

09/08/22

Most enroute clients are OOP oriented

## UML

We use this diagram to communicate with other developers about the app.

## OOP

Paradigma de la programación. Permite usar piezas de código y usarlas, estas se llaman clases.

### Class Grouping

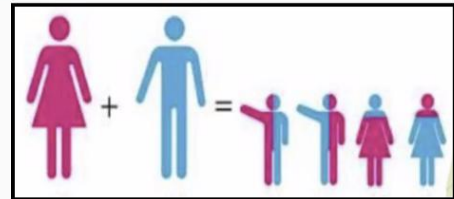
- What to do???



- Well... GROUP related information!!!

- Create a parent class
- Create a child that **inherits**
- Add unique attributes to unique children (**Polymorphism**)
- Instantiate Objects

- Grouping the related data and behaviors together to form a simple template then creating subgroups for specialized data and behavior.



If a child inherits a father, then these two classes are a group.  
Do not think so much as a group, but as a hierarchy.

### Four (main) Blocks of OOP – A PIE

- Classes: User defined data types that work as the blueprint for repeated code. (TEMPLATE)

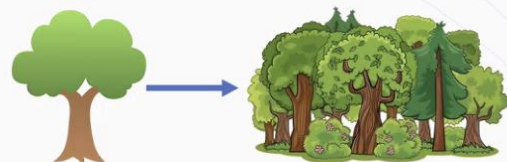


- Attributes: stored information that describes an object.

- eye color=blue



- Objects: Instances of Classes. Use Specifics



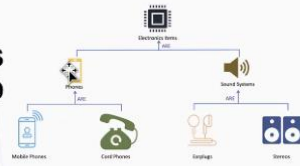
- Methods: Behaviors. Methods perform actions
- Return information
- Update information
- Do Something!
- Defined in class



# Four Principles of OOP

- **Inheritance:** parent classes extend attributes and behaviors to child classes.

- Inheritance supports reusability.



- **Encapsulation:** Contain info and display only that which is needed

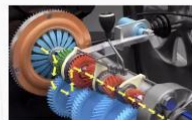
- Adds security
- Private vs Public



- **Abstraction:** User interacts with only selected attributes and methods of an object

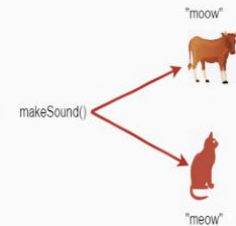
- `car.gear.shiftUp()`

- [VIDEO](#)



- **Polymorphism:** Share behaviors, BUT children can change!!!

- Overloading: if you give me this I'll do that, if not I can do something else...
- Overriding: I'm different to mom and dad!



Protected ones means that the child classes can access them.

## Abstract

Only the class knows how it works. All the functionality is abstracted through a method name.

## Overloading and overloading

### Overloading:

3 methods with the same name but different inputs.

### Overriding

Replace a method with another one.

# Ruby OOP

Everything on ruby is an object.

## Class Variable

`@@variable_name` -> shared by classes and classes children.

They are like static variables from java.

Both the person and child classes are gonna inherit the original value.

`Person.age = 20`

`Employee.age` -> 20

## Inheritance

- Note the `<` symbol. The parent class is located at the right.

```
class NewBankAccount < BankAccount

  def customerPhone
    @customerPhone
  end

  def customerPhone=( value )
    @customerPhone = value
  end

end
```



## Super

Calls the father version of the same method.

```

1.  class Animal
2.      def name
3.          puts "Animal"
4.      end
5.  end
6.
7.  class Cat < Animal
8.      def name
9.          super
10.     end
11. end
12.
13. cat = Cat.new
14. cat.name
15.
16. # "Animal"

```

Super with parenthesis vs without parenthesis:

```

def ok(parameter_one, parameter_two, parameter_three)
  super
  puts "hello from paco"
end
end

```

super is going to send the three params from the method.

```

def ok(parameter_one, parameter_two, parameter_three)
  super(parameter_one, parameter_two)
  puts "hello from paco #{ parameter_one }, #{ parameter_two } #{ parameter_three }"
end

```

This only happens on super -> special feature

## Overloading

Splat arguments... -> he will explain the later.

\*arg

You can do it manually with if clauses and depending on that check of the argument, it will have different behavior.

```
def method(*args)
  case args.size
  when 1
    puts "this is overloading with one parameter"
  when 2
    puts "the second case with two parameters"
  else
    puts "no method found"
  end
end
```

**Sad news, in ruby, there is no Overloading in ruby.**

You have to check the params manually.

=begin ????????

