3D Games Programming

CI7500

Prof Vasilis Argyriou

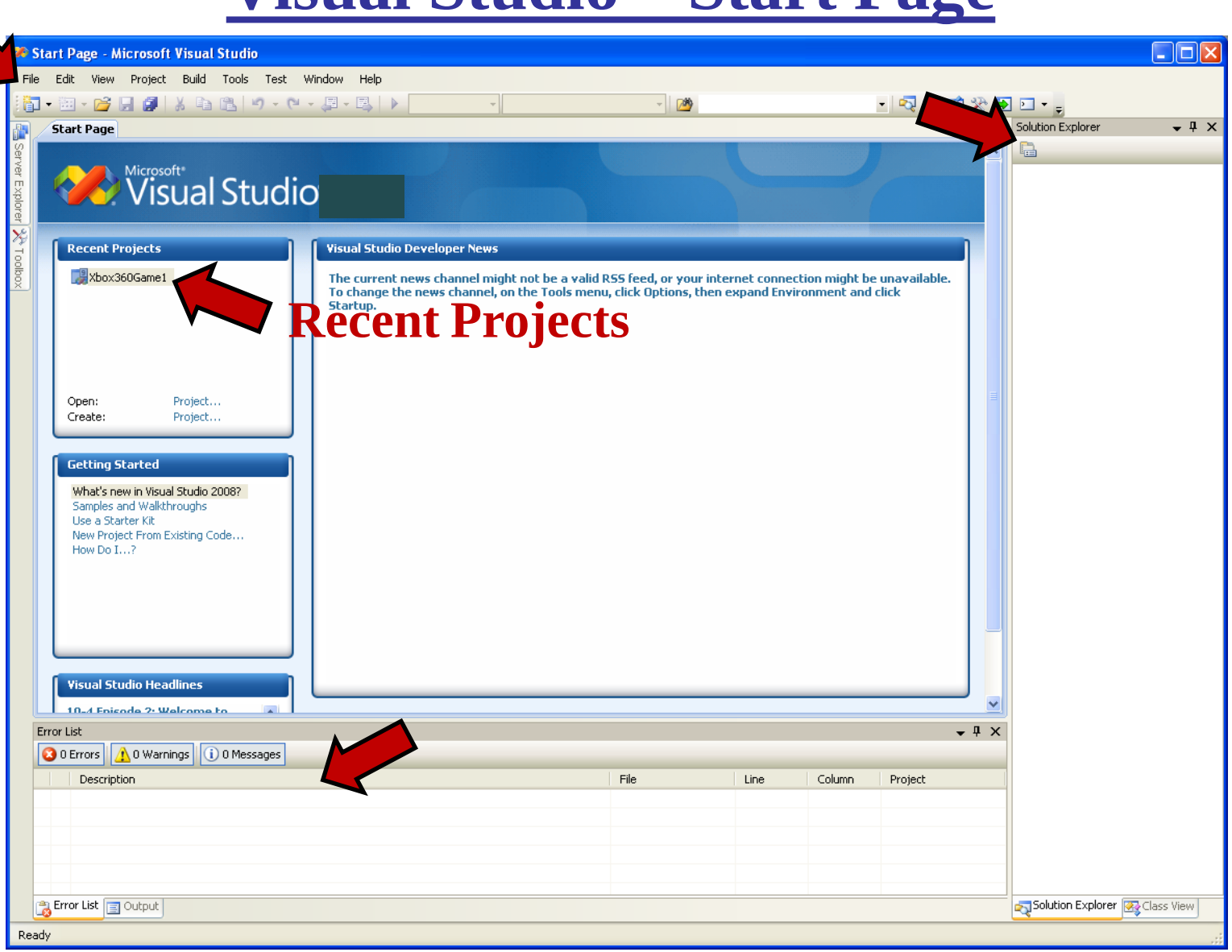
Vasileios.Argyriou@kingston.ac.uk

Lecture overview

- **Introduction to C#**

  Visual Studio, Hello World, Debugging, Control structures and operators, Strings, numbers, enumerations, Class members – fields, methods, properties, Inheritance, Polymorphism and overriding methods.

