R Cheat Sheet: R5 Reference Classes

Summary of some key class mechanisms

```
1) create/get object-generator:
   gen <- setRefClass('name', fields = ,
     contains = , methods =, where =, ...)
   gen <- getRefClass('name') - generator
   gen$lock('fieldName') - lock a field
     (better to lock with accessor methods)
   gen$help(topic) - get help on the class
   gen$methods(...) - add methods to class
   gen$methods() - get a list of methods
   gen$fields() - get a list of fields
   gen$accessors(...) - create get/set fns

2) generator object used to get instance:
   inst <- gen$new(...) - instantiation
     parameters passed to initialize(...)
   inst$copy(shallow=F) - copy instance
   inst$show() - called by print
   inst$field(name, value) - set
   inst$field(name) - get
   is(inst 'envRefClass') - is R5 test
   [envRefClass is the super class for R5]

3) code from within your methods
    initialize(...) - instance initializer
    finalize() - called by garbage collector
    .self - reference to the self instance
    .refClassDef - the class definition
   methods::show() - call the show function
   callSuper(...) - call the same method in
     the super class
   .self$classVariable <- localVariable
   classVariable <<- localVariable
   globalVariable <<- localVariable
   .self$classVariable <- localVariable
   .self$field(classVar, localVar)   # set
   localVar <- .self$field(classVar) # get
Trap: very easy to confuse <- and <<-
Trap: if x is not a class field; x <<- var
      assigns to x in global environment
```

Field list - code sample

```
A <- setRefClass('A',
    fields = list(
        # 1. typed, instance field:
        exampleVar1 = 'character',
        # Note: for untyped use 'ANY'
        # 2. instance field with accessor:
        ev2.private = 'character',
        exampleVar2 = function(x) {
            if (!missing(x))
                 ev2.private <<- x
            ev2.private
        }
    ),
    methods = list(
        initialize=function (c='default') {
            exampleVar1 <<- c
            exampleVar2 <<- c
        }
    )
)
instA <- A$new('instance of A'); str(instA)
```

Inheritance code sample

```
Animal <- setRefClass('Animal',
    # virtual super class
    contains = list('VIRTUAL'),
    fields = list(
        i.am = 'character',
        noiseMakes = 'character'
    ),
    methods = list(
        initialize=function(i.am='unknown',
            noiseMakes = 'unknown') {
            .self$i.am <- i.am
            .self$noiseMakes <- noiseMakes
        },
        show = function() {
            cat('I am a '); cat(i.am)
            cat('. I make this noise: ')
            cat(noiseMakes); cat('.\n')
        }
    )
)

Cat <- setRefClass('Cat',
    contains = list('Animal'),
    methods = list(
        initialize = function()
            callSuper('cat', 'meow'),
        finalize = function()
            cat('Another cat passes.\n')
     )
)
Dog <- setRefClass('Dog',
    contains = list('Animal'),
    methods = list(
        initialize = function()
            callSuper('dog', 'woof'),
        show = function() {
            callSuper()
            cat('I like to chew shoes.\n')
        }
    )
)

mongrel <- Animal$new() # FAILS!
fido = Dog$new(); felix = Cat$new()
print(fido); print(felix)
felix <- NULL; gc() # felicide!
```

What's neither C++ nor Java

1) No information hiding. Everything is public and modifiable. (But the R package mechanism helps here).
2) No static class fields.
3) Not as developed or robust OOP space.

Tips (safer coding practices) and traps

1) use named field list to type variables
2) use accessor methods in the field list to maintain class type & state validity
3) *Trap*: methodName <- function() in methods list. Use = (it's a named list!)
4) *Trap*: cant use enclosing environments within R5 classes (as they are in one).