CMU MSP 36601 Classes In R HJS, Jan. 16, 2019

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, print.data.frame, summary.lm).
   iv. Inheritance: a class may be derived from (extend) another class

b. Old-style (S3) classes in R
   i. `class(someObject)`
   ii. `methods(someGenericFunction)`
   iii. `methods(class="someClass")`
   iv. `names(someObject)`
   v. `someMethodFunction(someClass)`
   vi. `someObject$someNamedComponent`
   vii. `str(someObject)`
   viii. `getAnywhere(someMethodWithAStar)`
   ix. `argsAnywhere(someMethodWithAStar)`

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
library(lme4)
print(junk)  #  or nlme:::print.lme(junk)
# Linear mixed-effects model fit by
# Error in if (x$method == "REML") "REML\n" else "maximum
#   likelihood\n" :
# argument is of length zero
```

2) 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")
```

(Note: this is a general-purpose function for adding any attributes to an object.)

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
# [1] "2011-06-17 13:44:57 EDT"
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:

```
p1=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(p1)
plot(p1)
summary(p1)  # runs summary.default(p1)
#       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(p1)  # [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(p1) # "n's from 10 to 50 have power from 15.9 to 72.5"
```

3) New style ("S4") classes offer more control and safety, but are more complex

a. Example from the "lme4" package

```
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] "lmerMod"
# attr(,"package")
# [1] "lme4"

getSlots("lmerMod")  # or getSlots(class(rslt4))
#         resp           Gp         call        frame        flist
#   "lmerResp"    "integer"       "call" "data.frame"       "list"
#         cnms        lower        theta         beta            u
#       "list"    "numeric"    "numeric"    "numeric"    "numeric"
#      devcomp           pp      optinfo
#       "list"   "merPredD"       "list"

slotNames(rslt4)  # or slotNames("lmerMod")
#  [1] "resp"    "Gp"      "call"    "frame"   "flist"   "cnms"
#  [7] "lower"   "theta"   "beta"    "u"       "devcomp" "pp"
# [13] "optinfo"

rslt4@beta  # [1] -0.07009482
slot(rslt4,"beta")  # [1] -0.07009482
rslt4$beta  # $ operator not defined for this S4 class

sapply(slotNames(rslt4),function(x) length(slot(rslt4,x)))
#    resp      Gp    call   frame   flist    cnms   lower   theta    beta
#       1       2       3       3       1       1       3       3       1
#       u devcomp      pp optinfo
#      20       2       1       7

getClass("lmerMod") # or getClass(class(rslt4))
# Class "lmerMod" [package "lme4"]
#
# Slots:
#
# Name:        resp         Gp       call      frame      flist      cnms
# Class:   lmerResp    integer       call data.frame       list      list
#
# Name:       lower      theta       beta          u    devcomp        pp
# Class:    numeric    numeric    numeric    numeric       list  merPredD
#
# Name:     optinfo
# Class:       list
#
# Extends: "merMod"
```

b. Utility functions

i.
```
getGenerics()@.Data  # All method functions
#   [1] "-"         "!"         "!="     "$"
#   [5] "$<-"       "%%"        "%&%"    "%*%"
#   [9] "%/%"       "&"         "*"      ".DollarNames"
#  [13] "/"         "["         "[[<-"   "[<-"
#  [17] "^"         "|"         "+"      "<"
#  [21] "<="        "=="        ">"      ">="
#  [25] "abs"       "acos"      "acosh"  "addNextMethod" ...

getClasses(where="package:lme4")
# [1] "lmList4"  "lmerMod"  "glmerMod" "merMod"   "nlmerMod"

# classes for which a method function exists
showMethods("getL")
# Function: getL (package lme4)
# x="merMod"

showMethods("coef")
# Function "coef":
#  <not an S4 generic function>

showMethods(classes="lmerMod") # method functions for a class
# Function: $ (package base)
#
# Function ".DollarNames":
#  <not an S4 generic function>
#
# ...
#
# Function: initialize (package methods)
# .Object="lmerMod"
#     (inherited from: .Object="ANY")
# ...
#
# Function: summary (package base)
# object="lmerMod"
#     (inherited from: object="ANY")
```

But this does **not** give a complete list:

```
coef(rslt4)
# $id
#                 x (Intercept)
# 1   4.975400e-17 -0.07009482
# 2   5.528522e-18 -0.07009482
# 3  -4.025704e-17 -0.07009482
# ...
# 8  -7.260172e-17 -0.07009482
# 9  -1.774561e-17 -0.07009482
# 10  4.381128e-17 -0.07009482
#
# attr(,"class")
# [1] "coef.mer"
```

Note: `methods(coef)` does include `coef.merMod*`

ii. Creating your own S4 classes

```
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")
    if (is.na(birthDate)) stop("'birthDate' format is mm/dd/yyyy")
  } else if (!is(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))
}

d1 = demog(fname="Barack", lname="Obama", QPA=4.0,
           birthDate="8/4/1961")
d1

An object of class "demog"
Slot "fname":
[1] "Barack"

Slot "lname":
[1] "Obama"

Slot "QPA":
[1] 4

Slot "birthDate":
[1] "1961-08-04"
```

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 demog:

```
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 Obama Barak
QPA = 4
Born 08/04/1961 (age = 57 years and 4 months)
```

S4 objects run show() when named directly at the prompt

```
setMethod("show", "demog",
          definition = function(object) print(object))
d1
# Subject Barack Obama
# ...
```

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.