10. Introduction to the S4 Object System

CT5102 - J. Duggan

S4 (Wickam 2019, Chapter 15)

- A more formal approach to functional OOP
- Underlying ideas similar to S3, but implementation is much stricter
- Makes use of specialised functions for:
  - Creating classes **setClass()**
  - Generics **setGeneric()**
  - Methods **setMethod()**
- Provides multiple inheritance and multiple dispatch
- A new component of S4 is the slot, a named component of an object that can be accessed using @ (pronounced at)
- Include **library(methods)** when using S4

Basics

```
setClass("Person",
         slots = c(
           name = "character",
           age  = "numeric"
         )
)

john <- new("Person", name = "John Smith", age=35)
```

Given an S4 object, you can see its class with **is()** and access its slots with **@** and **slot()**

Calling S4 functions

```
is(john)
```

```
## [1] "Person"
```

```
john@name
```

```
## [1] "John Smith"
```

```
john@age
```

```
## [1] 35
```

```
slot(john,"age")
```

```
## [1] 35
```

```
slot(john,"name")
```

```
## [1] "John Smith"
```

Accessing slots: Guidelines

- Generally, only use @ in your methods
- Look for accessor functions that allow you to safely set and get slot values
- When you develop a class, provide your own access functions
- Creating a setter and getter for the age slot by
  - Creating getter and setter generics usiing **setGeneric()**
  - Defining methods with **setMethod()**

getter for slot age

```
setGeneric("age", function(x)standardGeneric("age")) # getter
```

```
## [1] "age"
```

```
setMethod("age", "Person", function(x) x@age)

age(john)
```

```
## [1] 35
```

setter for slot age

```
setGeneric("age<-", function(x, value)standardGeneric("age<-")
```

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

```
setMethod("age<-", "Person", function(x, value){
  x@age <- value
  x
})
age(john) <- 29
age(john)
```

```
## [1] 29
```

Creating S4 Classes

- To define an S4 class, call setClass() with three arguments
  - The class **name**. By convention S4 class names use UpperCamelCase
  - A named character vector that describes the names and classes of the **slots** (fields). The pseudo-class ANY allows a slot to accept objects of any type
  - A **prototype**, a list of default values for each type (optional but should be provided)

S4 class with 3 arguments

```
setClass("Person",
         slots = c(
           name = "character",
           age  = "numeric"
         ),
         prototype=list(
           name = NA_character_,
           age  = NA_real_
         )
)

test <- new("Person", name = "A.N. Other")
str(test)
```

```
## Formal class 'Person' [package ".GlobalEnv"] with 2 slots
##   ..@ name: chr "A.N. Other"
##   ..@ age : num NA
```

S4 class with inheritance

```
setClass("Employee",
         contains = "Person",
         slots = c(
           boss = "Person"
         ),
         prototype=list(
           boss = new("Person")
         )
)
str(new("Employee"))
```

```
## Formal class 'Employee' [package ".GlobalEnv"] with 3 slots
##   ..@ boss:Formal class 'Person' [package ".GlobalEnv"] wit
##   .. .. ..@ name: chr NA
##   .. .. ..@ age : num NA
##   ..@ name: chr NA
```

Helper

User facing classes should always be paired with a user-friendly helper. A helper should always:

- Have the same name as the class
- Have a thoughtfully crafted user interface with carefully chosen default values
- Create error messages tailored towards an end user
- Finish by calling **methods::new()**

Example

```
Person <- function(name, age=NA){
  age <- as.double(age)
  new("Person", name=name, age=age)
}

p1 <- Person("A.N. Other")
p1
```

```
## An object of class "Person"
## Slot "name":
## [1] "A.N. Other"
##
## Slot "age":
## [1] NA
```

Validator

- The constructor automatically checks that the slots have correct classes
- More complicated checks may be required
- For example, name and age may have to be the same vector length
- We can write a validator with **setValidity()**

```
setValidity("Person", function(object) {
  if(length(object@name) != length(object@age)){
    "@name and @age must be the same length"
  }else{
    TRUE
  }
})
```

Generics and Methods

- The job of a generic is to perform method dispatch (find the specific implementation for the combination of classes passed to the generic)
- To create a new S4 generic, call **setGeneric()** with a function that calls **stanardGeneric()**
- By convention, new S4 generics should use lowerCamelCase.
- It is bad practice to use {} as it triggers a special case that is more computationally expensive

```
setGeneric("myGeneric",
           function(x) standardGeneric("myGeneric"))
```

