Advanced C# Types

Hussam Abu-Libdeh
CS 2026, Spring 2009

Administrative Issues

- First homework assignment due Friday
- Questions on homework ?
- Second assignment will be emailed on Saturday
- Office hour today in 4161 Upson until 2pm

From previous lecture

- To answer a question from previous lecture: get/set in a property can have different access modifiers

Previously

- OO features
  - Accessibility
  - Virtual and Override
  - Class members
    - Properties
    - Indexers
    - Operators
- Function parameters
  - ref, out

Today's Agenda

- Function params
  - params keyword
- Iterators
- Advanced C# types
  - Nullable types
  - Partial types
  - Generics

`params` keyword

- Used in methods where the number of arguments is variable
- Only one `params` keyword can be used in a method
- No parameters defined after the `params` parameter

params syntax example

```
public int SumGrades (params int[] grades)
{
    int sum = 0;
    for (int i=0; i<grades.length; i++)
       sum += grades[i];
    return sum;
}
```

Iterators

- Common programming pattern
- Allows you to walk through a collection of elements in a data structure
- Example:
  - ```
    foreach (string name in Names) {
      Console.WriteLine(name);
    }
    ```
- Similar to Java:
  - `for (String name : Names) { .. }`
    ```

Making your own Iterator

- Implement a method `GetEnumerator()` that returns an `IEnumerator`
- Use `yield return` to return current element
- Use `yield break` to conditionally stop the iteration

Custom Iterator Example

- ```
  public class Foo {
  int[] myArray;
  public IEnumerator GetEnumerator()
  {
     for (int i=0; i<myArray.length; i++)
     {
        yield return myArray[i];
     }
  }
  ```

- IteratorExample.cs
  ```

Nullable Types

- Primitive value-types, like `int`, have default values
- References are assigned `null` by default
  - An int can not be assigned null value
- Null values are useful to test whether a variable has been assigned to or not

Nullable Types

- C# 2.0 added nullable types
  - Value-types that can accept null
  - Example: int? a = null;
- The HasValue and Value properties
  - ```
    if (x.HasValue)
      { Console.WriteLine(x.Value) }
    ```
- The `??` operator
  - `a ?? b` evaluate to `a` if `a` is non-null, and to `b` otherwise
    ```

Partial Types

- It is better to break up large classes into multiple files
  - Increase readability
  - Separate generate and hand-written code
- As of C# 2.0 , partial classes allow splitting code into multiple files
  - `public partial class Foo { …}`
  - Each file must use partial
  - Compiler joins all the classes

Generics

- Write public class Stack<T> { .. }
  - T is the type variable (a class name)
  - Will be instantiated when declared
  - Stack<int> intStack = new Stack<int>();
- Can have multiple types
  - public class Dictionary<TWord, Tmeaning>
- Similar to templates in C++

Generics – `where` Clause

- Sometimes need to enforce constraints on the type sent to the generic class
- Use “where” clause in class definition to enforce constraints
  - `public class Stack<T> where T: new()`
  - `public class Stack<T> where T: iFace`
  - `public class Stack<T> where T: struct`
  - ….