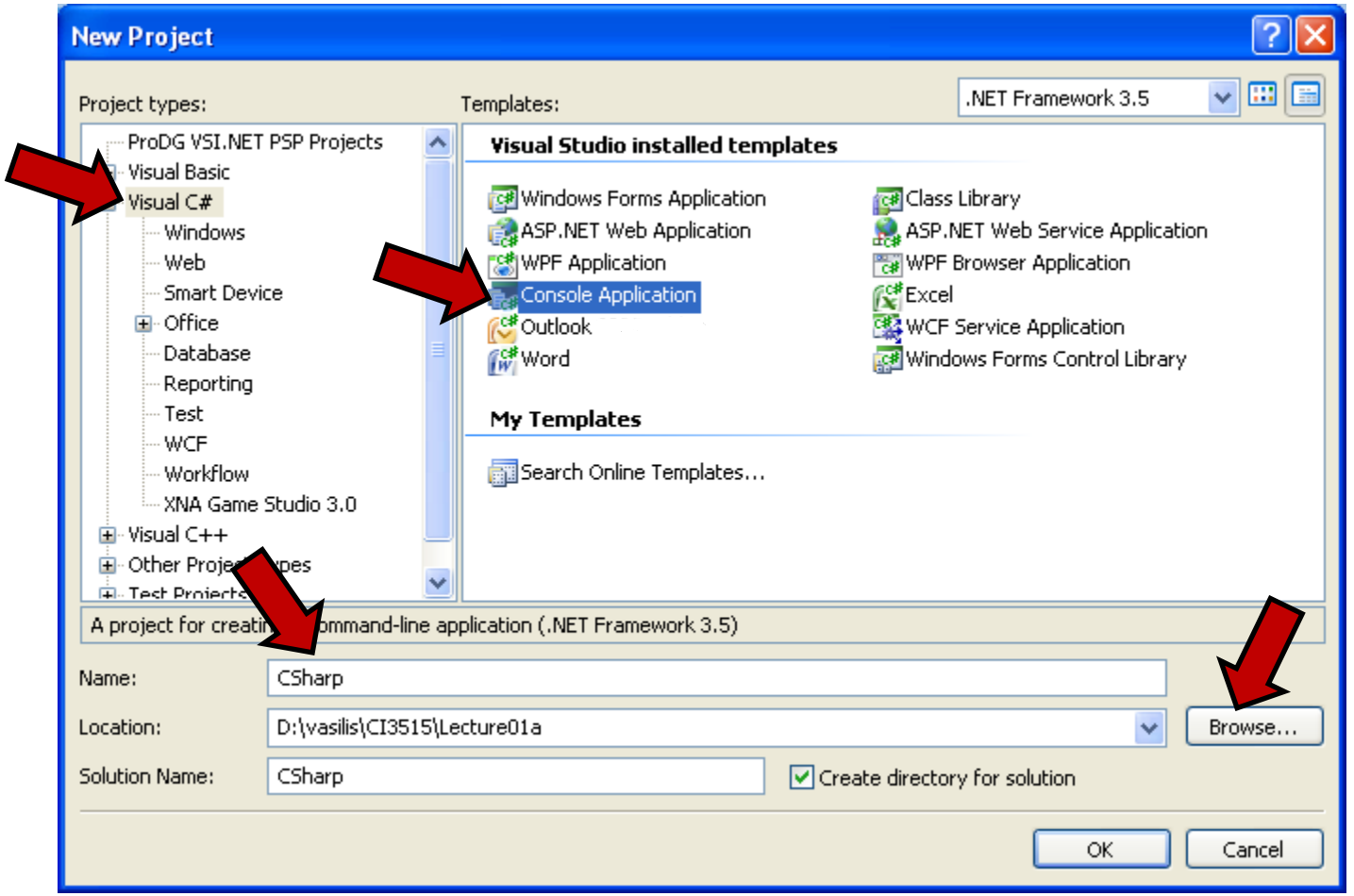
Visual Studio – Start Page



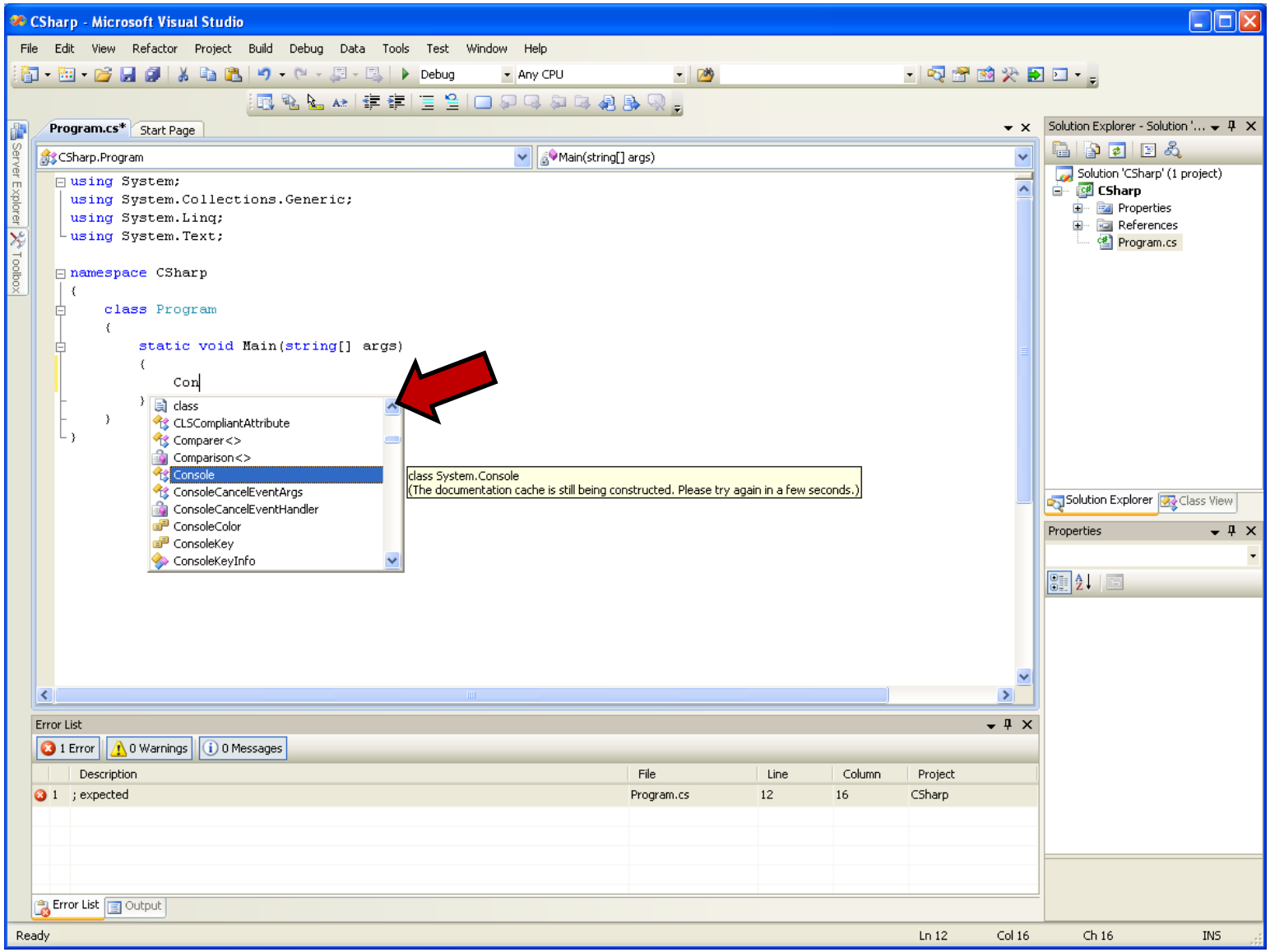
New console application

- **Start a new project**

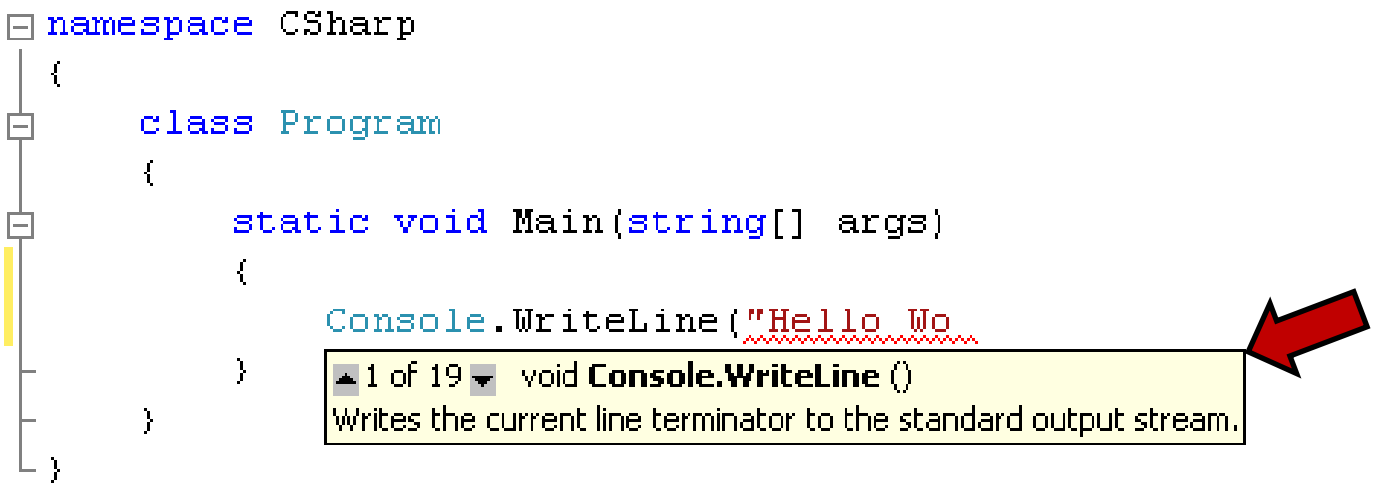
  File → New → Project



Intellisense



Intellisense



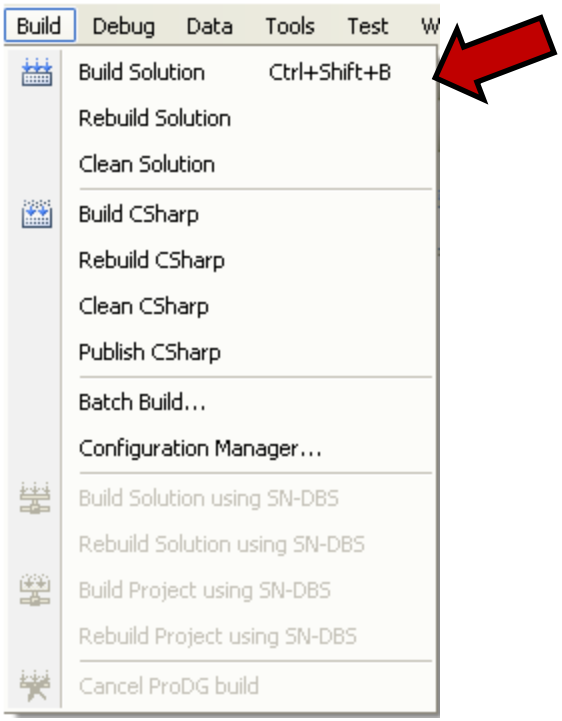
Hello World

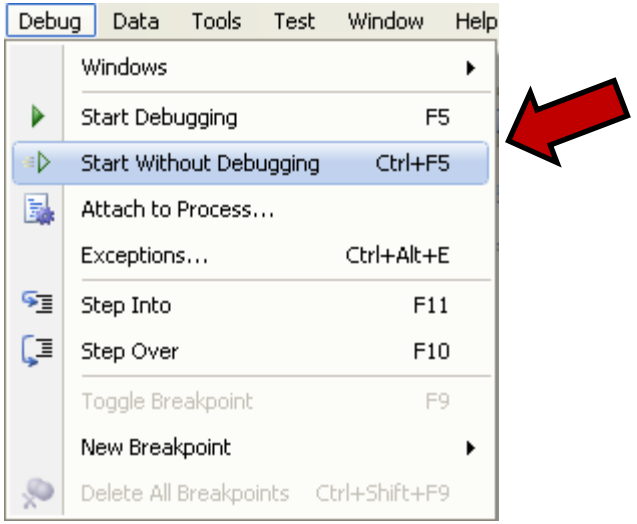
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

Hello World

- Build, Debug and Execute





How C# programs run

- **C# was developed specifically for the .NET platform**
  - Code is compiled to be machine-independent
- **Intermediate Language (MSIL)**
  - Similar to Java bytecode
