

French PowerShell Saturday



PowerShell Classes Part2: Advanced Concepts and feedback from the field

Stéphane van Gulick

15 Septembre 2018 - #FrPwshSat2018



French PowerShell Saturday

Merci à nos sponsors !!



 METSYS


smartview
conseil et formation

15 Septembre 2018 - #FrPwshSat2018



Introduction

15 Septembre 2018 - #FrPwshSat2018

Introduction



```

1 Function Get-Speaker {
2     $speaker =[Ordered] @{
3         "Name" = "van Gulick"
4         "FirstName" = "Stéphane"
5         "WorkPlace" = "Bern"
6         "isMVP" = $true
7     }
8     return New-Object psobject -Property $speaker
9 }
10
11
12
13 Get-Speaker
14

```

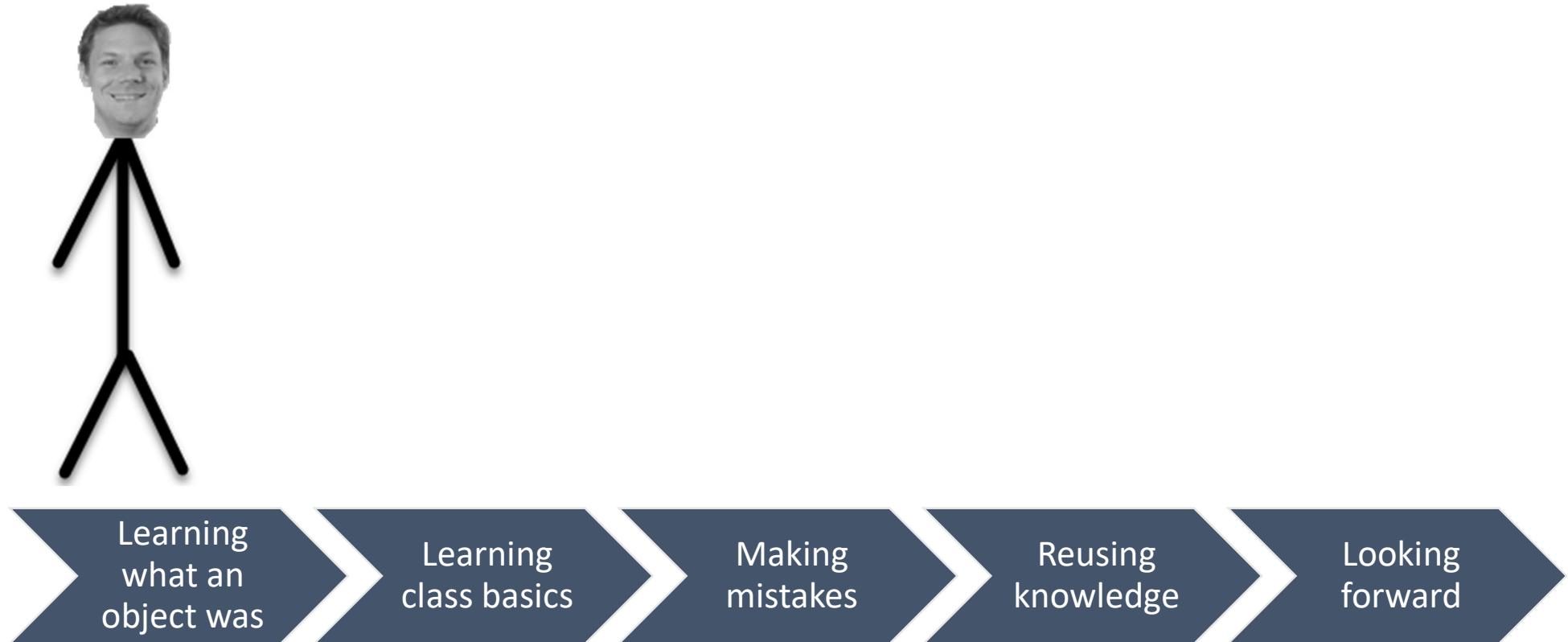
PROBLEMS TERMINAL ... 1: PowerShell Integrated + ⌂ ⌂

Name	FirstName	WorkPlace	isMVP
van Gulick	Stéphane	Bern	True

15 Septembre 2018 - #FrPwshSat2018



Plan:

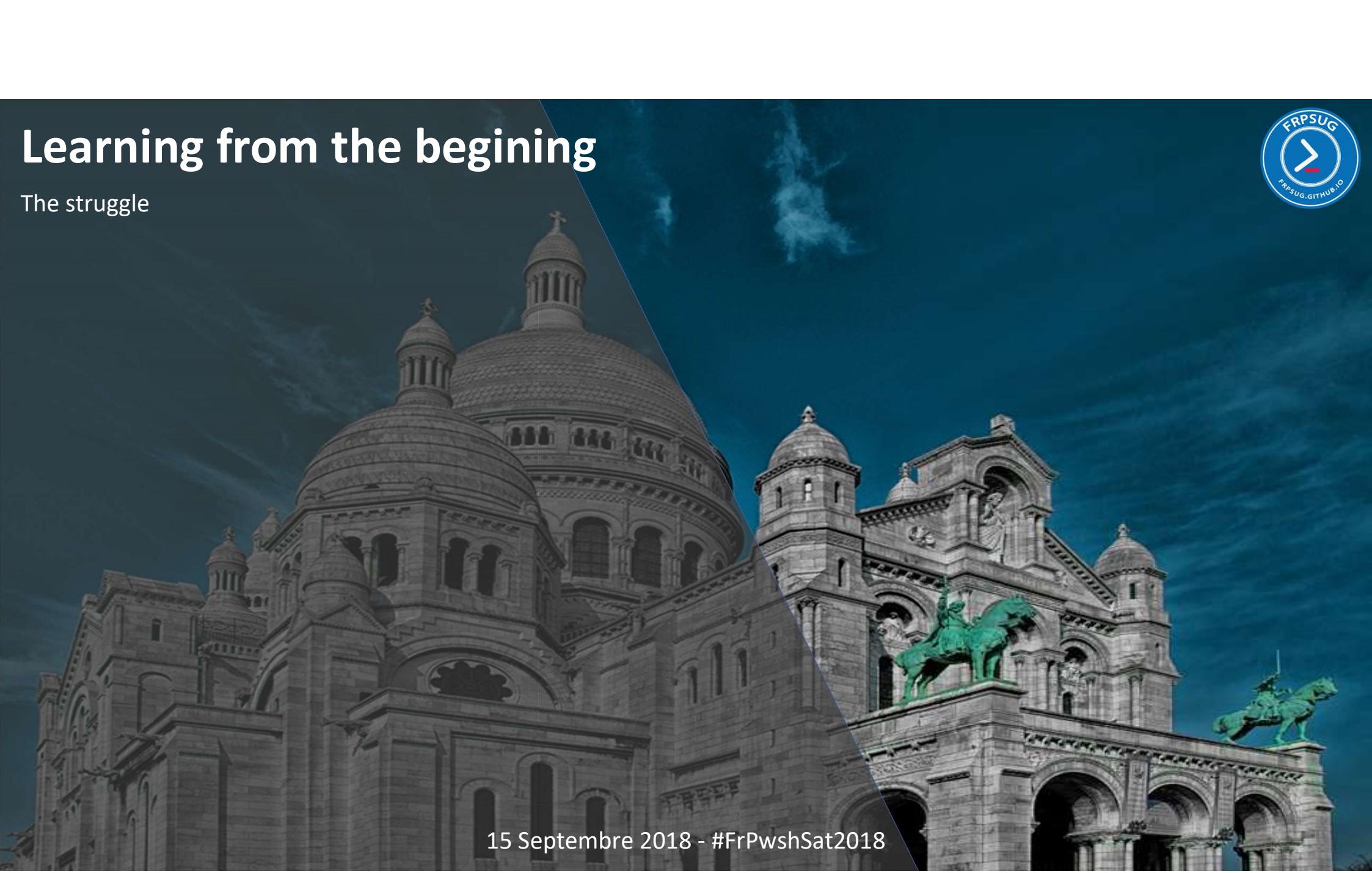


15 Septembre 2018 - #FrPwshSat2018



Learning from the begining

The struggle



A large image of the Sacré-Cœur cathedral in Paris is split diagonally. The left side is in grayscale, representing the 'struggle' or initial phase. The right side is in color, showing the cathedral's iconic green statues and blue sky, representing 'learning from the beginning'.

15 Septembre 2018 - #FrPwshSat2018

Introduction: assimilating class concepts



Function

```
1 Function User {
2     [CmdletBinding()]
3     Param(
4
5         [Parameter(Mandatory=$true)][String]$FirstName,
6         [Parameter(Mandatory=$true)][String]$LastName
7
8     )
9
10    $Hash = @{
11        FirstName = $FirstName
12        LastName = $LastName
13        FullName = $Name + " " + $LastName
14    }
15
16    return New-object psobject -Property $Hash
17
18 }
19
20 user -FirstName "Stéphane" -LastName "van Gulick"
```

Class

```
Class User {
    #Properties
    [String]$FirstName
    [String]$LastName
    [String]$FullName

    #Constructor

    User ([String]$FirstName,[String]$LastName){
        $this.FirstName = $FirstName
        $this.LastName = $LastName
        $this.SetFullName()
    }

    #Methods

    [void]SetFullName(){
        $this.FullName = $this.Name + " " + $this.LastName
    }

    [String]GetFullName(){
        return $this.FullName
    }
}
```

Click!



```
$Date = Get-Date  
$NewTime = $Date.AddDays(5)  
$NewTime.Day
```



15 Septembre 2018 - #FrPwshSat2018

Class Basics

```
00_BasicClass.ps1 x
1 Class User {
2     #Properties
3         [String]$Name
4         [String]$LastName
5         [String]$FullName
6
7     #Constructor
8
9     User ([String]$FirstName, [String]$LastName){
10        $this.Name = $FirstName
11        $this.LastName = $LastName
12        $this.SetFullName()
13    }
14
15    #Methods
16
17    [void]SetFullName(){
18        $this.FullName = $this.Name + " " + $this.LastName
19    }
20
21    [String]GetFullName(){
22        return $this.FullName
23    }
24 }
25 [User]::New("Stephane", "van Gulick")
```

- **Properties:**
 - Holds information on the object
- **Constructor:**
 - Creates your object
- **Methods:**
 - Adds functionality to your object (Like a Function)
- **Creating an object (Instantiate):**
 - New-Object ClassName
 - [ClassName]::New()

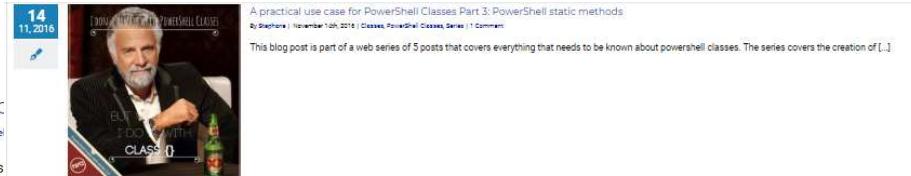


Plan:



A practical use case for PowerShell Classes
By Stephane | October 18th, 2016 | Powershell 5, PowerShell

In part two of this web series about powershell use a powershell enum and go through a us



A practical use case for PowerShell Classes Part 3: PowerShell static methods

By Stephane | November 10th, 2016 | Classes, PowerShell Classes, Series | 1 Comment

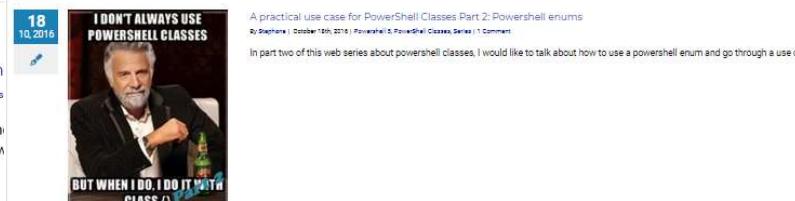
This blog post is part of a web series of 5 posts that covers everything that needs to be known about powershell classes. The series covers the creation of [...]



