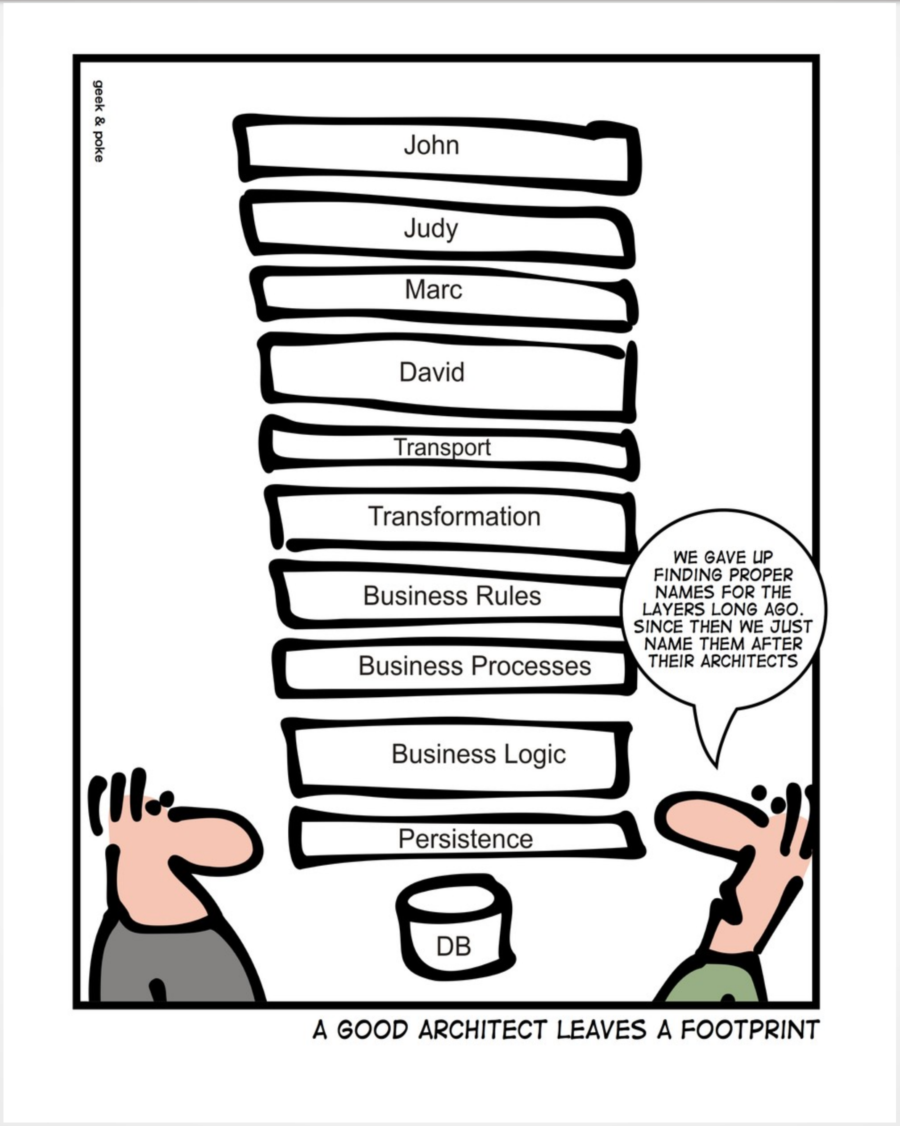
Metaclasses and Reflection


geek & poke
John
Judy
Marc
David
Transport
Transformation
Business Rules
Business Processes
Business Logic
Persistence
DB
WE GAVE UP FINDING PROPER NAMES FOR THE LAYERS LONG AGO. SINCE THEN WE JUST NAME THEM AFTER THEIR ARCHITECTS
A GOOD ARCHITECT LEAVES A FOOTPRINT