- **Programs execute within an environment called Common Language Runtime (CLR)**
  - Similar to Java virtual machine
- **Both MSIL and CLR designed to support code written in various languages (e.g. Visual Basic .NET)**
  - Modules written in different languages interact easily

Control flow statements

- if () , else, else if ()
- switch(){case :break;}
  - break and return must come after each case
- for(;;)
- while()
- do{}while()
- foreach(element in collection)
  - Repeats body of loop for each element in array or collection
- break
- continue
- goto

Operators

Primary: x.y, a[x], x++, x--, new
Unary: -, !, ~, ++x, --x, &, sizeof
Multiplicative: *,/,%
Additive:+,-
Shift: <<,>>
Relational and type testing: <, >, <=, >=
Equality: ==,!=
Logical AND:&
Logical XOR: ^
Logical OR: |
Conditional AND: &&
Conditional OR: ||
Conditional: ?:
Assignment: =,+=,- =,*=,/=,%=,&=,|=,^=,<<=, >>=

Some Types

object	**ALL types derive from object**
int	int i = 0;
string	string s = “hello world”;
char	char c = ‘A’;
double	double d = 18.5;
float	float f = 3.0f;
bool	bool b = true;
enums	eg: DayOfWeek.Friday

Strings

- C# strings use 2-byte Unicode characters

  string s = “some string”;

  string s += “ and some other string”;

  s.Length – read only attribute

  Many other members to the string class e.g. Split(), SubString(), etc.

