Package 'bandicoot'

March 9, 2024

Type Package

Title Light-Weight 'python'-Like Object-Oriented System

Version 1.0.0

Description A light-weight object-oriented system with 'python'-like syntax which supports multiple inheritances and incorporates a 'python'-like method resolution order.

License MIT + file LICENSE

URL https://tengmcing.github.io/bandicoot/, https://github.com/TengMCing/bandicoot/

BugReports https://github.com/TengMCing/bandicoot/issues

Encoding UTF-8

Imports cli, utils

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

RoxygenNote 7.2.1

VignetteBuilder knitr

NeedsCompilation no

Author Weihao Li [aut, cre, cph] (<https://orcid.org/0000-0003-4959-106X>)

Maintainer Weihao Li <llreczx@gmail.com>

Repository CRAN

Date/Publication 2024-03-09 11:10:02 UTC

R topics documented:

`as_bandicoot_oop` *Turn an environment into a* `bandicoot_oop` *object*

Description

This function tries to turn an environment into a `bandicoot_oop` object.

Usage

```
as_bandicoot_oop(
  env,
  ..class.. = NULL,
  ..type.. = NULL,
  ..instantiated.. = NULL,
  overwrite_container = FALSE,
  register = FALSE,
  in_place = FALSE,
  container_name = "..method_env..",
  self_name = "self"
)
```

Arguments

`env`	An environment.
`..class..`	Character. A series of class names.
`..type..`	Character. The class name of this object.
`..instantiated..`	Boolean. Whether this object is an instance.
`overwrite_container`	Boolean. Whether or not to overwrite the container.
`register`	Boolean. Whether or not to register functions if there are any.
`in_place`	Boolean. Whether or not to modify the environment in-place. If not, a new environment will be created.
`container_name`	Character. Name of the container.
`self_name`	Character. Name of the self reference.

Value

A Boolean value.

Examples

```
e <- new.env()
e$a <- function() self

as_bandicoot_oop(e,
                 ..class.. = "test",
                 ..type.. = "test",
                 ..instantiated.. = FALSE,
                 register = TRUE,
                 in_place = TRUE)

e
e$a()
```

BASE *BASE class environment*

Description

This class provides essential attributes and methods. It makes the assumption that the container name is `..method_env..` and the name of the reference to self is `self`. If you would like to use other container names and self names, you need to overwrite the class definition of BASE.

The class environment is defined as an empty environment by `new.env()` at build-time, and the class descriptor is run at load-time by `.onLoad()`. This ensures methods and attributes of the class is built with the load-time (usually latest) installed dependencies (if it depends on any). Derived classes should follow the same principle to avoid running the class descriptor at build-time, and only defines the content of the class at load-time.

Since `bandicoot` does not support dynamic dispatch, calling the correct parent method can be difficult in a complex class system. So, users can use the `..mro..` (method resolution order) attribute and the `super()` function to determine the correct super/next class. If users decide to store parent environments in the derived class such that parent method can be called more handily, awareness needs to be raised when saving and loading these classes/instances. It is very likely the same class stored in different objects becomes different environments.

Usage

```
base_(..., env = new.env(parent = parent.frame()), init_call = sys.call())
```

Arguments

`...`	Ignored.
`env`	Environment. The instance environment.
`init_call`	Call. Contents of the `..init_call...` It is recommended to leave it as default.

Value

An instance environment.

Functions

- `base_()`: Class constructor, same as `BASE$instantiate()`.

Class information

Attributes:

- B:
  - BASE$..bases..
- C:
  - BASE$..class..
  - BASE$..class_tree..
- I:
  - BASE$..instantiated..
- M:
  - BASE$..method_env..
  - BASE$..mro..

Methods:

- D:
  - `BASE$del_attr()`
  - `BASE$..dir..()`
- G:
  - `BASE$get_attr()`
- H:
  - `BASE$has_attr()`
- I:
  - `BASE$..init..()`
  - `BASE$instantiate()`
- L:
  - `BASE$..len..()`
- M:
  - `BASE$..methods..()`
- N:
  - `BASE$..new..()`

- R:
  - BASE$..repr..()
- S:
  - BASE$set_attr()
  - BASE$..str..()

BASE$..bases.. *Direct parent classes*

Description

Direct parent classes

Examples

```
BASE$..bases..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..bases..
```

BASE$..class.. *Class name and parent class names*

Description

A string vector.

