C# 3.0

Hussam Abu-Libdeh
CS 2026, Spring 2009
Today's Agenda

- Checked & Unchecked
- C# 3.0 features
Checked and Unchecked

- Two contexts for evaluating arithmetic
- Unchecked
  - Default context
  - Overflows do not throw exceptions
  - Use `unchecked` operator to make explicit
    ```
    double d = double.MaxValue;
    unchecked { int a = (int) d; }
    ```
- Checked
  - Overflows throw `System.OverflowException`
  - Use `checked` operator
C# 3.0

• Some of C# 3.0 language features
  – Implicitly typed variables
  – Automatic properties
  – Initializers
  – Anonymous types
  – Lambda expressions
  – Extension methods
C# Version 3

• High level points
  – Less (finger) typing → shorter programs → fewer bugs
  – Better functional programming features
  – LINQ: language-integrated query

• E.g. Select items with more than 5 letters
  ```csharp
  string[] words = {"foobar", "foo", "bar", "letter"};
  IEnumerable<string> subset = from w in words where w.Length > 3 select w;
  ```
Implicitly Typed Local Vars

- Type of variable *inferred* from expression
  - Must include initializer
    ```csharp
    var a = 5;
    var b = "Hello";
    var c = 1.0;
    var orders = new Dictionary<int, Order>();
    ```
  - Works for loops
    ```csharp
    var evenNumbers = new int {2, 4, 6, 8};
    foreach (var n in evenNumbers) {
        Console.WriteLine("Item value: {0}", n);
    }
    ```
  - Can not be null. Why not ?
Implicitly Typed Local Arrays

- Must have consistent types
  - var a = new [] {1, 10, 245, 9871};

- Or have implicit conversion
  - var b = new [] {1, 3.14};
  - var c = new [] {1, "3.14"};
  // fails, why?
Automatic Properties

- Previously
  ```csharp
  class Car {
    private string carName;
    public string CarName {
      get { return carName; }
      set { carName = value; }
    }
  }
  ```

- In C# 3.0, Automatic property syntax
  ```csharp
  class Car {
    public string CarName { get; set; }
  }
  ```
Initializers

- Initializers for public fields or writable properties

```csharp
public class Point {
    public int X { get; set; }
    public int Y { get; set; }
}
```

- Point p1 = new Point(); p1.X=1; p1.Y=5;
- Point p2 = new Point { X = 1, Y = 5 };
Initializers

- Works with arrays and lists
  - `int[] digits = new int[] {2,3};`
  - `List<int> digits = new List<int> {2,3};`

- Can have complex and nested initializers
  - `Rect rectangle = new Rect { TopLeft = new Point {X=10, Y=10}, BottomRight = new Point {X=0, Y=30}};`
  - Can have a list of Rectangles in an array initializer
Anonymous Types

- var x = new {P1 = 10, P2 = "name"};
  - x is of anonymous type with two properties
  - Type can not be referred to by name in program
- Structural type equivalence
  - Two anonymous types can be compatible
- Can be nested
Lambda Expressions

- Generalized function syntax
  - \( \lambda x.x+1 \)
  - In C# 3.0, have \( x \Rightarrow x+1 \)
  - Syntax: (input params) \( \Rightarrow \) {function body;}

- From anonymous method syntax:
  - delegate (int x) { return x+1; }

- Example
  - List<int> evenNumbers = list.FindAll(i => (i%2) == 0);
  - FindAll:
    http://msdn.microsoft.com/en-us/library/fh1w7y8z.aspx
Notes on Lambda Expressions

- Can have implicitly typed variables
- Can have zero or more variables
- Can have expression or statement body
- Can be converted a compatible delegate

```csharp
delegate R Func<A,R>(A arg);
Func<int, int> f1 = x => x + 1;
Func<int, double> f2 = x => x + 2;
```
Extension Methods

• Can add methods to existing classes
  – New methods to be added must be defined in a static class
  – `this` modifier on the first parameter refers to the object being operated on

• Example

```csharp
public static class Extensions {
    public static int Inc(this string a) {
        return Int32.Parse(a) + 1;
    }
}
```

```csharp
int x = "1234".Inc();  // x = 1235
```