Generics and Methods

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Richie Cotton
Data Evangelist at DataCamp

```r
summary(c(TRUE, FALSE, NA, TRUE))
```

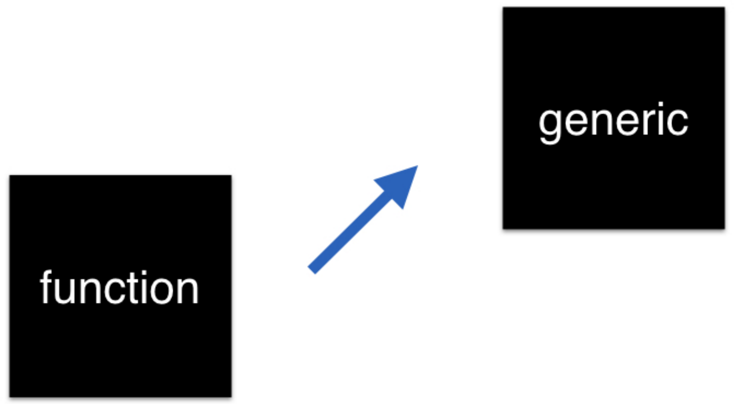
```
   Mode   FALSE    TRUE    NA's
logical       1       2       1
```

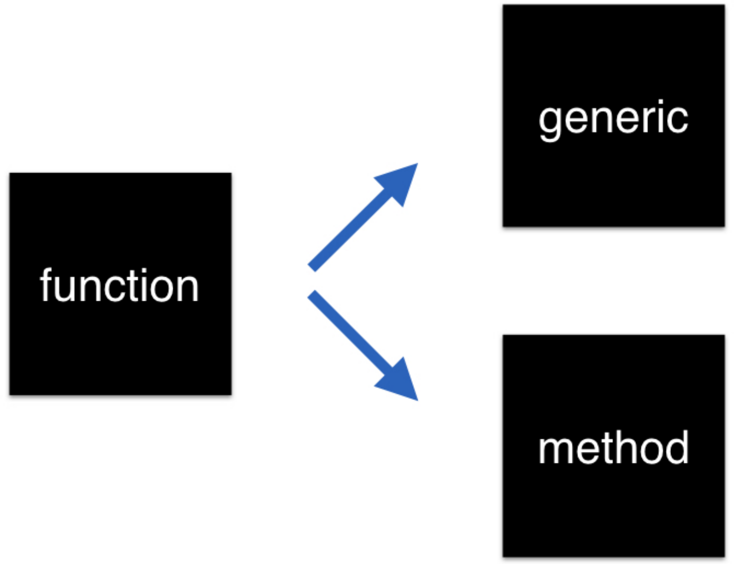
```r
summary(rgamma(1000, 1))
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000354 0.276500 0.690300 1.020000 1.384000 9.664000
```

function overloading = input-dependent function behavior

function


function
generic


function
generic
method

```r
print
```

```
function (x, ...)
UseMethod("print")
<bytecode: 0x1062f0870>
<environment: namespace:base>
```

Methods are named generic.class

- `print.Date`
- `summary.factor`
- `unique.array`

Method signatures contain generic signatures

```r
args(print)
```

```
function (x, ...)
NULL
```

```r
args(print.Date)
```

```
function (x, max = NULL, ...)
NULL
```

pass arguments between methods with `...`

include it in both generic and methods

```r
print.function
```

```
function (x, useSource = TRUE, ...)
.Internal(print.function(x, useSource, ...))
```

```
print.Date
```

```
function (x, max = NULL, ...)
{
    if (is.null(max))
        max <- getOption("max.print", 9999L)
    if (max < length(x)) {
        print(format(x[seq_len(max)]), max = max, ...)
        cat(" [ reached getOption(\"max.print\") -- omitted",
            length(x) - max, "entries ]\n")
    }
    else print(format(x), max = max, ...)
    invisible(x)
}
```

