To begin with, it is vital to remind of how objects in R are assigned in order to understand the aspect of object-oriented within the R language, users can assign a value to an object in R by simply using the left arrow sign <-, an example below:

a1 <- 10

Furthermore, by using the c function (short for combine), R would allow the users to combine different attributes into an object. It is made possible by implementing a set of attributes as parameters of the function. For example, in the case of an object that contains only a set of numbers, it would be as followed:

a2 <- c(100,200,33)

To get an object that is more “meaningful”, namely with an example of an Amazon customer’s profile, one can implement the small objects first and then combine it as follow:

user\_id <- “110011”

user\_rank <- “platinum”

user\_country <- “Vietnam”

user\_transaction <- “100”

userInfo <- c(user\_id, user\_rank, user\_country, user\_transaction)

Within the example code, you can find the example output when printing out the userInfo object. Beside the c function, one can also use the list() function where users also can list parameters into the bracket and parameters can be defined directly, for example:

userInfo <- list(user\_id = “110011”, user\_rank = “platinum”, user\_country = “Vietnam”, user\_transaction = “100)

*Adapted from Datacamp:* [*https://www.datacamp.com/community/tutorials/r-objects-and-classes*](https://www.datacamp.com/community/tutorials/r-objects-and-classes) *(“R Classes & Objects with S3 & S4” 2020)*

Object classes within R can be constructed using three different ways, in which each way specify how each object functions can be constructed. These three implementations include(“Classes in R Programming” 2020):

1. S3 Class
2. S4 Class
3. Reference classes

S3 CLASS:

Within S3 class, the class will be constructed based on the combined object by naming the object within a class() function. For example, based on the userInfo object has been mentioned above, I can assign a name to the object and make it a class by writing (“Classes in R Programming” 2020):

class(userInfo) <- “user”

Within S3 class, the function will be defined as a combination of two components: a generic method “dot” a class. As for the generic method, it can be methods that users commonly use in other language like *print* or it can be identified independently by the user. The below example is the case where users can create their own version of the generic methods (“R Classes & Objects with S3 & S4” 2020):

retrieve <- function(object){

UseMethod(“retrieve”)

}

The UseMethod is built-in within R and it is used to announce “retrieve” would be generic. These generic methods are based on the implementation of input parameters, the function will be different - indicating how R overloads methods. For example, one can define S3 class as followed (“R Classes & Objects with S3 & S4” 2020):

retrieve.user\_id <- function(object){

cat(“User ID to track is:”, object$user\_id)

}

retrieve(userInfo)

R will look for the class that has a method taking in one object – thus, it will be the case of retrieve.user\_id. However, if the input *object* of the function is a combination (for example, by the c(“foo”, “bar) of more than 1 parameter, R will try out each of them following the input parameter to see which one fits with a method’s parameters description. In the case above, R searches for a function that takes in one object – which would be retrieve.user\_id coe to run.(“S3 · Advanced R.” n.d.)

Within the example code, you can find the example output when running the defined function retrieve(userInfo)

By the way it’s constructed, S3 is fairly flexible and does not have a consolidated definition scope. For its inheritance aspect, with S3 class, only the method from base case is inherited. For example, provided a new object of the following list (“R - Inheritance” 2020):

newObj <- list(user\_id = “110011”, user\_rank = “platinum”, user\_country = “Vietnam”, user\_transaction = “100”, complaint =”0”)

class(newObj) <- (newClassName, userInfo)

retrieve(newObj)

The result will only print out the same output because first, the newClassName is not a defined method and that only retrieve.user\_id (with a set of 4 attributes) would satisfy the output result.

S4 CLASS:

S4 class will be defined completely different from that of S3 class, in which it would be more similar towards Java or Python in terms of structure. The generic way to define a S4 class object would be using the function setClass within the following structure (“R Classes & Objects with S3 & S4” 2020):

setClass(“Class Name”, slots=list( attr1 = “Character”, …) ) – The … indicating other attributes could be listed within the class.

And as users want to construct an object based on this class, they can implement:

obj <- new(“Class Name”, list( attr1 = x, …))

To create a method in S4 class, users still have to follow 2 steps where they call out the generic method first using setGeneric and then implement the method they want to define using SetMethod. For example:

setGeneric("getID", function(object) {

standardGeneric("getID")

})

setMethod("getID", "user",function(object) cat("User ID to track is (S4):", object@user\_id))

getID(newUser)

As for S4 Class, attributes are identified strictly with class types and the number of attributes is clearly declared as the class is constructed. Thus, it should be expected that the objects of this class would be following a specific convention and more difficult to be modified compared to the case of S3 class objects. (“R Classes & Objects with S3 & S4” 2020) Furthermore, because of a well-declared scope when the class is constructed, a base class object’s attribute and method could be inherited.

Examples of these class constructors and implementation can be found in the sample code

Different from the case of class S3, in S4, attributes of an object can be called out using a @ sign rather than a $ sign. And to modify these attributes, users again only need to use the <- (left arrow) to assign new value to the target attributes. (“R Classes & Objects with S3 & S4” 2020)

Reference Class:

Reference class constructors and implementation would be similar to that of the S4 class, however there are several differences need to be taken into account (“Classes in R Programming” 2020):

* Replace setClass with setRefClass and replacing “slots” with “fields”
* Inside the list specified when the class is constructed, function of the object classes now can be included in the list as attributes. In this way, when users want to implement the listed functions in the class, users will call it using a $ sign as if they are calling out an attribute name. For example, UserClass$callID(1) – in which callID(x) is a method of a class called UserClass. This also indicates a difference with S4 since S4 calls attributes out using a @ sign instead of a $ sign.

Bibliography

“Classes in R Programming.” 2020. *GeeksforGeeks* (blog). April 9, 2020. https://www.geeksforgeeks.org/classes-in-r-programming/.

“R - Inheritance.” 2020. *GeeksforGeeks* (blog). June 17, 2020. https://www.geeksforgeeks.org/r-inheritance/.

“R Classes & Objects with S3 & S4.” 2020. DataCamp Community. June 22, 2020. https://www.datacamp.com/community/tutorials/r-objects-and-classes.

“S3 · Advanced R.” n.d. Accessed April 25, 2022. http://adv-r.had.co.nz/S3.html.