



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 Shape2D?
 - What does a Shape2D look like?
 - Can we Draw it? Get its Area?
 - 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
Can create an instance	Cannot create an instance	Cannot 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
 - Shape2D
 - Circle
 - Rectangle
 - Clock
 - Grandfather Clock
 - Alarm Clock
 - Coffee Maker
 - Oven
 - Microwave

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