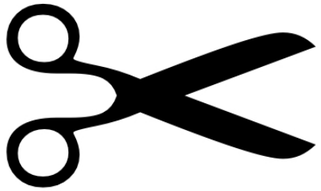
Propagating Functionality with Inheritance

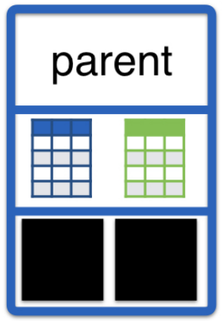
OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Richie Cotton
Data Evangelist at DataCamp

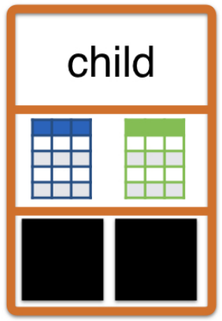
datacamp



```
thing_factory <- R6Class(
  "Thing",
  private = list(
    a_field = "a value",
    another_field = 123
  ),
  public = list(
    do_something = function(x, y, z) {
      # do something here
    }
  )
)
```

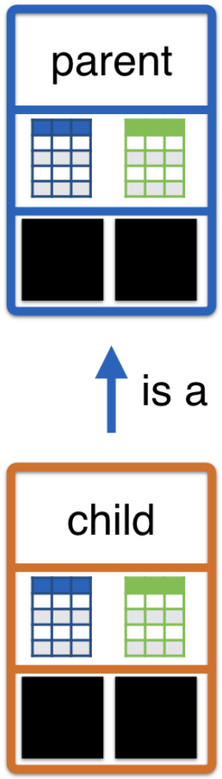


the class you inherit from

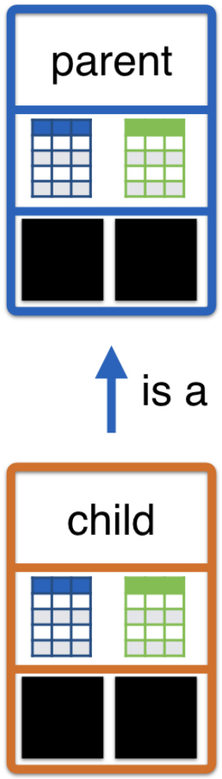


the class that inherits fields and methods

```
child_thing_factory <- R6Class(
  "ChildThing",
  inherit = thing_factory
  public = list(
    do_something_else = function() {
      # more functionality
    }
  )
)
```



- a fancy microwave is a microwave



- a fancy microwave is a microwave
- not all microwaves are fancy microwaves

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

```
"Thing" "R6"
```

```
inherits(a_thing, "Thing")
```

```
TRUE
```

```
inherits(a_thing, "R6")
```

```
TRUE
```

```
a_child_thing <- child_thing_factory$new()
class(a_child_thing)
```

```
"ChildThing" "Thing"      "R6"
```

```
inherits(a_child_thing, "ChildThing")
```

```
TRUE
```

```
inherits(a_child_thing, "Thing")
```

```
TRUE
```

```
inherits(a_child_thing, "R6")
```

```
TRUE
```

Summary

- **Propagate functionality** using **inheritance**
- Use the `inherit` **arg** to `R6Class()`
- **Children get** their **parent's functionality**
- ... but the **converse is not true**

Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

datacamp

Embrace, Extend, Override

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Richie Cotton
Data Evangelist at DataCamp

```
thing_factory <- R6Class(
  "Thing",
  public = list(
    do_something = function() {
      message("the parent do_something method")
    }
  )
)
```

```
child_thing_factory <- R6Class(
  "ChildThing",
  inherit = thing_factory,
  public = list(
    do_something = function() {
      message("the child do_something method")
    },
    do_something_else = function() {
      message("the child do_something_else method")



    }
  )
)
```

```
a_child_thing <- child_thing_factory$new()
```

```
a_child_thing$do_something()
```

```
the child do_something method
```

`private$` accesses **private** fields

`self$` accesses **public** methods in **self**

`super$` accesses **public** methods in **parent**

```
child_thing_factory <- R6Class(
  "ChildThing",
  inherit = thing_factory,
  public = list(
    do_something = function() {
      message("the child do_something method")
    },
    do_something_else = function() {
      message("the child do_something_else method")
     self$do_something()
     super$do_something()
    }
  )
)
```

```
a_child_thing <- child_thing_factory$new()
```

```
a_child_thing$do_something_else()
```

```
the child do_something_else method
the child do_something method
the parent do_something method
```

