

Lab 1: Overview of lab

- Two technologies:
 - Phidgets starting in week 2
 - Sensors, rfid, motors
 - PocketPC starting in week 6
 - Mobile device (PDA)
 - Smartphone starting in week 10
- Every week:
 - Must demonstrate you have completed lab before the beginning of the next week's lab
 - Attendance is mandatory

Lab Grades

- 15% of total grade
 - 5% weekly (in-class) assignments
 - 5% phidgets homework
 - Due 9/25, 9/27 in lab
 - 5% mobile homework
 - Due 11/13, 11/15 in lab

Programming Environment

- Visual Studio .NET
- C#

Visual Studio

- “Design” and “Code” views
- Help
- IntelliSense
- Keyboard Mapping

C# Overview

- Object oriented
- Everything belongs to a class
 - no global scope
- Complete C# program:

```
using System;
namespace ConsoleTest
{
    class Class1
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Statements and Comments

- Case sensitive (myVar != MyVar)
- Statement delimiter is semicolon ;
- Block delimiter is curly brackets { }
- Single line comment is //
- Block comment is /* */
 - Save block comments for debugging!

Data

- All data types derived from System.Object
- Declarations:
 - datatype varname;*
 - datatype varname = initvalue;*
- C# does not automatically initialize local variables (but will warn you)!

Value Data Types

- Directly contain their data:
 - int (numbers)
 - long (really big numbers)
 - bool (true or false)
 - char (unicode characters)
 - float (7-digit floating point numbers)
 - string (multiple characters together)

Data Manipulation

| | |
|----|------------------|
| = | assignment |
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |
| % | modulus |
| ++ | increment by one |
| -- | decrement by one |

Data Example

```
using System;
namespace ConsoleTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            int myInt;
            string myStr = "2";
            bool myCondition = true;

            while(myCondition) ;
        }
    }
}
```

strings

- Immutable sequence of unicode characters (char)
- Creation:
 - `string s = "Bob";`
 - `string s = new String("Bob");`
- Backslash is an escape:
 - Newline: `"\n"`
 - Tab: `"\t"`

string/int conversions

- string to numbers:
 - `int i = int.Parse("12345");`
 - `float f = float.Parse("123.45");`
- Numbers to strings:
 - `string msg = "Your number is " + 123;`
 - `string msg = "It costs " +
String.Format("{0:C}", 1.23);`

String Example

```
using System;
namespace ConsoleTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            int myInt;
            string myStr = "2";
            bool myCondition = true;

            Console.WriteLine("Before: myStr = " + myStr);
            myInt = int.Parse(myStr);
            myInt++;
            myStr = String.Format("{0}", myInt);
            Console.WriteLine("After: myStr = " + myStr);

            while(myCondition) ;
        }
    }
}
```

Arrays

- (page 21 of quickstart handout)
- Derived from `System.Array`
- Use square brackets `[]`
- Zero-based
- Static size
- Initialization:
 - `int [] nums;`
 - `int [] nums = new int[3]; // 3 items`
 - `int [] nums = new int[3] {10, 20, 30};`

Arrays Continued

- Use Length for # of items in array:
 - `nums.Length`
 - Static Array methods:
 - Sort `System.Array.Sort(myArray);`
 - Reverse `System.Array.Reverse(myArray);`
 - IndexOf
 - LastIndexOf
- `Int myLength = myArray.Length;`
`System.Array.IndexOf(myArray, "K", 0, myLength)`

Arrays Final

- Multidimensional

```
// 3 rows, 2 columns
```

```
int[ , ] myMultiIntArray = new int[3,2]
```

```
for(int r=0; r<3; r++)
```

```
{
```

```
    myMultiIntArray[r][0] = 0;
```

```
    myMultiIntArray[r][1] = 0;
```

```
}
```

Conditional Operators

| | |
|----|-----------------------|
| == | equals |
| != | not equals |
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |
| && | and |
| | or |

If Statement

```
if (expression)  
    { statements; }  
else if  
    { statements; }  
else  
    { statements; }
```

If Statement Example

```
using System;
namespace ConsoleTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            int myInt = 3;
            if(myInt == 1)
                Console.WriteLine("myInt = 1");

            else if(myInt == 2)
                Console.WriteLine("myInt = 2");

            else
                Console.WriteLine("myInt > 2");
            while(true) ;
        }
    }
}
```

Switch Statement

```
switch (i) {  
    case 1:  
        statements;  
        break;  
    case 2:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

Switch Statement

```
switch (str) {  
    case "ABC":  
        statements;  
        break;  
    case "XYZ":  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

Switch Example

```
switch(myInt)
{
    case 1:
        Console.WriteLine("myInt = 1");
        break;
    case 2:
        Console.WriteLine("myInt = 2");
        break;
    default:
        Console.WriteLine("myInt > 2");
        break;
}
```

Loops

```
for (initialize-statement; condition; increment-statement);  
{  
    statements;  
}
```

