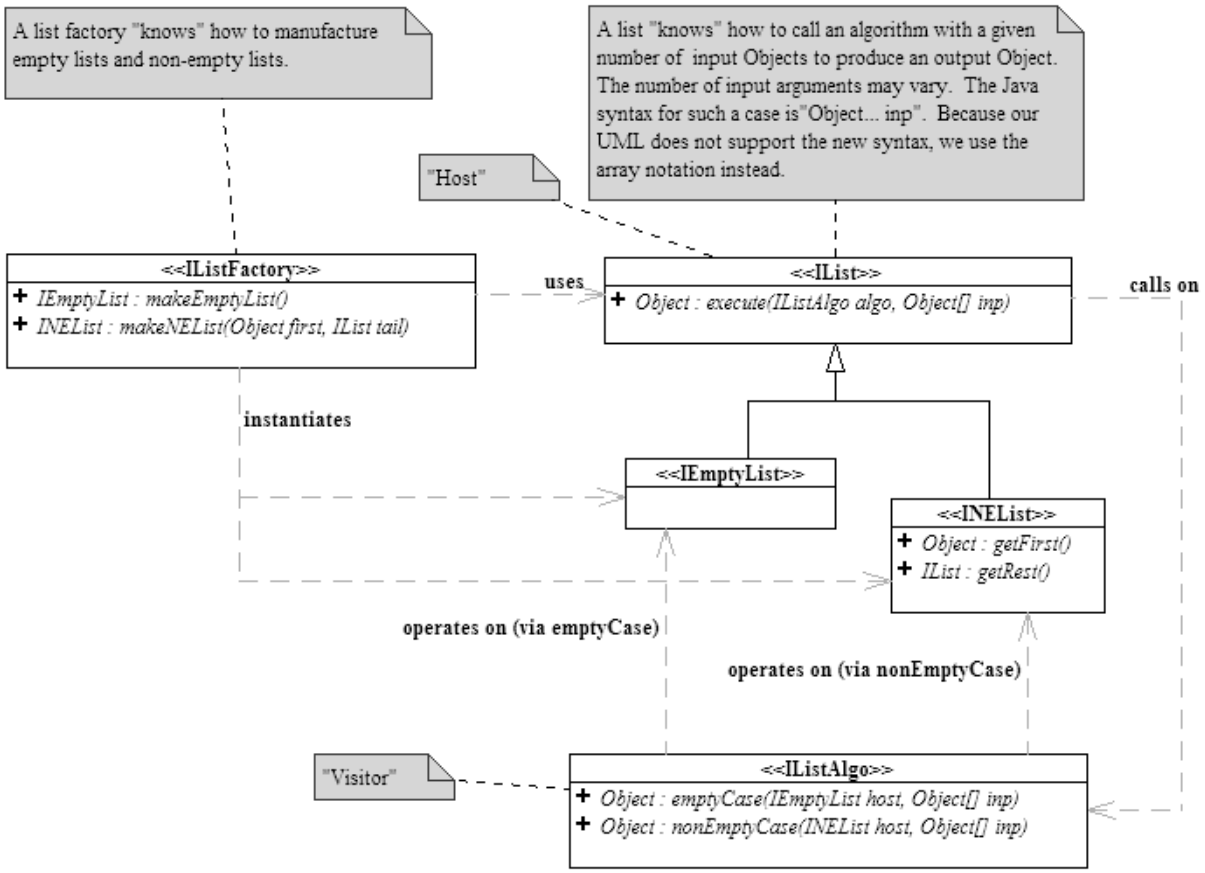
**Instructions**

1. This is an open-notes, open-book, open-computer, open-Internet exam.
2. **You have 1.5 hours to complete this exam.**
3. Please make sure you have all 16 pages of this exam.
4. You will not be penalized on trivial syntax errors, such as a missing parenthesis. Multiple errors or errors that lead to ambiguous code will have points deducted, however.
5. In all of the questions, feel free to write additional helper methods to get the job done.
6. The emphasis is on correctness of the code, not efficiency or on simply generating the right result.
7. You are free to use any code that was given to you in the lectures and labs.
8. You do not have to write generic code in this exam.

