Keeping your method calls in bounds...

Here's a Car class that demonstrates all the ways you can call methods and still adhere to the Principle of Least Knowledge:

```
public class Car {
        Engine engine;
        // other instance variables

        public Car() {
                // initialize engine, etc.
        }

        public void start(Key key) {
                Doors doors = new Doors();
                boolean authorized = key.turns();
                if (authorized) {
                        engine.start();
                        updateDashboardDisplay();
                        doors.lock();
                }
        }

        public void updateDashboardDisplay() {
                // update display
        }
}
```

Here's a component of this class. We can call its methods.

Here we're creating a new object; its methods are legal.

You can call a method on an object passed as a parameter.

You can call a method on a component of the object.

You can call a local method within the object.

You can call a method on an object you create or instantiate.

there are no Dumb Questions

Q: There is another principle called the Law of Demeter; how are they related?

A: The two are one and the same, and you'll encounter these terms being used interchangeably. We prefer to use the Principle of Least Knowledge for a couple of reasons: (1) the name is more intuitive, and (2) the use of the word "Law" implies we always have to apply this principle. In fact, no principle is a law; all principles should be used when and where they are helpful. All design involves tradeoffs (abstractions versus speed, space versus time, and so on) and while principles provide guidance, you should take all factors into account before applying them.

Q: Are there any disadvantages to applying the Principle of Least Knowledge?

A: Yes; while the principle reduces the dependencies between objects and studies have shown this reduces software maintenance, it is also the case that applying this principle results in more "wrapper" classes being written to handle method calls to other components. This can result in increased complexity and development time as well as decreased runtime performance.