Object Oriented Programming Systems

Robert Gentleman

BIOCONDUCTOR

FRED HUTCHINSON CANCER RESEARCH CENTER

OOP

- basically it has been observed that writing code, so that it contains components that represent real things, and the actions that you might perform on them, leads to better programs that are more easily understood and that interoperate
- hence OOP - objects, represent things and methods (generic functions) represent the actions that can be performed on these things

OOP

- the R implementation has 2 built in OOP systems and 3 others available as add-ons
- S3 and S4 are builtin
- some Java-style add-ons R.oop, OOP, etc
- some rationalization is needed - the technical part of this talk is about mechanisms that might lead to convergence

Basics

- four things every OOP system should support
  - objects: encapsulate state information and control behavior
  - classes: describe general properties for groups of ob jects
  - inheritance: new classes can be defined in terms existing classes
  - polymorphism: a (generic) function has different behaviors, although similar outputs, depending on the class of one or more of its arguments

Single versus multiple inheritance

- suppose we define two classes, A and B

  ```
  setClass("a", representation(s1="numeric"))
  setClass("b", contains="a",
     representation(s2 = "numeric"))
  ```
- then we say A extends B, or that A is a **superclass** of B, and that B is a subclass of A
- when a class can inherit from at most one subclass the language has single inheritance
- when a class can extend two more more classes at the same time the language has multiple inheritance

Dispatch

- if a generic function specializes the method used based on a single argument the language is said to have single dispatch
- if a generic can specialize based on multiple arguments it is said to have multiple dispatch

Comparison

- Java is single inheritance, single dispatch (that is slowly changing)
- R is multiple inheritance and multiple dispatch
- in a single inheritance, single dispatch language things are easy, you can associate methods with the class, since there is only one

Comparison

- for multiple dispatch and multiple inheritance this no longer makes sense
- code organization tends to separate class definitions from generic/method definitions
- we define our classes in one place and our generic functions somewhere else.
- methods group with generic functions, not with classes

S3 Design

- classes are determined (more or less) by the presence of a class attribute on the object
- there is no formal notion of “slots”, or of inheritance
- the order of the class in the class attribute determines inheritance, most specific first

OOP S3 Classes

- S3 - members of a class are not guaranteed to conform
- so testing of instances is constantly required (and inefficient)
- it is not possible to make use of inheritance in any real way

S3: Generic Functions

- generic functions - polymorphism
  - plot does something useful for many different data types
- methods - implement the actual mechanisms for different classes of objects
- the link between generic and method is very weak - things with the right names are methods
- this make it possible, likely, for unintended interactions

S3: Generic Functions

- the generic has a name, like plot
- the body is a call to UseMethod
- methods have names like plot.lm, for plotting instances of the lm class
- methods are almost regular functions
- they have special variables defined and can use NextMethod,
- not very nice things can happen if methods are called directly

Finding methods

- you can use the function <u>methods</u> to find out about methods

```
methods("plot")
methods(class="glm")
getS3method("plot", "Date")
getAnywhere("plot.Date")
```

- there does not seem to be an easy way to get all generics
- really no hope for classes