Object-Oriented Programming in R

Mohammad H. Ferdosi

Animal Genetics and Breeding Unit
University of New England

7/12/2020

Programming Paradigms

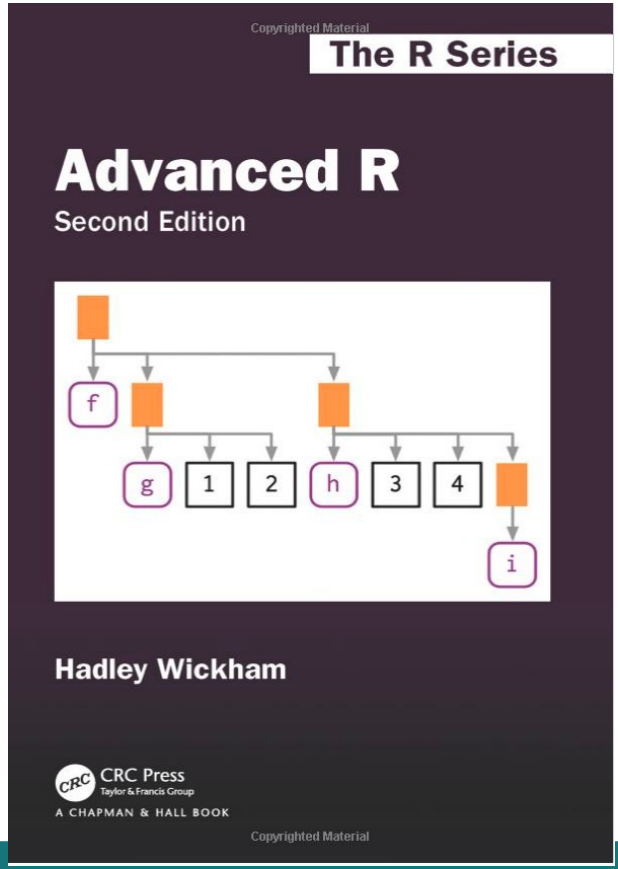
- Classifying programming languages by their features (style)
- Major types of programming paradigms:
  - Generic → templates (variable types specified later)
  - Declarative
  - Functional
    - Previous presentation:
      https://github.com/rgroupune/Session_material/blob/master/20191129_functional_programming.pdf
  - Imperative
    - Procedural → based on procedure, routines, subroutines or functions
    - Object-oriented → object, class, method, member

Object Oriented Programming

- “Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).”
  - wikipedia

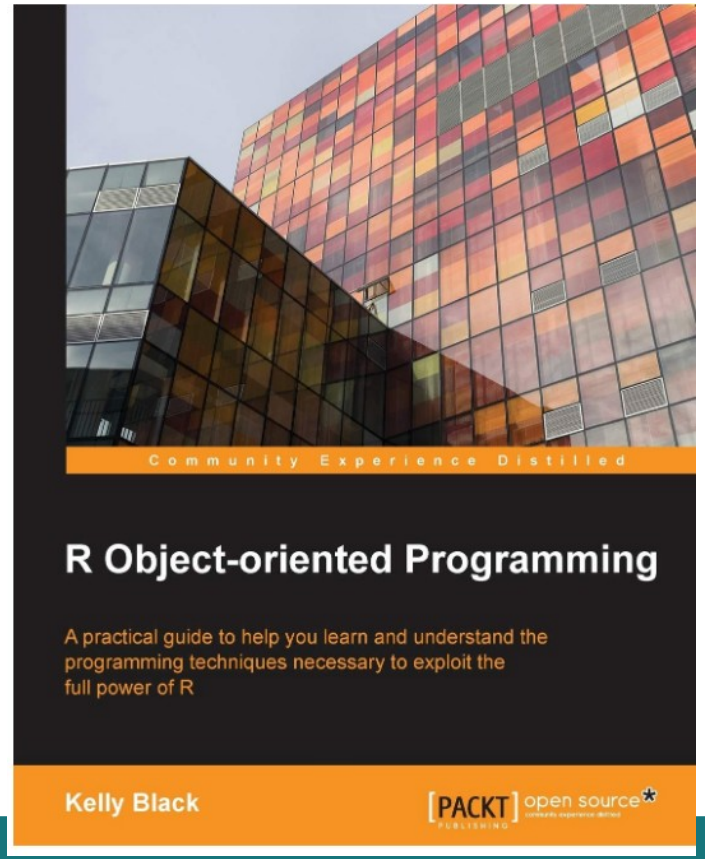
Object-Oriented Programming

- Advanced R, Second Edition
- Hadley Wickham
- The author of ggplot2, readr,
- dplyr, reshape2 and ...
- https://adv-r.hadley.nz/fp.html



