Inheritance: The "Is-A" Relationship

- **Concept**: A new class (derived/subclass) acquires properties and behaviors from an existing class (base/superclass).
- **Models**: "Is-A" relationship (e.g., `Dog` IS-A **`Animal`**).
- **Benefits**:
  - Code Reusability
  - Extensibility
  - Maintainability

Inheritance Syntax

- **Base Class**:

```
class BaseClass { /* ... */ }
```

- **Derived Class**:

```
class DerivedClass : BaseClass { /* ... */ } // ':' indicates inheritance
```

- **Single Inheritance**: C# supports only single inheritance (one base class).

Access Modifiers and Inheritance

- **`public`**: Inherited and accessible.
- **`protected`**: Inherited and accessible within base and derived classes.
- **`private`**: Not accessible by derived classes.
- **`internal`**: Inherited and accessible within the same assembly.
- **`protected internal`** / **`private protected`** (brief mention)

Constructors in Inheritance

- Derived class constructors implicitly call base class's default constructor.
- **`base()` keyword**: Explicitly call a specific base class constructor.
- **Example**:

```
public Derived(int x) : base(x) { /* ... */ }
```

Method Overriding - Subtype Polymorphism Run-time Polymorphism

- **Purpose**: Derived class provides specific implementation for a base class method.
- **Requirements**:
  - Base method: **`virtual`** keyword.
  - Derived method: `` `override` `` keyword.
- **Behavior**: Runtime polymorphism (actual object type determines method called).
- **Example**:

```
class Shape { public virtual void Draw() { /* ... */ } }
class Circle : Shape { public override void Draw() { /* ... */ } }
```

Method Hiding (using `new`)

- **Concept**: Derived class defines a method with same signature as base, but base method is ***not*** **`virtual`**.
- **Keyword**: **`new`** (explicitly hides base method).
- **Behavior**: Compile-time polymorphism (declared type determines method called).
- **Caution**: Generally less desirable for polymorphic behavior.
- **`sealed` keyword**: Prevents further overriding of a method or inheritance of a class.

Abstract Classes

- **Concept**: Cannot be instantiated directly; serves as a base class.
- **Characteristics**:
  - Declared with **`abstract`** keyword.
  - Can contain **`abstract`** (no implementation) and concrete members.
  - Abstract members ***must*** be implemented by non-abstract derived classes.
  - Cannot be **`sealed`**.
- **Purpose**: Define common interface and partial implementation for a hierarchy.
- **Example**:

```
abstract class Shape { public abstract double Area(); }
```

Interfaces

- **Concept**: A contract specifying a set of members a class must implement.
- **Characteristics**:
  - Declared with **`interface`** keyword (e.g., **`IDrawable`**).
  - Cannot be instantiated.
  - No fields or constructors.
  - All members implicitly **`public`** and **`abstract`** (pre C# 8).
  - A class can implement multiple interfaces.
- **Purpose**: Define capabilities, support loose coupling.
- **Example**:

```
interface IDrawable { void Draw(); }
```

Abstract Class vs. Interface

Feature	Abstract Class	Interface
Relationship	"Is-A"	"Can-Do"
Instantiation	Cannot be instantiated	Cannot be instantiated
Members	Abstract & Concrete	Abstract (pre C# 8), Default Impl. (C# 8+)
Fields/Ctors	Can have	Cannot have
Inheritance	Single inheritance	Multiple implementation

Polymorphism: "Many Forms"

- **Concept**: Ability of an object to take on many forms.
- **Types**:
  - **Compile-time (Method Overloading)**: Determined at compile time.
  - **Runtime (Method Overriding)**: Determined at runtime based on actual object type.
- **Key for flexible, extensible code.**

Runtime Polymorphism in Action

- **Scenario**: Treat objects of different derived classes as objects of their common base type.
- **Example**:

```
Shape myShape = new Circle(); // myShape is declared as Shape, but is a Circle
myShape.Draw(); // Calls Circle's Draw() method
```

- **Demo Code**:

```
Shape[] shapes = new Shape[2];
shapes[0] = new Circle();
shapes[1] = new Rectangle();
foreach (Shape s in shapes)
    s.Draw();
```

