COMP9103: Software Development in Java

W7: Abstract class& Interface

Presented by
Dr Ali Anaissi
School of Computer Science

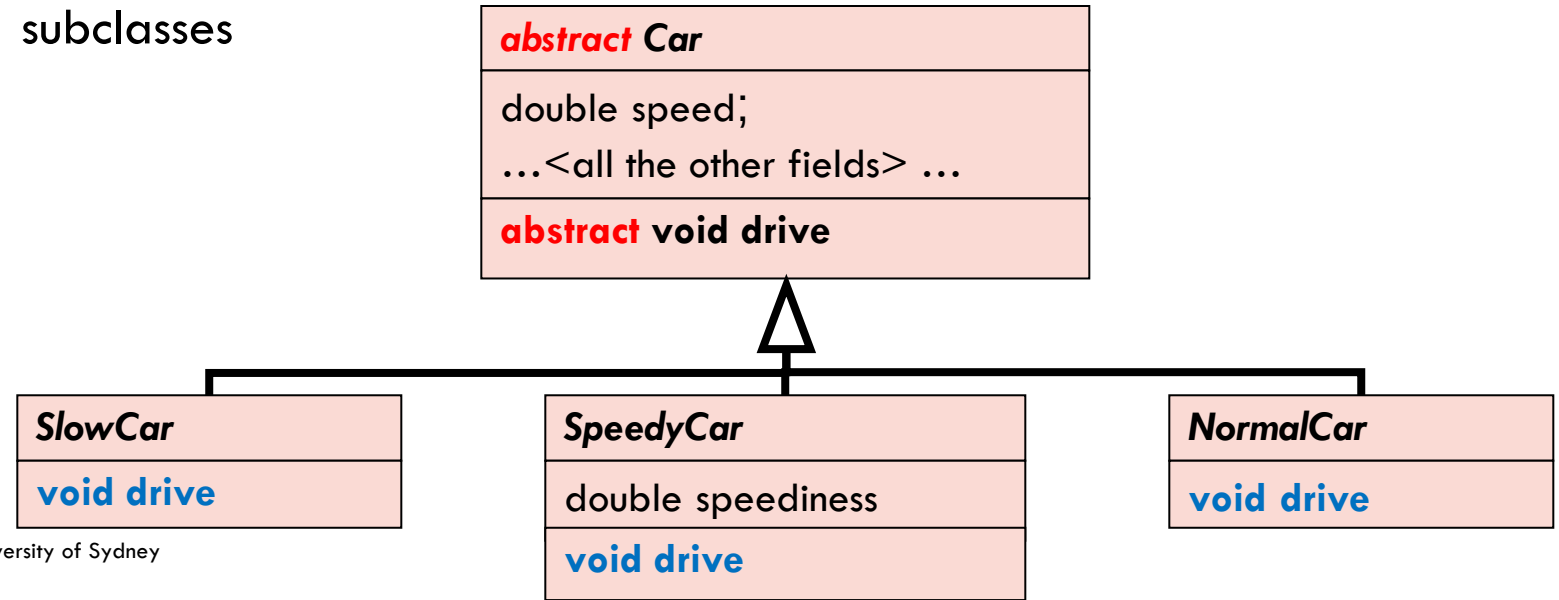
THE UNIVERSITY OF SYDNEY

Abstract Class

THE UNIVERSITY OF SYDNEY

Abstract methods and classes

- An **abstract method** is a method only has method header/signature but without method body
- Classes with abstract method(s) are defined as ***abstract*** classes.
- The concrete implementation of an abstract method is to be specified in the subclasses



Abstract methods and classes

- An ***abstract method*** is a method whose implementation is not specified
  - The concrete implementation of an abstract method is specified in the subclasses
- Car class, version 2

```
public abstract class Car {
   //… <instance and static fields as before> …
   //…<getters and setters as before> ….
   //… <other methods as before> ….

   public abstract void drive (double time);

}
```