~~lower.leopard.case~~

~~lower.leopard.case~~

lower_snake_case

~~lower.leopard.case~~

lower_snake_case

lowerCamelCase

Summary

- Functions **split** into **generic + method**
- Methods named `generic.class`
- Method args **contain generic** args
- Include a `...` arg
- Use `lower_snake_case` or `lowerCamelCase`

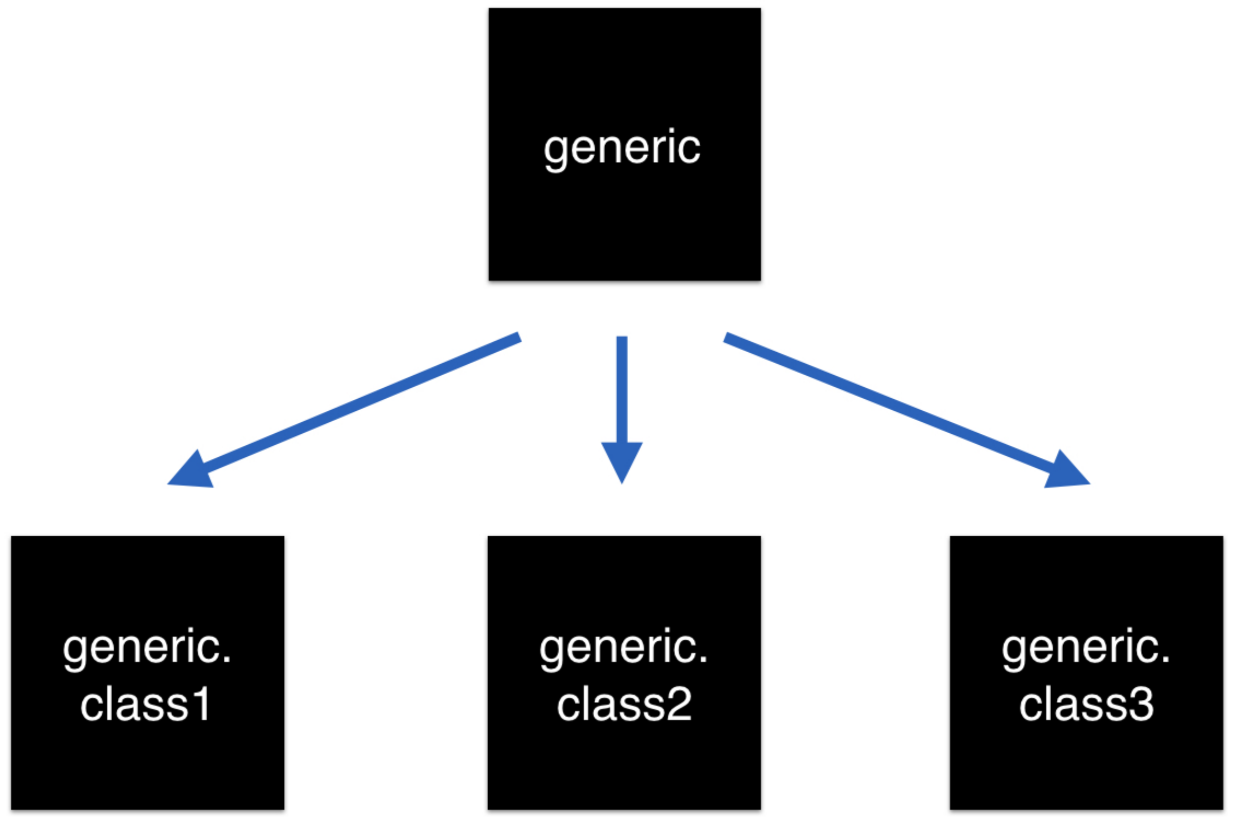
Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Methodical Thinking

