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

```powershell
Function Get-Speaker {
    $speaker =[Ordered] @{
        "Name" = "van Gulick"
        "FirstName" = "Stéphane"
        "WorkPlace" = "Bern"
        "isMVP" = $true

    }
    return New-Object psobject -Property $speaker

}

Get-Speaker
```

```
PROBLEMS  TERMINAL  ...  1: PowerShell Integrate

Name       FirstName WorkPlace isMVP
----       --------- --------- -----
van Gulick Stéphane  Bern       True
```

Plan:

Learning what an object was

Learning class basics

Making mistakes

Reusing knowledge

Looking forward

Learning from the begining

The struggle

15 Septembre 2018 - #FrPwshSat2018

Introduction: assimilating class concepts

Function

```powershell
Function User {
    [CmdletBinding()]
    Param(

        [Parameter(Mandatory=$true)][String]$FirstName,
        [Parameter(Mandatory=$true)][String]$LastName

    )

    $Hash = @{
        FirstName = $FirstName
        LastName = $LastName
        FullName = $Name + " " + $LastName
    }

    return New-object psobject -Property $Hash

}

user -FirstName "Stéphane" -LastName "van Gulick"
```

Class

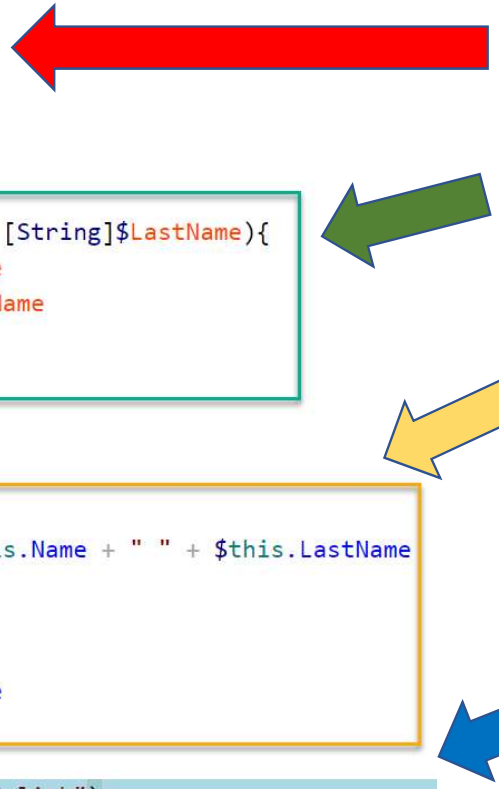
```powershell
Class User {
    #Properties
    [String]$FirstName
    [String]$LastName
    [String]$FullName

    #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!

Class Basics

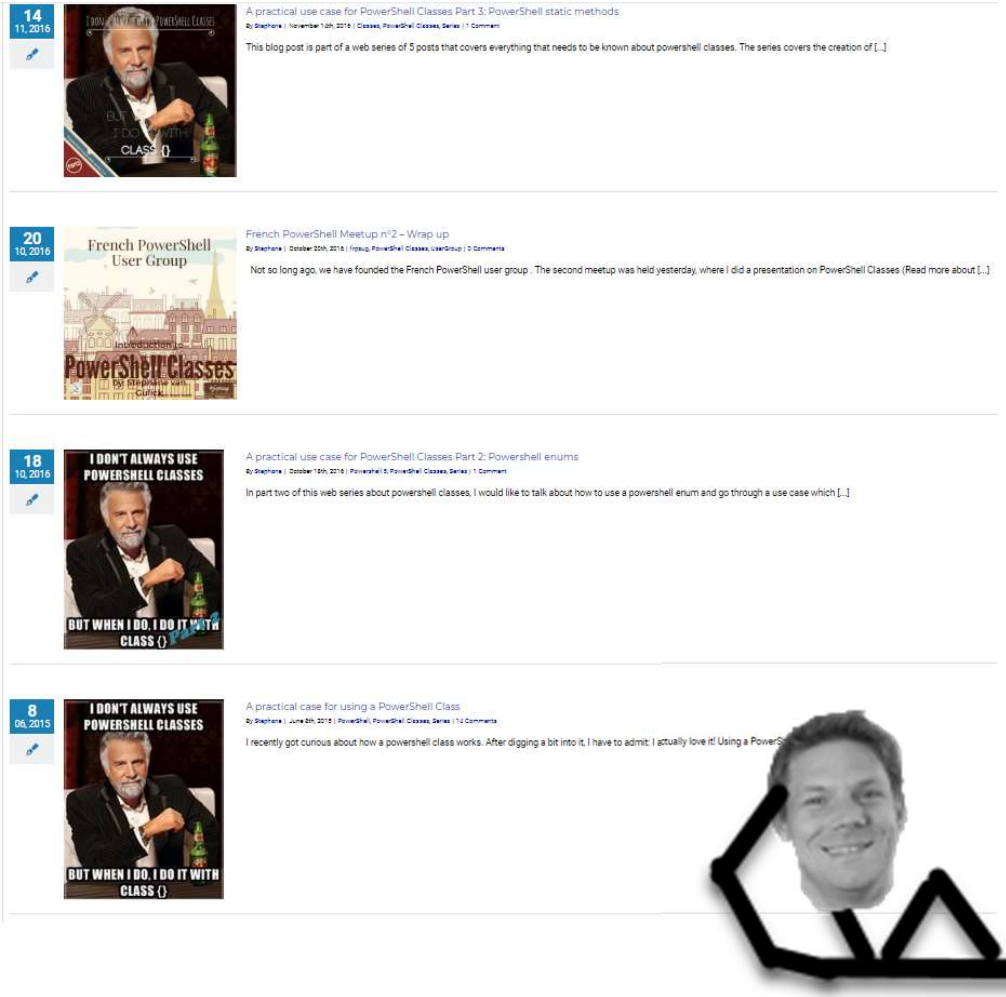
00_BasicClass.ps1

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

    #Constructor

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

    #Methods

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

    [String]GetFullName(){
        return $this.FullName
    }
}
[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 (Instanciate):
  - New-Object ClassName
  - [ClassName]::New()

Plan:



15 Septembre 2018 - #FrPwshSat2018

Plan:

$False

15 Septembre 2018 - #FrPwshSat2018

Plan:

Learning what an object was

Learning OOP basics

Making mistakes

Reusing knowledge

Looking forward

OOP Concepts

Programming guidlines

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

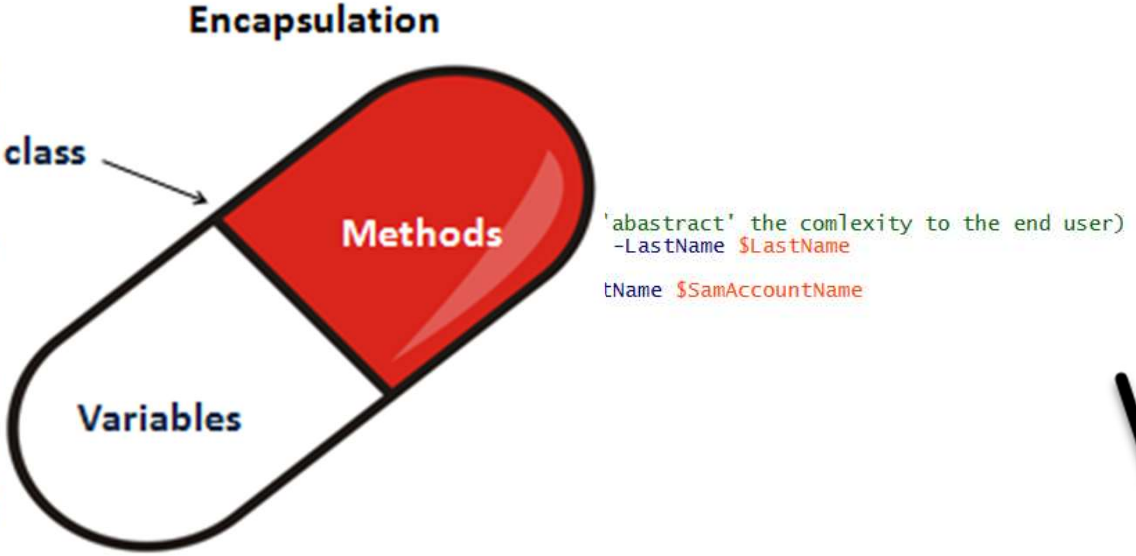
Abstraction

```
1735  Function Test-PackageDocument {
1736      <#…
1757      #>
1758      [CmdletBinding()]
1759      Param(
1760          [Parameter(Mandatory=$true)]
1761          [PackageDocument]$Document
1762      )
1763
1764      return $Document.ValidateDocument()
1765  }
```

4 Pillars of OOP: encapsulation

encapsulation

```
<#
    Function is pub
#>

Function New-Custom
    [Cmdletbinding(
    Param(
        [String]$Fi
        [String]$La
    )

    #Calling the pr
    $SamAccountName

    New-AdUser -Fir
}

<#
    Function is pri
#>

Function generateSa
    Param(
        [String]$Fi
        [String]$La
    )

    #Do stuff
}
```

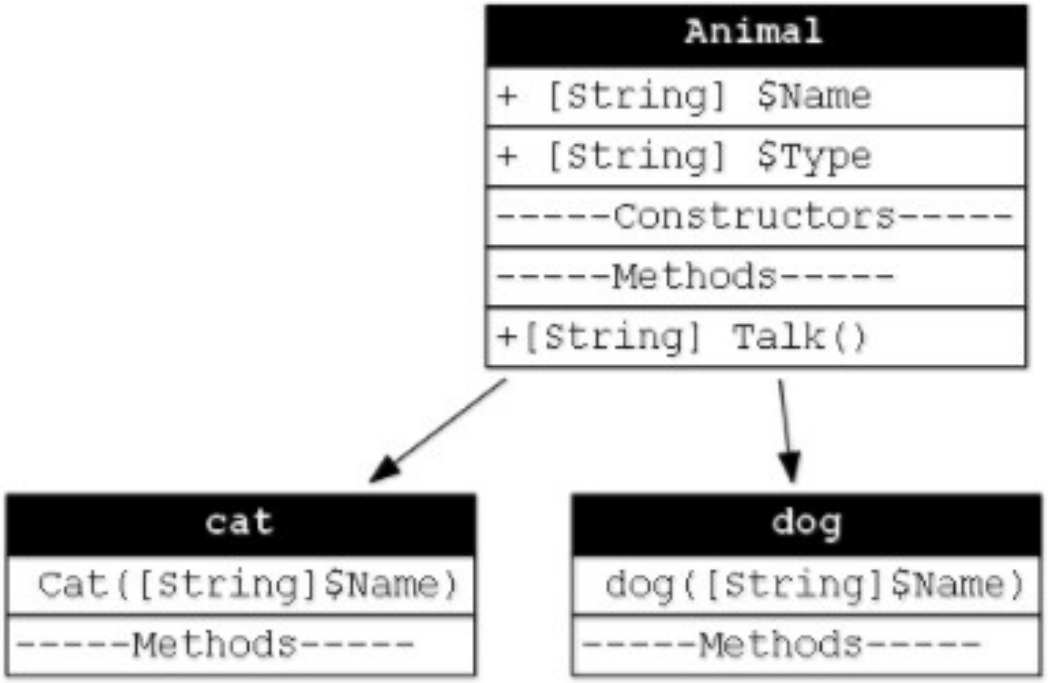


'abastract' the comlexity to the end user)
-LastName $LastName

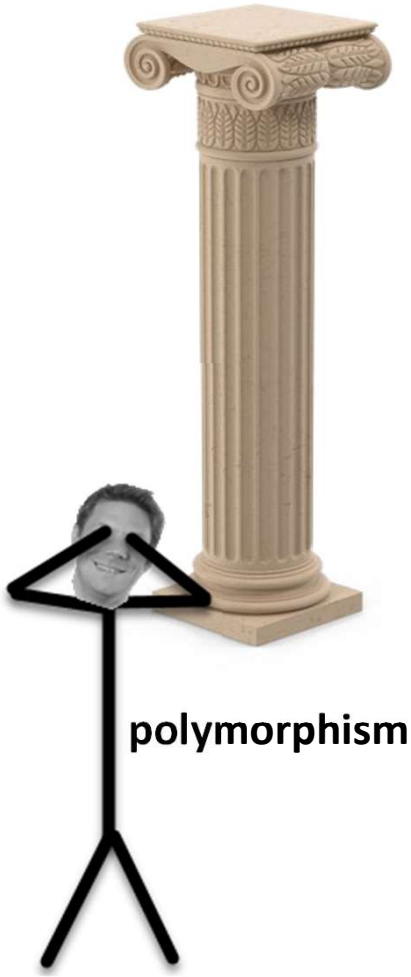
tName $SamAccountName

4 Pillars of OOP: inheritance

inheritance

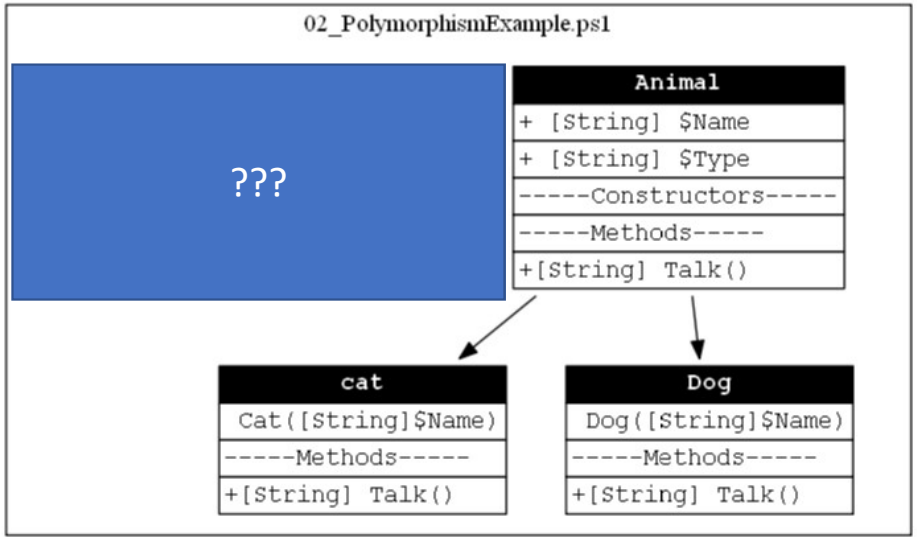


4 Pillars of OOP: polymorphism




