the GoF

Design Patterns

Ali Doğru

Contents

- Philosophy
- Catalog
- GoF (Gang of Four): Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides [1995 Addison Wesley]

Design Patterns: Elements of Reusable Software

Design Patterns ?

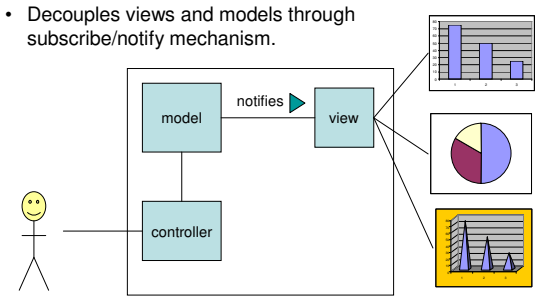
- Core solution to a repeating problem, that can be used many times, differently each time. Christopher Alexander (1977)
- Abstract
- Need instantiation
- Represent a small part of a system

Desing Pattern

- Name
- Problem: when to apply
- Solution: elements, relations, collaborations
- Consequences: trade-offs (flexibility, extensibility, portability...)

Model View Controller

- Decouples views and models through subscribe/notify mechanism.



Classification

		purpose	
	Creational	**Structural**	**behavioral**
class	Factory method	Adapter (class)	Interpreter
			Template method
object	Abstract factory	Adapter (object)	Chain of responsibility
	Builder	Bridge	Mediator
	Prototype	Composite	Memento
	singleton	Decorator	Command
		Facade	Iterator
		Flyweight	Observer
		Proxy	State
			Strategy
			Visitor

GoF Pattern Description Format

- Name and classification
- Intent
- Also known as
- Motivation
- Applicability
- Structure (OMT)
- Participants
- Collaborations
- consequences
- Implementation
- Sample code
- Known uses
- Related patterns

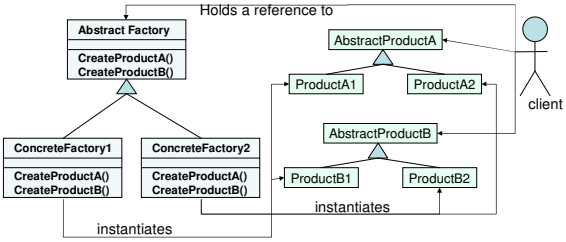
Late Binding

- Class (implementation) inheritance / interface inheritance (Java's interface)
- Abstract class (mainly interface) ~= mixin class
- Interface inheritance supports polymorphism
- Inheritance vs. delegation (to a contained object) or parameterized types (C++ template classes)

Clever selection/usage of patterns to support late binding (no hard coding for types): Design for Change

Abstract Factory

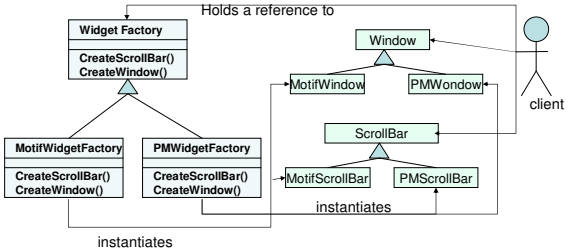
Intent: Provide an interface to create families of related objects without specifying concrete classes



Also known as: kit

Abstract Factory: widgets

- **Motivation**: e.g.User interface toolkit supporting multiple look-and-feel standards (motif / Presentation Manager)



- Portability across GUI standards: no hard-coding of widgets for a particular look&feel (by the application)

Abstract Factory: description continued

- Applicability
  - System independent of how its products are created, composed and represented
  - Configured with one of multiple families of products
  - Family of related objects should be used together, developer does not enforce this
  - A class library of products, with only revealing interfaces, no implementations.
- Participants
  - AbstractFactory: interface to create abstract objects
  - ConcreteFactory: implements creating objects
  - AbstractProduct: interface for a type of product object
  - Concrete Product: Product object: implements the interface
  - Client: uses only interfaces declared by "abstract" classes

Abstract Factory: description 3

- Collaborations
  - only one instance of a concrete factory will be used
  - AbstractFactory defers creation to ConcreteFactory
- Consequences
  - Isolates concrete classes
  - Makes product family exchanging easy
  - Promotes consistency among products
  - New kinds of products? difficult !
- Implementation..
- Sample Code ..

Abstract Factory: description 4

- Known Uses
  - InterViews and ET++ uses for different look and feel
- Related Patterns
  - Factory Method
  - Prototype
  - Singleton

Pattern Descriptions

- Abstract Factory: Create an interface for creating families of objects without specifying concrete classes
- Adapter:Convert the interface of a class to work with an incompatible class
- Bridge:Decouple abstraction from implementation (can vary independently)
- Builder: separate constrcution from representation (different representations)
- Chain of Responsibility: A request can travel a chain of objects until handled.
- Command: parameterize clients for a command, queue/log requests, undo..
- Composite: objects in a tree structure with part-whole relations

Pattern Descriptions 2

- Decorator: Attach dynamically, additional responsibilities to an object
- Façade: A unified interface for a related set of interfaces
- Factory Method: Interface to create an object: subclasses decide which class to instantiate
- Flyweight: Efficiently sharing of many small objects by sharing
- Interpreter: grammar representation + interpreter
- Iterator: traverse an aggregate object for access
- Mediator: an object through which others interact
- Memento: without violating encapsulation, capture other objects state

Pattern Descriptions 3

- Observer: one-to-many relation to dependents, to notify changes
- Prototype: Create new objects by copying a common prototype
- Proxy: Placeholder object for controlled access
- Singleton: Ensure one instance + global access
- State: Objects changes behavior when state changed
- Strategy: A family of algorithms – interchangable
- Template Method: A skeleton algorithm, details in the subclasses
- Visitor: Operation on the elements, new operation can be defined without changing elements classes