=end

## Protected methods


```
class Person
  protected

  def gimme_your_credit_card! # protected method
    puts "Fine. Whatever. Here it is: 1234-4567-8910"
  end
end

class Rib < Person
end

class Wife < Rib # wife inherits from Rib
  def i_am_buying_another_handbag_with_your_card(husband)
    husband.gimme_your_credit_card!
  end
end

@husband = Person.new
@mrs = Wife.new
@mrs.i_am_buying_another_handbag_with_your_card(@husband)
# => puts "Fine. Whatever. Here it is: 1234-4567-8910"
```



...



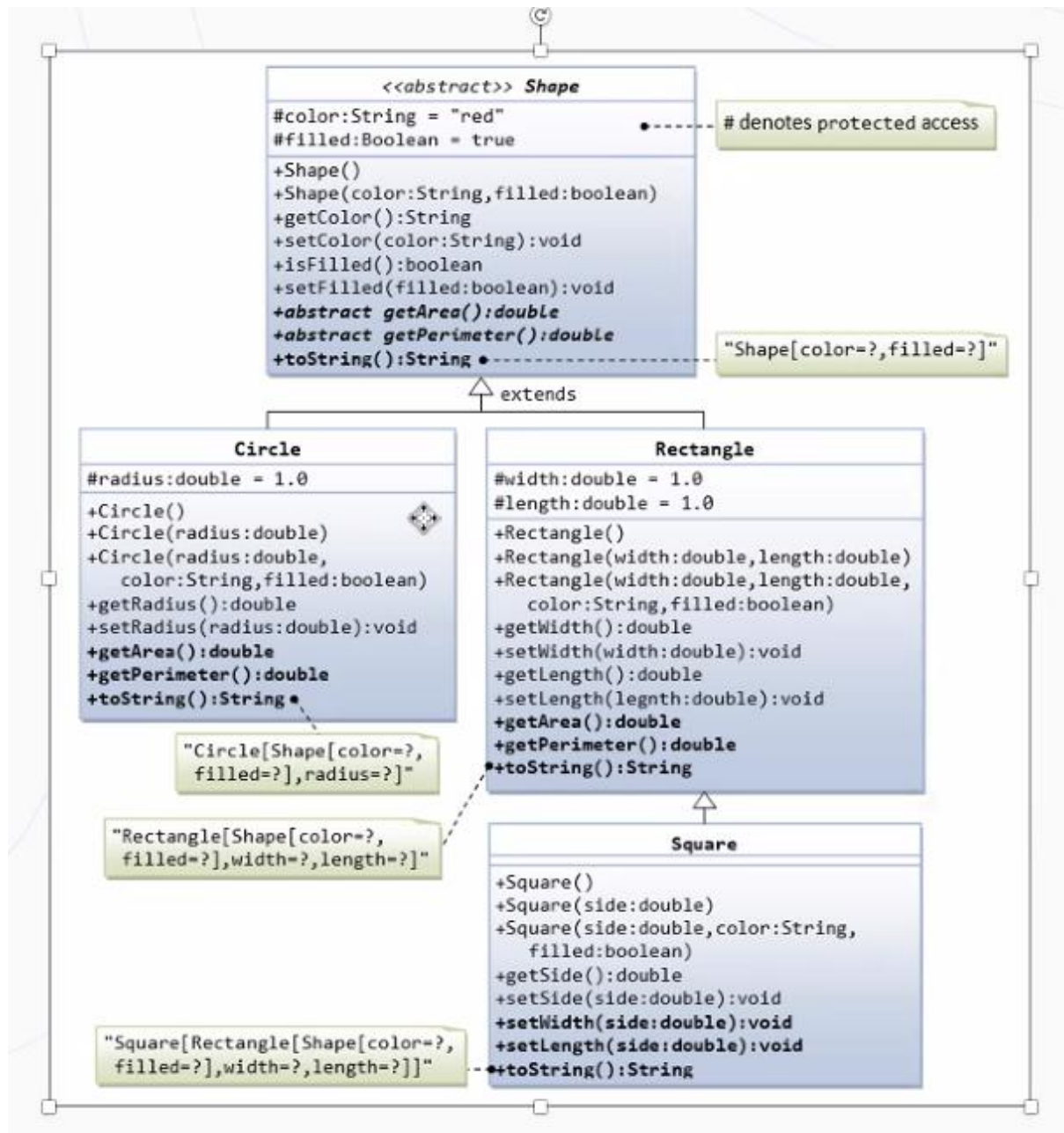
## Public methods

Can be accessed outside and inside the class.

## Private methods

Not very common on ruby

Can only be accessed inside the class.



There are no interfaces or abstract methods.

Crear también git repository privado con acceso al maestro, para ir subiendo todas las notas e ir revisando los commits que hacemos por días.

Poner solo ejercicios, pero si igual queremos poner notas.  
Ir creando el repo hoy de preferencia aunque no tenga acceso el.