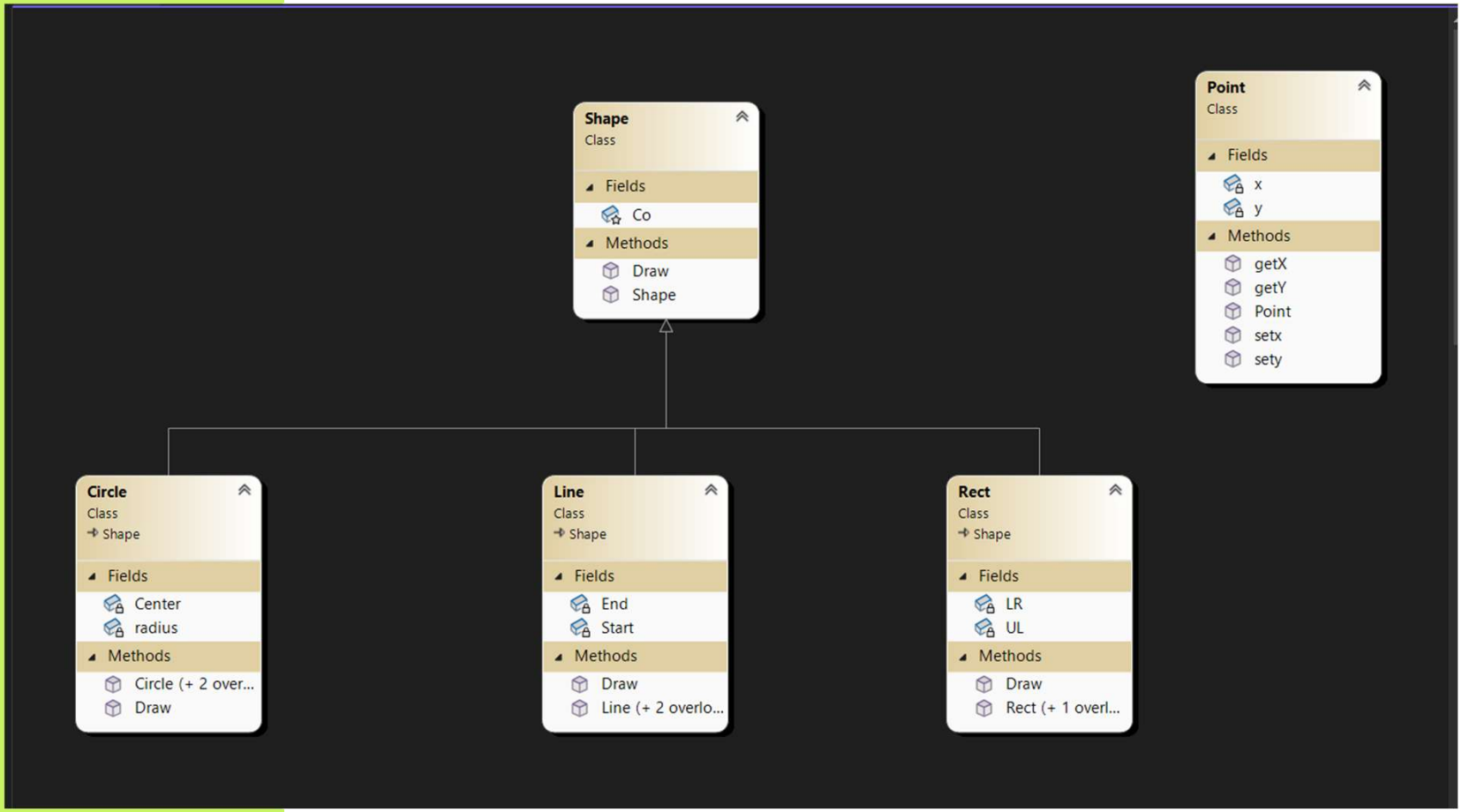
Day 3 Recap

- **Inheritance**: Code reuse, "Is-A" relationship.
- **Overriding**: `virtual` and `override` for runtime polymorphism.
- **Abstract Classes**: Partial implementation, enforce derived class behavior.
- **Interfaces**: Contracts, "Can-Do" relationship, multiple implementation.
- **Polymorphism**: Key to flexible and extensible OOP design.

Q&A and Next Steps

- **Questions**?
- **Tomorrow**: Collections and Generics!

Assignment


Shape
Class
Fields
Co
Methods
Draw
Shape
Point
Class
Fields
x
y
Methods
getX
getY
Point
setx
sety
Circle
Class
Shape
Fields
Center
radius
Methods
Circle (+ 2 over...
Draw
Line
Class
Shape
Fields
End
Start
Methods
Draw
Line (+ 2 overlo...
Rect
Class
Shape
Fields
LR
UL
Methods
Draw
Rect (+ 1 overl...