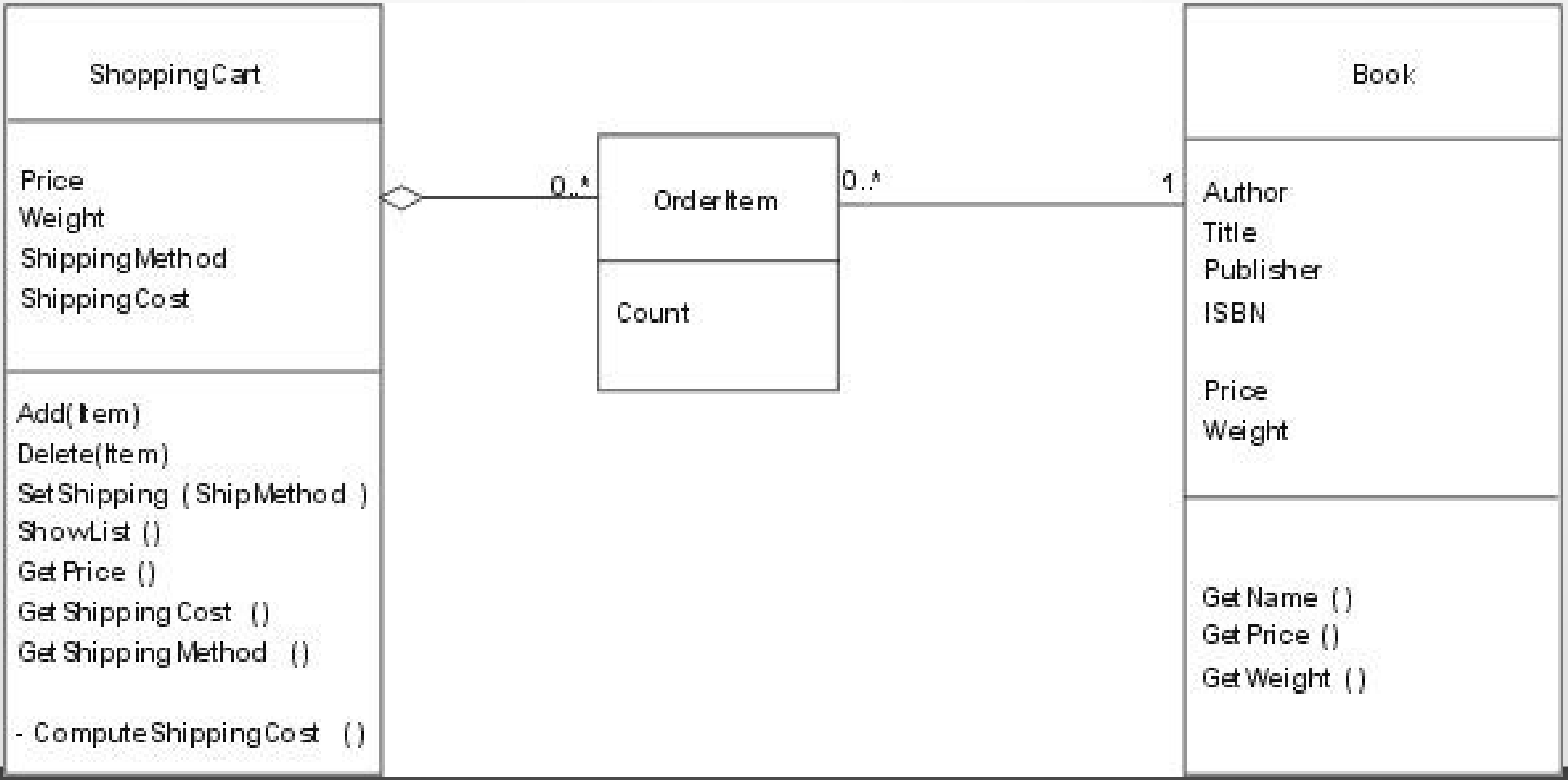
Krishna Kumar

Reflection

- Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time.
- Given some object we first should ask for its type. As a result we are given class descriptor, which provides information about class methods and fields. We can use these field descriptors to fetch/store object fields and can use method descriptors to lookup and invoke methods.
- Unfortunately C++ doesn't support reflection:
  - RTTI support was added to the language. But RTTI provides only very restricted subset of reflection: it allows to get object compile-time and runtime type (it is possible to get object runtime type only if object class contains virtual functions).
  - You can compare types and you can get type name - and that is all you can do with RTTI.