Generics and Methods

- A generic isn't useful without some methods, and in S4 methods are defined with **setMethod()**
- There are three important arguments:
  - The name of the generic
  - The name of the class
  - The method itself

```
setMethod("myGeneric", "Person", function(x){
  # method implementation
})
```

Show method

- The most commonly defined S4 method that controls printing is **show()**.
- To define a method for an existing generic, you must first determine its arguments
- Our show method needs to have a single argument object

```
args(getGeneric("show"))
```

```
## function (object)
## NULL
```

Writing the method

```
setMethod("show", "Person", function (object){
  cat(is(object)[[1]], "\n",
  "    Name: ", object@name, "\n",
  "    Age:  ", object@age, "\n",
  sep="")
})

p1
```

```
## Person
##     Name: A.N. Other
##     Age:  NA
```

Accessors - getter function

- Slots should be considered an internal implementation detail: user code should avoid accessing them directly
- Typically, a generic will be defined so that multiple classes can use the same interface

```
setGeneric("name", function(x) standardGeneric("name"))
```

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

```
setMethod("name", "Person", function(x) x@name)

name(p1)
```

```
## [1] "A.N. Other"
```

Accessors - setter function

- If the slot is writeable, a setter function should be provided
- Should also include **validObject()** in the setter

```
setGeneric("name<-", function(x, value)
             standardGeneric("name<-"))
```

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

```
setMethod("name<-", "Person", function(x, value){
  x@name <- value
  x})

name(p1) <- "Test Name"
name(p1)
```

```
## [1] "Test Name"
```

Exercises and Further features of S4

- Exercises
  - Add age() accessors for the Person class
  - In ther definition of the generic, why is it necessary to repeat the name of the generic twice
- Method Dispatch
  - Multiple inheritance - a class can have multiple parents
  - Multiple dispatch - a generic can use multiple arguments to pick a method

Examples of S4 - Mapping functions

- The **sp** package defines a number of S4 classes for handling spatial data: points, lines and areas (Brunsdon and Comber 2019)
- The package **GISTools** contains useful data sets

Without Attributes	With Attributes	ArcGIS equivalent
SpatialPoints	SpatialPointsDataFrame	Point shapefiles
SpatialLines	SpatialLinesDataFrame	Line Shapefiles
SpatialPolygons	SpatialPolygonsDataFrame	Polygon shapefiles

Georgia Data Set

```
library(GISTools)
library(sp)
library(tmap)

data(georgia)
class(georgia)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
getSlots("SpatialPolygonsDataFrame")
```

```
##          data      polygons     plotOrder          bbox  proj4s
## "data.frame"        "list"     "integer"      "matrix"
```

Exploring the data (1st county of 159)

```
length(georgia@polygons)
```

```
## [1] 159
```

```
str(georgia@polygons[[1]])
```

```
## Formal class 'Polygons' [package "sp"] with 5 slots
##   ..@ Polygons :List of 1
##   .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slot
##   .. .. .. ..@ labpt  : num [1:2] -82.3 31.7
##   .. .. .. ..@ area   : num 0.126
##   .. .. .. ..@ hole   : logi FALSE
##   .. .. .. ..@ ringDir: int 1
##   .. .. .. ..@ coords : num [1:125, 1:2] -82.2 -82.2 -82.2
##   ..@ plotOrder: int 1
##   ..@ labpt    : num [1:2] -82.3 31.7
##   ..@ ID       : chr "0"
```