Examples

```
BASE$..class..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..class..
```

BASE$..class_tree.. *Class name and parent class names represented in a tree*

Description

A list.

Examples

```
BASE$..class_tree..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..class_tree..
```

BASE$..dir.. *All names in the class or instance environment*

Description

This function returns all names in the environment.

Usage:

```
BASE$..dir..()
```

Value

A vector of string.

Examples

```
BASE$..dir..()

# Instantiate
test <- BASE$instantiate()
test$..dir..()
```

BASE$..init.. *Initialization method*

Description

This function will be called after an instance is built. User could override this function in derived class.

Usage:

```
BASE$..init..(...)
```

Arguments

...	Ignored by BASE, but user can define their owns.

Value

Return the object itself.

Examples

```
BASE$..init..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")

# Override the `..init..` method
register_method(TEST, ..init.. = function(a) {self$x <- a})

# Build a `TEST` instance
test <- TEST$instantiate(a = 2)

test$x
```

BASE$..instantiated.. *Instantiate status*

Description

Whether or not the object is an instance.

Examples

```
BASE$..instantiated..

# Instantiate
test <- BASE$instantiate()
test$..instantiated..
```

`BASE$..len..` *Length of the class or the instance*

Description

User could override this method in derived class.

Usage:

```
BASE$..len..()
```

Examples

```
BASE$..len..()

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")

# Override the `..len..` method
register_method(TEST, ..len.. = function() 1)
TEST$..len..()
```

`BASE$..methods..` *List all methods of a class or an instance*

Description

This function lists all methods of a class or an instance.

Usage:

```
BASE$..methods..()
```

Value

A string vector.

Examples

```
BASE$..methods..()
```

BASE$..method_env.. *The container*

Description

A container where methods will be executed.

Examples

```
BASE$..method_env..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..method_env..
```

BASE$..mro.. *Method resolution order*

Description

Method resolution order defined using C3 algorithm.

Examples

```
BASE$..mro..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..mro..
```

BASE$..new.. *Build a new instance from a class or an instance*

Description

This function will copy all methods and attributes, except the container, and the instantiate method. Then, the `..init_call..` attribute will be set to the current system call, and the `..instantiated..` attribute will be set to `TRUE`. Notice, the `..init..` method will not run.

Usage:

```
BASE$..new..(env = new.env(parent = parent.frame()), init_call = sys.call())
```

Arguments

`env`	Environment. The instance environment.
`init_call`	Call. Contents of the `..init_call..`. It is recommended to leave it as default.

Value

An instance environment.

Examples

```
BASE$..new..()

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")

TEST$..new..()
```

`BASE$..repr..` *"Official" String representation of the object*

Description

This function returns a "official" string representation of the object, which may be used to reconstruct the object given an appropriate environment.

Usage:

```
BASE$..repr..()
```

Value

A string.

Examples

```
BASE$..repr..()

test <- base_()
test$..repr..()

test <- BASE$instantiate()
test$..repr..()

test <- BASE$..new..()
test$..repr..()
```

BASE$..str.. *String representation of the object*

Description

This function returns a string representation of the object.

Usage:

```
BASE$..str..()
```

Value

A string.

Examples

```
BASE$..str..()

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..str..()

# Instantiate
test <- BASE$instantiate()
test$..str..()
```

BASE$..type.. *Class name*

Description

A string.

Examples

```
BASE$..type..

# Inherit from BASE
TEST <- new_class(BASE, class_name = "TEST")
TEST$..type..
```

BASE$del_attr *Delete an attribute*

Description

This function delete an attribute.

Usage:

```
BASE$del_attr(attr_name)
```

Arguments

attr_name	Character. Attribute name.

Value

Return the object itself.

Examples

```
test <- BASE$instantiate()
test$set_attr("x", 1)
test$x
test$del_attr("x")
test$x
```

BASE$get_attr *Get value of an attribute or a method*

Description

This function gets the value of an attribute or a method.

