

## A quick overview of the S4 class system

Hervé Pagès  
hpages.on.github@gmail.com

June 2016

What is S4?

S4 from an end-user point of view

Implementing an S4 class (in 4 slides)

Extending an existing class

What else?

# Outline

What is S4?

S4 from an end-user point of view

Implementing an S4 class (in 4 slides)

Extending an existing class

What else?

## The S4 class system

- ▶ The *S4 class system* is a set of facilities provided in R for OO programming.
- ▶ Implemented in the *methods* package.
- ▶ On a fresh *R* session:

```
> sessionInfo()
```

```
...
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets
```

```
[6] methods   base
```

- ▶ R also supports an older class system: the *S3 class system*.

# A different world

## The syntax

```
> foo(x, ...)
```

not:

```
> x.foo(...)
```

like in other OO programming languages.

## The central concepts

- ▶ The core components: *classes*<sup>1</sup>, *generic functions* and *methods*
- ▶ The glue: *method dispatch* (supports *simple* and *multiple* dispatch)

---

<sup>1</sup>also called *formal classes*, to distinguish them from the S3 classes aka *old style classes*

## The result

```
> ls('package:methods')
 [1] "addNextMethod"           "allGenerics"
 [3] "allNames"                "Arith"
 [5] "as"                      "as<-"
 [7] "asMethodDefinition"     "assignClassDef"
...
[211] "testVirtual"            "traceOff"
[213] "traceOn"                "tryNew"
[215] "unRematchDefinition"   "validObject"
[217] "validSlotNames"
```

- ▶ Rich, complex, can be intimidating
- ▶ The classes and methods we implement in our packages can be hard to document, especially when the class hierarchy is complicated and multiple dispatch is used

## S4 in Bioconductor

- ▶ Heavily used. In BioC 3.3: 3158 classes and 22511 methods defined in 609 packages! (out of 1211 software packages)
- ▶ Top 10: 128 classes in *ChemmineOB*, 98 in *flowCore*, 79 in *IRanges*, 68 in *rsbml*, 61 in *ShortRead*, 58 in *Biostrings*, 51 in *rtracklayer*, 50 in *oligoClasses*, 45 in *flowUtils*, and 40 in *BaseSpaceR*.
- ▶ For the end user: it's mostly transparent. But when something goes wrong, error messages issued by the S4 class system can be hard to understand. Also it can be hard to find the documentation for a specific method.
- ▶ Most Bioconductor packages use only a small subset of the S4 capabilities (covers 99.99% of our needs)

# Outline

What is S4?

S4 from an end-user point of view

Implementing an S4 class (in 4 slides)

Extending an existing class

What else?



## Where do S4 objects come from?

### From a dataset

```
> library(graph)
> data(apopGraph)
> apopGraph
```

A graphNEL graph with directed edges

Number of Nodes = 50

Number of Edges = 59

### From using an object constructor function

```
> library(IRanges)
> IRanges(start=c(101, 25), end=c(110, 80))
```

IRanges object with 2 ranges and 0 metadata columns:

	start	end	width
	<integer>	<integer>	<integer>
[1]	101	110	10
[2]	25	80	56

## From a coercion

```
> library(Matrix)
> m <- matrix(3:-4, nrow=2)
> as(m, "Matrix")

2 x 4 Matrix of class "dgeMatrix"
      [,1] [,2] [,3] [,4]
[1,]    3    1  -1  -3
[2,]    2    0  -2  -4
```

## From using a specialized high-level constructor

```
> library(GenomicFeatures)
> makeTxDbFromUCSC("sacCer2", tablename="ensGene")
```

TxDb object:

```
# Db type: TxDb
# Supporting package: GenomicFeatures
# Data source: UCSC
# Genome: sacCer2
# Organism: Saccharomyces cerevisiae
# Taxonomy ID: 4932
# UCSC Table: ensGene
# UCSC Track: Ensembl Genes
```

...

## From using a high-level I/O function

```
> library(ShortRead)
> path_to_my_data <- system.file(
+   package="ShortRead",
+   "extdata", "Data", "C1-36Firecrest", "Bustard", "GERALD")
> lane1 <- readFastq(path_to_my_data, pattern="s_1_sequence.txt")
> lane1

class: ShortReadQ
length: 256 reads; width: 36 cycles
```

## Inside another object

```
> sread(lane1)

DNAStrngSet object of length 256:
  width seq
 [1]    36 GGACTTTGTAGGATACCCTCGCTTTCCTTCTCCTGT
 [2]    36 GATTTCTTACCTATTAGTGGTTGAACAGCATCGGAC
 [3]    36 GCGGTGGTCTATAGTGTTATTAATATCAATTTGGGT
 [4]    36 GTTACCATGATGTTATTTCTTCATTTGGAGGTA AAA
 [5]    36 GTATGTTTCTCCTGCTTATCACCTTCTTGAAGGCTT
 ...    ...
[252]    36 GTTTAGATATGAGTCACATTTTGTTTCATGGTAGAGT
[253]    36 GTTTTACAGACACCTAAAGCTACATCGTCAACGTTA
[254]    36 GATGAACTAAGTCAACCTCAGCACTAACCTTGCGAG
```