- Can be indexed: char c = s[2];

- Usual escape sequences

  \” \n \\ etc.

- Precede with @ to make a literal string

  @”C:\temp\foo” is the same as

  “C:\\temp\\foo”

- Well defined logical operators like ==, >, etc.

Arrays, Generics

- Arrays: strongly typed, fixed length

  int[ ] i = { 1, 2, 4 };

  int[ ] i = int[3];

- Generics collections: strongly typed, variable length

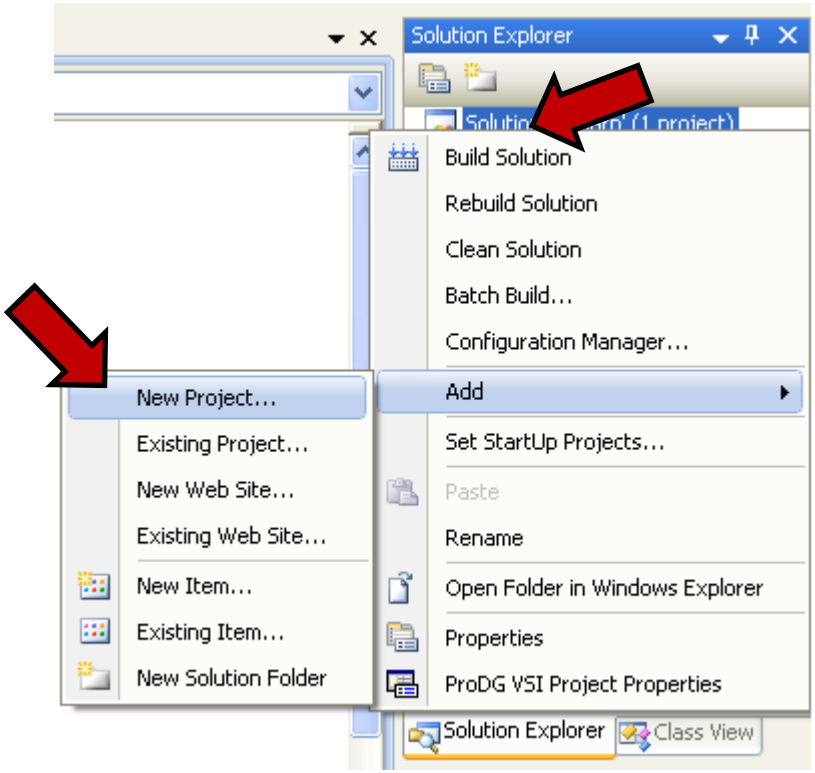
  Like C++ STL templates

  Examples: List<T>, Stack<T>, Queue<T>, Dictionary<K,T>

  Enable by “using System.Collections.Generic;”

