CMU Summer Statistical Computing 2013
Classes in R, Part 2

1) Review

a. Every object in R belongs to a class
   i. Classes are a key to Object Oriented Programming.
   ii. Encapsulation: one object has many components
   iii. Polymorphism: running a function (plot, print, summary, etc.) does different things for arguments of different types; these are “generic” functions which work by “dispatching” their arguments to an appropriate “method” function (e.g., plot.density, summary.lm).
   iv. Inheritance: a class may be derived from (extend) another class

b. Old (S3) style classes in R
   i. class(someObject)
   ii. methods(class="someClass")
   iii. names(someObject)
   iv. someMethodFunction(someClass)
   v. someObject$someNamedComponent
   vi. str(someObject)

c. Details on using S3 classes, methods, and generic functions
   i. To make a new S3 class object, you can just add a class attribute:
   junk = list(a=4, b="stupid")
class(junk)="lme"  # or attr(junk,"class")="lme"
class(junk)
  # [1] "lme"
print(junk)  # or nlme:::print.lme(junk)
  # Linear mixed-effects model fit by   Data: NULL
  #   Log-likelihood: NULL
  #   Fixed: list()
  # NULL
  # Length Class Mode
  # 0 NULL NULL
  # Number of Observations:  Error in 1:dd$Q : argument of length 0
More on S3 Classes

a. One “normal” way to make an S3 class object is with structure():
   junk = structure(list(a=4, b="stupid"), class="lme")

b. Note that S3 classes are usually lists, and you may add elements to a class without breaking them:
   
   rslt$runTime = Sys.time()
   names(rslt)
   [1] "modelStruct" "dims" "contrasts" "coefficients" "varFix"
   [6] "sigma" "apVar" "logLik" "numIter" "groups"
   [11] "call" "terms" "method" "fitted" "residuals"
   [16] "fixDF" "na.action" "data" "runTime" "runTime"
   rslt$runTime
   rslt

   # Linear mixed-effects model fit by REML
   # Data: dat
   # Log-restricted-likelihood: -56.46109
   # Fixed: y ~ x
   # ...
   # Number of Observations: 40
   # Number of Groups: 10

c. The main special feature of classes is that many functions (“methods”) are **generic**, and may be written in this special way:

   foo = function (x, ...)
   UseMethod("foo")

   and this allows them to **automatically check the class of the object that they are passed and “dispatch” the object to the most appropriate version of the function**. E.g.,

   foo(bar), when bar is of class “zip” runs foo.zip(bar).

   Generic functions include: print, summary, plot, coef, predict, fitted, update, residuals, var.test, t, terms, quantile, lines, diff, density, confint, aggregate, anova, deriv, median, model.frame, model.matrix, qqnorm, barplot, boxplot, contour, hist, image, pairs, points, text, all.equal, duplicated, as.array, as.character, as.data.frame, as.Date, as.list, as.matrix, cut, format, labels, mean, range, rev, scale, sort, split, subset, unique, with

d. Some information can (usually) be obtained with getClass(), e.g.,

   getClass("lm")
   # Virtual Class "lm" [package "methods"]
   #
   # Slots:
   # Name: .S3Class
   #
   # Class: character
   #
# Extends: "oldClass"
#
# Known Subclasses:
# Class "mlm", directly
# Class "aov", directly
# Class "glm", directly
# Class "maov", by class "mlm", distance 2
# Class "glm.null", by class "glm", distance 2

e. Simple extensions of the language with S3 classes:
pl=structure(data.frame(n=seq(10,50,10),power=runif(5,0,1)),
             class= "power")

plot.power = function(x,y,...) {
  plot(x$n, x$power*100, xlab= "n", ylab="power",
       ylim=c(0,100), type="b", ...)
  invisible(x)
}

class(pl)
plot(pl)
summary(pl)
  #       Length Class  Mode
  # n      5 -none- numeric
  # power  5 -none- numeric