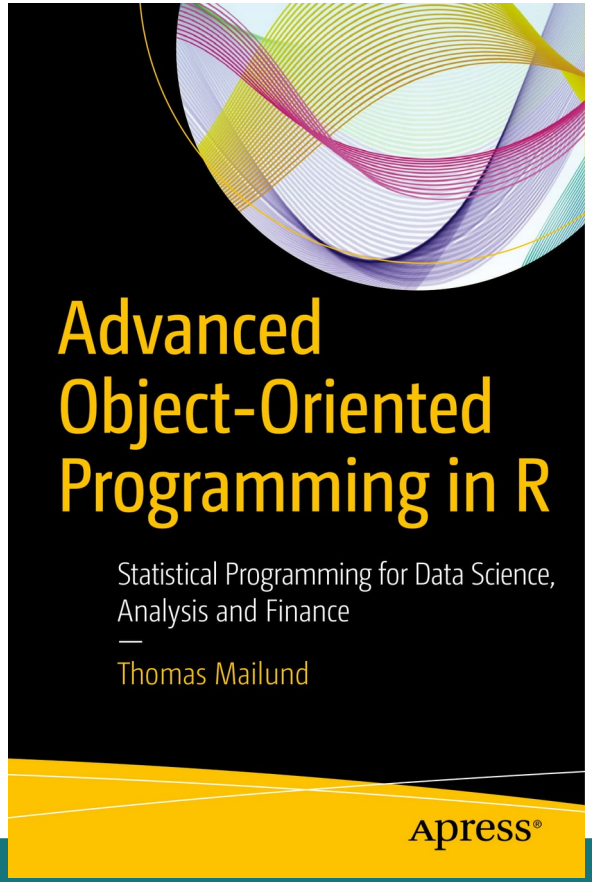
Object-Oriented Programming

- Kelly Black



Object-Oriented Programming

- Thomas Mailund



OOP languages:

- Java
- PHP
- Python
- C++
- S
  - R
- ...

- Full list:
  - https://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages

Object Types in R - Class

- For example:
  - A = matrix(1:4, 2)
  - class(A)
    - [1] "matrix"
  - B = list(1:4)
  - class(B)
    - [1] "list"

Object-Oriented

- Class
- Method (function)
- Member (data)

  - Inheritance

S3 and S4 Classes

- S3 Class
  - Classic and simple
  - Examples: plot, summary
- S4 Class
  - Stricter and complex
  - Example: R packages in Bioconductor project

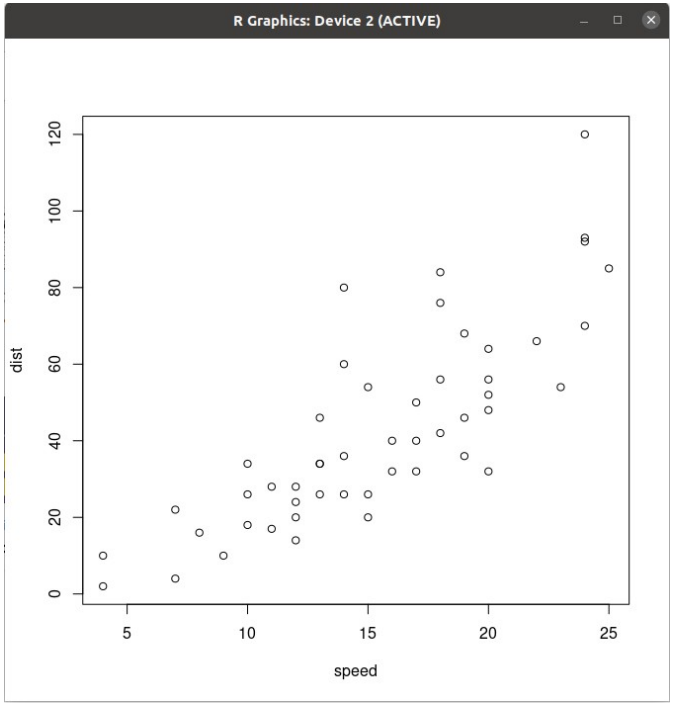
Simple Data-set

- cars
- head(cars)

