C# Fundamentals

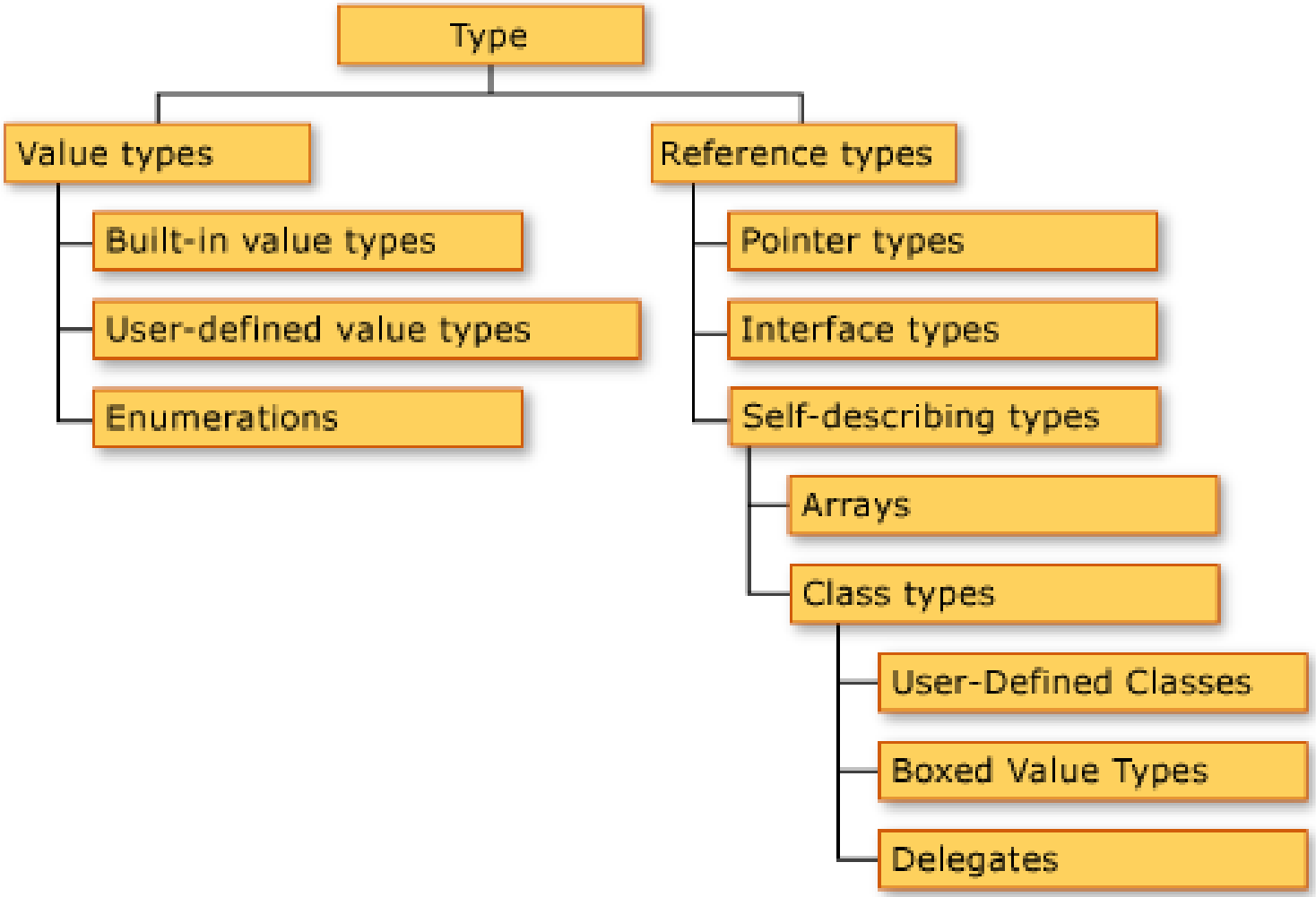
Hans-Wolfgang Loidl
<H.W.Loidl@hw.ac.uk>

