

Clases y Métodos

Oscar Perpiñán Lamigueiro

Universidad Politécnica de Madrid

① OOP en R

② Clases y métodos S3

③ Clases y métodos S4

Programación Orientada a Objetos (OOP)

- ▶ Los objetos encapsulan información y control de su comportamiento (*objects*).
- ▶ Las clases describen propiedades de un grupo de objetos (*class*).
- ▶ Se pueden definir clases a partir de otras (*inheritance*).
- ▶ Una función genérica se comporta de forma diferente atendiendo a la clase de uno (o varios) de sus argumentos (*polymorphism*).

OOP en R

En R coexisten dos implementaciones de la OOP:

- ▶ S3: elaboración informal con énfasis en las funciones genéricas y el polimorfismo.
- ▶ S4: elaboración formal de clases y métodos.

OOP en R

Referencias

- ▶ Software for Data Analysis
- ▶ How Methods Work
- ▶ S4 classes in 15 pages
- ▶ R Programming for Bioinformatics
- ▶ S4 System Development in Bioconductor

① OOP en R

② Clases y métodos S3

③ Clases y métodos S4

① OOP en R

② Clases y métodos S3

Clases

Métodos

③ Clases y métodos S4

Clases

Los objetos básicos en R tienen una clase implícita definida en S3. Es accesible con `class`.

```
x <- rnorm(10)
class(x)
```

```
[1] "numeric"
```

Pero no tienen atributo...

```
attr(x, 'class')
```

```
NULL
```

...ni se consideran formalmente objetos

```
is.object(x)
```

```
[1] FALSE
```


Clases

Se puede redefinir la clase de un objeto S3 con `class`

```
class(x) <- 'myNumeric'  
class(x)
```

```
[1] "myNumeric"
```

Ahora sí es un objeto...

```
is.object(x)
```

```
[1] TRUE
```

y su atributo está definido

```
attr(x, 'class')
```

```
[1] "myNumeric"
```

Sin embargo, su modo de almacenamiento (*clase intrínseca*) no cambia:

```
mode(x)
```

```
[1] "numeric"
```

Definición de Clases

```
task1 <- list(what='Write an email',  
             when=as.Date('2013-01-01'),  
             priority='Low')  
class(task1) <- 'Task'  
task1
```

```
$what  
[1] "Write an email"
```

```
$when  
[1] "2013-01-01"
```

```
$priority  
[1] "Low"
```

```
attr("class")  
[1] "Task"
```

```
task2 <- list(what='Find and fix bugs',  
             when=as.Date('2013-03-15'),  
             priority='High')  
class(task2) <- 'Task'
```

Definición de Clases

```
myToDo <- list(task1, task2)
class(myToDo) <- c('ToDo3')
myToDo
```

```
[[1]]
$what
[1] "Write an email"

$when
[1] "2013-01-01"

$priority
[1] "Low"

attr(,"class")
[1] "Task"

[[2]]
$what
[1] "Find and fix bugs"

$when
[1] "2013-03-15"

$priority
[1] "High"

attr(,"class")
[1] "Task"
```

Problemas de la sencillez de S3

```
notToDo <- list(task1, 2019)
class(notToDo) <- c('ToDo3')
notToDo
```

```
[[1]]
$what
[1] "Write an email"
```

```
$when
[1] "2013-01-01"
```

```
$priority
[1] "Low"
```

```
attr("class")
[1] "Task"
```

```
[[2]]
[1] 2019
```

```
attr("class")
[1] "ToDo3"
```

① OOP en R

② Clases y métodos S3

Clases

Métodos

③ Clases y métodos S4

Métodos con S3

Son **sencillos** de usar e implementar pero **poco robustos**.

