Strategy Design Pattern

Classes 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.

Client ➔ «interface» Abstraction

+doSomething()

Open for extension, closed for modification.

ImplementationOne
+doSomething()

ImplementationTwo
+doSomething()
Alternative solutions

• If in C

• Super-class on the data side

• Template in C++
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
    • Imagine that we started with an interpreter implementation
    • Then, we want to add expression-printing functionality
    • Then, we want to add expression-complexity-estimation functionality
    • Then ...

The above changes would always change the original interpreter class.

*Any design that would allow us leave the original class untouched?*
Class diagram

Client

«interface» Element

ElementOne
+accept(in v : Visitor)

ElementTwo

«interface» Visitor
+visit(in e : ElementOne)
+visit(in e : ElementTwo)

VisitorOne
+visit(in e : ElementOne)
+visit(in e : ElementTwo)

VisitorTwo
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