French PowerShell Meetup n°2 – Wrap up

By Stephane | October 20th, 2016 | French, PowerShell Classes, UserGroup | 0 Comments

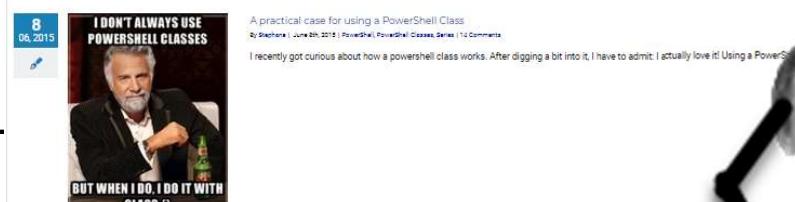
Not so long ago, we have founded the French PowerShell user group. The second meetup was held yesterday, where I did a presentation on PowerShell Classes (Read more about [...])



A practical use case for PowerShell Classes Part 2: Powershell enums

By Stephane | October 18th, 2016 | Powershell 5, PowerShell Classes, Series | 1 Comment

In part two of this web series about powershell classes, I would like to talk about how to use a powershell enum and go through a use case which [...]



A practical case for using a PowerShell Class

By Stephane | June 6th, 2015 | Powershell, PowerShell Classes, Series | 12 Comments

I recently got curious about how a powershell class works. After digging a bit into it, I have to admit: I actually love it! Using a Powe



15 Septembre 2018 - #FrPwshSat2018



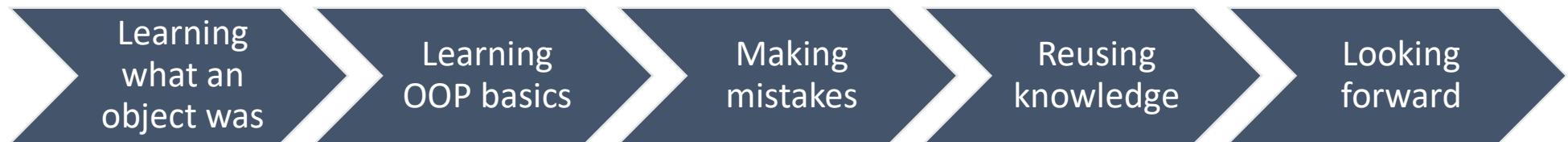
Plan:

\$False

15 Septembre 2018 - #FrPwshSat2018



Plan:



15 Septembre 2018 - #FrPwshSat2018



OOP Concepts

Programming guidelines

15 Septembre 2018 - #FrPwshSat2018

OOP: What's that?



Object Oriented Programming



15 Septembre 2018 - #FrPwshSat2018

4 pillars of OOP:



Abstraction



Encapsulation



Inheritance



Polymorphism

4 Pillars of OOP: Abstraction



Abstraction

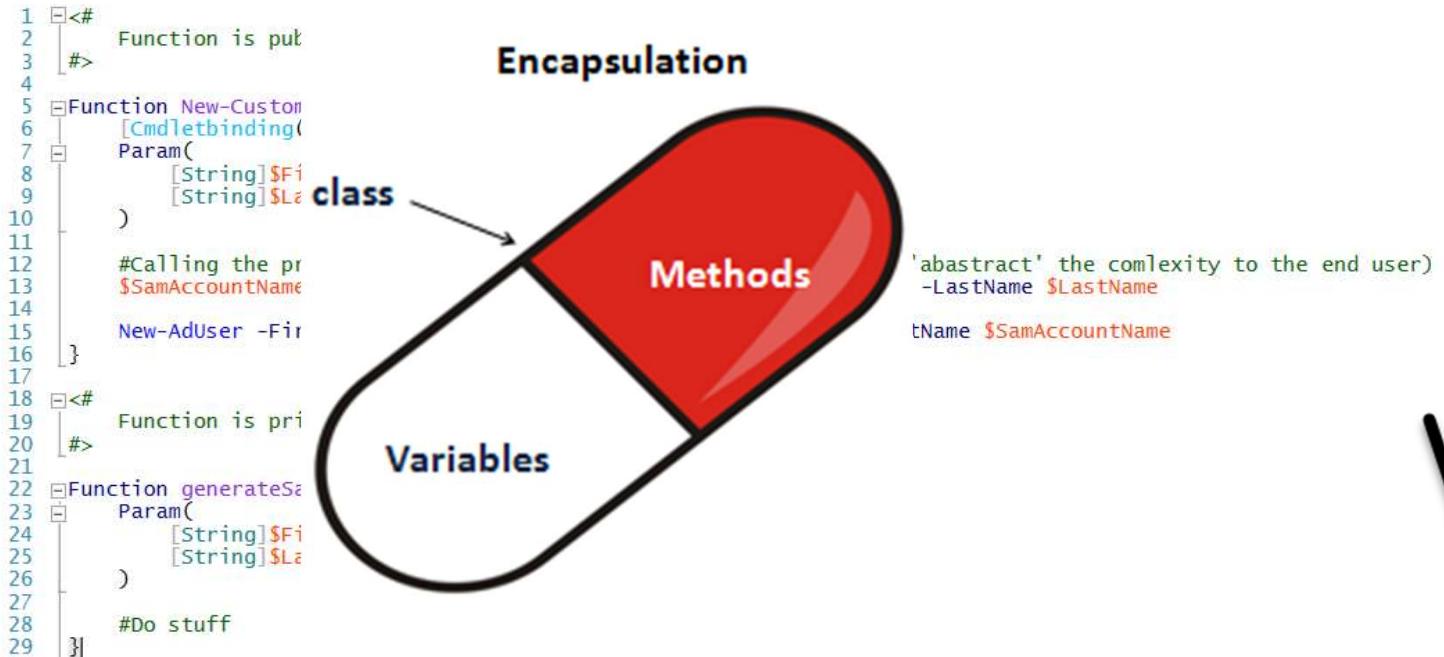
```
1735 Function Test-PackageDocument {  
1736     <#...  
1737     #>  
1738     [CmdletBinding()]  
1739     Param(  
1740         [Parameter(Mandatory=$true)]  
1741         [PackageDocument]$Document  
1742     )  
1743     return $Document.ValidateDocument()  
1744 }  
1745 }
```



4 Pillars of OOP: encapsulation



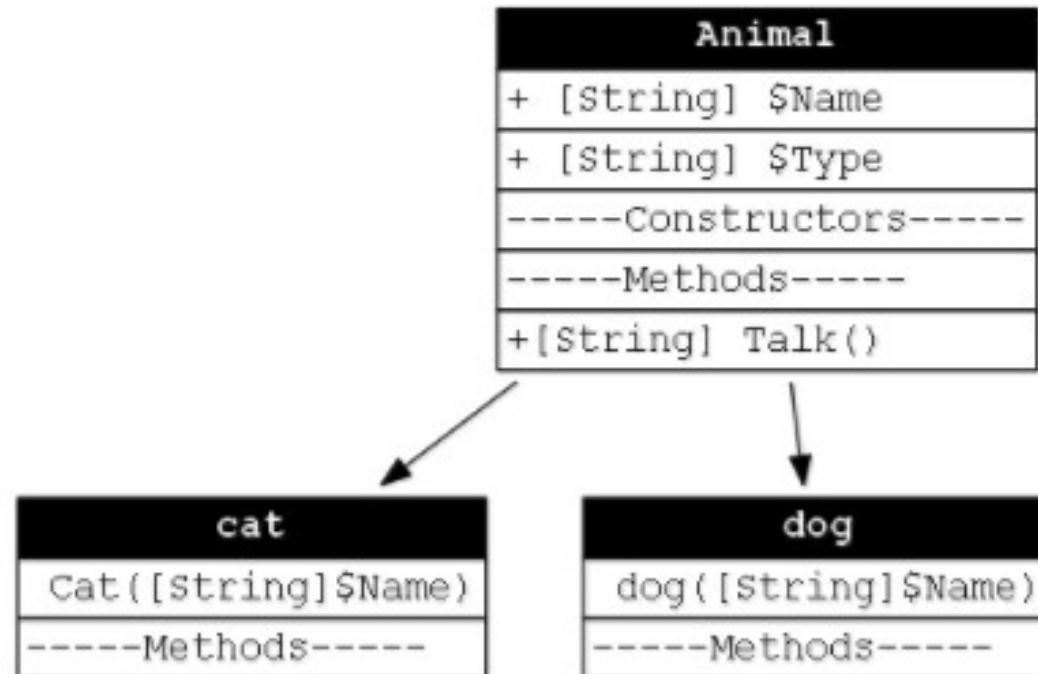
encapsulation



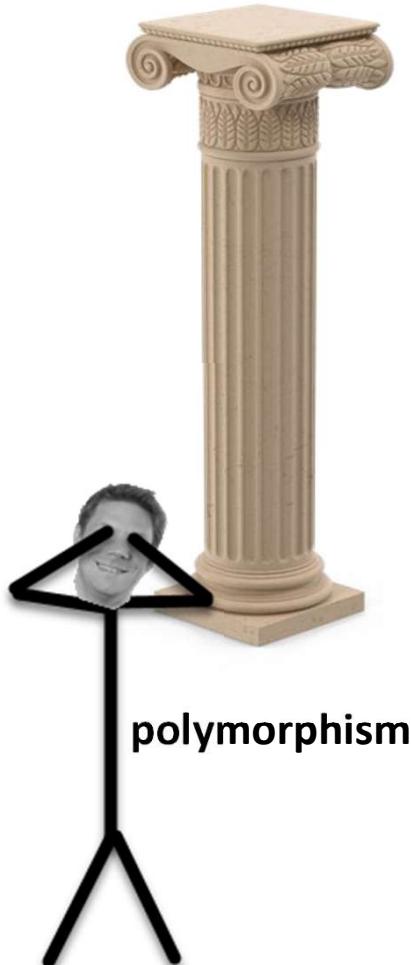
4 Pillars of OOP: inheritance



inheritance

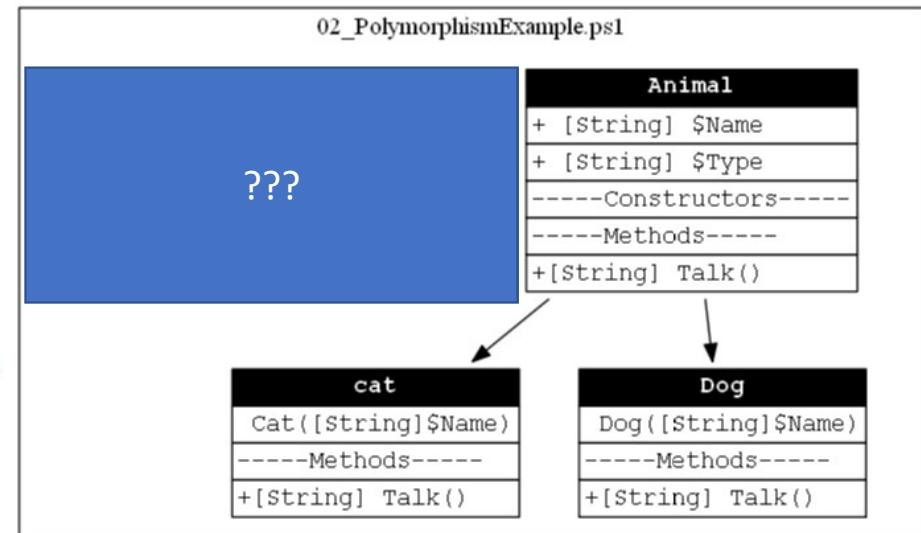


4 Pillars of OOP: polymorphism



```

1  Class Animal {
2      [String]$Name
3      [String]$Type
4
5      [String]Talk(){
6          Return "Simple Animal"
7      }
8
9
10 Class cat : Animal {
11     Cat([String]$Name){
12         $this.Name = $Name
13         $this.Type = "Cat"
14     }
15
16     [String]Talk(){
17         Return "Meow! I am a $($this.Type) and my Name is $($this.Name)"
18     }
19
20
21 Class Dog : Animal {
22     Dog([String]$Name){
23         $this.Name = $Name
24         $this.Type = "Dog"
25     }
26
27     [String]Talk(){
28         Return "Woof! I am a $($this.Type) and my Name is $($this.Name)."
29     }
30
31 }
32
33
34
35
36 $Array = @()
37 $Array += [Cat]::new("Minnie")
38 $Array += [Cat]::new("Jerry")
39 $Array += [Dog]::new("Rex")
40 $Array += [Dog]::new("Goofy")
41
42 $NohasBoat = [Boat]::New($Array)
43
44 Foreach($Passenger in $NohasBoat.Passengers){
45     $Passenger.Talk()
46 }
  
```



```

$NohasBoat = [Boat]::New($Array)

Foreach($Passenger in $NohasBoat.Passengers){
    $Passenger.Talk()
}

Meow! I am a Cat and my Name is Minnie
Meow! I am a Cat and my Name is Jerry
Woof! I am a Dog and my Name is Rex.
Woof! I am a Dog and my Name is Goofy.
  