```powershell
Class Animal {
    [String]$Name
    [String]$Type

    [String]Talk(){
        Return "Simple Animal"
    }
}

Class cat : Animal {
    Cat([String]$Name){
        $this.Name = $Name
        $this.Type = "Cat"
    }

    [String]Talk(){
        Return "Meow! I am a $($this.Type) and my Name is $($this.Name)"
    }
}


Class Dog : Animal {
    Dog([String]$Name){
        $this.Name = $Name
        $this.Type = "Dog"
    }

    [String]Talk(){
        REturn "Woof! I am a $($this.Type) and my Name is $($this.Name)."
    }
}
```

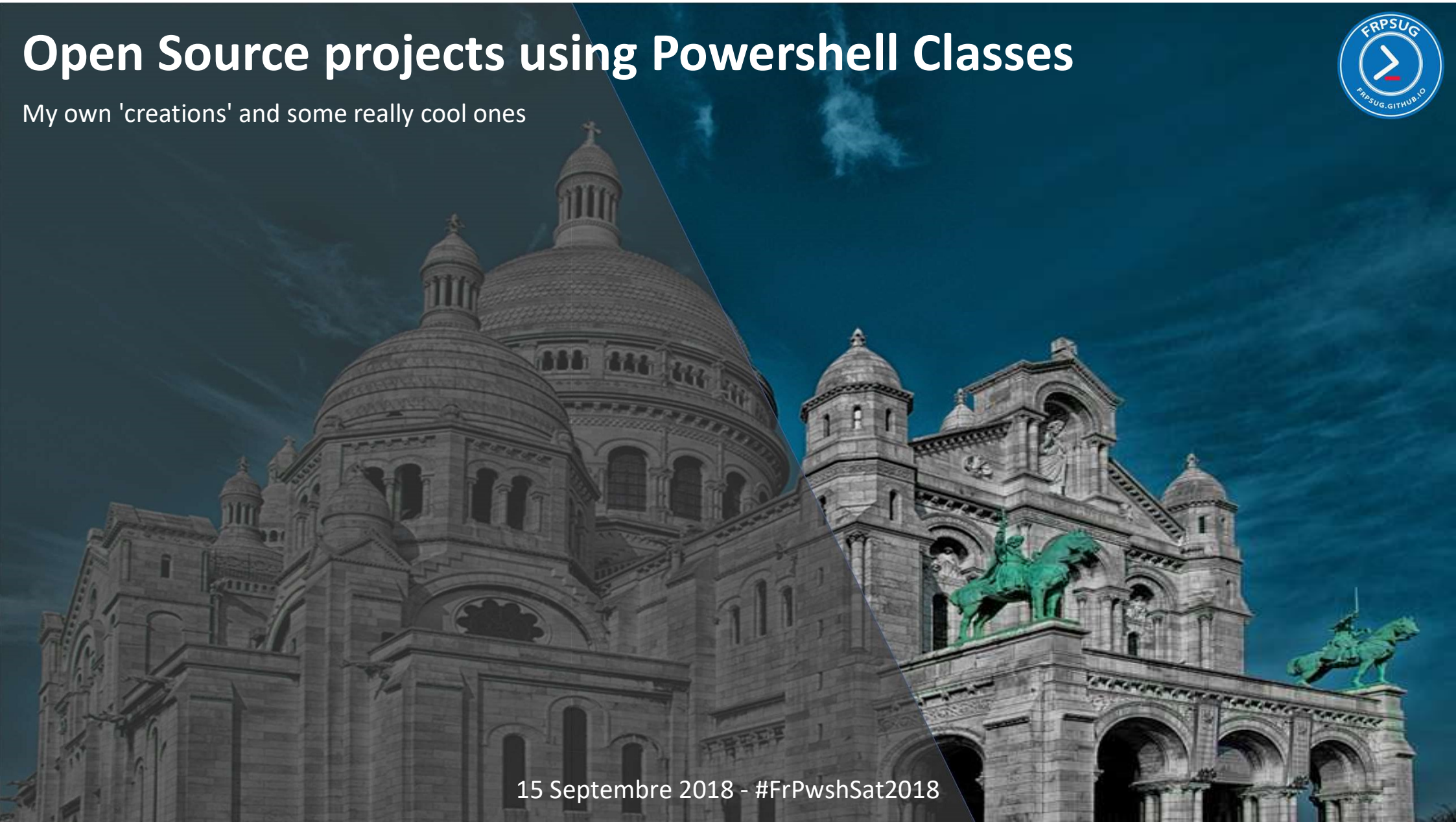
```powershell
Class Boat {
    [Animal[]]$Passengers = @()

    Boat([Animal[]]$Passengers){
        $this.Passengers += $Passengers
    }
}

$Array = @()
$Array += [Cat]::new("Minnie")
$Array += [Cat]::new("Jerry")
$Array += [dog]::new("Rex")
$Array += [dog]::new("Goofy")

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

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




```powershell
$NohasBoat = [Boat]::New($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:

Learning what an object was

Learning class basics

Making mistakes

Reusing knowledge

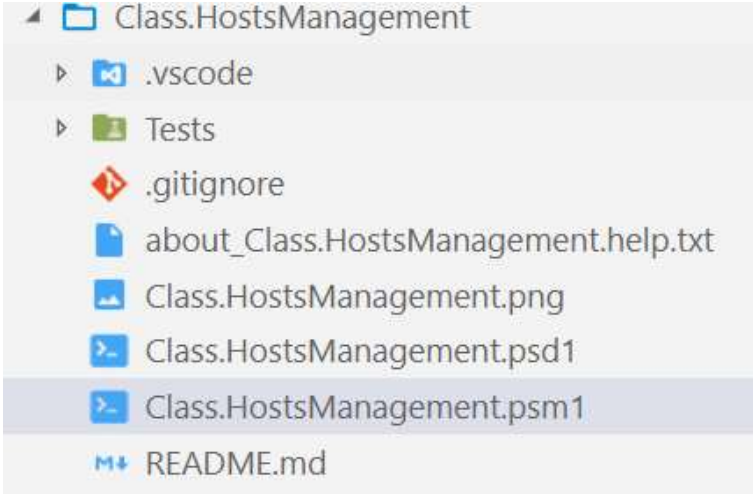
Looking forward

Class.HostsManagement: Me (Stéphane van Gulick)

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

Class.HostsManagement.psm1

```
Stephanevg, a year ago | 2 authors (Stéphane van Gulick and others)
Enum HostsEntryType{
  Entry
  Comment
  BlankLine
}

Class HostsEntry{
  [Ipaddress]$Ipaddress
  [string]$HostName
  [String]$FullQuallifiedName
  [String]$Description
  [HostsEntryType]$EntryType

  HostsEntry([IpAddress]$IpAddress,[String]$HostName,[String]$FQDN,[String]$Description,[HostsEntryType]$Type){…
  HostsEntry([IpAddress]$IpAddress,[String]$HostName,[String]$FQDN,[String]$Description){…
  HostsEntry([String]$Comment,[HostsEntryType]$Type){…
  hidden HostsEntry([string]$Line){…
  HostsEntry(){…

  [HostsEntryType] static hidden GetLineType([string]$Line){…
}

Class HostsFile{
  hidden [HostsEntry[]]$Entries
  [string]$Path
  hidden [int]$LogRotation = 5

  HostsFile(){…
  HostsFile([String]$ComputerName){…
  HostsFile([System.IO.FileSystemInfo]$Path){…
  [void]ReadHostsFileContent(){…
  [array]GetEntries(){…
  [void]AddHostsEntry([HostsEntry[]]$entries){…
  [void]RemoveHostsEntry([HostsEntry[]]$entries){…
  [void]Backup([System.IO.DirectoryInfo]$BackupFolder){…
  [void]Backup(){…
  <#…
  [void]Set(){…