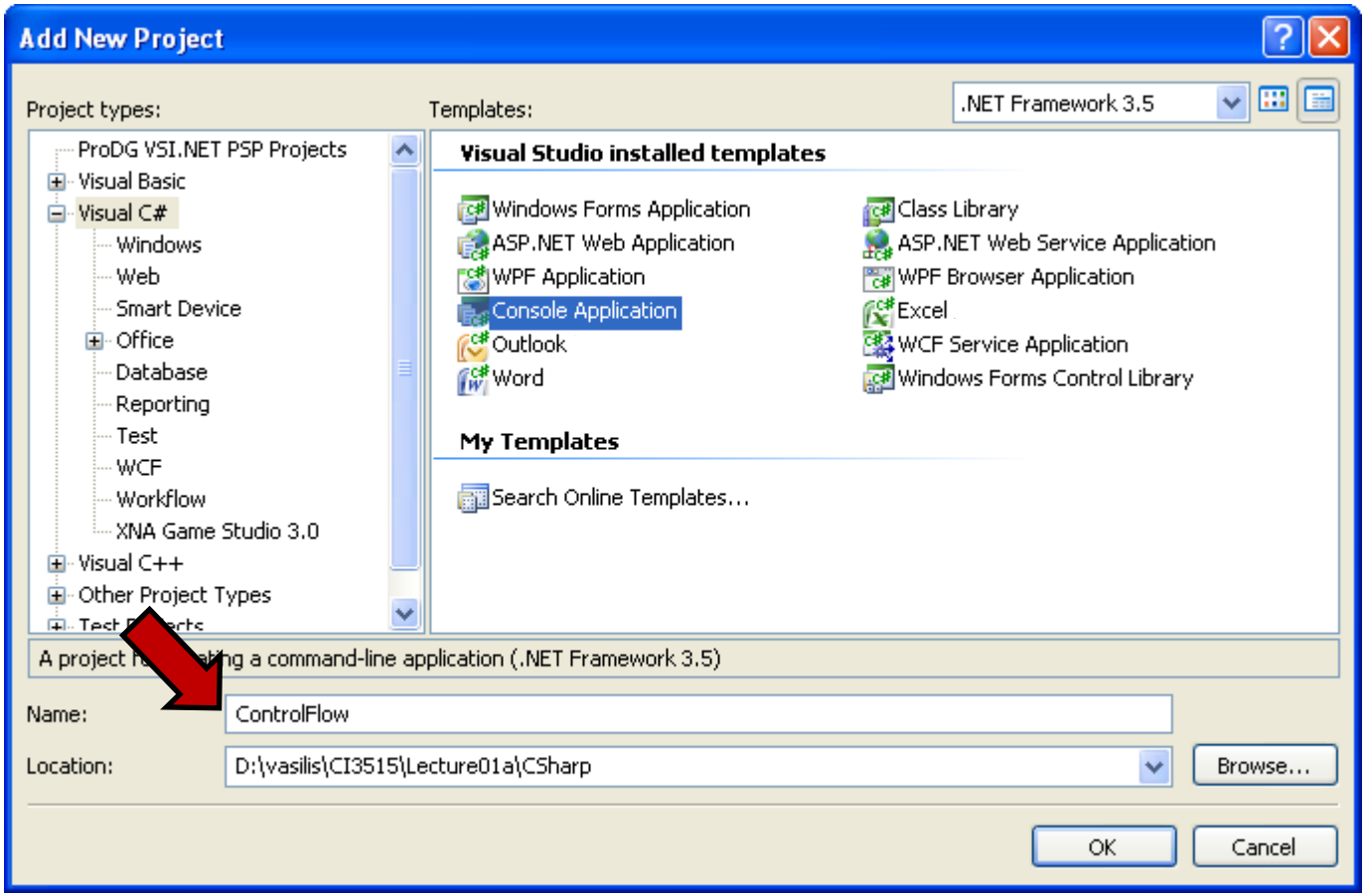
Add Second Project

- Right Click on the Solution name



Add Second Project

- Select ‘Console Application’ and give the name ‘ControlFlow’



Control Flow Example

- Part 1/3

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ControlFlow
{
    class Program
    {
        static void Main(string[] args)
        {
            String lastCommand;
            int commandCount = 0;
            Boolean done = false;
            while (!done)
            {
                Console.WriteLine("Enter your next command");
                lastCommand = Console.ReadLine();
                commandCount = commandCount + 1;
                System.Diagnostics.Debug.Assert(commandCount < 5);
                switch (lastCommand)
                {
                    case "hello":
                        Console.WriteLine("Hello world!");
                        break;
                    case "exit":
                        done = true;
                        break;
```

Control Flow Example

- Part 2/3

