



# Module 1 Day 13

Abstract Classes

# Little bits of Coolness

- Padding strings for alignment; justifying interpolated strings

`{<interpolationExpression>[,<alignment>][:<formatString>]}`

- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated>
- Format strings: <https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types>
- ToString() override

# Abstract Methods and Classes

- Abstract Method
  - Superclass provides **no implementation**
  - Subclass **must** implement the method
- Abstract Class
  - A user cannot create an instance of this class
  - **Only subclasses** can be instantiated
    - Should it really be possible to create a new FarmAnimal?
    - What does a FarmAnimal look like? What sound does it make?
  - Some (or even all) of its methods **may** have implementation
  - The opposite of *abstract* is *concrete* in this context
- If a class has an abstract method, then it **must** be an abstract class

# Classes, Abstract Classes, Interfaces, Oh My

| Concrete Class                                                                                                                          | Abstract Class                                                                                                                                                                         | Interface                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Class Inheritance (max 1 class)                                                                                                         | Class Inheritance (max 1 class)                                                                                                                                                        | Interface Implementation (many)                                                                                                                 |
| General → Specialized (is-a)                                                                                                            | General → Specialized (is-a)                                                                                                                                                           | Functionality (can do)                                                                                                                          |
| Implementation code (all)                                                                                                               | Implementation code (some or all)                                                                                                                                                      | No code, just a contract                                                                                                                        |
| May contain public, protected and private members                                                                                       | May contain public, protected and private members                                                                                                                                      | Public members only                                                                                                                             |
| <b>Can</b> create an instance                                                                                                           | <b>Cannot</b> create an instance                                                                                                                                                       | <b>Cannot</b> create an instance                                                                                                                |
| Use when there is a specialization relationship (a true is-a), and the class represents a real-world thing that can exist (e.g., a Pig) | Use when there is a specialization relationship (a true is-a), but the class represents something that doesn't make sense to exist without being further defined (e.g., a Farm Animal) | Use when there is a need to make a class "behave like" or "can do" some additional functionality; when there is not a true "is-a" relationship. |

# Classes, Abstract Classes, Interfaces, Oh My

- Knowing what we know now, how would we design...
  - BankAccount
    - SavingsAccount
    - CheckingAccount
  - FarmAnimal
    - Pig
    - Chicken
  - Clock
    - Grandfather Clock
    - Alarm Clock
    - Coffee Maker
    - Oven
    - Microwave




