Yin Lou
CS 2026, Spring 2010

C# Types

Announcement

- Assignment 1 will be released tonight
  - Due next Friday
  - Submit through CMS
  - Write a small C# program
    - Input: A matrix and an operator (I/D/T)
      - I: Increase all elements in matrix by 1
      - D: Decrease all elements in matrix by 1
      - T: Return the transpose of this matrix

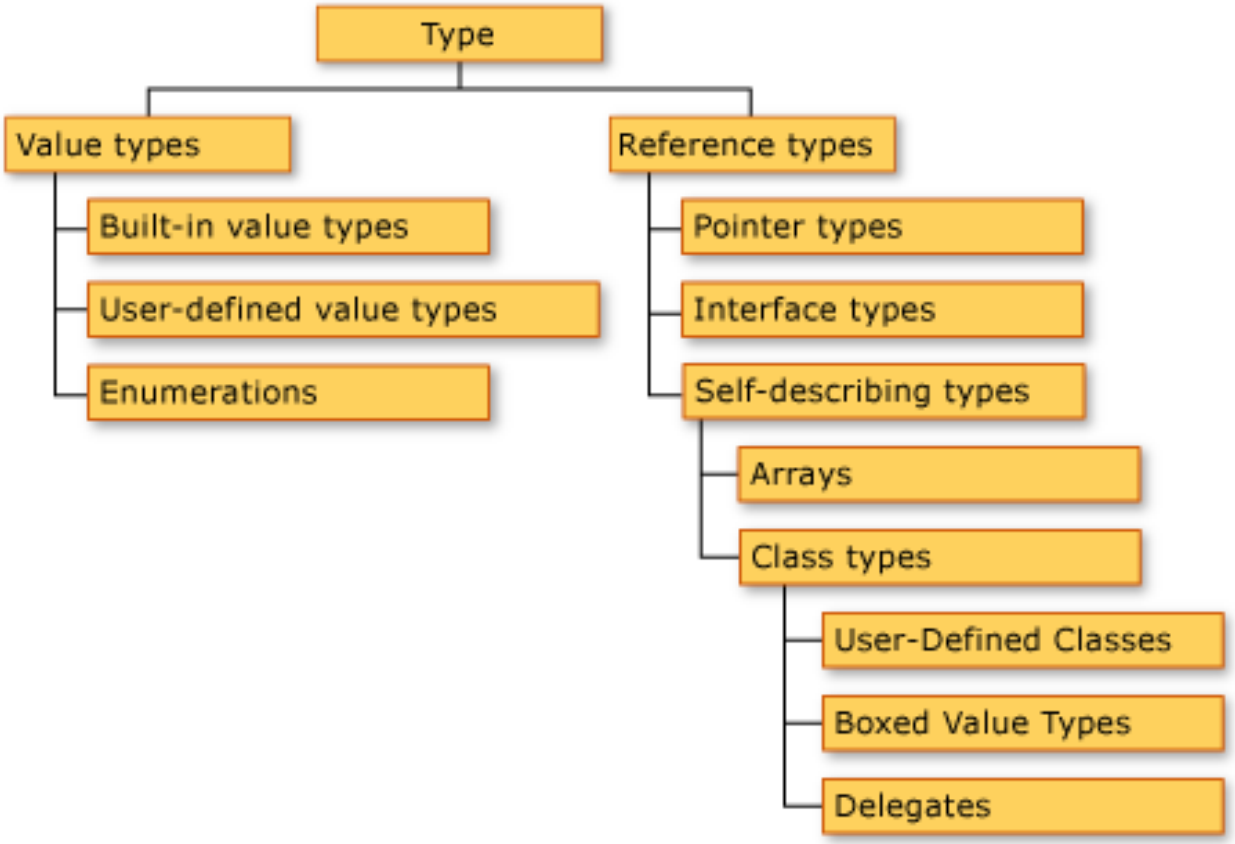
Review

- Visual C# 2008 Express walkthrough
- .Net framework, CLR, CTS
  - There's no concept of primitive type in C#
    - int is a value type
    - string is a reference type
- C# language guide