## How to manipulate S4 objects?

### Low-level: getters and setters

```
> ir <- IRanges(start=c(101, 25), end=c(110, 80))
```

```
> width(ir)
```

```
[1] 10 56
```

```
> width(ir) <- width(ir) - 5
```

```
> ir
```

```
IRanges object with 2 ranges and 0 metadata columns:
```

	start	end	width
	<integer>	<integer>	<integer>
[1]	101	105	5
[2]	25	75	51

### High-level: plenty of specialized methods

```
> qa1 <- qa(lane1, lane="lane1")
```

```
> class(qa1)
```

```
[1] "ShortReadQQA"
```

```
attr(,"package")
```

```
[1] "ShortRead"
```

## How to find the right man page?

- ▶ `class?graphNEL` or equivalently `?`graphNEL-class`` for accessing the man page of a class
- ▶ `?qa` for accessing the man page of a generic function
- ▶ The man page for a generic might also document some or all of the methods for this generic. The *See Also:* section might give a clue. Also using `showMethods()` can be useful:

```
> showMethods("qa")
```

```
Function: qa (package ShortRead)
```

```
dirPath="ShortReadQ"
```

```
dirPath="SolexaPath"
```

```
dirPath="character"
```

```
dirPath="list"
```

- ▶ `?`qa,ShortReadQ-method`` to access the man page for a particular method (might be the same man page as for the generic)
- ▶ In doubt: `??qa` will search the man pages of all the installed packages and return the list of man pages that contain the string `qa`

## Inspecting objects and discovering methods

- ▶ `class()` and `showClass()`

```
> class(lane1)
```

```
[1] "ShortReadQ"
```

```
attr(,"package")
```

```
[1] "ShortRead"
```

```
> showClass("ShortReadQ")
```

```
Class "ShortReadQ" [package "ShortRead"]
```

```
Slots:
```

```
Name:      quality      sread      id
```

```
Class: QualityScore DNASTringSet BStringSet
```

```
Extends:
```

```
Class "ShortRead", directly
```

```
Class ".ShortReadBase", by class "ShortRead", distance 2
```

```
Known Subclasses: "AlignedRead"
```

- ▶ `str()` for compact display of the content of an object
- ▶ `showMethods()` to discover methods
- ▶ `selectMethod()` to see the code

# Outline

What is S4?

S4 from an end-user point of view

Implementing an S4 class (in 4 slides)

Extending an existing class

What else?

# Class definition and constructor

## Class definition

```
> setClass("SNPLocations",
+   slots=c(
+     genome="character", # a single string
+     snpid="character",  # a character vector of length N
+     chrom="character",  # a character vector of length N
+     pos="integer"       # an integer vector of length N
+   )
+ )
```

## Constructor

```
> SNPLocations <- function(genome, snpid, chrom, pos)
+   new("SNPLocations", genome=genome, snpid=snpid, chrom=chrom, pos=pos)
> snplocs <- SNPLocations("hg19",
+   c("rs0001", "rs0002"),
+   c("chr1", "chrX"),
+   c(224033L, 1266886L))
```



# Getters

## Defining the length method

```
> setMethod("length", "SNPLocations", function(x) length(x@snpid))  
> length(snplocs) # just testing  
[1] 2
```

## Defining the slot getters

```
> setGeneric("genome", function(x) standardGeneric("genome"))  
> setMethod("genome", "SNPLocations", function(x) x@genome)  
> setGeneric("snpid", function(x) standardGeneric("snpid"))  
> setMethod("snpid", "SNPLocations", function(x) x@snpid)  
> setGeneric("chrom", function(x) standardGeneric("chrom"))  
> setMethod("chrom", "SNPLocations", function(x) x@chrom)  
> setGeneric("pos", function(x) standardGeneric("pos"))  
> setMethod("pos", "SNPLocations", function(x) x@pos)  
> genome(snplocs) # just testing  
[1] "hg19"  
> snpid(snplocs) # just testing  
[1] "rs0001" "rs0002"
```

## Defining the `show` method

```
> setMethod("show", "SNPLocations",
+   function(object)
+     cat(class(object), "instance with", length(object),
+         "SNPs on genome", genome(object), "\n")
+ )
> snplocs # just testing
SNPLocations instance with 2 SNPs on genome hg19
```

## Defining the `validity` method

```
> setValidity("SNPLocations",
+   function(object) {
+     if (!is.character(genome(object)) ||
+         length(genome(object)) != 1 || is.na(genome(object)))
+       return("'genome' slot must be a single string")
+     slot_lengths <- c(length(snpid(object)),
+                       length(chrom(object)),
+                       length(pos(object)))
+     if (length(unique(slot_lengths)) != 1)
+       return("lengths of slots 'snpid', 'chrom' and 'pos' differ")
+     TRUE
+   }
+ )
> snplocs@chrom <- LETTERS[1:3] # a very bad idea!
> validObject(snplocs)
Error in validObject(snplocs) :
  invalid class "SNPLocations" object: lengths of slots 'snpid', 'chrom'
  and 'pos' differ
```

