Strategy Design Pattern

Class centered on operations, instead of data
Strategy

- Multiple variants of one algorithm
- Different types of objects only differing in behavior
- **The key part of a class is its method, NOT its data**
  - Example: printer, sorter, comparator
  - The method works for multiple data types
Example

• Printers
  • Various font size, indentation, capitalization
Class diagram

Encapsulate algorithms into class

Program to an interface, not an implementation.

Open for extension, closed for modification.
Alternative solutions

• Switch-case and function pointers in C

• Super-class on the data side

• Template in C++

*Strategy design pattern is more flexible and easier to extend than the above alternative solutions*
Other examples

• Different sorting
• Different rendering
• ...
Template

• Provide a skeleton for similar algorithms
  • The key of the class is still operation, not data

• Example
Class diagram

FrameworkClass

+templateMethod()
+stepOne()
+stepTwo()
+stepThree()

stepOne(); stepTwo(); stepThree();

ApplicationClassOne

+stepTwo()

ApplicationClassTwo

+stepTwo()
Visitor

• How to add a class of operations for a variety of data classes?

• Example
  • Different operations for AST nodes
  • Different operations for Person (Female, Male)
In the AST tree example:

- The left side class hierarchy would be like this:
  - The super-class is “Expression”
  - The sub-classes are “constant”, “addition”, “multiplication”, etc. There are association relationships between these sub-classes and the Expression super-class, just like that in the composite pattern.

- The right side class hierarchy would be like this:
  - The subclasses would be things like “EvaluateExpression”, “PrintExpression”, “ComplexityOfExpression”.

The concrete types of the `Element` and `Visitor` objects have been "recovered". Perform the work appropriate for their pair of types.
Visitor

• Two class hierarchies: data & operations

• What is it good at?
  • If you add operations (Visitor classes), the interface of the Element classes remains unchanged

• What is it bad at?
  • If you add new Element sub-class, significant changes are needed for the Visitor side

• Double-dispatch
  • Imagine two dimensions of a function call
    • The exact algorithm
    • The type of data this algorithm works on
  • You will get chance to make choice along both dimensions dynamically, using visitor pattern
Creational design patterns
Factory Method

• Lets a class defer instantiation to subclasses
  • No need to decide which subclass I want to use statically

• Example
  • Date (US style, Europe style, Chinese style, ...)
  • Window
Class diagram

Date Example:
the “product” super class would be a “Date” super class
There will be three sub-classes “USDate”, “EUDate”, and “CHDate”
On the right side, the super class is “DateCreator”
The sub-classes of “DateCreator” would include three sub-classes, “USDateCreator”, “EUDateCreator”, and “CHDateCreator”. The factory-method would execute “return new USDate”, “return new EUDate”, and “return new CHDate”, respectively.

Factory design pattern is somewhat similar with Strategy design pattern
When to use factory design pattern?

• The type of the sub-class is determined at run time
• The type changes very infrequently once set
Abstract Factory

- For creating families of related or dependent objects without specifying their concrete classes

- Examples
  - Date, currency, data
  - Window, mouse, scroll bar, ...
Class diagram