IOT 1026
Object Oriented Programming

Week 9 – Class 1

Manmeet Singh Duggal

8th July 2025

Important Dates

- Applied Activity #4 is due today (Jul. 8*th*).

This week…

- Interfaces

Interface

- An interface is also used for the abstraction of data
- Interfaces are fully abstract classes (all methods have empty bodies)
- This means that they:
  - cannot have any data fields
  - cannot have any non-abstract members
  - all methods must be abstract
  - cannot have a constructor
- Interface members are by default public and abstract

Interface

- If you want to access an interface’s methods, you need to implement the interface
- Implementation uses the same notation “ : ” as inheritance.

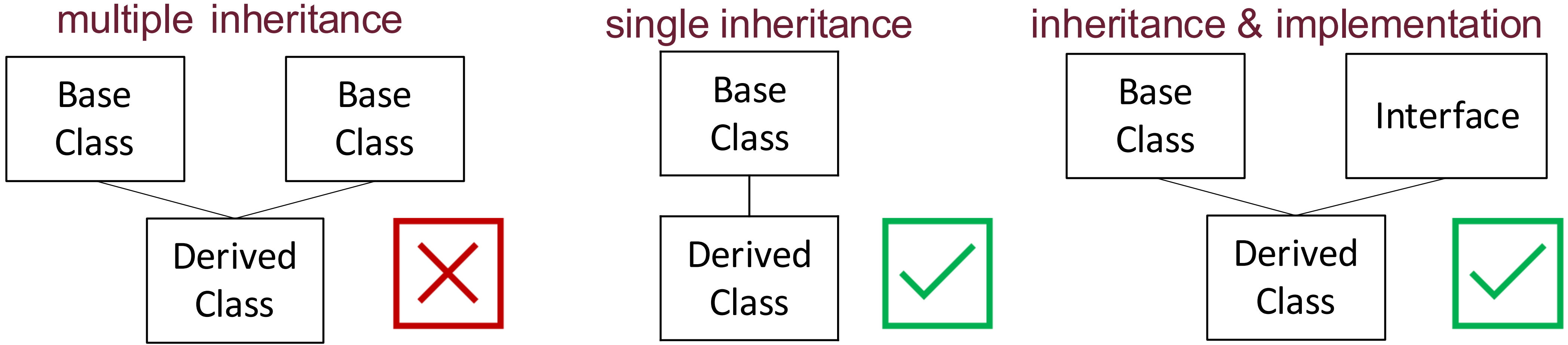
  Dog : IAnimal
- But when you implement an interface’s method you don’t need the keywork “override”
- All Interface methods bodies must be included in the class that implements the interface
- Like Abstract classes, interfaces can’t be be made into instances (objects)

Interface

```
interface IAnimal{     //interface names usually start with ‘I’
      void makeNoise();      //default public abstract
}
//Cat class implements IAnimal interface (not inherits)
class Cat : IAnimal{
      public void makeNoise(){    //doesn’t use override
            Console.WriteLine(“Meow!”);
      }
}
class Dog : IAnimal{
      public void makeNoise(){    //doesn’t use override
            Console.WriteLine(“Woof! Woof! Woof!”);
      }
}
```