Online bookstore

- Let's look at a simple example: Susan, the manager of a local bookstore, wants to expand into the Internet. So she asks you to write a simple program for an Internet bookshop.

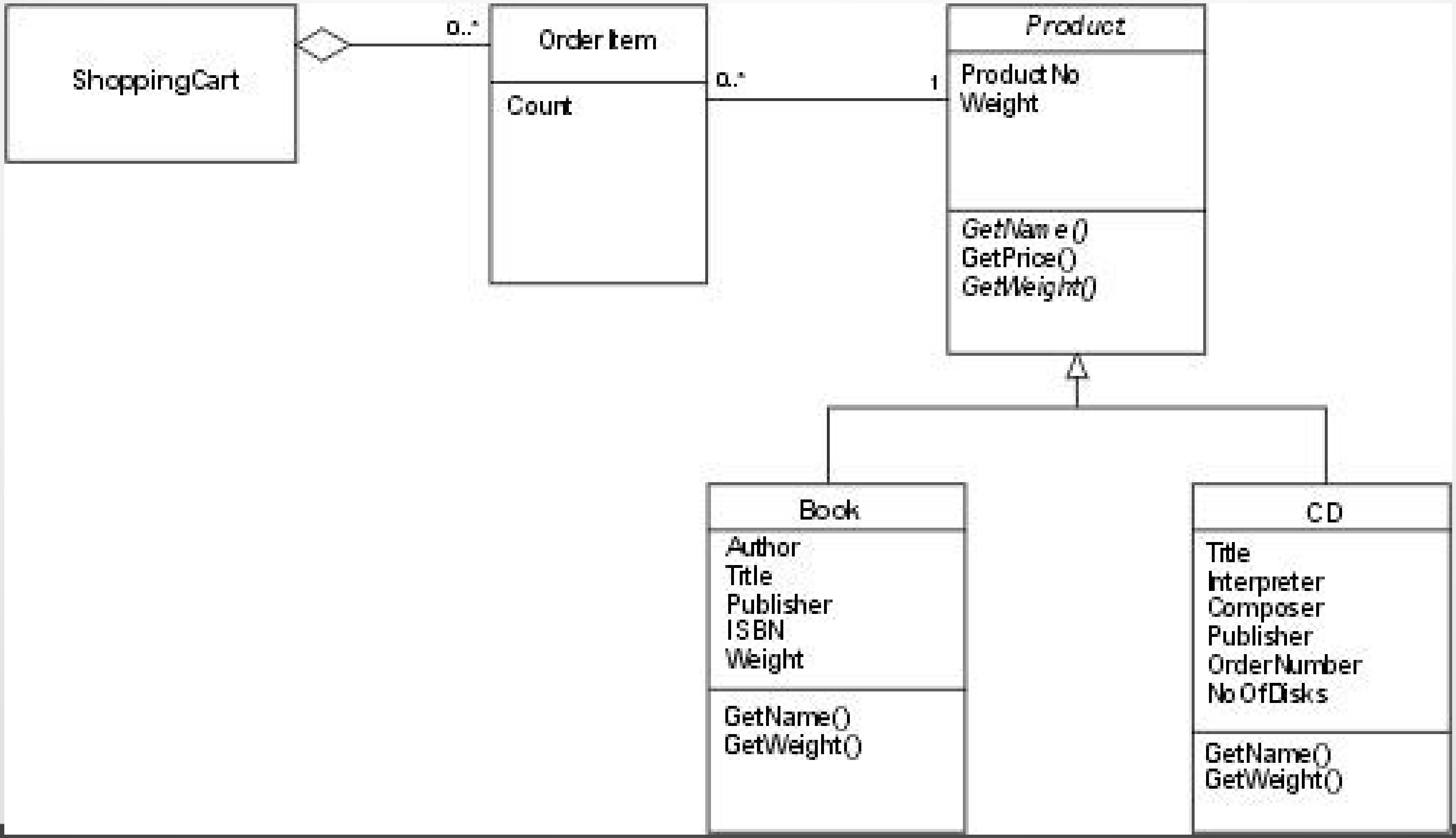


Online book store

```
class Book {
public:
    Book(const string & author_, const string & title_, const string & publisher_,
         double price_, double weight_);
    string getName() {
