Unit 6: Tell, Don't Ask

After taking this unit, students should:

- understand what accessor and mutator are used for, and why not to use them
- understand the principle of "Tell, Don't Ask"

Accessors and Mutators

Similar to providing constructors, a class should also provide methods to retrieve or modify the properties of the object. These methods are called the *accessor* (or *getter*) or *mutator* (or *setter*).

The use of accessor and mutator methods is a bit controversial. Suppose that we provide an accessor method and a mutator method for every private field, then we are exposing the internal representation, therefore breaking the encapsulation. For instance:

```
1  // Circle v0.4
2  class Circle {
3    private double x;
4    private double y;
5    private double r;
6
7    public Circle(double x, double y, double r) {
8      this.x = x;
9      this.y = y;
10     this.r = r;
11   }
12
13   public double getX() {
14     return this.x;
15   }
16
17   public void setX(double x) {
18     this.x = x;
19   }
20
21   public double getY() {
22     return this.y;
23   }
24
25   public void setY(double y) {
26     this.y = y;
27   }
28
29   public double getR() {
```

```
30      return this.r;
31    }
32
33    public void setR(double r) {
34      this.r = r;
35    }
36
37    public double getArea() {
38      return 3.141592653589793 * this.r * this.r;
39    }
40  }
```

The "Tell Don't Ask" Principle

The mutators and accessors above are pretty pointless. If we need to know the internal and do something with it, then we are breaking the abstraction barrier. The right approach is to implement a method within the class that does whatever we want the class to do. For instance, suppose that we want to check if a given point (x,y) calls within the circle, one approach would be:

```
1  double cX = c.getX();
2  double cY = c.getY();
3  double r = c.getR();
4  boolean isInCircle = ((x - cX) * (x - cX) + (y - cY) * (y - cY)) <= r * r;
```

where `c` is a `Circle` object.

A better approach would be to add a new `boolean` method in the `Circle` class, and call it instead:

```
1  boolean isInCircle = c.contains(x, y);
```

The better approach involves writing a few more lines of code to implement the method, but it keeps the encapsulation intact. If one fine day, the implementer of `Circle` decided to change the representation of the circle and remove the direct accessors to the fields, then only the implementer needs to change the implementation of `contains`. The client does not have to change anything.

The principle around which we can think about this is the "Tell, Don't Ask" principle. The client should tell a `Circle` object what to do (compute the circumference), instead of asking "what is your radius?" to get the value of a field then perform the computation on the object's behalf.

While there are situations where we can't avoid using accessor or modifier in a class, for beginner OO programmers like yourself, it is better to not define classes with any

accessor and modifier to the private fields, and forces yourselves to think in the OO way -- to tell an object what task to perform as a client, and then implement this task within the class as a method as the implementer.

Further Reading

- Tell Don't Ask by Martin Fowler
- Why getters and setters are evil, by Allen Holub, JavaWorld
- Getters and setters are evil. Period, by Yegor Bygayenko.