CS 635 Advanced Object-Oriented Design & Programming

Spring Semester, 2006
Doc 12 Strategy & Decorator
Mar 28, 2006

Copyright ©, All rights reserved. 2006 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

References

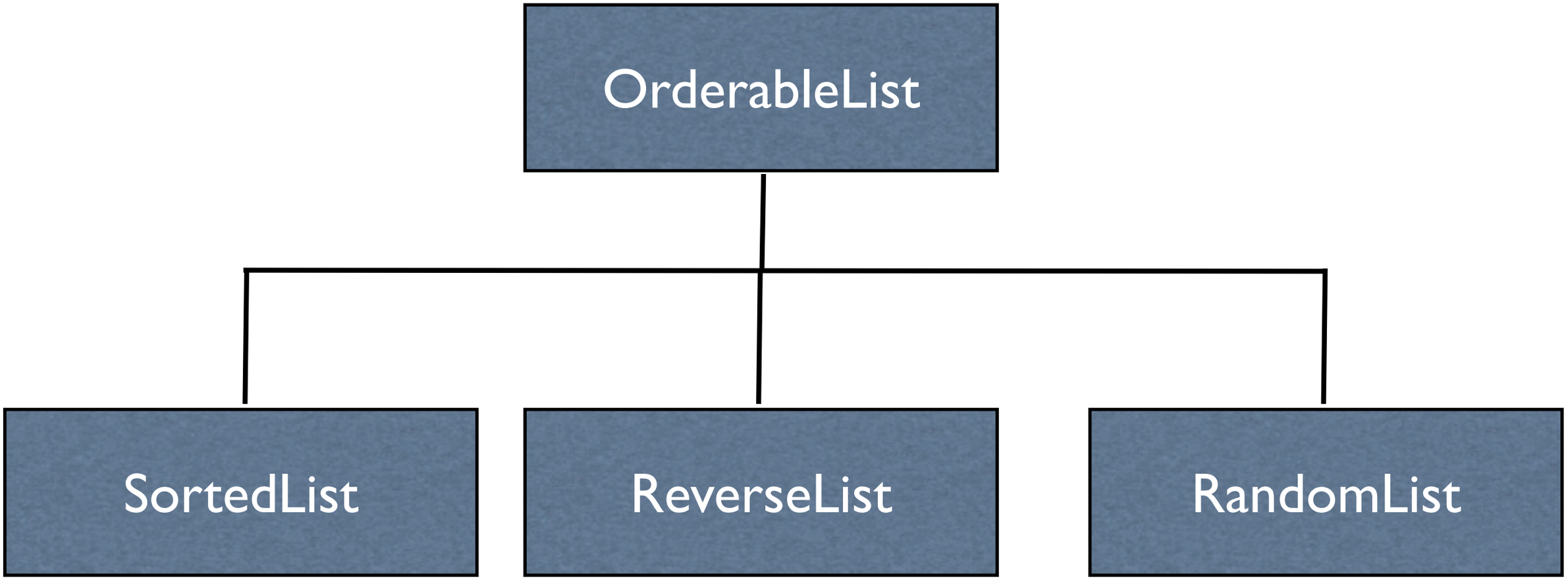
Design Patterns: Elements of Resuable Object-Oriented Software,
Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995, pp. 175-184, 315-324