f. New method functions

grok = function(x,...) UseMethod("grok") # The generic function
grok.default = function(x,...) {
  stop("don’t know how to grok() objects of class ", class(x))
}
grok(pl)  # [1] "don’t know how to grok() objects of class power"
grok.power = function(x,...) {
  paste("n’s from", min(x$n), "to", max(x$n), "have power from",
        round(100*min(x$power),1), "to", round(100*max(x$power),1))
  grok(pl)  # "n’s from 10 to 50 have power from 15.9 to 72.5"
3) New style (“S4”) classes offer more control and safety

a. Example from the “lme4” package

```r
library(lme4)
search()
# [1] ".GlobalEnv" "package:lme4" "package:Matrix" "package:lattice" ...

dat = data.frame(id=rep(1:10,each=4),x=rnorm(40),y=rnorm(40))
rslt4 = lmer(y~x|id, dat)
names(rslt4) # NULL
typeof(rslt4) # [1] "S4"

class(rslt4)
# [1] "mer"
# attr("package")
# [1] "lme4"

getslots("mer") # or getslots(class(rslt4))
#          env       nlmodel         frame          call         flist
#  "environment"        "call"  "data.frame"        "call"  "data.frame"
#           X           Xst            Zt           pWt        offset
#    "matrix"   "dgCMatrix"   "dgCMatrix" "numeric" "numeric"
#           y           Gp            dims         ST          V
#    "numeric" "numeric"     "integer"        "list"     "numeric"
#           A           Cm           Cx           L          deviance
# "numeric" "numeric"     "integer"        "numeric"        "numeric"
#           fixef         ranef           u           eta          mu
# "numeric" "numeric"     "numeric"        "numeric"        "numeric"
#           var          resid         sqrtXWt       sqrtrWt
# "numeric" "numeric"     "numeric"        "numeric"        "numeric"
#           RZX         RX           ghx          ghw
#    "matrix"   "matrix"     "numeric"        "numeric"

slotNames(rslt4) # or slotNames("mer")
# [1] "env"      "nlmodel"  "frame"    "call"     "flist"    "X"
# [7] "Xst"      "Zt"       "pWt"      "offset"   "y"        "Gp"
# [13] "dims"    "ST"       "y"        "A"        "Cm"       "Cx"
# [19] "L"       "deviance" "fixef"    "ranef"    "u"        "eta"
# [25] "mu"      "muEta"    "var"      "resid"    "sqrtXWt" "sqrtrWt"
# [31] "RZX"     "RX"       "ghx"      "ghw"

rslt4@RX  # 6.32455
slot(rslt4,"RX")  # 6.32455
rslt4$RX  # $ operator not defined for this S4 class

sapply(slotNames(rslt4),function(x)length(slot(rslt4,x)))
#          env       nlmodel         frame          call         flist
#       2        3        3        1        40        0       800
#           pWt      offset          y       Gp            dims         ST          V
#       0        0        1        40        2        18        1       0       800
#           Cm         Cx        L      deviance      fixef      ranef      u      eta
#       1        0        0        1        13        1        20       20        40
#           mu      muEta      var      resid      sqrtXWt      sqrtrWt      RZX         RX
#      40       0       0        40       0       0       0        0        20        1
#      ghx      ghw
#       0        0
```
getClass("mer") # or getClass(class(rslt4))
# Class "mer" [package "lme4"]
#
# Slots:
# Name: env nlmodel frame call flist
# Class: environment call data.frame call data.frame
# Name: X Xst Zt pWt offset
# Class: matrix dgCMatrix dgCMatrix numeric numeric
# Name: y Gp dims ST V
# Class: numeric integer integer list matrix
# Name: A Cm Cx L deviance
# Class: dgCMatrix dgCMatrix numeric CHMfactor numeric
# Name: fixef ranef u eta mu
# Class: numeric numeric numeric numeric numeric
# Name: muEta var resid sqrtXWt sqrtXWt numeric
# Name: RX ghx ghw
# Class: matrix matrix numeric numeric
#
# Known Subclasses:
# Class "summary.mer", directly
# Class "merExt", directly
# Class "lmerStratVar", by class "merExt", distance 2

b. Utility functions
  i. getGenerics()@.Data  # All method functions
      # [1] "-" "%!" "!=" "$\leq$"
      # [5] "%%" "%+%" "%/" "%\leq"
      # [9] "*" "/" ";" ";[<-
      #[13] "<-" ";\times" ";|" ";[<-"
      #[17] "<" "<=" ";=:" ";>
      #[21] ";>=" "abs" "acos" "acosh"
      #[25] "addNextMethod" "AIC" "all" "all.equal" ...

