Builder and Facade

10-4-2005

Opening Discussion

- Static inner classes have less memory overhead. They can't access “parent” object. Qualified this typically implicit.
- Don't call overridable methods (especially from constructors). Prohibit inheritance with final or private constructors.
- auto-ptr always called delete (not delete[], so you can't use it with arrays.
- Do you have any questions about the readings for today before we take the quiz?

Builder

- This is a creational pattern that separates the construction of a complex object from its representation so the same construction process can be used for many different representations.
- Obviously we do this when we want to vary the representations that we have for a certain piece of data, but we can find a common interface for how it is built.
- The thing calling the builder is referred to as the director. It produces a product.

Example

- GoF uses the example of reading in an RTF document. You can have one parser that reads in tokens and passes those to a builder interface. Each implementation of the builder interface builds a different representation of the document.
- The advantage is that you only write one parser and it work for all the representations that you might want to add. In this example, some of them ignore different tokens and some handle the tokens differently.

Benefits and Drawbacks

- You can easy vary the internal representation of the product.
- It encapsulates the code for construction and representation. This way the client code doesn't care about it.
- Gives fine grained control over the construction process. This is opposed to most other creational patterns where things are created in one fell swoop. Here the product is built up of a series of calls to the builder as orchestrated by the director.

Facade

- This structural pattern provides a uniform interface to multiple other interfaces in a subsystem. The intention is to provide a higher-level interface that is easier to use.
- Basically the facade is a single class with a number of functions that pass straight through to other functions. The idea is just to consolidate things to help out the user.
- Of course, the user doesn't even have to know that he/she is working with a facade.

Example

- GoF uses the example of a compiler that has many parts (scanner, parser, etc.) but where most applications don't need to know about all the parts. They can instead with with a facade and the life of the developer is made simpler.
- If some aspect of the program needs the power, it can go in and work with the things behind the facade, but that is optional.
- This especially helps when patterns create systems with lots of little classes.

Benefits and Drawbacks

- It shields the user from the details of the subsystem and makes life easier.
- It provides a weak coupling between the client code and in internal representation making it easier to change what is happening behind the facade.
- Doesn't technically prohibit clients from using the pieces of the subsystem, but doing so will remove some of the other benefits.

Progress Reports

- Who are we going to have present today?
- Remember, I want to see UML that helps me to understand how your design and how the parts of what you are doing work together.