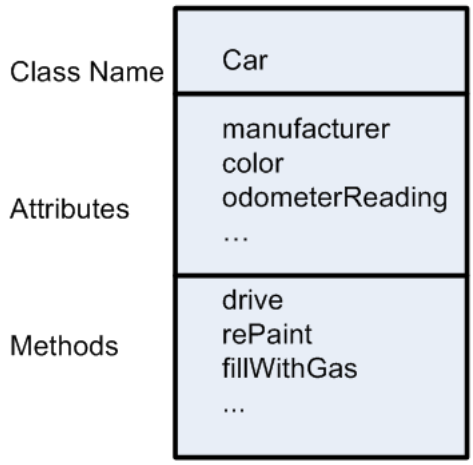
The Basics of Class Systems in R

Introduction

Since R is an object oriented programming language, users may create different types of classes. *Classes* are definitions of objects and they are a central part of any object oriented programming language. Within a class, an object's behavior, attributes, and relationship to other objects is defined. Moreover, *methods* are functions which are intended to be used for specific objects and act in accordance to the object's class. Classes can also have *parent classes* which allow the *child classes* to *inherit* things like methods from the parent for use of objects within the child class. Currently, R offers three main types of class systems which are called S3, S4, and R6. This post will give a general overview of each of the three systems.

Class, Attribute, and Method Example



This is visual example of a Car class with the attributes manufacturer, color, and odometerReading. It has the methods drive, rePaint, and fillWithGas.

Class Hierarchy Example

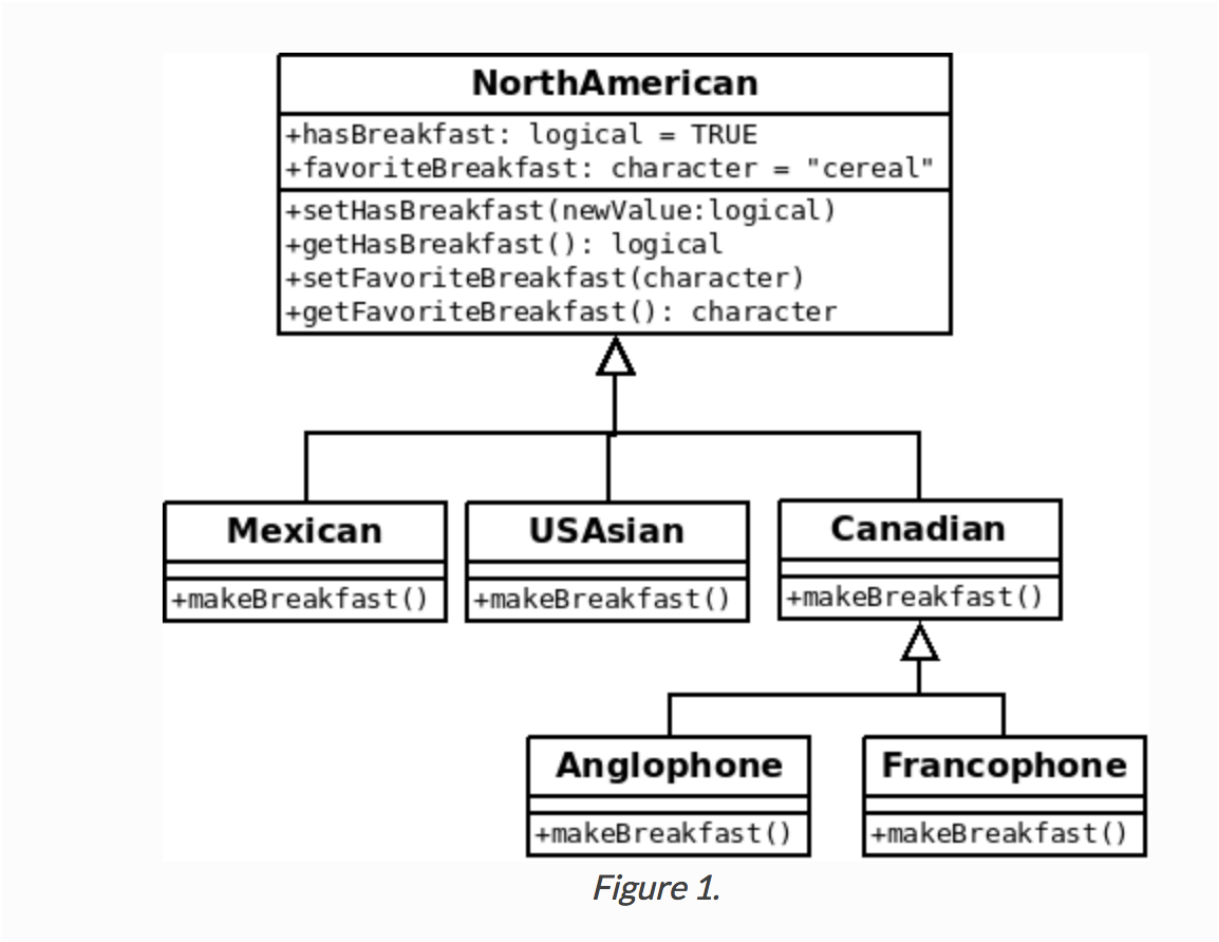


Figure 1.

