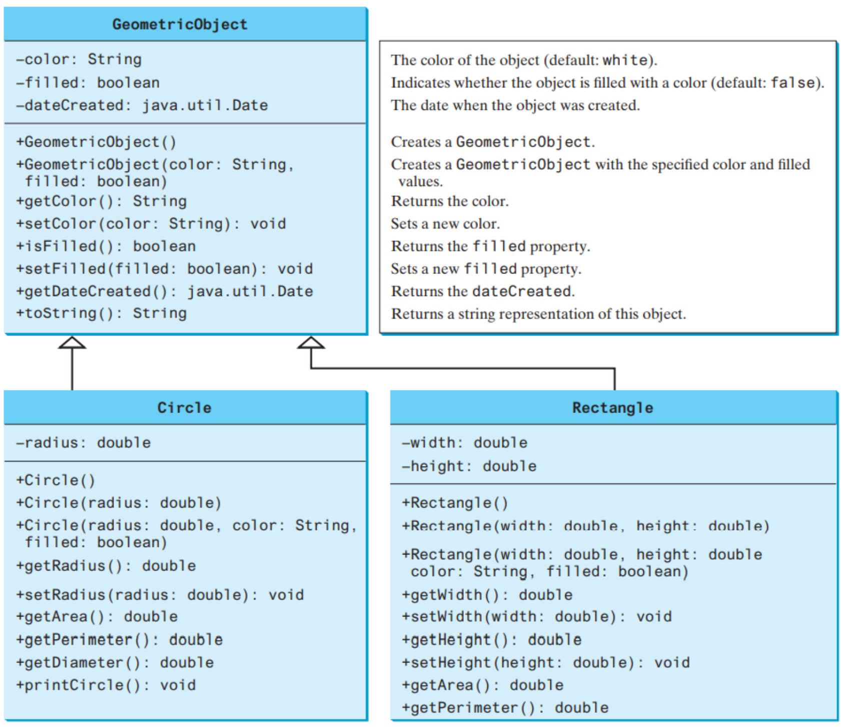
CMPE 160.01: Introduction to Object Oriented Programming

PS8: Inheritance

22/04/2022

1 Superclasses and Subclasses: GeometricObject

Remember that we have already written circle and rectangle classes on PS5 (source code is available on github page). This time, we will extend this project by adding GeometricObject class as the superclass and we will modify circle and rectangle classes so they will extend this subclass:



GeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
+GeometricObject()
+GeometricObject(color: String, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
The color of the object (default: white).
Indicates whether the object is filled with a color (default: false).
The date when the object was created.
Creates a GeometricObject.
Creates a GeometricObject with the specified color and filled values.
Returns the color.
Sets a new color.
Returns the filled property.
Sets a new filled property.
Returns the dateCreated.
Returns a string representation of this object.
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double color: String, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double

Figure 1: UML diagrams for geometric classes

Previously written circle class was as follows:

```java
class Circle {
  /** The radius of this circle */
```

```
private double radius;
private double x = 1;
private double y = 0;

// Constructors
/** Construct a circle object */
Circle() {
   // Calling the other constructor
   this(1, 2, 2);
   //this.radius = 1;
}
/** Construct a circle object with its radius*/
Circle(double radius) {
   this.radius = radius;
}
/** Construct a circle object with its radius*/
Circle(double radius, double x, double y) {
   this.radius = radius;
   this.x = x;
   this.y = y;
}
// Methods
/** Return the area of this circle */
public double getArea() {
   return radius * radius * Math.PI;
}
/** Return the perimeter of this circle */
public double getPerimeter() {
   return 2 * radius * Math.PI;
}
/** Getters and Setters */
public double getRadius() {
   return radius;
}
public void setRadius(double radius) {
   this.radius = radius;
}
public double getX() {
   return x;
}
public void setX(double x) {
   this.x = x;
}
public double getY() {
   return y;
}
public void setY(double y) {
   this.y = y;
}
/** String representation of the object */
```

```
    @Override
    public String toString() {
        return "Circle [radius=" + radius + ", x=" + x + ", y=" + y + "]";
    }
}
```

1.1 Exercise: Modify Geometric Classes

In this exercise, we will use the previously written **Circle** and **Rectangle** classes, and write a new superclass.

1- Create a new class called **GeometricObject**, following the UML diagram given above. This class represents objects, such as circle and rectangle therefore a **superclass**. For the date attribute, following the UML diagram, you can use the Java library 'java.util.Date'. For other attributes and methods, refer to the UML diagram.

2- Modify previously written **Circle** and **Rectangle** classes to fit the given UML diagram. The first thing to do is to make them extend GeometricObject class. This relationship is given with the arrows on the UML diagram. Note that arrows coming out from Rectangle and Circle classes show the GeometricObject class, this shows that these two classes extend the GeometricObject.

Also, when you look at the UML diagram you will see that some of the data fields of our previously written classes are absent (x and y), therefore the constructors and some methods are different. You can change the implementation to fit the UML diagram.

Some things to try at main method:

1- Try to create a Circle in two different ways and observe if there is any difference:

```
GeometricObject x = new Circle();
Circle t = new Circle();
```

2- Try to observe which constructors are called, and in which order, when you create a subclass. The behaviour you will observe is called **constructor chaining**.

3- Note that we do not have **toString** method in **Circle** or **Rectangle** but we have it on their superclass. Think about how we can extend this on the subclasses. This is called **Overriding**. **Important:** To override a method, the method must be defined in the subclass using the same signature and the same or compatible return type.