



## Syntax

```
<program>      := <class-decl>* <expr>

<class-decl>   := class <id> extends <id>
                <field-decl>*
                <method-decl>*

<field-decl>   := field <id>

<method-decl> := method <id> (<id>*,) <expr>

<expr> := new <id> (<expr>*,)
<expr> := send <expr> <id> (<expr>*,)
<expr> := super <id> (<expr>*,)
```

## Example

```
class fish extends object
  field size
  method initialize (s) set size = s
  method get_size() size
  method grow(food)
    set size = +(size, food)
  method eat(other_fish)
    let s = send other_fish get_size()
    in send self grow(s)

let f = new fish(10)
in begin
  send f grow(2);
  send f get_size()
end
```

## Example

```
class fish extends object
  field size
  method initialize (s) set size = s
  method get_size() size
  method grow(food)
    set size = +(size, food)
  method eat(other_fish)
    let s = send other_fish get_size()
    in send self grow(s)
class colorfish extends fish
  field color
  method set_color(c) set color = c
  method get_color() color
...
```

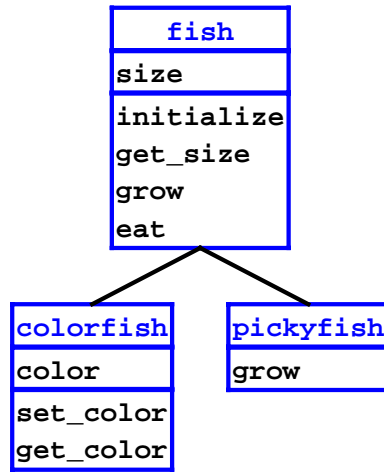
## Example

```
class fish extends object
  field size
  method initialize (s) set size = s
  method get_size() size
  method grow(food)
    set size = +(size, food)
  method eat(other_fish)
    let s = send other_fish get_size()
    in send self grow(s)
...
class pickyfish extends fish
  method grow(food)
    super grow(-(food, 1))
...
```

## Class Tree

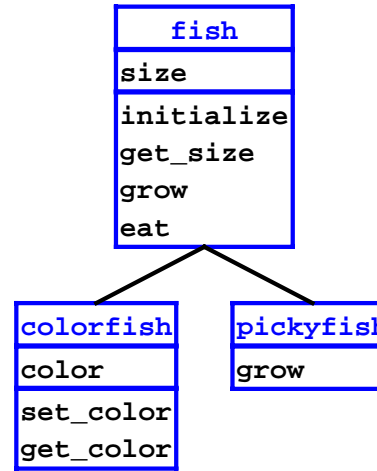
```
class fish ...  
  
class colorfish  
  extends fish  
  ...  
  
class pickyfish  
  extends fish  
  ...
```

⇒



## Evaluation

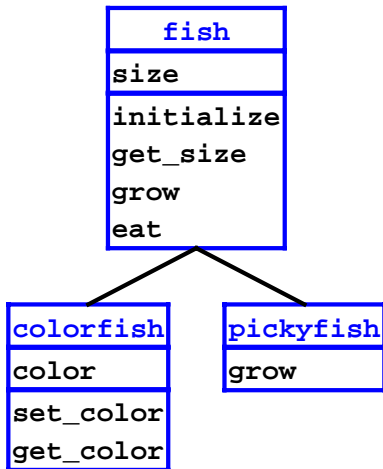
```
new colorfish(1)
```



## Evaluation

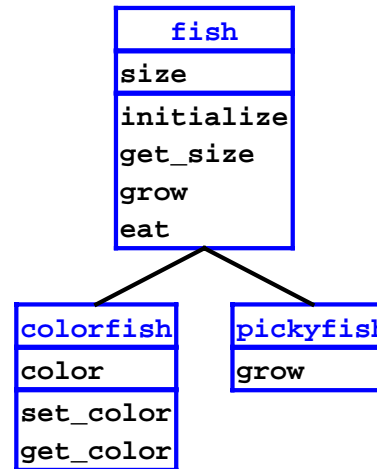
```
new colorfish(1)
```

```
obj = colorfish  
size = 1  
color = 0
```

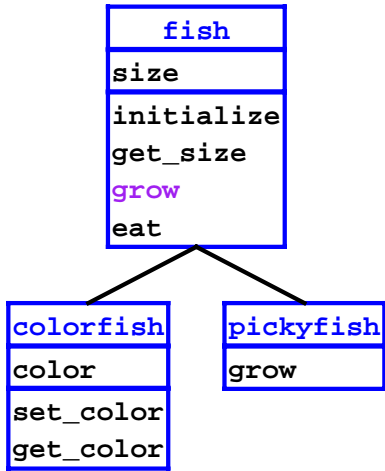


## Evaluation

```
let  
  o1 = new colorfish(3)  
in begin  
  send o1 grow(4);  
  send o1 get_size()  
end
```



### Evaluation



```

grow(f)
set size=+(size,f)
  
```

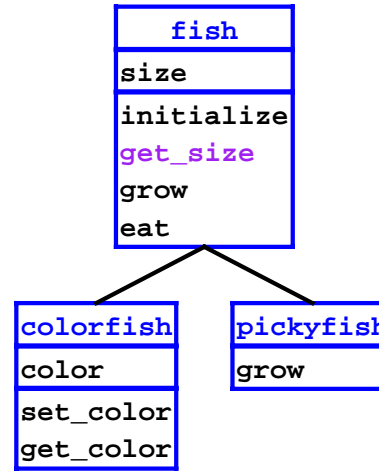
```

let
o1 = new colorfish(3)
in begin
send o1 grow(4);
send o1 get_size()
end
  
