Double Dispatch With Commands

Another great place to use this is with the command pattern. In this case, we can have an object that forces the execution of the command. The command, knowing what it is, understands (using polymorphism) what it has to do to achieve it's goal. In this dance, one object acts as a trigger while the action does just that, takes action. Consider the case where we have a DB class that accepts a Query. A Query could be a Read, Write, or Delete. To implement this, Query would define the method execute which the other three objects would implement. Query is defined in listing 2.

Listing 2: Query And DB

```
public interface Query {
   public ResultSet execute( DB db);
}
//
public class DB   {
   public ResultSet execute( Query query)
   {
     return query.execute( this);
   }
   public ResultSet read( ReadQuery read) { ... }
   public ResultSet write( WriteQuery write) { ... }
   public ResultSet delete( DeleteQuery delete) { ... }
}
//
public class ReadQuery implements Query  {
   public ResultSet execute( DB db){
          return db.read(this);}
}
//
public class WriteQuery implements Query
{
...
```

As can be seen in listing 2, the DB execute method accept a query. It then immediately turns that call back onto the query. Since the query has knowledge of self, it knows exactly what to do to get the job done.

What is interesting about this technique is that we can define an Update object well after the system has been deployed without needing to make any changes to DB. We could add other command like objects as well. We could also change the implementation of DB without having any effect on our Querys. The power here is that if our DB implementation doesn't perform as well as we'd like it to, we can change it while minimizing the impact on the overall code base.

Double Dispatch Performance

One final note now that performance has been brought into the picture. An alternate implementation would be to use instanceof statements in the DB.execute method (see listing 3). The results of such testing could be used to perform a class cast before continuing on with the DB logic. Though this feels like it should out-perform the double dispatch, in my micro-benchmarks, it doesn't. This is due to HotSpots ability to inline the method calls (yet another benefit of using small methods). So what were are left with is a good design technique that has no performance penalty which seems like a win-win to me.

Listing 3: DB execute using switch

```
public class DB {

   public ResultSet execute( Query query)    {

      if (query instanceof ReadQuery)

              return this.read((ReadQuery) query);

     else if (query instanceof WriteQuery)

      ...

   }

   ...
```