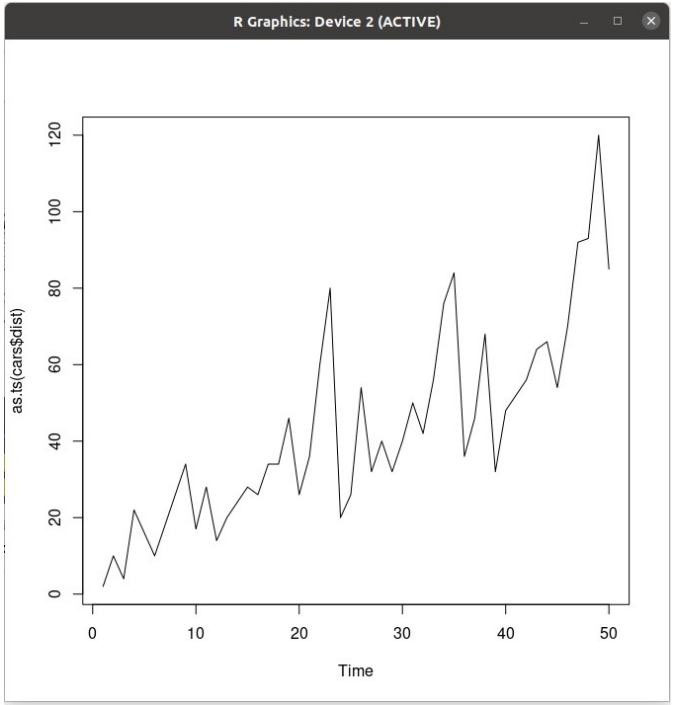
	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10

S3 Class - Examples

- plot(cars)