        string name = author + ": " + title;
        return name.substr(0, 40);
    }
    double getPrice();
    double getWeight();
private:
    string author, title, publisher;
    double price, weight;
};
```

Online book store – Sell CDs

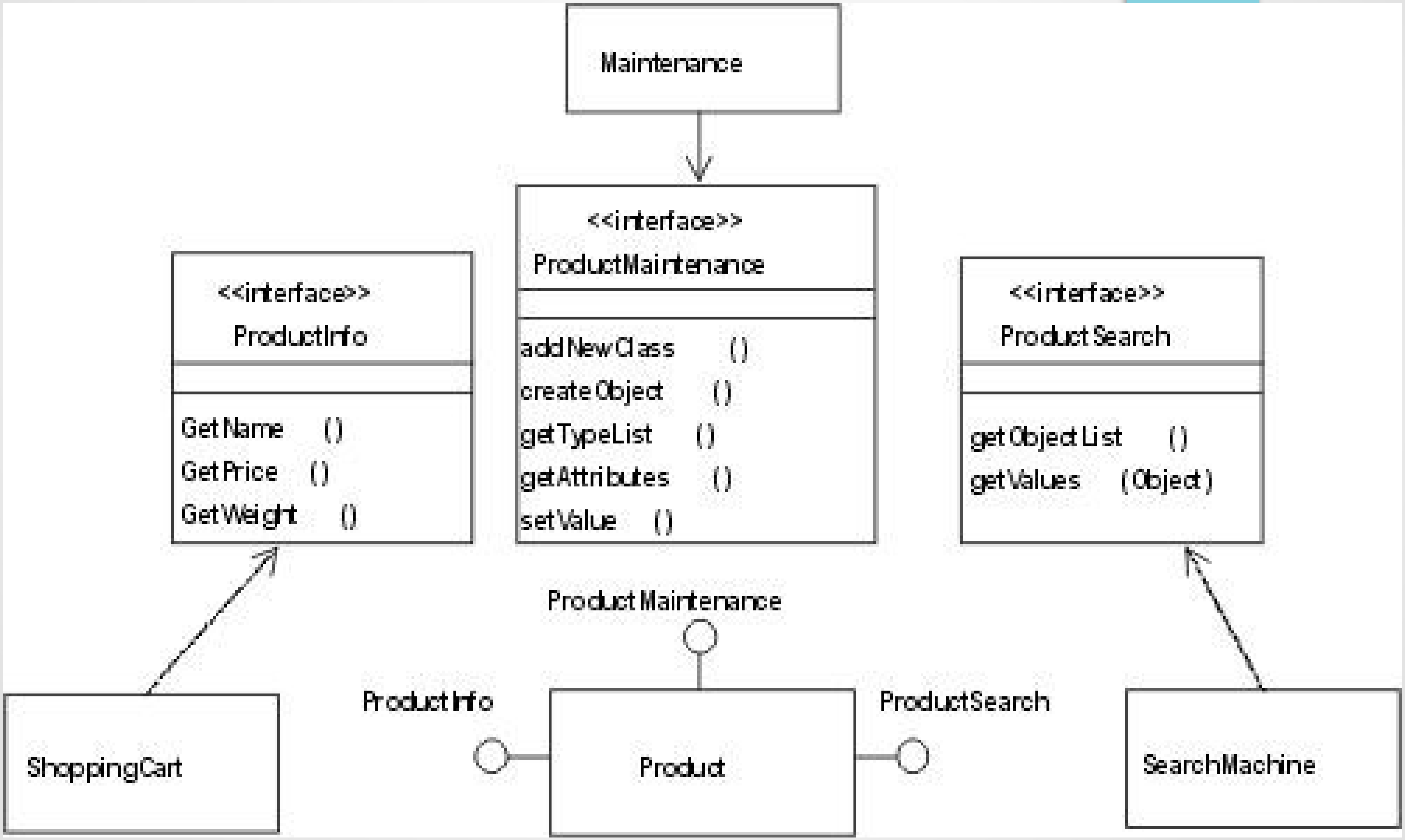
- Susan decides to sell CDs as well. So you have to change your program. With object orientation, you can do this quite easily and your modified class model will look like



