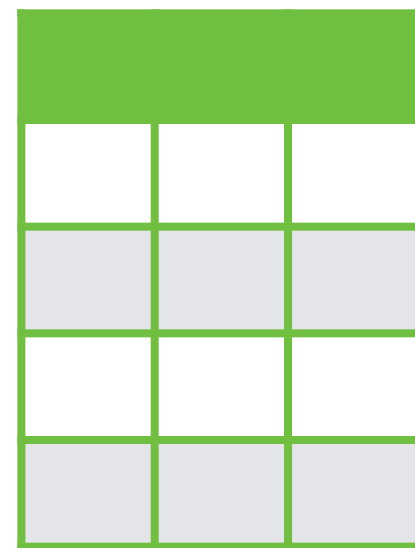
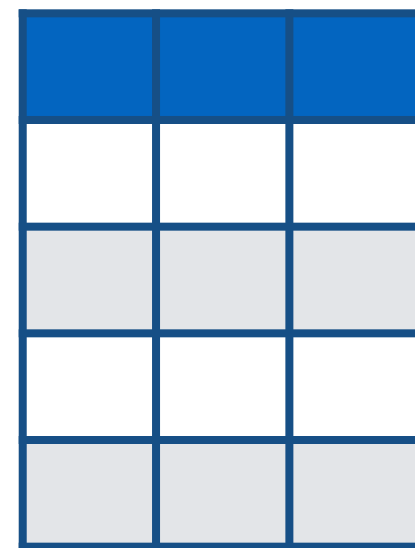




OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

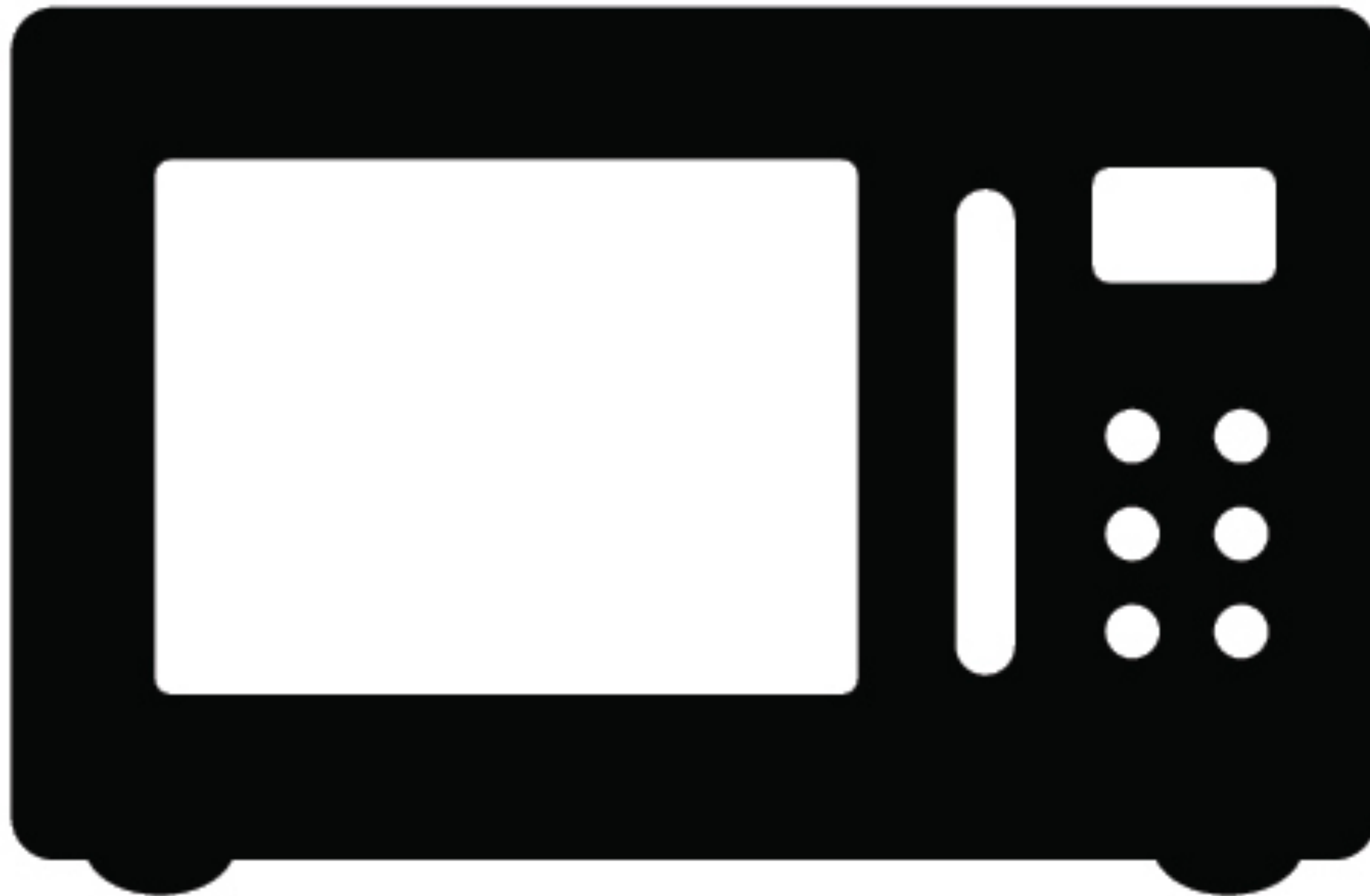
The Object Factory

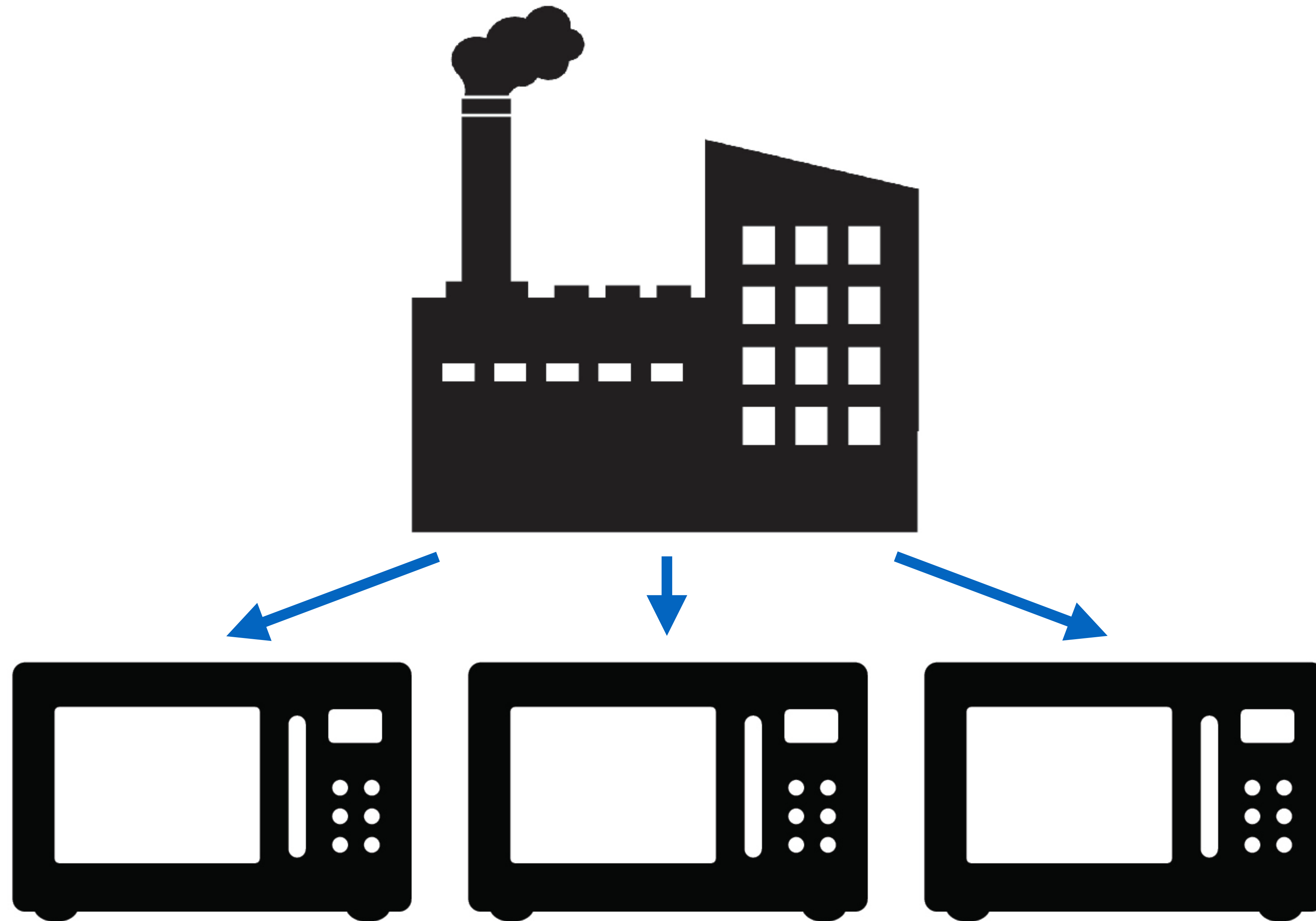


pour

refill

class generators are templates for objects
a.k.a. factories





```
library(R6)
thing_factory <- R6Class(
  "Thing",
  private = list(
    a_field = "a value",
    another_field = 123
  )
)
```

Coming soon...

public
active

```
> a_thing <- thing_factory$new()  
> another_thing <- thing_factory$new()  
> yet_another_thing <- thing_factory$new()
```


Summary

- Load the **R6** package to work with R6!
- Define **class generators** with **R6Class()**
- **Class names** should be **UpperCamelCase**
- **Data fields** stored in **private** list
- **Create objects** with factory's **new()** method



OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

Let's practice!



OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

Hiding Complexity with Encapsulation

Encapsulation

implementation



user interface

```
microwave_oven_factory <- R6Class(  
  "MicrowaveOven",  
  private = list(  
    power_rating_watts = 800,  
    door_is_open = FALSE  
  ),  
  public = list(  
    open_door = function() {  
      private$door_is_open <- TRUE  
    }  
  )  
)
```

private\$ accesses **private** elements
self\$ accesses **public** elements

Summary

- **Encapsulation = separating** implementation from UI
- Store **data** in **private** list
- Store **methods** in **public** list
- Use **private\$** to access **private** elements