}
```



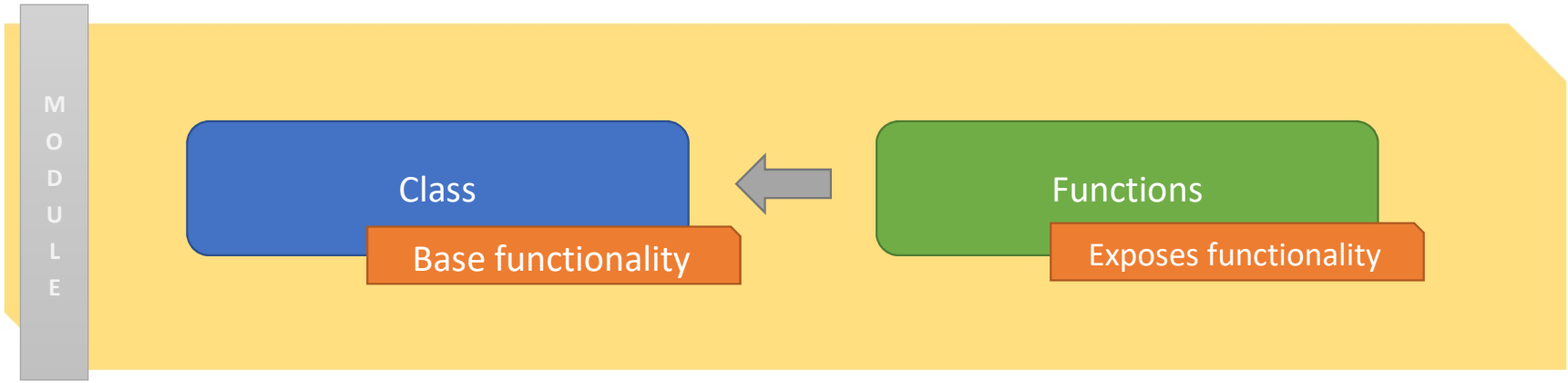
15 Septembre 2018 - #FrPwshSat2018

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

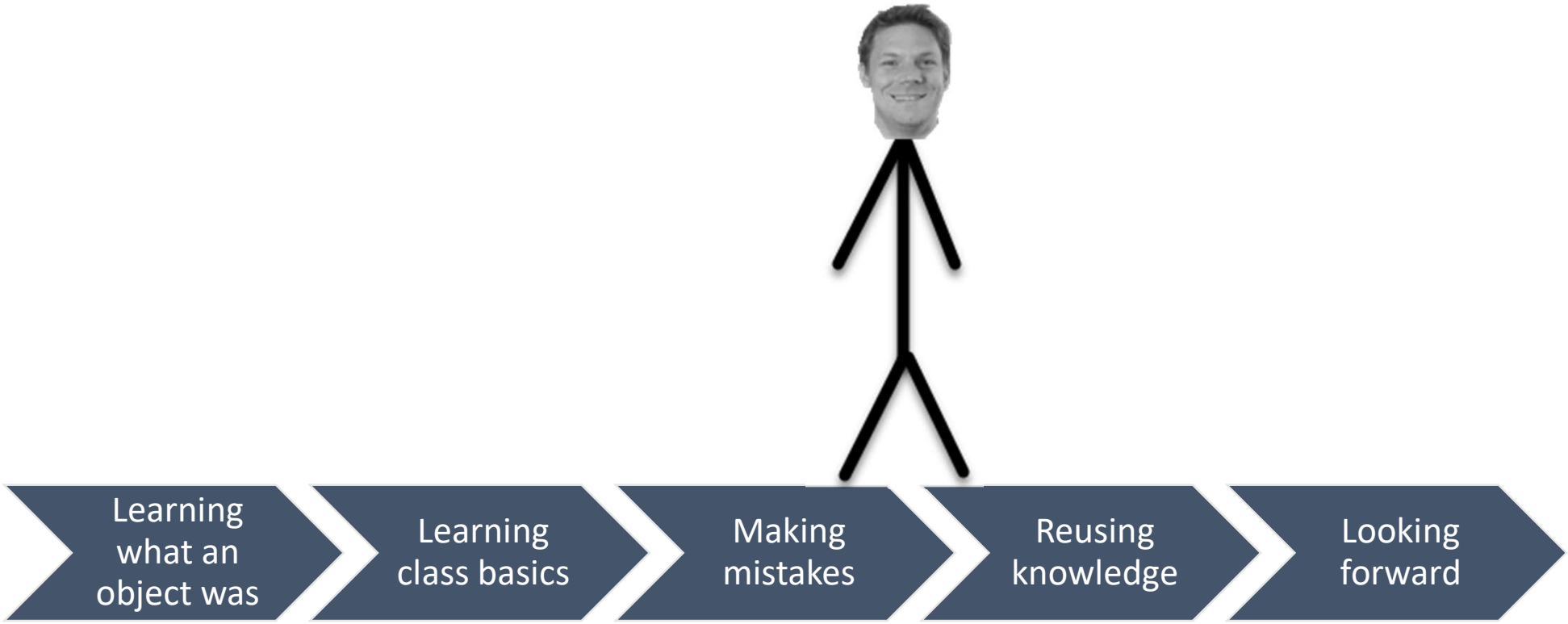



```
1   Class PackageDocument {
2       [system.io.Fileinfo]$File
3       [DocumentType]$Type
4       [CMPackage]$Document
5       [String]$SiteCode = (Get-CMSiteCode)
6
7       PackageDocument() {...}
9
10      PackageDocument([String]$File) {...}
30
31      [PackageDocument]AddPackage([CMPackage]$CMPackage) {...}
40
41      [PackageDocument] ExportDocument() {...}
116
117     [string] static GetDocumentTemplate([DocumentType]$Type) {...}
137
138     [string] SetSiteCode([String]$Sitecode) {...}
142
143     SetPsDrive() {...}
146
147     [Bool]ValidateDocument() {...}
156 }
```

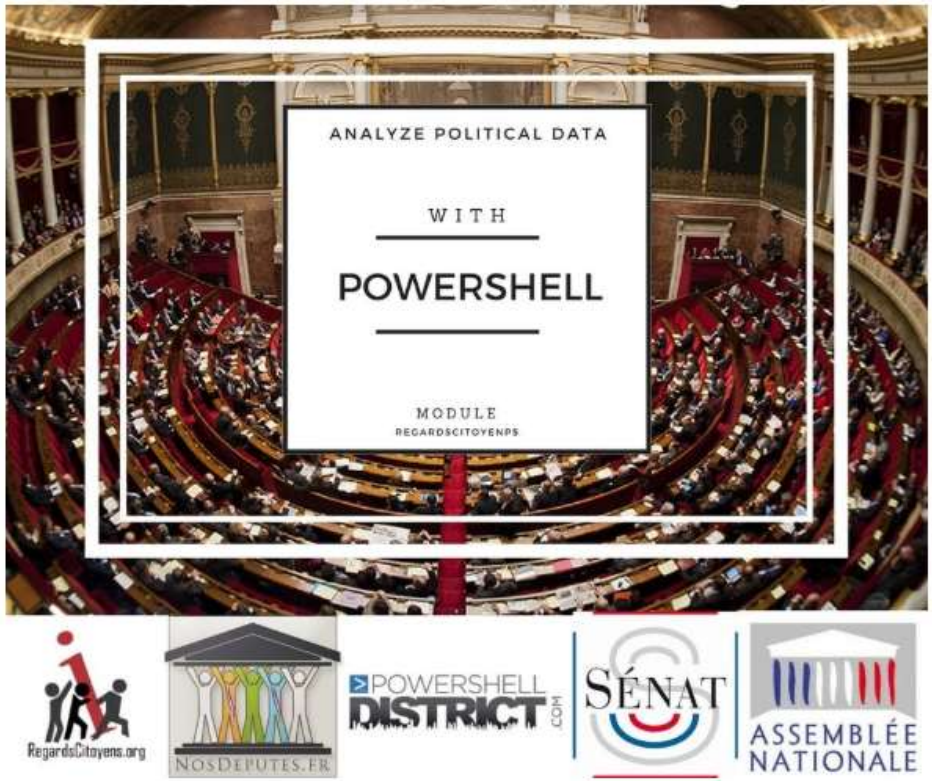
```
Function Get-PackageDocument {
    [CmdletBinding(
    Param(
        [String]$P
    )

    if ($Path) {
        $Document
        return $Do
    }

}
```

```
1735  Function Test-PackageDocument {
1736      <#...
1757      #>
1758      [CmdletBinding()]
1759      Param(
1760          [Parameter(Mandatory=$true)]
1761          [PackageDocument]$Document
1762      )
1763
1764      return $Document.ValidateDocument()
1765  }
```

Plan:


Learning what an object was
Learning class basics
Making mistakes
Reusing knowledge
Looking forward

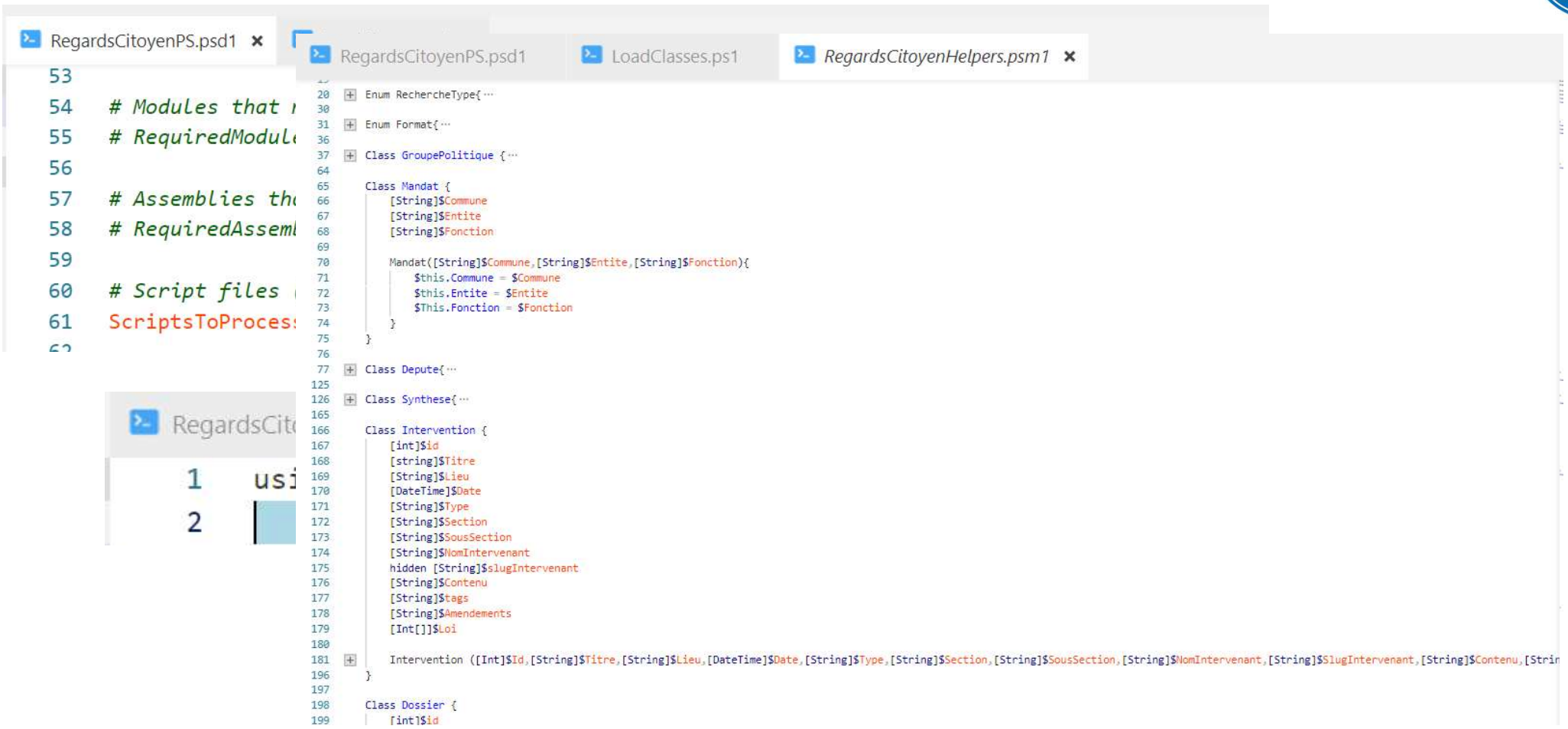
Learning by doing (the right way)



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



15 Septembre 2018 - #FrPwshSat2018

Example from the field

```
Class Dossier {
    [int]$id
    [string]$Titre
    [Datetime]$MinDate
    [DateTime]$MaxDate
    [int]$NbInterventions
    [Object[]]$Intervenants
    [Intervention[]]$Seances
    [String[]]$Documents
    [String[]]$SousSection
    hidden [string[]]$Id_intervenants
    hidden [string[]]$id_seances
    hidden [string[]]$id_documents
    hidden [string[]]$id_soussections


    Dossier([int]$id,[String]$Titre,[int]$NbInterventions,[DateTime]$minDate

    hidden [Void] _LoadIntervenants(){

    hidden [void] _LoadSeances(){

    hidden [Void] _LoadDocuments(){

        write-verbose "Chargement des seances.."
        if ($this.id_documents){
            $Docs += @()
            foreach ($id in $this.id_documents){

                $Docs += Get-RCDocument -id $id

            }

            $this.Documents = $Docs
        }


    }

    hidden [Void] _LoadSousSections(){

    [Dossier] Full(){
        $this._LoadSeances()
        $this._LoadDocuments()
        $this._LoadIntervenants()


        return $this
    }
```

```
Function Get-RCDossier {
    <#

    [cmdletBinding(DefaultParameterSetName="all")]
    Param(

        [Parameter(Mandatory=$false,ParameterSetName="id")]
        [ValidateNotNullOrEmpty()]
        [String]$id,

        [Parameter(Mandatory=$false)]
        [Switch]$Full

    )

    switch ($PsCmdlet.ParameterSetName) {


        "All"{

            try{
                $data = Invoke-RestMethod -Uri $RC_data.urls.Dossiers -ErrorAction Stop
            }Catch [System.Net.WebException]{
                write-warning "Server indisponible: Merci de vérifier vôtre connection internet."
                break