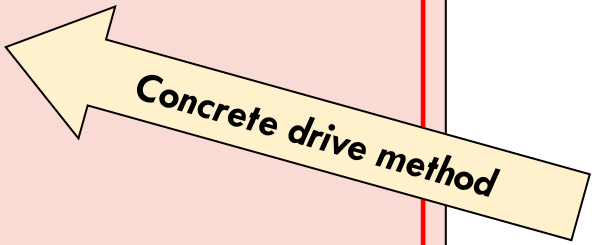
Abstract class

Abstract method:
No method body-
even without { }

Abstract methods and classes

- NormalCar class

```
public class NormalCar extends Car {
...
  public void drive(double time){
    if (getSpeed()<getLowerSpeedLimit()){
        setAcceleratorOn(true);
        setBrakeOn(false);
    }
    if (getSpeed()>getUpperSpeedLimit()){
        setBrakeOn(true);
        setAcceleratorOn(false);
    }
    double a = -getFriction();
    if (isAcceleratorOn()) a += getAccelerationPower();
    if (isBrakeOn()) a -= getBrakePower();
    setSpeed(getSpeed() + time*a);
    setPosition(getPosition() + time*getSpeed());
  }
}
```



Abstract methods and classes

- A class that defines an abstract method, or that inherits an abstract method without overriding it, *must* be declared as abstract class.

- An abstract class is to be EXTENDED

- An abstract class cannot be instantiated.
  - You cannot construct objects from abstract classes.

Inheritance Example

- Consider three classes which represent 2D shapes
  - Circle
  - Rectangle
  - Square

- Each of these classes has
  - A method to calculate the perimeter
    - distance around its edges
  - A method to calculate the area

Class diagrams for Shapes

Circle
-double radius
+double calcPerimeter ( ) +double calcArea ( )

Square
-double side
+double calcPerimeter ( ) +double calcArea ( )

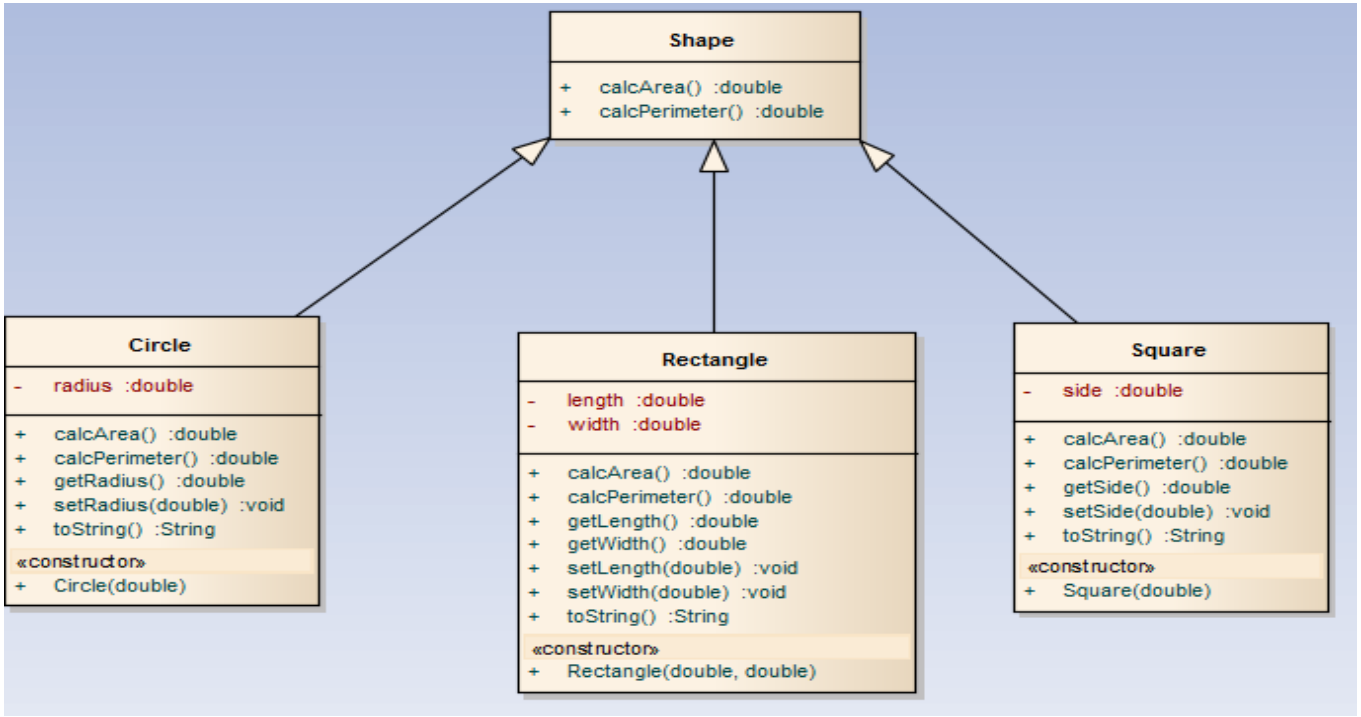
Rectangle
-double length -double width
+double calcPerimeter ( ) +double calcArea ( )

