Chapter 6 - Objects and Data Structures

Data Abstraction

Hiding implementation is not just a matter of putting a layer of functions between the variables. Hiding implementation is about abstractions! A class does not simply push its variables out through getters and setters. Rather it exposes abstract interfaces that allow its users to manipulate the essence of the data, without having to know its implementation.

Data/Object Anti-Symmetry

These two examples show the difference between objects and data structures. Objects hide their data behind abstractions and expose functions that operate on that data. Data structure expose their data and have no meaningful functions.

Procedural Shape

```
public class Square {
  public Point topLeft;
  public double side;
}

public class Rectangle {
  public Point topLeft;
  public double height;
  public double width;
}

public class Circle {
  public Point center;
  public double radius;
}

public class Geometry {
  public final double PI = 3.141592653589793;

  public double area(Object shape) throws NoSuchShapeException {
    if (shape instanceof Square) {
      Square s = (Square)shape;
      return s.side * s.side;
    }
    else if (shape instanceof Rectangle) { Rectangle r = (Rectangle)shape; return r.height *
r.width;
    }
    else if (shape instanceof Circle) {
      Circle c = (Circle)shape;
      return PI * c.radius * c.radius;
    }
    throw new NoSuchShapeException();
  }
}
```

Polymorphic Shape

```
public class Square implements Shape {
  private Point topLeft;
  private double side;

  public double area() {
    return side*side;
  }
}

public class Rectangle implements Shape {
  private Point topLeft;
  private double height;
  private double width;

  public double area() {
    return height * width;
  }
}

public class Circle implements Shape {
  private Point center;
  private double radius;
  public final double PI = 3.141592653589793;

  public double area() {
    return PI * radius * radius;
  }
}
```

Again, we see the complimentary nature of these two definitions; they are virtual opposites! This exposes the fundamental dichotomy between objects and data structures:

Procedural code (code using data structures) makes it easy to add new functions without changing the existing data structures. OO code, on the other hand, makes it easy to add new classes without changing existing functions.
The complement is also true:

Procedural code makes it hard to add new data structures because all the functions must change. OO code makes it hard to add new functions because all the classes must change.
Mature programmers know that the idea that everything is an object is a myth. Sometimes you really do want simple data structures with procedures operating on them.

The Law of Demeter

There is a well-known heuristic called the Law of Demeter that says a module should not know about the innards of the objects it manipulates.

More precisely, the Law of Demeter says that a method `f` of a class C should only call the methods of these:

- C

- An object created by `f`
- An object passed as an argument to `f`
- An object held in an instance variable of `C`

The method should not invoke methods on objects that are returned by any of the allowed functions. In other words, talk to friends, not to strangers.

Data Transfer Objects

The quintessential form of a data structure is a class with public variables and no functions. This is sometimes called a data transfer object, or DTO. DTOs are very useful structures, especially when communicating with databases or parsing messages from sockets, and so on. They often become the first in a series of translation stages that convert raw data in a database into objects in the application code.