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

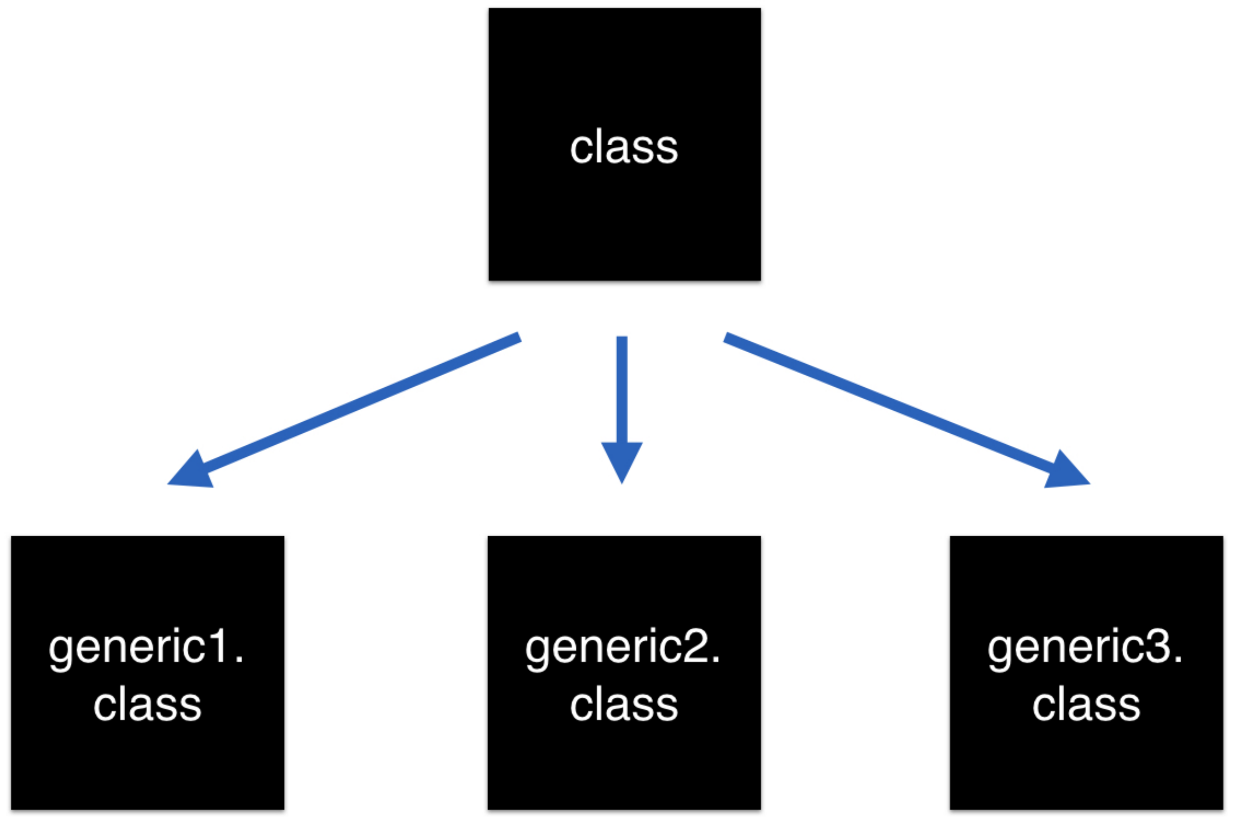
Richie Cotton
Data Evangelist at DataCamp


generic
generic.class1
generic.class2
generic.class3

```
methods
```

```r
methods("mean") # or methods(mean)
```

```
mean.Date     mean.default  mean.difftime mean.POSIXct
mean.POSIXlt
see '?methods' for accessing help and source code
```


class
generic1.
class
generic2.
class
generic3.
class

```r
methods(class = "glm") # or methods(class = glm)
```

```
add1            anova           coerce
confint         cooks.distance  deviance
drop1           effects         extractAIC
family          formula         influence
initialize      logLik          model.frame
nobs            predict         print
residuals       rstandard       rstudent
show            slotsFromS3     summary
vcov            weights
see '?methods' for accessing help and source code
```

`methods()` returns **S3** *and* **S4** methods

```r
.S3methods(class = "glm")
```

```
add1            anova           confint
cooks.distance  deviance        drop1
effects         extractAIC      family
formula         influence       logLik
model.frame     nobs            predict
print           residuals       rstandard
rstudent        summary         vcov
weights
see '?methods' for accessing help and source code
```

```r
.S4methods(class = "glm")
```

```
coerce       initialize   show         slotsFromS3
see '?methods' for accessing help and source code
```