```
case "time":
    Console.WriteLine("The time is " + DateTime.Now.ToShortTimeString());
    if (DateTime.Now.DayOfWeek == DayOfWeek.Friday)
    {
        Console.WriteLine("Have a nice weekend!");
    }
    else if (DateTime.Now.DayOfWeek == DayOfWeek.Monday)
    {
        Console.WriteLine("Have a nice week!");
    }
    else
    {
        Console.WriteLine("Lets play Xbox!");
    }
    break;
case "count":
    Console.WriteLine("The number of commands so far " + commandCount);
    for (int i = 0; i < commandCount; i++)
    {
        Console.Write("*");
    }
    Console.Write("\n");
    break;
case "math":
    int[] array = { 1, 2, 3 };
    foreach (int a in array)
    {
        Console.WriteLine(a*a);
    }
    break;
```

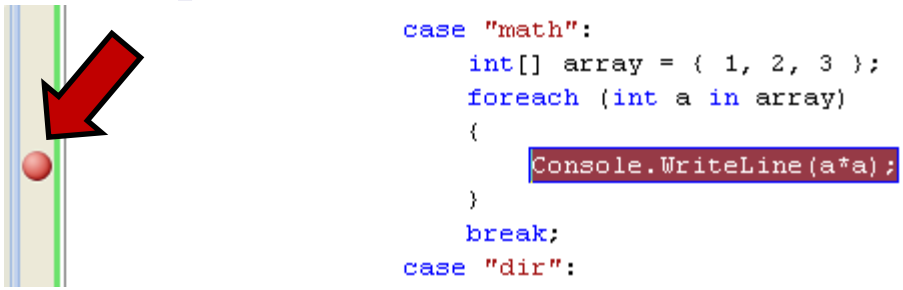
Control Flow Example

- Part 3/3

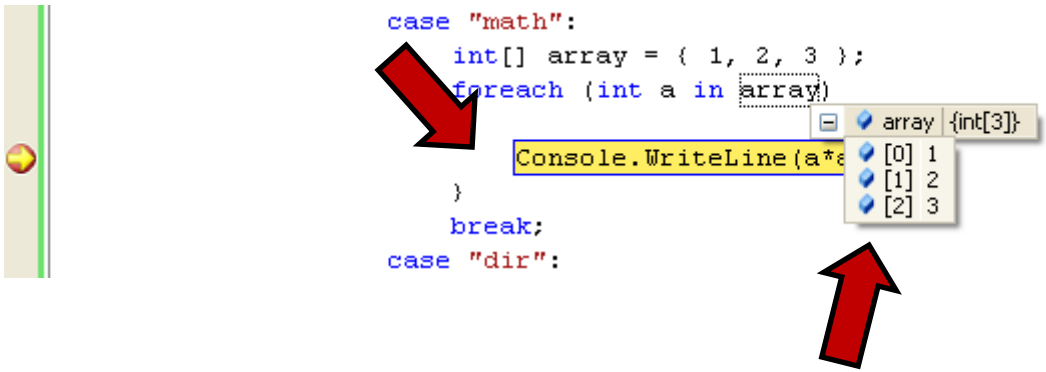
```
                    case "dir":
                        foreach (string fileName in Directory.GetFiles("."))
                        {
                            Console.WriteLine(fileName);
                        }
                        break;
                    case "game":
                        gameStage mygameStage = gameStage.exit;
                        if (mygameStage == gameStage.exit)
                        {
                            Console.WriteLine("Game Over!");
                        }
                        break;
                    default:
                        Console.WriteLine("Invalid command");
                        break;
                }
            }
            Console.WriteLine("Thank you for using the complex hello world");
        }
        private enum gameStage
        {
            menu,
            playing,
            exit
        }
    }
}
```

Debugging - Breakpoints

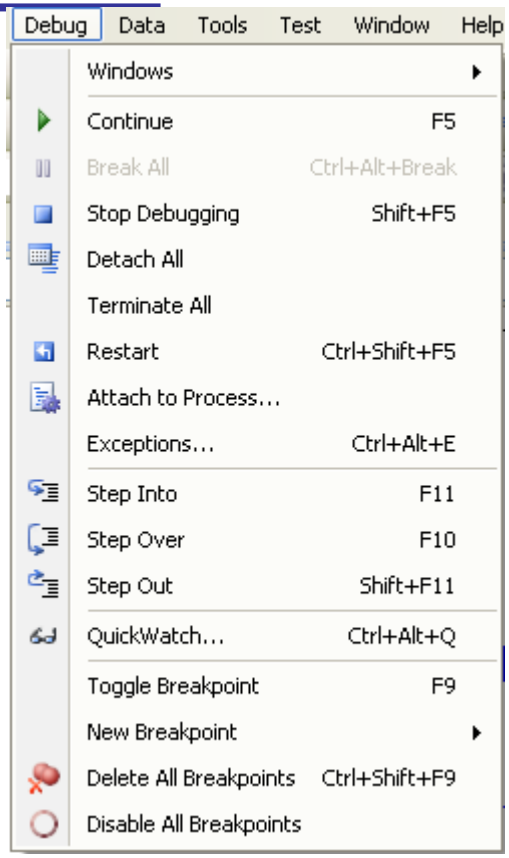
Breakpoint indicator



Indicates next line to be executed



Hovering over variable shows its value



F9 toggles breakpoints
F5 start debugging
F10 step over
F11 step into
Shift F5 kills debugging

C# Namespaces

- Used to avoid name conflicts (many classes with same name)
  - >Potential for name conflicts much higher are large projects or projects using many external libraries
  - >Every class has to be inside a namespace
  - >Namespaces can be nested

- The “using somenamespace;” directive gives the convenience of **not** having to use fully qualified names for all classes
  - >May lead to name conflicts, compiler detects ambiguities

Class

- At the heart of object oriented programming
  Application structure mirrors real world objects
- Related methods and data encapsulated in object
- Objects with the **same structure** are of **same type**
- A class is a **blueprint** for all things of that type
- **Instance** of a class is a thing, an object
- Classes have three main types of **members**
  Methods (functions in other languages)
  Fields (the data, sometimes called member variables)
  Properties (accessed like fields, but actually are methods)

Class