- get/set methods, toString, constructors should be included in class diagram
- left out of slides for space considerations

Example

- The problem becomes even more apparent in the future if we want to add new shapes – eg Triangle
- How can we make our program more flexible?
- We can use inheritance
- Let's create a class called Shape

Inheritance hierarchy



Inheritance

- We can make a Triangle class now and have it extend the Shape class

- However, we can't force it to make its own calcArea() and calcPerimeter() methods
  - It can inherit them from Shape

Inheritance

- Notice we won't ever need to make Shape objects.
- For example,
  - Shape s = new Shape ( );
- This wouldn't make sense – how do we calculate the area and perimeter of a generic shape?
- We can ensure that this can't happen by making the class abstract

Abstract Class

- **abstract class**: a class that has an abstract method

- **abstract method**: no method body, e.g.:

```
public abstract void draw( );
```

- the subclasses provide implementation of an abstract method

- **NO** object can be created from an abstract class.

- Abstract classes ***enforce*** code reuse and software design.

Abstract classes

- Any class that inherits from an abstract class must implement the abstract methods of that class
  - We can force Triangle to make its own versions of the calcArea() and calcPerimeter() methods by making them abstract in Shape

- An abstract method has no body and exists for inheritance purposes only.
  - Only abstract classes can contain abstract methods

Abstract Class

- We saw they are similar to ordinary (concrete) classes

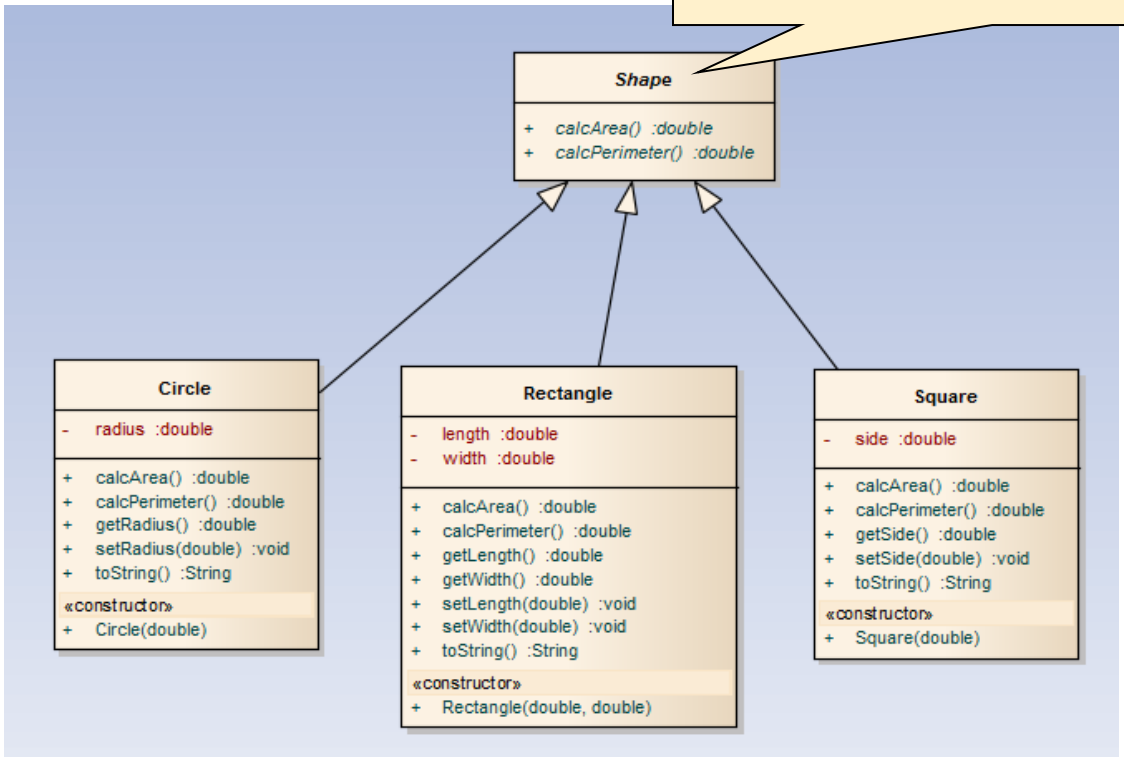
- The main difference is they cannot be instantiated
  - If you attempt to make an object of an abstract class (using 'new') a compiler error will occur