Outline

- C# Type
  - Value types
  - Reference types
  - Boxing and unboxing
- Basic C# Features: Arrays

Common Type System



Value Types

- Built-in value types
- User-defined value types
- Enumerations

Built-in Value Types

- Integer Types

Type Specifier	Bits	Range	Data Suffix
sbyte	8	-2^7 through 2^7-1	
byte	8	0 through 2^8-1	
short	16	-2^{15} through $2^{15}-1$	
ushort	16	0 through $2^{16}-1$	
int	32	-2^{31} through $2^{31}-1$	
uint	32	0 through $2^{32}-1$	U, u
long	64	-2^{63} through $2^{63}-1$	L, l
ulong	64	0 through $2^{64}-1$	UL, ul

Built-in Value Types

- Floating-Point Data Types

Type Specifier	Bits	Range	Data Suffix
float	32	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	F, f
double	64	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	D, d

- Declaration
  - float tax = 0.5f;
  - double tax = 0.5; / double tax = 0.5d;

Built-in Value Types

- Boolean Types

Type Specifier	Bits	Range	Data Suffix
bool	8	true, false	

- Traditionally, value types are stored outside dynamic memory allocation range
  - Stored in *stack* rather than *heap*

User-defined Value Types

- Structs
  - Contains a collection of fields, methods and properties
  - Non-extensible (sealed)
  - Suitable for representing lightweight objects
    - Struct is less expensive in terms of memory
    - e.g. Point

User-defined Value Types

- Definition

```
struct Point
{
        public int x;
        public int y;
        public Point(int x, int y)
        {
                this.x = x;
                this.y = y;
        }
}
```

- Instantiation

```
Point a = new Point(10, 10);
a.x = 20;
```

More on Structs

- System.Int32, System.Byte, System.Double, etc. are all structs
- Different from Java

Enumerations

- Definition

```
enum ClassDay
{
        Monday,
        Wednesday,
        Friday
}
```

- Instantiation

```
ClassDay today =  ClassDay.Friday;
```

Reference Types

- Reference types are always dynamically allocated
- Variables based on reference types (called objects) store *references* to the actual data
- The following keywords are used to declare a reference type
  - class, interface, delegate
- Example
  - String type: string

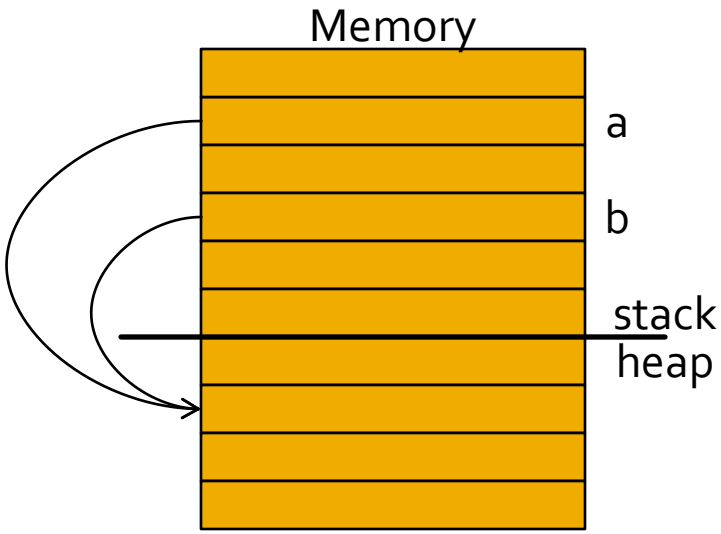
Comparison of Types

- Value
  - Value type variables directly contain values
  - Reference type variables contain a *pointer* to real object
- Inheritance
  - Value types inherit from System.ValueType
    - Treated specially by runtime; no subclassing
  - Reference types Inherit from System.Object
- Assignment
  - Assigning one value type to another copies the contained value
  - Assigning one reference type object to another duplicates the reference but not the actual value!

