Comp 212 - Intermediate Programming EXAM #1 February 11, 2004
Rice University - Instructors: Cox & Nguyen NAME: ______________________

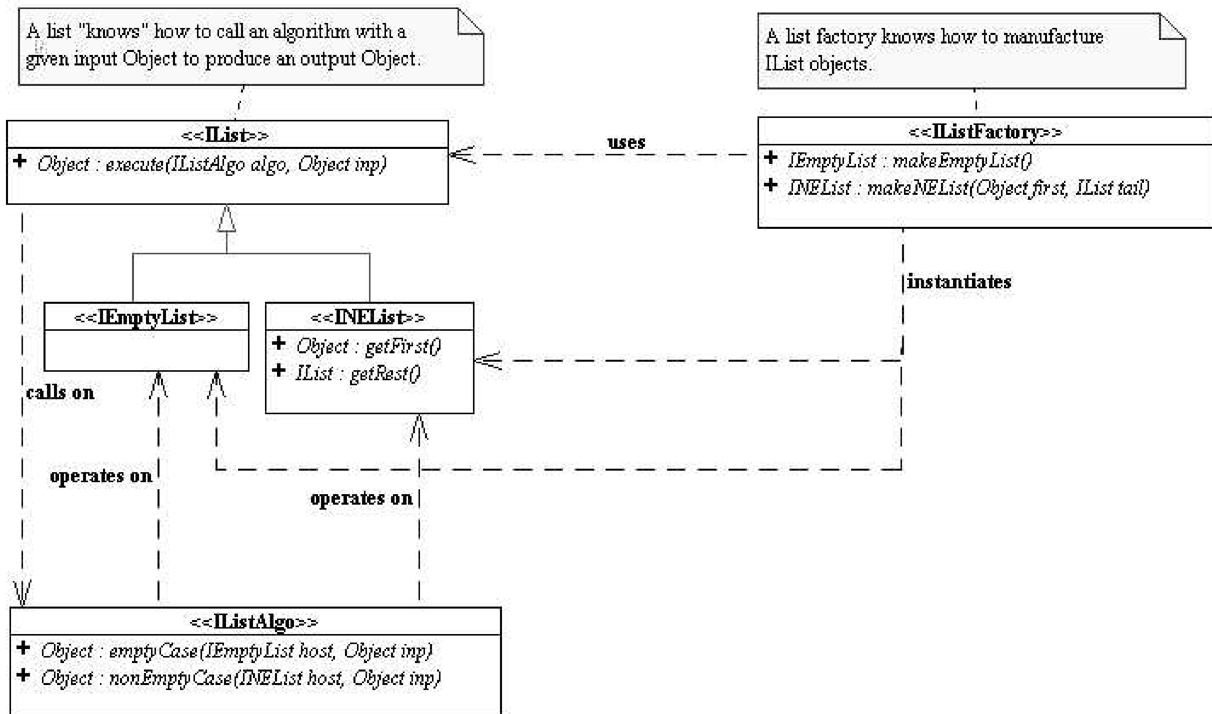
Instructions

1. This exam is conducted under the Rice Honor Code. It is a closed-notes, closed-book exam.
2. Fill in your name on every page of the exam.
3. If you forget the name of a Java class or method, make up a name for it and write a *brief* explanation in the margin.
4. You are expected to know the syntax of defining a class with appropriate fields, methods, and inheritance hierarchy. You will not be penalized on trivial syntax errors, such as missing curly braces, missing semi-colons, etc, but do try to write Java code as syntactically correct as possible. We are more interested in your ability to show us that you understand the concepts than your memorization of syntax!
5. Write your code in the most object-oriented way possible, that is, with the fewest number of control statements and no checking of the states and class types of the objects involved.
6. In all of the questions, feel free to write additional helper methods or visitors to get the job done.
7. Make sure you use the Singleton pattern whenever appropriate. Unless specified otherwise, you do not need to write any code for it. Just write "singleton pattern" as a comment.
8. For each algorithm you are asked to write, 90% of the grade will be for correctness, and 10% will be for efficiency and code clarity.
9. You have two hours and a half to complete the exam.

Please State and Sign your Pledge:

1) 15	2a) 15	2b) 15	3) 15	4a) 5	4b) 5	4c) 15	4d) 15	TOTAL 100

For your convenience, below is the UML class diagram for the scheme list framework with an abstract factory studied in class. You are free to use this list framework without explanation/implementation.