Summary

- **Override** by giving the **same name**
- **Extend** by giving a **new name**
- `self$` accesses **public** methods in **self**
- `super$` accesses **public** methods in **parent**

Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

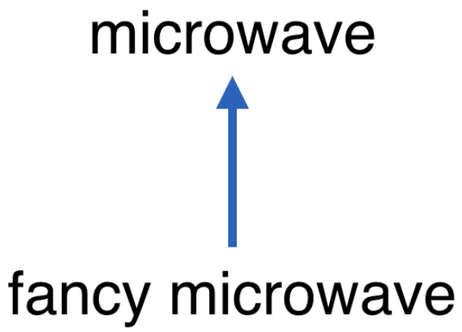
datacamp

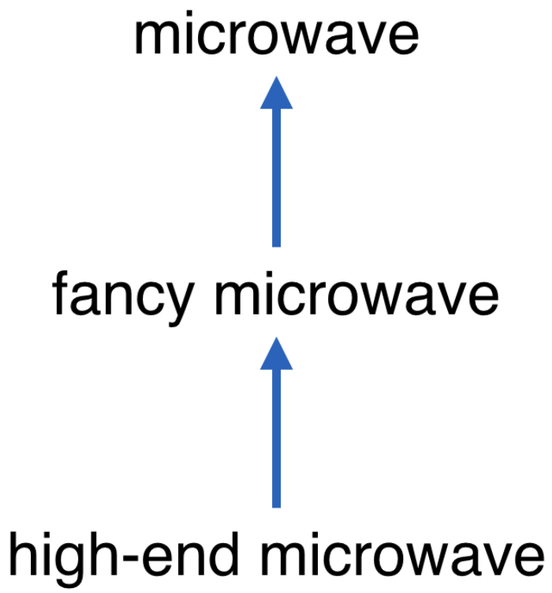
Multiple Levels of Inheritance

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

Richie Cotton
Data Evangelist at DataCamp


CLOCK
AM / PM
AUTO NITE LIGHT
TIMER
CONVENIENCE COOKING
POPCORN
REHEAT
BEVERAGE
COOK
POTATO
VEGETABLE
SNACKS
TIME COOK
DEFROST AUTO / TIME
REMINDER
HELP
DELAY START
START
CLEAR OFF
POWER LEVEL
SURFACE LIGHT
BEEPER VOLUME
TURNTABLE
VENT FAN


microwave
fancy microwave


microwave
fancy microwave
high-end microwave

```
thing_factory <- R6Class(
  "Thing"
)
```

```
child_thing_factory <- R6Class(
  "ChildThing",
  inherit = thing_factory
)
```

```
grand_child_thing_factory <- R6Class(
  "GrandChildThing",
  inherit = child_thing_factory
)
```

```
thing_factory <- R6Class(
  "Thing",
  public = list(
    do_something = function() {
      message("the parent do_something method")
    }
  )
)
```

```
child_thing_factory <- R6Class(
  "ChildThing",
  inherit = thing_factory,
  public = list(
    do_something = function() {
      message("the child do_something method")
    }
  )
)
```

```
grand_child_thing_factory <- R6Class(
  "GrandChildThing",
  inherit = child_thing_factory,
  public = list(
    do_something = function() {
      message("the grand-child do_something method")
      super$do_something()
      super$super$do_something()
    }
  )
)
```

```
a_grand_child_thing <- grand_child_thing_factory$new()
a_grand_child_thing$do_something()
```

```
the grand-child do_something method
the child do_something method
Error in a_grand_child_thing$do_something():
  attempt to apply non-function
```

```
child_thing_factory <- R6Class(
  "ChildThing",
  inherit = thing_factory,
  public = list(
    do_something = function() {
      message("the child do_something method")
    }
  ),
  active = list(
    super_ = function() super
  )
)
```

```
grand_child_thing_factory <- R6Class(
  "GrandChildThing",
  inherit = child_thing_factory,
  public = list(
    do_something = function() {
      message("the grand-child do_something method")
      super$do_something()
      super$super_$do_something()
    }
  )
)
```

```
a_grand_child_thing <- grand_child_thing_factory$new()
a_grand_child_thing$do_something()
```

```
the grand-child do_something method
the child do_something method
the parent do_something method
```

Summary

- **R6** objects can only **access their direct parent**
- **Intermediate** classes can **expose their parent**
- Use an **active binding** named `super_`
- `super_` should simply return `super`

Let's practice!

OBJECT-ORIENTED PROGRAMMING WITH S3 AND R6 IN R

datacamp