```

```

o1 = colorfish
size = 3
color = 0
  
```

### Evaluation



```

get_size() size
  
```

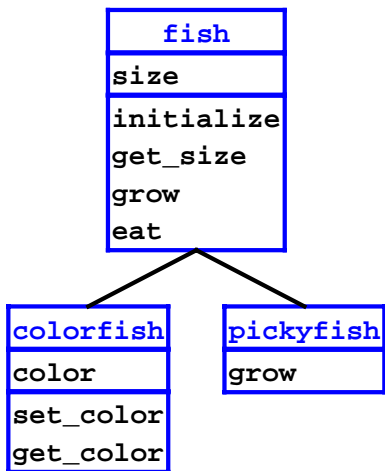
```

let
o1 = new colorfish(3)
in begin
send o1 grow(4);
send o1 get_size()
end
  
```

```

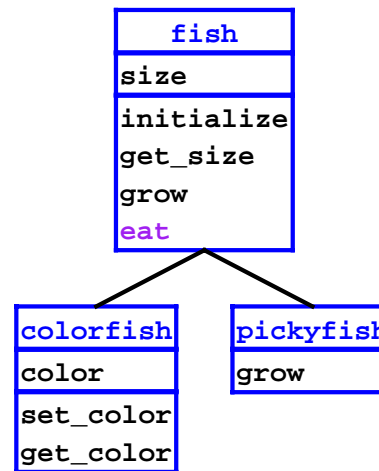
o1 = colorfish
size = 7
color = 0
  
```

### Evaluation



```

let
o1 = new colorfish(3)
o2 = new pickyfish(6)
in begin
send o2 eat(o1);
send o2 get_size()
end
  
```



```

eat(o) let
s = send o get_size()
in send self grow(s)
  
```

### Evaluation

```

let
o1 = new colorfish(3)
o2 = new pickyfish(6)
in begin
send o2 eat(o1);
send o2 get_size()
end
  
```

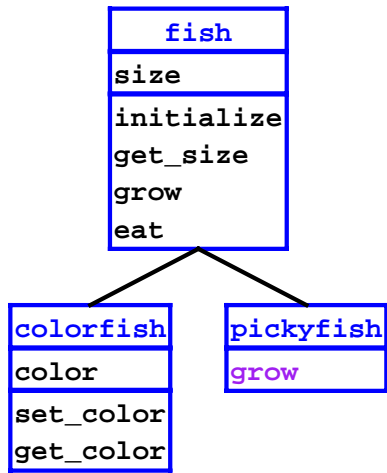
```

o1 = colorfish
size = 3
color = 0
  
```

```

o2 = pickyfish
size = 6
  
```

## Evaluation



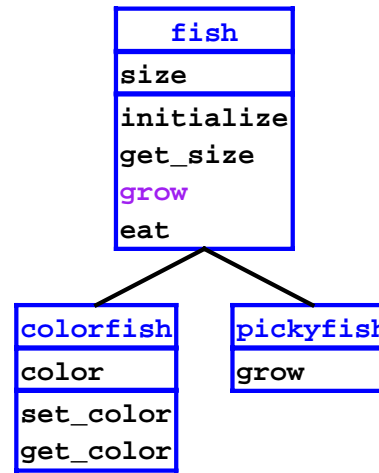
```
grow(f)
super grow(-(f, 1))
```

```
let
o1 = new colorfish(3)
o2 = new pickyfish(6)
in begin
send o2 eat(o1);
send o2 get_size()
end
```

```
o1 = colorfish
size = 3
color = 0
```

```
o2 = pickyfish
size = 6
```

## Evaluation



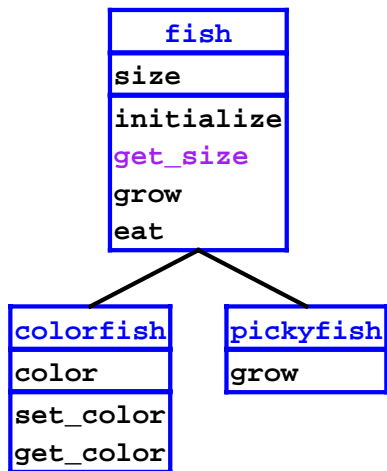
```
grow(f)
set size+=(size,f)
```

```
let
o1 = new colorfish(3)
o2 = new pickyfish(6)
in begin
send o2 eat(o1);
send o2 get_size()
end
```

```
o1 = colorfish
size = 3
color = 0
```

```
o2 = pickyfish
size = 6
```

## Evaluation



```
get_size() size
```

```
let
o1 = new colorfish(3)
o2 = new pickyfish(6)
in begin
send o2 eat(o1);
send o2 get_size()
end
```

```
o1 = colorfish
size = 3
color = 0
```

```
o2 = pickyfish
size = 8
```

## Interpreter

- Build class tree

```
(define eval-program
(lambda (pgm)
(cases program pgm
(a-program (c-decls exp)
(elaborate-class-decls! c-decls)
(eval-expression exp (init-env))))))
```

## Interpreter

- Expression form: object creation

```
(new-object-exp (class-name rands)
  (let ((args (eval-rands rands env))
        (obj (new-object class-name)))
    (find-method-and-apply
     'initialize class-name obj args)
    obj))
```

## Interpreter

- Expression form: method call

```
(method-app-exp (obj-exp method-name rands)
  (let ((args (eval-rands rands env))
        (obj (eval-expression obj-exp env)))
    (find-method-and-apply
     method-name (object->class-name obj)
     obj args)))
```

## Interpreter

- Expression form: super call

```
(super-call-exp (method-name rands)
  (let ((args (eval-rands rands env))
        (obj (apply-env env 'self)))
    (find-method-and-apply
     method-name (apply-env env '%super)
     obj args)))
```

## Interpreter: To Do

- Build tree given class declarations
- Implement object representation
- Implement method finding and invocation