Se definen a partir de un método genérico...

summary

```
function (object, ...)
UseMethod("summary")
<bytecode: 0x563c6f3687f0>
<environment: namespace:base>
```

...añadiendo a la función el nombre de la clase con un punto como separador.

summary.data.frame

```
function (object, maxsum = 7L, digits = max(3L, getOption("digits") -
  3L), ...)
{
  ncw <- function(x) {
    z <- nchar(x, type = "w", allowNA = TRUE)
    if (any(na <- is.na(z))) {
      z[na] <- nchar(encodeString(z[na]), "b")
    }
    z
  }
  z <- lapply(X = as.list(object), FUN = summary, maxsum = maxsum,
    digits = 12L, ...)
  nv <- length(object)
  nm <- names(object)
  lw <- numeric(nv)
  nr <- if (nv)
```

Métodos con S3

Con `methods` podemos averiguar los métodos que hay definidos para una función particular:

```
methods('summary')
```

```
[1] summary.aov                summary.aovlist*
[3] summary.aspell*           summary.check_packages_in_dir*
[5] summary.connection        summary.data.frame
[7] summary.Date              summary.default
[9] summary.ecdf*             summary.factor
[11] summary.glm               summary.infl*
[13] summary.lm                summary.loess*
[15] summary.manova            summary.matrix
[17] summary.mlm*              summary.nls*
[19] summary.packageStatus*    summary.POSIXct
[21] summary.POSIXlt           summary.ppr*
[23] summary.prcomp*           summary.princomp*
[25] summary.proc_time         summary.srcfile
[27] summary.srcref            summary.stepfun
[29] summary.stl*              summary.table
[31] summary.tukeysmooth*      summary.warnings
see '?methods' for accessing help and source code
```

Métodos con S3

Si no hay un método definido para la clase del objeto, UseMethod ejecuta la función por defecto:

`summary.default`

```
function (object, ..., digits, quantile.type = 7)
{
  if (is.factor(object))
    return(summary.factor(object, ...))
  else if (is.matrix(object)) {
    if (missing(digits))
      return(summary.matrix(object, quantile.type = quantile.type,
        ...))
    else return(summary.matrix(object, digits = digits, quantile.type = quantile.type,
      ...))
  }
  value <- if (is.logical(object))
    c(Mode = "logical", {
      tb <- table(object, exclude = NULL, useNA = "ifany")
      if (!is.null(n <- dimnames(tb)[[1L]]) && any(iN <- is.na(n))) dimnames(tb)[[1L]][iN] <- "NA's"
      tb
    })
  else if (is.numeric(object)) {
    nas <- is.na(object)
    object <- object[!nas]
    qq <- stats::quantile(object, names = FALSE, type = quantile.type)
    qq <- c(qq[1L:3L], mean(object), qq[4L:5L])
    if (!missing(digits))
      qq <- signif(qq, digits)
    names(qq) <- c("Min.", "1st Qu.", "Median", "Mean", "3rd Qu.",
      "Max.")
  }
}
```


Ejemplo de definición de método genérico

En primer lugar, definimos la función con UseMethod:

```
myFun <- function(x, ...)UseMethod('myFun')
```

... y la función por defecto.

```
myFun.default <- function(x, ...){  
  cat('Funcion genérica\n')  
  print(x)  
}
```

Ejemplo de definición de método genérico

Dado que aún no hay métodos definidos, esta función ejecutará la función por defecto.

```
methods('myFun')
```

```
[1] myFun.default  
see '?methods' for accessing help and source code
```

```
x <- rnorm(10)  
myFun(x)
```

```
Funcion genérica  
[1] 0.77353155 1.20335581 -1.03444954 -1.23169444 -0.55521295 -1.12915476 -0.29295587  
[8] 0.79667504 0.42126694 -0.06396166
```

```
myFun(task1)
```

```
Funcion genérica  
$what  
[1] "Write an email"  
  
$when  
[1] "2013-01-01"  
  
$priority  
[1] "Low"  
  
attr("class")  
[1] "Task"
```

Ejemplo de definición de método específico

```
myFun.Task <- function(x, number,...)
{
  if (!missing(number))
    cat('Task no.', number, ':\n')
  cat('What: ', x$what,
      '- When:', as.character(x$when),
      '- Priority:', x$priority,
      '\n')
}
```

`methods(myFun)`

```
[1] myFun.default myFun.Task
see '?methods' for accessing help and source code
```

`methods(class='Task')`

```
[1] myFun
see '?methods' for accessing help and source code
```

Método de Task

myFun(task1)

What: Write an email - When: 2013-01-01 - Priority: Low

myFun(task2)

What: Find and fix bugs - When: 2013-03-15 - Priority: High

myFun(myToDo)