## Defining slot setters

```
> setGeneric("chrom<-", function(x, value) standardGeneric("chrom<-"))
> setReplaceMethod("chrom", "SNPLocations",
+   function(x, value) {x@chrom <- value; validObject(x); x})
> chrom(snplocs) <- LETTERS[1:2] # repair currently broken object
> chrom(snplocs) <- LETTERS[1:3] # try to break it again
Error in validObject(x) :
  invalid class "SNPLocations" object: lengths of slots 'snpid', 'chrom'
  and 'pos' differ
```

## Defining a coercion method

```
> setAs("SNPLocations", "data.frame",
+   function(from)
+     data.frame(snpid=snpid(from), chrom=chrom(from), pos=pos(from))
+ )
> as(snplocs, "data.frame") # testing
```

	snpid	chrom	pos
1	rs0001	A	224033
2	rs0002	B	1266886

# Outline

What is S4?

S4 from an end-user point of view

Implementing an S4 class (in 4 slides)

**Extending an existing class**

What else?

## Slot inheritance

- ▶ Most of the time (but not always), the child class will have additional slots:

```
> setClass("AnnotatedSNPs",  
+   contains="SNPLocations",  
+   slots=c(  
+     geneid="character" # a character vector of length N  
+   )  
+ )
```

- ▶ The slots from the parent class are inherited:

```
> showClass("AnnotatedSNPs")
```

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

```
Slots:
```

```
Name:   geneid   genome   snpid   chrom   pos  
Class: character character character character integer
```

```
Extends: "SNPLocations"
```

- ▶ Constructor:

```
> AnnotatedSNPs <- function(genome, snpid, chrom, pos, geneid)  
+ {  
+   new("AnnotatedSNPs",  
+     SNPLocations(genome, snpid, chrom, pos),  
+     geneid=geneid)  
+ }
```

## Method inheritance

- ▶ Let's create an AnnotatedSNPs object:

```
> snps <- AnnotatedSNPs("hg19",  
+                       c("rs0001", "rs0002"),  
+                       c("chr1", "chrX"),  
+                       c(224033L, 1266886L),  
+                       c("AAU1", "SXW-23"))
```

- ▶ All the methods defined for SNPLocations objects work out-of-the-box:

```
> snps  
AnnotatedSNPs instance with 2 SNPs on genome hg19
```

- ▶ But sometimes they don't do the right thing:

```
> as(snps, "data.frame") # the 'geneid' slot is ignored  
  snpid chrom    pos  
1 rs0001 chr1 224033  
2 rs0002 chrX 1266886
```

- ▶ Being a `SNPLocations` *object* vs being a `SNPLocations` *instance*:

```
> is(snps, "AnnotatedSNPs")      # 'snps' is an AnnotatedSNPs object
[1] TRUE
> is(snps, "SNPLocations")      # and is also a SNPLocations object
[1] TRUE
> class(snps)                   # but is not a SNPLocations instance
[1] "AnnotatedSNPs"
attr(,"package")
[1] ".GlobalEnv"
```

- ▶ Method overriding: for example we could define a `show` method for `AnnotatedSNPs` objects. `callNextMethod` can be used in that context to call the method defined for the parent class from within the method for the child class.

- ▶ Automatic coercion method:

```
> as(snps, "SNPLocations")
SNPLocations instance with 2 SNPs on genome hg19
```

## Incremental validity method

- ▶ The *validity method* for AnnotatedSNPs objects only needs to validate what's not already validated by the *validity method* for SNPLocations objects:

```
> setValidity("AnnotatedSNPs",  
+   function(object) {  
+     if (length(object@geneid) != length(object))  
+       return("'geneid' slot must have the length of the object")  
+     TRUE  
+   }  
+ )
```

- ▶ In other words: before an AnnotatedSNPs object can be considered valid, it must first be a valid SNPLocations object.



# Outline

What is S4?

S4 from an end-user point of view

Implementing an S4 class (in 4 slides)

Extending an existing class

What else?

## Other important S4 features

- ▶ *Virtual* classes: equivalent to *abstract* classes in Java
- ▶ Class unions (see `?setClassUnion`)
- ▶ Multiple inheritance: a powerful feature that should be used with caution. If used inappropriately, can lead to a class hierarchy that is very hard to maintain

## Resources

- ▶ Man pages in the *methods* package: `?setClass`, `?showMethods`, `?selectMethod`, `?getMethod`, `?is`, `?setValidity`, `?as`
- ▶ The *Extending RangedSummarizedExperiment* section of the *SummarizedExperiment* vignette in the *SummarizedExperiment* package.
- ▶ Note: S4 is *not* covered in the *An Introduction to R* or *The R language definition* manuals<sup>2</sup>
- ▶ The *Writing R Extensions* manual for details about integrating S4 classes to a package
- ▶ The *R Programming for Bioinformatics* book by Robert Gentleman<sup>3</sup>

---

<sup>2</sup><http://cran.fhcrc.org/manuals.html>

<sup>3</sup><http://bioconductor.org/help/publications/books/r-programming-for-bioinformatics/>