- ...and **self\$** to access **public** elements



OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

Let's practice!



OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

Getting and Setting with Active Bindings



**CONTROLLED
ACCESS ZONE**

getting = **read** the data field
setting = **write** the data field

Active Bindings

defined like functions
accessed like data variables

```
thing_factory <- R6Class(  
  "Thing",  
  private = list(  
    ..a_field = "a value",  
    ..another_field = 123  
  ),  
  active = list(  
    a_field = function() {  
      if(is.na(private$..a_field)) {  
        return("a missing value")  
      }  
      private$..a_field  
    },  
    another_field = function(value) {  
      if(missing(value)) {  
        private$..another_field  
      } else {  
        assert_is_a_number(value)  
        private$..another_field <- value  
      }  
    }  
  )  
)
```

```
> a_thing <- thing_factory$new()
```

```
> a_thing$a_field  
[1] "a value"
```

```
> a_thing$a_field <- "a new value"  
Error in (function (value) : a_field is read-only.
```

```
> a_thing$another_field <- 456
```

```
> a_thing$another_field <- "456"  
Error in (function (value) : is_a_number : value is not of  
class 'numeric'; it has class 'character'.
```

Summary

- **Control** private access with **active bindings**
- **Defined** like **functions**
- **Accessed** like **data**
- Use **assertive** to **check** binding inputs



OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

Let's practice!