```



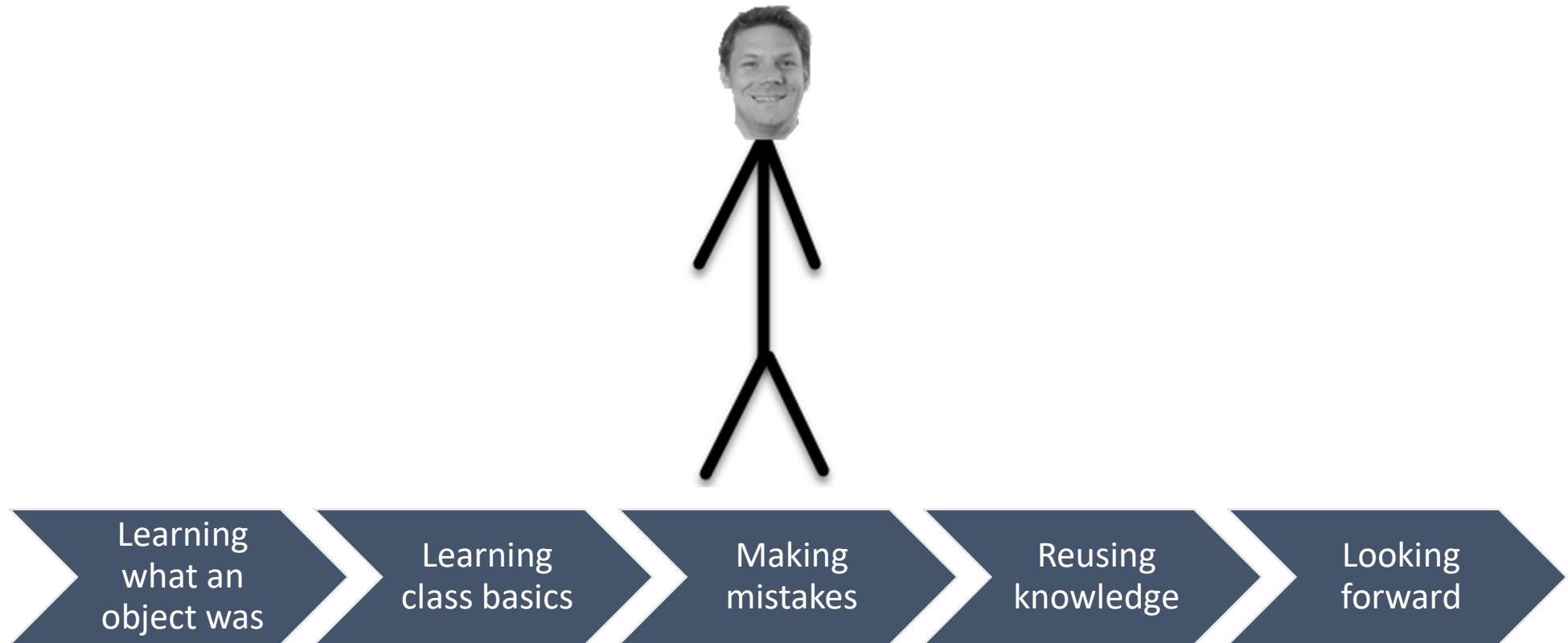
Open Source projects using Powershell Classes

My own 'creations' and some really cool ones

15 Septembre 2018 - #FrPwshSat2018



Plan:



15 Septembre 2018 - #FrPwshSat2018

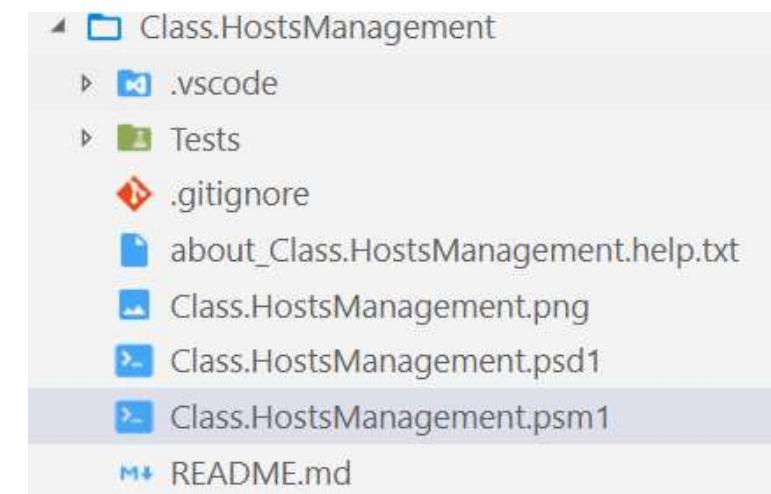
Class.HostsManagement: Me (Stéphane van Gulick)



Simple classes + Enum -> <https://github.com/Stephanevg/Class.HostsManagement>

```
Class.HostsManagement.psm1 x

Stephanie, a year ago | 2 authors (Stéphanie van Gulick and others)
1 Enum HostsEntryType{
2     Entry
3     Comment
4     BlankLine
5 }
6
7 Class HostsEntry{
8     [Ipaddress]$IpAddress
9     [string]$Hostname
10    [String]$FullyQualifiedName
11    [String]$Description
12    [HostsEntryType]$EntryType
13 }
14 HostsEntry(([IPAddress]$IpAddress,[String]$HostName,[String]$FQDN,[String]$Description,[HostsEntryType]$Type){...}
15
16 HostsEntry(([IpAddress]$IpAddress,[String]$HostName,[String]$FQDN,[String]$Description){...
17
18 HostsEntry([String]$Comment,[HostsEntryType]$Type){...
19
20 hidden HostsEntry([string]$Line){...
21
22 HostsEntry(){...
23
24 [HostsEntryType] static hidden GetLineType([string]$Line){...
25
26
27 Class HostsFile{
28     hidden [HostsEntry[]]$Entries
29     [string]$Path
30     hidden [int]$LogRotation = 5
31
32 HostsFile(){...
33
34 HostsFile([String]$ComputerName){...
35
36 HostsFile([System.IO.FileSystemInfo]$Path){...
37
38 [void]ReadHostsFileContent(){...
39
40 [array]GetEntries(){...
41
42 [void]AddHostsEntry([HostsEntry[]]$entries){...
43
44 [void]RemoveHostsEntry([HostsEntry[]]$entries){...
45
46 [void]Backup([System.IO.DirectoryInfo]$BackupFolder){...
47
48 [void]Backup(){...
49 <#...
50
51 [void]Set(){...
52
53
54
55
56
57 }
```



15 Septembre 2018 - #FrPwshSat2018



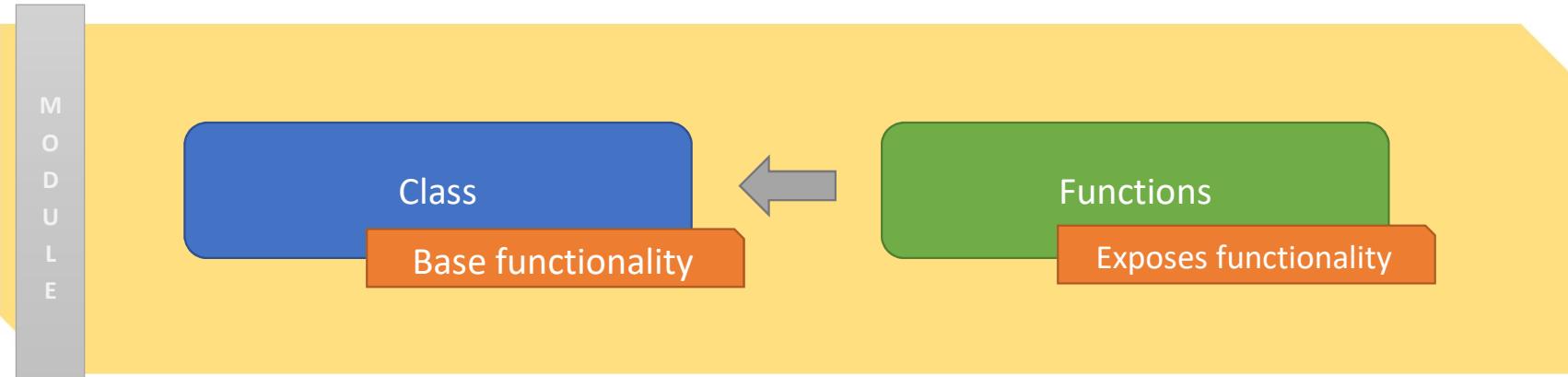
Feedback

Feedback Received:



- How does it work?
- How do I load the module?
- How can I do **\$Something**
- Where is the help? (get-Help?)

My recommendation: Module Structure using classes



```

1  Class PackageDocument {
2      [system.io.Fileinfo]$File
3      [DocumentType]$Type
4      [CMPackage]$Document
5      [String]$SiteCode = (Get-CMSiteCode)
6
7      PackageDocument() {...}
8
9      PackageDocument([String]$File) {...}
10
11     [PackageDocument]AddPackage([CMPackage]$CMPPackage) {...}
12
13     [PackageDocument] ExportDocument() {...}
14
15     [string] static GetDocumentTemplate([DocumentType]$Type) {...}
16
17     [string] SetSiteCode([String]$Sitecode) {...}
18
19     SetPsDrive() {...}
20
21     [Bool] validateDocument() {...}
22
23 }
```

