Module 3

"Value Types and Expressions"
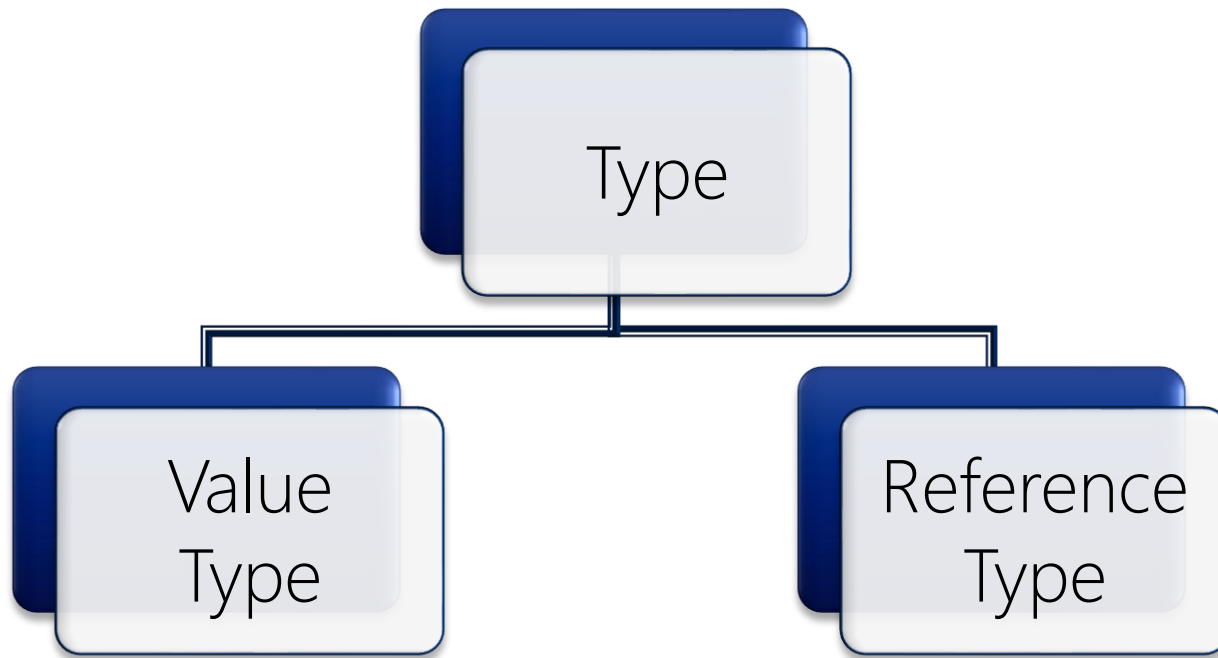
Agenda

▸ **.NET Common Type System**

▸ Predefined Value Types

▸ Expressions

▸ Data Type Conversions

▸ User-defined Value Types

▸ Lab 3

▸ Discussion and Review

Anatomy of the Common Type System

▸ Every variable has a specified type
▸ C# is type-safe…!

```
              ┌──────────┐
              │   Type   │
              └────┬─────┘
           ┌───────┴────────┐
     ┌─────┴─────┐    ┌──────┴──────┐
     │   Value   │    │  Reference  │
     │   Type    │    │    Type     │
     └───────────┘    └─────────────┘
```

Value Types vs. Reference Types

Value Types	Reference Types
▸ Directly contain data	▸ Store references to data ("objects")
▸ Allocated on the stack	▸ Stored on the heap
▸ Have to be initialized	▸ Has a default value of null
▸ Each copy has its own data	▸ Several references can refer to same data

Agenda

▸ .NET Common Type System

▸ **Predefined Value Types**

▸ Expressions

▸ Data Type Conversions

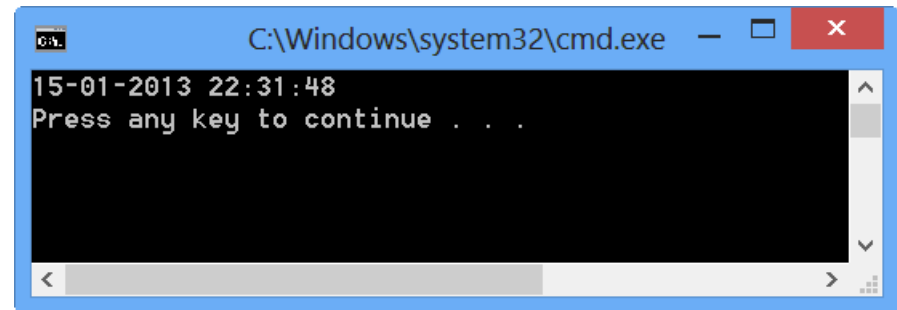▸ User-defined Value Types

▸ Lab 3

▸ Discussion and Review

Overview of Predefined Value Types

C# Data Type	CTS Type	Description
bool	System.Boolean	True or false values
int	System.Int32	Signed integers
short	System.Int16	Signed short integers
long	System.Int64	Signed long integers
uint	System.UInt32	Unsigned integers
char	System.Char	Character values
float	System.Single	Single-precision floating
double	System.Double	Double-precision floating
decimal	System.Decimal	128-bit precision number

System.DateTime

▸ A very important type with no C# keyword
▸ Has a lot of interesting details and features
  • High-precision
  • Versatile formatting is built-in

```
Console.WriteLine( DateTime.Now );
```


▸ A corresponding `System.TimeSpan` also exists

Agenda

▸ .NET Common Type System

▸ Predefined Value Types

▸ **Expressions**

▸ Data Type Conversions

▸ User-defined Value Types

▸ Lab 3

▸ Discussion and Review

Declaring Variables

▸ Declare by data type and variable name

```
bool isStarted;
```

▸ Multiple variables can be declared simultaneously

```
int favoriteNumber, i, j;
```

▸ Local variables can be declared everywhere in methods
▸ Class-level variables are called *members*

Assigning Values

▸ Assign values by using the assignment operator =

```
bool isStarted;
isStarted = true;
```

▸ Variables can be declared and assigned simultaneously

```
int favoriteNumber = 87, i = 0, j = i;
```

```
char c = 'A';
```

▸ Note: Variables <u>must</u> be initialized before use!

