CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2001
Doc 8 Strategy & Null Object
Contents

References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995, pp. 315-314

"Null Object", Woolf, in *Pattern Languages of Program Design 3*, Edited by Martin, Riehle, Buschmmann, Addison-Wesley, 1998, pp. 5-18

Reading

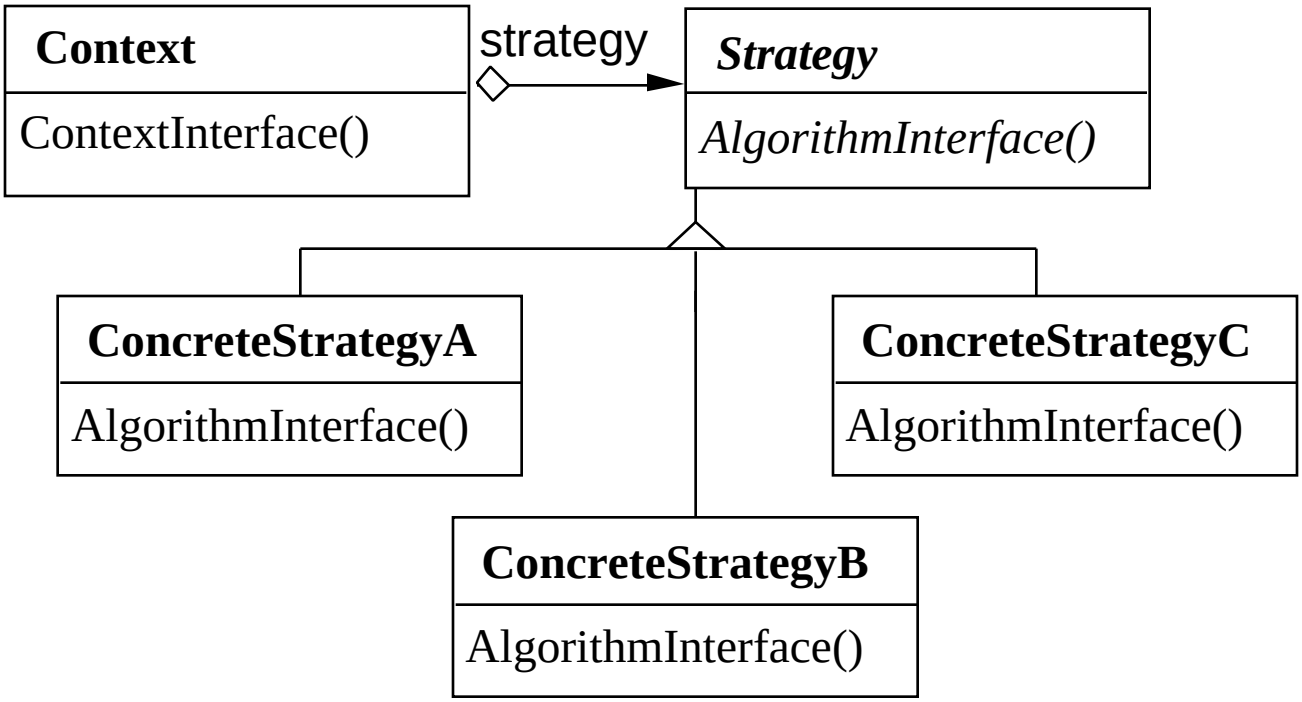
Design Patterns: pp. 315-314

Strategy
Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable

Strategy lets the algorithm vary independently from clients that use it

Structure



Examples

Sorting

Different types of sorts have different characteristics

Shellsort
- No extra space needed, Fast but not O(n*log(n))
- Very fast on nearly sorted data
- Does comparatively well on small lists

Quicksort
- Average case is O(n*log(n))
- Relatively poor performance on short lists
- Requires a stack of ~ log(n) in depth

MergeSort
- Worst case is O(n*log(n))
- Requires O(n) extra space
- Stable

Have a sorted list container, which one gives a sort algorithm

```
SortedList studentRecords = new SortedList( new ShellSort() );
studentRecords.add( “Sam” );

pubic class SortedList {
   Object[ ] elements;

   SortStrategy sorter;

   void sort( ) {
      sorter.sort( elements);
   }
}
```

Pattern Matching

Finding a pattern in text is a common operation

Find the first occurrence of the word “NullObject” in this set of notes after this line of text.

