Iron Scripter 2018: Prequel 7

A touch of class

The Puzzle

Greetings Iron Scripters. The countdown to Iron Scripter continues. The ultimate contest will soon be upon you.

In this weeks challenge you're required to create a PowerShell class. The class should have the following properties:

- Computer Name
- BIOS manufacturer and version
- Domain of which the machine is a member – you'll need to decide how to handle non-domain machines
- Number of processors and number of cores
- Total physical memory in GB
- Operating System name
- Operating System Architecture
- Time zone
- Size of C: drive (GB)
- C: drive free space (GB)

The class should have default constructor and a constructor that accepts a Computer name and domain as parameters. Methods should be created to populate the remaining properties. You should also create a method that will calculate the percentage free space on the C: drive.

Is it possible to create the class so that it will automatically populate ALL properties of the class when an instance of the class is instantiated on a machine?

Use best practice if it doesn't conflict with your faction's aims. The solution must be acceptable to your faction:

- Daybreak Faction - beautiful code
- Flawless Faction - flawless code
- Battle Faction - good enough to get the job done

Good luck and good coding.

The commentary

First step is to create the class. I've made all of the properties public:

```
class SystemInfo {

  [string] $ComputerName
  [string] $BIOSmanufacturer
  [string] $BIOSversion
  [string] $Domain
  [int] $NumberOfProcessors
  [int] $NumberOfCores
  [double] $TotalPhysicalMemory
  [string] $OperatingSystemName
  [string] $OperatingSystemArchitecture
  [string] $TimeZone
  [double] $SizeOfCdrive
  [double] $CdriveFreeSpace


}
[SystemInfo]::new()
```

The newly created empty object looks like this:

```
ComputerName                :
BIOSmanufacturer            :
BIOSversion                 :
Domain                      :
NumberOfProcessors          : 0
NumberOfCores               : 0
TotalPhysicalMemory         : 0
OperatingSystemName         :
OperatingSystemArchitecture :
TimeZone                    :
SizeOfCdrive                : 0
CdriveFreeSpace             : 0
```

You're using the default constructor to create an empty instance of the class. Notice that you didn't create a constructor – you get this for free.

You can also use New-Object:

```
PS> New-Object -TypeName SystemInfo


ComputerName                :
BIOSmanufacturer            :
BIOSversion                 :
Domain                      :
NumberOfProcessors          : 0
NumberOfCores               : 0
TotalPhysicalMemory         : 0
OperatingSystemName         :
OperatingSystemArchitecture :
TimeZone                    :
SizeOfCdrive                : 0
CdriveFreeSpace             : 0
```

Next step is to work out how to populate the class' properties:

```
class SystemInfo {

  [string] $ComputerName
  [string] $BIOSmanufacturer
  [string] $BIOSversion
  [string] $Domain
  [int] $NumberOfProcessors
  [int] $NumberOfCores
  [double] $TotalPhysicalMemory
  [string] $OperatingSystemName
  [string] $OperatingSystemArchitecture
  [string] $TimeZone
  [double] $SizeOfCdrive
  [double] $CdriveFreeSpace
```

```
}

$sinfo = [SystemInfo]::new()

$compsys = Get-CimInstance -ClassName Win32_ComputerSystem

$sinfo.ComputerName = $compsys.Name
$sinfo.TotalPhysicalMemory = [math]::Ceiling($compsys.TotalPhysicalMemory / 1GB)

$sinfo.Domain = $compsys.Domain

$sinfo.NumberOfProcessors = $compsys.NumberOfProcessors

$proc = Get-CimInstance -ClassName Win32_processor
$sinfo.NumberOfCores = $proc.NumberOfCores

$bios = Get-CimInstance -ClassName Win32_Bios
$sinfo.BIOSmanufacturer = $bios.Manufacturer
$sinfo.BIOSversion = $bios.Version

$os = Get-CimInstance -ClassName Win32_OperatingSystem
$sinfo.OperatingSystemName = $os.Caption
$sinfo.OperatingSystemArchitecture = $os.OSArchitecture

$sinfo.TimeZone = (Get-TimeZone).DisplayName

$disk = Get-CimInstance -ClassName Win32_LogicalDisk -Filter "DeviceID='C:'"

$sinfo.SizeOfCdrive = [math]::Round(($disk.Size / 1GB), 2)
$sinfo.CdriveFreeSpace = [math]::Round(($disk.FreeSpace / 1GB), 2)

$sinfo
```

For the most part the data is pulled from the relevant CIM class.

For Total Physical Memory I've rounded up the value from Win32_ComputerSystem as that class misses out the bit of memory the system reserves. As an alternative you could use:

```
(Get-CimInstance -ClassName Win32_PhysicalMemory | Measure-Object Capacity -Sum).Sum / 1GB
```

Win32_ComputerSystem reports the number of logical processors but this is doubled if Hyper-Threading is enabled. I prefer to know the real number of cores. If you want to report the larger number than feel free or even add another property to the class.

The time zone can also be found using CIM

```
PS> Get-CimInstance -ClassName Win32_TimeZone | select -ExpandProperty Caption
(UTC+00:00) Dublin, Edinburgh, Lisbon, London
```

You also have the alternative of using Get-TimeZone.

