



S4 Objects

Thomas Lumley

R Core Development Team

Birmingham, AL — 2006-2-28

Why?

- Class/method registration to allow reflectance, remove ambiguities (is `t.test.formula` a method for `t` or for `t.test`)?
- Multidispatch: eg `plot(x,y)` method should depend on `x` and `y`

Basic structure

- Formal classes: structure specified by `setClass()` objects created with `new()`, fields accessed with `object@slot`.
- Methods belonging to generic functions: `setMethod()` defines the method to be called based on the class of each argument (method **signature**)

Chambers **Programming with Data** is the basic reference, together with R, Omegahat and Bioconductor documentation.

Multidispatch

Consider matrix multiplication with matrices of different types (eg symmetric, diagonal, tridiagonal, Toeplitz,....).

In eg Java

- write a postMultiply method and call `A.postMultiply(B)`, so that the class definition of A has to be changed if a new class for B is designed.
- write both postMultiply and preMultiply methods so that a new class can be hygenically implemented, but then have to ensure that `A.postMultiply(B)` is the same as `B.preMultiply(A)`.

If the methods belong to the generic then just methods that depend symmetrically on classes of both arguments: `multiply(A,B)`

Converting `multiply(A,B)` to `A.postMultiply(B)` or `B.preMultiply(A)` is mechanical, but allows some compiler optimizations, especially when classes known at compile-time.

Simple example

```
setClass("fungi", representation(x="numeric", y="numeric",  
    species = "character"))
```

establishes a class with three slots named x, y and species

Alternative: define classes recursively

```
setClass("xyloc", representation(x="numeric", y="numeric"))  
setClass("fungi", representation("xyloc",  
    species = "character"))
```

Add show method (equivalent of print())

```
setMethod("show", "fungi",  
    function(object) cat(object@species,  
        " at (", object@x, ", ", object@y, ")\n")  
)
```

Simple example

Make an object

```
b<-new("fungi", x=5, y=10, species="Amanita muscaria")
```

See the class description

```
> getClass("fungi")
```

Slots:

Name:	species	x	y
Class:	character	numeric	numeric

Extends: "xyloc"

ROC example

Class definition

```
setClass("ROC", representation(sens="numeric", mspec="numeric",  
                               test="numeric", call="call"),  
        validity=function(roc) length(roc@sens)==length(roc@mspec)  
                               && length(roc@sens)==length(roc@test)  
        )
```

ROC example

```
ROC<-function(T,D){
  TT<-rev(sort(unique(T)))
  DD<-table(-T,D)

  sens<-cumsum(DD[,2])/sum(DD[,2])
  mspec<-cumsum(DD[,1])/sum(DD[,1])

  new("ROC", sens=sens, mspec=mspec, test=TT,call=sys.call())
}
```

ROC example

```
setMethod("show", "ROC",
function(object){
  cat("ROC curve: ")
  print(object@call)
})
setMethod("plot", c("ROC", "missing"), function(x, y, type="b",
  null.line=TRUE, xlab="1-Specificity",
  ylab="Sensitivity", main=NULL, ...){
  par(pty="s")
  plot(x@mspec, x@sens, type=type, xlab=xlab, ylab=ylab, ...)
  if(null.line)
    abline(0, 1, lty=3)
  if(is.null(main))
    main<-x@call
  title(main=main)
})
```

ROC example

```
setGeneric("lines", signature=c("x","y"))
```

```
setMethod("lines",c("ROC","missing"), function(x,y,...){  
  lines(x@mspec, x@sens,...)  
})
```

```
setGeneric("ROC")
```

```
setMethod("ROC",c("glm","missing"),  
function(T){  
  if (!(T$family$family %in%  
    c("binomial", "quasibinomial")))  
    stop("ROC curves for binomial glms only")
```

```
  test<-fitted(T)
```

```
  disease<-(test+resid(T,type="response"))
```

ROC example

```
disease<-disease*weights(T)
if (max(abs(disease %% 1))>0.01)
  warning("Y values suspiciously far
          from integers")

TT<-rev(sort(unique(test)))
DD<-table(-test,disease)

sens<-cumsum(DD[,2])/sum(DD[,2])
mspec<-cumsum(DD[,1])/sum(DD[,1])

new("ROC",sens=sens, mspec=mspec,
    test=TT,call=sys.call())
}
)
```