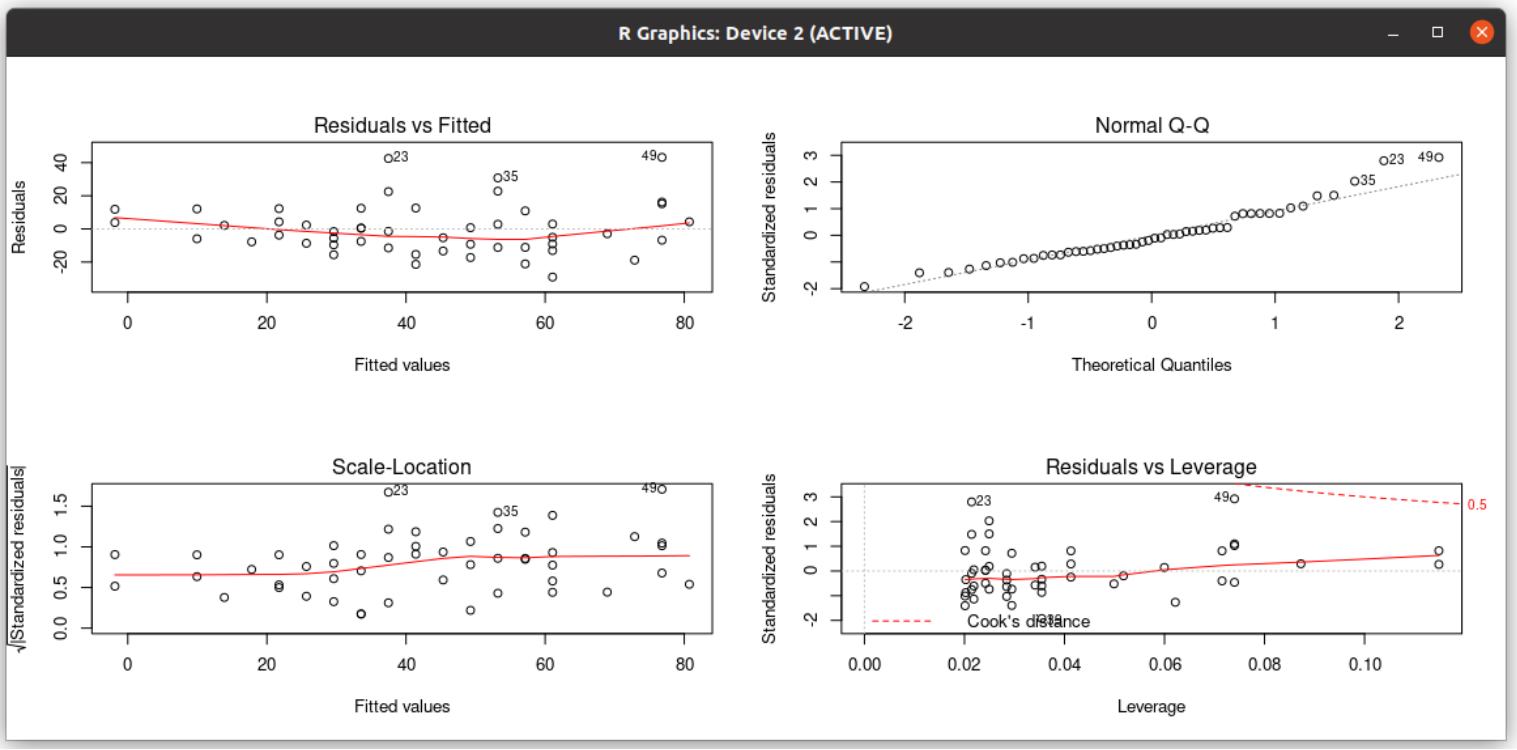


plot(as.ts(cars$dist))



Impressive!

- plot(lm(cars$dist~cars$speed))



What else can I plot?!

Listing all available methods

- methods(plot)

```
 [1] plot.acf*            plot.data.frame*     plot.decomposed.ts*
 [4] plot.default         plot.dendrogram*     plot.density*
 [7] plot.ecdf            plot.factor*         plot.formula*
[10] plot.function        plot.hclust*         plot.histogram*
[13] plot.HoltWinters*    plot.isoreg*         plot.lm*
[16] plot.medpolish*      plot.mlm*            plot.ppr*
[19] plot.prcomp*         plot.princomp*       plot.profile.nls*
[22] plot.raster*         plot.spec*           plot.stepfun
[25] plot.stl*            plot.table*          plot.ts
[28] plot.tskernel*       plot.TukeyHSD*
```

How do I know that “plot” is S3 class?

- isS3stdGeneric(plot)

  plot

  TRUE
- isS3stdGeneric(mean)
- isS3stdGeneric(summary)
- isS3stdGeneric(ls)

How do I know that “plot” is S3 class?

- isS3stdGeneric(plot)

  plot

  TRUE
- isS3stdGeneric(mean) → TRUE
- isS3stdGeneric(summary) → TRUE
- isS3stdGeneric(ls) → FALSE

Add a new method to a generic Class

- Adding a new method to the plot

```
library(hsphase)
plot.hsphase <- function(bmh)
{
   imageplot(bmh)
}
```

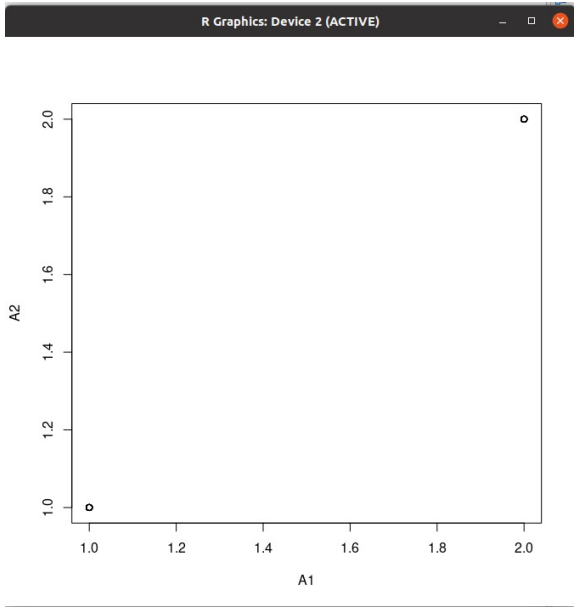
Preparing Data

```
data(genotypes)

data(map)

data(pedigree)

halfsib <- hss(pedigree, genotypes)

halfsib <- cs(halfsib,map)

halfsib <- halfsib[[1]]

blockMat <- bmh(halfsib)
```

Plotting the block matrix

class(blockMat); x11(); plot(blockMat)

- This is not what we expected!



