CSCI 253

Object Oriented Design:
Builder Pattern
George Blankenship

Overview

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

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

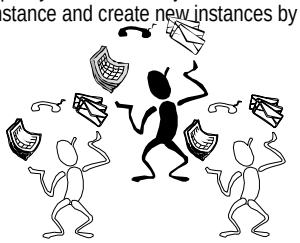
Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

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 Prototype Pattern: The Problem

Specify the kinds of objects to create using a prototypical instance and create new instances by copying this prototype

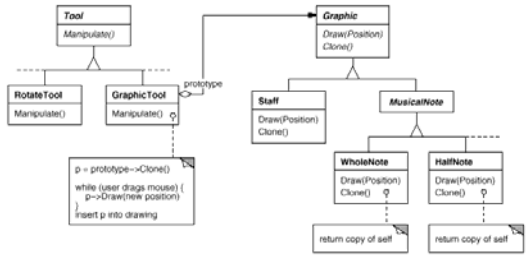


- when an application needs the flexibility to be able to specify the classes to instantiate at run time

- when instance of a class have only very few different combinations of state

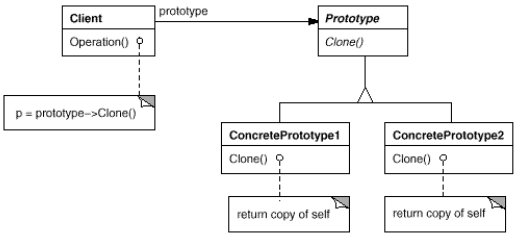
Prototype Pattern George Blankenship 4

Music Sheet



Prototype Pattern George Blankenship 5

Participants



Prototype Pattern George Blankenship 6

The Prototype Pattern Participants an Collaborations

- *Prototype*: declares an interface for cloning itself
- *ConcretePrototype*: implements an operation for cloning itself
- *Client*: creates a new object by asking the prototype to clone itself

- Client asks a Prototype to clone itself

The Prototype Pattern Consequences (1)

- + Hides the concrete product classes from the client: clients can work with application specific classes without modification
- + Products can be added and removed at run-time: new concrete products can be incorporated by just registering them with the client
- + Specifying new objects by varying values: new kinds of objects are effectively defined by instantiating a specific class, filling in some of the instance variables and registering this as a prototype
- + Specifying new objects by varying structure: complex user-defined structures can be registered as prototypes as well and used over and over again by cloning them

The Prototype Pattern Consequences (2)

- + Reduced subclassing: as opposed to the Factory Method pattern that often produces a hierarchy of creator classes that mirrors the hierarchy of ConcreteProducts
- + Configuring an application with classes dynamically: when the run-time environment supports dynamic loading of classes the prototype pattern is a key to exploiting these facilities in static languages (the constructors of the dynamically loaded classes cannot be addressed statically, instead the run-time environment creates automatically a prototype instance that the application can use through a prototype manager)
- - Implementing the Clone operation: is difficult when the classes under consideration already exist or when the internals include objects that do not support copying or have circular references

The Prototype Pattern Implementation

- Using a prototype manager: when the number of prototypes in a system is not fixed it is best to use a registry of available prototypes
- Implementing the clone operation: many languages have some support for implementing the clone operator (copy constructors in C++, copy method in Smalltalk, save + load in systems that support these) but in itself they do not solve the shallow / deep copy issue
- Initialising clones: some clients are happy with the clone as it is, others will want to initialise the clone; passing parameters to the clone operation precludes a uniform cloning interface; either use state changing operation that are provided on the clone immediately after cloning or provide a Initialise method
- In languages that treat classes as first class objects the class object itself is like a prototype for creating instances of each class

Prototype Pattern George Blankenship 10