            }Catch{
                $_.exception.message
            }

            Foreach ($entry in $Data.sections.section){

                $ret = (invoke-restmethod $Entry.url_nosdeputes_api).Section

                $id_seances = $ret.seances.seance.id
                $id_intervenants = $ret.intervenants.parlementaire.slug
                $id_soussections = $ret.soussections.soussection.id
                $id_documents = $ret.documents.document.id

                $Dos = [Dossier]::New($ret.id,$ret.Titre,$ret.nb_interventions,$ret.min_date,$ret.max_date,$id_intervenants,$id_sea

                if ($Full){

                    $Dos.Full()

                }else{
```

PsClassUtils – Stéphane van Gulick (me)

Simple classes

```powershell
Class ClassProperty {
    [String]$Name
    [String]$Type

    ClassProperty([String]$Name,[String]$Type){

        $this.Name = $Name
        $This.Type = $Type
    }
}
```

```powershell
Class ClassParameter {
    [String]$Name
    [String]$Type

    ClassParameter([String]$Name,[String]$Type){

        $this.Name = $Name
        $This.Type = $Type
    }
}
```

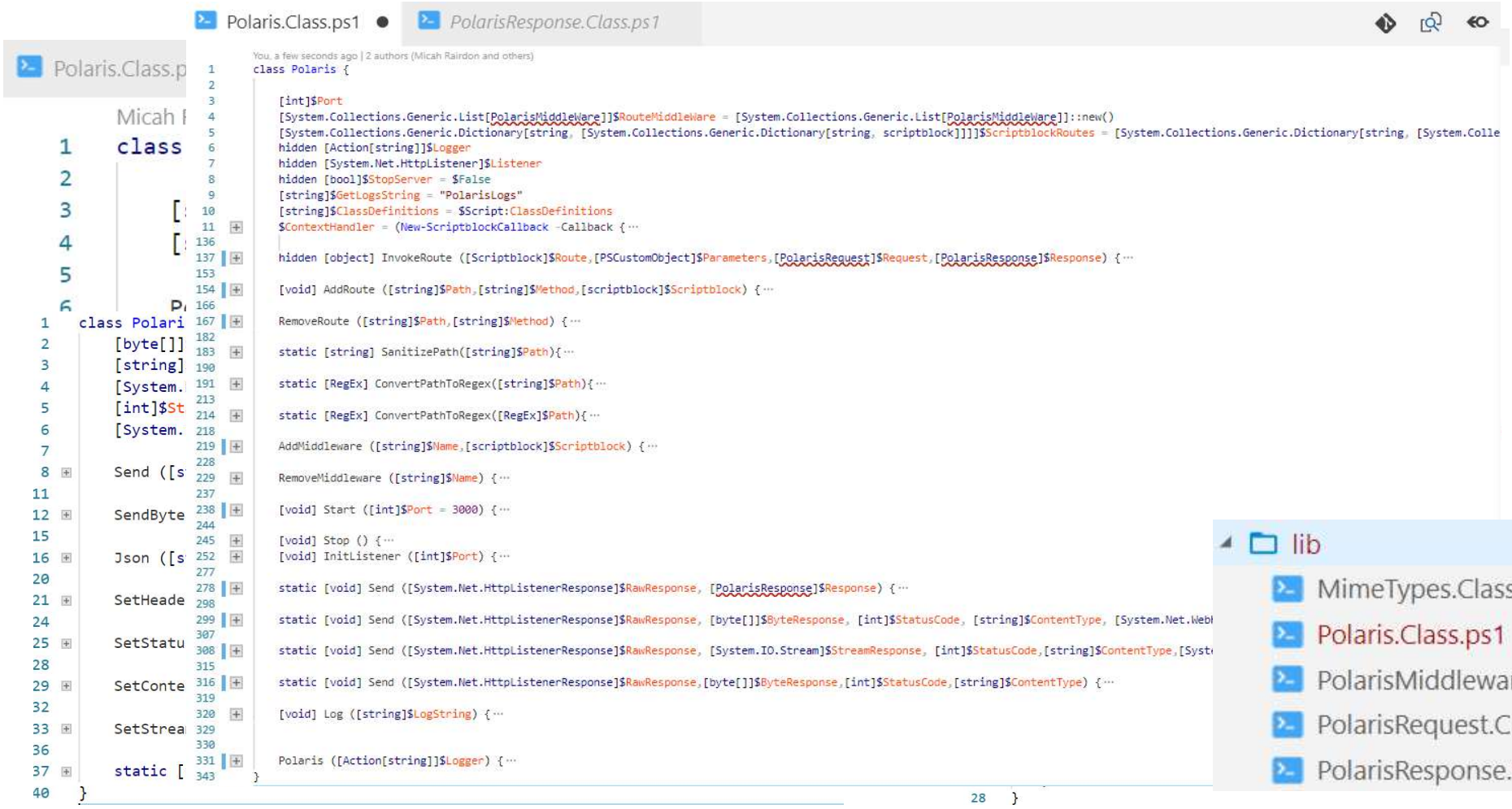
```powershell
Class ClassMethod {
    [String]$Name
    [String]$ReturnType
    [ClassParameter[]]$Parameter

    ClassMethod([String]$Name,[String]$ReturnType,[ClassParameter[]]$Parameters){
        $this.Name = $Name
        $This.ReturnType = $ReturnType
        $This.Parameter = $Parameters
    }

}
```

Unsaved changes (cannot determine recent change or authors)

```powershell
Class ClassConstructor {
    [String]$Name
    [ClassParameter[]]$Parameter

    ClassConstructor([String]$Name,[ClassParameter[]]$Parameter){
        $this.Name = $Name
        $This.Parameter = $Parameter
    }