Plotting the Block Matrix

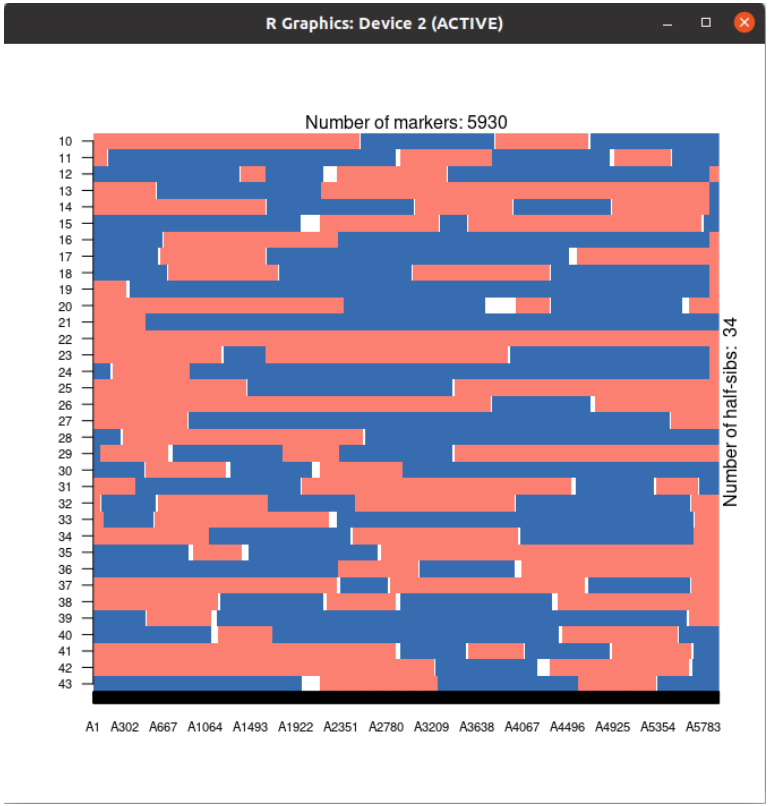
- Because “blockMat” is a matrix:

```
class(blockMat)
"matrix"
```

- Change the class:

```
class(blockMat) <- "hsphase"
X11(); plot(blockMat)
```

- Looks good!



Listing all available methods

- methods(plot)

```
[1] plot.acf*            plot.data.frame*    plot.decomposed.ts*
 [4] plot.default        plot.dendrogram*    plot.density*
 [7] plot.ecdf           plot.factor*        plot.formula*
[10] plot.function       plot.hclust*        plot.histogram*
[13] plot.HoltWinters*   plot.hsphase        plot.isoreg*
[16] plot.lm*            plot.medpolish*     plot.mlm*
[19] plot.ppr*           plot.prcomp*        plot.princomp*
[22] plot.profile.nls*   plot.raster*        plot.snowTimingData
[25] plot.spec*          plot.stepfun        plot.stl*
[28] plot.table*         plot.ts             plot.tskernel*

[31] plot.TukeyHSD*
```

Simple Genomic data

- Genotype data
  - AA: 0, AB: 1 and BB: 2 (Marker – SNPs)
  - A matrix (individuals by SNPs)
- GC score:
  - Between 0 and 1 → reflect genotype (allele) quality
  - A matrix (individuals by SNPs)
- Phenotype
  - ID
  - Age
  - Colour

Genotype Data Simulation

set.seed(1); geno = matrix(sample(c(0, 1, 2), 28, T), nrow = 4)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
• [1,]	0	0	1	0	1	2	0
• [2,]	2	2	2	0	1	0	1
• [3,]	0	2	2	1	2	0	0
• [4,]	1	1	0	1	0	0	0