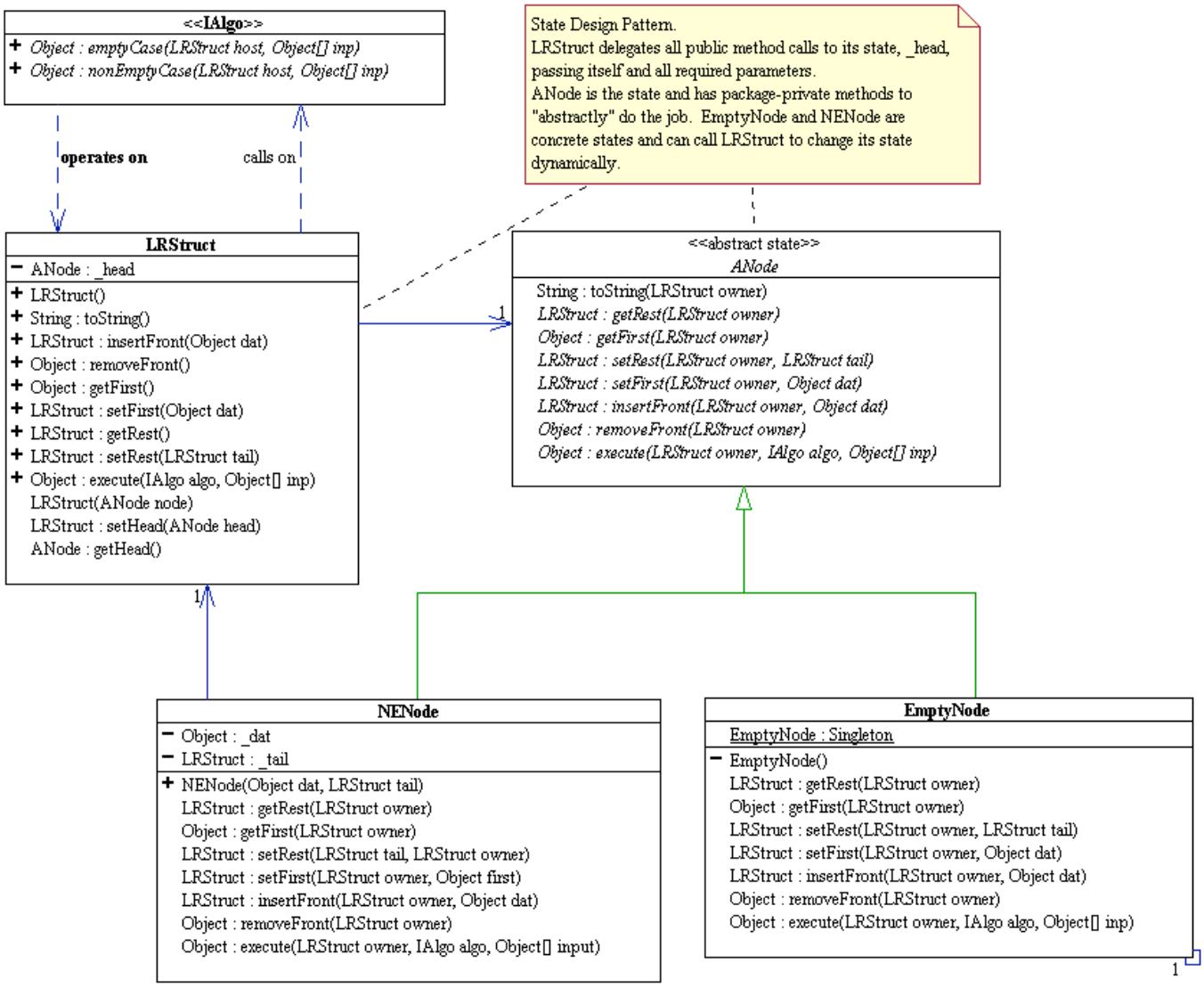
**Please write and sign the Rice Honor Pledge here:**

| 1a | 1b | 1c | 2 | Total |
|---|---|---|---|---|
| /40 pts | /10 pts | /10 pts | /40 pts | /100 pts |