}
```

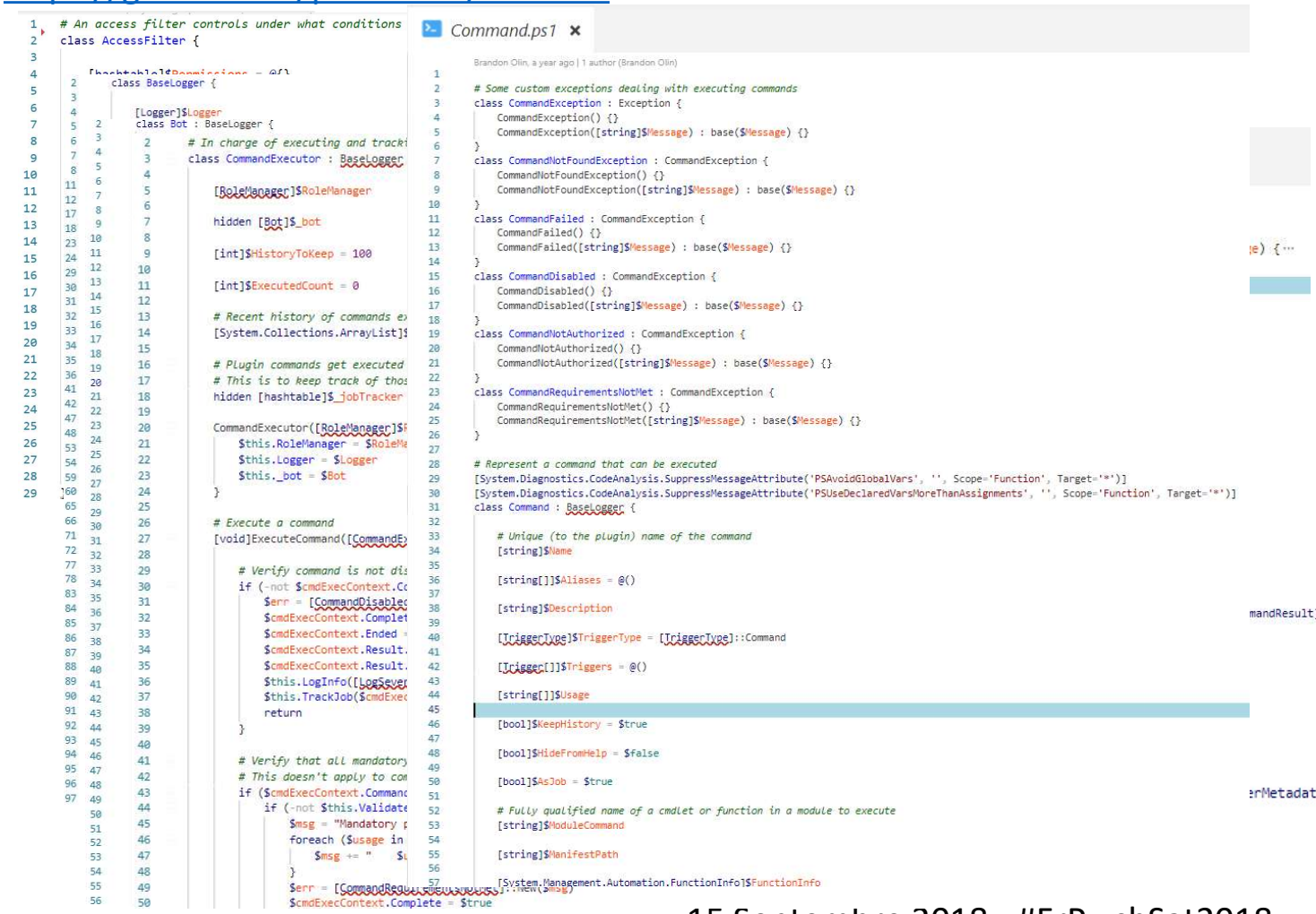
Polaris – Microsoft

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

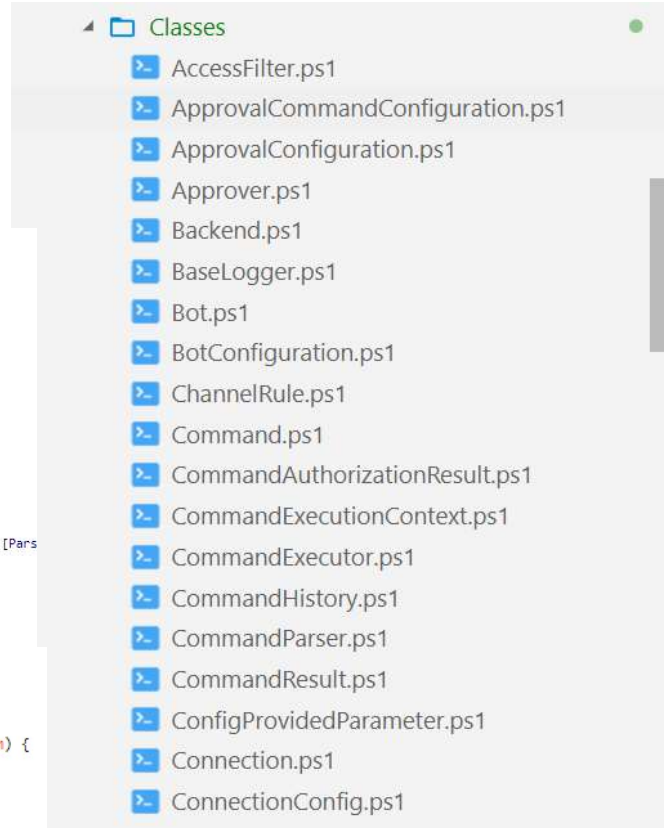


PoshBot – Brandon Olin

https://github.com/poshbotio/PoshBot



43 Classes!



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:

Learning what an object was

Learning class basics

Making mistakes

Reusing knowledge

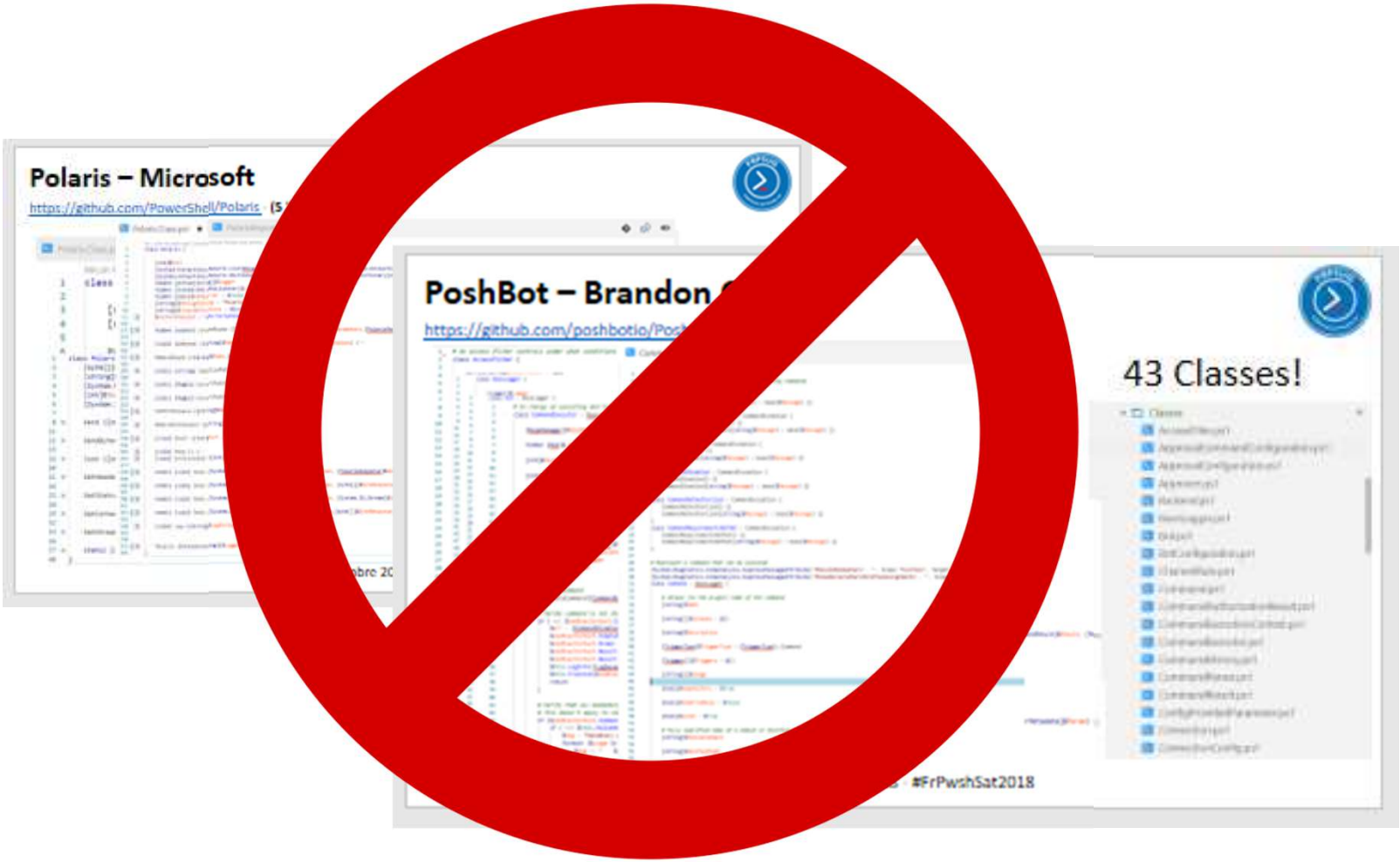
Looking forward

Communicate via diagrams: UML

A picture is worth a thousand words

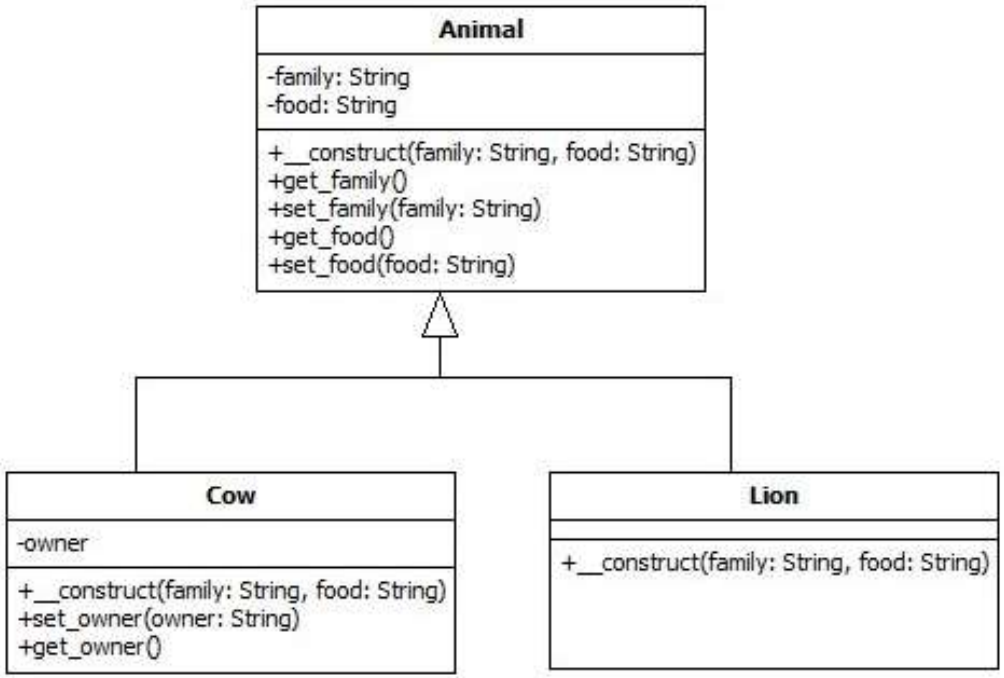
15 Septembre 2018 - #FrPwshSat2018

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

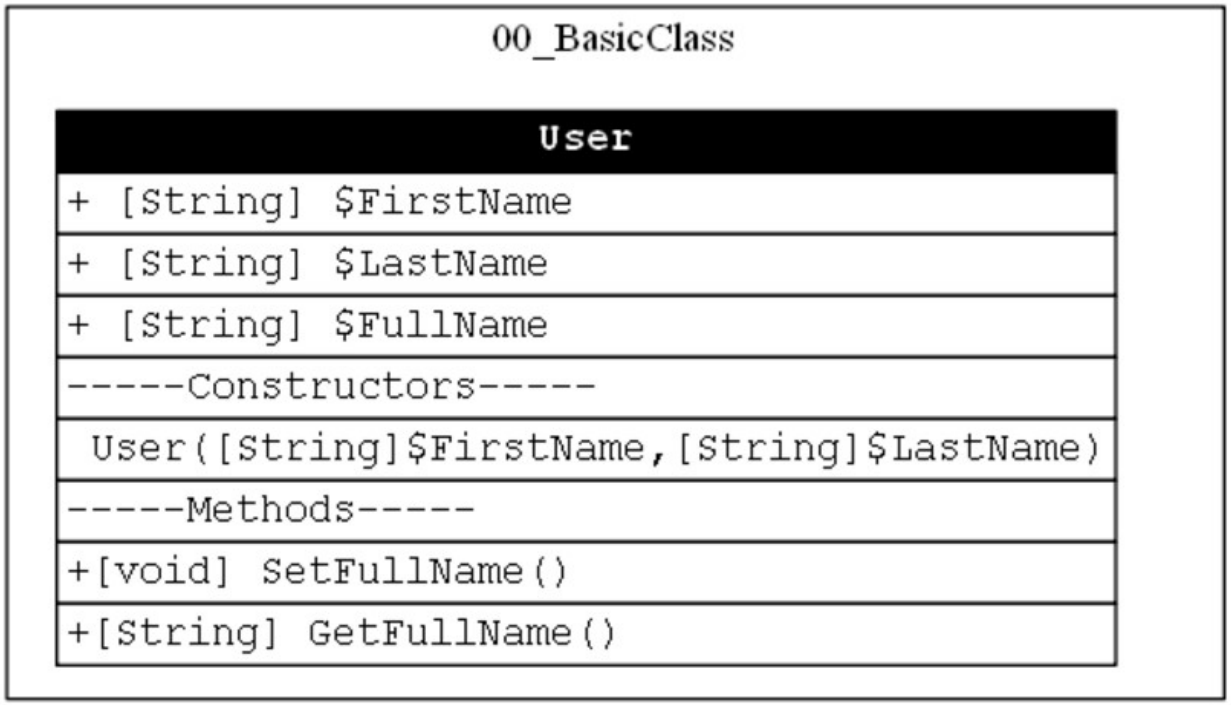

```
>  find-module psclassutils | install-module
```

```
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

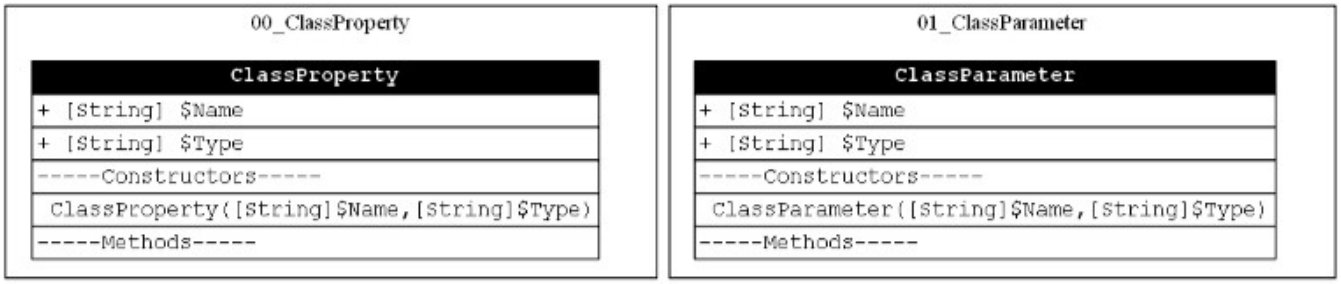
Visual communication

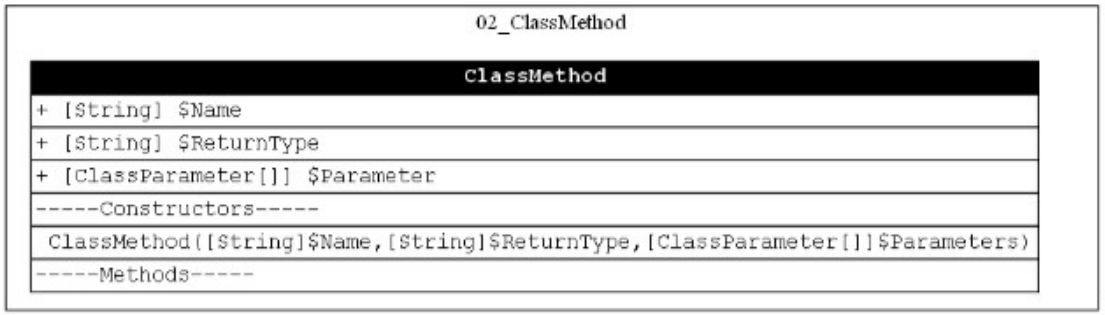
```
Class User {
    #Properties
    [String]$FirstName
    [String]$LastName
    [String]$FullName

    #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
    }
}
```

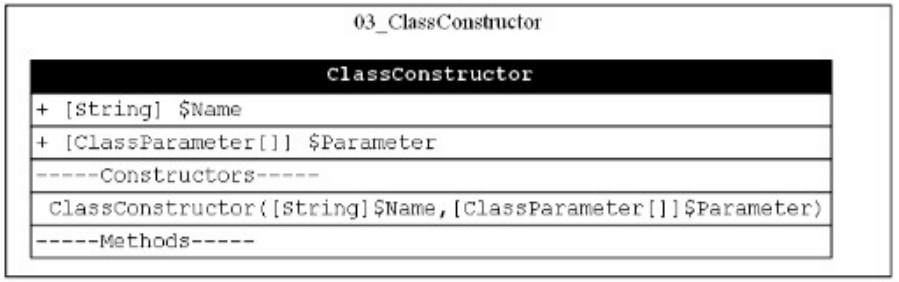


PsClassUtils – Stéphane van Gulick (me)