GC Score

- set.seed(1); gc = round(matrix(runif(28, 0, 1), nrow = 4), 2)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
[1,]	0.27	0.20	0.63	0.69	0.72	0.93	0.27
[2,]	0.37	0.90	0.06	0.38	0.99	0.21	0.39
[3,]	0.57	0.94	0.21	0.77	0.38	0.65	0.01
[4,]	0.91	0.66	0.18	0.50	0.78	0.13	0.38

Create your first S3 classes

- genotype <- list(genotype = geno)
- class(genotype) <- "genome"
- class(genotype)
- GC <- list(GC = gc)
- class(GC) <- "GenCall"
- class(GC)

Check the individual genotype

- UseMethod → Processing the generic function
  - Generic → Name of function
  - Object → Object to be dispatched

Example

```
rowStat <- function(myObject)
{
    UseMethod("rowStat", myObject)
}
rowStat.default <- function(myObject)
{
   stop("Undefined Object!")
}
```

Example

```
rowStat.genome <- function(myObject)
{
    res <- list()
  res$FRQ <- apply(myObject$genotype, 1, function(x)
    {
      x <- factor(x, levels = c(0, 1, 2))
            table(x)
     })
   res$geno <- myObject
      return(res)
}
```

Example

```
rowStat(geno)  # Show an error

rowStat(genotype)

rowStat(GC)
```

Phenotype

- set.seed(1)
- Phenotype = data.frame(ID = letters[1:4], Age = sample((1:10), 4),color = c("brown", "white", "black", "gray"))
- Phenotype

  ID Age color

  1 a 9 brown

  2 b 4 white

  3 c 7 black

  4 d 1 gray

Using JoinedUp class

- MyData = as.JoinedUp(Phenotype, geno, gc)
- class(MyData)

JoinedUp function

```
as.JoinedUp <- function(link_file, genotype, gc)
{
    stopifnot(is.data.frame(link_file), is.matrix(genotype), nrow(link_file) == nrow(genotype), nrow(genotype) == nrow(gc))
    stopifnot(rownames(genotype) == rownames(gc))
    stopifnot(colnames(genotype) == colnames(gc))
    stopifnot(max(as.vector(gc)) <= 1)
    alleles <- unique(as.vector(genotype))
    if (length(alleles) > 4)
    {
        print(alleles)
        warning(paste("There are more that 3 types of alleles plus NAs"))
    }
    x <- list(link_file = link_file, genotype = genotype, gc = gc)
    class(x) <- "JoinedUp"
    return(x)
}
```

JoinedUp function

```
MyData <- as.JoinedUp(Phenotype, geno, gc)

class(MyData)
```

JoinedUp function

```
`[.JoinedUp` <- function(x, i = NULL, j = NULL)
{
    ....
}
```

- Check the full source code of this function in the “joinUP.R” file

JoinedUp function

- You must check for all possible errors
- Because you are making tools!

JoinedUp Class

```
MyData[1:2,1:2] # '[' has been redefined!
$link_file
  ID Age color
1  a   2 brown
2  b   7 white
$genotype
  [,1] [,2]
a    0    0
b    2    2
$gc
  [,1] [,2]
a 0.27  0.2
b 0.37  0.9
attr(,"class")
[1] "JoinedUp"
```

JoinedUp Class

```
dim(MyData)
[1] "Link file: 4 3"
[1] "Genotype: 4 7"
[1] "GC: 4 7"
dimnames(MyData)
gc_max(MyData, .5)
rbind(MyData,gc_max(MyData, .5))
```

JoinedUp Class

```
summary(MyData)
[1] "GC Summary:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0100  0.2550  0.4450  0.5029  0.7325  0.9900
[1] "Allele Summary:"
 0  1  2
13  8  7
[1] "Link file summary"
'data.frame': 4 obs. of  3 variables:
 $ ID   : Factor w/ 4 levels "a","b","c","d": 1 2 3 4
 $ Age  : int  2 7 3 6
 $ color: Factor w/ 4 levels "black","brown",..: 2 4 1 3
NULL
[1] "Missing summary SNPs"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0       0       0       0       0       0
[1] "Missing summary individuals"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0       0       0       0       0       0
```

Introduction to S4 Class

- More like advanced class in other languages like C++ and Python

Defining S4 Class

- setClass
  - class → name of class
  - slots
  - prototype
  - validity

Example

```
setClass("Genotype", slots = representation(name = "character",
geno = "matrix", gc = "matrix"))
```

