Goals of the C# language

• A simple, modern, general-purpose object-oriented language
• Software robustness and programmer productivity
  – Strong type checking, array bounds checking, detection of use of uninitialized variables, source code portability, automatic garbage collection
• Useable in creating software components
• Ease of learning by programmers familiar with C++ and Java
• Usable for embedded and large system programming
• Strong performance, but not intended to compete with C or assembly language

Type II safety cans for flammables
Brief history of C#

- Originated by Microsoft as a response to Java
  - Initial public release in 2000
- Language name inspired by musical note C#
  - A “step above” C/C++ (and Java)
  - Linux wags: D♭ (D-flat, same note, different name)
- Lead designers: Anders Hejlsberg, Scott Wiltamuth
  - Hejlsberg experience: Turbo Pascal, Borland Delphi, J+
- C# standardized via ECMA and ISO
  - However, Microsoft retains architectural control
Key language features

- Unified object system
  - Everything type is an object, even primitives
- Single inheritance
- Interfaces
  - Specify methods & interfaces, but no implementation
- Structs
  - A restricted, lightweight (efficient) type
- Delegates
  - Expressive typesafe function pointer
  - Useful for strategy and observer design patterns
- Preprocessor directives

[Link to image source: www.flickr.com/photos/spotsgot/1414345/]

[Link to image source: www.flickr.com/photos/spotsgot/1414345/]
Hello World example

class Hello
{
    static void Main()
    {
        // Use the system console object
        System.Console.WriteLine("Hello, World!");
    }
}

Creates a new object type (class) called Hello.

It contains a single method, called Main.

Main contains one line, which writes "Hello, World!" on the display.

The method that performs this action is called WriteLine.

The WriteLine method belongs to the System.Console object.

The keyword "static" means that the method Main can be called even if there is no current instance of the class. It's a class method, not an instance method.

The line beginning with // is a comment, and does not execute.

*Demonstration of creating Hello World inside Visual C# Express*
Syntax

- Case-sensitive
- Whitespace has no meaning
  - Sequences of space, tab, linefeed, carriage return
- Semicolons are used to terminate statements (;)
- Curly braces {} enclose code blocks
- Comments:
  - /* comment */
  - // comment
  - /// <comment_in_xml>
    - Automatic XML commenting facility
Classes and Objects

• A class combines together
  – Data
    • Class variables
  – Behavior
    • Methods

• A key feature of object-oriented languages
  – Procedural languages, such as C, did not require clustering of data and behavior

• Class-instance distinction
  – Class defines variables & methods
  – Need to create instances of the class, called objects, to use variables & methods
  – Exception: static methods and variables
  – Analogy: a jelly bean mold (class) can be used to create a large number of jelly beans (objects, instances of the class)
Defining a class

[attributes] [access-modifiers] class identifier [:base-class
{ class-body }

Simple example:

class A
{
    int num = 0;    // a simple variable

    A (int initial_num) { num = initial_num; } // set initial value of num
}

• Attributes: used to add metadata to a class
  – Can safely be ignored
• Access modifiers: one of
  – public, private, protected, internal, protected internal
• Base-class
  – Indicates (optional) parent for inheritance
• Interfaces
  – Indicates (optional) interfaces that supply method signatures that need to be implemented in the class
• Class-body
  – Code for the variables and methods of the class
Inheritance

- Operationally
  - If class B inherits from base class A, it gains all of the variables and methods of A
  - Class B can optionally add more variables and methods
  - Class B can optionally change the methods of A

- Uses
  - Reuse of class by specializing it for a specific context
  - Extending a general class for more specific uses

- Interfaces
  - Allow reuse of method definitions of interface
  - Subclass must implement method definitions

[Link to image: www.flickr.com/photos/spotsgot/1500855/]
Inheritance Example

class A
{
    public void display_one()
    {
        System.Console.WriteLine("I come from A");
    }
}

class B : A
{
    public void display_two()
    {
        System.Console.WriteLine("I come from B, child of A");
    }
}

class App
{
    static void Main()
    {
        A a = new A();       // Create instance of A
        B b = new B();       // Create instance of B

        a.display_one();    // I come from A
        b.display_one();    // I come from A
        b.display_two();    // I come from B, child of A
    }
}

In-class demo of this code in Visual C# Express
Visibility

• A class is a container for data and behavior
• Often want to control over which code:
  – Can read & write data
  – Can call methods
• Access modifiers:
  – Public
    • No restrictions. Members visible to any method of any class
  – Private
    • Members in class A marked private only accessible to methods of class A
    • Default visibility of class variables (but is good to state this explicitly)
  – Protected
    • Members in class A marked protected accessible to methods of class A and subclasses of A.
Visibility Example

class A {
    public int num_slugs;
    protected int num_trees;
    ...
}
class B : A {
    private int num_tree_sitters;
    ...
}
class C {
    ...
}

