Running Bear (M2T) Transformations

In P1, you harvested data from Java byte codes to produce an instance of a **prog1** database, consisting of tables **bcClass** and **bcMember**. In this assignment, you will use RunningBear (**RB**) to translate a **prog1** database **x.prog1.pl** into a YUML specification, **x.yuml.yuml**, where **x** is specified on the command line.

You may work alone or in groups of two.

Assignment Big Picture

You are to write a program that implements a model-to-text transformation with the following command-line invocation:

```
> java prog1ToYuml.Main X.prog1.pl [outputFile]
```

where `X.prog1.pl` is a **prog1** database that you produced in assignment P1. The default output file is **X.yuml.yuml**, otherwise whatever name is given as the output file is used. The output of **prog1toYuml** is a Yuml specification, which you simply copy into the YUML web page to produce a drawing.

Running Bear and YUML Specs

Little is written on **RB**, simply because it is a trivial Java class of 8 methods. You will find **RB** source file in the **MDELite7/src/MDELite/** directory (a reasonably up-to-date copy is here). Have a look. I will provide you with a shell program that will get you started. As for YUML specifications, read the first part of the YUML manual in **MDELite7/Docs** (a reasonably up-to-date copy is here). It will tell you what you need to know.

As an even bigger hint, the **Yuml.ClassUnparser**, in **MDELite7/src/Yuml/ClassUnparser.java** is an **RB** program that translates a VPL to Yuml Spec (a reasonably up-to-date copy is here). So you have a lot to go on to complete this assignment.

Yuml Tables and Associations

Here is a repeat of the schema of **prog1** from P1:

```
dbase(prog1,[bcClass,bcMember]).

table(bcClass,[cid,"name","superName"]).
table(bcMember,[mid,cid,static,"type","sig"]).
```

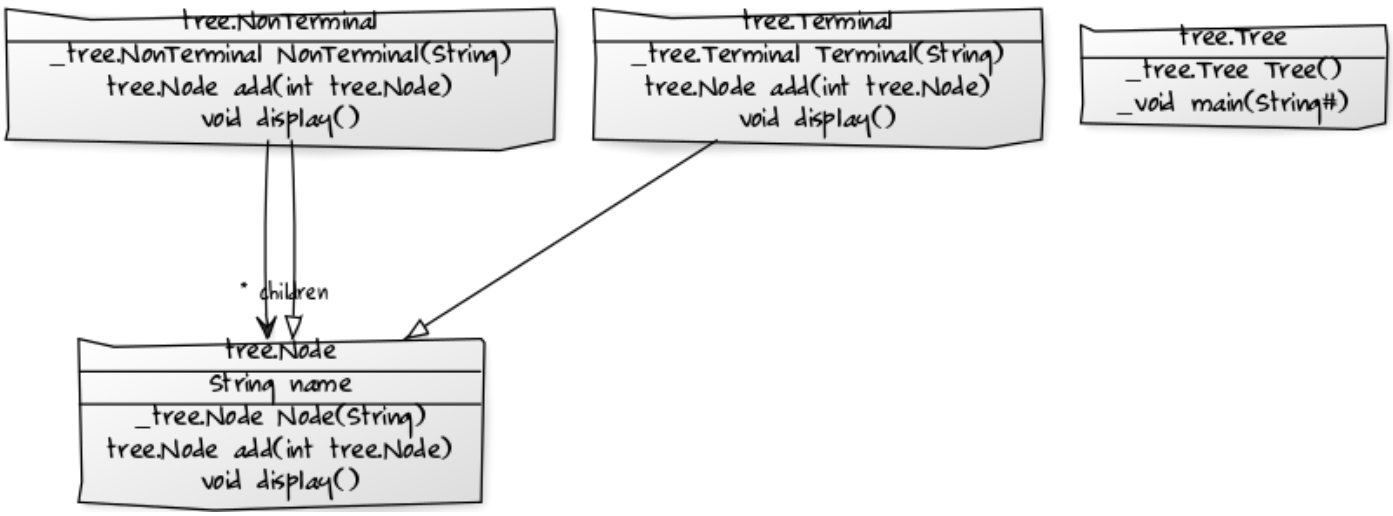
- Database schema **prog1** contains 2 table definitions: **bcClass** and **bcMember**.
- Table **bcClass** has 3 columns or fields or attributes (they all mean the same thing):
  - **cid** -- class identifier
  - **name** -- qualified class name
  - **superName** -- qualified name of superclass
- Table **bcMember** has 5 columns or fields or attributes (they all mean the same thing)
  - **mid** -- member identifier
  - **cid** -- identifier of class that mid is a member
  - **static** -- true or false -- is the member static?
  - **type** -- return type of the member (or simply for fields, type of the member)
  - **sig** -- signature of the member, for a constructor of class File it is "File(String)".

Producing Yuml specs for classes is not difficult. The tuples in **bcClass** are given to you. What is a challenge is to determine associations, which Java doesn't do a particularly good job to distinguish.

> **Hint:** if you are looking at the members of class C, and you find a variable of class D (where both C and D are represented by tuples in **bcClass**), you can infer an association from C→D. Symmetrically, if you are in class D

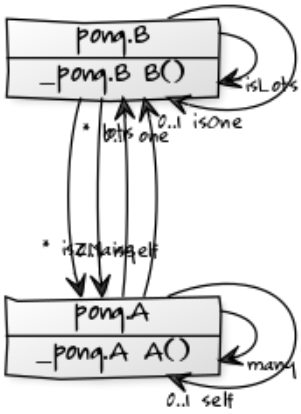
and find a variable of class C, you can infer an association from D→C. The problem is that without deep program analysis (which is debatable whether it is worth doing), you don't know if C→D and D→C are TWO distinct associations or ONE. So, what you do is to play it safe and assume C→D is a different association than D→C. Sad, but such is life.

Hint: As P1 specifically restricted its output to only public fields (static or non-static), you will find few opportunities to draw associations. I've added a new prog1 database to the mix, Tree.prog1.pl, which if you draw it you'll get the following diagram:



Note that there is an Node array in tree.Nonterminal named children. I converted this into "* children", meaning it is a * association. If it were just a variable children of type Node, I would have produced "0..1 children".

Take a look at **pong.prog1.pl** too -- Yuml doesn't draw it particularly well:



Use this Netbeans shell to get you started.

What to Submit to Canvas

A zip file with all source code **(including your Netbeans or Eclipse Project).** The zip file must unzip into **<yourLastName>/<YourFilesAndDirectories>** if you work alone or **<LastName1-LastName2>/<YourFilesAndDirectories> if you work in a group.**

1. **Your program needs to run correctly on Linux machines, even though you may have developed them on Macs and Windoze. The TA will grade your program running on Linux.**
2. The Grader should be able to invoke your program just like **"`java prog1ToYuml.Main x.prog1.pl`"** exactly like the above **from a command line** in the unzipped directory. The Grader will reject any submission where this is not the case.

3. In your PDF document, show the contents of each **X.yuml.yuml** file you produced, along with its YUML representation. The 8 Java packages that you are to draw Yuml diagrams are in this zip file.

4. A PDF file (in the required format) that the Grader should read to provide any information that is not obvious. The contents of the PDF file can be minimal.

You should expect that other packages, known only to the grader, will be used to evaluate your program.

A critical part of any design is clarity and understandability. Hence, you will be graded on the clarity of your project and its ability to work correctly. Sloppy code, documentation, or anything that makes grading or understanding your program difficult will cost you points! Beware, some of these "beauty" points are subjective.

No late assignments/submissions will be accepted.