**School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh**

Semester 1 — 2022/23

HERIOT WATT UNIVERSITY

C# Types



HERIOT WATT UNIVERSITY

Value Types

- **Variables stand for the value of that type (*"has value"*)**
- **Integers:**
  - **Signed:** `sbyte, int, short, long`
  - **Unsigned:** `byte, uint, ushort, ulong`
- **Floating point:**
  - `float`
  - `double`
- **Examples:**
  - `double average = 10.5`
  - `float total = 34.87f`

HERIOT WATT UNIVERSITY

Signed vs Unsigned

- **By default int, short, long are *signed* data types as they can hold a negative or a positive value of their ranges.**
- **Unsigned variables can only hold *positive* values of its range.**

Other value types:

- Decimal types: appropriate for storing monetary data. Provides greater precision.
  - `decimal profit = 2211655.76M;`
- Boolean variables: `True` or `False`.
  - `bool student = True;`

HERIOT WATT UNIVERSITY

Signed vs Unsigned

- By default int, short, long are *signed* data types as they can hold a negative or a positive value of their ranges.
- Unsigned variables can only hold *positive* values of its range.

Other value types:

- Decimal types: appropriate for storing monetary data. Provides greater precision.
  - `decimal profit = 2211655.76M;`
- Boolean variables: `True` or `False`.
  - `bool student = True;`

HERIOT WATT UNIVERSITY

Types and Values

Table 1, The Size and Range of C# Integral Types

Type	Size (in bits)	Range
sbyte	8	-128 to 127
byte	8	0 to 255
short	16	-32768 to 32767
ushort	16	0 to 65535
int	32	-2147483648 to 2147483647
uint	32	0 to 4294967295
long	64	-9223372036854775808 to 9223372036854775807
ulong	64	0 to 18446744073709551615
char	16	0 to 65535

Table 2. The Floating Point and Decimal Types with Size, precision, and Range

Type	Size (in bits)	precision	Range
float	32	7 digits	1.5×10^{-45} to 3.4×10^{38}
double	64	15-16 digits	5.0×10^{-324} to 1.7×10^{308}
decimal	128	28-29 decimal places	1.0×10^{-28} to 7.9×10^{28}

HERIOT WATT UNIVERSITY

Enumerations

- **Enum Types:**
  - **The enum keyword is used to declare an enumeration, a distinct type consisting of a set of named constants called the enumerator list.**
  - **Every enumeration type has an underlying type, which can be any integral type except char.**

Example:

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

- The default underlying type of the enumeration elements is `int`. By default, the first enumerator has the value 0, the next 1, etc.
- In the above example, `Sat` is 0, `Sun` is 1 etc
- Enumerators can have initialisers to override the default values, e.g.

  ```
  enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
  ```

- In this enumeration, the sequence of elements is forced to start from 1 instead of 0.

Enumerations

- **Enum Types:**
  - **The enum keyword is used to declare an enumeration, a distinct type consisting of a set of named constants called the enumerator list.**
  - **Every enumeration type has an underlying type, which can be any integral type except char.**

Example:

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

  - The default underlying type of the enumeration elements is `int`. By default, the first enumerator has the value 0, the next 1, etc.
  - In the above example, `Sat` is 0, `Sun` is 1 etc
  - Enumerators can have initialisers to override the default values, e.g.

    ```
    enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
    ```

  - In this enumeration, the sequence of elements is forced to start from 1 instead of 0.

Enumerations

- **Enum Types:**
  - **The enum keyword is used to declare an enumeration, a distinct type consisting of a set of named constants called the enumerator list.**
  - **Every enumeration type has an underlying type, which can be any integral type except char.**

  Example:

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

  - **The default underlying type of the enumeration elements is `int`. By default, the first enumerator has the value 0, the next 1, etc.**
  - **In the above example, `Sat` is 0, `Sun` is 1 etc**
  - **Enumerators can have initialisers to override the default values, e.g.**

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

  - **In this enumeration, the sequence of elements is forced to start from 1 instead of 0.**

Value Types

- A variable of *value type* directly represents its value (*"has value"*).
- Examples of value types are basic types such as `int`, `float`, `bool`
- Enumeration types, as above, are value types.
- Structures, that are collections of mixed types, are also value types.

