8. The S3 Object System

CT5102 - J. Duggan

Attributes in R

- All objects can have arbitrary additional attributes, used to store meta-data about the object
- Attributes can be thought of as a named list (with unique names)
- Attributes can be accessed:
  - Individually with **attr()**
  - All at once with **attributes()**

Attributes in R

```
x <- 1:5
attr(x,"Att1") <- "Hello World"
attr(x,"Att2") <- Sys.time()
x
```

```
## [1] 1 2 3 4 5
## attr(,"Att1")
## [1] "Hello World"
## attr(,"Att2")
## [1] "2019-10-24 15:00:49 IST"
```

The structure() Function

- The structure function returns a new object with modified attributes

```
x <- structure(1:5,Att1="Hello World",Att2=Sys.time())
x
```

```
## [1] 1 2 3 4 5
## attr(,"Att1")
## [1] "Hello World"
## attr(,"Att2")
## [1] "2019-10-24 15:00:49 IST"
```

Properties of Attributes

By default, most attributes are lost when subsetting a vector

```
x <- structure(1:5,Att1="Hello World",Att2=Sys.time())
attributes(x)
```

```
## $Att1
## [1] "Hello World"
##
## $Att2
## [1] "2019-10-24 15:00:49 IST"
```

```
attributes(x[1:2])
```

```
## NULL
```

Attributes that are not removed...

- **Names**, a character vector giving each element a name. names(x)
- **Dimensions**, used to turn vectors into matrices and arrays. dim(x)
- **Class**, used to implement the S3 object system. class(x)

```
x <- structure(1:5,names=letters[1:5])
attributes(x)
```

```
## $names
## [1] "a" "b" "c" "d" "e"
```

```
attributes(x[1:2])
```

```
## $names
## [1] "a" "b"
```

dim()

Can be used to reshape a matrix

```
v <- 1:6
v
```

```
## [1] 1 2 3 4 5 6
```

```
attr(v,"dim") <- c(2,3)
v
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Challenge 8.1

For the vector 1:100, convert this to a 10 x 10 matrix using the attr() function

The S3 System

- Most OO languages implement message-passing OO
- Object determines which function to call
  - canvas.drawRect("blue")
- S3 Implements generic-function OO
- A special type of function called a generic function decides which method to call (i.e. method dispatch)
  - drawRect(canvas, "blue")
- S3 is a very casual system, it has no formal definition of classes

S3

- The only OO system used in the base and stats packages, and the most commonly used in CRAN packages
- "S3 in informal an ad-hoc, but has a certain elegance in its minimalism" (Wickham 2015)

```
typeof(mtcars)
```

```
## [1] "list"
```

```
class(mtcars)
```

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

```
sloop::otype(mtcars)
```

```
## [1] "S3"
```

S3 Methods

- In S3, methods belong to functions, called generic functions
- S3 methods to not belong to objects or classes
- To determine if a function is an S3 generic, inspect the source code for a call to UseMethod()

```
print
```

```
## function (x, ...)
## UseMethod("print")
## <bytecode: 0x7fae3a418070>
## <environment: namespace:base>
```

- UseMethod() – Figures out the correct method to call, the process of method dispatch
- Method names tend to be **generic.class()**

Showing methods that belong to a generic function

```
methods(mean)
```

```
## [1] mean.Date         mean.default      mean.difftime     mean
## [5] mean.POSIXlt      mean.quosure*     mean.vctrs_vctr*
## see '?methods' for accessing help and source code
```

```
length(methods(print))
```

```
## [1] 263
```

```
methods(print)[1:2]
```

```
## [1] "print.acf" "print.AES"
```

Show generics that have methods for a class

```
methods(class="data.frame")
```

```
##   [1] [                 [[                  [[<-                [<-
##   [5] %within%          $<-                 aggregate           anti_joi
##   [9] anyDuplicated     arrange_            arrange             as_tibbl
##  [13] as.data.frame     as.list             as.matrix           as.tbl_c
##  [17] as.tbl            by                  cbind               coerce
##  [21] collapse          collect             complete_           complete
##  [25] compute           dim                 dimnames            dimnames
##  [29] distinct_         distinct            do_                 do
##  [33] drop_na_          drop_na             droplevels          duplicat
##  [37] edit              expand_             expand              extract_
##  [41] extract           fill_               fill                filter_
##  [45] filter            format              formula             fortify
##  [49] full_join         gather_             gather              ggplot_a
##  [53] glimpse           group_by_           group_by            group_da
##  [57] group_indices     group_indices       group_keys          group_ma
```

Creating S3 objects

- S3 objects usually built on top of lists, or atomic vectors with attributes
- class(x) shows the class of an object