```
while (condition)  
{  
    statements;  
}
```

- can include *break* and *continue* statements

Classes, Members and Methods

- Everything is encapsulated in a class
- Can have:
 - member data
 - member methods

```
Class clsName{  
    modifier dataType varName;  
    modifier returnType methodName (params)  
    { statements;  
        return returnVal; }  
}
```

Example

```
using System;
namespace ConsoleTest
{
    public class Class1
    {
        public string FirstName = "Kay";
        public string LastName = "Connelly";

        public string GetWholeName()
        {
            return FirstName + " " + LastName;
        }

        static void Main(string[] args)
        {
            Class1 myClassInstance = new Class1();

            Console.WriteLine("Name: " + myClassInstance.GetWholeName());

            while(true) ;
        }
    }
}
```

Class Constructors

- Automatically called when an object is instantiated:

```
public className(parameters)
{
    statements;
}
```

Example

```
using System;
namespace ConsoleTest
{
    public class Class1
    {
        public string FirstName;
        public string LastName;

        public Class1(string fName, string lName)
        {
            FirstName = fName;
            LastName = lName;
        }
        public string GetWholeName()
        {
            return FirstName + " " + LastName;
        }
        static void Main(string[] args)
        {
            Class1 myClassInstance = new Class1("Kay", "Connelly");

            Console.WriteLine("Name: " + myClassInstance.GetWholeName());

            while(true) ;
        }
    }
}
```

Get/Set Accessors

- Accessors can be used to mediate any access to class variables:

```
private dataType mVarName  
public dataType varName  
{  
    get { statements; return mVarName;}  
    set { statements; mVarName = value;}  
}
```

Accessor Example

```
using System;  
namespace ConsoleTest  
{
```

```
    public class Class1  
    {
```

```
        private string mFirstName;
```

```
        private string mLastName;
```

```
        public string FirstName
```

```
        {
```

```
            get {return mFirstName;}  
            set {mFirstName = value;}
```

```
        }  
    }  
    public string LastName
```

```
    {
```

```
        get {return mLastName;}  
        set {mLastName = value;}
```

```
    }  
    // read only, composes to class variables
```

```
    public string FullName
```

```
    {
```

```
        get {return mFirstName + " " + mLastName;}
```

```
    }  
    public Class1(string fName, string lName)
```

```
    {
```

```
        FirstName = fName;
```

```
        LastName = lName;
```

```
    }  
    static void Main(string[] args)
```

```
    {
```

```
        Class1 myClassInstance = new Class1("Kay", "Connelly");
```

```
        Console.WriteLine("Name: " + myClassInstance.FullName);
```

```
        while(true) ;
```

```
    }  
}
```

```
}
```

```
}
```

Why Accessors?

- Can have read or write-only variables
- Can change internal data representation without altering code that access it
- Can execute code whenever variable is accessed
- Can make properties that are a combination of variables (FullName is combination of FirstName and LastName)

Collections

- Arrays are a collection
- Also have:
 - BitArray // array of bits (booleans)
 - Queue // first-in, first-out
Enqueue, Dequeue, Count
 - Stack // last-in, first-out
Push, Pop, Peek, ToArray
 - HashTable // key-value pairs
Add, Clear, Contains, Values
ContainsKey, ContainsValue,
Item, KeyEquals, Keys, Remove

Collections Continued

- SortedList
- ArrayList

// Collection of objects that can dynamically be resized. (Arrays are static size!)

// Add, BinarySearch, Clear, Contains, Insert, remove, RemoveAt, Reverse, Sort

.NET Namespaces of Interest

- System.

- Collections
- Data.
 - Common
 - SqlClient
 - SqlServer
 - SqlTypes
- Drawing
- IO
- Net

- System.

- Runtime.
 - Remoting
 - Serialization
- Security
- Text.
 - RegularExpressions
 - StringBuilder
- Threading
- Web.
 - Services
- Windows.Forms
- Xml

Assignment

- Write a class named “Task” that has:
 - Class members:
 - Label (string),
 - Priority (integer between 0 and 10)
 - Class methods:
 - Constructor: that has a label name and priority as input
 - PrettyFormat: returns a string that contains the label and priority of the task
 - SetPriority: sets the priority between 0 and 10
 - Note: can do this with set accessor if you want!
 - Have a Main function that instantiates the class with whatever values you like and prints it to the console.

Backup Assignment (switch)

- Write a program that:
 - Declares a string
 - You can initialize the string to whatever value you like, but it should work if I type in a different string!
 - Prints a different msg to the console depending on the value of the string. You should have at least 3 string-specific msgs and 1 default msg.

Assignment

- Write a program that:
 - Declares an array of integers
 - You can initialize the array to have whatever values and whatever size, but it should work if I type in a different array!
 - Computes the sum of the integers in the array using a loop
 - Prints to the console if the sum is:
 - ≤ 10
 - > 10 but ≤ 100
 - > 100 but ≤ 1000
 - > 1000