Massive online retail store

- Now it is clear that it is not acceptable to modify. We need to provide different interfaces for the Product class: A simple interface for ShoppingCart providing getName(), getPrice(), and getWeight(). Then you need a different interface for a general search machine, which must provide information like:
  - what is the actual class of the object
  - what attributes does that class have
  - what are the actual values of these attributes for the object.
- This is a classic reflection interface that gives you information about the properties of classes and objects.
- But you also need a third interface for product maintenance that allows you to define new product classes, specify the attributes for them, create instances of these classes, and set the attribute values of these instances. Such an interface is called a "Meta-Object Protocol (MOP)". The reflection protocol is a subset of such a MOP.

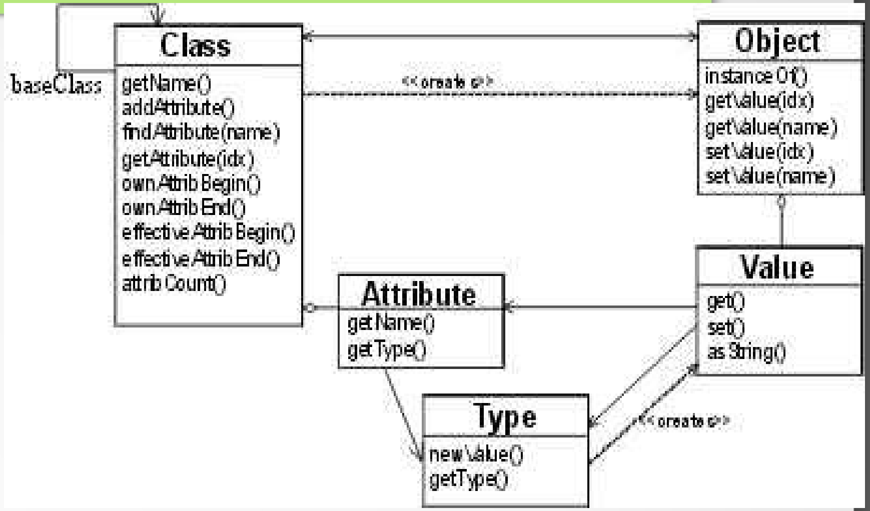
Online retail store – Better Model



Meta Classes for C++

- What is the meaning of "Meta-Object Protocol"? Well, meta information is information about something else seen from a level beyond -- a meta level.
- So, information about the attribute values of an object, say someBook.author, is information on the object level. But information about the properties of the object itself, about its attributes, its structure, etc. is meta information.
- In C++, this information is captured in the class definition for the object, so the class is a meta-object. And in C++, you have all the functionality of a MOP at class level -- which is at development time. But that level is not available at runtime: You cannot manipulate classes like objects, you cannot add new classes at runtime.
- The idea of a MOP is to collapse the meta-level (classes) and the object level (objects); i.e. make the class definitions normal objects and the object properties are normal attribute values of the class definitions that can be manipulated at runtime.

MOP Overview



- definition of new classes
- adding attributes to classes
- querying attributes of classes
- creating objects
- querying the class of an object
- setting attribute values of an object
- querying attribute values of an object
- deleting objects

• For our example with the Internet shop and a product hierarchy this would probably quite useful.

References

- Scott Meyer's “Effective C++: 35 New Ways to Improve Your Programs and Designs”
- Dov Bulka & David Mayher's “Efficient C++ Performance Programming Techniques”