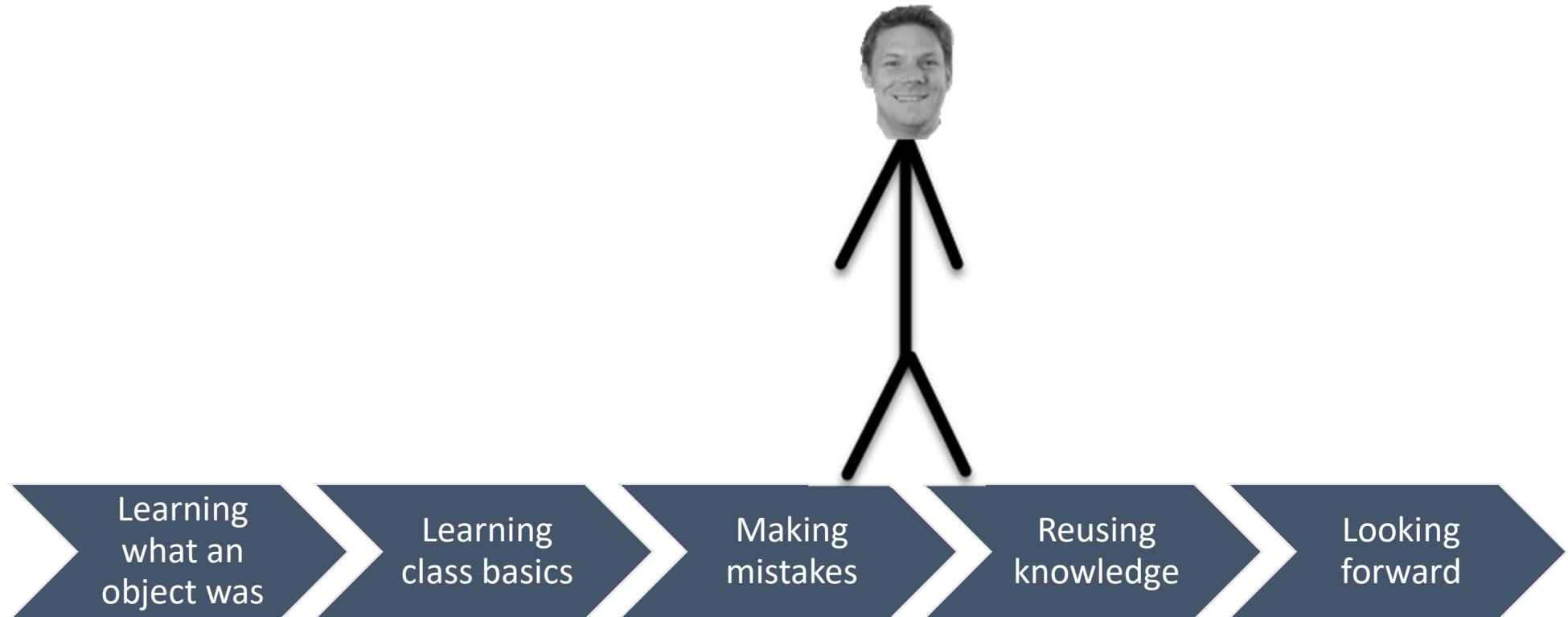
```

Function Get-PackageDocument {
    [CmdletBinding()]
    Param(
        [String]$Path
    )
    if ($Path) {
        $Document = Get-PackageDocument -Path $Path
        return $Document
    }
}

Function Test-PackageDocument {
    <#...
    #>
    [CmdletBinding()]
    Param(
        [Parameter(Mandatory=$true)]
        [PackageDocument]$Document
    )
    return $Document.ValidateDocument()
}
```

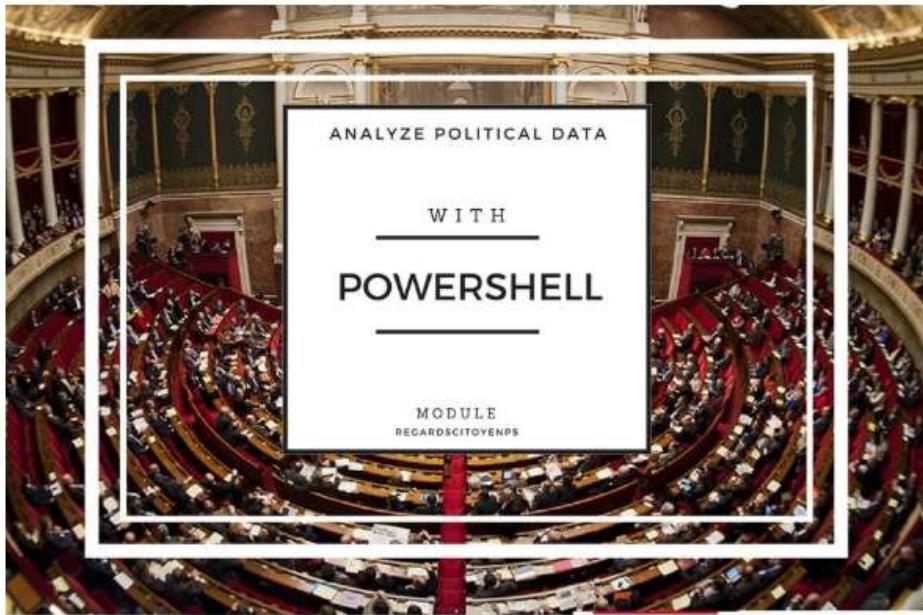


Plan:



15 Septembre 2018 - #FrPwshSat2018

Learning by doing (the right way)



Updated

- Encapsulated functionality in class methods. Exposed via classic powershell functions.
- Added functions to module (New-blabla, Get-Blabla etc..)
- Added comment based help in functions
- Added About_help file
- Used ScriptsToProcess



Simplify import of modules using classes

A screenshot of a PowerShell IDE interface. At the top, there are four tabs: 'RegardsCitoyenPS.psd1', 'RegardsCitoyenPS.psd1', 'LoadClasses.ps1', and 'RegardsCitoyenHelpers.psm1'. The 'RegardsCitoyenPS.psd1' tab is active and shows a large amount of PowerShell script code. The 'LoadClasses.ps1' tab shows a snippet of code starting with '1 using', and the 'RegardsCitoyenHelpers.psm1' tab shows a snippet of code starting with '2 |'. The code in the active tab includes definitions for classes like 'GroupePolitique', 'Mandat', 'Depute', 'Synthese', 'Intervention', and 'Dossier', along with various properties and methods. The code uses a mix of English and French terms, such as 'RequiredModule', 'RequiredAssembly', 'Script files', and 'ScriptsToProcess'. The overall theme is the simplification of module imports through the use of classes.

15 Septembre 2018 - #FrPwshSat2018

Example from the field



```
198 Class Dossier {
199     [int]$id
200     [string]$titre
201     [Datetime]$minDate
202     [DateTime]$maxDate
203     [int]$nbInterventions
204     [Object[]]$intervenants
205     [Intervention[]]$seances
206     [String[]]$documents
207     [String[]]$sousSection
208     hidden [string[]]$id_intervenants
209     hidden [string[]]$id_seances
210     hidden [string[]]$id_documents
211     hidden [string[]]$id_soussections
212
213
214     Dossier([int]$id,[String]$Titre,[int]$NbInterventions,[DateTime]$minDate
215
216     hidden [Void] _loadIntervenants(){...}
217
218     hidden [void] _loadSeances(){...}
219
220     hidden [Void] _loadDocuments(){
221
222         write-verbose "Chargement des seances.."
223         if ($this.id_documents){
224             $docs += @()
225             foreach ($id in $this.id_documents){
226
227                 $docs += Get-RCDocument -id $id
228             }
229
230             $this.Documents = $docs
231         }
232
233     }
234
235
236     hidden [Void] _loadSousSections(){...}
237
238
239     [Dossier] Full(){
240         $this._loadSeances()
241         $this._loadDocuments()
242         $this._loadIntervenants()
243
244
245         return $this
246     }
247
248
249
250
251 }
```



PsClassUtils – Stéphane van Gulick (me)

Simple classes

```
1 Class ClassProperty {
2     [String]$Name
3     [String]$Type
4
5     ClassProperty([String]$Name,[String]$Type){
6
7         $this.Name = $Name
8         $This.Type = $Type
9     }
10 }
```

```
1 Class ClassParameter {
2     [String]$Name
3     [String]$Type
4
5     ClassParameter([String]$Name,[String]$Type){
6
7         $this.Name = $Name
8         $This.Type = $Type
9     }
10 }
```

```
1 Class ClassMethod {
2     [String]$Name
3     [String]$ReturnType
4     [ClassParameter[]]$Parameter
5
6     ClassMethod([String]$Name,[String]$ReturnType,[ClassParameter[]]$Parameters){
7         $this.Name = $Name
8         $This.ReturnType = $ReturnType
9         $This.Parameter = $Parameters
10    }
11 }
12 }
```

Unsaved changes (cannot determine recent change or authors)

```
1 Class ClassConstructor {
2     [String]$Name
3     [ClassParameter[]]$Parameter
4
5     ClassConstructor([String]$Name,[ClassParameter[]]$Parameter){
6         $this.Name = $Name
7         $This.Parameter = $Parameter
8     }
9
10 }
```

The screenshot shows a file explorer window with the following structure:

- Classes (parent folder)
 - 00_ClassProperty.ps1
 - 01_ClassParameter.ps1
 - 02_ClassMethod.ps1 (selected)
 - 03_ClassConstructor.ps1
- Private



Polaris – Microsoft

<https://github.com/PowerShell/Polaris> - (5 'big' classes)

```
You, a few seconds ago | 2 authors (Micah Raidon and others)
class Polaris {
    [int]$Port
    [System.Collections.Generic.List[PolarisMiddleware]]$RouteMiddleWare = [System.Collections.Generic.List[PolarisMiddleware]]::new()
    [System.Collections.Generic.Dictionary[string, [System.Collections.Generic.Dictionary[string, scriptblock]]]]$ScriptblockRoutes = [System.Collections.Generic.Dictionary[string, [System.Collections.Generic.Dictionary[string, [System.Collections.Generic.List[scriptblock]]]]]]::new()
    hidden [Action[string]]$Logger
    hidden [System.Net.HttpListener]$Listener
    hidden [bool]$StopServer = $False
    [string]$GetLogsString = "PolarisLogs"
    [string]$ClassDefinitions = $Script:ClassDefinitions
    $ContextHandler = (New-ScriptblockCallback -Callback { ...
        hidden [object] InvokeRoute ([Scriptblock]$Route,[PSCustomObject]$Parameters,[PolarisRequest]$Request,[PolarisResponse]$Response) { ...
            [void] AddRoute ([string]$Path,[string]$Method,[scriptblock]$Scriptblock) { ...
                RemoveRoute ([string]$Path,[string]$Method) { ...
                    static [string] SanitizePath([string]$Path){...
                        static [RegEx] ConvertPathToRegex([string]$Path){...
                            static [RegEx] ConvertPathToRegex([RegEx]$Path){...
                                AddMiddleware ([string]$Name,[scriptblock]$Scriptblock) {...
                                    Send ([string]$Name,[scriptblock]$Scriptblock) {...
                                        RemoveMiddleware ([string]$Name) {...
                                            static [void] Start ([int]$Port = 3000) {...
                                                [void] Stop () {...
                                                    [void] InitListener ([int]$Port) {...
                                                        static [void] Send ([System.Net.HttpListenerResponse]$RawResponse, [PolarisResponse]$Response) {...
                                                            static [void] Send ([System.Net.HttpListenerResponse]$RawResponse, [byte[]]$ByteResponse, [int]$StatusCode, [string]$ContentType, [System.Net.WebHeader]$Header) {...
                                                                static [void] Send ([System.Net.HttpListenerResponse]$RawResponse, [System.IO.Stream]$StreamResponse, [int]$StatusCode,[string]$ContentType,[System.Net.WebHeader]$Header) {...
                                                                    static [void] Send ([System.Net.HttpListenerResponse]$RawResponse,[byte[]]$ByteResponse,[int]$StatusCode,[string]$ContentType) {...
                                                                        static [void] Log ([string]$LogString) {...
                                                                            static [Action[string]]$Logger {...

```

PoshBot – Brandon Olin



<https://github.com/poshbotio/PoshBot>

The screenshot shows two code files side-by-side in a code editor.