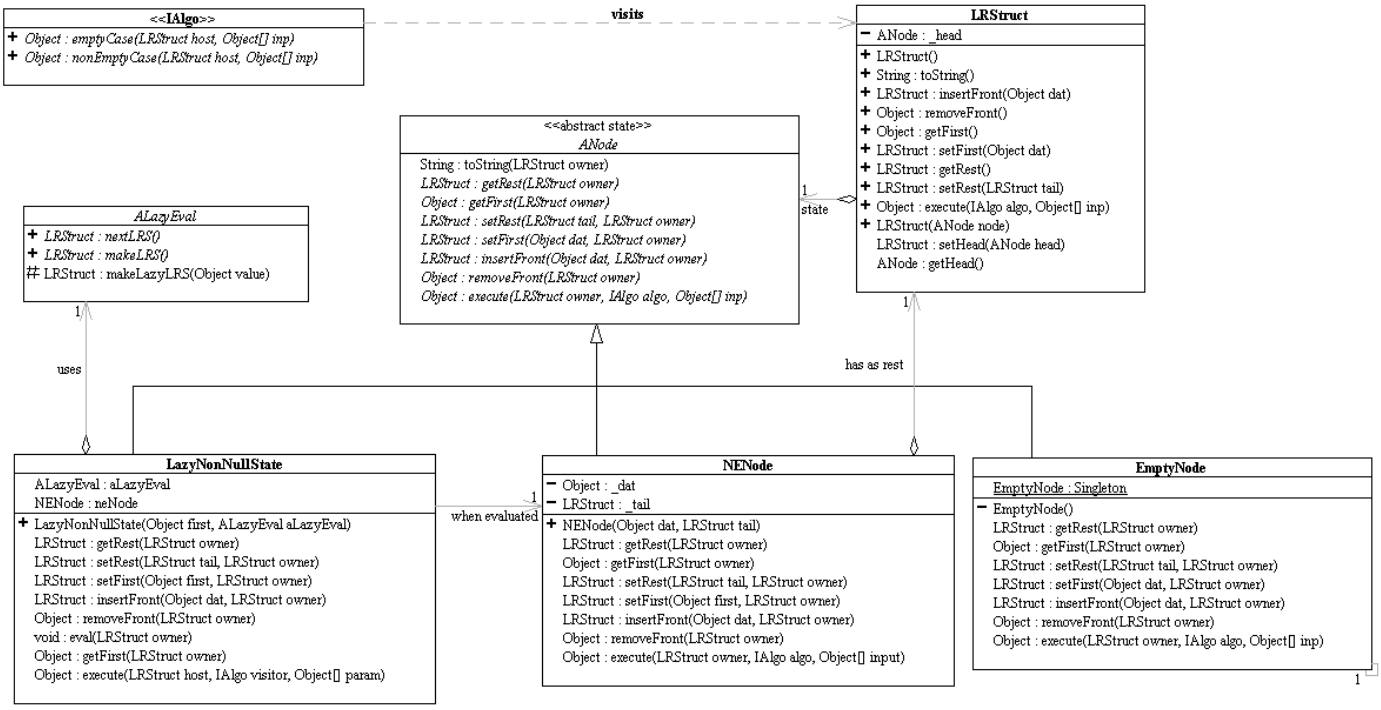
Reference: Immutable List Framework (`IList`)



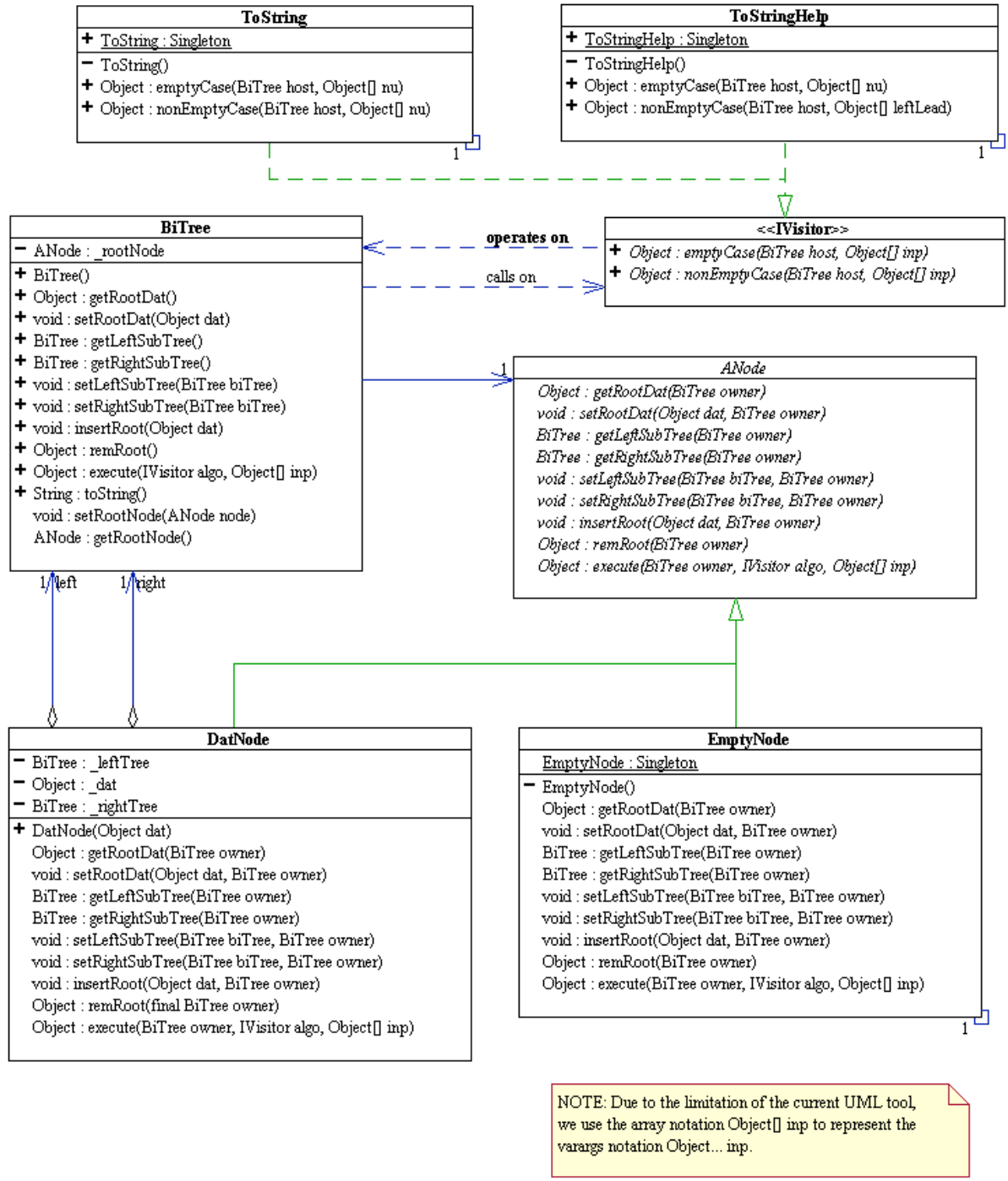
Reference: Mutable List Framework (`LRStruct`)