```
o <- list(Name="Test")
str(o)
```

```
## List of 1
##  $ Name: chr "Test"
```

```
attr(o,"class") <- "my_class"
str(o)
```

```
## List of 1
##  $ Name: chr "Test"
##  - attr(*, "class")= chr "my_class"
```

```
class(o)
```

```
## [1] "my_class"
```

Using class()

```
o <- list(Name="Test")
class(o) <- "my_class"
str(o)
```

```
## List of 1
##  $ Name: chr "Test"
##  - attr(*, "class")= chr "my_class"
```

```
sloop::otype(o)
```

```
## [1] "S3"
```

Using structure()

```
o <- structure(list(Name="Test"),class="my_class")
str(o)
```

```
## List of 1
##  $ Name: chr "Test"
##  - attr(*, "class")= chr "my_class"
```

```
class(o)
```

```
## [1] "my_class"
```

```
sloop::otype(o)
```

```
## [1] "S3"
```

Most S3 classes provide a constructor function

```
my_class <- function(x){
  structure(list(Name=x),class="my_class")
}
o <- my_class("Test")
class(o)
```

```
## [1] "my_class"
```

```
sloop::otype(o)
```

```
## [1] "S3"
```

Writing methods for S3 - using existing generic functions

```
print.my_class <- function (x){
  summary(x)
}

v <- 1:5
print(v)
```

```
## [1] 1 2 3 4 5
```

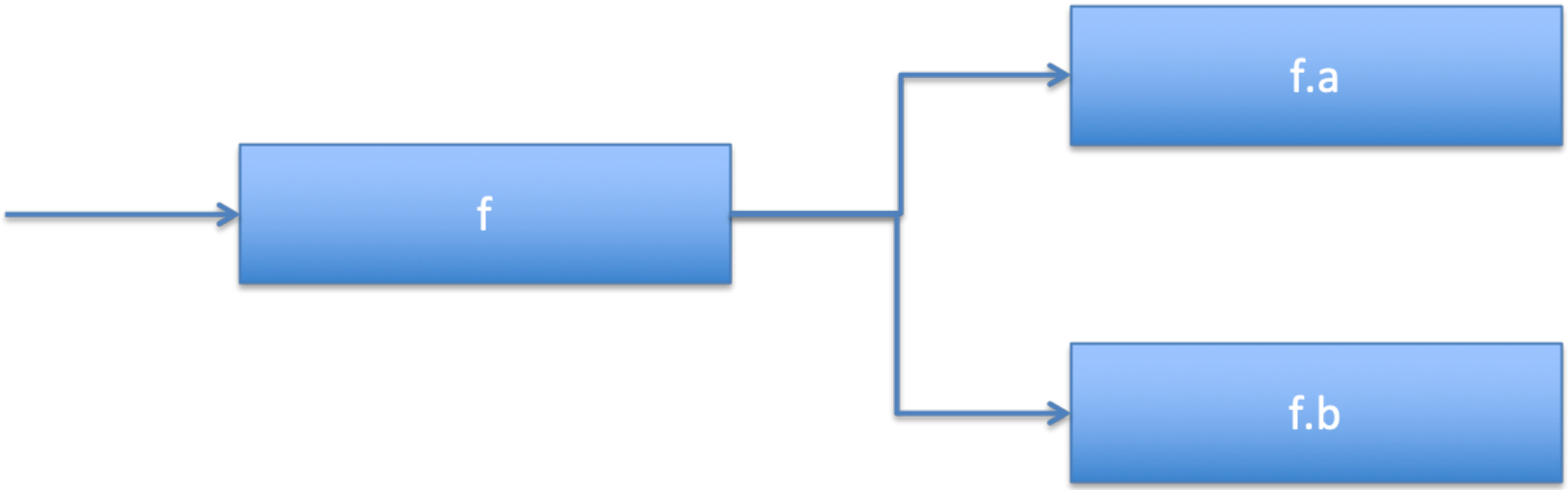
```
class(v)<-"my_class"
print(v)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       2       3       3       4       5
```

Adding a new generic function

- To add a new generic, create a function that calls UseMethod()
- UseMethod takes two arguments
  - The name of the generic function
  - The argument to use for method dispatch
- If the 2nd argument is omitted, it will dispatch on the first argument to the function
- Methods are then added, using a regular function with the name generic.class

Overall Idea f() is a generic function



Format for specific functions are *[generic function].[class name]*

Example