getClasses(where="package:lme4")
# [1] "lmList" "mer" "merMCMC" "summary.mer"

showMethods("coef")  # classes for which a method function exists
# Function: coef (package stats)
# object="ANY"
# object="lmList"
# object="mer"
# object="mle"
# object="summary.mer"
# object="summary.mle"

showMethods(classes="mer")  # method functions for a class
# Function: anova (package stats)
# object="mer"
# Function: coef (package stats)
# object="mer" ... (Very long!!)
ii. Creating your own S4 classes

```r
setClass("demog",
    representation=representation(
        fname="character", lname="character",
        QPA="numeric", birthDate="Date")
)

getSlots("demog")

# A function to create objects of the class
# 'birthDate must be "mm/dd/yyyy" or a Date object

demog <- function(fname, lname, QPA, birthDate) {
    if (!is.character(fname) || length(fname)>1 || nchar(fname)<1)
        stop("Invalid 'fname'")
    if (!is.character(lname) || length(lname)>1 || nchar(lname)<1)
        stop("Invalid 'lname'")
    if (is.na(QPA)) {
        QPA = NA_real_
    } else if (!is.finite(QPA) || QPA<0 || QPA>4) {
        stop("'QPA' must be NA or a valid number between 0.0 and 4.0")
    }

today = Sys.Date()
if (is.character(birthDate)) {
    birthDate = as.Date(birthDate, format="%m/%d/%Y")
} else if (!is.real(birthDate, "Date")) {
    stop("'birthDate' format is mm/dd/yyyy")
}
age = as.numeric(today - birthDate) / 365.25
if (age<=0 || age>121) stop("Invalid age (0-121 allowed)")

return(new("demog", fname=fname, lname=lname, QPA=QPA, birthDate=birthDate))
}

dl = demog(fname="Howard", lname="Seltman", QPA=4.0, birthDate="11/23/1954")
dl
```

An object of class "demog"

Slot "fname":
[1] "Howard"

Slot "lname":
[1] "Seltman"

Slot "QPA":
[1] NA

Slot "birthDate":
[1] "1954-11-23"
# Query needed before making a method for a class:
getMethod(print)
# Method Definition (Class "derivedDefaultMethod"):
# function (x, ...)
# UseMethod("print")
# <environment: namespace:base>
# Signatures:
#     x
# target "ANY"
# defined "ANY"

# Making a print method for class powerSim:
setMethod("print", "demog",
  definition = function(x, ...) {
    cat("Subject", x@fname, x@lname, "\n")
    if (is.na(x@QPA)) {
      cat("QPA is missing\n")
    } else {
      cat("QPA =", x@QPA, "\n")
    }
    age = as.numeric(Sys.Date() - x@birthDate) / 365.25
    years = floor(age)
    months = floor((age - years)*12)
    cat("Born", format(x@birthDate, "%m/%d/%Y"), ", (age =",
      years, "years and", months, "months)\n")
    invisible(x)
  })
)

print(d1)
# Subject Howard Seltman
# QPA = 4
# [1] 58.62286
# [1] 58
# [1] 0.6228611
# Born 11/23/1954 (age = 58 years and 7 months)

# S4 objects run show() when named directly at the prompt
setMethod("show", "demog",
  definition = function(object) print(object))
d1
# Subject Howard Seltman
# ...

In addition we can use setValidity(class, method) to incorporate a mechanism to assure that only “valid” objects of the class are created, e.g., enforce positive definite on a covariance component.