- However they,
  - Belong to the class hierarchy
  - Can contain attributes, methods (including abstract), constructors and other elements of classes

Inheritance hierarchy

Shape now is an abstract class with two abstract methods:

calcArea() & calcPerimeter()



Abstract classes

- An abstract class suffers from the same limitations as concrete classes
- They can only directly inherit from one superclass
- One of their subclasses cannot inherit from another superclass
- This is called single inheritance

Shape hierarchy

- Let's assume we want to be able to draw Rectangle objects and Square objects but not (for some reason) Circle objects

- We could
  - Add an abstract draw ( ) method to Shape and let its subclasses implement it appropriately. However, this means that Circle would have a draw ( ) method and we don't want that

  - Create another abstract class called Drawable that has an abstract method called draw ( ). However, Rectangle can't inherit from both Shape and Drawable so this would not work

Interface

Interface

- We can solve our problem through the use of an interface
  - Interfaces are not the same as Graphical User Interfaces (GUIs) which will be be covered later.

- An interface is a class-like structure which provides a “contract” for objects.

- Interfaces guarantee that an object has implemented certain methods.

Interface

- While interfaces look a little bit like classes they serve quite a different purpose.
- Features of an interface are:
  - Use of the keyword “interface” instead of “class”
  - Can contain only public abstract methods
    - And public static final attributes (class constants)
  - Do not belong to the Java class hierarchy
  - Cannot extend classes but can extend other interfaces
  - Classes do not extend interfaces – they implement them