Funcion genérica

```
[[1]]
```

```
$what
```

```
[1] "Write an email"
```

```
$when
```

```
[1] "2013-01-01"
```

```
$priority
```

```
[1] "Low"
```

```
attr("class")
```

```
[1] "Task"
```

```
[[2]]
```

```
$what
```

```
[1] "Find and fix bugs"
```

```
$when
```

```
[1] "2013-03-15"
```

Definición del método para ToDo3

Ejercicio

Define un método de `myFun` para la clase `ToDo3` con dos enfoques: sin tener en cuenta el método definido para `Task`; teniendo en cuenta el método para `Task`.

Definición del método para ToDo3

```
myFun.ToDo3 <- function(x, ...){
  cat('This is my ToDo list:\n')
  ## Cada uno de los elementos de un
  ## objeto ToDo3 son Task. Por tanto,
  ## x[[i]] es de clase Task y
  ## print(x[[i]]) ejecuta el metodo
  ## print.Task
  for (i in seq_along(x)) myFun(x[[i]], i)
  cat('-----\n')
}
```

myFun(myToDo)

```
This is my ToDo list:
Task no. 1 :
What: Write an email - When: 2013-01-01 - Priority: Low
Task no. 2 :
What: Find and fix bugs - When: 2013-03-15 - Priority: High
-----
```

① OOP en R

② Clases y métodos S3

③ Clases y métodos S4

① OOP en R

② Clases y métodos S3

③ Clases y métodos S4

Clases en S4

Métodos en S4

Clases S3 con clases y métodos S4

Clases en S4

Se construyen con `setClass`, que acepta varios argumentos

- ▶ `Class`: nombre de la clase.
- ▶ `slots`: una lista con las clases de cada componente. Los nombres de este vector corresponden a los nombres de los componentes (`slot`).
- ▶ `contains`: un vector con las clases que esta nueva clase extiende.
- ▶ `prototype`: un objeto proporcionando el contenido por defecto para los componentes definidos en `slots`.
- ▶ `validity`: a función que comprueba la validez de la clase creada con la información suministrada.

Datos de ejemplo

Vamos a ilustrar esta sección con datos de seguimiento GPS de gaviotas¹ empleando un extracto del conjunto de datos².



¹<https://lifewatch.inbo.be/blog/posts/bird-tracking-data-published.html>

²https://lifewatch.inbo.be/blog/files/bird_tracking.zip

Definición de una nueva clase

```
setClass('bird',  
  slots = c(  
    name = 'character',  
    lat = 'numeric',  
    lon = 'numeric',  
    alt = 'numeric',  
    speed = 'numeric',  
    time = 'POSIXct')  
)
```

Funciones para obtener información de una clase

```
getClass('bird')
```

```
Class "bird" [in ".GlobalEnv"]
```

```
Slots:
```

```
Name:      name      lat      lon      alt      speed      time
Class: character numeric numeric numeric numeric POSIXct
```

```
getSlots('bird')
```

```
      name      lat      lon      alt      speed      time
"character" "numeric" "numeric" "numeric" "numeric" "POSIXct"
```

```
slotNames('bird')
```

```
[1] "name" "lat" "lon" "alt" "speed" "time"
```

Creación de un objeto con la clase definida

Una vez que la clase ha sido definida con `setClass`, se puede crear un objeto nuevo con `new`. Es habitual definir funciones que construyen y modifican objetos para evitar el uso directo de `new`:

```
readBird <- function(name, path)
{
  csvFile <- file.path(path, paste0(name, ".csv"))

  vals <- read.csv(csvFile)

  new('bird',
      name = name,
      lat = vals$latitude,
      lon = vals$longitude,
      alt = vals$altitude,
      speed = vals$speed_2d,
      time = as.POSIXct(vals$date_time)
  )
}
```

Creación de objetos con la clase definida

```
eric <- readBird("eric", "data")  
nico <- readBird("nico", "data")  
sanne <- readBird("sanne", "data")
```

Acceso a los slots

A diferencia de \$ en listas y `data.frame`, para extraer información de los *slots* hay que emplear @ (pero no es recomendable):

```
eric@name
```

```
[1] "eric"
```

```
summary(eric@speed)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's  
0.0000  0.3437  1.0031  2.3005  2.4792 63.4881    85
```

