An Exercise

Suppose that we have written a class named `Play` that has a `String` instance variable named `fileName`.

A `Play`s respond to a `startsWith( char initial )` message by returning the number of `String`s in its `File` that begin with `initial`. The words in the file are separated by the usual white space characters.

Your task: write the `startsWith( char )` method.

```
public class Play
{
   private String fileName;

   public Play( String fileName )
   {
      this.fileName = fileName;
   }

   public int startWith( char targetChar )
                         throws IOException
   {
                         ?????

   }
}
```

Here is the tokenizer code from the previous page:

```
  StringTokenizer words = new StringTokenizer( buffer, " ," );
  int total = 0;
  while( words.hasMoreTokens() )
  {
     String number = words.nextToken();
     int    value  = Integer.parseInt( number );
     total += value;
  }
```

A Possible Solution

```
public int startWith( char targetChar ) throws IOException
{
   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;
}
```

Does It Work?

```
chaos> cat PlayDemo.java
public class PlayDemo
{
   public static void main( String[] args )
                       throws IOException
   {
      Play hamlet = new Play( "hamlet.txt" );
      System.out.println( "The play 'Hamlet' contains " +
                          hamlet.startWith( 't' )       +
                          " words that start with 't'." );
   }
}


chaos > java PlayDemo
The play 'Hamlet' contains 3603 words that start with 't'.
```

Another Exercise
(But an Easy One!)

Suppose now that I asked you to add to the `Play` class a method named `wordsOfLength( int initial )`. In response to this message, a `Play` returns the number of `String`s in the `File` named `fileName` of the given length.

What would we have to change from our previous solution?

Answer: *just the test on the loop counter!*

```
public int wordsOfLength( int targetLength )
                                    throws IOException
{
   // all the same until...

      while( words.hasMoreTokens() )
      {
         String word = words.nextToken();
          if ( word.length() == targetLength )
            wordCount++;
      }

   // ... and then all the same to finish
}
```

```
chaos > java PlayDemo
The play 'Hamlet' contains 4313 words of length 5.
```

Yet Another Exercise
(Still Easy!)

Suppose now that I asked you to add to the `Play` class a method named `numberOfPalindromes()`. In response to this message, a `Play` returns the number of `String`s in the `File` named `fileName` that read the same frontward and backward.

What would we have to change from our previous solution?

Answer: *just the test on the loop counter!*

```
public int numberOfPalindromes()
                                  throws IOException
{
   // all the same until...

      while( words.hasMoreTokens() )
      {
         String word = words.nextToken();
          if ( isAPalindrome( word ) )
            wordCount++;
      }

   // ... and then all the same to finish
}


protected boolean isAPalindrome( String word )
{
   ...
}
```

```
chaos > java PlayDemo
The play 'Hamlet' contains 1190 words that are palindromes.
```

Repeat After Me:

"Say it once and only once."

We know it.
We believe it.
We love it.

But how do we do it?

Actually, we have already done it. Instead of writing a method `startsWithT()` and a method `startsWithJ()` and a method `startsWithX()` and ..., we wrote one method named `startsWith(char)` that takes the starting character of interest as an argument.

`startsWith(char)` i*mplements the behavior*.

An argument to the method *parameterizes the behavior* to work for a specific character.

We did the same sort of thing with the `wordsOfLength()` method.

So the answer to "How can we do it?" is:

Make the test we use on the `String` *a parameter.*

Um, Excuse Me,
Professor Wallingford, ...

Then I wake up the next morning, go to my office as happy as can be, and find in my e-mail the following:

> Date: Wed, 18 Apr 2001 00:59:33 -0500 (CDT)
> From: Happy User (somestudent@cns.uni.edu)
> To: My Favorite Teacher (wallingf@cs.uni.edu)
> Subject: say it once and only once
>
> How exactly do I do ***that***??
>
> I mean, I know how to pass characters and numbers to functions because, like, they're data, ya know? How can I pass a function as an argument? How can a test be a parameter?
>
> Patiently awaiting enlightenment from my favorite professor...

Ah, Little One, you know the answer. You even know two possible answers, one of which is better than the other. You just don't realize that you know.

Idea #1: Objects are data, too.

Idea #2: Objects can do things!

Idea #3: The AWT does this.

One Possible Solution:
The Template Method Pattern

We could subclass `Play` to implement specific counting behaviors. We could create a template method named `countWords` with a hook for the testing method:

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

Then subclasses such as `WordsStartWithPlay` would implement the `passesTest` method in their own way:

```
public boolean passesTest( String word )
{
   return word.charAt( 0 ) == targetChar;
}
```

This solution works just fine. But it has one drawback. What is it?

A Better Solution

What changes in our set of applications is not the *kind of play*, but the **way we count words** in the query methods.

Why not make what changes—the test— an object?

So, for our `Play` problem:

- 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 `Play` whenever we want to count the words in a particular way.

This is the **Strategy** pattern. (Section 15.3)

We have encountered it before, in the AWT. A `Container` holds a `LayoutManager`, which is an algorithm that lays out the Container's items on the screen.

Making tests and functions and whole algorithms into objects that can be created, passed, and replaced is a common idea in object-oriented programming. The result is much more flexible programs!

Using the Strategy Pattern in our Word-Counting Program

First, let's factor out what is different about our solutions. The "variable" here is determining whether a certain word has a certain characteristic.

We implement this **strategy** using an interface:

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

We can then implement different checking strategies as classes. For example:

```
public class StartWith
       implements TestFeatureStrategy

public class OfLength
       implements TestFeatureStrategy

public class Palindrome
       implements TestFeatureStrategy
```

Each of these classes will implement the `hasFeature` method in its own way.

Implementing the Strategy Object

For example, here's how we might implement one:

```
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;
   }
}
```

Any class that needs to test strings to see whether they start with a particular letter can create an instance of `StartsWith`:

```
StartsWith test = new StartsWith( 't' );
```

... and send it a message:

```
if ( test.hasFeature( "the" ) ) ...
```

Using the Strategy in Play, Part 1

First, let's extract the common parts of our previous three solutions into a word-counting method:

```
public int countWords( TestFeatureStrategy test )
                  throws IOException
{
   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;
}
```

This is the code that we re-wrote before...

Using the Strategy in `Play`, Part 2

Next, we implement the specific methods we need:

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

And:

```
public int wordsOfLengths( int targetSize )
{
   return countWords( new OfLength( targetSize ) );
}
```

And:

```
public int numberOfPalindromes()
{
   return countWords( new Palindrome() );
}
```

Now, we can ask a `Play` to count its words in any way we please, simply by implementing the `TestFeatureStrategy` classes that we proposed above.

Writing such a class isn't much more work than writing a method, because that is all there is!

Exercise: Implementing a Strategy

Write the class `OfLength` that tests strings to see if they are a particular length.

```
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 size )
   {
      targetSize = size;
   }

   public boolean hasFeature( String s )
   {
      return s.length() == targetSize;
   }
}
```

It's that easy!