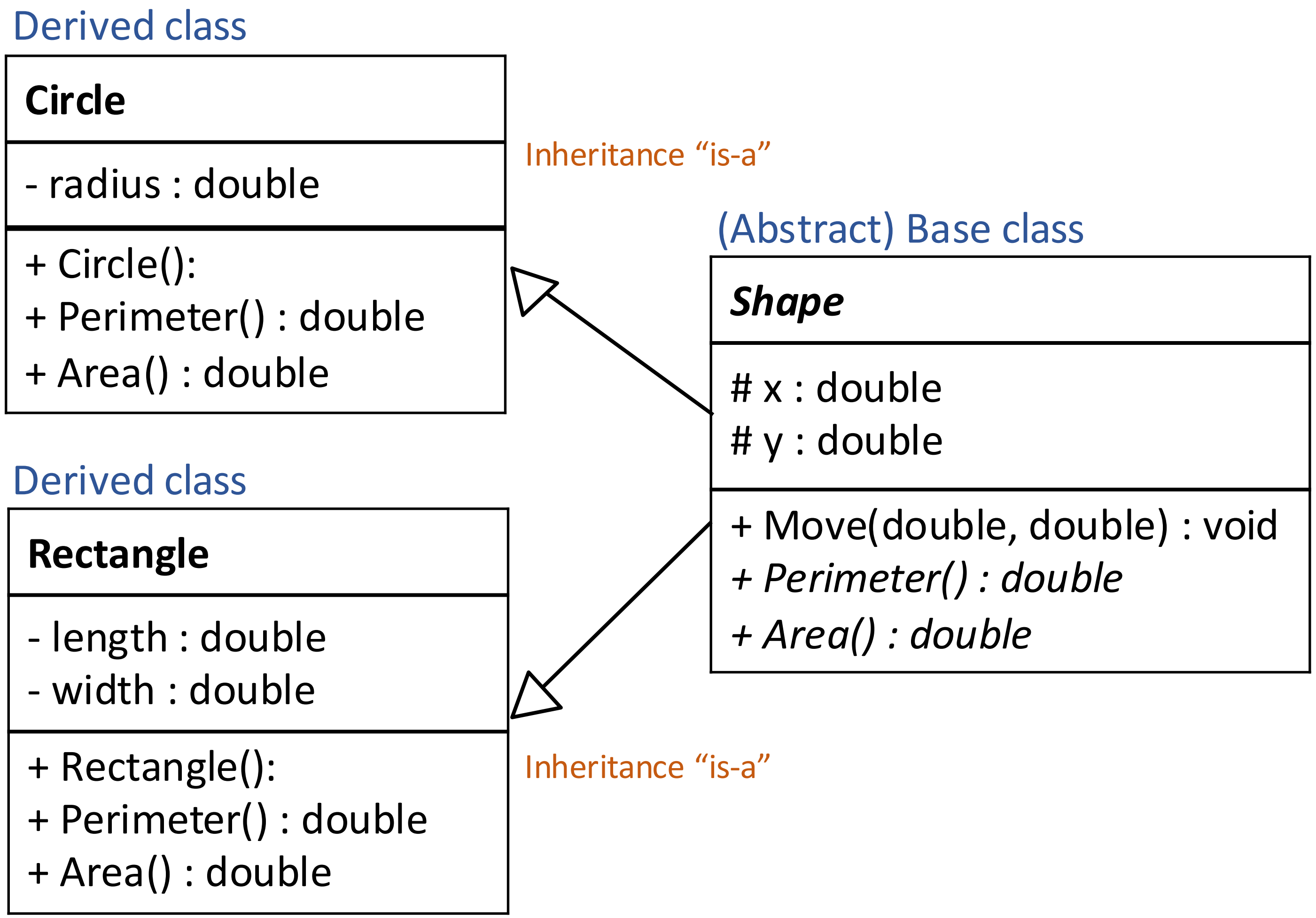
Interface - Why use an interfaces?

- 1) Similar role as abstract classes. To improve security by hiding certain details of an object.
- 2) In C#, a class can only inherit from one base class. However, it can implement multiple interfaces and achieve the same goal. *Note: To implement multiple interfaces, separate them with a comma. E.g.* `Cat : IAnimal, IAlive`



Live Codding

UML – Shape Abstract Class



Derived class

Circle

- radius : double

+ Circle():
+ Perimeter() : double
+ Area() : double

Inheritance “is-a”