Memory Layout: Reference Types

- Refer to a memory location
  - very much like pointers in other languages like C/C++
- Can be set to null

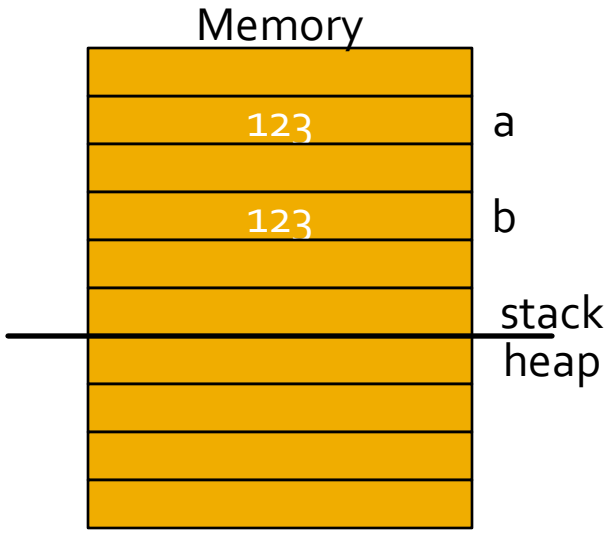
```
TypeA a = new TypeA();
TypeA b = a;
```



Memory Layout: Value Types

- Contain the actual value, not the location
- Inherit from System.ValueType
  - treated specially by the runtime: no subclassing
- Copies of value types make a *real* copy

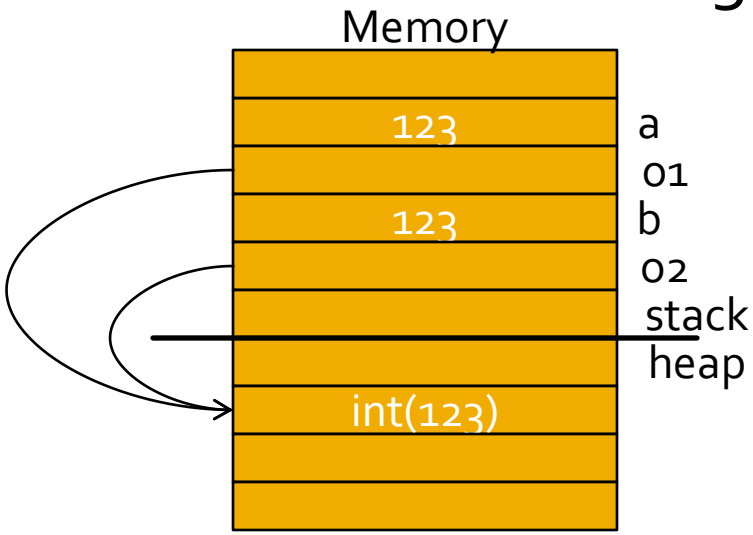
```
int a = 123;
int b = a;
```



Boxing and Unboxing

- Value types variables are not objects
  - This gives performance gain in most cases
  - But value variables can become objects on demand
  - Called "boxing"; reverse is called "unboxing"

```
int a = 123;
object o1 = a;
object o2 = o1;
int b = (int) o2;
```



Quiz: what will happen?

- Foo a = new Foo();
  Foo b = a;
  b.X = 10;
  Console.WriteLine(a.X); // output ?

- int a = 1;
  int b = a;
  b = 10;
  Console.WriteLine(a); // output ?

- This is important for parameter passing

C# Variables

- Definite assignment
  - int i; //i is a local variable
  - Console.WriteLine(i); //error CS0165: Use of unassigned local variable 'i'
- Default values
  - only for instance variables, static variables, and array elements
  - string s; // s == null
  - double d; // d == 0.0

C# Arrays

- Simple definition
  - int[] array = new int[30];
- "Jagged" Arrays
  - int[][] array = new int[2][];

    array[0] = new int[100];

    array[1] = new int[5];
  - The "outer" array with two elements will be stored consecutively.
  - However, the inner arrays (array[0] & array[1]) will not be stored consecutively in heap.
    - Recall that arrays are reference-type objects
  - Can have arbitrary dimensions

C# Arrays

- Multidimensional Arrays
  - Stored sequentially
  - Visually look like a rectangle (or a cube or a hypercube depending on the number of dimensions)
- Example
  - int[,] array = new int[9, 9];

    array[3,8] = 100;