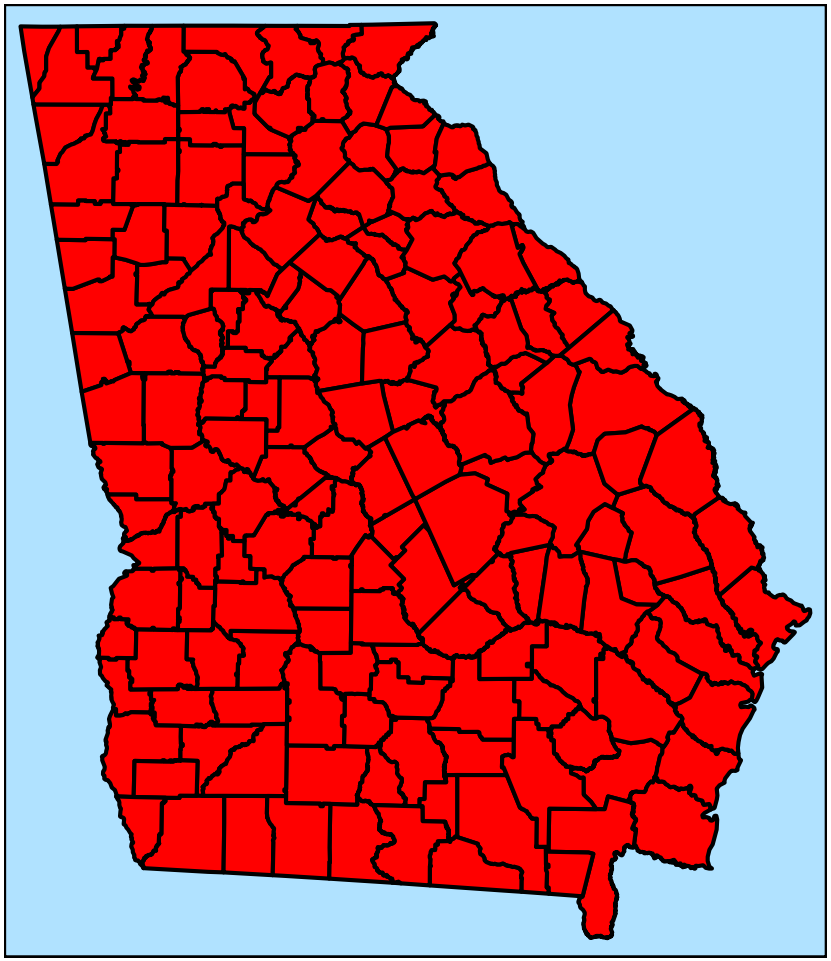
Getting the data

```
head(georgia@data)
```

```
##    Latitude  Longitud TotPop90 PctRural PctBach PctEld PctFE
## 0 31.75339 -82.28558    15744      75.6      8.2  11.43  0.64
## 1 31.29486 -82.87474     6213     100.0      6.4  11.77  1.58
## 2 31.55678 -82.45115     9566      61.7      6.6  11.11  0.27
## 3 31.33084 -84.45401     3615     100.0      9.4  13.17  0.11
## 4 33.07193 -83.25085    39530      42.7     13.3   8.64  1.43
## 5 34.35270 -83.50054    10308     100.0      6.4  11.37  0.34
##    PctBlack         X       Y    ID      Name MedInc
## 0     20.76 941396.6 3521764 13001   Appling  32152
## 1     26.86 895553.0 3471916 13003 Atkinson  27657
## 2     15.42 930946.4 3502787 13005     Bacon  29342
## 3     51.67 745398.6 3474765 13007     Baker  29610
## 4     42.39 849431.3 3665553 13009  Baldwin  36414
## 5      3.49 819317.3 3807616 13011     Banks  41783
```

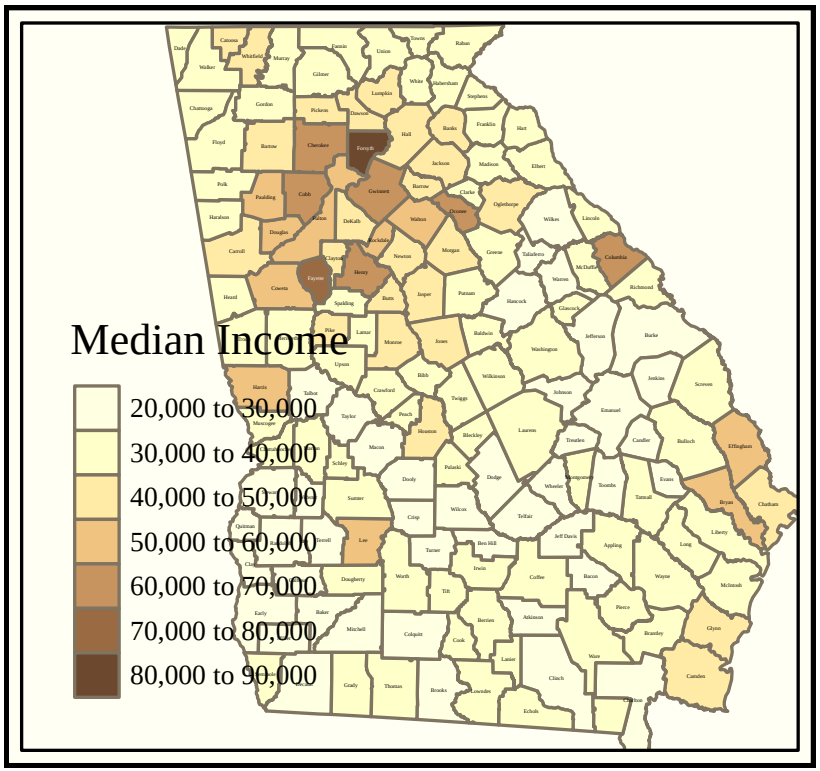
Displaying the Map

```
qtm(georgia,fill="red",style="natural")
```



