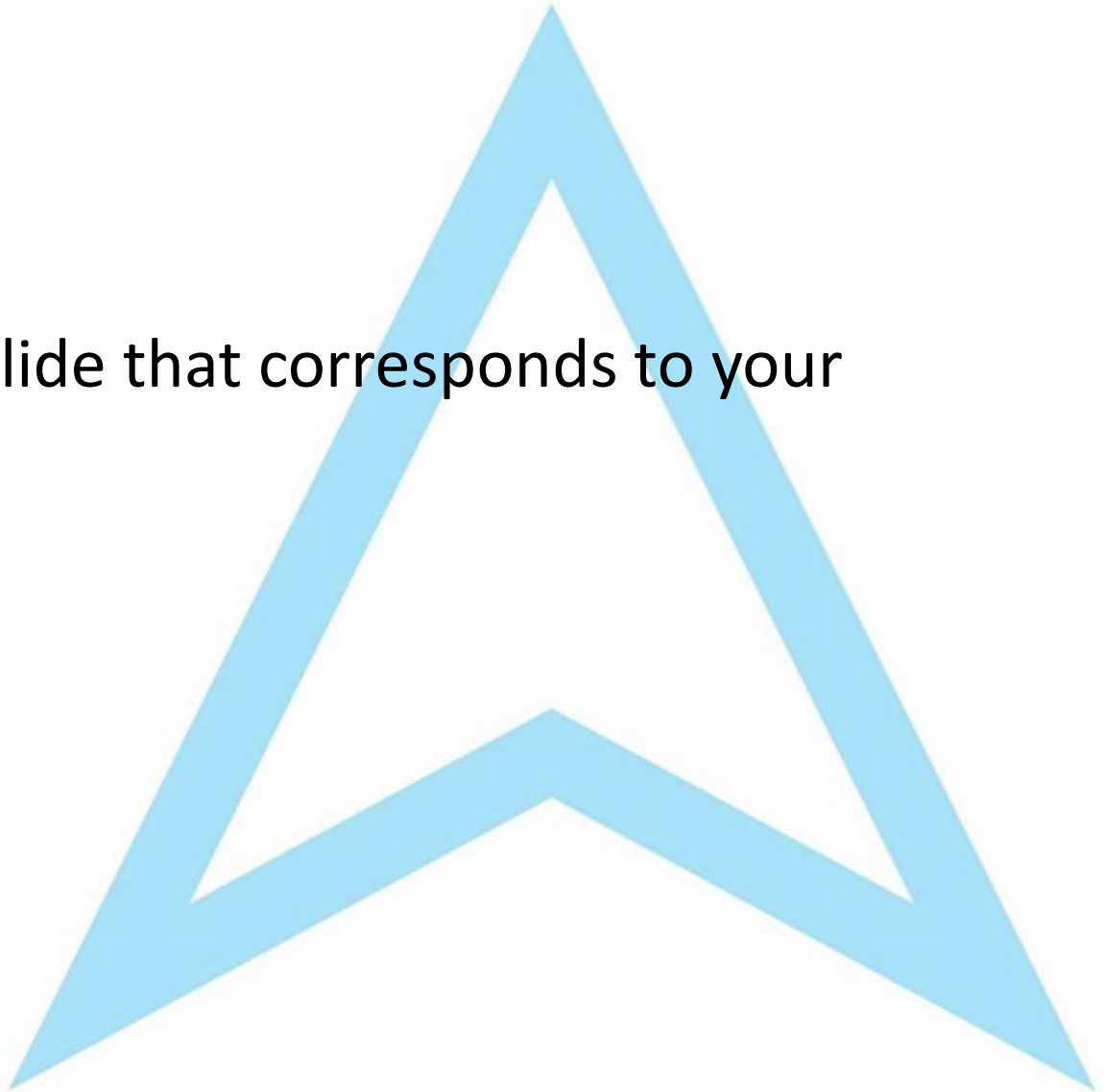
Module 1 Day 12

Polymorphism and Interfaces

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



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?

Polymorphism

- “Many forms”
- Two distinct aspects:
  - If B is a subclass of A and a function can accept A as a parameter, then it can also accept B. If we have a collection of A, we can also store B in the collection
  - Subclasses can override methods defined on the superclass, and the appropriate method gets invoked based on the Type of the target object
- You cannot talk Polymorphism without talking Inheritance

Interfaces

- Defines a contract between a class and its user
- Defines the **public** properties and methods, but NEVER the implementation
  - No need for access modifiers because all members are *public* by definition
- Classes which inherit the interface MUST provide the implementation
  - For **ALL** its methods
- Class inheritance ➔ “is a”; Interface inheritance ➔ “Can do”

Interfaces

- All members of an interface are public
- A class can
  - derive from ONE class
  - Implement MANY interfaces
- Polymorphism works with Interfaces!
  - If B is a class that *implements* interface IA and a function can accept IA as a parameter, then it can also accept B