Simple class

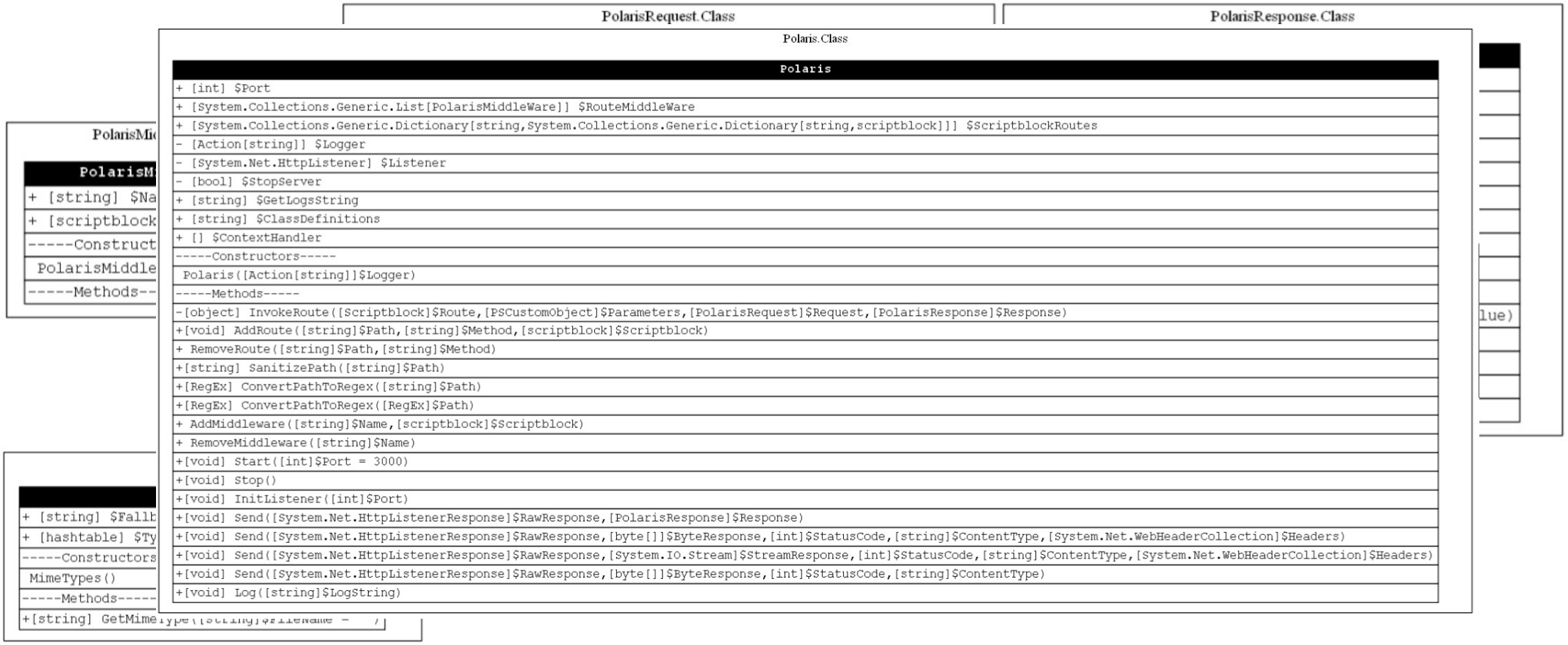






Polaris - Microsoft

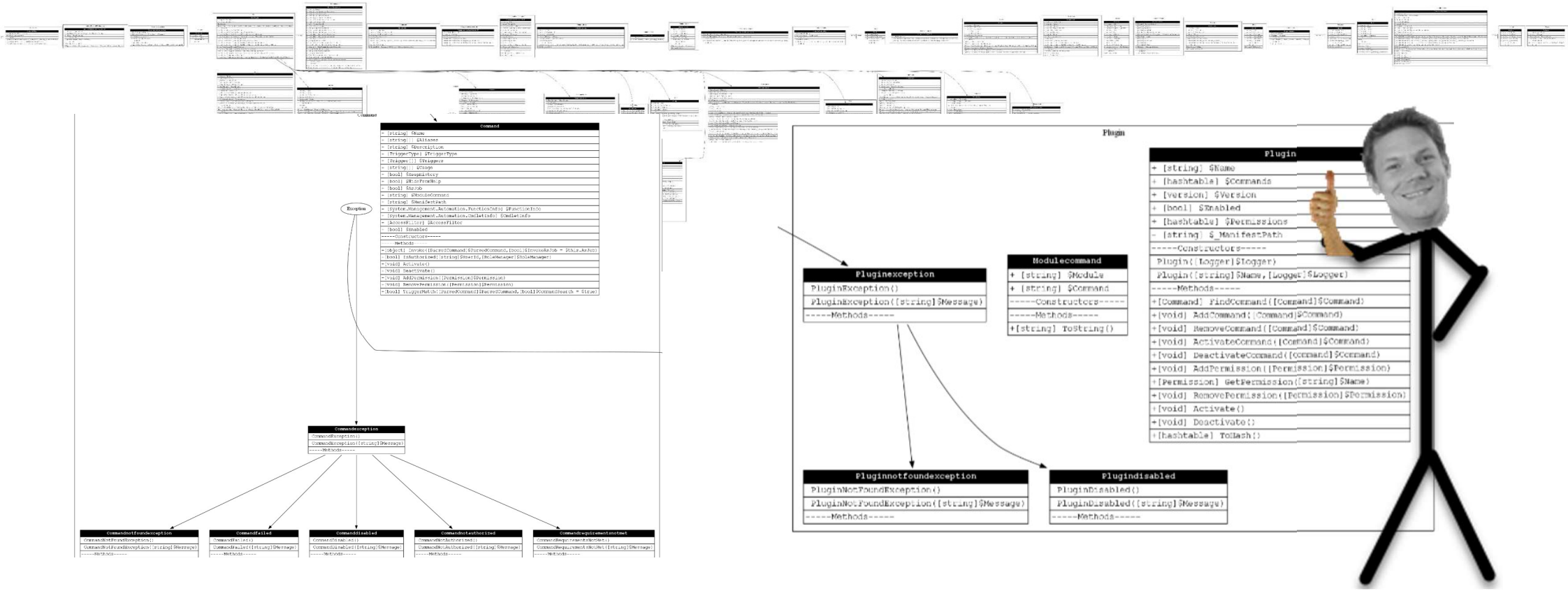
https://github.com/PowerShell/Polaris



15 Septembre 2018 - #FrPwshSat2018

PoshBot -

Yep…



15 Septembre 2018 - #FrPwshSat2018

Plan:

Learning what an object was

Learning class basics

Making mistakes

Reusing knowledge

Looking forward

Chill:

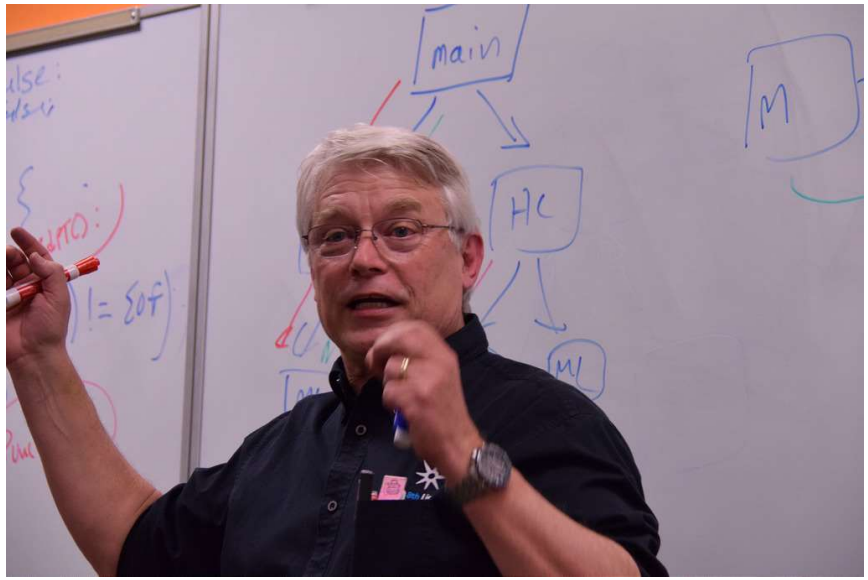
15 Septembre 2018 - #FrPwshSat2018

Use well known standards:

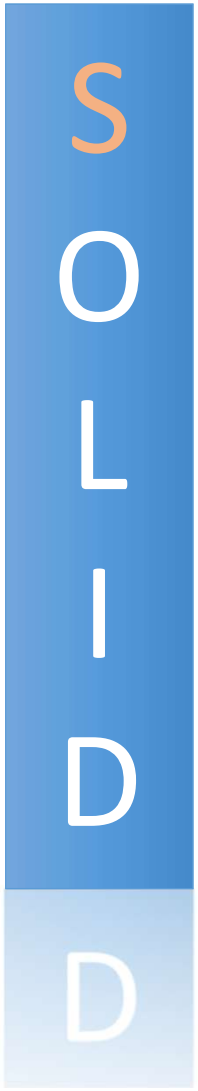
SOLID

Use well known standards:

Robert C. Martin (Aka – Uncle Bob)



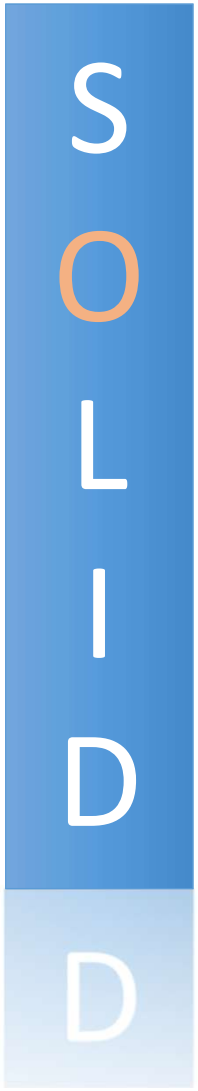
Use well known standards:



Single responsibility principle

One class / method **must** have
only *one responsibility*

Use well known standards:



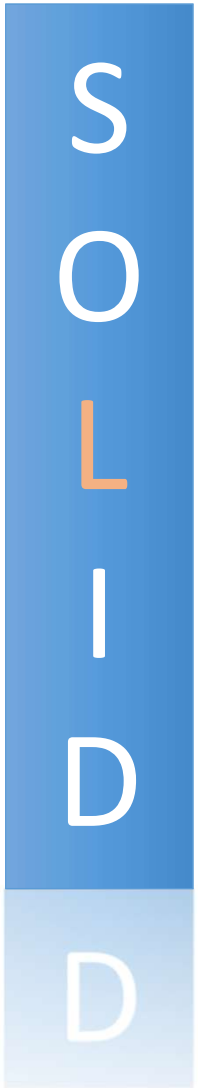
Open Closed Principle

Open for extension

But

Closed for modification

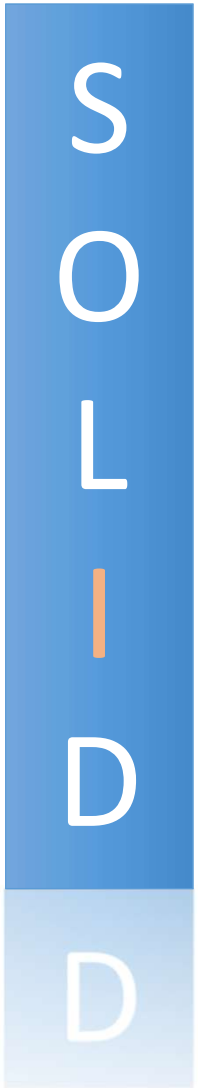
Use well known standards:



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:

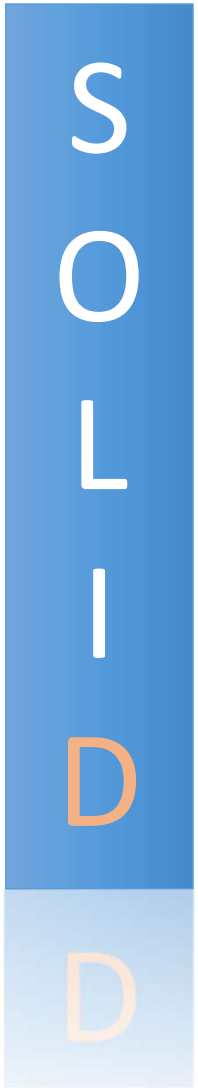


Interface segragation principle

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:

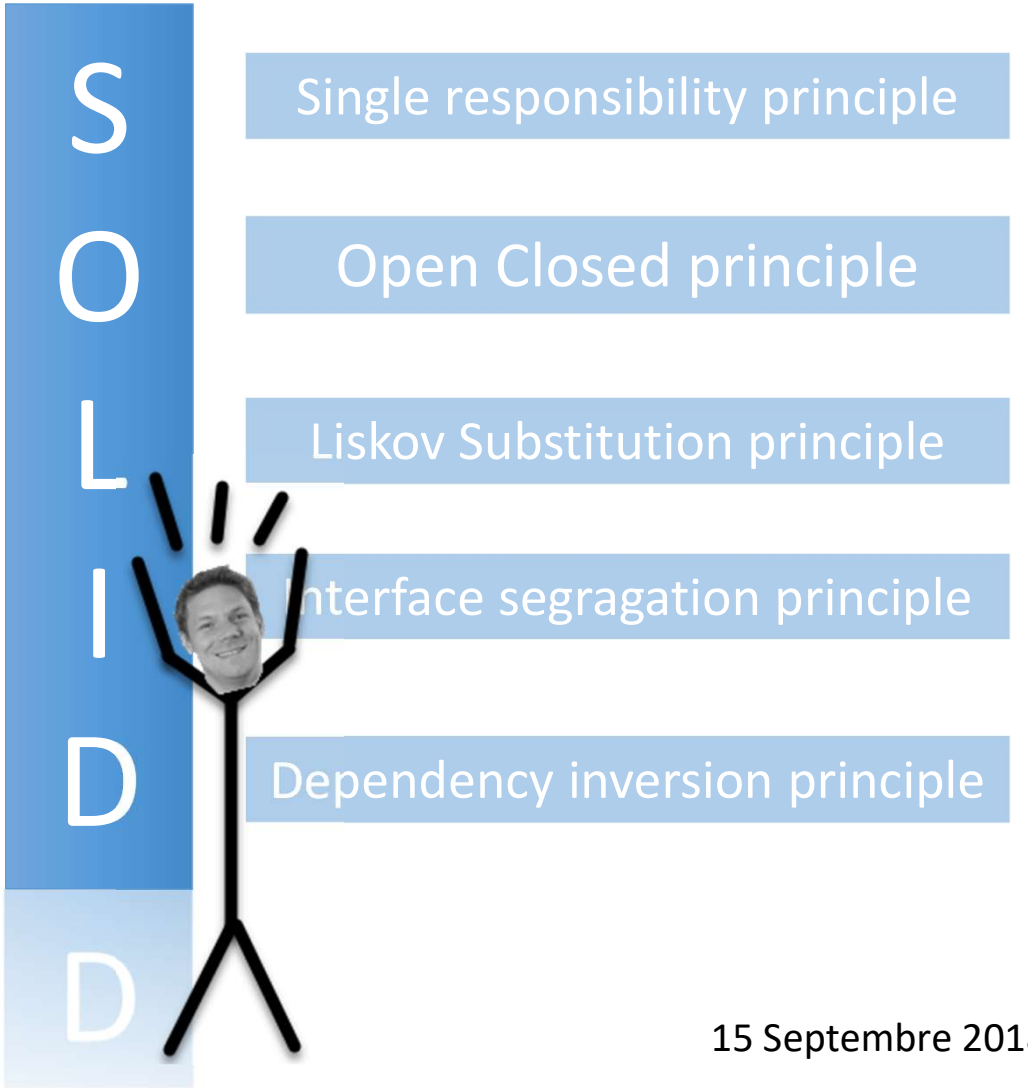


Dependency inversion principle

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:

Learning what an object was

Learning class basics

Making mistakes

Reusing knowledge

Looking forward

Use well known standards:

TDD

Test Driven Developement

15 Septembre 2018 - #FrPwshSat2018

Test Driven Developement:

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

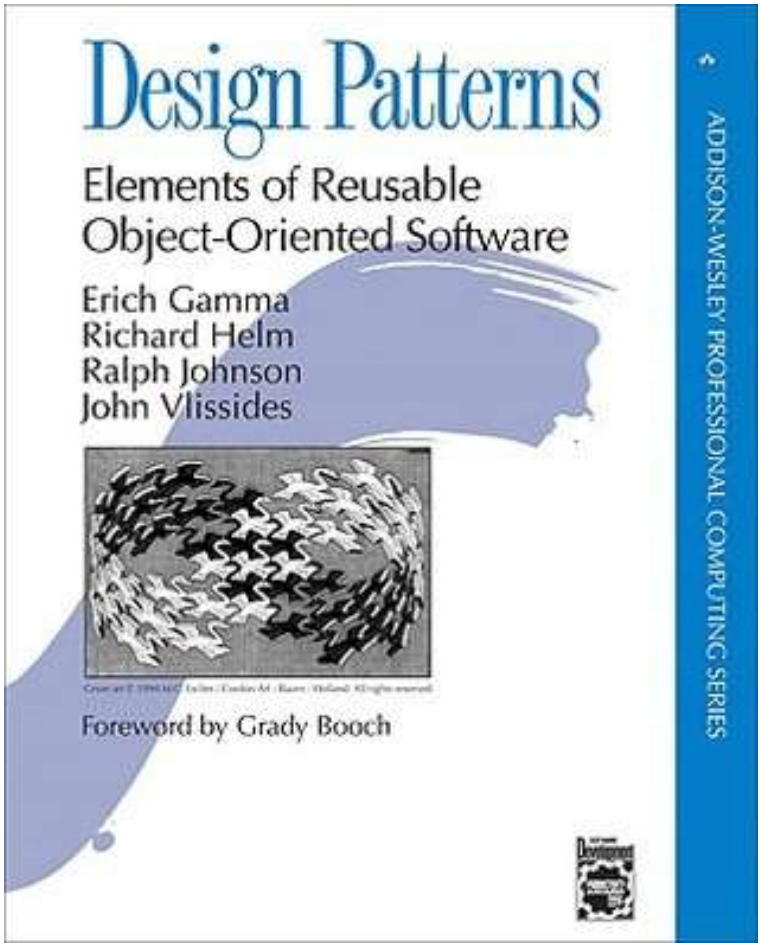
2

3

3

Design patterns

Gang of four

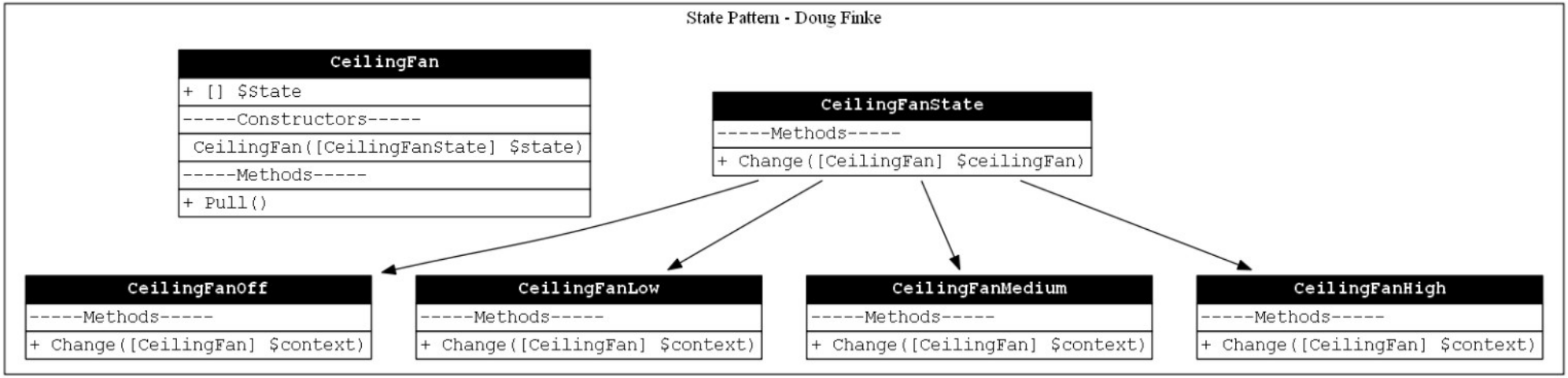


Design pattern: Example

State Pattern

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

```
class CeilingFanHigh: CeilingFanState {

    Change([CeilingFan] $context) {
        $context.State = [CeilingFanLow]::new()
        "Change state from High to Off." | 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

```
class CeilingFa
    [CeilingFan
    CeilingFan
    Pull() { $t
}
```

```
59  $ceilingFan.Pull()
60  $ceilingFan.Pull()
61  $ceilingFan.Pull()
62  $ceilingFan.Pull()
```

s.State = $state }

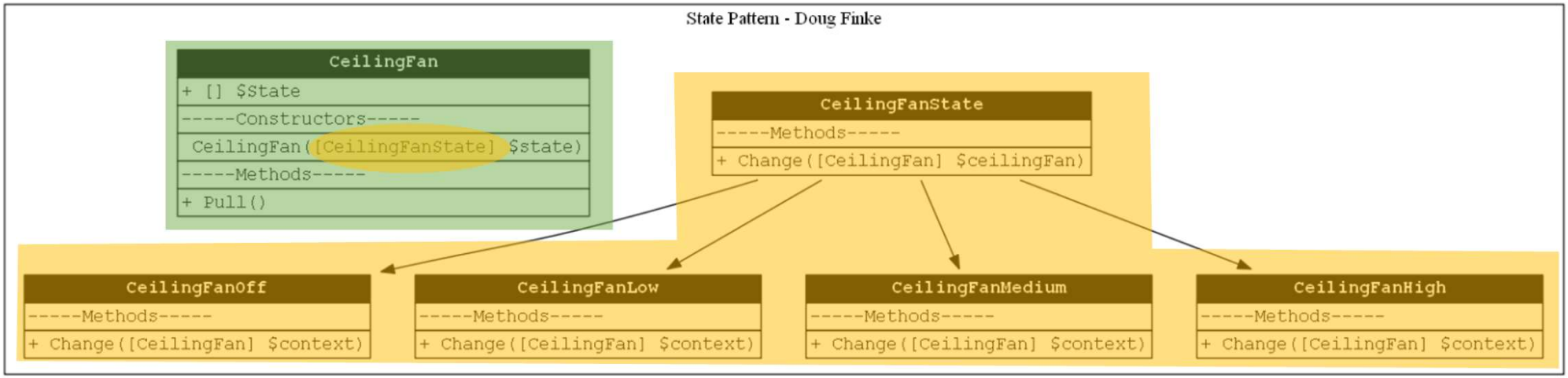
PROBLEMS 3 OUTPUT DEBUG CONSOLE

