in C#

```
int intMaxValue = int.MaxValue;

int intMinValue = int.MinValue;

double doubleMaxValue = double.MaxValue;
```

Members on Primitive Types

```
char myChar = 'a';

bool isWhiteSpace = char.IsWhiteSpace(myChar);

bool isDigit = char.IsDigit(myChar);

bool isPunctuation = char.IsPunctuation(myChar);
```

Members of char Type

Demo

Working with members of int and char

Working with DateTime

Working with Dates

DateTime

TimeSpan

```
DateTime someDateTime = new DateTime(2021, 03, 28);

DateTime today = DateTime.Today;

DateTime twoDaysLater = someDateTime.AddDays(2);

DayOfWeek day = someDateTime.DayOfWeek;

bool isDST = someDateTime.IsDaylightSavingTime();
```

Working with DateTime

Demo

Working with DateTime

Converting between Types

This Doesn’t Work...

```
int a = 3;
a = "Hello world";
```

Changing between Types

Implicit conversion

Casting

Explicit conversion

Helpers

```
int a = 123456789;

long l = a;
```

◀ **Implicit cast**

```
double d = 123456789.0;

int a = (int) d;
```

◀ **Explicit cast**

```
double d = 12345.0;

int a =
(int)Convert.ChangeType(d, typeof(int));
```

◀ **Using a helper**

Demo

Converting between types

We'll take a look at parsing in the next module

Implicit Typing

So Far, We've Used Explicit Typing

Explicit typing

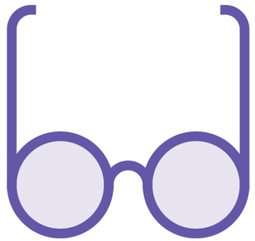
```
int a = 123;

bool b = true;

double d = 11.0;
```

Implicit typing

```
var a = 123;//a will be an integer

var b = true;//b will be a boolean

var d = 11.0;//d will be a double
```

Understanding Implicit Typing

Type is inferred



Not always as readable

Sometimes required (using LINQ)

Demo

Using var

Summary

C# is a strongly typed language

Contains primitive data types

- Value types

Conversion between types is supported

Resources

Other relevant courses in the C# path:

- **Controlling Program Flow in C#**
  - **Alex Wolf**
- **C# Language Integrated Query (LINQ)**
  - **Paul Sheriff**

INPUT

Up next:

Working with strings