Chapter 14: R6

Ryan Heslin

May 30, 2022

All classes are created using `R6Class("classname", list(...)`, where the second argument is a list of methods and attributes.

Classes and Methods

```
library(tidyverse)
library(R6)
Accumulator <- R6Class("Accumulator", list(
  sum = 0,
  add = function(x = 1) {
    self$sum <- self$sum + x
    invisible(self)
  }
))

x <- Accumulator$new()
x$add()
x
```

```
<Accumulator>
  Public:
    add: function (x = 1)
    clone: function (deep = FALSE)
    sum: 1
```

Fields and methods can be made public.

Side-effect methods should return, invisibly, `self`, enabling chaining.

```
x$add()$add(10)
x
```

```
<Accumulator>
  Public:
    add: function (x = 1)
    clone: function (deep = FALSE)
    sum: 12
```

Classes should generally have `initialize` and `print` methods defined.

Modify with `set`, though existing objects are not updated.

```
Accumulator$set("public", "displ", function(...) {
  cat("An Accumulator\n")
  cat(paste("Value:", self$sum))
  invisible(self)
})
x <- Accumulator$new()
x$displ()
```

```
An Accumulator
Value: 0
```

Methods are instance methods, not class.

To use inheritance, override methods or use `super$method` to refer to superclass.

```
AccumulatorChatty <- R6Class("AccumulatorChatty",
  inherit = Accumulator,
  public = list(
    add = function(x = 1) {
      cat("Adding ", x, "\n", sep = "")
      super$add(x = x)
    }
  )
)

x2 <- AccumulatorChatty$new()
x2$add(10)$add(1)$sum
```

```
Adding 10
Adding 1
```

```
[1] 11
```

```
#> Adding 10
#> Adding 1
#> [1] 11k
```

List all methods and fields with `names`.

1, 2.

```
names(x)
```

```
[1] ".__enclos_env__" "sum"
[3] "displ"           "clone"
[5] "add"
```

```
Account <- R6Class("Account", list(
  balance = 0,
  deposit = function(amount) {
    self$balance <- self$balance + amount
    cat(sprintf("Deposited $%.2f in account", amount), sep = "\n")
    invisible(self)
  },
  withdraw = function(amount) {
    self$balance <- self$balance - amount
    cat(sprintf("Withdrew $%.2f from account", amount), sep = "\n")
    invisible(self)
  }
))
acc <- Account$new()
acc$deposit(50)
```

```
Deposited $50.00 in account
```

```
acc$withdraw(25)
```

```
Withdrew $25.00 from account
```

```
Account2 <- R6Class("Account2",
  inherit = Account,
  list(
    withdraw = function(amount) {
      stopifnot("Overdraft" = amount <= self$balance)
      self$balance <- self$balance - amount
```

```
      cat(sprintf("Withdrew $%.2f from account", amount), sep = "\n")
      invisible(self)
    }
  )
)
acc2 <- Account2$new()
tryCatch(acc2$withdraw(50), error = function(e) print(e))
```

```
<simpleError in acc2$withdraw(50): Overdraft>
```

```
Account3 <- R6Class("Account",
  inherit = Account,
  list(
    withdraw = function(amount) {
      if (amount > self$balance) {
        cat("Overdraft", sep = "\n")
        self$balance <- self$balance - 25
      }

      self$balance <- self$balance - amount
      cat(sprintf("Withdrew $%.2f from account", amount), sep = "\n")
      invisible(self)
    }
  )
)
acc3 <- Account3$new()
acc3$withdraw(50)
```

```
Overdraft
Withdrew $50.00 from account
```

3.

How would you represent the bound methods? I don't see a way absent serious hacking.

4.

```
Deck <- R6Class("Deck", list(
  cards = NULL,
  initialize = function() {
    suit <- c("spades", "hearts", "diamonds", "clubs")
    value <- c("A", 2:10, "J", "Q", "K")
```

```
    self$cards <- paste0(rep(value, 4), suit)
    return(invisible(self))
  },
  draw = function(n) {
    stopifnot(n <= length(self$cards))
    to_draw <- sample(seq_along(self$cards), n, replace = FALSE)
    cards <- self$cards[to_draw]
    self$cards <- self$cards[-to_draw]
    cards
  },
  shuffle = function() {
    self$initialize()
    self$cards <- self$cards[sample(length(self$cards), length(self$cards),
      replace = FALSE
    )]
    invisible(self)
  }
))
deck <- Deck$new()$shuffle()
deck$draw(5)
```

```
[1] "4hearts"   "Aclubs"    "Kdiamonds"
[4] "Kclubs"    "3diamonds"
```

```
deck$shuffle()
```

5.

```
Timezone <- R6Class("Timezone", list(
  get = function() Sys.timezone(),
  set = function(new_ts) {
    stopifnot(new_ts %in% OlsonNames())
    Sys.setenv(TZ = new_tz)
  }
))
tz <- Timezone$new()

old <- getwd()
WD <- R6Class("WD", list(
  get = function() normalizePath(getwd()),
  set = function(new_wd) {
    new_wd <- normalizePath(new_wd)
    tryCatch(setwd(new_wd),
      error = function(e) {
        cat(new_wd, "is not a valid directory", sep = "\n")
      }
    )
```

```
  }
))
wd <- WD$new()
wd$set("..")

wd$get()
```

```
[1] "/home/rheslin/R/Projects/advanced_R"
```

```
wd$set(normalizePath("~"))
```

6.