```
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

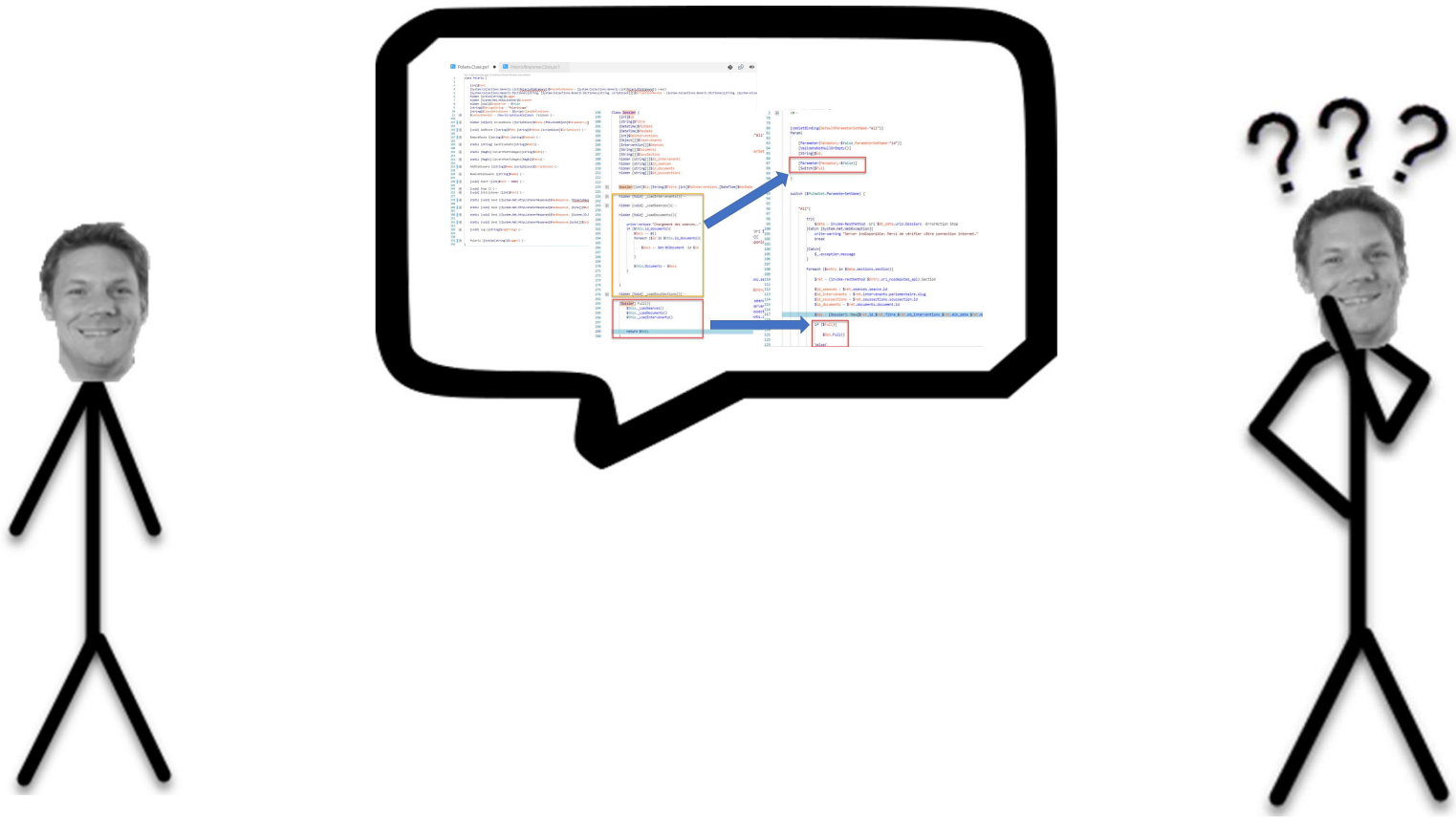
State Pattern

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

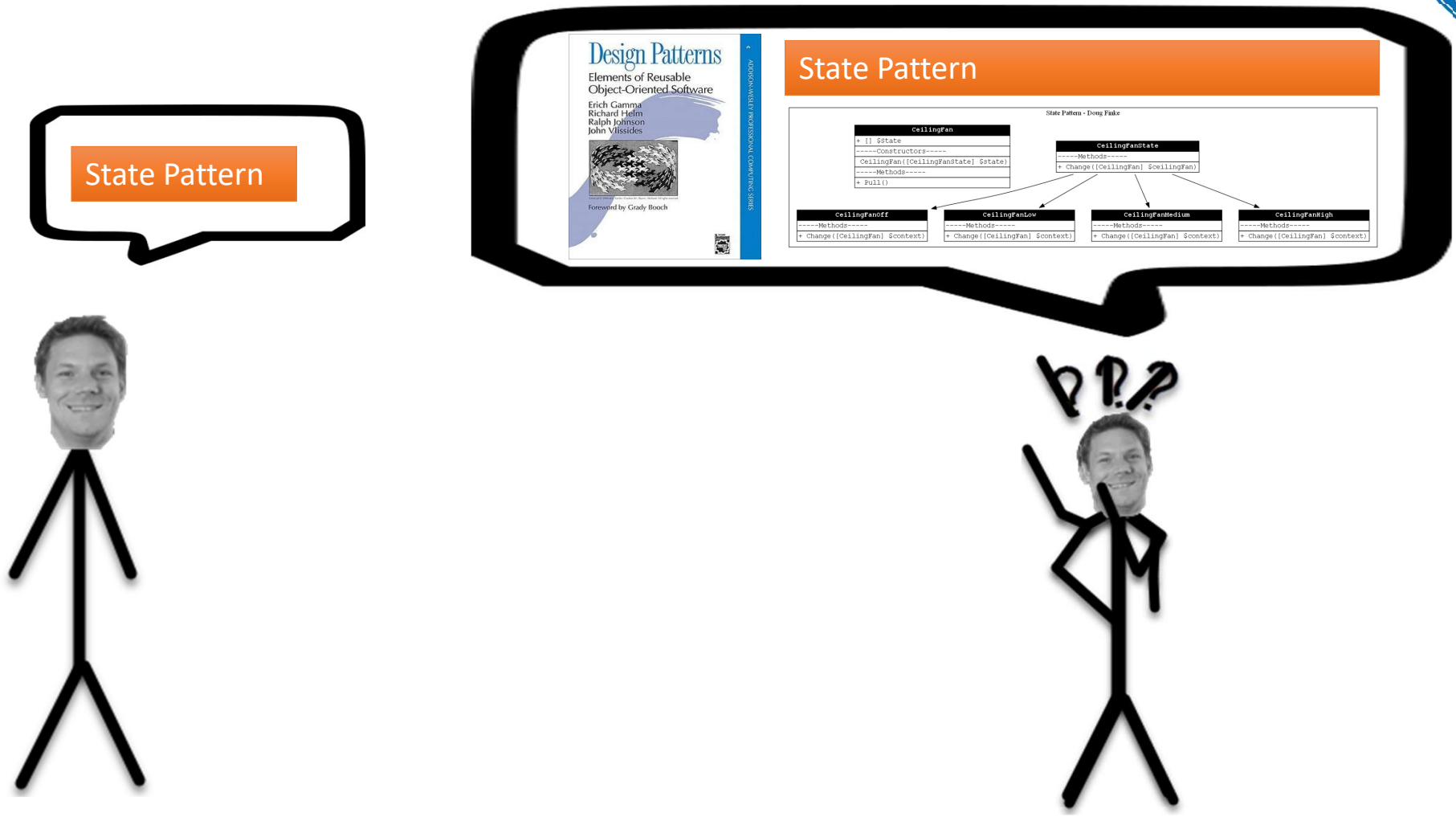
State Pattern

- Standard Answer to a common / Well known problem
- Reusable design
- Reusable Code

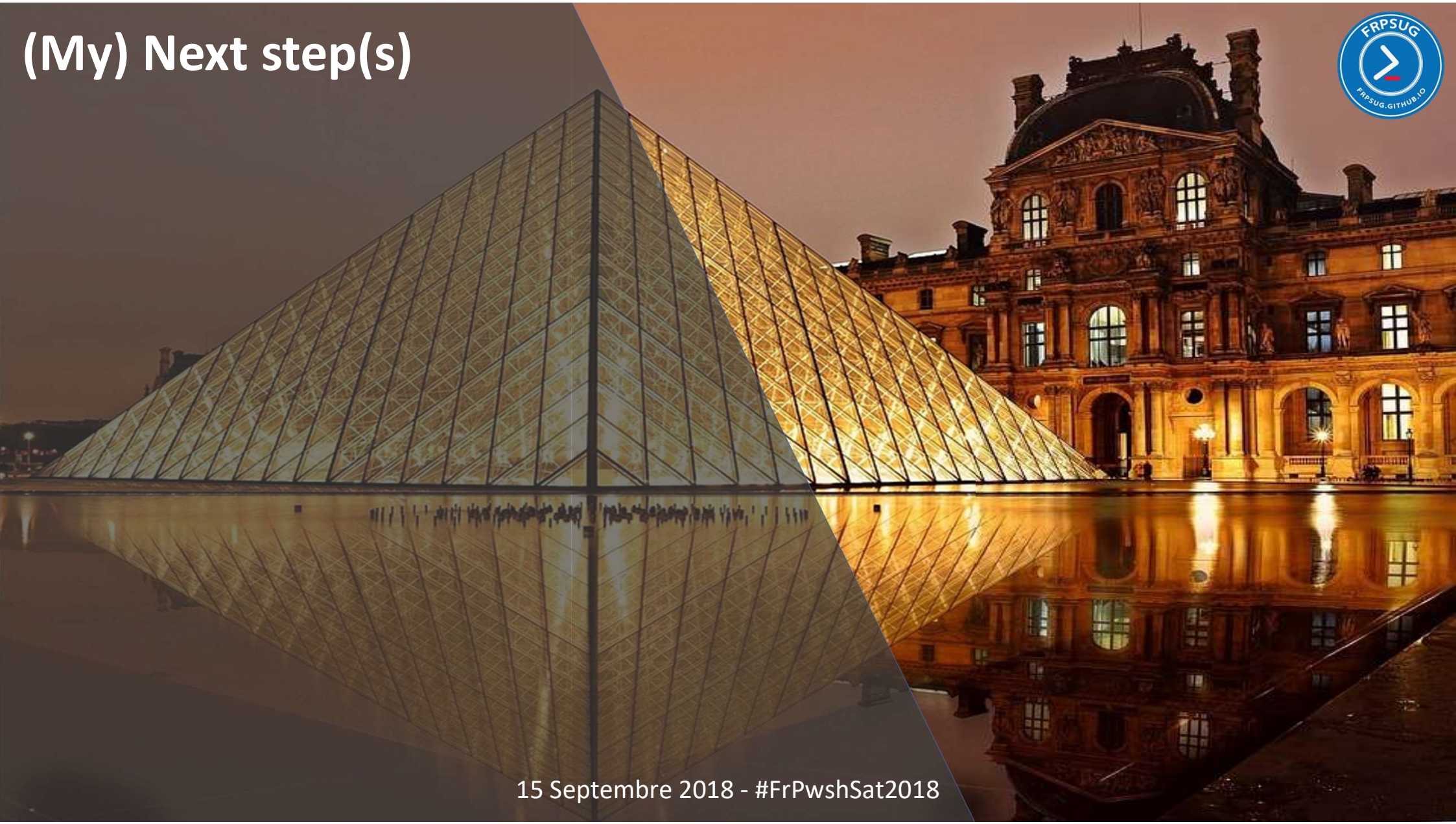
Design pattern: Key take aways



Design pattern: Key take aways



(My) Next step(s)



15 Septembre 2018 - #FrPwshSat2018

Plan:

Learning what an object was → Learning class basics → Making mistakes → Getting curious → Looking forward

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
- Facede Pattern
- Bridge Pattern
- Decorator Pattern

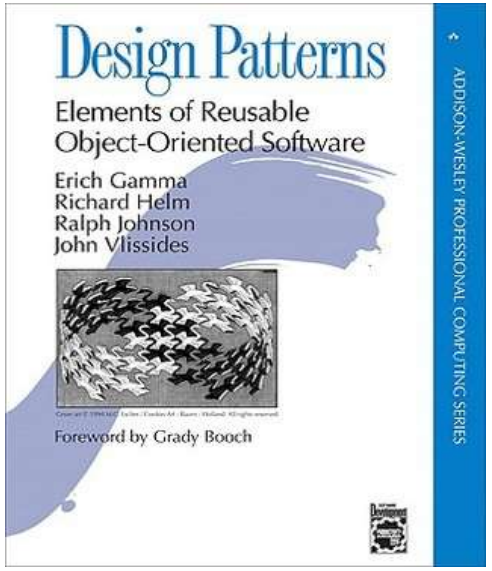
Behavioral Design Patterns

- State Pattern
- Observer Pattern
- Strategy Pattern
- Iterator Pattern
- Visitor Pattern
- Template Method
- Mediator Pattern
- Command Pattern
- Interpreter Pattern
- Memento Pattern
- Chain of responbility 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
FRPSUG
FRPSUG.GITHUB.IO

15 Septembre 2018 - #FrPwshSat2018

French PowerShell Saturday

Merci à nos sponsors !!

METSYS

smartview
conseil et formation

15 Septembre 2018 - #FrPwshSat2018