```
class Vehicle
{
    private float m_posX;
    private float m_posY;
    private float m_speed;
    private float m_dirX;
    private float m_dirY;

    public float Speed
    {
        get { return m_speed; }
        set { m_speed = value; }
    }

    public Vehicle(float posX, float posY, float speed, float dirX, float dirY)
    {
        this.m_posX = posX;
        this.m_posY = posY;
        this.m_dirX = dirX;
        this.m_dirY = dirY;
        this.m_speed = speed;
    }

    public void move()
    {
        this.m_posX += m_dirX * m_speed;
        this.m_posY += m_dirY * m_speed;
    }
    public void myLocation()
    {
        Console.WriteLine("X=" + m_posX + " Y=" + m_posY);
    }
}
```

Class

A **property** looks just like a **field** outside the class

Just like a field, a property has a type associated with it

Accessors are methods for reading or writing the property – each field may have **get** and **set** accessor

Class author writes body of accessors

get accessor must return value of same type as property

set accessor receives implicit parameter “value”

By controlling the protection level of the accessors (or omitting one of them) the class author can control who can **read** and who can **write** property

Class

Class members and classes can have one of the following protection levels

- **public** – accessible to everyone
- **private** – accessible only inside class
- **protected** – accessible for descendants
- **internal** – accessible within the same assembly

Default protection levels

- Class members, struct members – **private**
- Classes, structs, enums – **internal**

Class

Methods

- Must exist in a surrounding class or struct
- They have access to private members of the class
- Typically they are public
- "Global" methods done as **static public** methods
- Each method has name, return type, and 0 or more typed arguments
- The **void** return type indicates that the method does not return anything
- **Overloading**: two methods can have the same name, but differ in number or type of the arguments
- The various overloaded methods have separate bodies

Class

Constructors

- Constructors are special methods
  - same name as class
  - no return type
  - used for initialization of a class instance
  - may be overloaded

- If no constructor specified, compiler generates one that initializes members to default values

Class

Static versus instance

All instances of a class share certain traits – but have individual copies

> Rexx and Fido are Dogs but have different names

> **Name** is a trait shared by all instances of the class Dog but each instance of Dog has its own copy

A trait present in all instances of a class and physically shared by all instances is a "**static**" trait

> Can be methods or fields (e.g. Counter of total enemies at each moment)

Class

The "this" keyword

- Refers to the current instance (the object whose method is executed)
- Used to qualify access to members of the current instance
- Typically used for disambiguating a member variable from a method parameter of the same name (e.g. Collision detection)
- Cannot be used in **static** methods

Inheritance

Why use inheritance?

- Code reuse

  The derived class has all members of the base class
- Polymorphism

  > An object belonging to the derived class can be used where the program expects an object from the base class

  > Some methods of the derived class may behave differently than the same methods in base class

  > Methods in different derived classes may differ

  > Polymorphism means that at **run time** the environment picks the method to run based on **actual type** of object

Inheritance

Answers the question ‘is’. For example a Car is a Vehicle

```
class Car : Vehicle
{
    private int m_wheels;
    private int m_doors;

    public Car(float posX, float posY, float speed, float dirX, float dirY, int wheels, int doors)
        : base(posX, posY, speed, dirX, dirY)
    {
        this.m_wheels = wheels;
        this.m_doors = doors;
    }
    public void openDoor(int door)
    {
        Console.WriteLine("Door " + door + " is open");
    }
}
```

Polymorphism

Polymorphic *virtual* methods

- In base class, use keyword "**virtual**" for methods you want to behave in a polymorphic way
- In derived class, use keyword "**override**" for methods that implement polymorphic behaviour
- Can use the "**base.method()**" syntax to call the named method in the base class
- Use keyword "**abstract**" for polymorphic methods for which the base class does not define a body
- If any method in class is abstract, class must be abstract
- Non-abstract derived class overrides abstract methods
- Abstract classes cannot be instantiated
- You **cannot** use the **virtual** modifier with the **static**, **abstract**, **private** or **override** modifiers.

Polymorphism

```
public class DrawingObject
{
    public virtual void Draw()
    {
        Console.WriteLine("I'm just a generic drawing object.");
    }
}
public class Line : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Line.");
    }
}

public class Circle : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Circle.");
    }
}

public class Square : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Square.");
    }
}
```

Polymorphism

```
class Program
{
    static void Main(string[] args)
    {
        Vehicle myVehicle = new Vehicle(1f, 1f, 2.6f, 1.6f, 2.7f);
        myVehicle.myLocation();
        myVehicle.move();
        myVehicle.myLocation();

        DrawingObject[] dObj = new DrawingObject[4];
        dObj[0] = new Line();
        dObj[1] = new Circle();
        dObj[2] = new Square();
        dObj[3] = new DrawingObject();

        foreach (DrawingObject drawObj in dObj)
        {
            drawObj.Draw();
        }
    }
}
```

Summary

- Introduction to C#