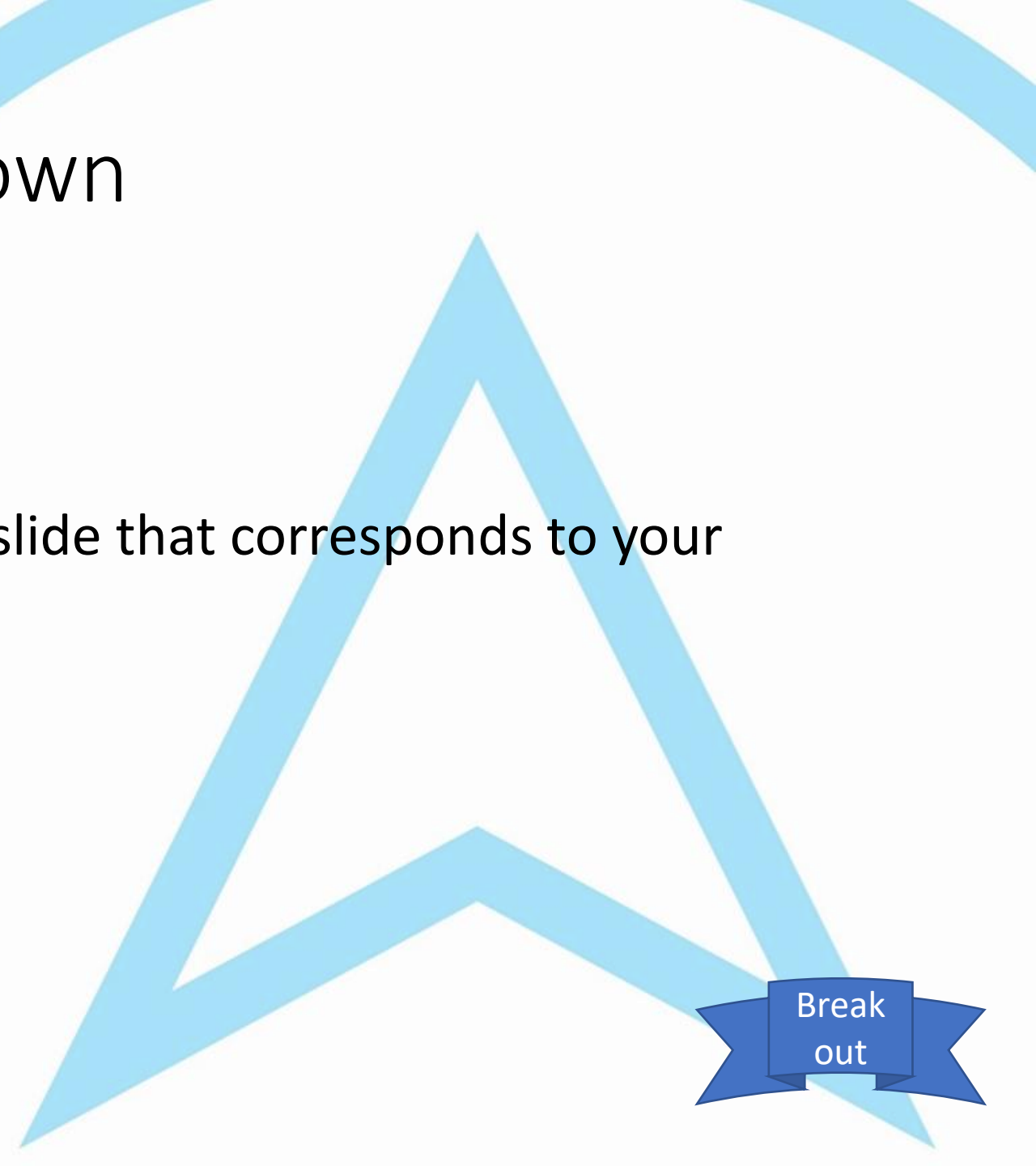
Let's  
Code

# Communicating Design Intent

- Sealed
  - On a class, prevents the class from being sub-classed
  - On a method, prevents further overrides by subclasses
    - This would only be used alongside *override*
- Access modifiers – when to use public, protected, private
  - Easier to give than to take away

# Shapes – Code Breakdown

- Demo
- Breakout rooms
- 15 minutes, with a partner or two
- Address the question on the next slide that corresponds to your breakout room number
- We'll discuss as a group



Break  
out



# Shapes – Code Breakdown

1. What is the purpose of Program.Main? Where is the meat of the UI logic?
2. Explain the overall purpose of DrawingManager. Explain how the Run() loop works.
3. What is a Shape2D? What classes currently derive from Shape2D?
  - Which methods and properties **may** be overridden by a class derived from Shape2D?
  - Are there any that you think **must** be overridden to make sense?
4. Where are the Circles and Rectangles stored as the drawing is built? What type is this collection?
5. What methods / properties does the Circle override from its ancestor classes?
  - What “specialization” (additional properties/methods) has been added to Circle?
6. What methods / properties does the Rectangle override from its ancestor classes?
  - What “specialization” (additional properties/methods) has been added to Rectangle?
7. Describe the relationship between the Shape2D constructor and the constructors of the Circle and Rectangle.
8. Explain the code in DrawingManager that *draws* all of the shapes.
9. Explain the code in DrawingManager that *lists* all of the shapes.
  - How does simply printing out “shape” print a detailed description of the shape?



# New Feature Request

- In addition to shapes, we need to capture, store and print text labels on our drawing.
  - How should we implement this? Are these shapes (is-a)?

# New Feature Request

- We are also asked to determine the total area of all the shapes. Can we do that?