• Class A can see:
  – num_slugs: is public
  – num_trees: is protected, but is defined in A

• Class B can see:
  – num_slugs: is public in A
  – num_trees: is protected in parent A
  – num_tree_sitters: is private, but is defined in B

• Class C can see:
  – num_slugs: is public in A
  – Can’t see:
    • num_trees: protected in A
    • num_tree_sitters: private in B
Constructors

- Use “new” to create a new object instance
  - This causes the “constructor” to be called
- A constructor is a method called when an object is created
  - C# provides a default constructor for every class
    - Creates object but takes no other action
    - Typically classes have explicitly provided constructor
- Constructor
  - Has same name as the class
  - Can take arguments
  - Usually public, though not always
    - Singleton design pattern makes constructor private to ensure only one object instance is created
Type System

- Value types
  - Directly contain data
  - Cannot be null
  - Allocated on the stack
- Reference types
  - Contain references to objects
  - May be null
  - Allocated on the heap

```csharp
int i = 123;
string s = "Hello world";
```

Slide adapted from “Introduction to C#”, Anders Hejlsberg
www.ecma-international.org/activities/Languages/Introduction%20to%20Csharp.ppt
Predefined Types

• C# predefined types
  – Reference object, string
  – Signed sbyte, short, int, long
  – Unsigned byte, ushort, uint, ulong
  – Character char (2 byte, Unicode)
  – Floating-point float, double, decimal
  – Logical bool

• Predefined types are simply aliases for system-provided types
  – For example, int == System.Int32
Unusual types in C#

- **Bool**
  - Holds a boolean value, “true” or “false”
  - Integer values do not equal to boolean values
    - 0 does not equal false
    - There is no built-in conversion from integer to boolean

- **Decimal**
  - A fixed precision number up to 28 digits plus decimal point
  - Useful for money calculations
  - 300.5m
    - Suffix “m” or “M” indicates decimal
Unified type system

- All types ultimately inherit from object
  - Classes, enums, arrays, delegates, structs, ...
- An implicit conversion exists from *any* type to type object
Unified Type System (Boxing)

• Boxing
  – Process of converting a value type to the type **object**
  – Wraps value inside a System.Object and stores it on the managed heap
    • Can think of this as allocating a “box”, then copying the value into it

• Unboxing
  – Extracts the value type from the object
  – Checks type of box, copies value out

```csharp
int i = 123;
object o = (object)i;
int j = (int)o;
```

Slide adapted from “Introduction to C#”, Anders Hejlsberg
www.ecma-international.org/activities/Languages/
Introduction%20to%20Csharp.ppt
Variables

type variable-name [= initialization-expression];

Examples:

int number_of_slugs = 0;
string name;
float myfloat = 0.5f;
bool hotOrNot = true;

Also constants:

const int freezingPoint = 32;

• Variables must be initialized or assigned to before first use
• Class members take a visibility operator beforehand (private by default)
• Constants cannot be changed
Enumerations

```
enum identifier [: base-type]  
{ enumerator-list}

Example:
enum Grades  
{  
  gradeA = 94,  
  gradeAminus = 90,  
  gradeBplus = 87,  
  gradeB = 84  
}
```

• Base type can be any integral type (ushort, long) except for char
• Defaults to int
• Must cast to int to display in Writeln
  – Example: (int)g.gradeA
Conditionals

if (expression)  
  statement1  
[else  
  statement2]

Example:

if (i < 5) {  
  System.Console.WriteLine("i is smaller than 5");  
} else {  
  System.Console.WriteLine("i is greater than or equal to 5");  
}

• C# supports C/C++/Java syntax for “if” statement
• Expression must evaluate to a bool value  
  – No integer expressions here
• == means “equal to” for boolean comparison  
  – if (i == 5)  // if i equals 5  
  – if (i = 5)  // error, since i = 5 is not a boolean expression
Switch statement

```
switch (expression)
{
    case constant-expression:
        statement(s);
        jump-statement
    [default: statement(s);]
}
```

Example:
```
const int raining = 1;
const int snowing = 0;
int weather = snowing;
switch (weather) {
    case snowing:
        System.Console.WriteLine("It is snowing!");
        goto case raining;
    case raining:
        System.Console.WriteLine("I am wet!");
        break;
    default:
        System.Console.WriteLine("Weather OK");
        break;
}
```