S3 and S4 systems in R

Chao-Jen Wong @ RLadies

January 23, 2019

Object-oriented programming in R

Major uses

- construction of self-describing data classes, easier to store and share
- different manipulations performed on the objects
- more maintainable and reliable packages

Object-oriented programming in R

Supports

- objects
- classes
- inheritance
- polymorphism

Two internal OOP systems

S3 (1992)

- classes are loosely defined
- simple, linear inheritance
- single dispatch system for generic functions and methods

S4 (1998, 2008)

- overcome many deficiencies of S3
- explicit defined classes and generic function
- multiple inheritance
- increased complexity of use

Both are not *class-centric* systems.

S3: generic functions

Responsibilities

- setting up evaluation environment
- initiate dispatch system

Example

```
#' print() is a generic function
print
```

```
## function (x, ...)
## UseMethod("print")
## <bytecode: 0x7fde1ffea038>
## <environment: namespace:base>
```

- a single expression which is the call to `UseMethod`
- `UseMethod` initiates the dispatch on single argument, usually the first one

S3: methods and dispatch mechanism

Methods are regular functions identified by their names.

Naming convention: a concatenation of the name of the generic and the name of the class, separated by a dot.

```
#' list of all available methods for a generic function
methods(print)[92:96]
```

```
## [1] "print.fseq"        "print.ftable"      "print.function"
## [4] "print.getAnywhere" "print.glm"
```

Dispatch

- a call to `UseMethod` and find most specific method
- make a new function call with arguments in the same order as there were supplied to the generic function

S3: some useful functions

```
#' Find available methods for a given class
methods(class="glm")
#' get a specific method for a given class
getS3method("print", "glm")
```

S3: classes

Loose definition. No valid check. No initialization.

Example: `Passenger` class

```
# Define S3 class: set value of the class attribute
x <- list(name="Josie Andre", origin="SEA", destination="KCM")
class(x) <- "Passenger" # set value of the class attribute
# define a child of Passenger
y <- list(name="Josie Andre", origin="SEA", destination="KCM",
          ffnumber=10)
class(y) <- c("FreqFlyer", "Passenger")
class(x)
```

```
## [1] "Passenger"
```

```
inherits(y, "Passenger")
```

```
## [1] TRUE
```

S3: generic functions and methods

Using `print.default()` to print a `Passenger` instance.

```
print(x)
```

```
## $name
## [1] "Josie Andre"
##
## $origin
## [1] "SEA"
##
## $destination
## [1] "KCM"
##
## attr(,"class")
## [1] "Passenger"
```

S3: generic functions and methods

```
# define foo generic function
foo <- function(x, ...) UseMethod("foo")
# define default. invoked when no applicable methods are found
foo.default <- function(x, ...) {
   print.default(x)
}
# define method applied to Passenger
foo.Passenger <- function(x) {
  cat(paste0("name: ", x$name, "\n"))
  cat(paste0("origin: ", x$origin, "\n"))
  cat(paste0("dest: ", x$destination), "\n")
  #NextMethod() - transfer control to the next most specific method
}
foo(x)
```

```
## name: Josie Andre
## origin: SEA
## dest: KCM
```

```
# getS3method("foo", "Passenger")
```

S4 system

Overcome the deficiencies of S3 system

- multiple dispatch
- object validation
- explicit representation of classes virtual/concrete classes
- multiple inheritance
- class unions

S4 system

Core components

- class definition
- validation
- constructor function (named as the class)
- generic and methods: getters, setters, and other accessories
- `show` method

S4: concrete class

Define classes: `setClass`, `slots`, `contains`, `prototype` (initialization), `validity` (validation).

```
setClass("Passenger",
  slots = c(name = "character",
            origin = "character",
            destination = "character"),
  prototype = prototype(name = NA_character_,
                        origin = NA_character_,
                        destination = NA_character_))


#' subclass inheriting properties of superclass
setClass("FreqFlyer",
  contains = "Passenger",
  slots= c(ffnumber = "numeric"))
```

S4: classes

Create a `Passenger` instance:

```
x <- new("Passenger", name="Josephine Andre",
         origin = "SEA",
         destination = "KCM")
x
```

```
## An object of class "Passenger"
## Slot "name":
## [1] "Josephine Andre"
##
## Slot "origin":
## [1] "SEA"
##
## Slot "destination":
## [1] "KCM"
```

```
# showClass("Passenger")
# slotNames("Passenger")
```