Usage:

```
BASE$get_attr(attr_name)
```

Arguments

attr_name	Character. Attribute name.

Value

The attribute value.

Examples

```
BASE$get_attr("test")

BASE$get_attr("..methods..")
```

`BASE$has_attr` *Whether or not an attribute or method exists*

Description

This function checks whether or not an attribute or method exists.

Usage:

```
BASE$has_attr(attr_name)
```

Arguments

`attr_name`	Character. Attribute name.

Value

True or FALSE.

Examples

```
BASE$has_attr("test")

BASE$has_attr("..len..")
```

`BASE$instantiate` *Instantiate method*

Description

This function will new an instance using the `..new..` method, then initialized the instance with the `..init..` method.

Usage:

```
BASE$instantiate(
  ...,
  env = new.env(parent = parent.frame()),
  init_call = sys.call()
)
```

Arguments

`...`	Arguments passed to `..init..` method.
`env`	Environment. The instance environment.
`init_call`	Call. Contents of the `..init_call..`. It is recommended to leave it as default.

Value

An instance environment.

Examples

```
BASE$..dir..()

# Build an instance
base_instance <- BASE$instantiate()

base_instance$..dir..()
```

`BASE$set_attr` *Set value of an attribute or a method*

Description

This function sets the value of an attribute or a method.

Usage:

```
BASE$set_attr(attr_name, attr_val)
```

Arguments

`attr_name`	Character. Attribute name.
`attr_val`	Any value.

Value

Return the object itself.

Examples

```
test <- BASE$instantiate()
test$set_attr("x", 1)
test$x
```

`bind_fn_2_env` *Bind functions of the current environment to a target environment*

Description

This function is equivalent to `environment(fn) <- env`. Hence functions must bind to names.

Usage

```
bind_fn_2_env(env, ...)
```

Arguments

`env`	Environment.
`...`	Functions.

Details

Pass character function names to `...` will cause error.

Value

No return value, called for side effects.

Examples

```
# Access the associated environment inside a function

self <- NULL
e <- new.env()

# The associated environment needs to have a reference to itself
e$self <- e

e$show_self <- function() return(self)

# The function can only access the global variable `self`
e$show_self()

# Bind the function to the environment `e`
bind_fn_2_env(env = e, e$show_self)

# Both point to the same environment
e$show_self()
e
```

check_method *Check each method body in an object if it contains names that do not explicitly bind to a specified namespace via* ::.

Description

Method body could contain names like "mutate" that are from packages, it usually would not be a problem as long as the package namespace is in the search path or it is available in the parent environment of the object. However, if the package is not loaded via functions like `library()` and the name used in the method body is unavailable in the parent environment of the object, then an error may be raised saying that "object `name` not found" when the method is run.

This function helps detect this kind of problems. Users needs to specify the names they want to detect, and specify the package they belong to.

Usage

```
check_method(env, symbol_name, target_namespace)
```

Arguments

`env`	Environment. An environment.
`symbol_name`	Character. Names that want to be detected.
`target_namespace`	Character. Name of the package that symbols belong to.

Value

No return value, called for side effects.

Examples

```
e <- new.env()
register_method(e, test = function() cli_alert_info("test"))
check_method(e, "cli_alert_info", "cli")

register_method(e, test = function() cli::cli_alert_info("test"))
check_method(e, "cli_alert_info", "cli")
```

copy_attr *Copy attributes and methods from classes or instances*

Description

This function copy attributes and methods from classes or instances to class or instance.

Usage

```
copy_attr(
  env,
  ...,
  avoid = c("..method_env..", "..init_call.."),
  class_name = env$..type..
)
```

Arguments

env	Environment. The destination environment.
...	Environments. Source environments.
avoid	Character. Names that don't want to be copied.
class_name	Character. Name of the class the method is defined. This is important for super() to resolve the correct parent class.

Details

Multiple classes or instances can be provided in ..., where the right one will override the left one if they have the same attribute or method name. Attributes or methods that don't want to be copied can be specified in avoid.

Value

Return the object itself.