(Abstract) Base class

Shape

# x : double
# y : double

+ Move(double, double) : void
+ Perimeter() : double
+ Area() : double

Derived class

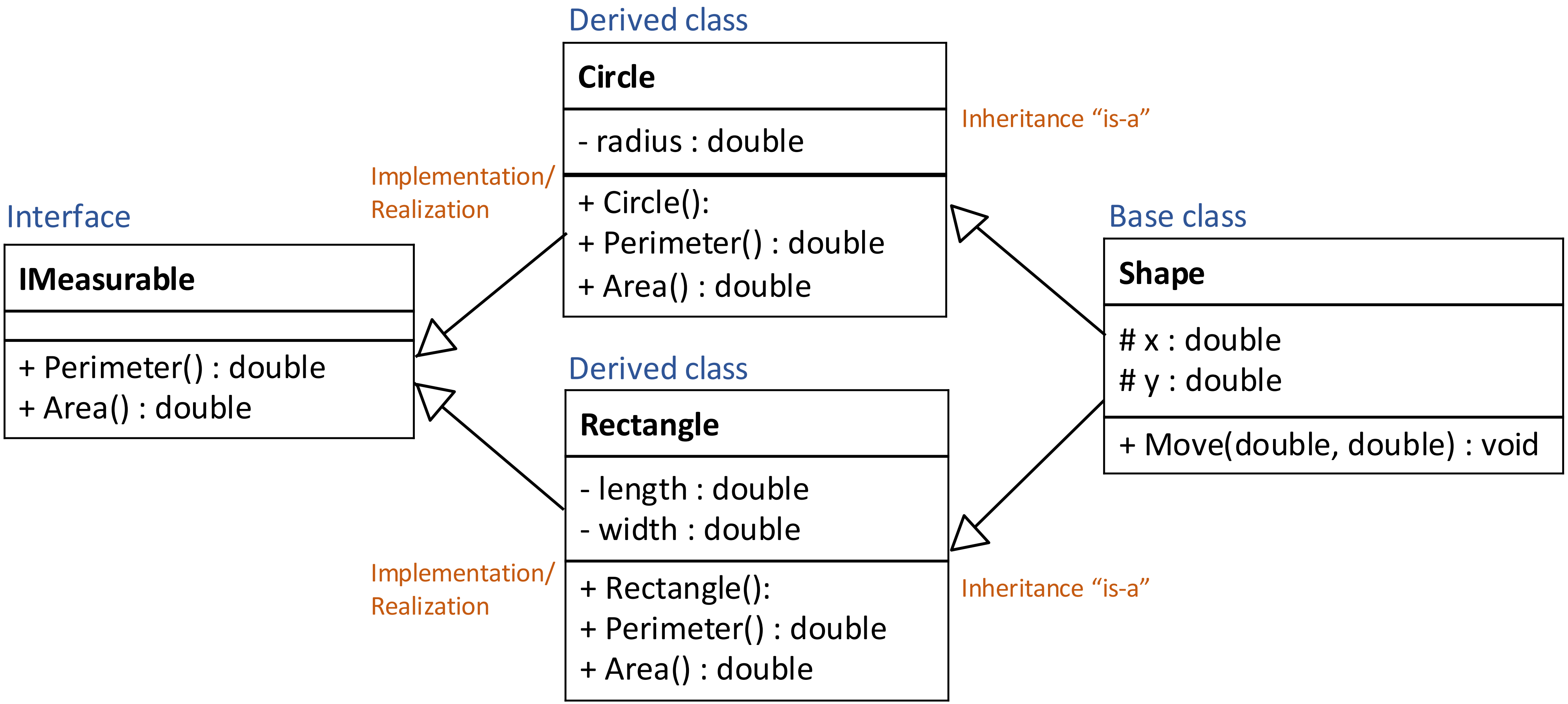
Rectangle

- length : double
- width : double

+ Rectangle():
+ Perimeter() : double
+ Area() : double

Inheritance “is-a”

UML – Shape Base Class & Measurable Interface



Derived class

Circle

- radius : double

+ Circle():

+ Perimeter() : double

+ Area() : double

Inheritance “is-a”

Implementation/
Realization

Interface

IMeasurable

+ Perimeter() : double

+ Area() : double

Base class

Shape

# x : double

# y : double

+ Move(double, double) : void

Derived class

Rectangle

- length : double

- width : double

+ Rectangle():

+ Perimeter() : double

+ Area() : double

Implementation/
Realization

Inheritance “is-a”