Your Final Bow

Last week, we learned about the Strategy pattern as a way to implement a `Play` class that can count the words in its text that satisfy an arbitrary test. The test is passed as an argument to the word-counting method.

The folks in the English department are pleased with us! They study texts to determine who the author was, based on stylistic criteria such as average word length, number of different words, etc. Our program allows them to do this with great ease. But...

They would like to be able to **combine** tests, such as "count all the 5-letter words that start with `'t'`".

You think to yourself, "Boy, this sounds familiar! I sure am happy I took 810:053!! I always knew that all my effort would be worth something some day."

You dig out your notes and implement a "compound test" class for your application.

Do it now, please.

```
public interface TestFeatureStrategy
{
   public boolean hasFeature( String s );
}

public class StartsWith implements TestFeatureStrategy
{
   private char targetChar;

   public StartsWith( char target )
   {
      targetChar = target;
   }

   public boolean hasFeature( String s )
   {
      if ( s == null || s.length() == 0 )
         return false;
      return s.charAt(0) == targetChar;
   }
}

public class OfLength implements TestFeatureStrategy
{
   private int targetSize;

   public OfLength( int target )
   {
      targetSize = target;
   }

   public boolean hasFeature( String s )
   {
      if ( s == null )
         return false;
      return s.length() == targetSize;
   }
}

public class IsAPalindrome implements TestFeatureStrategy
{
   public IsAPalindrome()
   {
      // ... no data to initialize
   }
```

```
   public boolean hasFeature( String s )
   {
      if ( s == null)
         return false;

      int length = s.length();
      if ( length < 2 )
         return true;

      int half = length/2;
      for ( int i = 0; i < half; ++i )
          if ( s.charAt(i) != s.charAt(length - 1 - i) )
             return false;

      return true;
   }
}

/* =============== FYI: THE CLIENT CODE =============== */

public int countWords( TestFeatureStrategy test ) ...
{
   BufferedReader inputFile =
      new BufferedReader( new FileReader( fileName) );
   String         buffer    = null;
   int            wordCount = 0;

   buffer = inputFile.readLine();
   while( buffer != null )
   {
      StringTokenizer words =
         new StringTokenizer( buffer );
      while( words.hasMoreTokens() )
      {
         String word = words.nextToken();
         if ( test.hasFeature( word ) )
            wordCount++;
      }

      buffer = inputFile.readLine();
   }

   return wordCount;
}
```

One Possible Solution

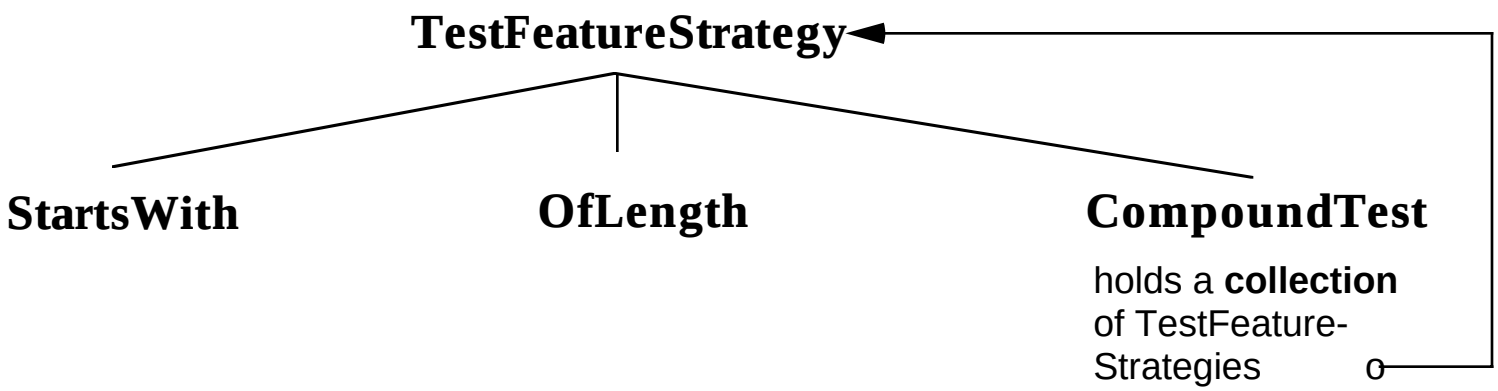
What we need is an object that acts like a single test but that does several tests. Each of the several tests is just a single test like the others.