Classes S4 con slots tipo lista

```
setClass("flock",  
  slots = c(  
    name = "character",  
    members = "list")  
)
```

```
notAFlock <- new("flock",  
  name = "flock0",  
  members = list(eric,  
                 3,  
                 "hello"))  
sapply(notAFlock@members, class)
```

```
[1] "bird"      "numeric"  "character"
```


Función de validación

```
valida <- function (object) {  
  if (any(sapply(object@members,  
                function(x) !is(x, "bird"))))  
    stop("only bird objects are accepted.")  
  return(TRUE)  
}
```

```
setClass("flock",  
        slots = c(  
          name = "character",  
          members = "list"),  
        validity = valida  
        )
```

Ejemplo de objeto S4 con slot tipo list

```
newFlock <- function(name, ...){  
  birds <- list(...)  
  new("flock",  
      name = name,  
      members = birds)  
}
```

```
notAFlock <- newFlock("flock0",  
                      eric, 2, "hello")
```

Error en validityMethod(object): only bird objects are accepted.

```
myFlock <- newFlock("flock1",  
                   eric, nico, sanne)
```

① OOP en R

② Clases y métodos S3

③ Clases y métodos S4

Clases en S4

Métodos en S4

Clases S3 con clases y métodos S4

Métodos en S4: setMethod

- ▶ Normalmente se definen con setMethod suministrando:
 - ▶ la clase de los objetos para *esta* definición del método (signature)
 - ▶ la función a ejecutar (definition).

```
setMethod('show',  
  signature = "bird",  
  definition = function(object)  
  {  
    cat("Name: ", object@name, "\n")  
    cat("Latitude: ", summary(object@lat), "\n")  
    cat("Longitude: ", summary(object@lon), "\n")  
    cat("Speed: ", summary(object@speed), "\n")  
  })
```

```
eric
```

```
Name: eric  
Latitude: 30.17401 30.43032 30.4624 39.05512 50.11692 51.36129  
Longitude: -9.928351 -9.643971 -9.630419 -4.409152 2.65808 3.601085  
Speed: 0 0.3436568 1.00312 2.300545 2.479153 63.48807 85
```

Métodos en S4: setMethod

```
setMethod('show',  
  signature = "flock",  
  definition = function(object)  
  {  
    cat("Flock Name: ", object@name, "\n")  
    N <- length(object@members)  
    lapply(seq_len(N), function(i)  
    {  
      cat("Bird #", i, "\n")  
      print(object@members[[i]])  
    })  
  })
```

myFlock

```
Flock Name: flock1  
Bird # 1  
Name: eric  
Latitude: 30.17401 30.43032 30.4624 39.05512 50.11692 51.36129  
Longitude: -9.928351 -9.643971 -9.630419 -4.409152 2.65808 3.601085  
Speed: 0 0.3436568 1.00312 2.300545 2.479153 63.48807 85  
Bird # 2  
Name: nico
```

Métodos en S4: setGeneric

- ▶ Es necesario que exista un método genérico ya definido.

```
isGeneric("as.data.frame")
```

```
[1] FALSE
```

- ▶ Si no existe, se define con setGeneric (y quizás standardGeneric).

```
setGeneric("as.data.frame")
```

```
[1] "as.data.frame"
```

- ▶ La función definition debe respetar los argumentos de la función genérica y en el mismo orden.

```
getGeneric("as.data.frame")
```

```
standardGeneric for "as.data.frame" defined from package "base"
```

```
function (x, row.names = NULL, optional = FALSE, ...)  
standardGeneric("as.data.frame")  
<environment: 0x563c6edfabb8>  
Methods may be defined for arguments: x, row.names, optional  
Use showMethods(as.data.frame) for currently available ones.
```

Métodos en S4: ejemplo con `as.data.frame`

```
setMethod("as.data.frame",  
  signature = "bird",  
  definition = function(x, ...)  
  {  
    data.frame(  
      name = x@name,  
      lat = x@lat,  
      lon = x@lon,  
      alt = x@alt,  
      speed = x@speed,  
      time = x@time)  
  })
```

```
ericDF <- as.data.frame(eric)
```

Métodos en S4: ejemplo con `as.data.frame`

Ejercicio

Define un método de `as.data.frame` para la clase `flock` a partir del método para la clase `bird`.

