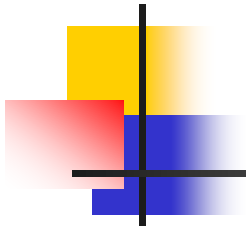
# CS 3201 – Program Construction I

Introduction to Visual Studio and C#

# Items to remember

- Drop/add ends today, Tue, Aug 18th at 11:59p.
- Install Visual Studio, ReSharper, and GhostDoc on own machine.
- Assignment 1
  - Available Thursday
  - Due Mon, Aug 31st at 9a.

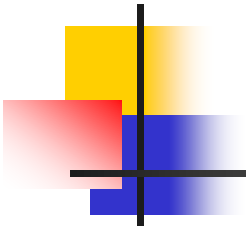


# Reading assignment

- Read chapters 1 and 2 in *Clean Code*.
  - Chapter 1 – Introduction
  - Chapter 2 – Meaningful names

# Last class

- Course introduction
  - Program
  - Programming language
- Clean code: Writing code for humans
  - Introduction: Why should I care?
  - Principles: Three Core Principles
    - The right tool for the job
    - High signal to noise ratio (SNR)
      - DRY principle
      - TED (Terse, Expressive, Do one thing)
    - Self-documenting code



# Learning a new language

- Best way
  - Jump in and get started.
- Take advantage of tutorials and code samples.

# C# resources

- C# resources
- Docs.microsoft.com
  - Home for Microsoft technical documentation.
- Googling for help
  - *C# or .NET classname*
  - Verify the Product or Version in the upper left-hand corner.

# Main C# architect

- Anders Hejlsberg
  - Created Turbo Pascal.
  - Led team that designed Borland Delphi.
    - One of the first successful IDEs for client/server programming.
- Based on C++, Java, and Delphi.
  - Some say it is “Microsoft’s version of Java.”
    - However, it is not just a Microsoft clone of Java.

# The .NET development platform

- Set of languages
  - C#, Visual Basic, F#
- Set of development tools and centralized documentation system.
  - Visual Studio and docs.microsoft.com
- Consistent API
  - .NET Standard
    - Base set of APIs that are common to all .NET implementations.
- Libraries
  - NuGet is a package manager built specifically for .NET that contains over 90,000 packages.
- Common Language Runtime (CLR) environment.
  - Executes objects/programs built within the framework.

# Cross platform

- .NET Core
  - Cross-platform .NET implementation for websites, servers, and console apps on Windows, Linux, and macOS.
- .NET Framework
  - Supports websites, services, desktop apps, and more on Windows.
- Xamarin/Mono
  - .NET implementation for running apps on all the major mobile operating systems.

# .NET Core Class Libraries example

- Namespace.Class.Method
  - `System.Console.WriteLine("Hi");`

- "Importing" a namespace
  - `using namespace;`
  - Can call without the use of the namespace.
    - `Console.WriteLine("Hi");`
  - Can only do `using` on the namespace.

# First C# program

```
public class HelloWorld
{
   public static void Main()
   {
        System.Console.WriteLine("Hello World!");
   }
}
```

- **Entry point of the program**
- **Must be capitalized**
- **Must be declared as static**

# First C# program using a namespace

```
using System;

public class HelloWorld
{
   public static void Main()
   {
       Console.WriteLine("Hello World!");
   }
}
```

- **Entry point of the program**
- **Must be capitalized**

# Equivalent Visual Basic Hello World program

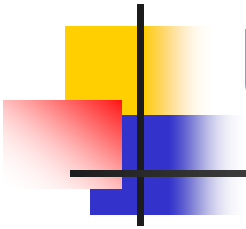
```
Imports System

Module Hello

    Sub Main()
        Console.WriteLine("Hello World")
    End Sub

End Module
```

# Creating first program in Visual Studio .NET

- Create New project that is a Visual C# `Console Application`
  - Selecting .NET Framework vs. .NET Core
  - How to pick a .NET runtime for an application
- Name the project `GettingStarted`
  - Note the location of the project
- Renaming the class to `HelloWorld`
  - In C# the filename and class name do not have to match like in Java.

# Outputting text

- Console class has methods for I/O for the console.
  - E.g., `Read, WriteLine`
- Example
  - `Console.WriteLine("Hello");`



# Using the IDE

- Outline/Code folding
- Intellisense
  - Code snippets
  - Surround with
- Building and running a program
  - Rebuilding an application
  - Debug builds
- Syntax errors
  - Error list

# Misc C# items

- C# is case sensitive, so variable names, `grade` and `Grade` represent two different variables.