Struct Types

- **Struct types:**
  - **are user-defined types**
  - **can contain data members of different types**
  - **cannot be extended**

Example:

```
struct Person {
  public string fName, lName;
  public Person(String fName, String lName) {
    this.fName = fName;
    this.lName = lName;
  }
}
Person p = new Person("John", "Smith");
```

HERIOT WATT UNIVERSITY

Struct Types

- **Struct types:**
  - **are user-defined types**
  - **can contain data members of different types**
  - **cannot be extended**

Example:

```
1 struct Person {
2   public string fName, lName;
3   public Person(String fName, String lName) {
4     this.fName = fName;
5     this.lName = lName;
6   }
7 }
8 Person p = new Person("John", "Smith");
```

HERIOT WATT UNIVERSITY

Structs vs Classes

Classes	Structs
Reference type	*Value type*
Used w/ dynamic instantiation	Used with static instantiation
Ancestors of class Object	Ancestors of class Object
Can be extended by inheritance	*Cannot* be extended by inheritance
Can implement one or more interfaces	Can implement one or more interfaces
Can initialize fields with initializers	*Cannot* initialize fields with initializers
Can have a parameterless constructor	*Cannot* have a parameterless constructor

HERIOT WATT UNIVERSITY

Reference Types

- **A variable of reference type contains a reference to a memory location where data is stored (as pointers in C/C++) (*"contains value"*). Properties:**
  - **Direct inheritance from Object.**
  - **Can implement many interfaces.**
  - **Two predefined reference types in C#:**
    - **String, e.g.:** `string name = "John";`
    - **Object, root of all types**

HERIOT WATT UNIVERSITY

Value vs Reference Types

- If x and y are of *value type*, the assignment
  `x = y`
  copies the contents of y into x.
- If x and y are of *reference type*, the assignment
  `x = y`
  causes x to point to the same memory location as y.

Example:

```
1  Person p = new Person("John", "Smith");
2  Person q = p;
3  p.fName = "Will";
4  // what is the value of q.fName?
```

Value vs Reference Types

- If x and y are of *value type*, the assignment
  `x = y`
  copies the contents of y into x.
- If x and y are of *reference type*, the assignment
  `x = y`
  causes x to point to the same memory location as y.

Example:

```
1  Person p = new Person("John", "Smith");
2  Person q = p;
3  p.fName = "Will";
4  // what is the value of q.fName?
```

HERIOT WATT UNIVERSITY

Boxing and Unboxing

- **Boxing is the conversion of a value type to a reference type. Unboxing is the opposite process.**
- **Using boxing, an int value can be converted to an object to be passed to a method (that takes an object as argument).**

Example:

```
1 int n = 5;
2 object nObject = n;       //boxing
3 int n2 = (int) nObject; //unboxing
```

HERIOT WATT UNIVERSITY

Boxing and Unboxing

- **Boxing is the conversion of a value type to a reference type. Unboxing is the opposite process.**
- **Using boxing, an int value can be converted to an object to be passed to a method (that takes an object as argument).**

Example:

```
1 int n = 5;
2 object nObject = n;       //boxing
3 int n2 = (int) nObject; //unboxing
```

HERIOT WATT UNIVERSITY

Casting

- **There are 2 ways of changing the type of a value in the program**
  - ***Implicit conversion* by assignment e.g.**

```
1 short myShort = 5;
2 int myInt = myShort;
```

  - ***Explicit conversion* using the syntax `(type)expression`**

```
1 double myDouble = 4.7;
2 int myInt = (int)myDouble;
```