Summary

- `methods()` **finds methods** for a generic
- `...` or for a **class**
- `.S3methods()` finds **only S3** methods

Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Method Lookup for Primitive Generics

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Richie Cotton
Data Evangelist at DataCamp

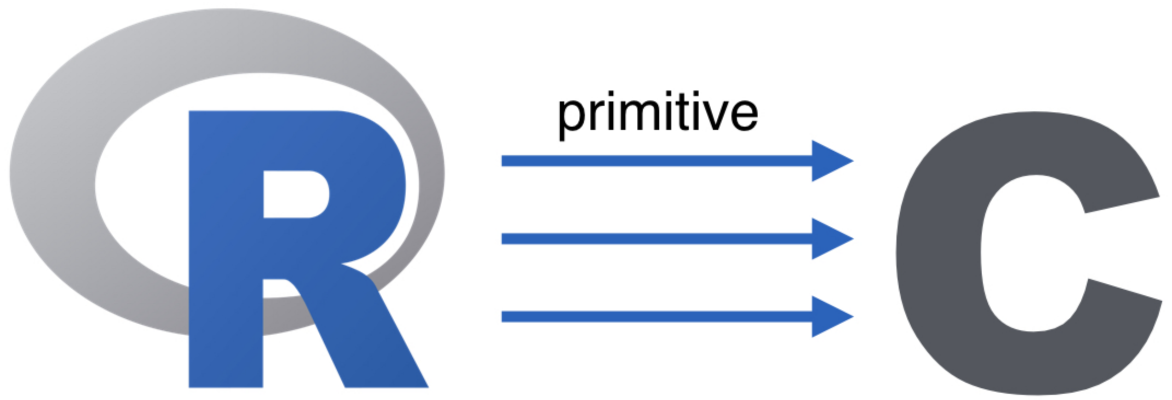
- Writing code
- Debugging code
- Maintaining code

- Running code

R vs. C

- **C** code often runs faster
- **R** code is usually easier to **write**
- ... and easier to **debug**

C


primitive

```r
exp
```

```
function (x)  .Primitive("exp")
```

```r
sin
```

```
function (x)  .Primitive("sin")
```

```r
+
```

```
function (e1, e2)  .Primitive("+")
```

```r
-
```

```
function (e1, e2)  .Primitive("-")
```

```r
if
```

```
.Primitive("if")
```

```r
for
```

```
.Primitive("for")
```

```
.S3PrimitiveGenerics
```

```
"anyNA"          "as.character"   "as.complex"
"as.double"      "as.environment" "as.integer"
"as.logical"     "as.call"        "as.numeric"
"as.raw"         "c"              "dim"
"dim<-"          "dimnames"       "dimnames<-"
"is.array"       "is.finite"      "is.infinite"
"is.matrix"      "is.na"          "is.nan"
"is.numeric"     "length"         "length<-"
"levels<-"       "names"          "names<-"
"rep"            "seq.int"        "xtfrm"
```

```r
all_of_time <- c("1970-01-01", "2012-12-21")
as.Date(all_of_time)
```

```
"1970-01-01" "2012-12-21"
```

```r
class(all_of_time) <- "date_strings"
as.Date(all_of_time)
```

```
Error in as.Date.default(all_of_time) :
  do not know how to convert 'all_of_time' to class"Date"
```

```r
length(all_of_time)
```

```
2
```

Summary

- Some R functions are actually **written in C**
- The **primitive** interface gives **best performance**
- `.S3PrimitiveGenerics` lists **primitive S3 generics**
- Primitive generics **don't throw an error** when no method is found

Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Too Much Class

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Richie Cotton
Data Evangelist at DataCamp

```r
x <- c(1, 3, 6, 10, 15)
class(x) <- c(
  "triangular_numbers", "natural_numbers", "numeric"
)
```

```r
is.numeric(x)
```

```
TRUE
```

```r
is.triangular_numbers(x)
```

```
Error: could not find function "is.triangular_numbers"
```

```r
inherits(x, "triangular_numbers")
```

```
TRUE
```

```r
inherits(x, "natural_numbers")
```

```
TRUE
```

```r
inherits(x, "numeric")
```

```
TRUE
```

```r
what_am_i <- function(x, ...) {
  UseMethod("what_am_i")
}
```

```r
what_am_i.triangular_numbers <- function(x, ...) {
  message("I'm triangular numbers")
  NextMethod("what_am_i")
}
```

```r
what_am_i.natural_numbers <- function(x, ...) {
  message("I'm natural numbers")
  NextMethod("what_am_i")
}
```

```r
what_am_i.numeric <- function(x, ...) {
  message("I'm numeric")
}
```

```
what_am_i(x)
```

```
I'm triangular numbers
I'm natural numbers
I'm numeric
```

what_am_i

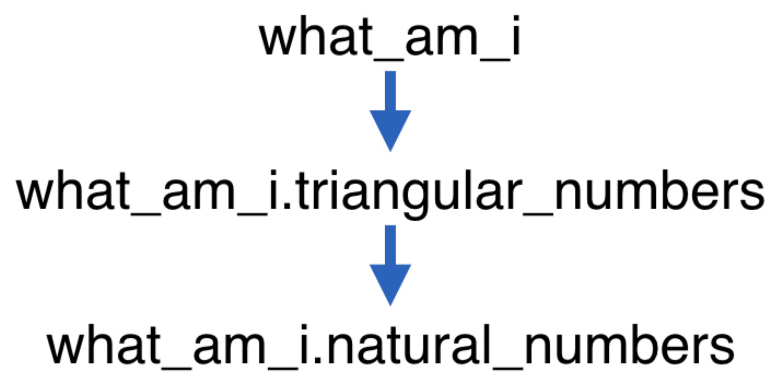
```r
what_am_i(x)
```

```
I'm triangular numbers
I'm natural numbers
I'm numeric
```

$$\begin{tikzcd}
\text{what\_am\_i} \arrow[d] \\
\text{what\_am\_i.triangular\_numbers}
\end{tikzcd}$$

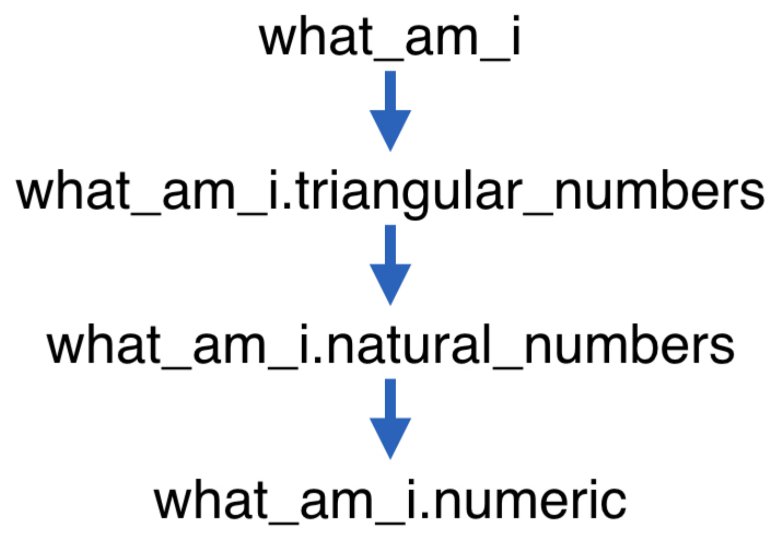
```
what_am_i(x)
```

```
I'm triangular numbers
I'm natural numbers
I'm numeric
```



```r
what_am_i(x)
```

```
I'm triangular numbers
I'm natural numbers
I'm numeric
```



Summary

- **Multiple classes** are allowed
- Use `inherits()` to test for **arbitrary classes**
- Use `NextMethod()` to **chain method calls**

Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R