Reference: Lazy Mutable List Framework (`LRStruct` with `ALazyEval`)



Reference: Binary Tree Framework (`BiTree`)



**Lazy Evaluation**:

a) (40 pts) **You are to write an `ALazyEval` called `LazyCombineEval` that will create a possibly infinite list that contains the result of an `ILambda` that takes in two provided source lists, given as two `LRStructs`.**

- If both input lists are empty, the list created by `LazyCombineEval` is empty.
- If one of the input lists is empty, the list created contains the first element in the non-empty list as first. The rest is created lazily using the rest of the non-empty list.
- If both input lists are non-empty, the list created contains the result of passing the first elements of the input lists to the lambda as first. The rest is created lazily from the rests of the two lists.
- Example:
  - lambda: "remove the firsts from both lists and return the sum"
  - list1 = (), list2 = () → result = ()
  - list1 = (1), list2 = () → result = (1)
  - list1 = (1), list2 = (10) → result = (11)
  - list1 = (1, 2, 3, …), list2 = (10) → result = (11, 2, 3, …)
  - list1 = (1, 2, 3, …), list2 = (10, 10, 10, …) → result = (11, 12, 13, …)
- Example:
  - lambda: "check which first is less; remove the first that is less and return it"
  - list1 = (), list2 = () → result = ()
  - list1 = (1), list2 = () → result = (1)
  - list1 = (1), list2 = (10) → result = (1, 10)
  - list1 = (1, 3, 5, …), list2 = (2) → result = (1, 2, 3, 5, …)
  - list1 = (1, 3, 5, …), list2 = (2, 4, 6, …) → result = (1, 2, 3, 4, 5, 6, …)

Notes:
a. Think delegation always.
b. There are no `if` statements needed, unless you are working with primitive data (ints, doubles, etc.)!
c. Don't forget about the possibility that the source lists may be or become empty.

**Please insert your code for `LazyCombineEval.java` below:**

```
package lrs.lazyLRSEvaluators;

import lrs.*;

/**
 * Lazily creates a possibly infinite list that contains the result of
 * a lambda on two input lists, given as two LRStructs.
 * The lambda determines how the firsts of the two input lists will be
 * combined, and which of the elements will be "consumed", i.e. removed
 * from the front of the input lists.
 * If one of the lists ends, the created list will contain just the elements
 * in the other list. If both lists end, this list will end.
 */
public class LazyCombineEval extends ALazyEval {
    /** Constructs a LazyCombineEval, which creates a list from two input
      * lists using a lambda. The lambda determines how the firsts of the
      * two input lists will be combined, and which of the elements will
      * be "consumed", i.e. removed from the front of the input lists.
      * @param src    first input list
      * @param src2   second input list
      * @param op     a lambda that takes src1 as param[0] and src2 as param[1]
      *               and returns the value for the output list.
      */
    public LazyCombineEval(LRStruct src1, LRStruct src2, ILambda op) {
        this.src1 = src1;
        this.src2 = src2;
        this.op = op;
    }
```

```
    // FOR STUDENT TO COMPLETE: add fields and methods here
```

