Using Patterns to Help Students See the Power of Polymorphism

Supplement: Using the Strategy Pattern

Eugene Wallingford
University of Northern Iowa
wallingf@cs.uni.edu

SIGCSE Technical Symposium
February 24, 2001

1. Students implement a method named `int startsWith( char initial )` in a simple `Document` class.

```
public class Document
{
   private String fileName;

   public Document( String fileName )
   {
      this.fileName = fileName;
   }
   ...
}
```

2. We discuss a typical solution.

```
public int startsWith( char targetChar ) ...
{
   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 ( word.charAt( 0 ) == targetChar )
            wordCount++;
      }

      buffer = inputFile.readLine();
   }

   return wordCount;
}
```

3. Students implement a method named `int wordsOfLength( int initial )` in the same class.

   What must they change from their previous solution?

   Only the test on the loop counter!

4. Suppose now that we want to implement a suite of tests for lexical analysis?

   What must they change from their previous solution?

   Only the test on the loop counter!

5. Students propose ways to eliminate this unseemly duplication of code. They usually suggest that we subclass to implement specific counting behaviors:

```
public int countWords() ...
{  ...
   while( words.hasMoreTokens() )
   {
     String word = words.nextToken();
     if ( passesTest( word ) )
        wordCount++;
   }
   ...
}
```

Then we can write a subclass that implements the `passesTest` method:

```
// in class, say, WordsStartWith
public boolean passesTest( String word )
{
   return word.charAt(0) == targetChar;
}
```

6. We discuss why this approach (the Template Method pattern) comes up short in this situation.

7. Then we use `startsWith(char)` as an inspiration: parameterize the behavior that changes.

Make the test on the `String`
***a parameter* to the method.**

But how can we do that?

Remember that:

- Objects are data, too.
- Objects can do things!

So make the test an object.

8. Design a solution:

   - Provide a common interface for objects that compute a `boolean` function of a `String`.
   - Write classes that implement this interface for each kind of test.
   - Pass an instance of such a class to the `Document` whenever we ask it to count its words in a particular way.

9. Implement the solution:

   First, the test interface:

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

Then, tests as classes that implement the interface:

```
public class StartsWith
             implements TestFeature
{
   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;
   }
}
```

Then, Document's countWords method, which takes a TestFeature argument:

```
public int countWords( TestFeature 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;
}
```

Finally, the specific methods in `Document`, which invoke `countWords`:

```
public int startsWith( char targetChar )
{
   return countWords(
               new StartsWith(targetChar));
}
```

Now, we can ask a `Document` to count its words in a new way by implementing a new `TestFeature` class.

Regardless of the type of test on the `String`, all of the tests can be used by the `countWords` method because they all implement a common interface.