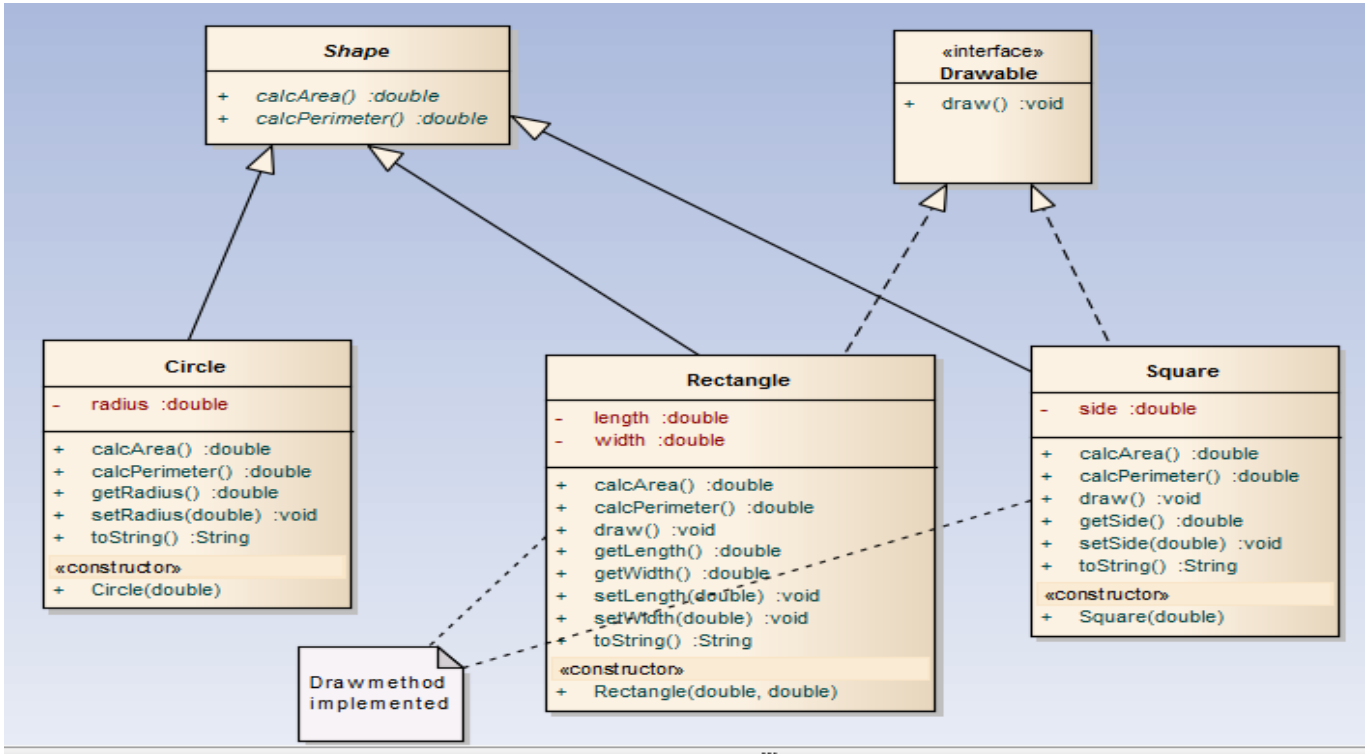
Example

```
public interface Drawable {
        public abstract void draw ( );
}
```

Any class that implements Drawable must contain a concrete method called draw ( );

```
public class Rectangle extends Shape implements Drawable {
                . . . . . .
                public void draw ( ) {
                // code for drawing a Rectangle object
                }
}
```

Inheritance hierarchy- interface



Previous slide

- Shows an example of a class diagram with an abstract class and an interface

- All three concrete classes extend the abstract class
  - Thus they implement the calcArea() and calcPerimeter () methods

- Rectangle and Square both implement the Drawable interface
  - Shown by the dashed line (rather than the unbroken line)
  - Thus they implement the draw () method
  - Circle does not implement draw ( )

Interfaces

What does this achieve?

- Like inheritance, an IS_A relationship is created between the interface and the class that implements it.
- If our Rectangle class implements Drawable, then it IS_A Drawable, as well as being IS_A Shape.
- Consider a method in the ShapeDriver class which will draw the object that is passed to it.
- We will only want Drawable objects passed to this method
    - otherwise we couldn't guarantee the object would have a draw method

```
public void drawItem (Drawable dr) {
        dr.draw ( );
}
```

drawObject

```
public void drawObject (Drawable d) {
        d.draw  ( );
}
```

- This method will allow us to pass Rectangle and Square objects to it, but will not allow Circle objects because Circle does not implement Drawable.

- This is true even if Circle does in fact implement a method called draw ( )

Furthermore

- We don't have to just include classes on the Shape hierarchy.
- We could (for instance) let our Customer class implement Drawable. We would need to implement a method called draw ( ).
- We could then pass objects of class Customer to the drawObject method
- So, by using an interface, we have created a relationship between Rectangle, Square and Customer
  - They are all of type Drawable.

Questions?