IS 2104 - Rapid Application Development

Object Oriented Concepts

Abstraction

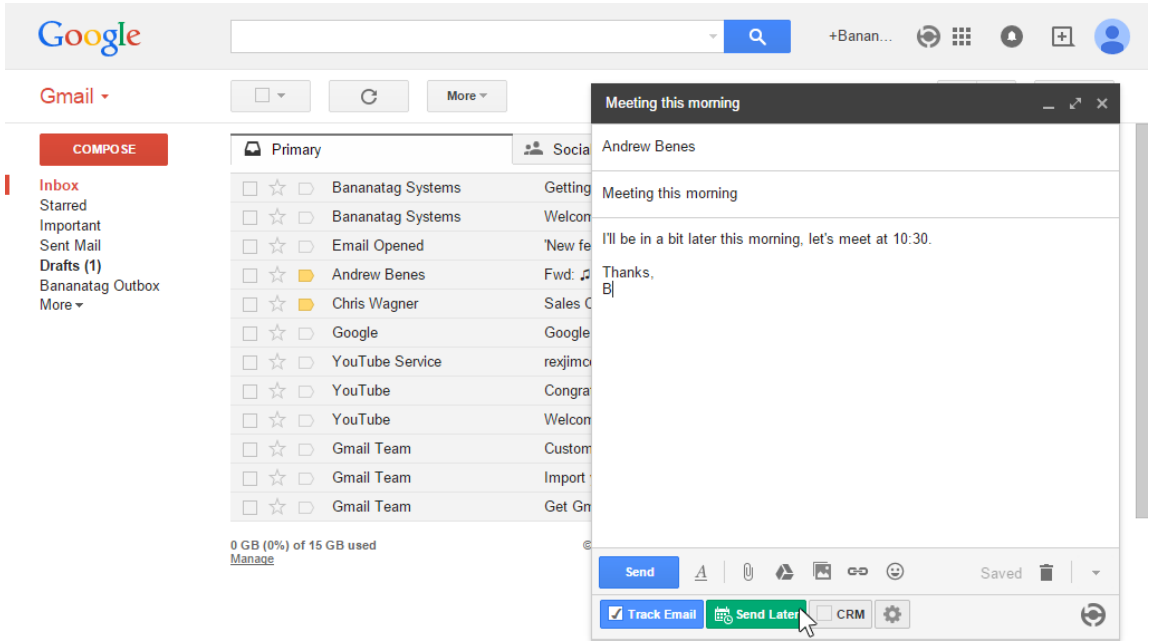
M.V.P. Thilini Lakshika
University of Colombo School of Computing
tlv@ucsc.cmb.ac.lk

Lesson Outline

- What is Abstraction?
- Abstract Class
- Abstract Method
- What is an Interface?

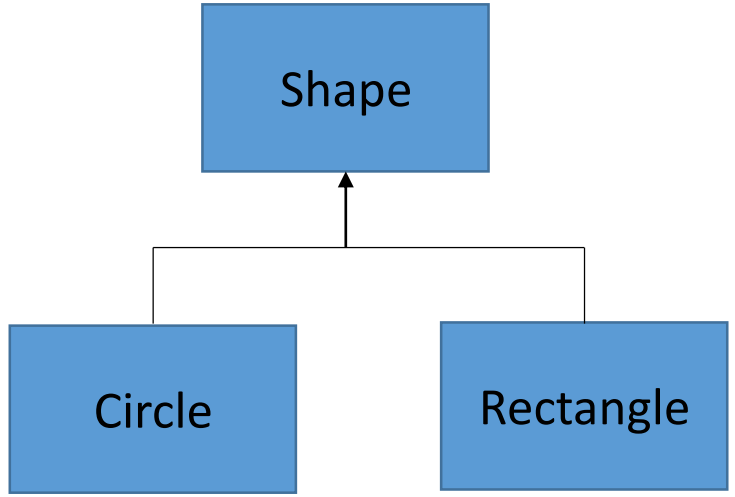
Abstraction

- Consider the case of sending e-mail.
- Complex details such as the protocol your e-mail server uses are hidden from the user.
- Likewise in OOP, abstraction is the process of hiding such implementation details from the user.
- Only the functionality will be provided to the user.
- User will have the information on what the object does instead of how it does it.
- In Java, abstraction is achieved using Abstract classes and Interfaces.



Abstract Class

- In the inheritance hierarchy, classes become more specific and concrete with each new subclass.
- Superclass become more general and less specific.
- Therefore, class design should ensure that a superclass contains common features of its subclasses.
- Sometimes a superclass is so abstract that it cannot have any specific instances.
- Such a class is referred to as an abstract class.



- Abstract classes are like regular classes, but you cannot create objects of abstract classes using the **new** operator.

Abstract Class

- The **Shape** abstract class defines the common features (attributes and methods) for shapes and provides appropriate constructors.

