Factory Method & Interpreter

11-1-2005

Opening Discussion

- Do you have any questions about the reading for today?

Factory Method

- Defines in interface for creating an object, but allows for subclasses to decide what class to instantiate.
- This pattern is also known as a virtual constructor. Remember that constructors are inherently non-virtual.
- This pattern involves two abstractions: the Creator and the Product. Each has different concrete implementations where the concrete Creators each create their own type of concrete Product.

Example

- GoF uses the example of a framework that allows the user to view multiple different types of documents. The framework can be instantiated with different application types and each application type has its own type of document. The problem is that the framework only knows of abstract applications and abstract documents. It also knows when a new document must be created, but not the type. The application interface should have a factory method.

Benefits and Drawbacks

- Allows your code to deal with an interface for creating objects so you can abstract the details away.
- You might be forced to create new subclasses of the creator when they don't do much.
- They can also be beneficial when you have parallel hierarchies. This is when you have a hierarchy of types and those types have some other data associated with them, but it varies by the type. I have done this for RMI.

Interpreter

- This pattern involves creating an interpretor and a representation for a particular (formal) language.
- This has many similarities to a Composite. I would even say that this is a special form of a composite where you are parsing sentences in the language to a composite tree. This tree then has the ability to recognize/operate on certain inputs.
- You have a common abstract supertype and subtypes for each terminal/non-terminal.

Example

- GoF uses the example of an interpreter for regular expressions. The supertype has the operation of interpreting a string. The subtypes represent the different possibilities for combining expressions and well as the literals at the leaves of the tree.
- More generally you could implement any type of grammar that you wanted in this way. You have the supertype as well as nonterminal and terminal subtypes.

Benefits and Drawbacks

- This makes it very easy to change or extend the grammar. All you have to do is change or add subtypes.
- It's easy to create grammar as implementing the inheritance tree is quite simple.
- Once the grammars get complex they can be hard to maintain.
- It is also easy to add new ways to interpret expressions. Here the Visitor pattern could come into play.

Progress Reports

- Has anyone gotten enough stuff together that they want to show it? Remember to talk about design as well as implementation.