

# Comp 411 Notes

Scribed by Seth Fogarty

March 4th, 2005

## 1 Objects

We want to see if we can create objects with records and lambda abstraction. So first let us discuss the interface to an object. It's a collection of methods with names. This sounds familiar. Perhaps we can encode this as a record.

So an object has state and some methods that alter state. So they might look like

$$\{s : ref\ t, \quad m_1 : unit \rightarrow unit..\}$$

But some methods can take parameters, in which case they might have the type  $\prod t_i \rightarrow unit$

And they can also return arbitrary types, so  $\prod t_i \rightarrow t$

If the record represents what is visible from the outside, we don't need the type of the state to be visible. Somehow the state will need to be there, but for now we are concerned with the interface. The values in the record will have to have access to the state of this object in some way, but let us restrict the object to simply being a record of methods..

$$\{m_i : \prod t_i \rightarrow t\}$$

## 2 Classes

We should discuss three things: what an object is, what a class is, and what inheritance is. We have discussed what an object should look like, a class is how we create it. A class is some form of object factory: either a template which you can create objects from (Java), an object that creates

other objects through some interface (Smalltalk), or a lambda expression that returns objects. We will use the later-most.

So now we need to consider what a private field is. Mattias wants to know how we can have private fields. Where are these?

If we have an object type

$$\{m_i : \prod t_i \rightarrow t\}$$

Why don't we create a function

```
newrabbit : unit → Rabbit
= λ (). let s1 = ref 17
        let s2 = ref ...
        let m1 = λ().!s1
        let m2 = λ().!s2
        let m3 = λ i.s1 := i
        ...
        in {m1 = m1, m2 = m2, m3 = m3...}
```

So a class is something that allows us to create the objects.

### 3 Inheritance

What is inheritance? We aren't yet concerned with sub-typing, but with what inheritance is: When a inherits from b, a has everything b has. So inheritance produces a class given a class. or a set of classes to a class. So we want to make fancyrabbit has everything rabbit has and more, without duplicating everything all over again..

$$\{m_j : \prod t_i \rightarrow t\}^{j \in J+K}$$

```
newfancyrabbit : unit → Rabbit
= λ (). let r = newrabbit ()
        let s17 = ref 42.
        let let m5 = λ() →!s17
        ...
        in {m1 = m1, m2 = m2, m3 = m3...}
```

Some things to note:

1. Subclasses do not have access to the fields of the superclass.
2. If  $m5 = \lambda().!s17+!r.m1()$ , we have to include the  $r$ .
3. If we wanted to change  $m2$  it's easy enough just to change *with* to  $\{m1 = r.m1, m2 = r.m2, m3 = r.m3, m4 = r.m4, m5 = m5\}$  and then change it to  $\{m1 = r.m1, m2 = m2, m3 = r.m3, m4 = r.m4, m5 = m5\}$  This gives you override.