Abstract Class

Protected Constructors

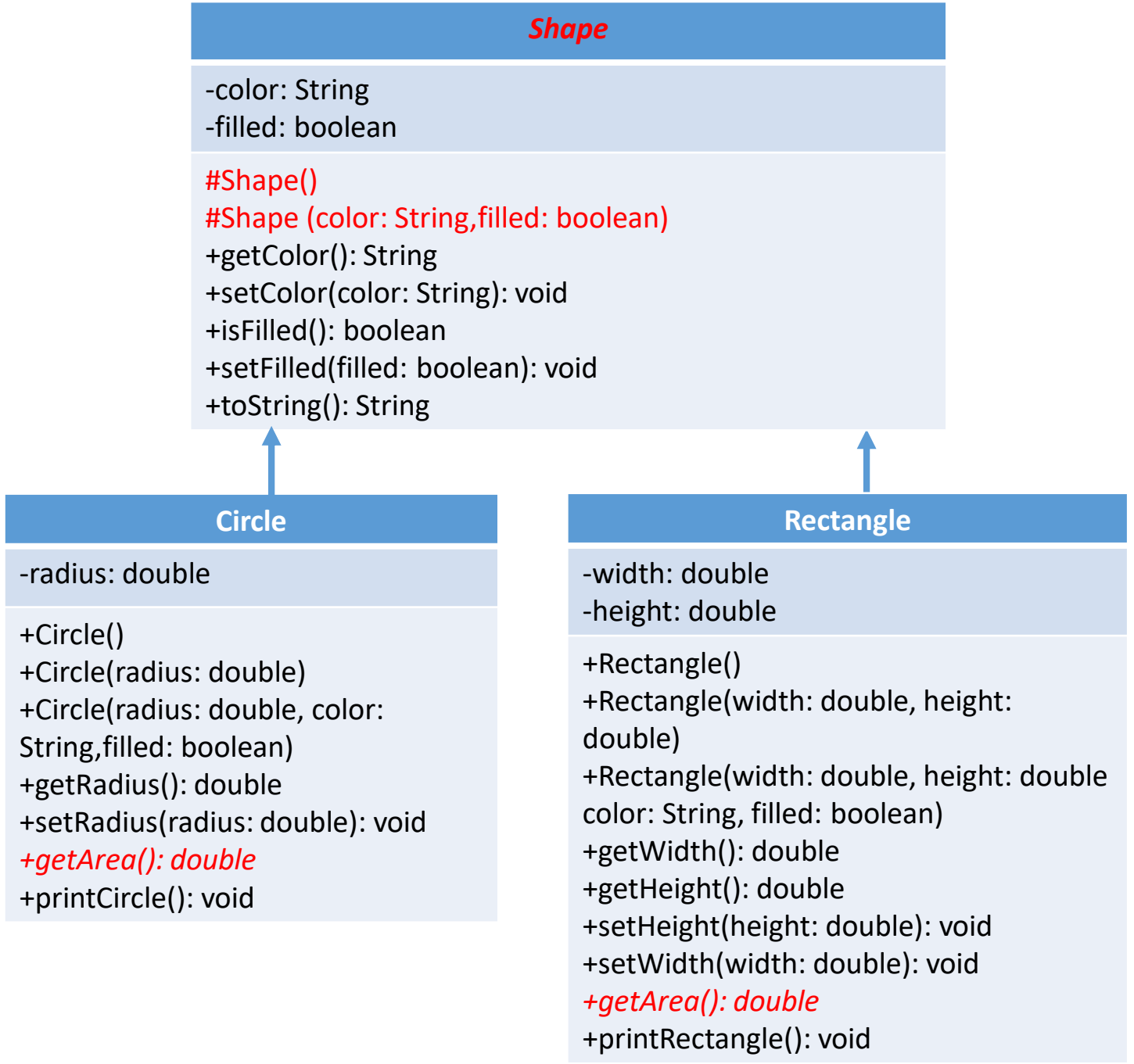
```
1   package lesson12;
2
3   public abstract class Shape {
4       private String color = "white";
5       private boolean filled=true;
6       /** Construct a default shape */
7       protected Shape() {
8       }
9       /** Construct a shape with the specified color and filled value **/
10      protected Shape(String Color, boolean filled) {
11          this.color = color;
12          this.filled = filled;
13      }
14      /** Set a new color */
15      public void setColor(String color) {
16          this.color = color;
17      }
```

Abstract Class

- Constructors in the abstract class is defined protected, because it is used only by subclasses.
- When creating an instance of a concrete subclass, its superclass's constructor is invoked to initialize data fields defined in the superclass.

Abstract Method

- **Shape** class models common features of shapes.
- Both **Circle** and **Rectangle** contain the **getArea()** method for computing the area of a circle and rectangle.
- Since you can compute areas for all the shapes, it is better to define the **getArea()** method in the **Shape** class.



Abstract Method

- However, **getArea()** method cannot implemented in the **Shape** class, because its implementation depends on the specific types of shapes.
- Such methods are referred to as abstract methods.
  - Denoted using the **abstract** modifier in the method header.
- Once you define an abstract method in the **Shape** class, it becomes an abstract class.
  - Denoted using the **abstract** modifier in the class header.
- In UML graphic notation, the names of abstract classes and their abstract methods are italicized.

Abstract Method

- Different shapes compute area using different formulas. So, **getArea()** is defined as an abstract methods.