S4: Validation

Better practice: instead of `setClass(..., validity=...)`, use `setValidity()`

```
setValidity("Passenger", function(object) {
  #' slots must have the same length
  slot_lengths <- c(length(object@name),
                    length(object@origin),
                    length(object@destination))
  if (length(unique(slot_lengths)) != 1)
    return("'name', 'origin' and 'destination' must have the same length")

  return(TRUE)
})
```

```
## Class "Passenger" [in ".GlobalEnv"]
##
## Slots:
##
## Name:         name      origin destination
## Class:   character   character   character
##
## Known Subclasses: "FreqFlyer"
```

S4: generic functions/methods

Define `show` method

```
#` methods("show")
#` show is already a generic function for S4 classes
setMethod("show", signature="Passenger",
          function(object) {
          cat(paste0("name: ", object@name, "\n"))
          cat(paste0("origin: ", object@origin, "\n"))
          cat(paste0("destination: ", object@destination, "\n"))
          })
show(x)
```

```
## name: Josephine Andre
## origin: SEA
## destination: KCM
```

```
# showMethods("show")
# selectMethod("show", "Passenger")
```

S4: constructor

Instead of `new()`, make a constructor function using the same name as the object.

```
#` constructor, user-friendly, should be documented
Passenger <- function(name, origin, destination) {
    new("Passenger", name=name, origin=origin,
        destination=destination)
}
y <- Passenger(name="Josie Andre", origin="SEA", destination="NYC")
```

How to access the internal of an instance of S4 classes ?

```
#` example
y@name
```

```
## [1] "Josie Andre"
```

S4: accessors

- allow user to access the internal of an object via accessors (getters and setters). R does not have provision of encapsulation.
- crafted setters ensure the object remains valid

```
#' generic getter
setGeneric("name", function(object) standardGeneric("name"))
```

```
## [1] "name"
```

```
setMethod("name", signature("Passenger"),
    function(object) {object@name})
```

S4: accessors

```
#' setter (replacement methods)
setGeneric("name<-", function(object, value) standardGeneric("name<-"))
```

```
## [1] "name<-"
```

```
setMethod("name<-", signature("Passenger"),
    function(object, value) {
        #validate length
        # validate value
        if (is.character(value))
            object@name <- value
        if (!is.character(value))
            warning("`name` in a Passenger class must be character")
        object
    })
```

S4: accessors

Test

```
name(y)
```

```
## [1] "Josie Andre"
```

```
name(y) <- "Derek Andre"
name(y) <- 0
```

```
## Warning in `name<-`(`*tmp*`, value = 0): `name` in a Passenger class must
## be character
```

S4: coercion

Coerce `Passenger` to `data.frame`

```
setAs("Passenger", "data.frame",
      function(from)
          data.frame(name = from@name,
                     origin = from@origin,
                     destination = from@destination))
as(y, "data.frame")
```

```
##          name origin destination
## 1 Derek Andre    SEA         NYC
```

Method discovery: `methods()`, `.S3methods()`, `.S4methods()`

```
.S4methods(class="Passenger")
```

```
## [1] coerce name   name<- show
## see '?methods' for accessing help and source code
```

S4: virtual classes

Cannot be instantiated. Used as the parent of one or more concrete classes.

```
setClass("A", contains="VIRTUAL") # can have slots or not
# can define generic functions delicate to the virtual class
setClass("A1", contains = "A", slots = ...add slots here...)
setClass("A2", contains = "A", slots = ...add slots here...)
```

S4: virtual class example - `GenomicRanges`

```
suppressPackageStartupMessages(library(GenomicRanges))
showClass("GenomicRanges")
```

```
## Virtual Class "GenomicRanges" [package "GenomicRanges"]
##
## Slots:
##
## Name:  elementMetadata     elementType        metadata
## Class:       DataFrame       character            list
##
## Extends:
## Class "Ranges", directly
## Class "GenomicRanges_OR_missing", directly
## Class "GenomicRanges_OR_GRangesList", directly
## Class "List", by class "Ranges", distance 2
## Class "Vector", by class "Ranges", distance 3
## Class "list_OR_List", by class "Ranges", distance 3
## Class "Annotated", by class "Ranges", distance 4
##
## Known Subclasses:
## Class "GenomicPos", directly
## Class "GRanges", directly
## Class "DelegatingGenomicRanges", directly
```

S4: virtual class example - `GenomicRanges`

- A virtual class of three virtual superclass.
- Widely-used subclasses in Bioconductor. Subclasses such as `GRanges` and `GPos` inherite the properties of `GenomicRanges`

```
selectMethod("names", "GRanges")
```

```
## Method Definition:
##
## function (x)
## names(ranges(x))
## <bytecode: 0x7fde2a702d60>
## <environment: namespace:GenomicRanges>
##
## Signatures:
##         x
## target  "GRanges"
## defined "GenomicRanges"
```

