C# Types

Hussam Abu-Libdeh
CS 2026, Spring 2009
Announcement

- **Reminder: Office Hours**
  - When: Wednesdays after class until 2pm
  - Where: 4161 Upson Hall

- **Homework 1 will be emailed to you tonight**
  - *If you do not receive it by 5pm, send me an email!*
    - hussam@cs.cornell.edu
  - Due next Friday 1/30
  - Submit through CMS

- **Write a small C# Program**
  - Takes in web page HTML code
  - Produce a list of all hypertext links
  - Count number of links pointing to the same website
Today's Agenda

- C# Types
  - Reference types
  - Value types
  - Boxing and unboxing

- Basic C# Features
  - Arrays
    - Example: Sudoku Maker
  - OO features
  - Iterators
Value Types

• Examples:
  • Integer types:
    – Signed: sbyte, int, short, long
    – Unsigned: byte, uint, ushort, ulong
  • Floating point: float, double, decimal
• Traditionally, value types are stored outside dynamic memory allocation range.
  • Stored in stack rather than heap
Value Types

- Value types in C# are of two main categories:
  - Structs
  -Enumerations
- Structs fall into these categories:
  - Numeric types
    - Integer, floating point, decimal
  - Bool
  - User-defined structs
Value Types

• All value types have default constructors assigning them values

• Generally can not be assigned `null`
  • Except in special cases
    – Won't be discussed in this class
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 type variables directly contain values

Inheritance:
- Value types inherit from System.ValueType
  - Treated specially by runtime; no subclassing
- 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
  
  ```java
  { 
    A a = new A();
    A b = a;
  }
  ```

- Value Types
  
  ```java
  { 
    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”

```c
{  
  int i = 123;
  object o = i;
  // object o = (object) i;
}
```
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
Enum Example

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

- Instantiation
  
  ```
  ClassDay today = ClassDay.Friday;
  ```
Structs

- A struct is a user-defined value type
- Contains a collection of fields, methods, and properties
- Suitable for representing lightweight objects such as Point, Rectangle ..etc
  - You can define a Point as a class (reference type)
  - However, struct can be less expensive in terms of memory
  - Example, array of 100 struct points is less expensive than an array of 100 class points. Why?
Struct Example

• Definition

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

• Instantiation

```java
Point a = new Point(10, 20);
a.x = 30;
```
Default Values for C# Variables

• Often variables in C# are given default values even if not assigned explicitly
  • `string s; // s == null`
  • `double x; // x == 0.0`

• Default values are only given to:
  • Instance variables, static variables, array elements

• Class example
  • `DefaultValues.cs`
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;`

- Class Example
  - SudokuMaker.cs