Filling by attribute

```
qtm(georgia,fill="MedInc",text="Name",text.size=0.1,
    format="World_wide",style="classic",
    text.root=5,fill.title="Median Income")
```



New sf package

```
library(GISTools)
library(sp)
library(sf)
library(tmap)

data(georgia)
georgia_sf <- st_as_sf(georgia)
class(georgia_sf)
```

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

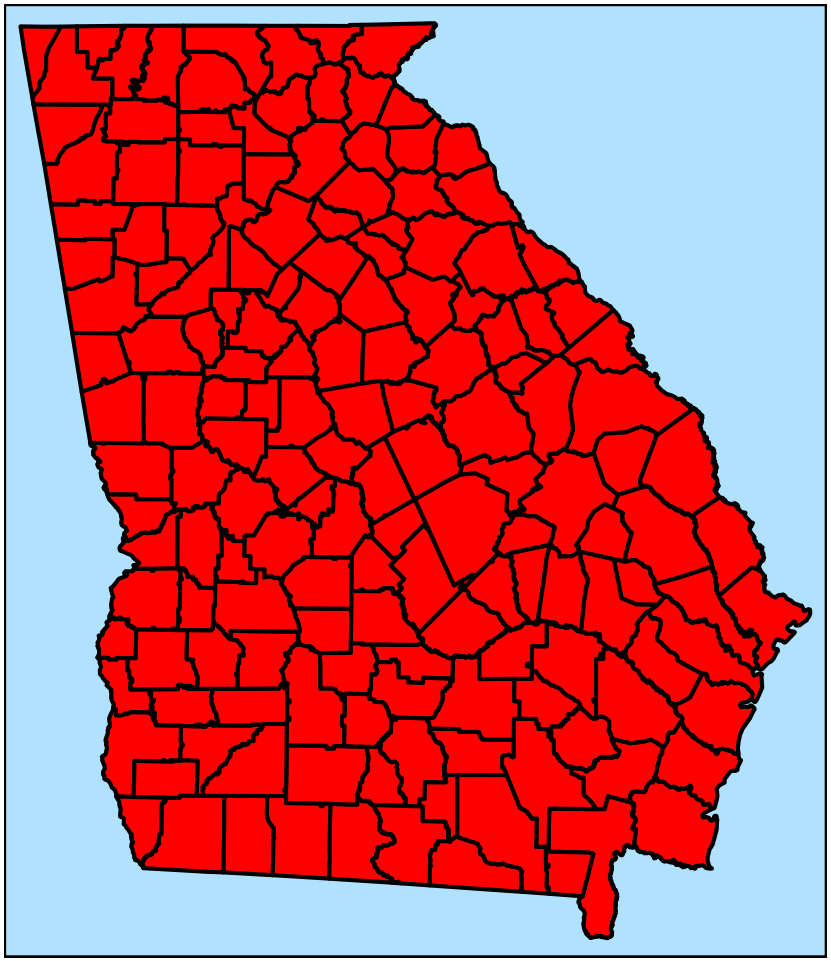
New sf package

```
georgia_sf[1:2,]
```

```
## Simple feature collection with 2 features and 14 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -83.14098 ymin: 31.18356 xmax: -82.04
## epsg (SRID):    4326
## proj4string:    +proj=longlat +ellps=WGS84 +no_defs
##   Latitude  Longitud TotPop90 PctRural PctBach PctEld PctFE
## 0 31.75339 -82.28558    15744     75.6     8.2  11.43  0.64
## 1 31.29486 -82.87474     6213    100.0     6.4  11.77  1.58
##   PctBlack        X       Y    ID     Name MedInc
## 0    20.76 941396.6 3521764 13001  Appling  32152
## 1    26.86 895553.0 3471916 13003 Atkinson  27657
##                         geometry
## 0 MULTIPOLYGON (((-82.2252 31...
## 1 MULTIPOLYGON (((-82.6202 31...
```

Displaying the Map (S3 class)

```
qtm(georgia_sf,fill="red",style="natural")
```



Summary

- S4 is a more formal approach to functional OOP
- Underlying ideas similar to S3, but implementation is much stricter
- Include **library(methods)** when using S4
- Popular packages
  - **sp** for maps (**sf** uses **sp**)
  - Bioconductor (R libraries for BioInformatics)