### **Reading**

This material is covered in Chapter 13 of book.

### **Abstract Classes**

As we move from subclass up to superclass, classes become more general and less specific.

Superclasses should only contain common features.

If you move up the class hierarchy enough, superclass become so abstract that it can't be instantiated.

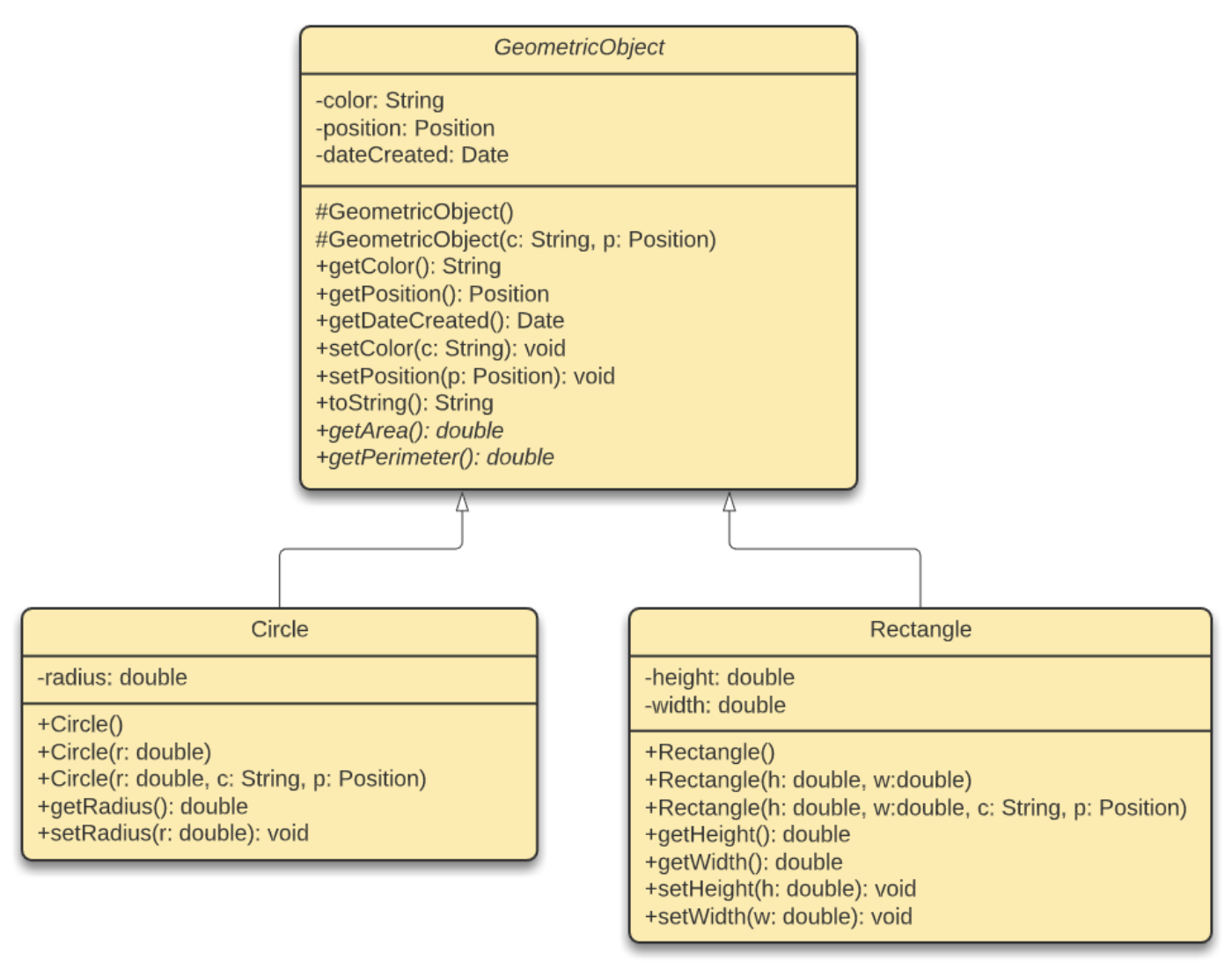
Such a class is called an **abstract class**.

Both Circle and Rectangle have getArea() and getPerimeter() methods.

Since all geometric objects have area and perimeters, it is better to have these method the GeometricObject class.

Note the these methods can not be implemented in the GeometricObject class as how to do this depends on the subclass.

Such a method is called an **abstract method** and class containing it is an **abstract class**.

Consider this updated UML:

Note that that abstract methods and classes are represented as italics.

Again, note that the # sign indicates a protected modifier.

The corresponding Java code becomes:

public **abstract** class GeometricObject {

private String color;

private Position position;

...

// Constructors

**protected** GeometricObject() {

dateCreated = new java.util.Date();

}

...

// Methods

public getColor() {

return color;

}

public setColor(String c) {

color = c;

}

...

// Abstract Methods

**public abstract double getArea();**

**public abstract double getPerimeter();**

}

public class Circle **extends** GeometricObject {

...

// Methods

public double getArea() {

return radius \* radius \* Math.PI;

}

...

}

public class Rectangle **extends** GeometricObject {

...

// Methods

public double getArea() {

return width \* height;

}

...

}

Abstract methods and classes are denoted by the **abstract** modifier.

Note that GeometricObject can no longer be instantiated.

No implementation is provided for abstract methods; they are implemented in the subclass.

Constructor in the abstract class is defined as protected as it is only used by the subclass.

### **Using Abstract Methods**

Consider the following example:

public class TestProgram {

public static boolean equalArea(GeometricArea object1, GeometricArea object2) {

return object1.getArea() == object2.getArea();

}

public static void main(String[] args) {

GeometricObject obj1 = new Circle(5);

GeometricObject obj2 = new Rectangle(5, 3);

if (equalArea(obj1, obj2))

System.out.println("Objects have the same area.");

else

System.out.println("Objects do not have the same area.");

}

}

Variables obj1 and obj2 are of type GeometricObject.

When invoking object1.getArea() the method in Circle is used. When invoking object2.getArea() the method in Rectangle is used.

This is due to dynamic binding by the JVM at runtime.

You can use equalAreal() to compare any two GeometricObject objects! This is the big advantage of abstract classes.

### **Notes on Abstracts Classes**

If a subclass does not implement all abstract methods, it must also be abstract.

Abstract classes can define constructors which are called by constructors of the subclasses.

Can have abstract classes without any abstract methods; this is used as a base class for defining subclasses.

A subclasses can override a method to make it abstract; this is useful if a method in superclass can becomes invalid in a subclass.

A subclass can be abstract (eg. GeometricObject) even if superclass is concrete (eg. Object).

Abstract classes can be used as a data type (for variable declaration or parameter as above) even when they can not be instantiated.

In-class Exercise: Are these examples of legal abstract classes?

// (a)

class A {

abstract void unfinished() {

}

}

// (b)

abstract class A {

abstract void unfinished();

}

// (c)

class A {

abstract void unfinished();

}

// (d)

abstract class A {

protected void unfinished();

}

// (e)

public class abstract A {

abstract void unfinished();

}

// (f)

abstract class A {

abstract int unfinished();

}