CSCI 253

Object Oriented Design:
Abstract Factory Pattern
George Blankenship

Overview

Creational Patterns
- Singleton
- Abstract factory
- Factory Method
- Prototype
- Builder

Structural Patterns
- Composite
- Façade
- Proxy
- Flyweight
- Adapter
- Bridge
- Decorator

Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Abstract Factory Pattern: The Problem

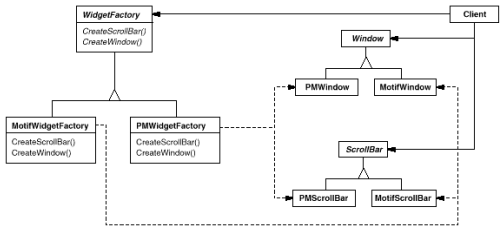
Provide an Interface for creating families of related or dependent objects without specifying their concrete classes

- A GUI toolkit that supports multiple look-and-feel standards
- Achieve portability of an application across different windowing systems

Also known as : Kit

Abstract Factory Pattern George Blankenship 4

Widget Factory



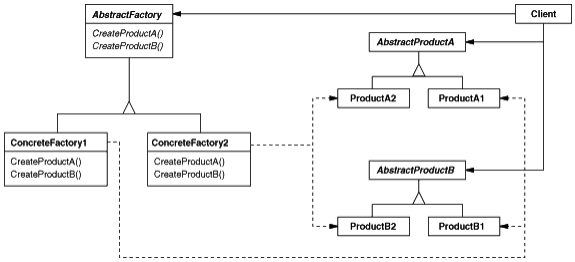
Abstract Factory Pattern George Blankenship 5

The Abstract Factory Pattern Participants

- *AbstractFactory*: declares an interface for operations that create abstract product objects
- *ConcreteFactory*: implements the operations to create concrete product objects
- *AbstractProduct*: declares an interface for a type of product object
- *ConcreteProduct*: defines a product object to be created by the corresponding concrete factory; implements the AbstractProduct interface
- *Client*: uses only interfaces declared by AbstractProduct and AbstractFactory

Abstract Factory Pattern George Blankenship 6

Participant Map



Abstract Factory Pattern George Blankenship 7

The Abstract Factory Pattern Collaboration

- AbstractFactory defers creation of product objects to its ConcreteFactory subclass
- A single instance of a ConcreteFactory is created at run-time; this concrete factory creates product objects having a particular implementation

Abstract Factory Pattern George Blankenship 8

UI Look and Feel

```
String laf =
   UIManager.getSystemLookAndFeelClassName();
   try {UIManager.setLookAndFeel(laf);}
   catch (UnsupportedLookAndFeelException exc)
        {System.err.println("UnsupportedL&F: " + laf);}
   catch (Exception exc)
        {System.err.println("Error loading " + laf);}
```

Abstract Factory Pattern George Blankenship 9

The Abstract Factory Pattern Conseq. (1)

- + Isolates concrete classes: the AbstractFactory encapsulates the responsibility and the process to create product objects, it isolates clients from implementation classes; clients manipulate instances through their abstract interfaces, the product class names do not appear in the client code
- + Makes exchanging product families easy: the ConcreteFactory class appears only once in an application -that is, where it is instantiated- so it is easy to replace; because the abstract factory creates an entire family of products the whole product family changes at once

Abstract Factory Pattern George Blankenship 10

The Abstract Factory Pattern Conseq. (2)

- + Promotes consistency between products: when products in a family are designed to work together it is important for an application to use objects from one family only; the abstract factory pattern makes this easy to enforce
- +- Supporting new types of products is difficult: extending abstract factories to produce new kinds of products is not easy because the set of Products that can be created is fixed in the AbstractFactory interface; supporting new kinds of products requires extending the factory interface which involves changing the AbstractFactory class and all its subclasses

Abstract Factory Pattern George Blankenship 11

The Abstract Factory Pattern Implement. (1)

- Factories as singletons: an application needs only one instance of a ConcreteFactory per product family, so it is best to implement this as a singleton
- Creating the products:
  - AbstractFactory only declares an interface for creating products, it is up to the ConcreteFactory subclasses to actually create products
  - The most common way to do this is use a factory-method for each product; each concrete factory specifies its products by overriding each factory-method; it is simple but requires a new concrete factory for each product family even if they differ only slightly
  - An alternative is to implement the concrete factories with the prototype pattern; the concrete factory is initialised with a prototypical instance of each product and creates new products by cloning

Abstract Factory Pattern George Blankenship 12

The Abstract Factory Pattern Implement. (2)

- Defining extensible factories:
  - a more flexible but less safe design is to provide AbstractFactory with a single “make” function that takes as a parameter (a class identifier, a string) the kind of object to create
  - is easier to realise in a dynamically typed language than in a statically typed language because of the return type of this “make” operation
  - can for example be used in C++ only if all product objects have a common base type or if the product object can be safely coerced into the type the client that requested the object expects; in the former the products returned all have the same abstract interface and the client will not be able to differentiate or make assumptions about the class of the product

Abstract Factory Pattern George Blankenship 13