Examples

```
test <- new.env()
names(BASE)
copy_attr(test, BASE, avoid = c("..method_env..", "..init_call..", "..dir.."))
names(test)
```

define_pkg_fn *Load functions from package namespaces into current environment*

Description

This function loads functions from package namespaces and assigns them to the preferred function names in the current environment.

Usage

```
define_pkg_fn(pkg, ...)
```

Arguments

pkg	Package.
...	Functions. Preferred names can be provide via named arguments.

Details

Preferred function names can be provide via named arguments like `info = cli_alert_info`.

Value

No return value, called for side effects.

Examples

```
define_pkg_fn(pkg = cli, cli_alert_info, cli_alert_warning)
define_pkg_fn(cli, cli_alert_warning, info = cli_alert_info)
```

is_bandicoot_oop *Check whether the object is a* bandicoot_oop *object*

Description

This function check whether the object is a `bandicoot_oop` object.

Usage

```
is_bandicoot_oop(obj, why = FALSE)
```

Arguments

obj	Any object.
why	Boolean. Whether or not to print the reason when the check fail.

Value

A Boolean value.

Examples

```
e <- new.env()
is_bandicoot_oop(e)

e <- new_class(class_name = "test")
is_bandicoot_oop(e)
```

`iter` *Build an iterator*

Description

Build an iterator

Usage

```
iter(x, ...)
```

Arguments

`x`	Object.
`...`	Additional arguments needed for building an iterator.

Value

An iterator.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..iter.. = function(...) {
                  split(data.frame(name = self$name, age = self$age),
                        1:length(self$name))
                })

good_company <- company(c("patrick", "james"),
```

```
                        c(33, 34))
for (person in iter(good_company)) {
  print(person)
}
```

`iter.bandicoot_oop` *S3 method of building an iterator of a* `bandicoot_oop` *object*

Description

This function builds an iterator using the `..iter..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
iter(x, ...)
```

Arguments

`x`	`bandicoot_oop` object.
`...`	Additional arguments needed for building an iterator.

Value

An iterator.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..iter.. = function(...) {
                  split(data.frame(name = self$name, age = self$age),
                        1:length(self$name))
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
for (person in iter(good_company)) {
  print(person)
}
```

len *Compute the length of the object*

Description

Compute the length of the object

Usage

```
len(x, ...)
```

Arguments

x	Object.
...	Additional arguments needed for computing the length.

Value

An integer.

Examples

```
EMAIL <- new_class(class_name = "EMAIL")
email <- EMAIL$instantiate
register_method(EMAIL, ..init.. = function(address) self$address = address)
register_method(EMAIL, ..len.. = function() nchar(self$address))

patrick <- email('patrick@test.com')
len(patrick)
```

len.bandicoot_oop *S3 method of computing the length of* bandicoot_oop *object*

Description

This function computes the length of the object by using the ..len..() method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
len(x, ...)
```

Arguments

x	bandicoot_oop object.
...	ignored.

Value

An integer.

Examples

```
EMAIL <- new_class(class_name = "EMAIL")
email <- EMAIL$instantiate
register_method(EMAIL, ..init.. = function(address) self$address = address)
register_method(EMAIL, ..len.. = function() nchar(self$address))

patrick <- email('patrick@test.com')
len(patrick)
```

---

new_class *Define a new class*

Description

This function declare a new class, and copies attributes and methods from parent classes.

Usage

```
new_class(
  ...,
  env = new.env(parent = parent.frame()),
  class_name = NULL,
  empty_class = FALSE
)
```

Arguments

...	Environments. Parent class environments.
env	Environment. The new class environment.
class_name	Name of the new class.
empty_class	Boolean. Whether to create an empty class. This should only be used when you don't want to inherited from BASE, or you want to define your own base object class. Will be ignored if ... is not empty. If ... is empty and empty_class == FALSE, BASE will be used as the parent class.

Details

Parents can be provided in `...`, where methods and attributes will be overrided by the left classes because `bandicoot` does not support dynamic dispatch at the moment. However, this behaviour usually aligns with the method resolution order defined by the C3 algorithm used in Python. If `...` is empty and `empty_class == FALSE`, BASE will be used as the parent class.

Value

A class environment with S3 class "bandicoot_oop".

