A brief Introduction to object-oriented programming in R

Object-oriented programming (OOP)

- OOP is a programming paradigm that uses "objects" and their interactions to design applications and computer programs.
- OOP-related programming techniques may include features such as information hiding, data abstraction, encapsulation, modularity, **polymorphism**, and **inheritance**.
- R has a system for OOP, based on **generic functions**.

Some basic concepts

3

- A **class** is a description of a thing (i.e., how objects of a certain type look like).
- An **object** is an instance of a class.
- A **generic function** is a function which dispatches methods. A generic function typically encapsulates a “generic" concept, such as plot, mean, logLik, residuals, predict, summary, and so on and so forth. The generic function does not actually do any computation.
- A **method** is the implementation of a generic function for an object of a particular class.

S3 vs. S4 systems

4

- The S language (hence R) has two object systems, known informally as S3 and S4, and each system can be used fairly independently of the other.
- S3 objects, classes and methods have been available in R from the beginning (Chambers and Hastie, 1992). They are informal and very interactive.
- S4 objects, classes and methods are much more formal and rigorous, but less interactive (Chambers, 1998), which is available through the methods package, attached by default since version 1.7.0.

S3 vs. S4 generic functions

5

```
> summary    # - S3 generic function
function (x, ...)
UseMethod("summary")
<environment: namespace:base>

> show    # - S4 generic function
standardGeneric for "show" defined from package
   "methods"
function (object)
standardGeneric("show")
<environment: 0x8d7cdc8>
Methods may be defined for arguments: object
```

The *summary* function

6

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

Or, it can be written as:

```
> Summary <- function (object, ...) UseMethod("summary")
```

How does it work?

7

```
> x <- as.factor(rep(c("a","b"),c(7,13)))
> class(x)
[1] "factor"
> summary(x)
 a  b
 7 13

y <- rnorm(20)
> class(y)
[1] "numeric"
> summary(y)
    Min.      1st Qu.     Median     Mean     3rd Qu.     Max.
-1.78300 -0.55330  0.06459  0.01969  0.66960  1.72900
```

How does it work?

8

```
> lm.demo <- lm(y~x)
> class(lm.demo)
[1] "lm"
> summary(lm.demo)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7500 -0.6665 -0.1008  0.5537  1.5156

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.3769     0.3609  -1.044    0.310
xb            0.6102     0.4477   1.363    0.190

Residual standard error: 0.9549 on 18 degrees
    of freedom
Multiple R-squared: 0.09355,    Adjusted R-
    squared: 0.04319
F-statistic: 1.858 on 1 and 18 DF,  p-value:
    0.1897
```

Method dispatch

- > methods(summary)

```
 [1] summary.agnes*         summary.aov            summary.aovlist
 [4] summary.areg.boot      summary.clara*         summary.connection
 [7] summary.data.frame     summary.Date           summary.default
[10] summary.diana*         summary.dissimilarity* summary.ecdf*
[13] summary.factor         summary.fanny*         summary.find.matches
[16] summary.formula        summary.glm            summary.impute
[19] summary.infl           summary.ldBands        summary.lm
[22] summary.loess*         summary.manova         summary.matrix
[25] summary.mChoice        summary.mlm            summary.mona*
[28] summary.nls*           summary.packageStatus* summary.pam*
[31] summary.POSIXct        summary.POSIXlt        summary.ppr*
[34] summary.prcomp*        summary.princomp*      summary.shingle*
[37] summary.silhouette*    summary.stepfun        summary.stl*
[40] summary.table          summary.transcan       summary.trellis*
[43] summary.tukeysmooth*
```

Method: *summary.factor*


```
> summary.factor
function (object, maxsum = 100, ...)
{
    nas <- is.na(object)
    ll <- levels(object)
    if (any(nas))
……
}
    if (any(nas))
        c(tt, `NA's` = sum(nas))
    else tt
}
<environment: namespace:base>
```

Method: summary.default

> summary.default

```
function (object, ..., digits = max(3, getOption("digits") -
    3))
{
    if (is.factor(object))
        return(summary.factor(object, ...))
    else if (is.matrix(object))
……
}
    else c(Length = length(object), Class = class(object), Mode = mode(object))
    class(value) <- "table"
    value
}
<environment: namespace:base>
```

Method: summary.loess

> getAnywhere(summary.loess)

```
A single object matching ‘summary.loess’ was found
It was found in the following places
  registered S3 method for summary from namespace stats
  namespace:stats
with value

function (object, ...)
{
    class(object) <- "summary.loess"
    object
}
<environment: namespace:stats>
```

A golden rule

13

- DO NOT call methods directly in OOP. Instead, use a GENERIC function, which dispatches methods to objects according to their classes.

