Department of Computer Science and Engineering
University of California at San Diego

CSE 230
Winter 1998

Problem Set 8

DUE TUESDAY MARCH 3 IN CLASS.

1. The task here is to formalize the description of the semantics of objects and classes as higher-order functions. Imagine an object-oriented extension of ML with single inheritance, initialization of objects at the time they are declared, and dynamic binding of method names to method definitions. Define an abstract syntax for class definitions and object declarations in this language, and corresponding ML concrete datatypes. Implement two ML functions: one that given a class definition, produces a function that can generate objects of that class, and a second that processes object declarations.

   Write a brief description of your implementation strategy, and give an interesting toy example of object-oriented programming using your implementation.

2. Gregory Alan Bolcer of UC Irvine wrote the following about the XML language, according to the TBTF newsletter:

   > An object has state. Imagine an XML document sitting around someplace describing the structure of the state. It's an encapsulation, lightweight, easy to transport around the net, to parse, understand, and change. This it shares with some object models. It's self describing in some way; this it shares with fewer models.
   >
   > Also imagine you have behaviors (methods). These methods aren't physically co-located and in fact are little snippets of code located elsewhere on the network. This distributed method inheritance is shared with even fewer object models. These behavior snippets can be applets written in Java, Python, Tcl, Ada95, Perl, whatever. They include an XML parser (or not) and have the ability to query and change values in the state – the XML document – easily and directly.
   >
   > Also imagine you have communication restrictions on this document, again very lightweight: filtering domains, requiring permissions, etc. Imagine you can restrict the behaviors you want to by digitally signing these behavior snippets. This capability is embodied by most object models in public, private, and protected methods, but now you have a much finer granularity of how you can control the methods than is offered by just these three classical method types.
   >
   > Also imagine these behavior snippets, being all distributed, heterogeneous (that means cross-language and -platform mostly), tightly controlled but more loosely typed depending upon the enforcement, are sometimes competing with one another. They allow query, locking, renaming, versioning, instancing – so that concurrency happens at a finer level of abstraction than in the object encapsulations, allowing a type of re-entrancy and multithreading. Sort of like a threaded,

> persistent object built on top of a lightweight database. Even fewer object models and their implementation languages have persistence and threading built into them.
>
> Imagine having a wide-area event mechanism that allows you to select portions of your state that are relevant to an appropriate task, download it, execute the behavior, and then resynchronize with the original state. The only object model I know that's getting close is Informix Datablades.
>
> Now, here's the kicker: Imagine you have the ability to declare variables dynamically, to declare methods, to version and transition state, to transport, cache, replicate, broadcast it all over the place to make it mobile and ubiquitous, but be able to utilize a naming and routing scheme such that you always know exactly where it is and how to get to it.
>
> On one end of the spectrum you can do enforcement such that it has all the properties of a C++/CORBA/database program; on the other the possibilities are wide open depending on how you want to enforce the data consistency, access, state tracking, loc- ation, whatever.

Which of the language features mentioned above are most important? That is, which will be most critical in future software applications? Does XML really possess these features? Does Java? Your answer to these questions should run two to three pages. Use technical terminology as necessary, but use as little jargon as possible. Most important: think analytically and critically, and make your answer clear and organized.