Examples

```
MYCLASS <- new_class(class_name = "MYCLASS")
MYCLASS
names(MYCLASS)

# Inhert from BASE class
TEST <- new_class(BASE, class_name = "TEST")
TEST
names(TEST)
```

`print.bandicoot_oop` *S3 method of printing* `bandicoot_oop` *object*

Description

This function print the string representation of the object by using the `..str..()` method.

Usage

```
## S3 method for class 'bandicoot_oop'
print(x, ...)
```

Arguments

`x`	`bandicoot_oop` object.
`...`	ignored.

Value

No return value, called for side effects.

Examples

```
print(base_())
```

`register_method` *Register method for an object environment*

Description

This function register a function as a method of an object environment.

Usage

```
register_method(
  env,
  ...,
  container_name = "..method_env..",
  self_name = "self",
  class_name = env$..type..
)
```

Arguments

`env`	Environment. Object environment.
`...`	Named Functions. Functions needs to be provided in named format, like `a = function() 1`.
`container_name`	Character. Name of the container. Methods will be executed inside this container.
`self_name`	Character. Name of the self reference. Methods needs to use this name to access the object environment.
`class_name`	Character. Name of the class of the object environment. This is important for `super()` to resolve the correct parent class.

Details

Methods will be executed inside a container, which is a child environment of the parent of the object environment. Thus, methods can not access variables of the object environment directly, but can access variables of the parent of the object environment directly. The designed way for methods to access the object environment is by using the name "self", this name can be changed by specifying a string in `self_name`. The default name of the container is "..method_env..". This also can be changed by specifying a string in `container_name`. An object can have multiple containers, but every container is recommended to contain only one self reference.

Method needs to be provided as `a = function() 1`, where `a` is the name of the method and the right hand side of the equal sign is the function. Warning will be raised if the container contains contents other than the self reference.

Value

Return the object itself.

Examples

```
a <- function() self$x

e <- new.env()
e$x <- 1

# Register the method `aa` for environment `e` with `self_name = "self"`
register_method(e, aa = a, self_name = "self", class_name = "test")

# There is an environment `..method_env..` in the environment `e`
names(e)

# The container is empty (except `self`)
names(e$..method_env..)

# `self` is a reference to `e`
identical(e, e$..method_env..$self)

# The method `aa` will be evaluated in the container
identical(environment(e$aa), e$..method_env..)

# Therefore, `self$x` is a reference to variable `x` of the environment `e`
e$aa()
```

---

repr *The "official" string representation of an object.*

Description

The "official" string representation of an object. If at all possible, this should look like a valid R expression that could be used to recreate an object wit the same value (given an appropriate environment). This description is copied from the python documentation.

Usage

```
repr(x, ...)
```

Arguments

x	Object.
...	Additional arguments needed for computing the string.

Value

A string.

Examples

```
EMAIL <- new_class(class_name = "EMAIL")
email <- EMAIL$instantiate
register_method(EMAIL, ..init.. = function(address) self$address = address)

patrick <- email('patrick@test.com')
repr(patrick)
```

`repr.bandicoot_oop` *S3 method of computing the "official" string representation of a* `bandicoot_oop` *object*

Description

This function computes the "official" string representation of a `bandicoot_oop` object using the `..repr..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
repr(x, ...)
```

Arguments

`x`	`bandicoot_oop` object.
`...`	ignored.

Value

An integer.

Examples

```
EMAIL <- new_class(class_name = "EMAIL")
email <- EMAIL$instantiate
register_method(EMAIL, ..init.. = function(address) self$address = address)

patrick <- email('patrick@test.com')
repr(patrick)
```

sub_fn_body_name *Substitute a symbol in a function body*

Description

This function substitute all `old_names` with `new_names` in a function body, **and drops all the attributes**.

Usage

```
sub_fn_body_name(fn, old_name, new_name)
```

Arguments

`fn`	Function.
`old_name`	Character. Name that needs to be replaced.
`new_name`	Character. Replacement of the old name.

Value

A function.

See Also

`body()`

Examples

```
a <- function() self$x + self$y
a

sub_fn_body_name(a, "self", "this")
```

super *Get the parent class (the next class based on the method resolution order)*

Description

This function gets the parent class or the next class based on the method resolution order. This is useful when one wants to access the overwritten parent class method or the overwritten parent class attribute.

Usage

