Memento & Prototype

4-27-2004

Evaluations

- We need to do evaluations and since this is the last regularly scheduled meeting of this class we should do them today. Once they are done we can start the normal class material.

Opening Discussion

- MI mainly causes ambiguity, but more people should have mentioned the diamond. What are the implications/solutions for that?
- Use templates when the functionality does NOT change with the type. Basically, the code for foo<int> and foo<TypeB> has to be identical. Use inheritance otherwise. On a more fundamental level, templates are static and virtual functions are dynamic.
- Do you have any questions about the reading for today?

Memento

- With this pattern we capture the internal state of an object and externalize it so that it can be restored to that state later.
- The idea is that there is some originator that wants a client to be able to “remember” its current state, but doesn't want to break encapsulation. It gives the client a memento that has no meaning to the client, but which stores the information needed by the originator.
- This is a place where downcasts might be required.

Example

- An example of this would be an undo mechanism where we don't want the undo to have internal information about the things being altered. We talked about undo before with command. Memento can help out to “fortify” the cases where simply undoing an alteration doesn't always return to the original state.
- GoF uses an example of a drawing program with connections between boxes. The connection solver is the originator.

Benefits and Drawbacks

- Helps to maintain encapsulation.
- Keeps the originator simple, it doesn't have to remember older states.
- It can get expensive as the mementos might be large and you might keep quite a few around.
- In some languages it is hard to keep the contents of the memento encapsulated.

Prototype

- This is a creational pattern where new objects are constructed as copies of some prototype object.
- The client has a reference to a prototype object. Prototype is an interface and it doesn't know the exact type. The interface has a clone method that is invoked when we want a new object. That returns an object of the proper subtype.

Example

- GoF uses the example of a graphical editor that is part of a general framework. In one application this framework is used for a music editor. There are GraphicTools in the framework that are used to create different GraphicObjects. The GraphicTool class though doesn't know about the music stuff at all. It just clones a prototype.
- Cloning in Java is an example of this. Some languages (e.g. SELF and Cecil) use prototyping for almost everything.

Benefits and Drawbacks

- The primary benefit is that the client code doesn't have to know the exact type that it is creating.
- It makes it easier to add or remove products at runtime because the creation is dynamically bound.
- You can effectively create new types by composition and prototype off of those.
- You do a lot less subclassing than with Factory Method.
- You can configure classes dynamically.
- All subtypes must implement clone.

Progress Reports

- Mike is going to give his main progress report and anyone else who has something to show briefly should do so.
- 1000 lines rule.
- Demonstrations start at 5:30pm on the 10th of May.