1. **Lazy Evaluation**:

b) (10 pts) **Write a subclass of the `LazyCombineEval` above called `LazyAddEval` and provide an `ILambda` that creates a list of pair-wise sums of the elements in the two input lists**

- Example:
  - lambda: "remove the firsts from both lists and return the sum"
  - list1 = (), list2 = () → result = ()
  - list1 = (1), list2 = () → result = (1)
  - list1 = (1), list2 = (10) → result = (11)
  - list1 = (1, 2, 3, …), list2 = (10) → result = (11, 2, 3, …)
  - list1 = (1, 2, 3, …), list2 = (10, 10, 10, …) → result = (11, 12, 13, …)

**Please insert your code for `LazyAddEval.java` below:**

```
package lrs.lazyLRSEvaluators;

import fp.*;
import lrs.*;

/**
 * Lazily creates a possibly infinite list that contains the pair-wise sums
 * of the elements in two provided source lists, given as two LRStructs.
 * If one of the lists ends, the created list will contain just the elements
 * in the other list. If both lists end, this list will end.
 */
public class LazyAddEval extends LazyCombineEval {
    // FOR STUDENT TO COMPLETE: add fields and methods here
```

1. **Lazy Evaluation**:

c) (10 pts) **Write a subclass of the `LazyCombineEval` above called `LazyMergeEval` and provide an `ILambda` that creates a sorted list of Integers, given two sorted input lists of Integers. The lambda compares the firsts of both lists, removes the one that is less and returns it for the output list.**

- Example:
  - lambda: "check which first is less; remove the first that is less and return it"
  - list1 = (), list2 = () → result = ()
  - list1 = (1), list2 = () → result = (1)
  - list1 = (1), list2 = (10) → result = (1, 10)
  - list1 = (1, 3, 5, …), list2 = (2) → result = (1, 2, 3, 5, …)
  - list1 = (1, 3, 5, …), list2 = (2, 4, 6, …) → result = (1, 2, 3, 4, 5, 6, …)

**Please insert your code for `LazyMergeEval.java` below:**

```
package lrs.lazyLRSEvaluators;

import fp.*;
import lrs.*;

/**
 * Lazily creates a possibly infinite list that, given two sorted LRStructs
 * of Integers, will contain the elements from both LRStructs sorted. If
 * both LRStructs are non-empty, the first elements will be compared, and the
 * one that is less will be removed and returned as element for the output
 * list.
 */
public class LazyMergeEval extends LazyCombineEval {
    // FOR STUDENT TO COMPLETE: add fields and methods here
```

2. **Binary Trees**:

(40 pts) **You are to write an `IVisitor` called `BRSTrim` that will create an `LRStruct` containing the leaves of the `BiTree` that it visits. The leaves are then removed from the tree.**

- If the tree is empty, the list returned is empty, and the tree remains empty.
- If the tree is a leaf (i.e. its left and right subtrees are both empty), the list returned contains the single element in the root of the tree, and the tree becomes empty.
- If the tree is not a leaf, the list returned contains the list returned from recursing into the left subtree concatenated to the list returned from recursing into the right subtree, and the tree remains non-empty.

- Example:

```
tree=   1
        |_ 2
        | |_ 3                    ← leaf
        | | |_ []
        | | |_ []
        | |_ 4                    ← leaf
        |   |_ []
        |   |_ []
        |_ 5
          |_ 6                    ← leaf
          | |_ []
          | |_ []
          |_ []
```

trim, list returned is (3 4 6)
and the tree is changed to

```
tree =  1
        |_ 2                    ← leaf
        | |_ []
        | |_ []
        |_ 5                    ← leaf
          |_ []
          |_ []
```

trim, list returned is (2 5)
amd the tree is changed to

```
tree =  1                      ← leaf
        |_ []
        |_ []
```

trim, list returned is (1)
and the tree is changed to

tree = []
trim, the list returned is ()
and the tree remains unchanged:

tree = []

Notes:

a. Think delegation always.
b. There are no `if` statements needed!
c. Don't forget about the possibility that the tree may be or become empty.

**Please insert your code for `BRSTrim.java` below:**

```
package brs.visitor;

import brs.*;
import lrs.*;

/**
 * This visitor returns an LRStruct that contains all the leaves of
 * the BiTree. The nodes that are leaves in the BiTree are then removed
 * from the BiTree.
 **/
public class BRSTrim implements IVisitor {
    public static final BRSTrim Singleton = new BRSTrim();
    private BRSTrim() {}

    // STUDENT TO COMPLETE
```