```
super(self_name = "self", mro_current_name = "..mro_current..", where = NULL)
```

Arguments

self_name	Character. The name of the self reference.
mro_current_name	Character. The name of the variable storing the current class. This is used to determine the next class.
where	Environment/Character. The target environment to search for the parent class. If where == NULL, the parent environment of self will be used. If a character value is provided, the package with the same name will be used.

Details

Note that this function assumes the parent class can be found in the parent environment of the current object. If one wants to find the parent class from a package, it needs to be specified via the where argument.

Value

A bandicoot object which is an environment.

Examples

```
# Define class O
O <- new_class(class_name = "O")
register_method(O, foo = function() {
  print("Calling class O `foo` method")
  print(paste0("Self is ", self$my_name))
  print(paste0("Next class is ", super()$..type..))
})

# Define class F
F <- new_class(O, class_name = "F")
register_method(F, foo = function() {
  print("Calling class F `foo` method")
  print(paste0("Self is ", self$my_name))
  print(paste0("Next class is ", super()$..type..))
  use_method(self, super()$foo)()
})

# Define class E
E <- new_class(O, class_name = "E")
register_method(E, foo = function() {
  print("Calling class E `foo` method")
  print(paste0("Next class is ", super()$..type..))
  use_method(self, super()$foo)()
})
```

```
# Define class D
D <- new_class(O, class_name = "D")
register_method(D, foo = function() {
  print("Calling class D `foo` method")
  print(paste0("Self is ", self$my_name))
  print(paste0("Next class is ", super()$..type..))
  use_method(self, super()$foo)()
})

# Define class C
C <- new_class(D, F, class_name = "C")
register_method(C, foo = function() {
  print("Calling class C `foo` method")
  print(paste0("Self is ", self$my_name))
  print(paste0("Next class is ", super()$..type..))
  use_method(self, super()$foo)()
})

# Define class B
B <- new_class(E, D, class_name = "B")
register_method(B, foo = function() {
  print("Calling class B `foo` method")
  print(paste0("Self is ", self$my_name))
  print(paste0("Next class is ", super()$..type..))
  use_method(self, super()$foo)()
})

# Define class A
A <- new_class(B, C, class_name = "A")
register_method(A, foo = function() {
  print("Calling class A `foo` method")
  print(paste0("Self is ", self$my_name))
  print(paste0("Next class is ", super()$..type..))
  use_method(self, super()$foo)()
})

# To understand why the order is A, B, E, C, D, F, O,
# please check [https://www.python.org/download/releases/2.3/mro/].
a <- A$instantiate()
a$my_name <- "a"
a$foo()
```

---

use_method *Use a method in an object environment*

Description

This function makes a copy of the function, then set the evaluation environment to the container of the object environment.

Usage

```
use_method(env, fn, container_name = "..method_env..")
```

Arguments

env	Environment. Object.
fn	Function. Method.
container_name	Character. Name of the container.

Value

A method.

Examples

```
TEST <- new_class(class_name = "TEST")

register_method(TEST, ..str.. = function() "test")

test <- TEST$instantiate(dist = "uniform", prm = list(a = 1, b = 2))
test$..str..()

# Use method `..str..` from BASE class
use_method(test, BASE$..str..)()
```

%contains%.bandicoot_oop

S3 method of performing membership test operator of a bandicoot_oop *object*

Description

This function performs the membership test operator using the `..contains..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %contains% y
```

Arguments

x	bandicoot_oop object.
y	Another object.

Value

A Boolean value.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..contains.. = function(y) y %in% self$name)

good_company <- company(c("patrick", "james"),
                        c(33, 34))
good_company %contains% "patrick"
```

%contains% *Membership test operator*

Description

Membership test operator

Usage

```
x %contains% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
```

```
register_method(COMPANY,
                ..contains.. = function(y) y %in% self$name)

good_company <- company(c("patrick", "james"),
                        c(33, 34))
good_company %contains% "patrick"
```

`%eq%.bandicoot_oop` *S3 method of performing the equals to operator of a* `bandicoot_oop` *object*

Description

This function performs the equals to operator using the `..eq..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %eq% y
```

Arguments

x	bandicoot_oop object.
y	Object.