AccessFilter.ps1 (Left):

```
# An access filter controls under what conditions
class AccessFilter {
    class BaseLogger {
        [Logger]$Logger
        class Bot : BaseLogger {
            # In charge of executing and tracking commands
            class CommandExecutor : BaseLogger {
                [RoleManager]$RoleManager
                hidden [Bot]$_bot
                [int]$HistoryToKeep = 100
                [int]$ExecutedCount = 0
                # Recent history of commands executed
                [System.Collections.ArrayList]$RecentHistory
                # Plugin commands get executed
                # This is to keep track of those commands
                hidden [hashtable]$_jobTracker
                CommandExecutor([RoleManager]$RoleManager)
                $this.RoleManager = $RoleManager
                $this.Logger = $Logger
                $_bot = $Bot
            }
            # Execute a command
            [void]ExecuteCommand([Command]$cmd)
            # Verify command is not disabled
            if (-not $cmdExecutionContext.CommandDisabled) {
                $err = [CommandDisabled]$cmdExecutionContext
                $cmdExecutionContext.Completed = $true
                $cmdExecutionContext.Ended = $true
                $cmdExecutionContext.Result = $cmdExecutionContext.Result
                $cmdExecutionContext.Result = $this.LogInformation([LogSeverity]$Info)
                $this.TrackJob($cmdExecutionContext)
                return
            }
            # Verify that all mandatory parameters are present
            # This doesn't apply to cmdlets
            if ($cmdExecutionContext.CommandParameters) {
                if (-not $this.ValidateParameters($cmdExecutionContext)) {
                    $msg = "Mandatory parameters are missing: "
                    foreach ($usage in $cmdExecutionContext.Usage) {
                        $msg += "$usage"
                    }
                    $err = [CommandParameter]$cmdExecutionContext
                    $cmdExecutionContext.Complete = $true
                }
            }
        }
    }
}
```

Command.ps1 (Right):

```
Brandon Olin, a year ago | 1 author (Brandon Olin)

1 # Some custom exceptions dealing with executing commands
2 class CommandException : Exception {
3     CommandException() {}
4     CommandException([string]$Message) : base($Message) {}
5 }
6 class CommandNotFoundException : CommandException {
7     CommandNotFoundException() {}
8     CommandNotFoundException([string]$Message) : base($Message) {}
9 }
10 class CommandFailed : CommandException {
11     CommandFailed() {}
12     CommandFailed([string]$Message) : base($Message) {}
13 }
14 class CommandDisabled : CommandException {
15     CommandDisabled() {}
16     CommandDisabled([string]$Message) : base($Message) {}
17 }
18 class CommandNotAuthorized : CommandException {
19     CommandNotAuthorized() {}
20     CommandNotAuthorized([string]$Message) : base($Message) {}
21 }
22 class CommandRequirementsNotMet : CommandException {
23     CommandRequirementsNotMet() {}
24     CommandRequirementsNotMet([string]$Message) : base($Message) {}
25 }

# Represent a command that can be executed
[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute('PSAvoidGlobalVars', '', Scope='Function', Target='*')]
[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute('PSUseDeclaredVarsMoreThanAssignments', '', Scope='Function', Target='*')]
class Command : BaseLogger {
    # Unique (to the plugin) name of the command
    [string]$Name

    [string[]]$Aliases = @()

    [string]$Description

    [TriggerType]$TriggerType = [TriggerType]:Command

    [Trigger[]]$Triggers = @()

    [string[]]$Usage

    [bool]$KeepHistory = $true

    [bool]$HideFromHelp = $false

    [bool]$AsJob = $true

    # Fully qualified name of a cmdlet or function in a module to execute
    [string]$ModuleCommand

    [string]$ManifestPath

    [System.Management.Automation.FunctionInfo]$FunctionInfo
}
```

43 Classes!

- ▶ Classes
 - ▶ AccessFilter.ps1
 - ▶ ApprovalCommandConfiguration.ps1
 - ▶ ApprovalConfiguration.ps1
 - ▶ Approver.ps1
 - ▶ Backend.ps1
 - ▶ BaseLogger.ps1
 - ▶ Bot.ps1
 - ▶ BotConfiguration.ps1
 - ▶ ChannelRule.ps1
 - ▶ Command.ps1
 - ▶ CommandAuthorizationResult.ps1
 - ▶ CommandExecutionContext.ps1
 - ▶ CommandExecutor.ps1
 - ▶ CommandHistory.ps1
 - ▶ CommandParser.ps1
 - ▶ CommandResult.ps1
 - ▶ ConfigProvidedParameter.ps1
 - ▶ Connection.ps1
 - ▶ ConnectionConfig.ps1



My key take aways:

- 1) Write core logic in classes
- 2) abstract functionality in public methods.
- 3) Expose functionality to end users through functions (cmdlets)
- 4) Load functions through ScriptsToProcess





Taking advantage of existing knowledge

Don't re-invent the wheel

15 Septembre 2018 - #FrPwshSat2018



Plan:



15 Septembre 2018 - #FrPwshSat2018



Communicate via diagrams: UML

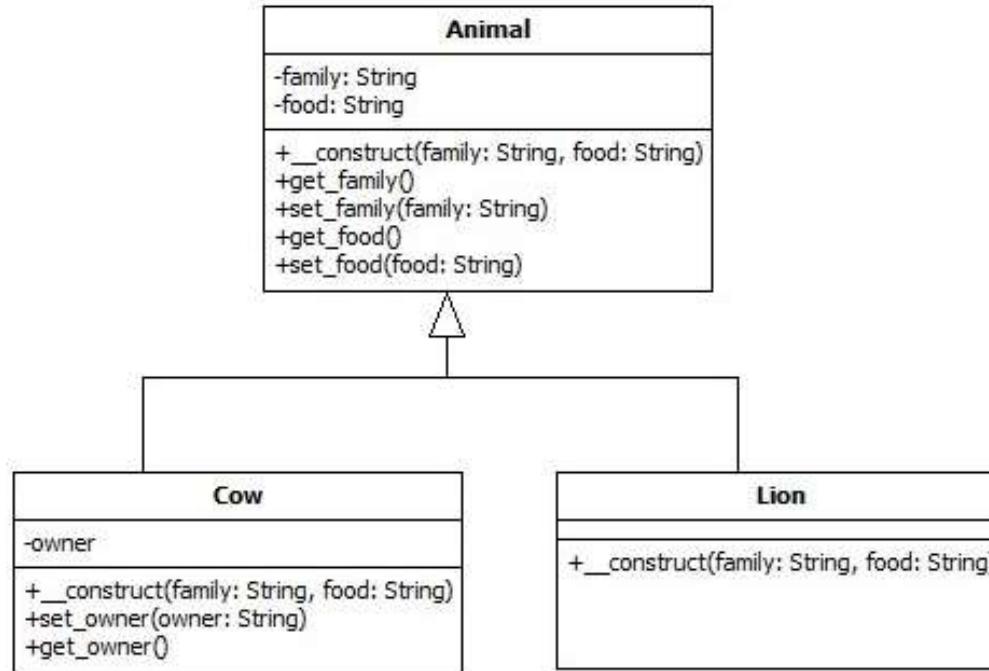
A picture is worth a
thousand words

Communicate via images using UML diagrams



15 Septembre 2018 - #FrPwshSat2018

Communicate via images using UML diagrams





Generate your own diagrams for your own scripts

PSClassUtils

15 Septembre 2018 - #FrPwshSat2018



Generate your own diagrams for your own scripts

```
> find-module psclassutils | install-module
```

CommandType	Name	Version	Source
Function	ConvertTo-titleCase	2.1.0	psclassutils
Function	Get-CUClass	2.1.0	psclassutils
Function	Get-CUClassConstructor	2.1.0	psclassutils
Function	Get-CUClassMethod	2.1.0	psclassutils
Function	Get-CUClassProperty	2.1.0	psclassutils
Function	Write-CUClassDiagram	2.1.0	psclassutils



UML Diagrams

Visual communication

```
Class User {
    #Properties
    [String]$FirstName
    [String]$LastName
    [String]$FullName

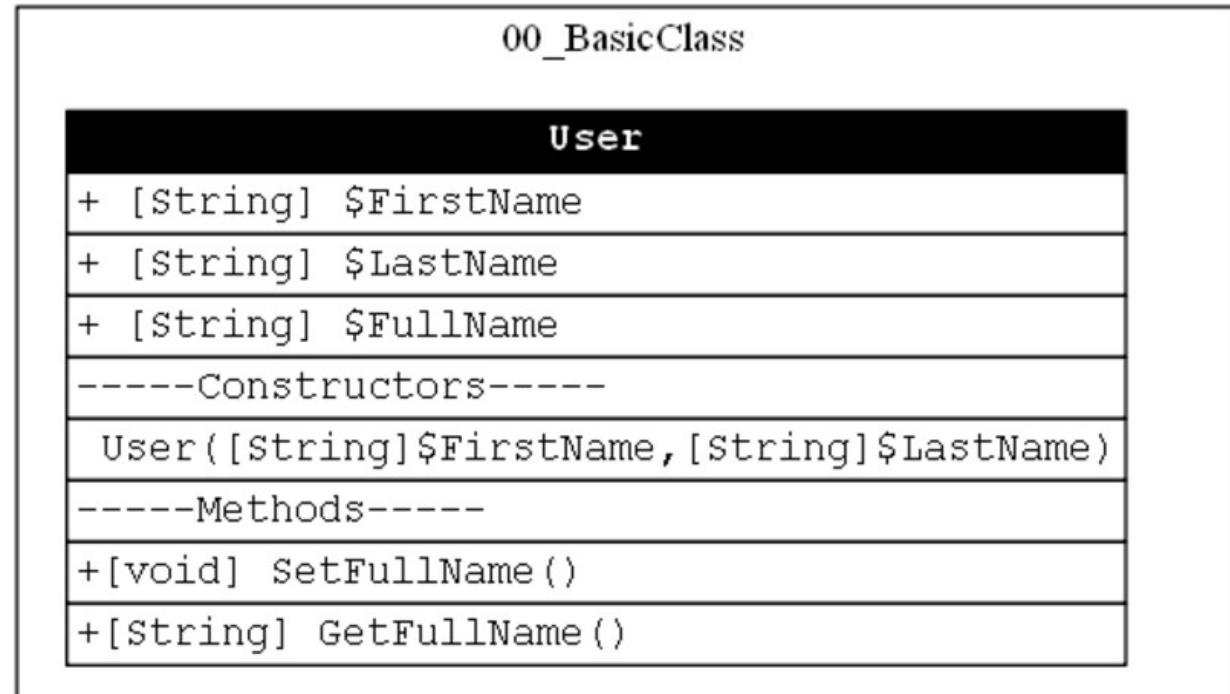
    #Constructor

    User ([String]$FirstName,[String]$LastName){
        $this.FirstName = $FirstName
        $this.LastName = $LastName
        $this.SetFullName()
    }

    #Methods

    [void]SetFullName(){
        $this.FullName = $this.Name + " " + $this.LastName
    }

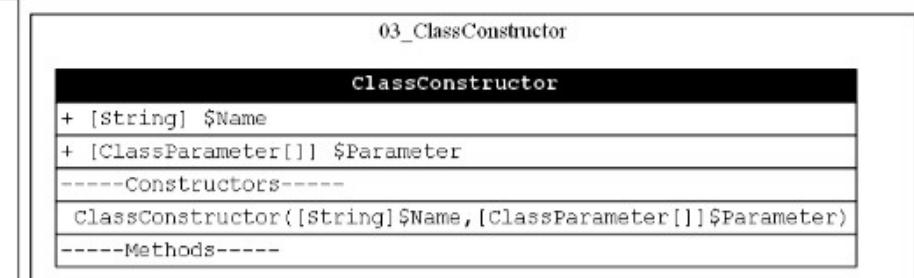
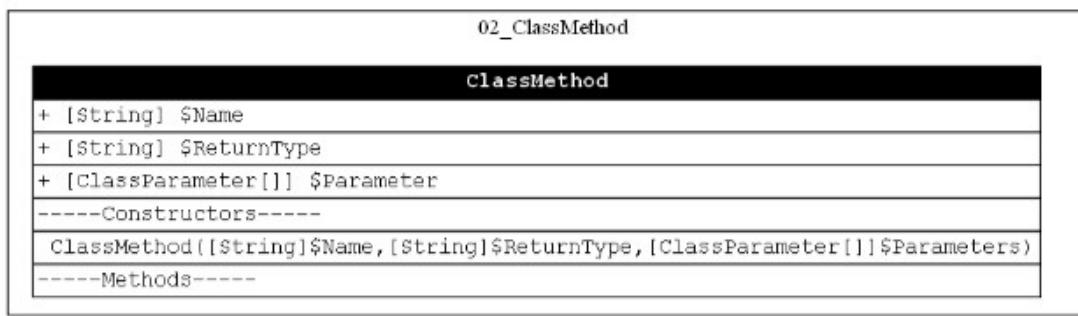
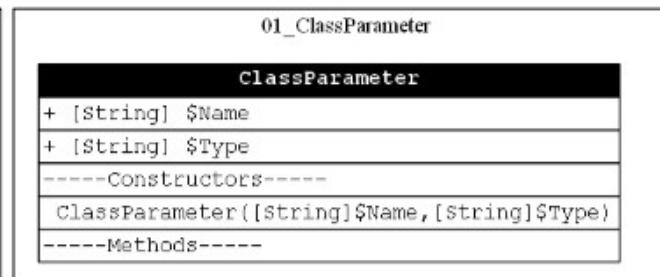
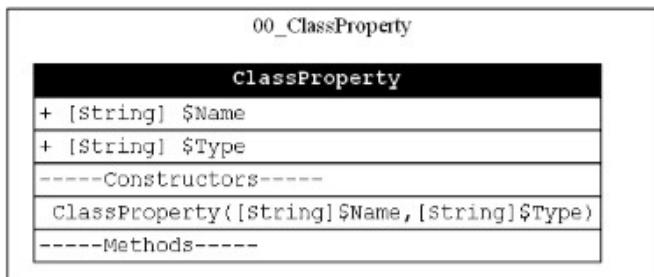
    [String]GetFullName(){
        return $this.FullName
    }
}
```





