Basic Object Orientation

---

1) An abstract Shape

- Start Eclipse and create a new Java project called ‘lab1’
- Type in the **Shape** example given below (do not copy and paste)
- Add another class, called **Rectangle**, which extends the **Shape** class (i.e. a sub-class).
- Add the additional attributes, setters and getters which allow the width/height of the **Rectangle** to be set.
- Provide an implementation of the **getArea()** method for the **Rectangle** class.
- Build a **Driver** class to create several rectangles and test.

2) Adding a Circle

- Add a third class to the project called **Circle**, again extending the abstract **Shape** class.
- Add the additional attributes, setters and getters which allow the radius of the **Circle** to be set.
- Provide an implementation of the **getArea()** method for the **Circle** class. What comments can you make about the return type of the result?
- Extend the **Driver** class to create several circle instances and test.

3) Adding an Ellipse

- Add a fourth class to the project called **Ellipse**, but this time extend the **Rectangle** class.
- Think about whether you need to add any more attributes or getters/setters.
- Provide an implementation of the **getArea()** method for the **Ellipse** class.
- Extend the **Driver** class to create several ellipse instances and test.
- Add a **toString()** method to the three concrete classes, which displays the appropriate attribute values, the area, and the number of sides.

Think you’re finished? Go back through all of your code and make sure everything has an appropriate comment, and that every attribute and method has either ‘private’ or ‘public’ visibility modifiers.

Now, think you’re finished again? Do all your classes correctly setup the number of sides within their constructor? if not try to do this without calling the **setSides()** method.

```
/**
 * Shape class. Designed to act as an abstract base class to other shapes.
 *
 * Shows good use of Javadoc comments, annotations, visibility scope, modifiers and class hierarchies.
 *
 * class marked as 'abstract' since it is not designed to be instantiated directly.
 *
 * @author mdixon
 */
abstract class Shape {

    /**
     * The number of sides within the shape.
     *
      * Uses ‘private’ visibility modifier . Could use ‘protected’ so it can be accessed by sub-classes.
     */
    private int sides;

    /**
     * Gets the number of sides within the shape.
     *
     * @return the number of sides within the shape.
     */
    public int getSides() {

        return sides;
    }

    /**
     * Sets the number of sides within the shape.
     *
     * @param sides the number of sides within the shape.
     */
    public void setSides(int sides) {

        this.sides = sides;// use this.sides to distinguish between parameter and attribute.
    }

    /**
     * Gets the size of the shapes area.
     *
     * @return the size of the shapes area.
     */
    abstract public int getArea();

    /**
     * Constructor.
     *
     * @param sides the number of sides within the shape.
     */
    Shape(int sides) {

        this.sides = sides;
    }

}
```