Naming of Variables

▸ Compiler requires that only letters, digits, and underscores are used
▸ C# keywords are reserved
▸ Case-sensitive!

▸ Recommendations
- Don't abbreviate. Characters are cheap! ☺
- Use camelCasing for variables
- Use PascalCasing for types, classes, methods.

Constants

▸ Use the const keyword to declare constants

```
const int favoriteNumber = 87;
```

▸ Must be initialized when declared

Operators

Operator Type	Operators
Equality	`==` `!=`
Relational	`<` `<=` `>` `>=` `is`
Conditional	`||` `&&` `?:`
Arithmetic	`+` `-` `*` `/` `%`
Increment, Decrement	`++` `--`
Assignment	`=` `+=` `-=` `*=` `/=`

Operator Precedence

▸ Operator precedence determines order of evaluation
  • Multiplicative > Additive.

  ```
  x = y + 87 * z
  ```

  ```
  x = y + ( 87 * z )
  ```

▸ Associativity
  • All binary (except assignment) is left-associative

  ```
  x + y + z
  ```
  ```
  ( x + y ) + z
  ```

  ```
  a = b = c
  ```
  ```
  a = ( b = c )
  ```

▸ Use parentheses whenever there is doubt!

Agenda

▸ .NET Common Type System

▸ Predefined Value Types

▸ Expressions

▸ **Data Type Conversions**

▸ User-defined Value Types

▸ Lab 3

▸ Discussion and Review

Implicit Conversions

▸ Also known as "widening" conversions
▸ Never lose precision or value

```
short i = 16384;

// Implicit/Widening conversion
int j = i;
```

▸ Are always allowed by the compiler
▸ Always succeeds

Explicit Conversions

▸ Also known as "narrowing" conversions
▸ Can lose precision or value

```
int i = int.MaxValue;

// Explicit/Narrowing conversion
short j = (short) i;
```

▸ Are allowed by the compiler
▸ Might fail!

Overflow Checking

▸ The **checked** keyword turns on over/underflow checking

```
checked
{
    short j = (short) i;
}
```

```
short j = checked( (short) i );
```

▸ There is an corresponding **unchecked** keyword

▸ Default (un)checking can be set in the Visual Studio project properties
  • "Build" -> "Advanced…"

Implicitly Typed Variables

▸ You can define <u>local</u> implicitly typed variables using the **var** keyword

```
var myInteger = 87;
var myBoolean = true;
var myString = "Hello, there...";
```

▸ The compiler infers the type of the local variable!
▸ Everything is still completely type-safe

```
var i = 87;          ✔
i = 112;             ✔
int j = i + 42;      ✔
i = "Forbidden!";    ✖
```

▸ Must be assigned a value when declared

```
var myInteger;       ✖
myInteger = 87;
```

Agenda

▸ .NET Common Type System

▸ Predefined Value Types

▸ Expressions

▸ Data Type Conversions

▸ **User-defined Value Types**

▸ Lab 3

▸ Discussion and Review

Enumerations

▸ Used for creating a set of symbolic names

```
enum Fruit
{
    Apple,
    Banana,
    Orange
}
```

```
Fruit f = Fruit.Banana;
```

▸ Ordering does not have to be sequential – and can also be bit flags!
▸ Underlying enumeration type can be explicitly chosen

```
enum Team : byte
{
    AGF = 1,
    Brøndby = 6,
    FCK = 5,
    Randers = 12
}
```

```
Team t = Team.AGF;
Console.WriteLine( t ); // ???
```

Structures

▸ Used for defining a structured value consisting of several subvalues

```
struct Point
{
    public int x, y;
}
```

```
Point pt;
pt.x = 42;
Console.WriteLine( pt.y ); // Oops!
```

▸ Members are private by default
▸ All subvalues <u>must</u> be initialized before use!

▸ Value can be default initialized using the **new** construct

```
Point pt = new Point();
Console.WriteLine( pt.y ); // ???
```

Quiz: Variables and Scope

```
int i = 0, j = 112;
Point pt;
pt.x = 42;
var l = ( 87 * Math.Sin( 11.2 ) / Math.PI ) + "Yo!";
string s = l;


{
    int k = 255;
    pt.x++;
    i = k;

    int j = 13;
}


Console.WriteLine( i );    // ???
Console.WriteLine( j );    // ???
Console.WriteLine( k );    // ???
Console.WriteLine( pt.x ); // ???
Console.WriteLine( s );    // ???
```

Lab 3: Creating and Using Value Types

Discussion and Review

▸ .NET Common Type System

▸ Predefined Value Types

▸ Expressions

▸ Data Type Conversions

▸ User-defined Value Types

WINCUBATE

Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net

Hasselvangen 243
8355 Solbjerg
Denmark