Comp 212 - Intermediate Programming EXAM #1 February 12, 2003
Rice University - Instructors: Cox & Nguyen NAME: _____

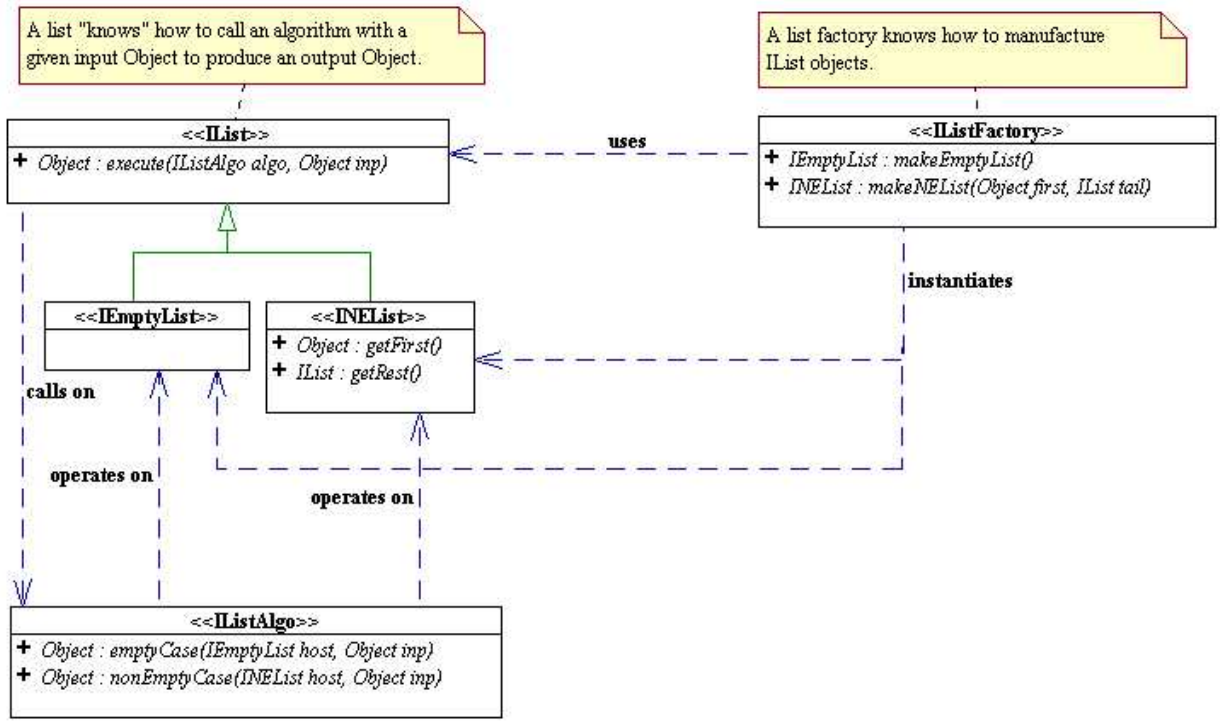
Instructions

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 may use as helpers the following visitors from the lectures and the homeworks: GetLength, GetMin, GetSum, ToString, Reverse, MakeClone, LastElement, GetNth, FirstNElements, Concatenate, MinFront2 without explanation/implementation.
10. You have two hours and a half to complete the exam.

Please State and Sign your Pledge:

1) 20	2) 15	3a) 15	3b) 15	4) 35	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 LastN, to return the list that contains the last N elements of an IList. Assume N >= 0. If N > the length of the list, throw an exception.

2. Assume the host list contains Integer elements. Write an IListAlgo, called `SelectionSort`: to sort the host list in ascending order according to the following algorithm.

   - Empty case: return the empty list (since an empty list is sorted).
   - Non empty case: find and move the minimum to the front of the host and recur on the rest of the resulting list.

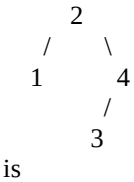
3. In the same object-oriented style as the list, we define an immutable binary tree as follows. There is an abstract notion of a binary tree. Its one behavior, execute(), is a visitor pattern "hook". An empty binary tree is a binary tree. It has no data. A non-empty binary tree is a binary tree. It has a data object, a left subtree called left, and a right subtree called right. Left and right are themselves binary trees. The non-empty binary tree has three behaviors for "getting" data, left, and right. This design translates to the following collection of Java interfaces.

```
// Represents a binary tree that knows how to call an algorithm IBTAlgo.
interface IBiTree {
    Object execute(IBTAlgo algo, Object inp);
}

// Represents the empty binary tree.
interface IMTTree extends IBiTree {
}

// Represents the non-empty binary tree.
// It has a data element,
// a IBiTree substructure called left, and
// a IBiTree substructure called right.
interface INETree extends IBiTree {
    Object getDat();
    IBiTree getLeftSubtree():
    IBiTree getRightSubtree();
}

// Represents an algorithm that operates on an IBiTree.
// Visitor for IBiTree.
interface IBTAlgo {
    Object emptyCase(IMTTree host, Object inp);
    Object nonEmptyCase(INETree host, Object inp);
}
```

continued on next page…

a) Write a visitor IBTAlgo called IsLeaf that returns Boolean.TRUE if the host is a leaf node, Boolean.FALSE otherwise. A tree is said to be a leaf node when both of its subtrees are empty. The algorithm must not check for emptiness of the subtrees. Hint: ask the subtrees for help!

b) Write a visitor IBTAlgo called InFixOrder that returns an IList containing all the elements in the host tree in the infix order. Infix ordering is defined recursively as follows: all elements in the left subtree are listed in infix order, followed by the current element, followed by all the elements in the right subtree listed in infix order. For example, the infix ordering of the tree:



is

1 2 3 4

4. There is an abstract notion of a robot. Robots are manufactured by a factory. Each factory produces its own kind of robots. Each robot remembers its current location, its factory of manufacture and has a unique serial number. (In other words, no two robots, regardless of factory have the same serial number.) Robots have a behavior "go to a location", as well as "getter" methods. A factory has a location and knows all of the robots it has made. A location is an object having a longitude and a latitude. Factories have two behaviors: "make a robot" and "recall robots". Occasionally, the government is given a failed robot and discovers a design defect in that robot. At which point, the government requests the factory that made the robot to recall all such robots. A recall is defined as directing all robots made by that factory to go to the location of the factory. Your task is to produce an object-oriented design simulating robots, factories and locations. Express your design in Java code. Note that a factory may need additional data structures to keep track of its robots.