PsClassUtils – Stéphane van Gulick (me)

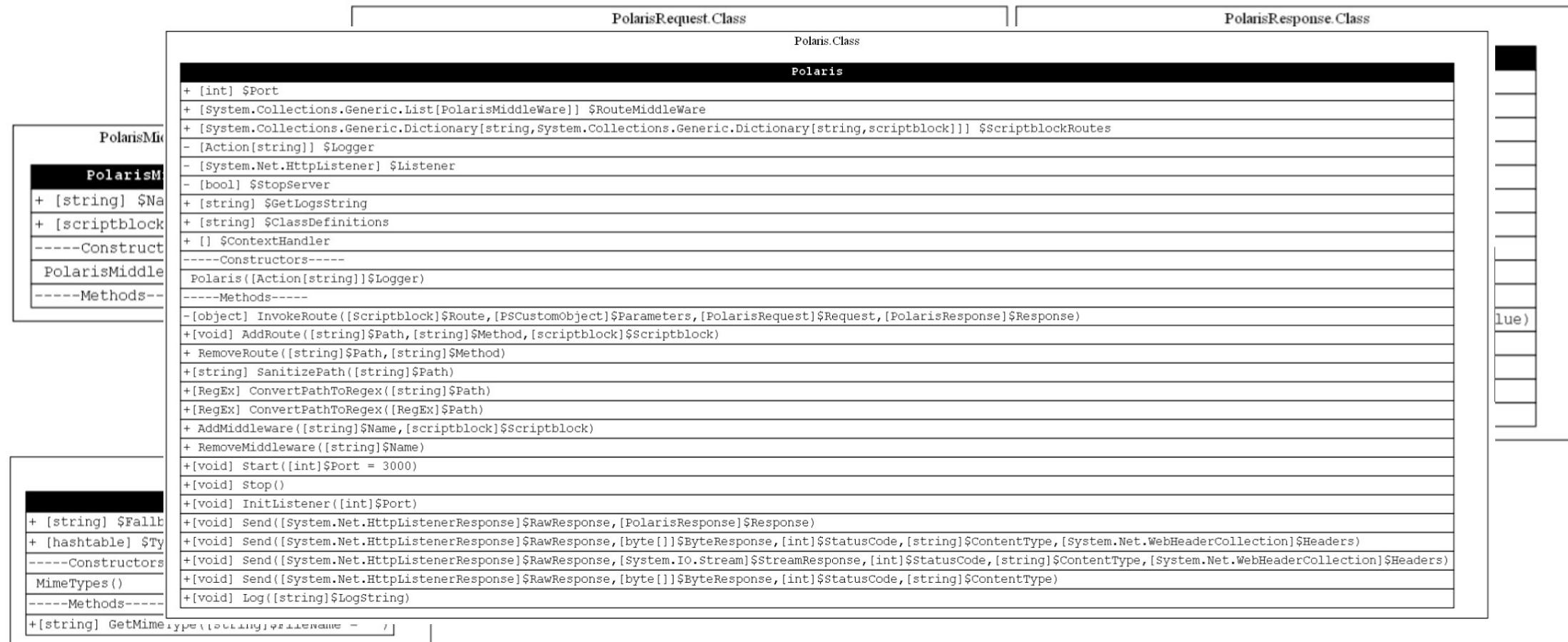
Simple class





Polaris - Microsoft

<https://github.com/PowerShell/Polaris>



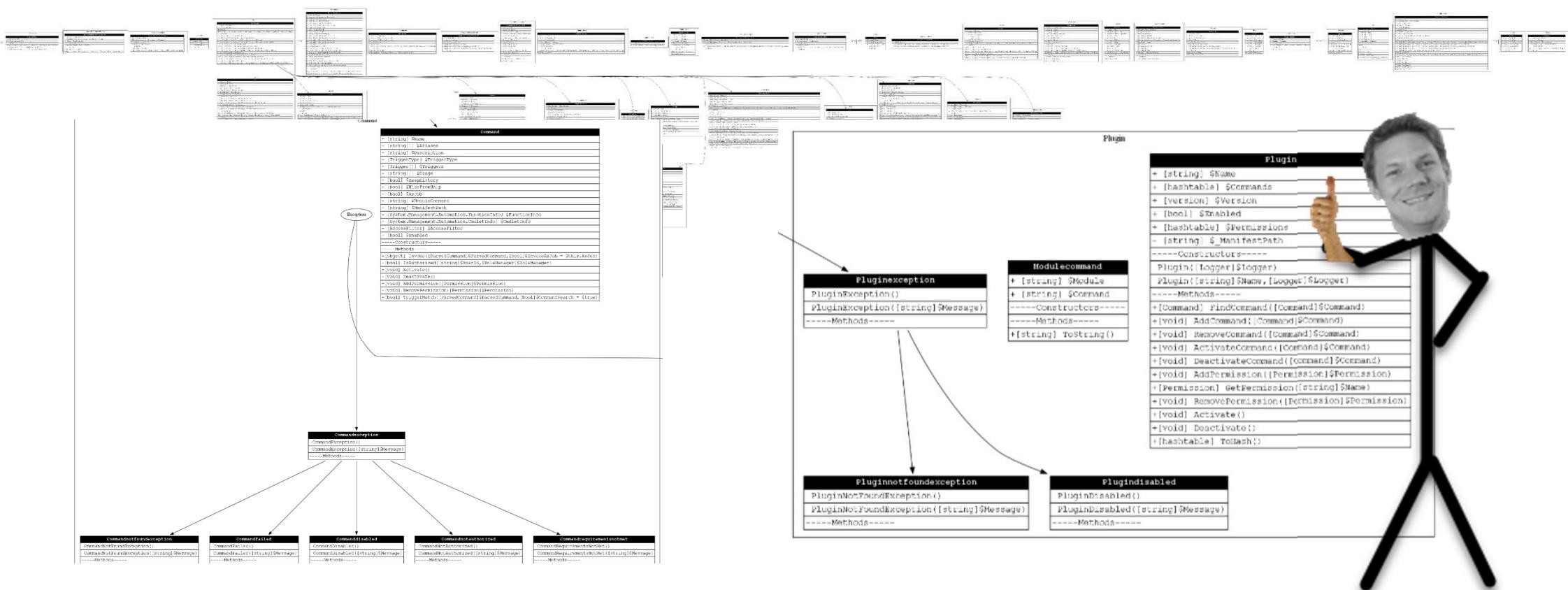
The diagram shows a UML class structure for the Polaris framework. At the top level, there is a **Polaris** class containing several methods. Below it, there are two main classes: **PolarisRequest** and **PolarisResponse**. The **PolarisRequest** class has a single attribute, **\$RouteMiddleWare**, which is a list of **PolarisMiddleWare** objects. The **PolarisResponse** class also has a single attribute, **\$Headers**, which is a collection of headers. On the left side of the diagram, there is a sidebar with a tree view of the code structure:

- PolarisModule**:
 - PolarisModule**:
 - + [string] \$Name
 - + [scriptblock]
 - Constructors----
 - PolarisModule()
 - Methods----
- MimeTypes**:
 - + [string] \$Fallback
 - + [hashtable] \$Types
 - Constructors----
 - MimeTypes()
 - Methods----
 - + [string] GetMimeType([string]\$filename = '')

15 Septembre 2018 - #FrPwshSat2018

PoshBot -

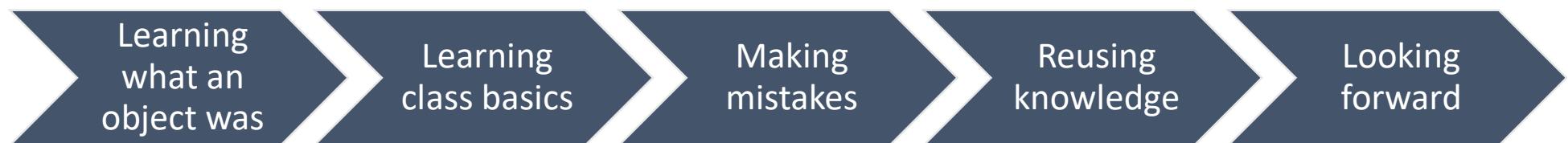
Yep...



15 Septembre 2018 - #FrPwshSat2018



Plan:



15 Septembre 2018 - #FrPwshSat2018



Chill:



15 Septembre 2018 - #FrPwshSat2018

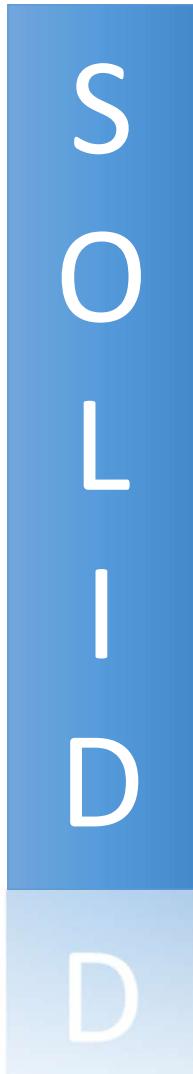


Use well known standards:

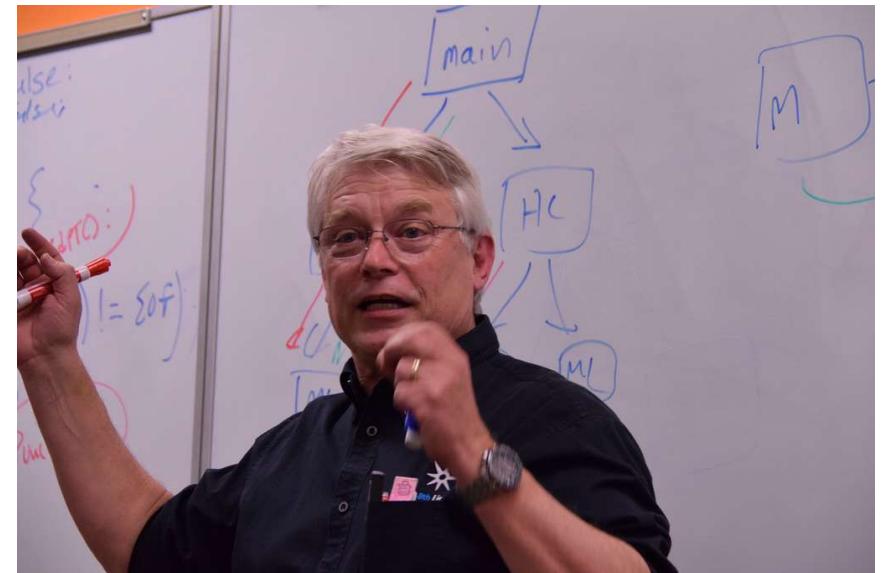
SOLID

15 Septembre 2018 - #FrPwshSat2018

Use well known standards:



Robert C. Martin (Aka – Uncle Bob)