1. Write an IListAlgo, called First2Last, that moves the first element of an IList to the end of the list and returns the resulting list. That is: suppose R is the result of L executing First2Last, then if a is the first element of L then a is the last element of R. Thus, when the host list is empty, the visitor should return the empty list itself.

2. Design an object model for the following *immutable* data structure that represents lists of lists, called generalized lists.

   A generalized list, `GenList`, is a list that can contain lists as data objects.

   - `EmptyGenList` is a `GenList`; it contains no data.
   - `NonEmptyGenList` is a `GenList`; it contains a substructure called rest that is a `GenList`.
   - `AtomGenList` is a `NonEmptyGenList`; it contains a data object called first, which is simply an `Object` that is not a `GenList`.
   - `NonAtomGenList` is a `NonEmptyGenList`; it contains a data object called first, which is a `GenList`.

   Your class design should make use of the composite and visitor design patterns to decouple the data structure from the algorithms and achieve the highest degree of flexibility and extensibility. Use the singleton pattern wherever is appropriate. Do not apply the abstract factory pattern to this problem.

   a) Draw the corresponding UML class diagrams. The class diagrams should clearly show all the fields with their types, methods with their parameter lists and their return types, and constructors with their parameter lists. The field and method names should be self-explanatory and declared with appropriate public, private, protected access specifiers. To save space and time, show only the abstract methods of the super classes but not the concrete implemented methods in the subclasses. Use comment boxes to indicate the various design patterns in the diagram.

b) Write a visitor called CountObjects to compute the total number of data objects in a `GenList`.

3. "Higher order" functions on a list are operations on lists that take other functions as input parameters.

foldl ("fold-left") is an operation on a list, aList which has a first and rest, a value, b, and a function, f of two variables, defined recursively as follows:

foldl(empty, b , f) = b

and

foldl(aList, b, f) = foldl(rest, f(first, b) ,f)

For example, suppose

aList = (x_0, x_1, x_2, … x_{n-2}, x_{n-1}, x_n)

then

foldl(aList, b, f) = f(x_n, f(x_{n-1}, f(x_{n-2}, …f(x_2, f(x_1, f(x_0, b)))…)))

Note that if the list is empty, the result of foldl is simply b.

The following interface represents an abstract function of two arguments.

```
public interface ILambda2 {
  /**
  * Applies the function with the two given input arguments.
  * @param x the first parameter.
  * @param y the second parameter.
  * @return the value of this function applied to the two given input arguments.
  */
  public Object apply(Object x, Object y);
}
```

Write an IListAlgo called Foldl to implement the foldl operation on IList. Pass an ILambda2 to the constructor of Foldl.

In computing, a queue is a container structure with a restricted access policy, that is there are rules that restrict how data objects can be inserted and removed from the container. For this question, you will write a class Queue that implements a queue structure with a First-In, First-Out (FIFO) access policy. In general, a FIFO queue supports two operations, dequeue and enqueue. Enqueue adds an object to the queue and dequeue removes and returns the object that has been in the queue for the longest time. It is from these behaviors that the FIFO queue receives its name: the first object placed in the queue is the first object taken out of the queue. To receive credit, your implementation of class Queue must use two ILists that we call "*entrance*" and "*exit*" in the following way: An object is enqueued by inserting it at the front of the IList *entrance*. An object is dequeued by removing it from the front of the IList *exit*. If, however, the IList *exit* is empty, a new IList *exit* is constructed by reversing the contents of the IList *entrance*. At the same time, the new IList *entrance* becomes the empty list.

Notes:

- If the FIFO queue is empty, any attempt to dequeue an object should throw an exception. (Any type of exception is fine.)
- Use the factory interface for constructing ILists.
- Be sure to declare any fields, methods, constructors or classes as private, public, or protected as appropriate.
- In parts (c) and (d) below you may define helper visitors if desired.

a) Write the data fields and a constructor for class Queue.

b) Write the method enqueue. This method should take a value of type Object as its sole parameter and return void.

c) Write an IList visitor called "ToExit" as an inner class of class Queue. ToExit should be applied when the IList *exit* is empty. It should both construct the new IList *exit* as the reverse of the old IList *entrance* and make the new IList *entrance* become the empty list.

d) Write an IList visitor called "ServeQ" as an inner class of class Queue and the method dequeue that uses this visitor. ServeQ should remove and return the object at the front of the IList *exit*, invoking the visitor ToExit if the IList *exit* is empty. The method dequeue takes no parameters and returns a value of type Object.

A blank page for writing code…