S4: class union

`setClassUnion()`: defined classes as the union of other classes

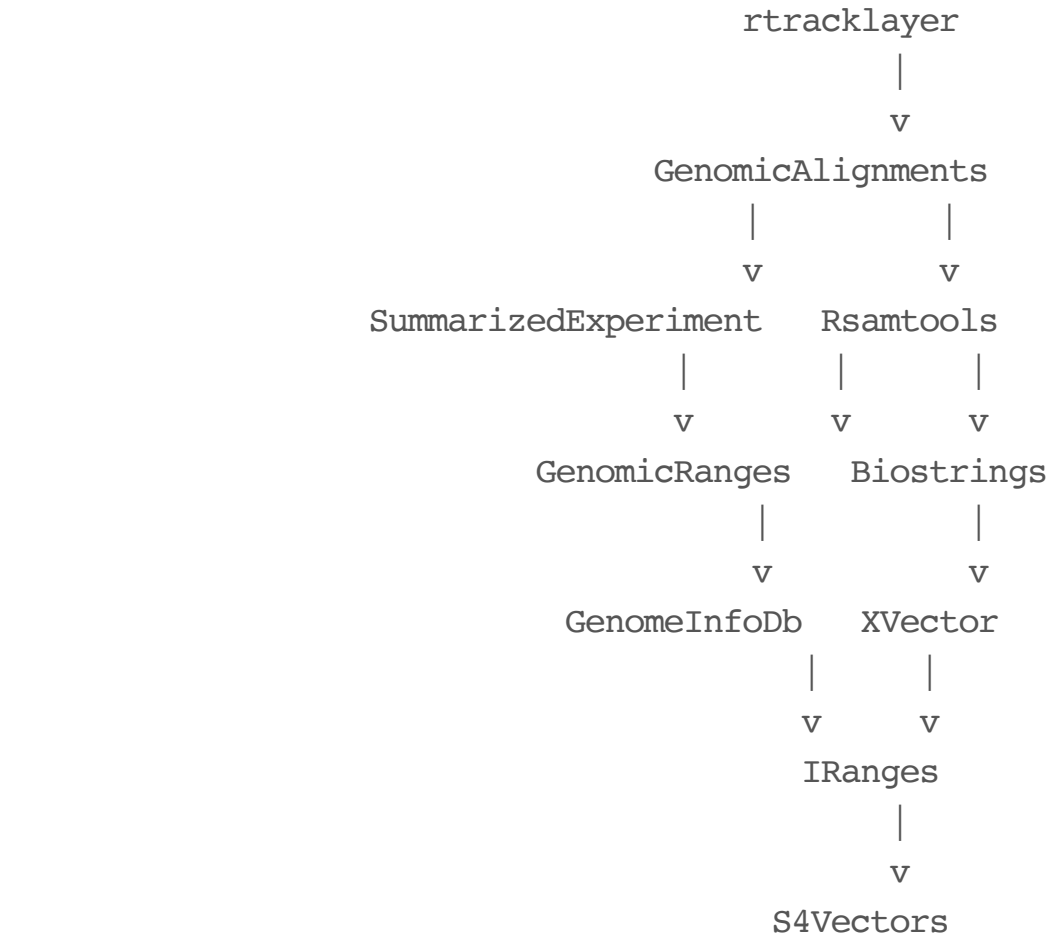
```
## a class for either numeric or logical data
setClassUnion("maybeNumber", c("numeric", "logical"))
```

Reference classes (R5)