Abstract Method

```
    }
    /** Return filled. Since filled is boolean,its get method is named isFilled */
    public boolean isFilled() {
        return filled;
    }
    /** Return a string representation of this object */
    public String toString() {
        return " color: " + color + " and filled: " + filled;
    }
    /** Abstract method getArea */
    public abstract double getArea();
}
```

Abstract Method

- An abstract method is declared in an abstract class without implementation.
- The implementation of abstract methods is provided by the subclasses.
- An abstract method will never be final because the sub classes must implement all the abstract methods.
- The implementation of **Circle** and **Rectangle classes** is the same as in Lesson 10 (Inheritance).
- The only change is, those classes extend the **Shape** class defined in this Lesson.

```
public class Circle extends Shape {
    //Same code as in Inheritance example
}
```

Circle.java

```
public class Rectangle extends Shape {
    // Same code as in Inheritance example
}
```

Rectangle.java

Abstract Method

- Abstract Class can have abstract methods as well as concrete methods (well implemented methods).
- But a normal class cannot have abstract methods.
- If a regular class extends an abstract class, then that class must implement all the abstract methods of the abstract parent.

  Ex: Circle and Rectangle class should implement the abstract method getArea() in the Shape class.
- If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined abstract.

  - If a class is using an abstract method they must be declared abstract. The opposite cannot be true.
- An abstract class does not necessarily have an abstract method.

Abstract Method

- What advantage is gained by defining the method getArea as abstract in the Shape class instead of defining them only in each subclass ?

Ex: Creates two shapes, a circle and a rectangle, and invokes the equalArea method and displayShapes method to display them.

equalArea() - Check whether circle and rectangle have equal areas.

displayShapes() - Display the value of area in each shape.

Abstract Method

```
public class TestShapes{
    public static void main(String[] args) {
        Shape shapeObject1 = new Circle( radius: 5);          //create a circle
        Shape shapeObject2 = new Rectangle( width: 5,  height: 3);    //create a rectangle
        System.out.println("The two objects have the same area? " + equalArea(shapeObject1,shapeObject2));
        displayShape(shapeObject1);    // Display circle
        displayShape(shapeObject2);    //Display rectangle
    }
    /** A method for comparing the areas of two shapes */
    public static boolean equalArea(Shape object1,Shape object2){
        return object1.getArea() == object2.getArea();
    }
    /** A method for displaying a shape*/
    public static void displayShape(Shape object){
        System.out.println();
        System.out.println("The area is " + object.getArea());
        }
}
```

Abstraction

Advantages in Abstraction

- Abstraction helps to reduce the complexity of the design and implementation process of software.
- Abstraction allows you to group several related classes.

When to use Abstract Methods & Abstract Classes?

- Abstract methods are mostly declared where two or more subclasses are also doing the same thing in different ways through different implementations.
- Abstract classes help to describe generic types of behaviors and OOP class hierarchy.

Interface

- An interface is treated like a special class in Java.
- It contains only constants and abstract methods.
- In many ways an interface is similar to an abstract class, but its intent is to specify common behavior for objects.
- Interface is a way of describing what classes should do without specifying how.
- Interfaces are not allowed to have any concrete methods (A purely abstract class).
- Interfaces express an aspect of a class other than what it inherits from its parent.

Interface

Syntax:

```
modifier interface InterfaceName {
        /** Constant declarations */
        /** Method signatures */
}
```

Ex:

```
public interface Edible {
/** Describe how to eat */
public abstract String howToEat();
}
```

Interface

- Similar to the abstract classes, you cannot create an instance from an interface using the **new** operator.
- Interfaces are allowed to use as a data type for a reference variable.
- You can now use the **Edible** interface to specify whether an object is edible.
- This is accomplished by letting the class for the object implement this interface using the **implements** keyword.

Ex: The classes **Chicken** implement the **Edible** interface.

```
class Chicken implements Edible {
    public String howToEat() {
        return "Chicken: Fry it";
    }
}
```

Interface

Ex: The **Fruit** class **implements** **Edible** interface.

- Since it does not implement the **howToEat** method, Fruit must be denoted as abstract class.

```
abstract class Fruit implements Edible {
        // Data fields, constructors, and methods omitted here
 }
```

- The concrete subclasses of Fruit must implement the **howToEat** method.

```
class Apple extends Fruit {
   public String howToEat() {
     return "Apple: Make apple cider";
   }
}
```

```
class Orange extends Fruit {
   public String howToEat() {
     return "Orange: Make orange juice";
   }
}
```

- When a class implements an interface, it implements all the methods defined in the interface with the exact signature and return type.

Summary

- In general a strong is-a relationship clearly describes a parent-child relationship.

  - Those should be modeled using abstract classes.

- A weak is-a relationship (is-kind-of relationship) indicates that an object possesses a certain property.

  - Those modeled using interfaces.

- In general, interfaces are preferred over abstract classes because an interface can define a common super type for unrelated classes.
- An interface can be used more or less the same way as an abstract class, but defining an interface is different from defining an abstract class.

Exercise

- Summarizes the differences between abstract class and interface.

END