Because S3 objects lack reference semantics, it would be impossible to update the objects when necessary.

7.

They are environments. They have just a name and class; the method's aren't S3 attributes.

Controlling Access

Private fields are known only inside instances, and dynamic fields are defined using accessors.

Active fields use active bindings. A missing argument to an accessor gets, a provided one sets.

```
Person <- R6Class("Person",
  private = list(
    .age = NA,
    .name = NULL
  ),
  active = list(
    age = function(value) {
      if (missing(value)) {
        private$.age
      } else {
        stop("`$age` is read-only", call. = FALSE)
      }
    },
    name = function(value) {
      if (missing(value)) {
```

```
      private$.name
    } else {
      stopifnot(is.character(value), length(value) == 1)
      private$.name <- value
      self
    }
  }
),
public = list(
  initialize = function(name, age = NA) {
    private$.name <- name
    private$.age <- age
  }
)
)

hadley4 <- Person$new("Hadley", age = 38)
hadley4$name
```

```
[1] "Hadley"
```

1.

```
Account4 <- R6Class("Account",
  private = list(.balance = 0),
  public = list(
    withdraw = function(amount) {
      if (amount > private$.balance) {
        cat("Overdraft", sep = "\n")
        private$.balance <- self$balance - 25
      }

      private$.balance <- self$balance - amount
      cat(sprintf("Withdrew $%.2f from account", amount), sep = "\n")
      invisible(self)
    }
  ),
  active = list(
    balance = function(value) {
      if (missing(value)) {
        private$.balance
      } else {
        stop("Can't set `$.balance`", call. = FALSE)
      }
    }
  )
)
```

```
acc4 <- Account4$new()
acc4$withdraw(50)
```

```
Overdraft
Withdrew $50.00 from account
```

2.

```
Rando <- R6::R6Class("Rando",
  private = list(.prev = NA_real_),
  active = list(
    random = function(value) {
      if (missing(value)) {
        private$.prev <- runif(1)
        private$.prev
      } else {
        stop("Can't set `$random`", call. = FALSE)
      }
    },
    prev = function(value) {
      if (missing(value)) {
        private$.prev
      } else {
        stop("Can't set `$.prev`", .call = FALSE)
      }
    }
  )
)
x <- Rando$new()
x$random
```

```
[1] 0.326393
```

```
x$prev
```

```
[1] 0.326393
```

```
tryCatch(x$prev <- 5, error = function(e) cat("Success!\n"))
```

```
Success!
```

3.

```
Password <- R6Class("Password",
  private = list(.password = NA_character_),
  public = list(
    set_password = function(password) {
      private$.password <- password
      invisible(self)
    },
    check = function(guess) {
      guess == private$.password
    }
  )
)
pass <- Password$new()
pass$set_password(password = "swordfish")
pass$check("swordfish")
```

```
[1] TRUE
```

```
pass$check("xyz")
```

```
[1] FALSE
```

4.

Nope!

```
Account5 <- R6Class("Account",
  inherit = Account4,
  active = list(test = function(value) {
    super$private$.balance
  })
)
acc <- Account5$new()
acc$test
```

```
NULL
```

Reference Semantics

The `clone` method copies R6 objects (remember `nested = TRUE`!)

Because of reference semantics, a destructor method, `finalizer`, is desirable. Only ever use this to clean up private resources.

Do not use R6 objects as field default values - you'll just get references to the same object. Prevent this by putting the relevant assignment in `initialize`.

1.

```
WriteLine <- R6Class("WriteLine", public = list(
  con = NULL,
  initialize = function(path) {
    self$con <- file(path)
    invisible(self)
  },
  append_line = function(newline) {
    cat(newline, file = self$con, sep = "\n", append = TRUE)
    invisible(self)
  },
  finalize = function() {
    close(self$con)
  }
))

write <- WriteLine$new("test.txt")
write$append_line("This is a test")
rm(write)
readLines("test.txt")
```

```
[1] "This is a test"
```

```
if (file.exists("test.txt")) file.remove("test.txt")
```

```
[1] TRUE
```