Object Oriented design

Mackenzie Norman

April 3, 2025

Why?

“Object - oriented approaches localize Information around objects”
- *Edward V. Berard*

Object Oriented design allows for

- Greater extensibility
- Less repeated Code
- Easier design process

Core Concepts of Object Oriented Design

There are key three features of Object Oriented design.

- Encapsulation & data-hiding
- Inheritance
- Polymorphism

Note: In small software projects encapsulation is often less important than the other 2, so I will most likely ignore it in this presentation, if you are writing large software in R please find a new profession

Polymorphism and Inheritance

Polymorphism and Inheritance are what make object oriented programming so powerful. Lets use the example of a car. We can think of some other types of cars:

- A Sedan
- A Truck
- A hatchback

We inherently know that these *objects* are all types of cars and will preform the same activity (driving) However - these vehicles have very different engines *under the hood*. Object Oriented Code is an attempt to make the future code writer, the driver. It doesn't matter how the car drives or what the engine does, a driver can drive all three of these cars

Polymorphism and Inheritance

Continuing the analogy, polymorphism is the idea that road shouldn't handle a sedan differently from a truck.

In terms of code we can see this like:

```r
        s <- sedan$new()
        t <- truck$new()

        #these should do the same thing
        s$drive()
        t$drive()

```

Polymorphism and Inheritance

While polymorphism is fairly abstract and able to be understood without code, Inheritance is the mechanism in which polymorphism is realized.
Unfortunately (or fortunately) to understand Inheritance we need to look at code

Polymorphism and Inheritance

Here is our pet class in code, its very simple, just a name and a favorite food

```r
Pet <- R6Class("Pet",
  public = list(
    name = NULL,
    favorite_food = NULL,
    initialize = function(name = "Bram", fav_food = "Pet Food") {
      self$name <- name
      self$favorite_food <- fav_food
    },
    print = function() {
      cat(paste0("The pets name is ", self$name, "\n"))
    },
```

Polymorphism and Inheritance

Now we can create a cat class that *inherits* from this pet class. This relationship is often referred to parent and child classes.

```r

Cat <- R6Class("Cat",
  inherit = Pet,
  public = list(
    color = NULL,
    initialize = function(name = "Bram", favorite_food = "Chickenm", color = NA) {
      super$initialize(name, favorite_food)
      self$color <- color
    }
  )
)
```

Polymorphism and Inheritance

```r
pet1 <- Pet$new("Walter", "fish")
pet2 <- Pet$new()
cat1 <- Cat$new("Bramble", "Chicken", "Black")
cat2 <- Cat$new("Susie", "Chicken", "Black")
# we can print both of these! even though we never wrote a print function for the Cat Class
print(pet1)
# > The pets name is Walter
print(cat1)
# > The pets name is Bramble
for (pet in list(pet1, pet2, cat1, cat2)) {
  print(pet)
}
# > The pets name is Walter
# > The pets name is Bram
# > The pets name is Bramble
# > The pets name is Susie
```

Polymorphism and Inheritance

```r

Cat <- R6Class("Cat",
  inherit = Pet,
  public = list(
    color = NULL,
    initialize = function(name = "Bram", favorite_food = "Chickenm", color = NA) {
      super$initialize(name, favorite_food)
      self$color <- color
    },
    print = function() {
      cat(paste0(super$print(), "It is a ", self$color, " cat. \n"))
    }
  )
)
```

Polymorphism and Inheritance

```r
pet1 <- Pet$new("Walter", "fish")
pet2 <- Pet$new()
cat1 <- Cat$new("Bramble", "Chicken", "Black")
cat2 <- Cat$new("Susie", "Chicken", "Black")

print(pet1)
# > The pets name is Walter
print(cat1)
# > The pets name is Bramble
# > It is a Black cat.
for (pet in list(pet1, pet2, cat1, cat2)) {
  print(pet)
}
# > The pets name is Walter
# > The pets name is Bram
# > The pets name is Bramble
# > It is a Black cat.
# > The pets name is Susie
```