- Commenting like Java.

```
// Starts a comment

/* Allows a multiline comment,
   without putting a starting comment
   symbol on each line */
```

# .NET Framework Data Types

| C# type | VB type | .NET type | Description |
|---|---|---|---|
| byte | Byte | Byte | Unsigned values (0 to 255) |
| char | Char | Char | Unicode characters |
| bool | Boolean | Boolean | **true** or **false** |
| short | Short | Int16 | Signed values (-32,768 to 32,767) |
| int | Integer | Int32 | Signed values (-2,147,483,648 to 2,147,483,647) |
| uint | UInteger | UInt32 | Unsigned values (0 to 4,294,967,295) |
| double | Double | Double | Floating point values |
| string | String | String | Character strings: “Hello” |

# Implicitly-typed **local** variables

- Explicit
  - `int sum = 0;`
  - `string name = “Duane”;`
- Implicit
  - Determines the date type of the variable based on the data type on the right-hand side of the assignment statement when **initially** declare the variable.
  - `var sum = 0;`
  - `var name = “Duane”;`

# Iteration

- Using classic for loop

```
for (var i = 0; i<grades.Count; i++)
{
  Console.WriteLine(grades[i]);
}
```

- Using foreach

```
foreach (var currGrade in grades)
{
  Console.WriteLine(currGrade);
}
```

# C# `List<T>` class

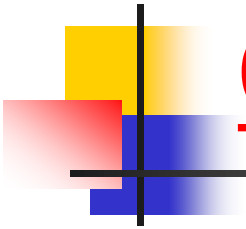
- Can access elements via [] notation.
- `Count` property has number of elements.
- No equivalents of get and set methods like Java.
  - Not needed in .NET with [] accessibility.

- Note:
  - There is an `ArrayList` class in C#, but do **NOT** use it.
    - **Use the `List` class.**

# Classes in C#

- Like Java, everything is defined within a class in C#.
- Example

```
public class Student
{
   // Define attributes

   // Define behavior

}
```



# C# properties

- Allow access to the state of an object as if accessing the field/data member directly, without providing public access to the field.
  - Access state without creating a public data member or providing methods, such as get/set.

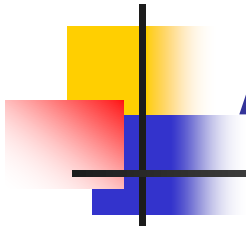
# Defining a property example

- Defining property

```
public class Student {
    private string firstName;

    public string FirstName {
        get { return this.firstName; }
        set { this.firstName = value; }
    }
}
```

# Using a property example

```
using System;

public class Hello
{
   public static void Main()
   {
        var student = new Student();
        student.FirstName = "Sallie";
        student.LastName = "Mae";

        Console.WriteLine("Hello, {0} {1}",
                student.FirstName, student.LastName);
   }
}
```



# Auto-implemented properties

- In C# 3.0 and later, auto-implemented properties make property-declaration more concise *when no additional logic* is required in the property accessors. When you declare a property as shown in the following example, the compiler creates a private, anonymous backing field that can only be accessed through the property's get and set accessors.

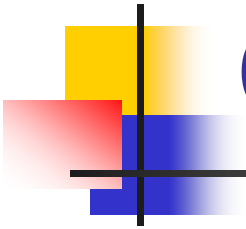
# Auto-implemented property example

- Defining property

```
public class Student {

    public string FirstName { get; set; }

}
```

# Expression-bodied members

- Provide a compact and cleaner way to implement a method or property that consists of a single expression.
  - Will discuss more next class.
- Declared using a lambda expression.
  - `public DateTime CurrentDate => DateTime.Now;`
- Example
  - C# 6.0 gets more concise
- Expression-bodied members

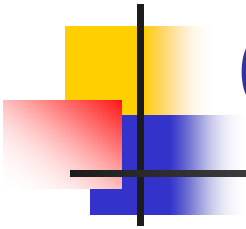


# Coding conventions

- Rules regarding naming conventions, format, etc. of the code.
  - Naming guidelines
  - C# Coding Conventions

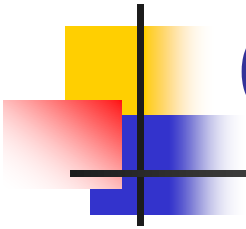
# Accessing Microsoft's documentation

- Within Visual Studio – F1



# On your own

- Create a `Student` data class that has the following:
  - Properties for first name, last name, full name, and grade.
  - Appropriate constructors.
    - Generating constructors with ReSharper code generation.
  - `ToString` method to identify the object.
- Enforcing preconditions
  - if vs. ?? operator when checking for null parameter.
  - ?? operator – null-coalescing operator



# On your own (2)

- Create a `Roster` class that has the following:
  - A collection of student objects.
  - A constructor that adds a few students to the list of students.
    - You can explicitly create and add the student objects to the roster.
  - Functionality to return the roster.
  - Functionality to return the number of students in the class roster.
- In the `Main` method, demo your `Roster` object by printing out the roster.

# Submitting a Visual Studio project

- Make sure to zip Solution file (sln) and the project folder(s).

# Next class

- More C#
- ReSharper
- Discussion of Assignment 1