Métodos en S4: ejemplo con `as.data.frame`

```
setMethod("as.data.frame",  
  signature = "flock",  
  definition = function(x, ...)  
  {  
    dfs <- lapply(x@members, as.data.frame)  
    dfs <- do.call(rbind, dfs)  
    dfs$flock_name <- x@name  
    dfs  
  })
```

```
flockDF <- as.data.frame(myFlock)
```

Métodos en S4: ejemplo con xyplot

```
library(lattice)

setGeneric("xyplot")

setMethod('xyplot',
  signature = "bird",
  definition = function(x, data = NULL, ...)
  {
    df <- as.data.frame(x)
    xyplot(lat ~ lon, data = df, ...)
  })
```

```
[1] "xyplot"
```

```
xyplot(eric)
```

Métodos en S4: ejemplo con xyplot

Ejercicio

Define un método de `xyplot` para la clase `bird` que permita elegir entre diferentes modos de representación:

- ▶ `lontime`
- ▶ `lattime`
- ▶ `latlon`
- ▶ `speed`

Métodos en S4: ejemplo con xyplot

```
setMethod('xyplot',  
  signature = "bird",  
  definition = function(x, data = NULL,  
                        mode = "latlon", ...)  
  {  
    df <- as.data.frame(x)  
    switch(mode,  
      lontime = xyplot(lon ~ time, data = df, ...),  
      lattime = xyplot(lat ~ time, data = df, ...),  
      latlon = xyplot(lat ~ lon, data = df, ...),  
      speed = xyplot(speed ~ time, data = df, ...)  
    )  
  })
```

```
xyplot(eric, mode = "lontime")
```

Métodos en S4: ejemplo con `xyp1ot`

Ejercicio

Define un método de `xyp1ot` para la clase `flock` usando el color para distinguir a los diferentes integrantes (argumento `group` en `xyp1ot`).

Métodos en S4: ejemplo con xyplot

```
setMethod('xyplot',  
  signature = "flock",  
  definition = function(x, data = NULL, ...)  
  {  
    df <- as.data.frame(x)  
    xyplot(lon ~ lat,  
      group = name,  
      data = df,  
      auto.key = list(space = "right"))  
  })
```

```
xyplot(myFlock)
```

① OOP en R

② Clases y métodos S3

③ Clases y métodos S4

Clases en S4

Métodos en S4

Clases S3 con clases y métodos S4

Clases S3 con clases y métodos S4

Para usar objetos de clase S3 en signatures de métodos S4 o como contenido de slots de una clase S4 hay que registrarlos con `setOldClass`:

```
setOldClass('lm')
```

```
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
```


Ejemplo con lm y xyplot

Definimos un método genérico para xyplot

```
library(lattice)
setGeneric('xyplot')
```

```
[1] "xyplot"
```

Definimos un método para la clase lm usando xyplot.

```
setMethod('xyplot',
  signature = c(x = 'lm',
                data = 'missing'),
  definition = function(x, data,
                        ...)
  {
    fitted <- fitted(x)
    residuals <- residuals(x)
    xyplot(residuals ~ fitted, ...)
  })
```

Ejemplo con lm y xyplot

Recuperamos la regresión que empleamos en el apartado de Estadística:

```
lmFertEdu <- lm(Fertility ~ Education, data = swiss)
summary(lmFertEdu)
```

Call:

```
lm(formula = Fertility ~ Education, data = swiss)
```

Residuals:

Min	1Q	Median	3Q	Max
-17.036	-6.711	-1.011	9.526	19.689

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	79.6101	2.1041	37.836	< 2e-16 ***
Education	-0.8624	0.1448	-5.954	3.66e-07 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.446 on 45 degrees of freedom

Multiple R-squared: 0.4406, Adjusted R-squared: 0.4282

F-statistic: 35.45 on 1 and 45 DF, p-value: 3.659e-07

Ejemplo con lm y xyplot

```
xyplot(lmFertEdu, col='red', pch = 19,  
       type = c('p', 'g'))
```