Photographs used with permission from www.istockphoto.com

Favor
Composition
over
Inheritance

Orderable List

Sorted
Reverse Sorted
Random


OrderableList
SortedList
ReverseList
RandomList

One size does not fit all

Issue 1 - Orthogonal Features

Order

Sorted

Reverse Sorted

Random

Threads

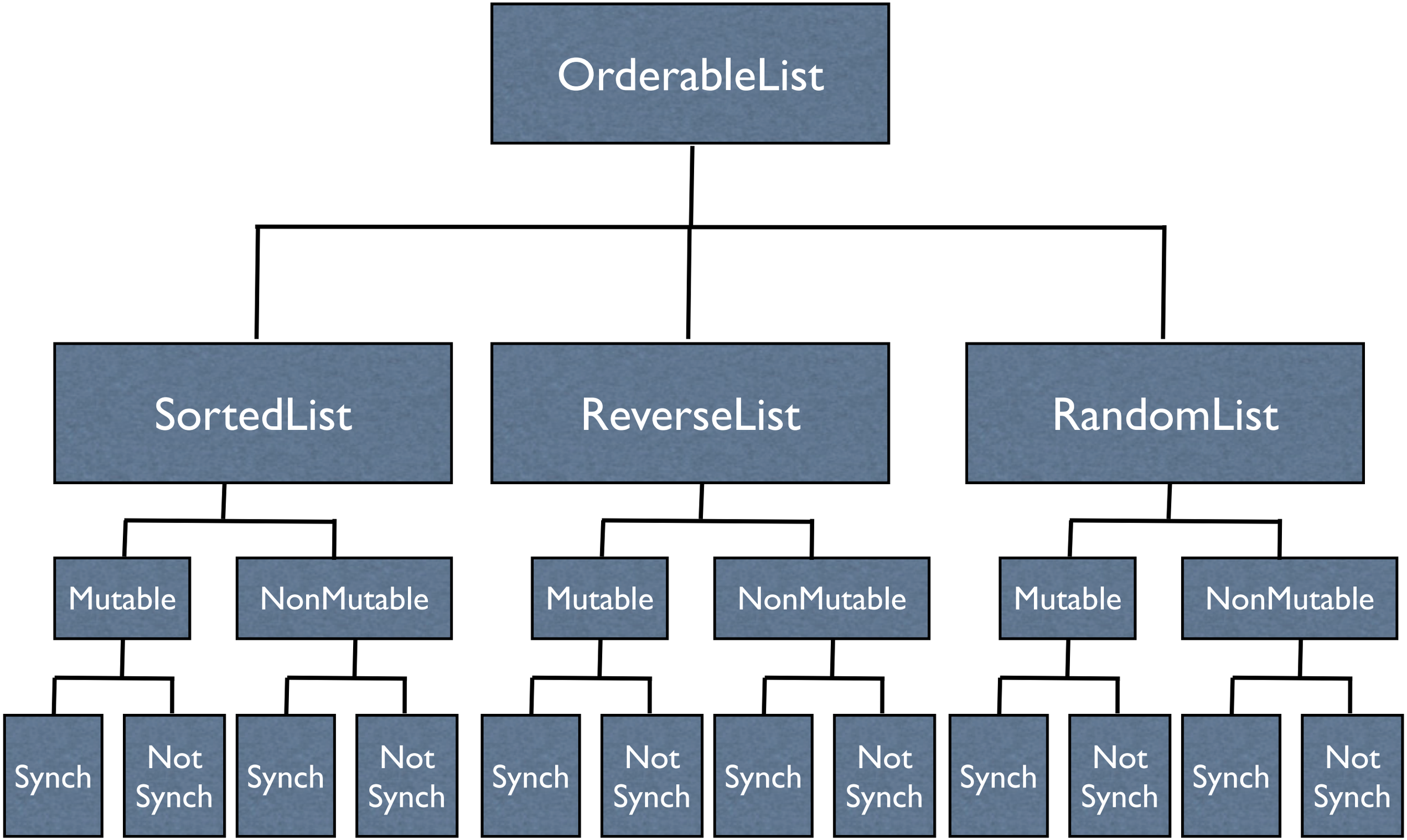
Synchronized

Unsynchronized

Mutability

Mutable

Non-mutable


OrderableList
SortedList
ReverseList
RandomList
Mutable
NonMutable
Mutable
NonMutable
Mutable
NonMutable
Synch
Not Synch
Synch
Not Synch
Synch
Not Synch
Synch
Not Synch
Synch
Not Synch
Synch
Not Synch

Issue 2 - Flexibility

Change behavior at runtime

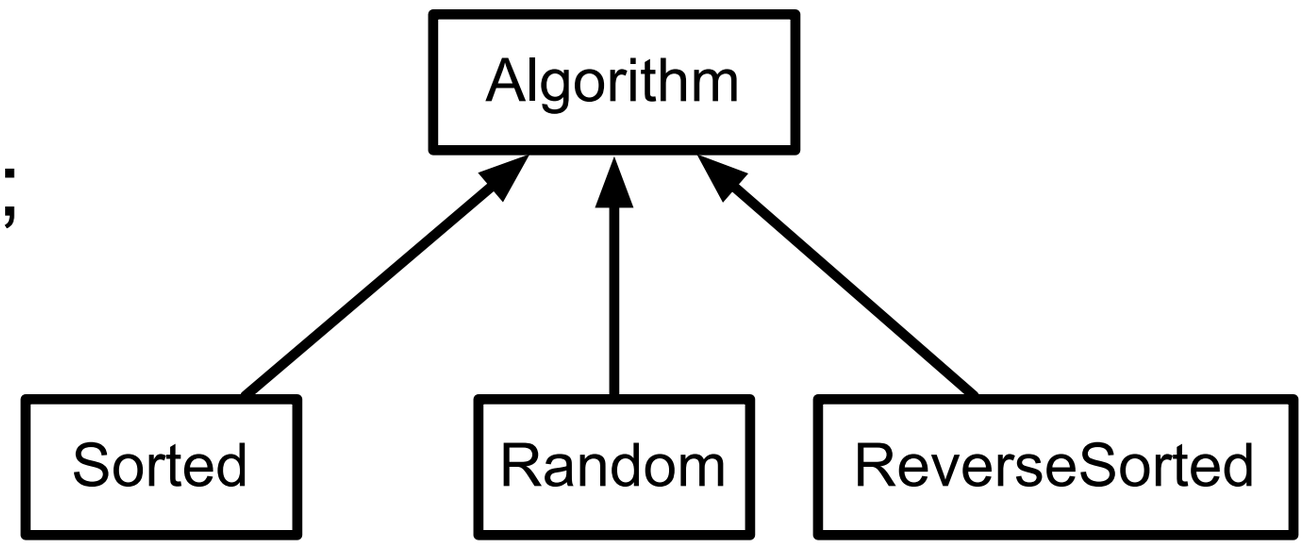
```
OrderableList x = new OrderableList();
x.makeSorted();
x.add(foo);
x.add(bar):
x.makeRandom();
```