```
f <- function (x){
  UseMethod("f")
}

f.a <- function(x){
  print("This is function f.a()")
}

f.b <- function(x){
  print("This is function f.b()")
}
o <- structure(list(Test=1:2),class="a")
f(o)
```

```
## [1] "This is function f.a()"
```

Default functions

```
f.default <- function(x){
  print("This is function f.default()")
}

o <- structure(list(Test=1:2),class="a")
f(o)
```

```
## [1] "This is function f.a()"
```

```
p <- structure(list(Test=1:2),class="c")
f(p)
```

```
## [1] "This is function f.default()"
```

Challenge 8.2

Write a function that will result in the following behaviour

```
mtcars[1:2,1:5]
```

```
##                mpg cyl disp  hp drat
## Mazda RX4       21   6  160 110  3.9
## Mazda RX4 Wag   21   6  160 110  3.9
```

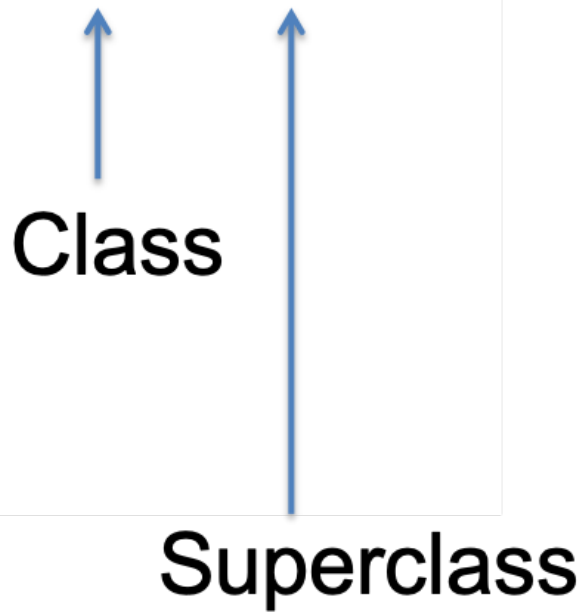
```
summary(mtcars)
```

```
## [1] "Hello World"
```

Inheritance

The idea of inheritance is to form new classes of specialised versions of existing ones.

```
> z<-structure(list(),class=c("b","a"))
>
> z
list()
attr(,"class")
[1] "b" "a"
```



Example - define two generic functions f() and g()

```
f <- function(x){
  UseMethod("f")
}

g <- function(x){
  UseMethod("g")
}
```

Create methods for two classes (a and b)

```
f.a <- function(x){
  print("Calling method f.a()")
}

f.b <- function(x){
  print("Calling method f.b()")
}

g.a <- function(x){
  print("Calling method g.a()")
}
```

z is class b, inherits from class a

```
z <- structure(1:2, class=c("b","a"))
class(z)
```

```
## [1] "b" "a"
```

```
f(z)
```

```
## [1] "Calling method f.b()"
```

```
g(z)
```

```
## [1] "Calling method g.a()"
```

Challenge 8.3

- Create a new class my_lm that inherits from lm
- Write a summary function for this new class

```
ans <- my_lm(faithful$eruptions, faithful$waiting, faithful)
class(ans)
```

```
## [1] "my_lm" "lm"
```

```
summary(ans)
```

```
## [1] "My Summary will appear here..."
```

```
coefficients(ans)
```

```
## (Intercept)           x
## -1.87401599  0.07562795
```

S3 Summary

- Implements generic-function OO
- A special type of function called a generic function decides which method to call (i.e. method dispatch)
- S3 is a very casual system, it has no formal definition of classes
- Inheritance can be used to leverage existing R S3 classes