Value

A Boolean value.

Examples

```
AGE <- new_class(class_name = "AGE")
age <- AGE$instantiate
register_method(AGE, ..init.. = function(current) self$current = current)
register_method(AGE, ..eq.. = function(y) self$current == y$current)

patrick <- age(33)
james <- age(33)
patrick %eq% james
```

%eq% *The equals to operator*

Description

The equals to operator

Usage

```
x %eq% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
AGE <- new_class(class_name = "AGE")
age <- AGE$instantiate
register_method(AGE, ..init.. = function(current) self$current = current)
register_method(AGE, ..eq.. = function(y) self$current == y$current)

patrick <- age(33)
james <- age(33)
patrick %eq% james
```

%-=%.bandicoot_oop *S3 method of in-place subtraction operator of a* bandicoot_oop *object*

Description

This function performs the in-place subtraction operator using the `..iadd..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %-=% y
```

Arguments

x	bandicoot_oop object.
y	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..isub.. = function(y) {
                  self$age <- self$age[self$name != y]
                  self$name <- self$name[self$name != y]
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
good_company %-=% "patrick"
good_company$name
```

%-=% *In-place subtraction operator*

Description

In-place subtraction operator

Usage

```
x %-=% y
```

Arguments

x	Object.
y	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..isub.. = function(y) {
                  self$age <- self$age[self$name != y]
                  self$name <- self$name[self$name != y]
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
good_company %-=% "patrick"
good_company$name
```

`%ge%.bandicoot_oop` *S3 method of performing the greater or equals operator of a* `bandicoot_oop` *object*

Description

This function performs the greater or equals operator using the `..ge..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %ge% y
```

Arguments

`x`	`bandicoot_oop` object.
`y`	Object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..ge.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name >= y$first_name)
                  }
                  return(self$last_name >= self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %ge% james
```

`%ge%` *The greater or equals to operator*

Description

The greater or equals to operator

Usage

```
x %ge% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
```

```
                })
register_method(NAME,
                ..ge.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name >= y$first_name)
                  }
                  return(self$last_name >= self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %ge% james
```

`%-%.bandicoot_oop` *S3 method of subtraction operator of a* `bandicoot_oop` *object*

Description

This function performs the subtraction operator using the `..sub..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %-% y
```

Arguments

x	`bandicoot_oop` object.
y	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..sub.. = function(y) {
                  company(self$name[self$name != y],
                          self$age[self$name != y])
```

```
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
new_company <- good_company %-% "patrick"
new_company$name
```

`%-%` *Subtraction operator*

Description

Subtraction operator

Usage

```
x %-% y
```

Arguments

x	Object.
y	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..sub.. = function(y) {
                  company(self$name[self$name != y],
                          self$age[self$name != y])
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
new_company <- good_company %-% "patrick"
new_company$name
```

`%gt%.bandicoot_oop` *S3 method of performing the greater than operator of a* `bandicoot_oop` *object*

Description

This function performs the greater than operator using the `..gt..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %gt% y
```

Arguments

`x`	`bandicoot_oop` object.
`y`	Object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..gt.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name > y$first_name)
                  }
                  return(self$last_name > self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %gt% james
```

`%gt%` *The greater than operator*

Description

The greater than operator

Usage

```
x %gt% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..gt.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name > y$first_name)
                  }
                  return(self$last_name > self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %gt% james
```

`%le%.bandicoot_oop` *S3 method of performing the less or equals operator of a* `bandicoot_oop` *object*

Description

This function performs the less or equals operator using the `..le..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %le% y
```

Arguments

`x`	`bandicoot_oop` object.
`y`	Object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..le.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name <= y$first_name)
                  }
                  return(self$last_name <= self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %le% james
```

`%le%` *The less or equals to operator*

Description

The less or equals to operator

Usage

```
x %le% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..le.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name <= y$first_name)
                  }
                  return(self$last_name <= self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %le% james
```

`%lt%.bandicoot_oop` *S3 method of performing the less than operator of a* `bandicoot_oop` *object*

