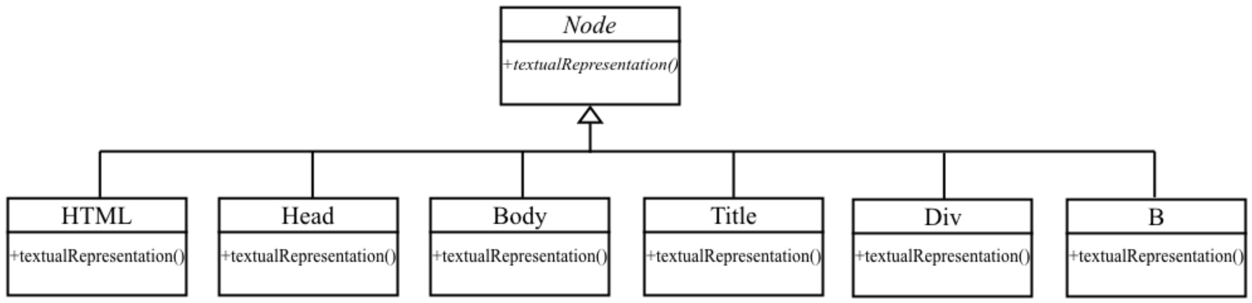
CS 5500 Homework 3: Design Patterns

(Assigned: March 21, Due: April 4 23:59pm)

Frank Tip and Mike Weintraub

Homework 3 consists of several programming tasks that involve the application of design patterns. Unless explicitly stated otherwise, your solutions may not rely on classes from the Java standard collections.

1. Abstract Factory (25 points)



The UML Class Diagram shown in the above figure shows a hierarchy of classes that model HTML documents (only a very small subset of HTML is modeled). The classes in this hierarchy are sufficient to model the following HTML page:

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <div id="first" class="foo">a</div>
    <b><div id="second" class="bar">b</div></b>
    <div id="third" class="foo">c</div>
  </body>
</html>
```

Each class in the hierarchy overrides the abstract method `Node.textualRepresentation()` so that calling `textualRepresentation()` on a node computes a string that reflects the entire subtree rooted at that node.

(a) Implement the class hierarchy and create a test suite for it. You should use the type `java.util.Map<String,String>` to store attributes associated with nodes. The following JUnit test should be included in your test suite, and it should pass.

```
  @Test
  public void test1(){
    Map<String,String> divAtts = new HashMap<String,String>();
    divAtts.put("id", "second");
    divAtts.put("class", "bar");
    Div div = new Div(divAtts, "b");
    Map<String,String> noAttributes = new HashMap<String,String>();
    B b = new B(noAttributes, div);
    assertEquals(b.textualRepresentation(), "<b><div class=bar id=second>b</div></b>");
  }
```

Please add some additional tests to your test suite, including one that checks that the correct textual representation is computed for the HTML page shown above. Please ensure that all your code is properly documented. Place your code in a package "`html`" and your test suite in a class "`HTMLTests`" in a package "`tests`".

(a) Now create a new version of the previous application in which all nodes are created using an instance of the "Abstract Factory" design pattern. We suggest that you use the names `AbstractHTMLNodeFactory` and `StandardHTMLNodeFactory` for the classes involved in the pattern. Place your code in a package "`html_factory`" and your tests in a class "`HTMLFactoryTests`" in package "`tests`".

(b) Now define a class `LoggingHTMLFactory` that defines another concrete factory that prints the textual representation of each node as it is being created. Please create another version of the previous test suite that uses this factory. Place this test suite in a class `LoggingHTMLFactoryTests` in package "`tests`".

(c) How much work was it to modify your tests to use a `LoggingHTMLFactory` instead of a `StandardHTMLNodeFactory`? Describe the benefits of the Abstract Factory design pattern in your solution (in 5 sentences or less).

2. Iterators (25 points)

A set of natural numbers can be represented compactly using an array of int values. The basic idea is to use the ith bit to indicate whether the number i is in the set. In Java, an int is represented using 32 bits, so in this representation the set { 0, 1, 4, 7, 31 } is represented by the binary number 10000000000000000000000010010011. For sets containing larger numbers, an array of ints can be used, where the first element represents bits 0-31, the second represents bits 32-63, and so on. Below, an outline is given for a class `BitVector` and an interface `Iterator` to which it refers.

```
public interface Iterator<T> {
   boolean hasAnotherElement();
   T nextElement();
}

public class BitVector {
  public boolean get(int i){ ... } // Determine if the bit at position i is set.
  public void set(int i){ ... } // Set the bit at position i.
  public void clear(int i){ ... } // Clear the bit at position i.
  public void addAll(BitVector b){ ... } // Set the bits in the argument BitVector b.
  public Iterator<Integer> iterator(){ throw new Error("unimplemented"); }
  public int size(){ ... }

  private int[] words;
}
```

(a) Complete the implementation of `BitVector` by adding bodies to all methods declared above *except the* `iterator()` *method*. Please ensure that all code is properly documented, and provide a JUnit test suite `BitVectorTests` that thoroughly tests your `BitVector` class.

(b) Now implement the `iterator()` method. You may add additional private classes, methods and fields if needed. Extend your test suite to test your iterator.

(c) Describe the benefits of applying the "Iterator" design pattern in this solution (in 5 sentences or less).

3. Adapter (25 points)

Below, an interface `Set` is defined, and a partial outline is given for a class `StringSet` that represents a set of `String` values.

```
interface Set<T> {
   void add(T t); // add an element to the set
   void addAll(Set<T> s); // add all elements in the argument set
   void remove(T t); // remove an element from the set
   int size(); // return the number of arguments in the set
   Iterator<T> iterator(); // return an iterator over the set
}

public class StringSet implements Set<String> {
  public void add(String s){ ...  }
  public void addAll(Set<String> s){ ... }
  public void remove(String s){ ...  }
  public int size(){ ... }
  public Iterator<String> iterator(){ ... }
  private BitVector adaptee = new BitVector();
}
```

a) Implement `StringSet` using the "Adapter" design pattern, where your previously developed `BitVector` plays the role of Adaptee. You must complete the implementations of all of the above methods, ensure that all code is properly documented, and provide a JUnit test suite StringSetTests that thoroughly tests your `StringSet` class. A few hints:
   - You will need to maintain an index that associates an integer value with each `String` that is stored in a `StringSet`. To maintain such mappings, you may use the class `java.util.HashMap`.
   - Your implementation does not need to concern itself with "garbage-collection" in situations where strings that are referenced in the index do not occur in any StringSet.

b) Describe the benefits of using the Adapter design pattern in your solution (in 5 sentences or less).

4. Visitors (25 points)

For this task, we will continue working with your solution to question 1(c).

```
public interface NodeVisitor {
  void visitHTML(HTML h);
  void visitHead(Head h);
  void visitBody(Body b);
  void visitTitle(Title t);
  void visitDiv(Div d);
  void visitB(B b);
}
```

a) Using the "Visitor" design pattern and the `NodeVisitor` interface defined above, define a class `NodeCountVisitor` that counts the number of times each type of node occurs in a document. Create a test suite `HTMLVisitorTests` that tests your `NodeCountVisitor`.

b) Using the "Visitor" design pattern and the `NodeVisitor` interface defined above, define a class `AttributeSearchVisitor` that finds node(s) that have an attribute with a specified name and value. Using your answer to question 3, the solution should be computed as a `StringSet` where the elements of the set are the textual representations of the nodes that meet the search criterion. Add tests to your `HTMLVisitorTests` suite to test this visitor as well.

c) Describe the benefits of using the "Visitor" design pattern in your solution (in 5 sentences or less).