Baboosh - OOP Bash

Using OOP in Bash

About

Baboosh is a simple script to include (using "source" command or "." command) into your script to permit to write your script with pseudo Oriented Object syntax.

Baboosh implements class with "one level" inheritance.

Importing

In you script, you can use

```
#!/bin/bash
. /path/to/baboosh.sh
```

If baboosh is in you script path, try this

```
#!/bin/bash
. ./baboosh.sh

#or
. $(dirname $0)/baboosh.sh
```

This includes scripts and set the "new" function.

Creating classes

class definition

To create a class, you need to use a list. Each element is formed with: "type name".

Types can be:

- function: implements a method
- var : implements a property
- extends : set inheritance

> **Caution!**
>
> "extends" must be the first elements on list !

To create a human class, you can use this

```
Human=(
    function birth
    function die
    function eat
```

```
    function sleep
    var name
)
```

Methods declaration

You must implement methods, this way

```
Human::birth(){
    #getting "this" reference
    local this=$1; shift
    #each time you need $this, use an eval... see Tips section
}

Human::die(){
    #...
}

Human::eat(){
    #...
}

Human::sleep(){
    #...
}
```

To use "$this" please see Tips section.

Now, to instantiate a human named "john" is pretty simple

```
new Human john
```

"john" can birth, die, eat or sleep, like that:

```
john.birth
john.sleep
jonh.eat
john.die
```

Using properties

To set values, automatic setters should be used

```
john.set_name "John"
```

This way, "name" property is set to "John".

Properties are accessible by "eval" (for now...)

```
echo $(john.name)
```

Keep in mind that property is in fact an alias to an "echo command". Calling "john.name" will do "echo "John"".

Constructor

Constructor should not be declared in definition list, this is a special function named "__init__". You only have to implement

```
Human::__init__(){
    #here is a constructor
}
```

> **Note**
>
> inherited child class will call parent constructor implicitly.

Inheritance

It's possible to extend classes. For example, an Employee is an Human, so

```
Employee=(
    extends Human
    function work
)

Employee::work(){
    echo "working..."
}
```

Now, Employee can birth, eat, sleep and die as Human declared those functions. Employee has got a name, as declared into Human class.

> **Caution!**
>
> Limitations
>
> - "extends" must be the **very first** element in declaration list
> - redeclaring a method overwrites the parent method. There is no way to access "parent"

As explained in Constructor section, Human

Tips

Remember to use "$(...)" to get vars, this is easier to work with values

```
the_name=$(john.name)
```

Inside methods, "this" if passed as first argument, so you need to do

```
local this=$1; shift
```

"shift" is used to unset "$1".

"this" is now a variable unlike "john" which is an alias. So, to play with properties, do that

```
#set property
eval $this.set_name "Other"

#read property
prop=$(eval $this.prop)

#call method
eval $this.methodName
```

Copyright

- Documentation is under LGPLv3
- Baboosh scripts are under BSD Licence