C# Types

Mingsheng Hong
CS 215, Spring 2008

Administration

- CMS student list populated
  - Let me know if you have *not* received CMS emails
- First assignment released
  - Due on Feb. 1
  - Write a C# program that
    - takes as input regular English text
    - returns the transpose
  - Use C# style
    - much of what we discussed so far will be useful
    - should be short

Review

- Visual C# 2008 Express walkthrough
- .NET framework, CLR, CTS, CLS

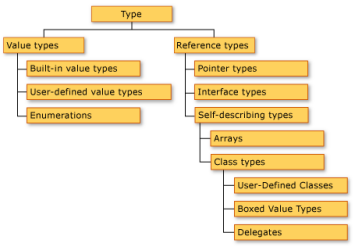
.NET Based Frameworks

- ASP.NET: web application framework
  - Use .NET languages in server-side web pages
  - Now supports AJAX!
- ADO.NET: Use the CLR to access databases
  - in the manner of ODBC
  - provides classes for access
- Microsoft XNA: game design framework
  - GDIAC at http://gdiac.cis.cornell.edu
  - CIS 300: Introduction to Computer Game Design

Roadmap for Today's Lecture

- C# types
  - Reference types
  - Value types
  - Boxing and unboxing
- Basic C# features
  - Arrays
  - OO features
  - Function parameters
  - Iterators

Common Type System



From MSDN

Common Value Types

- Integer types:
  - signed: `sbyte, int, short, long`
  - unsigned: `byte, uint, ushort, ulong`
- Floating point: `float, double, decimal`
- Everything in C# inherits from `object`
  - regular objects are heavy-weight
  - for efficiency, need primitive types

Common Reference Type

- string type: `string`
  - can index like char array
  - has methods such as Split
- e.g.,
  -
```
string s = "Hello";
char third = s[2];
string[] split = s.Split(third);
```

Common Types

- What is the difference between string and double?
  - reference types vs. value types
  - two families of types in C#

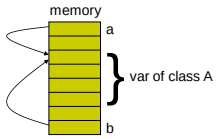
Reference Types

- Single inheritance in class hierarchy, rooted in *object*
  - Simple to Java classes
- Implement arbitrarily many interfaces
  - Similar to Java interfaces
- C# classes can be abstract
  - must be explicitly marked as abstract (contrast with C++)
- May contain non-method non-data members

Reference Types: Memory Layout

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

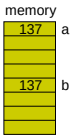
```
{
    A a = new A();
    A b = a;
}
```



Value Types: Memory Layout

- 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

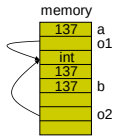
```
{
    int a = 137;
    int b = a;
}
```



Boxing and Unboxing

- Value types not objects
  - performance gain in common case
  - but can become objects on demand
  - called “boxing”. Reverse is “unboxing”
- Note: boxing still copies

```
{
    int a = 137;
    object o1 = a;
    object o2 = o1;
    int b = (int)o2;
}
```



Differences between Types

- Copy semantics for assignment:
  - ```
    MyClass a = new MyClass();
    MyClass b = a;
    b.X = 10;
    Console.WriteLine(a.X); //output ?
    ```
  - ```
    int a = 1;
    int b = a;
    b = 10;
    Console.WriteLine(a); //output ?
    ```
- Important for parameter passing, too

Common Value Types

- Simple types (i.e., integer and floating point)
- Enum types
  - has a corresponding *underlying type* (default: int)

Enum Example

- Definition

```
enum Color
{
  Red,
  Green,
  Blue
}
```

- Instantiation

```
Color c = Color.Red;
```

Common Value Types

- Simple types (i.e., integer and floating point)
- Enum types
  - has a corresponding *underlying type* (default: int)
- Struct types
  - user-defined value types
  - can contain arbitrary data
  - non-extensible (sealed subclasses)
  - examples: Point, KeyValuePair

Struct Example

- Definition

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

- Instantiation

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

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 elts
- eg.
  - `string s; // s == null`
  - `double x; // x == 0.0`

Basic C# Features

- Arrays
- OO features
  - Accessibility
  - Class members
    - Property
    - Indexer
    - Operator
- Function parameters
- Iterators

C# Arrays

- Can have standard C arrays
  - `int[] array = new int[30];`
  - `int[][] array = new int[2][];`
    `array[0] = new int[100];`
    `array[1] = new int[1];`
  - Called "jagged" arrays
  - Stored in random parts of the heap
- Can have arbitrary dimensions
- Recall that an array is an object

C# Arrays

- Multidimensional
  - stored sequentially
- `int[,] array = new int[10,30];`
  `array[3,7] = 137;`