Abstract Factory & Adapter

1-27-2004

Opening Discussion

- Do you have any quick questions about the readings before we take the quiz?
- I have seen significant discussion and work by a number of people in the class. That is good. Also, quite a few of you have already been busy looking up information and methods to do your parts of the project. Once again that is a very good thing for you to be doing.

Some Principles of Design Patterns

- Abstract that which varies.
  - If there is something that could change in your program, you should abstract that so that different implementations can easily be added.
- Program to an Interface, not an Implementation.
  - Also, inherit from interfaces, not implementations.
- Prefer Composition to Inheritance
  - Only use inheritance when you really want subtyping. For code reuse and data sharing, stick with composition instead.

Creational Patterns

- Constructors can not be abstracted. This leads to a strong need to abstract the process of creating new items.
- The general idea of these patterns is that you have a normal method that returns a type that serves as the supertype for a number of other types.
- This hides not only the exact type that is created, but how it is created, and what data it is created with. All those things can be varied dynamically or statically.

Abstract Factory

- Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- You have an interface for the abstract factory as well as interfaces for all of the types that it might create.
- This can be especially helpful for dealing with families of objects that need to be created together.
- The main drawback is that it is difficult to extend it to create new types of products.

Example

- Having a GUI with different look-and-feels is a place where you would use an abstract factory.
- The factory interface can create any widget type (button, text field, scrollbar, etc.). You have a different concrete implementation for each look-and-feel.
- You also have an interface for each type of widget and concrete types for each look-and-feel.

Structural Patterns

- As the name implies, these patterns deal with how classes and objects are composed together to form larger structures.
- Structural class patterns combine classes/interfaces through inheritance.
- Structural object patterns give way to compose objects to arrive at new functionality. These allow the composition to be changed at runtime.

Adapter

- An adapter converts the interface of existing code to another interface that a client expects.
- This can be done as a class pattern by inheriting the existing class and adding methods.
- A more flexible, and advisable, approach is to use composition and forward the requests to the object. This also allows you to have a single adapter for multiple classes.

Example

- The people doing graphics for the project could use this pattern.
- In this case they are working with something like OpenGL. The interface for the graphics that others see won't be OpenGL. Some parts of OGL will need to be wrapped up (an adapter) so that they fit the interface that is generally used.