This is an example of the hierarchical structure of classes. In this example, NorthAmerican is the parent class of Mexican, USAsian, and Canadian.

The S3 Class System

The first of these systems is the S3 object system. This system utilizes *generic functions* when deciding which method to call. The following code is an example of an instantiation of an object x of class basic.

```
x <- structure(3, class = "basic")
x
```

```
## [1] 3
## attr(,"class")
## [1] "basic"
```

In S3, the generic function is implemented as follows with the UseMethod function. UseMethod takes in the name of the function (which is foo in this case) and the object as inputs in order to indicate that the function should be carried out on the object.

```
foo <- function (x, ...) {
   UseMethod("foo", x)
 }
```

The S4 Class System

The second system is the S4 class system which is similar to the S3 system besides a few main differences. Two major differences are that the class definition of S4 is more formal and generic functions in S4 can take in multiple inputs.

Rather than simply indicating the class of an object like in S3, the user must use the setClass function. SetClass takes in the arguments name, representation, and contains. Name is the string which indicates the class, representation is a list of attributes with their corresponding classes, and contains is a character vector of the classes which the class indicated by name inherits from. Below is an example of the instatiation of a class Person with S4.

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

```
To instantiate a generic function in S4, one must use the setGeneric function which takes in a function as an inpu
t. The following code demonstrates how to set a previously defined function as a generic function.
```

```
foo <- function(object)
setGeneric("foo")
```

The following code shows how to use the standardGeneric function to create a new generic function without a previously defined function.

```
setGeneric("foo", function(object) {
  standardGeneric("foo")
})
```

```
## [1] "foo"
```

In order to define a method for a class in S4, one must use the setMethod function which takes in the generic function's name, the signature of the method, and a function to output the final result. The *signature* is used to determine whether the class of the objects in the call and the class of the method itself are of the same class or whether the hierarchical distance between the classes must be computed. The following code demonstrates the use of setMethod.

`setMethod("foo", signature(object = "Square"), function(object) { object@foo })`

The R6 Class System

Lastly, the R6 class system is a package which can be installed like other R packages using install.packages('R6') and loaded using libarary('R6'). In R6, a class can be created using the R6Class function which has the documentation as follows.

```
R6Class(classname = NULL, public = list(), private = NULL,
active = NULL, inherit = NULL, lock_objects = TRUE, class = TRUE,
portable = TRUE, lock_class = FALSE, cloneable = TRUE, parent_env = parent.frame(), lock_objects)
```

One of the main differences between R6 and the other class systems is that R6 supports the use of *private* and *public* methods. Private methods may only be used within the declared class while public methods may be used outside of the declared class. Thus, the public argument of R6Class takes in a list of both functions and fields which are members of the class while private can take in any private members.

Once the class has been defined using R6Class, an object can be assigned to the class by using $new() which takes in arguments to be used as inputs for the initialize method if it exists. Below is an example of the creation of the class Dog and the instantiation of the object Spot with the field fur_color assigned to brown and the field size assigned to small.

```
library('R6')

Dog <- R6Class("Dog",
  public = list(
    fur_color = NULL,
    size = NULL,
    initialize = function(fur_color = NA, size = NA) {
      self$fur_color <- fur_color
      self$size <- size
}
  )
)
Spot <- Dog$new("brown", "small")
Spot
```

```
## <Dog>
##   Public:
##     clone: function (deep = FALSE)
##     fur_color: brown
##     initialize: function (fur_color = NA, size = NA)
##     size: small
```

To access values or methods assigned to the object, one may use `$`. For example, to access the fur_color value of Spot, one would use `Spot$fur_color` to output "brown." If any private members are defined, one may access these members with the `private$` command.

Conclusion

It is important to understand the different types of class systems within R because each of them has their own strengths and weaknesses and will provide different benefits to one's work in R. For creating most classes and methods, S3 is a suitable class system to utilize because it requires the least amount of code and can handle simple tasks. For creating more complex objects and methods, one might use S4 because of its strict and precise nature. For more efficiency of code, however, one might decide to use R6 instead of S4.

Sources

1. http://blog.revolutionanalytics.com/2017/07/the-r6-class-system.html
2. http://adv-r.had.co.nz/S3.html
3. http://adv-r.had.co.nz/S4.html
4. https://cran.r-project.org/web/packages/R6/vignettes/Introduction.html
5. https://www.rdocumentation.org/packages/R6/versions/2.2.2
6. http://adv-r.had.co.nz/OO-essentials.html
7. https://cran.r-project.org/web/packages/R6/R6.pdf