13. Inheritance :: Private Accessor/Mutator Methods

In a proper object oriented design, the instance variables of all classes will be **private** and a class will provide either **public** or **protected** accessor and/or mutator methods to provide read/write access to the instance variable, as required. If no external access to the instance variable is required, then it would be incorrect to provide **public** or **protected** accessor/mutator methods; however, I still provide **private** accessor/mutator methods to read/write the instance variable within the class:

```
public class C {
  private int mX;
  ...
  // Private accessor method for mX.
  private int getX() {
    return mX;
  }
  // Private mutator method for mX.
  private void setX(int pX) {
    mX = pX;
  }
  public double cube() {
    // Calculate and return the cube of mX. We get the value of mX by calling the
    // getX() accessor method.
    return Math.pow(getX(), 3);
  }
}
```

13. Inheritance :: Private Accessor/Mutator Methods (continued)

getX() and *setX*() are not strictly necessary as *mX* is directly accessible in the methods of *C*:

```
public class C {
  private int mX;
  ...
  public double cube() {
    // Calculate and return the cube of mX. We can directly access mX.
    return Math.pow(mX, 3);
  }
}
```

So why write **private** accessor/mutator methods?

With the **private** accessor/mutator methods we are restricting all accesses to *mX* to method calls to the accessor/mutator methods. In particular, there is only **one statement in the entire program** that modifies *mX* and that statement is in the mutator method *setX*(). If during testing and debugging we determine that the value of *mX* is being erroneously changed, we know who the culprit is: it *has* to be *setX*(). Of course, *setX*() may not be where the bug actually is, but it gives us a starting point for debugging. We trace backward from *setX*() to see what method called *setX*(). We examine that method and if the source of the bug is not there, we trace backward to see what method call that method. Keep tracing backward looking for the bug and eventually you are guaranteed to find it. On the other hand, if an instance variable can be changed from many different places in the code, it makes locating the bug a bit more difficult.