Configure objects behavior at runtime

Strategy Pattern

```
class OrderableList {
    private Object[ ] elements;
    private Algorithm orderer;

    public OrderableList(Algorithm x) {
        orderer = x;
    }
```



The algorithm is the operation

Context contains the data

How does this work?

Prime Directive

Data + Operations

Example - Java Layout Manager

```
import java.awt.*;
class  FlowExample  extends Frame  {

        public FlowExample( int  width, int height ) {
                setTitle( "Flow Example" );
                setSize( width, height );
                setLayout( new FlowLayout( FlowLayout.LEFT) );

                for ( int label = 1; label < 10; label++ )
                        add( new Button( String.valueOf( label ) ) );
                show();
        }

        public  static  void  main( String  args[] ) {
                new  FlowExample( 175, 100 );
                new  FlowExample( 175, 100 );
        }
}
```

Example - Smalltalk Sort blocks

```
| list |
list := #( 1 6 2 3 9 5 ) asSortedCollection.
Transcript
        print: list;
        cr.
list sortBlock: [:x :y | x > y].
Transcript
        print: list;
        cr;
        flush.
```

Costs

Clients must be aware of different Strategies

Communication overhead between Strategy and Context

Increase number of objects

Benefits

Alternative to subclassing of Context

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();
```

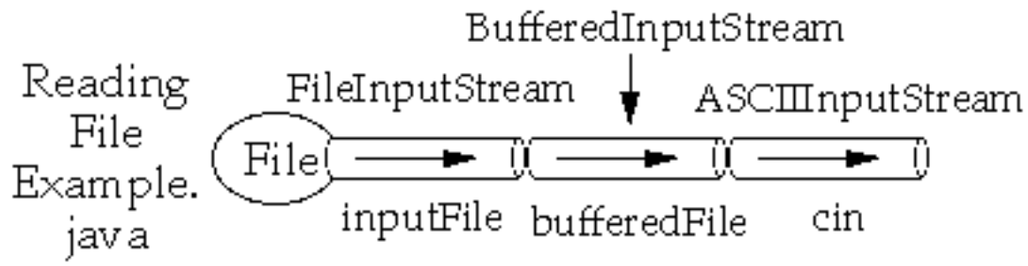
Gives a choice of implementations

Decorator Pattern

Adds responsibilities to individual objects

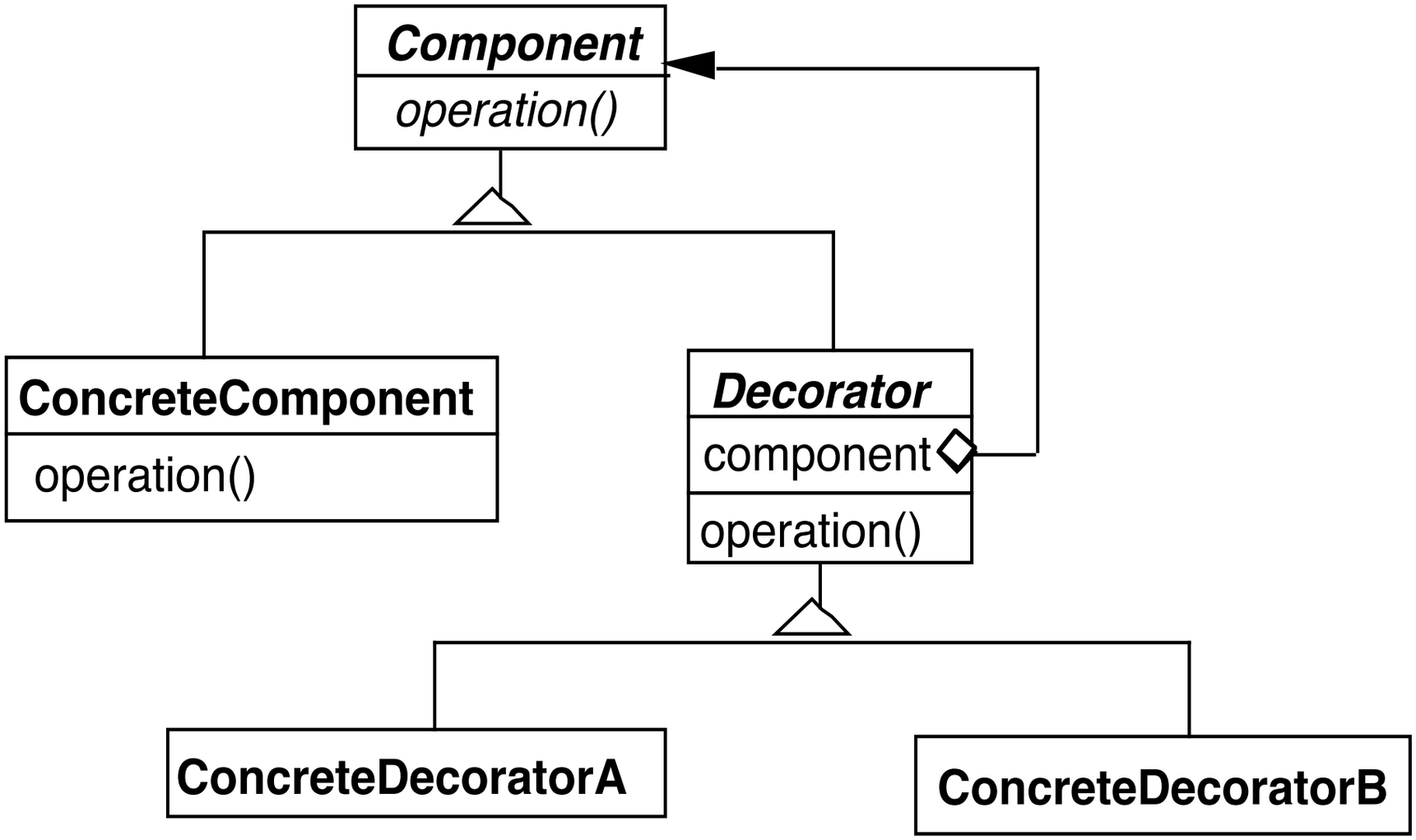
Dynamically

Transparently

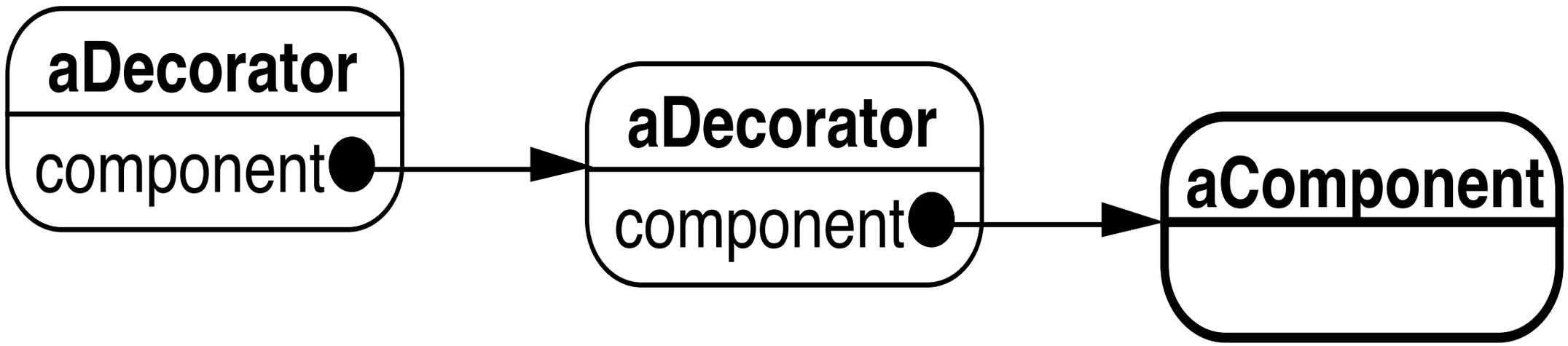



```
import java.io.*;
import sdsu.io.*;
class  ReadingFileExample
     {
     public  static  void  main( String  args[]  ) throws Exception
          {
          FileInputStream inputFile;
          BufferedInputStream bufferedFile;
          ASCIIInputStream  cin;

          inputFile = new FileInputStream( "ReadingFileExample.java" );
          bufferedFile = new BufferedInputStream( inputFile );
          cin = new ASCIIInputStream( bufferedFile );
```


Component
operation()
ConcreteComponent
operation()
Decorator
component
operation()
ConcreteDecoratorA
ConcreteDecoratorB

Decorator forwards all component operations



What is the difference between
Decorator & Strategy?