We could use an idea like Decorator, but with a collection of "workers":

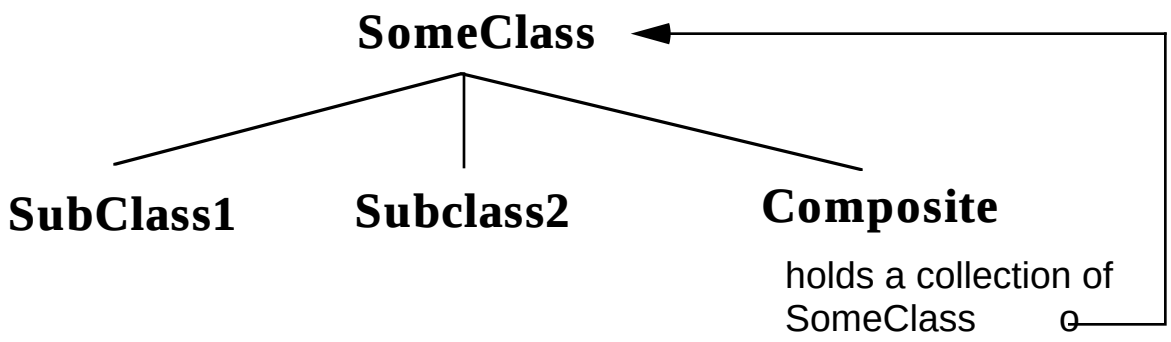
```
public class CompoundTest implements TestFeatureStrategy
{
   private Vector tests;

   public CompoundTest()
   {
      tests = new Vector();
   }

   public void addTest( TestFeatureStrategy t )
   {
      tests.addElement( t );
   }

   public boolean hasFeature( String s )
   {
      for ( int i = 0; i < tests.size(); i++ )
      {
          TestFeatureStrategy t =
                (TestFeatureStrategy) tests.elementAt( i );
          if ( !t.hasFeature( s ) )
             return false;
      }

      return true;
   }
}
```

How else might we create a `CompoundTest`? What would be the advantage of the new approach?

The New `TestFeatureStrategy` Hierarchy



The Composite Pattern



So, What's the Answer?

First, let's fix a couple of features from our original `Play` class:

```
StringTokenizer words =
   new StringTokenizer( buffer,
                          " .?!()[]{}|?\\,;:-\'\"\t\n\r"
                        );
while( words.hasMoreTokens() )
{
   String word = words.nextToken().toLowerCase();
   if ( test.hasFeature( word ) )
      wordCount++;
}
```

And then let's run our new compound test:

```
public class CompoundTestDemo
{
   public static void main( String[] args ) ...
   {
      Play hamlet = new Play( "hamlet.txt" );

      CompoundTest fiveLetterTeeWords = new CompoundTest();
      fiveLetterTeeWords.addTest( new StartsWith( 't' ) );
      fiveLetterTeeWords.addTest( new OfLength  (  5  ) );

      System.out.println(
         "The play 'Hamlet' contains "          +
         hamlet.countWords( fiveLetterTeeWords ) +
         " 5-letter words that start with 't'."  );
      System.out.println();
      System.out.println( "The play 'Hamlet' contains " +
                          hamlet.wordsOfLength( 5 )     +
                          " 5-letter words."            );
      System.out.println( "The play 'Hamlet' contains " +
                          hamlet.startWith( 't' )       +
                          " words that start with 't'." );

   }
}
```

```
/* ============================================================= */


math-cs > java CompoundTestDemo
The play 'Hamlet' contains 463 5-letter words that start with 't'.

The play 'Hamlet' contains 3785 5-letter words.
The play 'Hamlet' contains 4533 words that start with 't'.
```