HERIOT WATT UNIVERSITY

Nullable types

- Variables of reference type can have the value `null`, if they don't refer to anything.
- Variables of value type cannot have the value `null`, because they represent values.
- Sometimes it is useful to have a variable of value type that may have "no value".
- To this end, a *nullable type* can be used:

```
int?  i;
```

- Here, `i` is of type `int`, but may have the value `null`

HERIOT WATT UNIVERSITY

Arrays

- C# supports one- and multi-dimensional arrays.
- One-dimensional array
  - are declared like this
    ```
    string[] names = new string[30];
    ```
  - starts at index 0 up to index 29 (in general, bound - 1).
  - are accessed like this:
    ```
    names[2] = "John";
    ```
- Multi-dimensional array:
  ```
  int[,] numbers = new int[5,10];
  ```

HERIOT WATT UNIVERSITY

Some useful methods on arrays

- `Length` ... **Gives the number of elements in an array.**
- `Rank` ... **Gives the number of dimensions of the array.**
- `GetLength(n)` ... **Gives the number of elements in the n-th dimension**

HERIOT WATT UNIVERSITY

Jagged Arrays

- A jagged array is a multi-dimensional array, where the "rows" may have different sizes. It is declared like this
  `int [][] myJaggedArray = new int[4][];`
- The rows are filled in separately
  `myJaggedArray[0] = new int[5];`
- Access to array elements works like this:
  `myJaggedArray[0][2];`

HERIOT WATT UNIVERSITY

Control Structures: Conditional

```
1 if (expression)
2   statement 1
3 [else
4   statement 2]
```

In the above if statement:

- **The `expression` must evaluate to a `bool` value.**
- **If `expression` is true,**
  - **flow of control is passed to `statement 1`**
  - **otherwise, control is passed to `statement 2`.**
- **Can have multiple else clauses (using else if).**

HERIOT WATT UNIVERSITY

Logical Operators

- For comparing values these operators exist:
  `==, !=, <=, >=, <, >`
- *NB:* `=` is for assignment; `==` is for equality test
- These operators combine boolean values: `&&, ||, !`
- Operators over `int` and `float`: `+, -, *, /` (`%` int only)
- A conditional expression is written like this:

```
boolean_expr ?  expr_true :  expr_false
```

HERIOT WATT UNIVERSITY

Control Structures: Switch

```
switch (switch_expression) {
  case constant-expression:
    statement
    jump statement
  ...
  case constant-expressionN:
    statementN
    jump statement
  [default]
}
```

- `switch_expression` **must be of type** `sbyte, byte, short, ushort, int, uint, long, ulong, char or string`.
- **Each case clause must include a *jump-statement* (e.g. break statement) apart from the last case in the switch.**
- **Case clauses can be combined by writing them directly one after the other.**

Control Structures: Iteration

```
while (boolean_expression)
  statement
```

In a while statement, the `boolean_expression` is evaluated before the `statement` is executed, which is iterated while the `boolean_expression` remains true.

```
do
  statement
while (boolean_expression)
```

In a do/while statement the `boolean_expression` is evaluated *after* the `statement` is executed, which is iterated until the `boolean_expression` becomes false.

HERIOT WATT UNIVERSITY

Control Structures: Iteration

```
while (boolean_expression)
  statement
```

In a while statement, the `boolean_expression` is evaluated before the `statement` is executed, which is iterated while the `boolean_expression` remains true.

```
do
  statement
while (boolean_expression)
```

In a do/while statement the `boolean_expression` is evaluated *after* the `statement` is executed, which is iterated until the `boolean_expression` becomes false.

HERIOT WATT UNIVERSITY

Control Structures: Iteration (cont'd)

```
for (initialization; boolean_expression; step)
  statement
```

The for statement

- **performs `initialization` before the first iteration**
- **iterates while `boolean_expression` remains true**
- **performs `step` at the end of each iteration**

```
foreach (type identifier in expression)
  statement
```

The foreach statement iterates over arrays and collections. The variable `identifier` is bound to each element in turn.