Class resolved at object creation

14

- x <- as.factor(rep(c("a","b"),c(7,13)))
- class(x)
- [1] "factor"
- > x.new<-x
- > class(x.new)
- [1] "factor“
- > print(x.new) # same as typing *x.new*
- [1] b b a a b a b a b a b a b b b b a b b b
- Levels: a b

Change the class of an object in S3


```
> # change the class of x.new to “myvector”
> class(x.new)<-“myvector“   # - or, attr(x.new,”class”)<-”myvector”)
> class(x.new)
[1] “myvector“

> print.myvector<-function(x,...) {
+ cat("This is a new vector, and its content is confidential!\n")
+ }
> # try to print the content of newx
> print(x.new)  # same as typing x.new
This is a new vector,  and its content is confidential!
```

But you could do something wrong!


```
> class(x.new) <- "lm"
> class(x.new)
[1] "lm"
> x.new
Error in x$call : $ operator is invalid for atomic vectors
```

Create a new class in S4

17

```
> setClass("rectangle", representation(length =
   "numeric", width = "numeric"))
[1] "rectangle“

> rect <- new("rectangle",length=10,width=5)


> rect
An object of class “rectangle”
Slot "length":
[1] 10
Slot "width":
[1] 5
```

```
> summary(rect)
  Length     Class          Mode
       1     rectangle      S4
```

Why OOP in statistical programs?

18

- Calculate mean and sd for binomial data:
  E(x) = np; Var(x) = np(1-p)

```
stats <- function(x) {
   n = length(x)
   p = mean(x)
   mu = n*p
   sigma = sqrt(n*p*(1-p))
   return( list(mu = mu, sigma =
sigma, n = n) )
}
```

```
> y1 <- rbinom(100,
   size=1, p=.3)
$mu
[1] 33
$sigma
[1] 4.702127
$n
[1] 100
```

But it can be falsely used!

- Misuse of the function

```
> y2<-rnorm(100,mean=0.3,sd=1)
> stats(y2)
$mu
[1] 42.52809
$sigma
[1] 4.943855
$n
[1] 100
```

True mean and standard deviation:

```
> mean(y2)
[1] 0.4252809
> sd(y2)
[1] 1.005184
```

Solution

- Use Object-oriented programming!

Define a class constructor (S3)


```
> as.binomial <- function(x) {
+    class(x) <- "binomial"
+    return( x )
+ }

> as.normal <- function(x) {
+    class(x) <- "normal"
+    return( x )
+ }
```

```
> y1 <- as.binomial(y1)
> class(y1)
[1] "binomial“

> y2 <- as.normal(y2)
> class(y2)
[1] "normal"
```

Generic function & methods (1)


```
stats <- function(x) UseMethod("stats")

stats.binomial <- function(x) {
   n = length(x)
   p = mean(x)
   mu = n*p
   sigma = sqrt(n*p*(1-p))
   return( list(mu = mu, sigma = sigma, n = n) )
}
```

Generic function & methods (1)


```
stats.normal <- function(x) {
   n = length(x)
   mu = mean(x)
   sigma = sd(x)
   return( list(mu = mu, sigma = sigma, n = n) )
}
stats.default <- function(x) stop("Unknown data
   type!")
```

Using the *stats* function (1)

- Objects with known classes

```
> Stats(y1)
$mu
[1] 33
$sigma
[1] 4.702127
$n
[1] 100
```

```
> stats(y2)
$mu
[1] 0.4252809
$sigma
[1] 1.005184
$n
[1] 100
```

Using the *stats* function (1)

- Objects with unknown classes

```
> y3 <- rnorm(100,mean=2,sd=1)
> stats(y3)
Error in stats.default(y3) : Unknown data type!
```

Inheritance

- A way to form new classes using classes that have already been defined.
- By the mechanism of inheritance, a new class (known as a derived class) takes over or inherits attributes and behavior of pre-existing classes (referred to as base classes or ancestor classes)

A class for standard normal


```
as.standardNormal <- function(x){
   x <- as.normal(x)
   class(x) <- c("standardNormal", class(x))
   return( x )
}

stats.standardNormal <- function(x){
   object <- stats.normal(x)
   object$sigma <- 1
   return(object)
}
```

Get methods by inheritance


```
> y4<-rnorm(100)
> y4<-as.standardNormal(y4)
> stats(y4)
$mu
[1] 0.03292102
$sigma
[1] 1
$n
[1] 100
> class(y4)
[1] "standardNormal" "normal"
```

Self-study

29

- **OOP used for building R statistical packages**

Enjoy using R!

30

- Any feedback, please forward to:

- Xiao-Lin (Nick) Wu
- E-mail: nick.wu@ansci.wisc.edu;
- xnwu@ansci.wisc.edu
- Office: AS building, rm# 448
- Tel: + 608 263 7824

* Thanks to Professors Gianola, Weigel, and Rosa for making this course to happen!