15 Septembre 2018 - #FrPwshSat2018



Use well known standards:

S
O
L
I
D

Single responsibility principle

One class / method must have
only one responsibility



Use well known standards:

S
O
L
I
D

Open Closed Principle

Open for extension

But

Closed for modification



Use well known standards:

S
O
L
I
D
D

Liskov substitution principle

Ability to replace any instance of a parent class with an instance of one of its child classes without side effects



Use well known standards:

S
O
L
I
D
D

Interface segregation principle

Write a lot of small interfaces instead of one huge interface

A class should not be forced to depend on methods it doesn't use



Use well known standards:

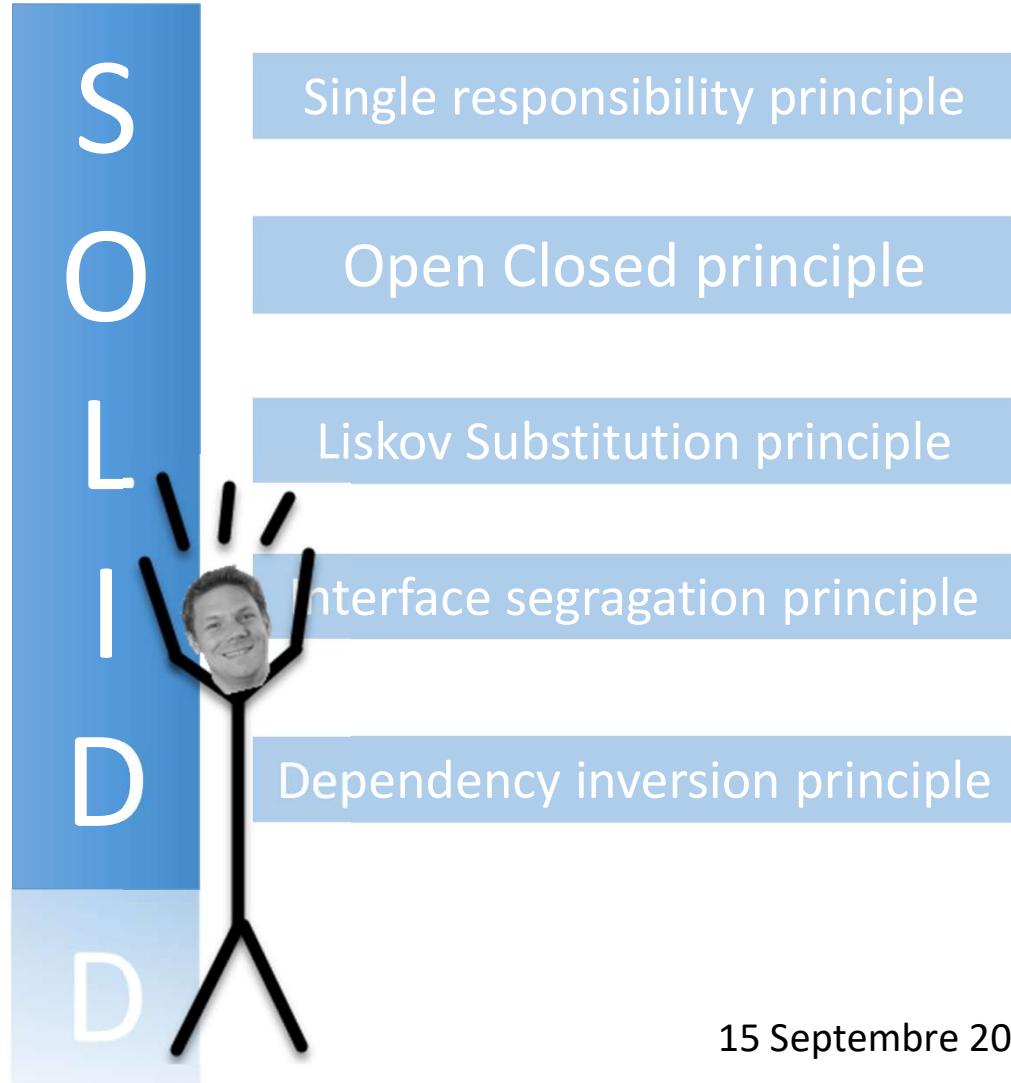
S
O
L
I
D

Dependency inversion principle

High level Modules should not depend on
low level implementations

The ActiveDirectory Module should depend
on your script to work.
The ActiveDirectory Module should not
depend on your script

Use well known standards:



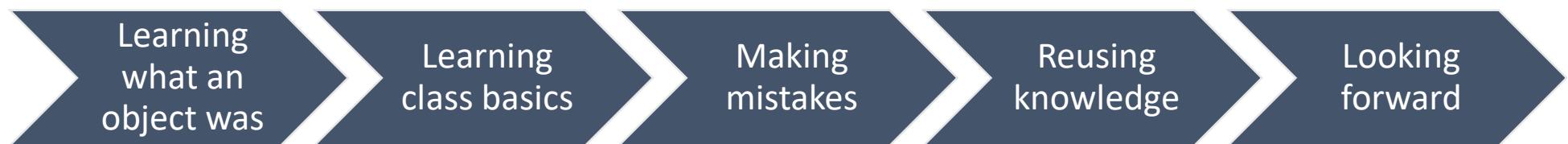


New Concepts

15 Septembre 2018 - #FrPwshSat2018



Plan:



15 Septembre 2018 - #FrPwshSat2018



Use well known standards:

TDD

Test Driven Developement

15 Septembre 2018 - #FrPwshSat2018



Test Driven Development:

Concepts

You are **NOT allowed** to write
production code that **DOESN'T**
has a test.

Write a test, and make it fail

Write **ONLY** enough *production code* to make the **test pass**.

Write a test, and make it fail

1

2

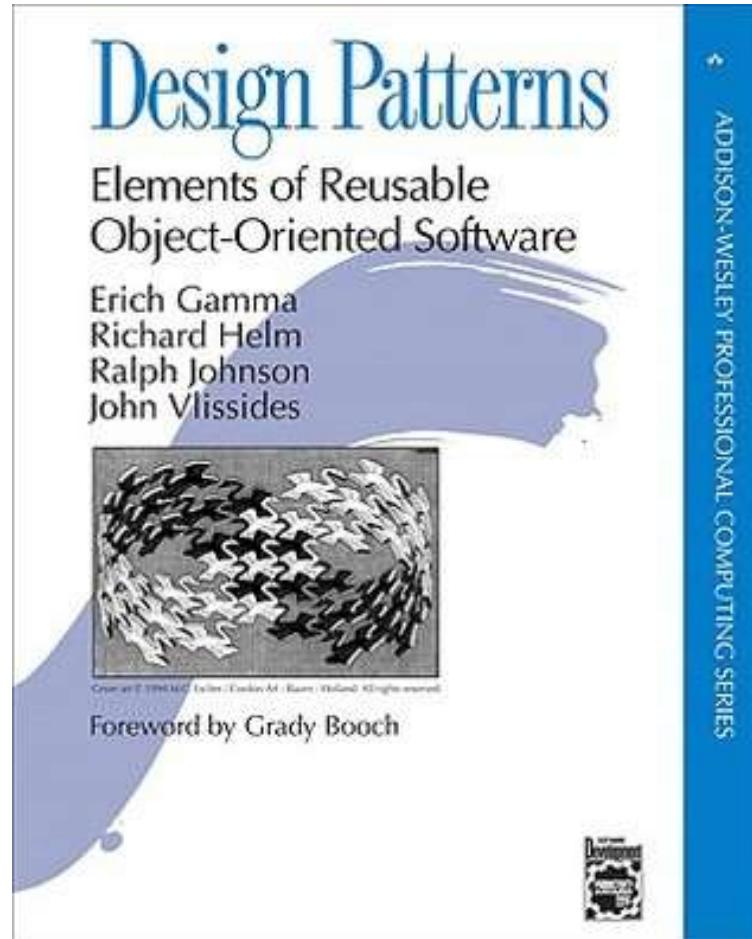
3

3



Design patterns

Gang of four



15 Septembre 2018 - #FrPwshSat2018

Design pattern: Example

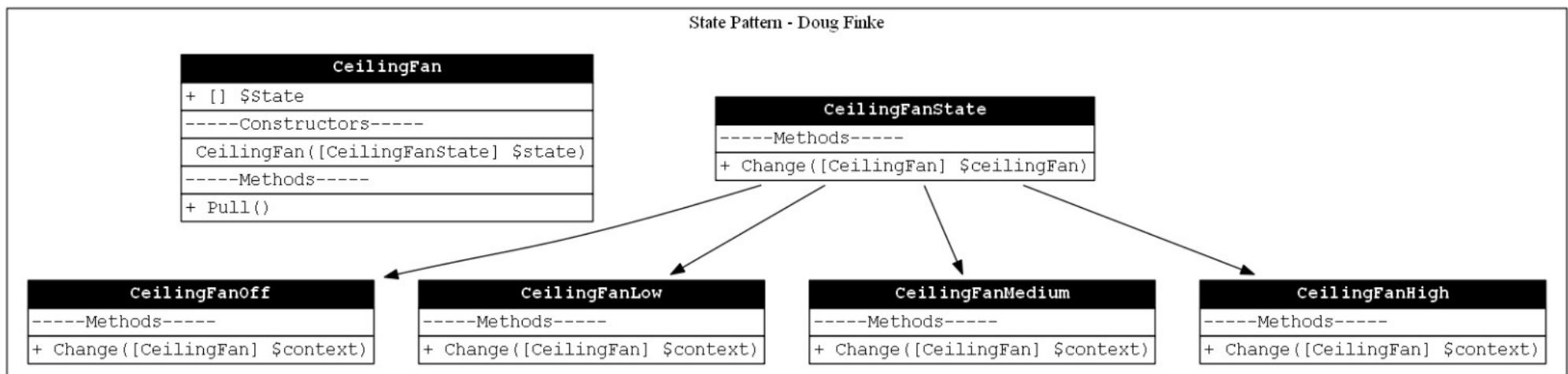
State Pattern



15 Septembre 2018 - #FrPwshSat2018

Design pattern: Example

State Pattern





Design pattern: Example

State Pattern

```
class CeilingFanState {
    Change([CeilingFan] $ceilingFan) {
        throw "not implemented"
    }
}

class CeilingFanLow: CeilingFanState {
    Change([CeilingFan] $context) {
        $context.State = [CeilingFanMedium]::new()
        "Change state from Low to Medium." | Out-Host
    }
}

class CeilingFanHigh: CeilingFanState {
    Change([CeilingFan] $context) {
        $context.State = [CeilingFanLow]::new()
        "Change state from High to Off." | Out-Host
    }
}

class CeilingFanMedium: CeilingFanState {
    Change([CeilingFan] $context) {
        $context.State = [CeilingFanHigh]::new()
        "Change state from Medium to High." | Out-Host
    }
}

class CeilingFanOff : CeilingFanState {
    Change([CeilingFan] $context) {
        $context.State = [CeilingFanLow]::new()
        "Change state from Off to Low." | Out-Host
    }
}
```