There are various algorithms one can use:

Brute Force
- Easy to implement
- Bad worst case, but good performance in practice

KMP
- Good worst case

Boyer-Moore
- Excellent worst case
- Very hard to implement

QuickSearch
- Easy to implement
- Good performance
- Good worst case

State Machines
- Very general

Could use a text object that has a pattern search object

Applicability

Use the Strategy pattern when

- You need different variants of an algorithm
- An algorithm uses data that clients shouldn't know about
- A class defines many behaviors, and these appear as multiple switch statement in the classes operations
- Many related classes differ only in their behavior

Consequences

- Families of related algorithms
- Alternative to subclassing of Context

  What is the big deal? You still subclass Strategy!

- Eliminates conditional statements

  Replace in Context code like:

```
switch  ( flag ) {
   case A: doA(); break;
   case B: doB(); break;
   case C: doC(); break;
}
```

  With code like:

```
strategy.do();
```

- Gives a choice of implementations
- Clients must be aware of different Strategies

```
SortedList studentRecords = new SortedList(new ShellSort());
```

- Communication overhead between Strategy and Context
- Increase number of objects

Implementation

- Defining the Strategy and Context interfaces

  How does data flow between them

  Context pass data to Strategy

  Strategy has point to Context, gets data from Context

- Strategies as template parameters

  Can be used if Strategy can be selected at compile-time and does not change at runtime

```
SortedList<ShellSort> studentRecords;
```

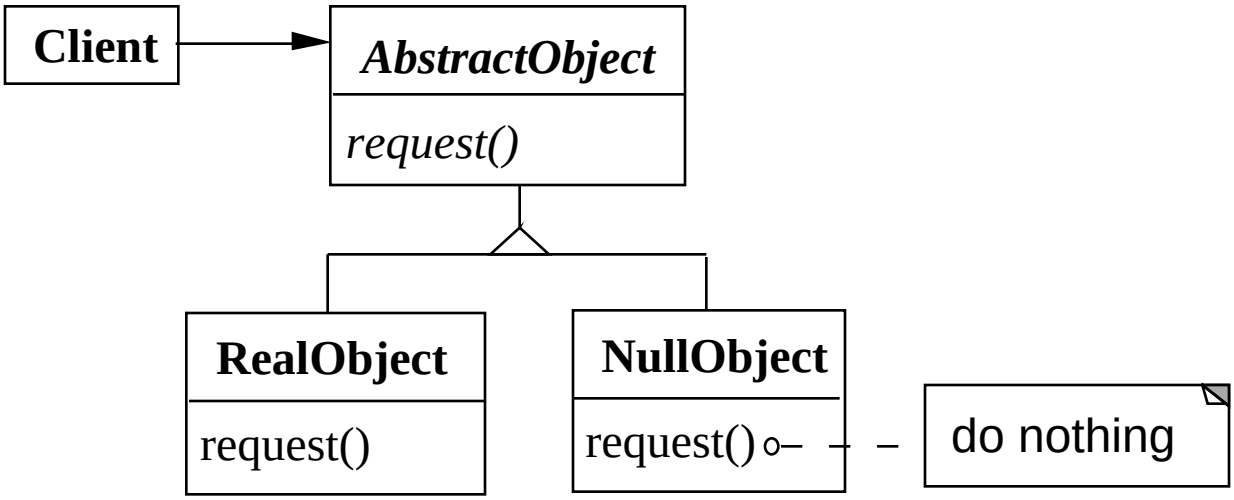
- Making Strategy objects optional

  Give Context default behavior

  If default used no need to create Strategy object

NullObject

Structure



NullObject implements all the operations of the real object,

These operations do nothing or the correct thing for nothing

Applicability

Use the Null Object pattern when:

- Some collaborator instances should do nothing
- You want clients to ignore the difference between a collaborator that does something and one that does nothing

  Client does not have to explicitly check for null or some other special value
- You want to be able to reuse the do-nothing behavior so that various clients that need this behavior will consistently work in the same way

Use a variable containing null or some other special value instead of the Null Object pattern when:

- Very little code actually uses the variable directly
- The code that does use the variable is well encapsulated - at least in one class
- The code that uses the variable can easily decide how to handle the null case and will always handle it the same way

Consequences
Advantages

- Uses polymorphic classes
- Simplifies client code
- Encapsulates do nothing behavior
- Makes do nothing behavior reusable

Disadvantages

- Forces encapsulation

  Makes it difficult to distribute or mix into the behavior of several collaborating objects

- May cause class explosion
- Forces uniformity

  Different clients may have different idea of what “do nothing” means

- Is non-mutable

  NullObject objects can not transform themselves into a RealObject

Implementation

Too Many classes

Eliminate one class by making NullObject a subclass of RealObject

Multiple Do-nothing meanings

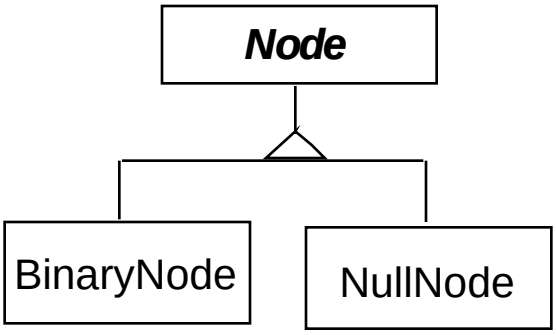
If different clients expect do nothing to mean different things use Adapter pattern to provide different do-nothing behavior to NullObject

Transformation to RealObject

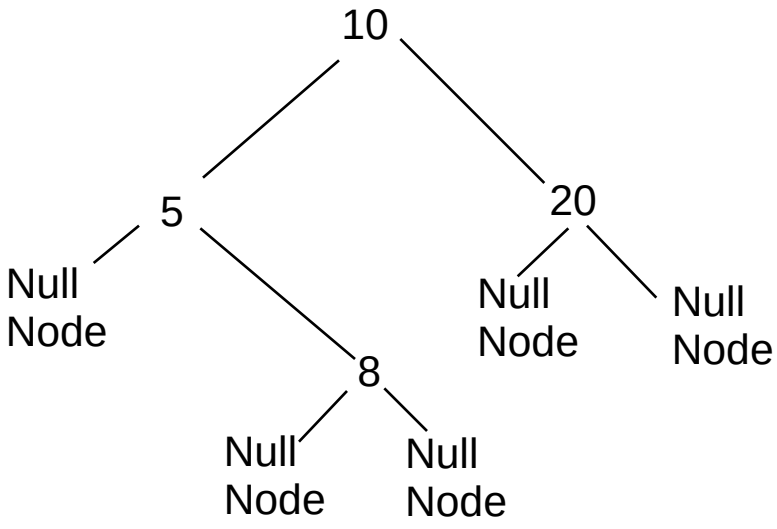
In some cases a message to NullObject should transform it to a real object

Use the proxy pattern

Binary Search Tree Example
Class Structure



Object Structure



Searching for a Key

```
public class BinaryNode extends Node {
   Node left = new NullNode();
   Node right = new NullNode();
   int key;

   public boolean includes( int value ) {
      if (key == value)
         return true;
      else if (value < key )
         return left.includes( value );
      else
         return right.includes(value);
   }
etc.
}

public class NullNode extends Node {
   public boolean includes( int value ) {
      return false;
   }
etc.
}
```

Comments on Example

- BinaryNode always has two subtrees

  No need check if left, right are null

- Since NullNode has no state just need one instance

  Use singleton pattern for the one instance

- Access to NullNode is usually restricted to BinaryNode

  Forces indicate that one may not want to use the Null Object pattern

  However, familiarity with trees makes it easy to explain the pattern

- Implementing an add method in NullNode

  Requires reference to parent or

  Use proxy