**Ac Week 1 - Tuesday**

**Revature Associate Experience**

- Time spent after class (studying) - dependent on the person
- Project time - require **a lot** of time investment
- Quiz - mostly MC's, but some short answer
- Bring paper/pen
    - Need them for diagrams
- Reproduce the stuff that we did in class for reviews
- Stay on top of the vocabulary

**Staging Manager:** Julie (julie.seals@revature.com)

- Staging will occur after training (10 weeks)
- Will have interviews towards the end of the 10 weeks and a bit after
- Go to person for any questions you have during the staging process
- Week 11/12 - Remote Staging (Stay at Arlington)
- Will have to go to Tampa, Florida if you don't get placed within the 2 weeks
- Need to stay at a Revature location until placed with a client
- **Virtual Staging** - where we can work from home
    - Placed with client and passed background check just waiting for the start date
    - Need to check in once a day and must be readily available
    - Will be paid associate pay (minimum wage)
- **Panel** - comprehensive technical interview
- Average assignment time is 1 year at which point they will have the option to buy you out
- In-between project will drop you down back to associate pay

**C# - Everything we will learn this week**

- **Anatomy of code** - language compiler, runtime, platform
- **Environment setup** - IDE, editor, version control
- **Basic topics**- core C#, program structure, testing, logging
- **.net Building blocks** - Framework, Standard, core, project, solution, assembly, library application
- **Common language runtime** - BCL, CIL, CLI, CLR, CTS, JIT, VES
- **Runtime Environment** - garbage collection, managed, unmanaged
- **Data types** - reference, value,
- **Access modifiers** - internal, private, protected, public
- **Extended modifiers** - abstract, const, new, override, partial, read only, sealed, static, virtual
- **Class** - constructor, field, method, property, references
- **Struct** - value type
- **Interface**
- **Enum**
- **Semantic code** - DRY, inline/XML, comments, separation of concerns, KISS

- **Object Oriented Programming (OOP)** - abstraction, encapsulation, polymorphism, inheritance
- **Working with Types** - casting, as, boxing, is, out, ref, typeof, generics
- **Collections** - array, list, set, dictionary, stack, queue
- **Serialization** - file, I/O, regular expression, JSON, XML
- **Exception handling**, try, watch, finally, custom exception
- **Testing** - unit testing, xUnit, Fact, theory, TDD
- **Debugging** - breakpoint, step, logging log level
- **SOLID**
- **Delegates** - func, action, event, lambda, LINQ
- **Multithreading** - task, await, async, thread
- **Git** - add, commit, log, pull, push, status, clone

**Things we covered today**

- Types & Variables
- Operators
- Control Flow Statement
    - For loop
- Loops
    - if/else, while, do while, switch
- Classes & Inheritance
- Access Modifiers
    - Public & Private
- Properties vs Methods
- Interfaces

**Solution** is a container for some related projects

- Project - unit of compilation and deployment
    - Namespace - naming container for classes
        - Class
            - Properties (data)
                - Methods (behavior)
                    - Variables

**Two Projects**

- Console app
    - Application has a plain method
        - Starting point of run
- Class library
    - Library has no main method
    - cannot run except in that same application uses it

**To create another program,**

1. right click on “Solution”
2. Click on “Add” then “New Project”
3. Click on “Class Library”

**To add a class**

1. Right click on C#
2. “Add”
3. “New”
4. “Class”

**To add a folder**

1. Right click
2. “Add”
3. “New Folder”

**To add a interface**

1. Right click
2. Add
3. New Interface
    a. Name should begin with an “I”

In a interface, all members (method, properties, etc.) must share the access of the whole interface

**How to implement an interface:**

```
// "rectangle implements IShape interface"
// this means, the Rectangle
public class Rectangle : IShape
```

**Purpose of interface:**
Can write “generic” methods that can be used for various classes

**Purpose of modifiers:**
We use access modifiers for maintainability (+ security)

For private access modifiers, put a underscore (“_”) before the name

**The benefits of using properties:**

1. We can access them using better syntax.
2. Much shorter syntax due to “auto-property”

```
public double X { get; set; }      // auto property in action
public double Y { get; set; }
```

**Method vs Property**

- Area is a property of circle so it makes sense for it to be a property
- More complicated things are usually methods
- r.area() vs r.area
- “=>” is called “arrow” or “goes to” and it used with “getter”

**Shape Inheritance**

- Area is a property that does the (Length * Width) computation when called upon
    - When talking about the third way to find the area

**Inheritance**

```
// circle inherits all of the behavior of shape
public class circle : shape {
}
```

**Overriding the behavior of parent class**

- In C#, by default, you cannot override, you can only add new properties/methods/etc.
- To bypass this, we need to add the “virtual” modifier in the parent class to allow override
- We can always access the parent class’s implementation with “base”

```
// here is an example of an override
// method to find the area
public virtual double Area() {
      return x * y;
}

// override
public override double Area() {
      Console.WriteLine(Area());
}
```

**Access Modifiers (https://bit.ly/2ITn6pZ) - need to know these five**

- **Public** - accessible from anywhere in program
- **Internal** - accessible by any code in the same assembly
- **Private** - only within the class
    - NOT EVEN SUBCLASS HAS ACCESS
- **Protected** - can be accessed by this class and derived classes
- **Protected Internal** - accessible by any code in the assembly in which it is declared, or from within a derived class in another assembly.

**Types** (classes, interfaces, structs, etc)

- **Makes sense:** public and internal
- **Doesn't make sense:** protected and private
    - Since you can't use it anywhere else

**Members** (properties, methods, fields, etc.)

- **Makes sense:** protected and private

**Definitions**
**Class** - is a template to create objects
**Object** - bundle of data and behavior that goes well together
Class : Objects == 1 : N

- Circle - was the parent/base/super class
    - Better Circle
        - Colored Circle
        - Noisy Circle

**Interfaces** - a contract or role that class(es) can fulfill

**Things to Review**

- Compare "Circle" to the "BetterCircle" ("BetterCircle" uses properties)
- Interface
- Properties
- Abstract Base
- Create a Calculator
- Get more familiar with VS Code

**Tomorrow**
CLR
.NET Platform
Source code to how it runs on computer

**Study ahead for**
Reference Types
Value Types
Casting
Collection (arrays, list, generics)