Object Orientation in R

Biostat 140.776

2004-10-04

Roger D. Peng

Object oriented?

- A system for abstracting data in programs
- Different languages have (very) different ways of implementing this system
- Languages that support object oriented programming
  - Java, C++, Python, Lisp, Perl
- R implements two systems of object orientation – "S3" and "S4" *classes* and *methods*

Object orientation in R

- R is unique because it is interactive and has a system for object orientation
- S3
  - included with version 3 of the S language
  - informal, a little kludgey
  - sometimes referred to as "old-style" classes/methods
- S4
  - included with S-PLUS 6.0 and R 1.4.0 (version 4 of S)
  - more formal and rigorously enforced
  - sometimes called "new-style" classes/methods
- S3 and S4 are separate systems operating independently of each other

Classes and objects

- A *class* is a description of a *thing*
  - ex. linear model, data frame, sparse matrix, microarray, point process dataset
- An *object* is an instance of a class

```
x <- matrix(1, nrow = 4, ncol = 4)
y <- matrix(2, nrow = 5, ncol = 2)
```

x and y are both objects of class "matrix"

Generics and methods

- A *generic function* provides an interface to a particular computation
  - The computation is "generic" in that it could have different "meanings" for different classes
  - ex. print, summary, confidence intervals, prediction, mean, subsetting ("[")
- A *method* implements a computation for a particular class

The basic idea

- If `foo()` is a generic function, then calling `foo()` on object `x` may produce a completely different kind of output from calling `foo()` on object `y` if `x` and `y` are of different classes

Identifying generic functions

```
> mean
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
>
> print
function (x, ...)
UseMethod("print")
<environment: namespace:base>

> summary
function (object, ...)
UseMethod("summary")
<environment: namespace:base>
>
```

Functions to know

- `methods()` – shows the available methods for a given generic or for a given class
- `getS3method()` – methods can be hidden in namespaces (a different topic altogether)
  - e.g. `getS3method("logLik", "lm")`
- `getAnywhere()` – searches everywhere for a particular function

Methods: Example

```
> methods(summary)
 [1] summary.aov            summary.aovlist        summary.connection
 [4] summary.data.frame     summary.Date           summary.default
 [7] summary.ecdf*          summary.factor         summary.glm
[10] summary.infl           summary.lm             summary.loess*
[13] summary.manova         summary.matrix         summary.mlm
[16] summary.nls*           summary.packageStatus* summary.POSIXct
[19] summary.POSIXlt        summary.ppr*           summary.prcomp*
[22] summary.princomp*      summary.stepfun        summary.stl*
[25] summary.table          summary.tukeysmooth*

   Non-visible functions are asterisked
>
```

Example: “lm” methods

```
> methods(class = "lm")
 [1] add1.lm*           alias.lm*          anova.lm           case.names.lm*
 [5] confint.lm*        cooks.distance.lm* deviance.lm*       dfbeta.lm*
 [9] dfbetas.lm*        drop1.lm*          dummy.coef.lm*     effects.lm*
[13] extractAIC.lm*     family.lm*         formula.lm*        hatvalues.lm
[17] influence.lm*      kappa.lm           labels.lm          logLik.lm*
[21] model.frame.lm     model.matrix.lm    plot.lm            predict.lm
[25] print.lm           proj.lm*           residuals.lm       rstandard.lm
[29] rstudent.lm        summary.lm         variable.names.lm* vcov.lm*

   Non-visible functions are asterisked
>
```

```
> x <- 1:4
> y <- matrix(1:4, nrow = 2, ncol = 2)
> class(x)
[1] "integer"
> class(y)
[1] "matrix"
> print(x)
[1] 1 2 3 4
> print(y)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
>
```

```
> x <- rnorm(100)
> y <- x + rnorm(100)
> fit <- lm(y ~ x)
> class(fit)
[1] "lm"
> print(fit)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.1227       1.1620

>
```

```
> summary(fit)

Call:
lm(formula = y ~ x)

Residuals:
      Min        1Q    Median        3Q       Max
-2.750957 -0.592462 -0.007884  0.717936  2.783677

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.1227     0.1085   1.131    0.261
x             1.1620     0.1126  10.321   <2e-16 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 1.079 on 98 degrees of freedom
Multiple R-Squared: 0.5209,     Adjusted R-squared: 0.516
F-statistic: 106.5 on 1 and 98 DF,  p-value: < 2.2e-16

>
```

Method searching

- When a generic *foo* is called on an object of class *bar*, the function R searches for the function `foo.bar()`
- If an appropriate method is not found, then R searches for the function `foo.default()`
- If a default method is not found, then an error is thrown

A note about auto-printing

- Not all is what it seems!
- When R completes a computation in a function, the function returns an object
- If the object is not assigned to another variable, then it is *auto-printed*
  - the “print” method for that object is called and executed

```
> x <- seq(1, 10)
> seq(1, 10)
 [1]  1  2  3  4  5  6  7  8  9
10
> class(x)
[1] "integer"
> methods(class = "integer")
[1] as.data.frame.integer
>
```

```
> summary(fit)

Call:
lm(formula = y ~ x)

Residuals:
      Min        1Q    Median        3Q       Max
-2.750957 -0.592462 -0.007884  0.717936  2.783677

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.1227     0.1085   1.131    0.261
x             1.1620     0.1126  10.321   <2e-16 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 1.079 on 98 degrees of freedom
Multiple R-Squared: 0.5209,     Adjusted R-squared: 0.516
F-statistic: 106.5 on 1 and 98 DF,  p-value: < 2.2e-16

>
```

A note about `summary()`

- Summary methods do not actually print anything!
- The summary method for class *bar* returns an object of class *summary.bar*
- This object is then printed using the `print.summary.bar()` method
- Summary methods sometimes compute things that are not stored in the original object
- One can store the “summary” object and extract relevant quantities (e.g. p-values, std. errors)

Extracting elements from objects

- Classes are usually implemented using *lists*
- Elements of a list can be extracted using the $ or the `[[` operators
- Names of elements in a list can be found using the `names()` function

```
> names(fit)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
> s <- summary(fit)
> names(s)
 [1] "call"          "terms"         "residuals"     "coefficients"
 [5] "aliased"       "sigma"         "df"            "r.squared"
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
> fit$coefficients
(Intercept)           x
  0.1227497   1.1620332
>
> s$coefficients
             Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 0.1227497  0.1084929  1.131408 2.606444e-01
x           1.1620332  0.1125841 10.321465 2.430299e-17
```

Defining your own classes and methods

The *class* of an object can be assigned using the `class()` function

```
> x <- 1:10
> print(x)
 [1]  1  2  3  4  5  6  7  8  9 10
> class(x) <- "roger"
> print.roger <- function(x, ...) cat("Yo!", x, "\n")
> print(x)
Yo! 1 2 3 4 5 6 7 8 9 10
>
```

S4 classes and methods

- The S4 implementation of classes and methods can be found in the “method” package (e.g. via `library(methods)`)
- The authoritative reference is *Programming with Data* by J. Chambers (1998), a.k.a. the “green book”
- S4 classes/methods are used extensively in Bioconductor

Bonus question

- Setup the R environment so that typing the letter `q` (and then `Enter`) quits the R session immediately (no questions asked)