Description

This function performs the less than operator using the `..lt..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %lt% y
```

Arguments

x	bandicoot_oop object.
y	Object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..lt.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name < y$first_name)
                  }
                  return(self$last_name < self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %lt% james
```

`%lt%` *The less than operator*

Description

The less than operator

Usage

```
x %lt% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
NAME <- new_class(class_name = "NAME")
name <- NAME$instantiate
register_method(NAME,
                ..init.. = function(first_name, last_name) {
                  self$first_name = first_name
                  self$last_name = last_name
                })
register_method(NAME,
                ..lt.. = function(y) {
                  if (self$last_name == y$last_name) {
                    return(self$first_name < y$first_name)
                  }
                  return(self$last_name < self$last_name)
                })

patrick <- name("Patrick", "Li")
james <- name("James", "Li")
patrick %lt% james
```

%ne%.bandicoot_oop *S3 method of performing the not equals to operator of a* bandicoot_oop *object*

Description

This function performs the not equals to operator using the `..ne..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %ne% y
```

Arguments

x	bandicoot_oop object.
y	Object.

Value

A Boolean value.

Examples

```
AGE <- new_class(class_name = "AGE")
age <- AGE$instantiate
register_method(AGE, ..init.. = function(current) self$current = current)
register_method(AGE, ..ne.. = function(y) self$current != y$current)

patrick <- age(33)
james <- age(33)
patrick %ne% james
```

%ne% *The not equals to operator*

Description

The not equals to operator

Usage

```
x %ne% y
```

Arguments

x	Object.
y	Another object.

Value

A Boolean value.

Examples

```
AGE <- new_class(class_name = "AGE")
age <- AGE$instantiate
register_method(AGE, ..init.. = function(current) self$current = current)
register_method(AGE, ..ne.. = function(y) self$current != y$current)

patrick <- age(33)
james <- age(33)
patrick %ne% james
```

%+=%.bandicoot_oop *S3 method of in-place addition operator of a* bandicoot_oop *object*

Description

This function performs the in-place addition operator using the `..iadd..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %+=% y
```

Arguments

x	bandicoot_oop object.
y	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..iadd.. = function(y) {
                  self$name <- c(self$name, y$name)
                  self$age <- c(self$age, y$age)
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
bad_company <- company(c("pat", "jam"),
                       c(3, 4))
good_company %+=% bad_company
good_company$name
```

`%+=%` *In-place addition operator*

Description

In-place addition operator

Usage

```
x %+=% y
```

Arguments

`x`	Object.
`y`	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
```

```
                  self$age <- age
                })
register_method(COMPANY,
                ..iadd.. = function(y) {
                  self$name <- c(self$name, y$name)
                  self$age <- c(self$age, y$age)
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
bad_company <- company(c("pat", "jam"),
                       c(3, 4))
good_company %+=% bad_company
good_company$name
```

`%+%.bandicoot_oop` *S3 method of addition operator of a* `bandicoot_oop` *object*

Description

This function performs the addition operator using the `..add..()` method. If it is not applicable, error will be raised.

Usage

```
## S3 method for class 'bandicoot_oop'
x %+% y
```

Arguments

`x`	`bandicoot_oop` object.
`y`	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..add.. = function(y) {
```

```
                  company(c(self$name, y$name),
                          c(self$age, y$age))
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
bad_company <- company(c("pat", "jam"),
                       c(3, 4))
new_company <- good_company %+% bad_company
new_company$name
```

`%+%` *Addition operator*

Description

Addition operator

Usage

```
x %+% y
```

Arguments

x	Object.
y	Another object.

Value

Depends on the method.

Examples

```
COMPANY <- new_class(class_name = "COMPANY")
company <- COMPANY$instantiate
register_method(COMPANY,
                ..init.. = function(name, age) {
                  self$name <- name
                  self$age <- age
                })
register_method(COMPANY,
                ..add.. = function(y) {
                  company(c(self$name, y$name),
                          c(self$age, y$age))
                })

good_company <- company(c("patrick", "james"),
                        c(33, 34))
```

```
bad_company <- company(c("pat", "jam"),
                       c(3, 4))
new_company <- good_company %+% bad_company
new_company$name
```

Index