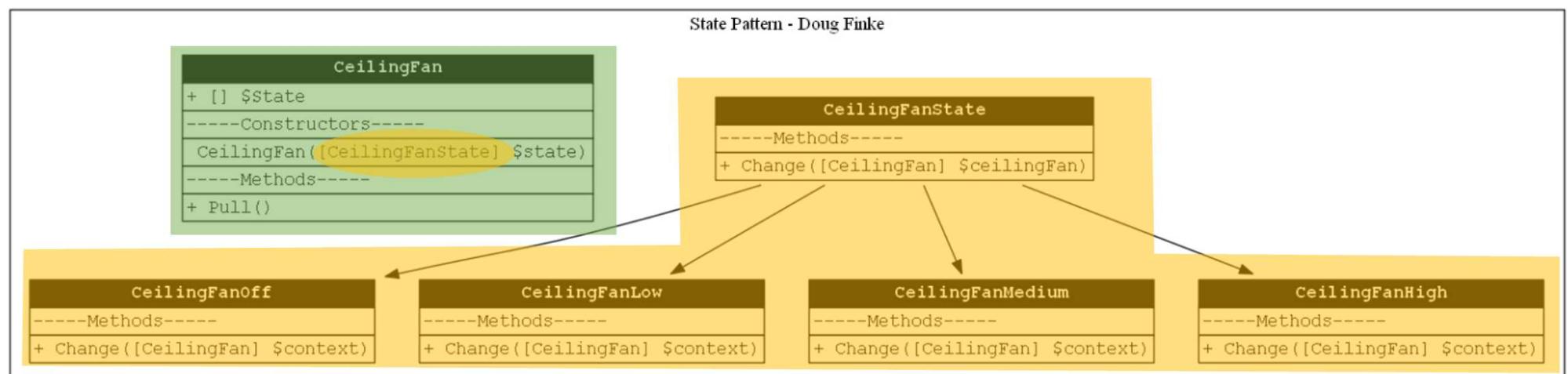
Design pattern: Example

State Pattern

```
59  $ceilingFan.Pull()  
class CeilingFa 60  $ceilingFan.Pull()  
[CeilingFan 61  $ceilingFan.Pull()  
62  $ceilingFan.Pull()  
  
CeilingFan PROBLEMS 3 OUTPUT DEBUG CONSOLE s.State = $state }  
Pull() { $t Change state from Off to Low.  
}  
Change state from Low to Medium.  
Change state from Medium to High.  
Change state from High to Off.  
  
$ceilingFan = [CeilingFan]::new([CeilingFanOff]::new())
```

Design pattern: Example

State Pattern



Design pattern: Key take aways

State Pattern

- Used different classes to express the notion of 'State'
- Each class is (represents) a specific state, with it's own behaviour
- Used inheritance
- Used Polymorphism

State Pattern

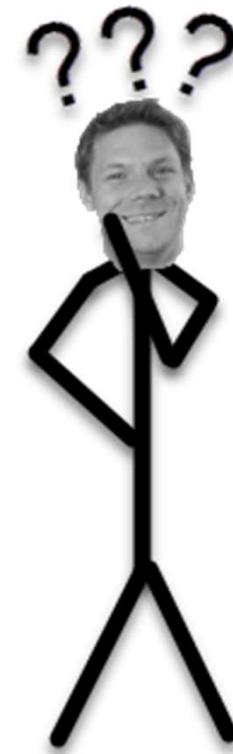
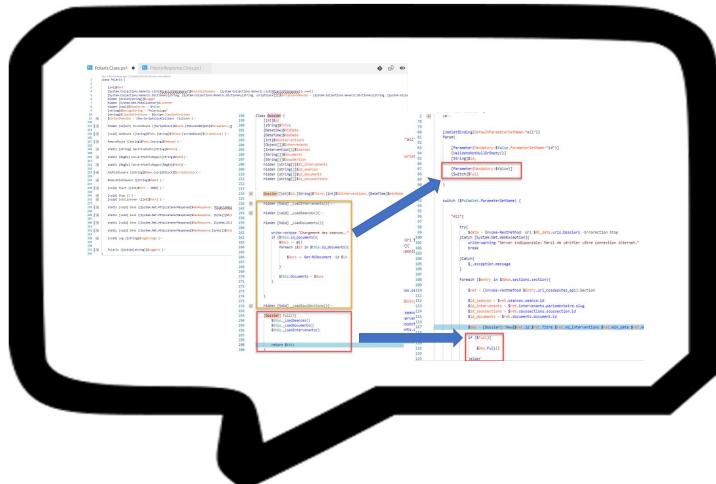
Standard Answer to a common / Well known problem

Reusable design

Reusable Code



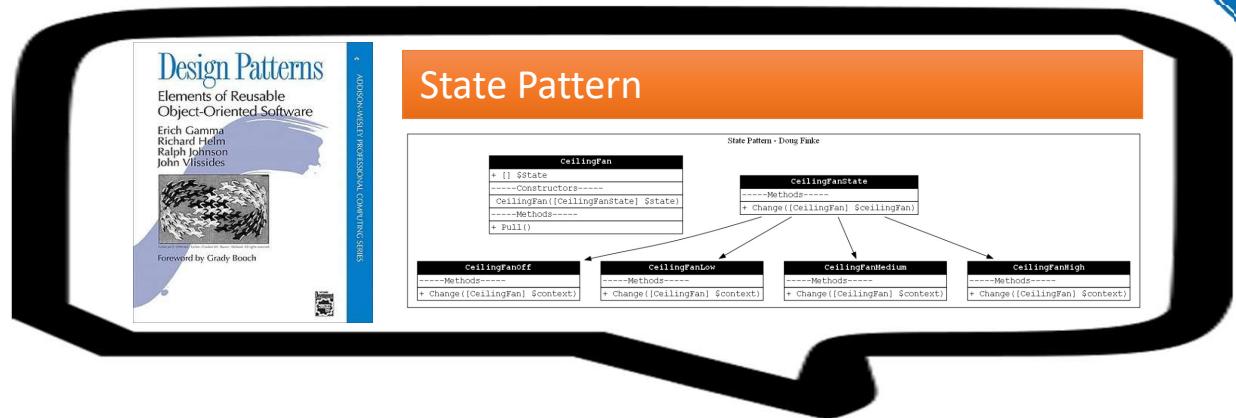
Design pattern: Key take aways



15 Septembre 2018 - #FrPwshSat2018



Design pattern: Key take aways



15 Septembre 2018 - #FrPwshSat2018

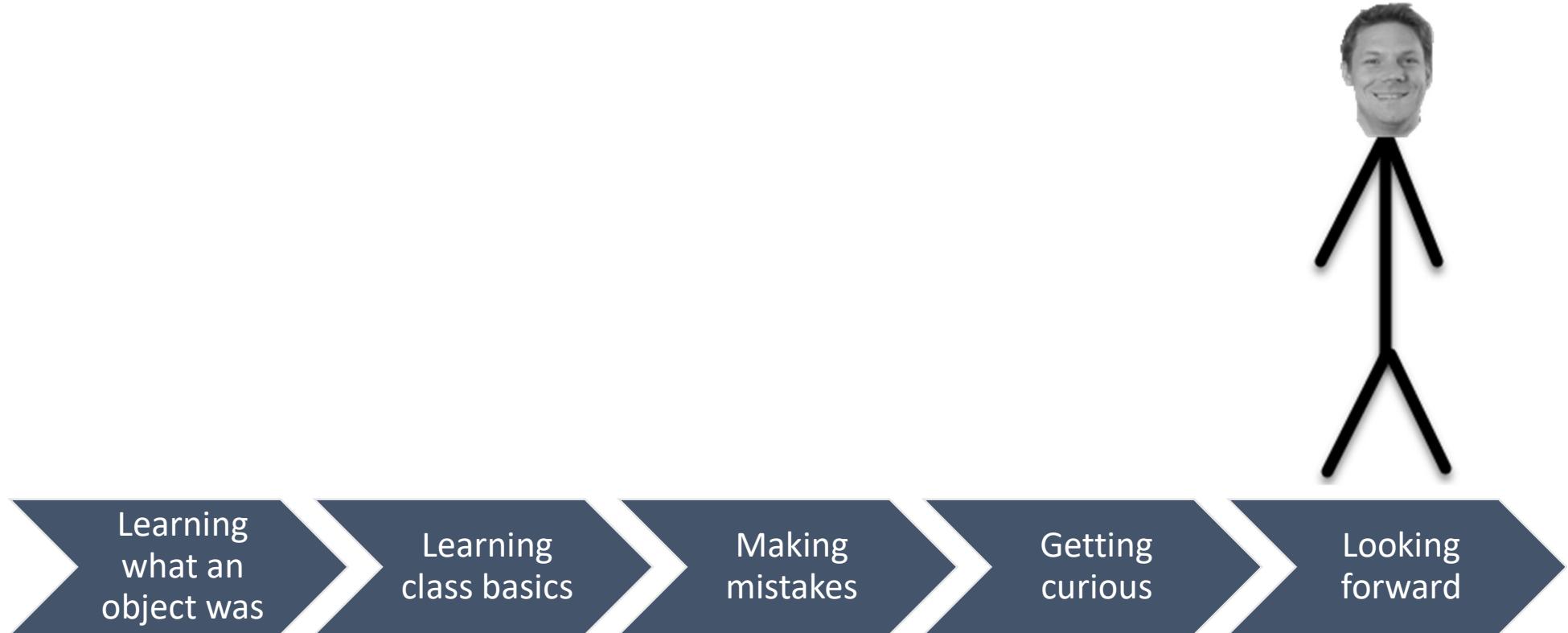


(My) Next step(s)

15 Septembre 2018 - #FrPwshSat2018



Plan:



15 Septembre 2018 - #FrPwshSat2018



Use well known standards:

TDD

Test Driven Developement

15 Septembre 2018 - #FrPwshSat2018

Design patterns:



15 Septembre 2018 - #FrPwshSat2018



Design patterns:

Creation Design Patterns

Singleton Pattern

Factory Pattern

Abstract factory

Builder Pattern

Prototype Pattern

Structural Design Patterns

Adapter Pattern

Compositie Pattern

Proxy Pattern

Flyweight Pattern

Facade Pattern

Bridge Pattern

Decorator Pattern

Behavioral Design Patterns

State Pattern

Observer Pattern

Strategy Pattern

Iterator Pattern

Visitor Pattern

Template Method

Mediator Pattern

Command Pattern

Interpreter Pattern

Memento Pattern

Chain of responsibility pattern

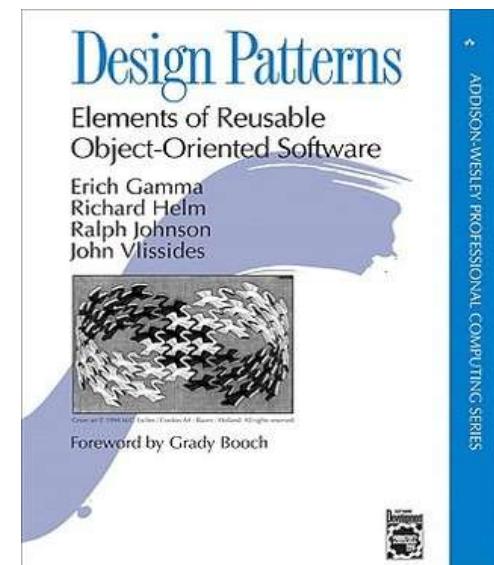


Books I (really) recommend



<https://www.amazon.fr/First-Design-Patterns-Elisabeth-Freeman/dp/0596007124>

<https://www.amazon.fr/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612>

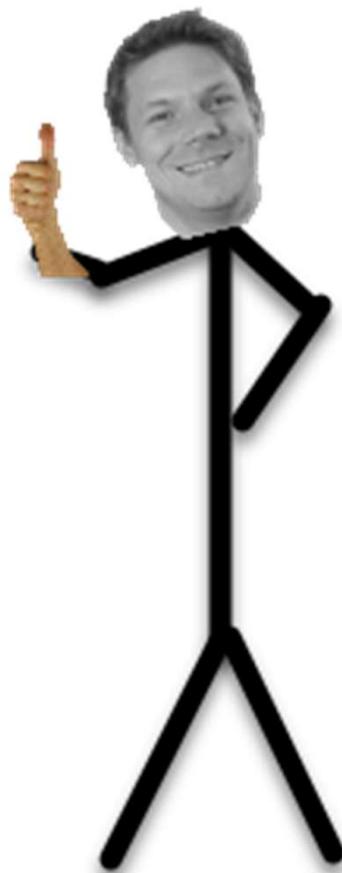




What do you see?

A bridge!

15 Septembre 2018 - #FrPwshSat2018



15 Septembre 2018 - #FrPwshSat2018



French PowerShell Saturday

Merci à nos sponsors !!



 METSYS


smartview
conseil et formation

15 Septembre 2018 - #FrPwshSat2018