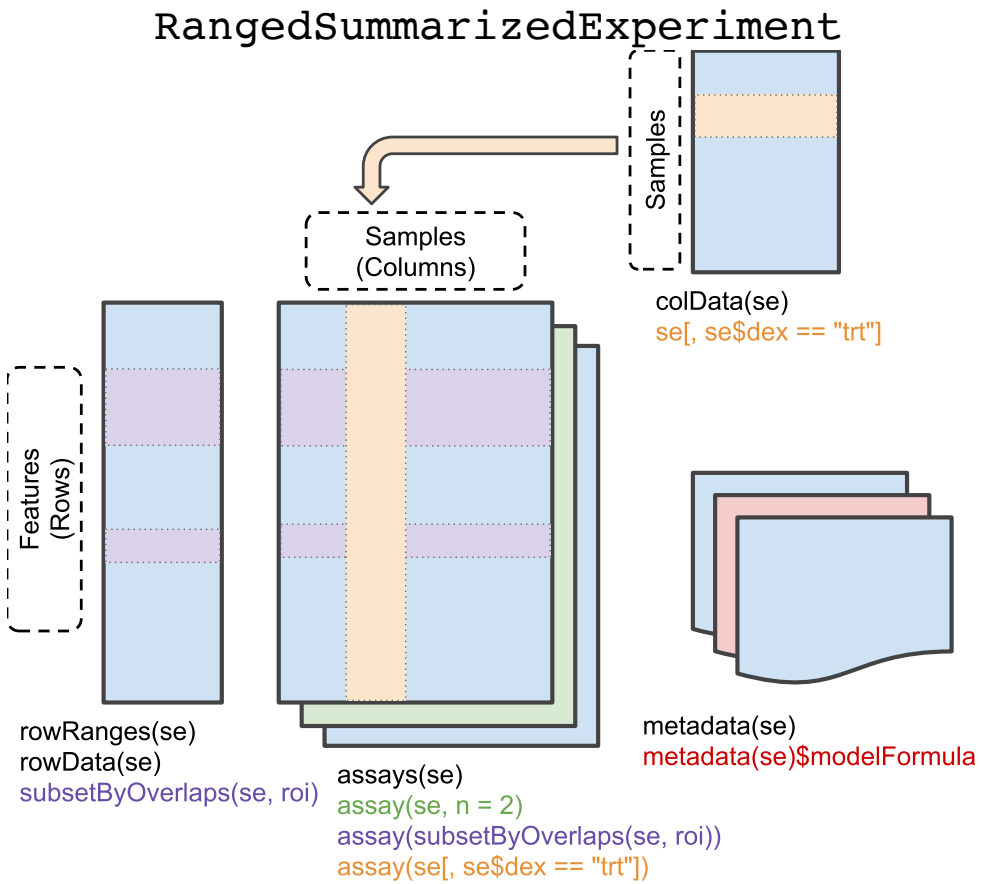
```
?ReferenceClasses
```

- appeared in R-2.21
- pass-by-reference semantic
- methods are encapsulated in the class definition
- John Chambers (2016), *Extending R*, Capman & Hall

Application - Bioconductor (S4Vectors)



Applications - Bioconductor



Applications - Bioconductor

The `airway` package contains an example dataset (`RangedSummarizedExperiment`) from an RNA-Seq experiment of read counts per gene for airway smooth muscles.

```
suppressPackageStartupMessages(library(SummarizedExperiment))
data(airway, package="airway")
se <- airway
se
```

```
## class: RangedSummarizedExperiment
## dim: 64102 8
## metadata(1): ''
## assays(1): counts
## rownames(64102): ENSG00000000003 ENSG00000000005 ... LRG_98 LRG_99
## rowData names(0):
## colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
## colData names(9): SampleName cell ... Sample BioSample
```

How to install Bioconductor packages

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("S4Vectors", version = "3.8")
```

Bioconductor packages used for this presentation

- `GenomicRanges`
- `IRanges`
- `S4Vectors`
- `SummarizedExperiment`
- `airway`

Exercise

Single nucleotide polymorphisms (SNPs) is a genetic variation in a single nucleotide.

1. Implement an S4 class for the SNP location with slots

- genome, character, (i.e., "hg38" for latest homo sapiens genome built)
- snpid, character vector indicating the ID of the snp (i.e., "rs_xxx")
- chrom, character vector indicating the chromsome (i.e., "ch1")
- pos, integer indicating the position in the chromsome

1. Set validation such that all slots have the some length
2. Define `length` method

Session info

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] SummarizedExperiment_1.10.1 DelayedArray_0.6.6
##  [3] BiocParallel_1.14.2         matrixStats_0.54.0
##  [5] Biobase_2.40.0              GenomicRanges_1.32.7
##  [7] GenomeInfoDb_1.16.0         IRanges_2.14.12
##  [9] S4Vectors_0.18.3            BiocGenerics_0.26.0
```

Reference

- The *R Programming fro Bioinformatics* by Robert Gentleman
- The *Writing R Extensions manual*
- Vignette of `S4Vectors` package: A quick overview of S4 class system by Herv´e Pag`es, https://bioconductor.org/packages/release/bioc/vignettes/S4Vectors/inst/doc/S4Qu
- Bioconductor training courses: https://master.bioconductor.org/help/course-materials/2017/Zurich/S4-classes-and-methods.html