Make an instance of the object using new function

```
Sheep_1 <- new("Genotype", name = "mySheep", geno = geno, gc = gc)
```

- But the “new” function is only for you! It is better to wrap in a function:

```
Genotype <- function(name,geno,gc)
{
    new("Genotype", name = "mySheep", geno = geno, gc = gc)
}
```

- Check the class:

```
class(Sheep_1)
```

Accessing the class member

You can Use “@”, for example:

```
Sheep_1@name

Sheep_1@geno

Sheep_1@dob # The dob (date of birth) has not defined!
```

Accessing the class member

Or you can use “slot” function:

```
slot(Sheep_3, "geno")
```

Changing the class member

```
Sheep_3 <- new("Genotype", name = "myThSheep")

Sheep_3@geno <- geno # be careful!

Sheep_3@geno <- 2 # Error: 2 is not a matrix – a very basic type check
```

It is always a good practice to use Accessors/Mutator to access or change the member of a class (Explained later).

Set default values for the class member (Prototype)

```
setClass("Genotype", representation(name = "character", geno = "matrix", gc = "matrix"), prototype = list(name = "graySheep", geno = matrix(0), gc = matrix(0)))

Sheep_4 <- new("Genotype")

Sheep_4

An object of class "Genotype"

Slot "name":

"graySheep"

Slot "geno":

<0 x 0 matrix>

Slot "gc":

<0 x 0 matrix>
```

Validating the class member

```
setClass("Genotype", representation(name = "character",geno = "matrix", gc = "matrix"),
prototype = list(name = "graySheep", geno = matrix(0), gc = matrix(0)),
validity = function(object)
{
    if(any(object@gc<0 | object@gc>1))
    {
      print(range(object@gc))
      return("Error: The gc must be between 0 and 1!")
    }
})
```

Validating the class member

```
Sheep_1 <- new("Genotype", name = "mySheep", geno = gc, gc = gc) # validating the genotype!

Sheep_1 <- new("Genotype", name = "mySheep", geno = t(geno), gc = gc) # validating the genotype!

Sheep_1@geno = t(geno) # Never do this - it is working without validation,

validObject(Sheep_1)   # although you can check it later!
```

S4 methods

- setMethod
  - Arguments
    - F: Name of generic function
    - Signature: class name
    - Definition: A function definition

Inheritance (Getting the property of the other class/s)

```
setClass("Merino", representation(woolWeight = "numeric"), contains = "Genotype")

dolly <- new("Merino", woolWeight = 3, geno = geno, gc = gc)


setClass("Suffolk", representation(weight = "numeric"), contains = "Genotype")

dolly_2 <- new("Suffolk", weight = 3, geno = geno, gc = gc) # Do you know a better  name for sheep:)
```

Defining methods/functions

```
setGeneric("qualityControl", function(object) standardGeneric("qualityControl"))
setMethod("qualityControl", "Genotype", function(object)
{
  object@geno[object@gc < .6] <- NA
  object
})
dolly <- qualityControl(dolly)
dolly_2 <- qualityControl(dolly_2)
Sheep_1 <- qualityControl(Sheep_1)
```

Accessors

```
setGeneric("name", function(x) standardGeneric("name"))
setMethod("name", "Genotype", function(x) x@name)
name(dolly)
setGeneric("name<-", function(x, value) standardGeneric("name<-"))
setMethod("name<-", "Genotype", function(x, value)
{
    x@name <- value
    validObject(x)
    x
})
name(dolly) <- "A Sheep!"
name(dolly)
```

Changing the genotype / Using Accessors

```
# Changing the genotype
setGeneric("genotype", function(object) standardGeneric("genotype"))
setMethod("genotype", "Genotype", function(object) object@geno)
setGeneric("genotype<-", function(object, value) standardGeneric("genotype<-"))
setMethod("genotype<-", "Genotype", function(object, value) {
  print(value)
  object@geno<- value
  validObject(object)
  object
})
genotype(dolly_2)
genotype(dolly_2) <- matrix(sample(c(0, 1, 2), 28, T), nrow = 4)
```

Acknowledgement

- Majid Khansefid
  - For providing additional reference and valuable comments on my slides

Thanks for your Attention