HERIOT WATT UNIVERSITY

Control Structures: Iteration (cont'd)

```
for (initialization; boolean_expression; step)
  statement
```

The for statement

- **performs `initialization` before the first iteration**
- **iterates while `boolean_expression` remains true**
- **performs `step` at the end of each iteration**

```
foreach (type identifier in expression)
  statement
```

The foreach statement iterates over arrays and collections. The variable `identifier` is bound to each element in turn.

HERIOT WATT UNIVERSITY

Functions

- **Functions (or static methods) encapsulate common sequences of instructions.**
- **As an example, this function returns the n-th element of an array, e.g.**

```
1 static int Get (int[] arr, int n) {
2   return arr[n];
3 }
```

- **This static method is called directly, e.g.**

```
1 i = Get(myArr, 3);
```

Exercise: **check that n is in a valid range**

HERIOT WATT UNIVERSITY

Function Parameters

- **All objects, arrays and strings are passed by reference, i.e. changes effect the argument that is passed to the function:**

```
1 static void Set (int[] arr, int n, int x) {
2   arr[n] = x;
3 }
```

- **But, value types are copied. The keyword `ref` is needed for passing by reference:**

```
1 static void SetStep (int[] arr, ref int n, int x)
    {
2   arr[n] = x;
3   n += 1 ;
4 }
```

HERIOT WATT UNIVERSITY

Example: nullable types

```
1  public static int? Min(int[] sequence){
2    int theMinimum;
3    if (sequence.Length == 0)
4      return null;
5    else {
6     theMinimum = sequence[0];
7     foreach (int e in sequence)
8       if (e < theMinimum)
9         theMinimum = e;
10   }
11   return theMinimum;
12 }
```

HERIOT WATT UNIVERSITY

Discussion

- The type `int?` is a nullable int type.
- The value `null` of this type is used to indicate that there is no minimum in the case of an empty sequence.
- The method `HasValue` can be used to check whether the result is `null`:

```
int?  min = Min(seq);
if (min.HasValue) ...
```

- The combinator `??` can be used to select the first non-null value:

```
min ??  0
```

- This returns the value of `min`, if its value is non-null, 0 otherwise.

HERIOT WATT UNIVERSITY

Example: nullable types (cont'd)

```
1  public static void ReportMinMax(int[] sequence) {
2   if (Min(sequence).HasValue && Max(sequence).HasValue)
3     Console.WriteLine("Min: {0}. Max: {1}",
4                       Min(sequence), Max(sequence));
5   else
6     Console.WriteLine("Int sequence is empty");
7  }
8
9  public static void Main(){
10  int[] is1 = new int[] { -5, -1, 7, -8, 13};
11  int[] is2 = new int[] { };
12  ReportMinMax(is1);
13  ReportMinMax(is2);
14 }
```

HERIOT WATT UNIVERSITY

Exercises

Recommended Exercises:

(a) Define `Weekday` as an enumeration type and implement a `NextDay` method

(b) Implement a `WhatDay` method returning either `WorkDay` or `WeekEnd` (use another enum)

(c) Write a method calculating the sum from 1 to n, for a fixed integer value n

(d) Write a method calculating the sum over an array (one version with foreach, one version with explicit indexing)

HERIOT WATT UNIVERSITY

Exercises (cont'd)

(e) Use the `SetStep` method to implement a method Set0, which sets all array elements to the value 0.

(f) Implement a method, reading via `ReadLine`, and counting how many unsigned short, unsigned int and unsigned long values have been read.

(g) Define complex numbers using structs, and implement basic arithmetic on them.

Mandatory exercises:

(I) Implement Euclid's greatest common divisor algorithm as a static method over 2 int parameters.

(II) Implement matrix multiplication as a static method taking two 2-dimensional arrays as arguments.

HERIOT WATT UNIVERSITY