I filtered the Win32_LogicalDisk call to only get the C: drive information.

Next step is to add the constructors and the method(s) to populate the class.

Let's start with the constructors.

```
class SystemInfo {

  ## class properties
  [string] $ComputerName
  [string] $BIOSmanufacturer
  [string] $BIOSversion
  [string] $Domain
  [int] $NumberOfProcessors
  [int] $NumberOfCores
  [double] $TotalPhysicalMemory
  [string] $OperatingSystemName
  [string] $OperatingSystemArchitecture
  [string] $TimeZone
  [double] $SizeOfCdrive
  [double] $CdriveFreeSpace

  ## default constructor
  SystemInfo(){}
```

```
  ## constructor with arguments
  SystemInfo ([string]$ComputerName, [string] $Domain ){

     if (($ComputerName -eq '') -or ($ComputerName -eq $null)) {
       throw [InvalidOperationException]::new(
       "ComputerName is empty or null")
     }

     if (($Domain -eq '') -or ($Domain -eq $null)) {
       throw [InvalidOperationException]::new(
       "Domain is empty or null")
     }

     $this.ComputerName = $ComputerName
     $this.Domain = $Domain
  }

}

[SystemInfo]::new()
```

A constructor is of the form:

```
SystemInfo(){}
```

This is the default constructor that creates and empty object:

```
ComputerName               :
BIOSmanufacturer           :
BIOSversion                :
Domain                     :
NumberOfProcessors         : 0
NumberOfCores              : 0
TotalPhysicalMemory        : 0
OperatingSystemName        :
OperatingSystemArchitecture :
TimeZone                   :
SizeOfCdrive               : 0
CdriveFreeSpace            : 0
```

NOTE:- once you add a constructor with arguments you have to create the default constructor yourself as well.

A constructor with arguments builds on the default constructor

```
   ## constructor with arguments
  SystemInfo ([string]$ComputerName, [string] $Domain ){

     if (($ComputerName -eq '') -or ($ComputerName -eq $null)) {
       throw [InvalidOperationException]::new(
       "ComputerName is empty or null")
     }

     if (($Domain -eq '') -or ($Domain -eq $null)) {
       throw [InvalidOperationException]::new(
       "Domain is empty or null")
     }

     $this.ComputerName = $ComputerName
     $this.Domain = $Domain
  }
```

You have to define the argument AND its type – this is the signature of the constructor. Notice the use of $this to refer to the instances properties.

```
PS> [SystemInfo]::new('W510W10', 'WORKGROUP')


ComputerName               : W510W10
BIOSmanufacturer           :
BIOSversion                :
Domain                     : WORKGROUP
NumberOfProcessors         : 0
NumberOfCores              : 0
TotalPhysicalMemory        : 0
OperatingSystemName        :
OperatingSystemArchitecture :
```

```
TimeZone                  :
SizeOfCdrive              : 0
CdriveFreeSpace           : 0
```

When you create the object, the properties related to the arguments are populated.

You also have to perform any error checking yourself. This how it works:

```
PS> [SystemInfo]::new('', 'WORKGROUP')
ComputerName is empty or null
At C:\MyData\2018 Summit\Iron Scripter prequels\Puzzle07\working3.ps1:24 char:8
+         throw [InvalidOperationException]::new(
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], InvalidOperationException
    + FullyQualifiedErrorId : ComputerName is empty or null


PS> [SystemInfo]::new($null, 'WORKGROUP')
ComputerName is empty or null
At C:\MyData\2018 Summit\Iron Scripter prequels\Puzzle07\working3.ps1:24 char:8
+         throw [InvalidOperationException]::new(
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], InvalidOperationException
    + FullyQualifiedErrorId : ComputerName is empty or null


PS> [SystemInfo]::new('W510W10', '')
Domain is empty or null
At C:\MyData\2018 Summit\Iron Scripter prequels\Puzzle07\working3.ps1:29 char:8
+         throw [InvalidOperationException]::new(
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], InvalidOperationException
    + FullyQualifiedErrorId : Domain is empty or null


PS> [SystemInfo]::new('W510W10', $null)
Domain is empty or null
At C:\MyData\2018 Summit\Iron Scripter prequels\Puzzle07\working3.ps1:29 char:8
+         throw [InvalidOperationException]::new(
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], InvalidOperationException
    + FullyQualifiedErrorId : Domain is empty or null
```

But what if you only supply 1 argument?

```
PS> [SystemInfo]::new('W510W10')
Cannot find an overload for "new" and the argument count: "1".
At line:1 char:1
+ [SystemInfo]::new('W510W10')
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [], MethodException
    + FullyQualifiedErrorId : MethodCountCouldNotFindBest
```

The class doesn't have a constructor with that signature and throws an error.

That's the constructors built now let's look at the methods. The puzzle was (deliberately) vague on how many methods were needed. I'm going to use 6 – one for each call to a cmdlet or CIM class. I'll then use an overriding method to call all of the other methods so I can populate all properties in one hit – one method to find them and in the dark to bind them precious…

Ahem. Normal service is now resumed.

Adding the methods gives us this:

```
class SystemInfo {

  ## class properties
  [string] $ComputerName
  [string] $BIOSmanufacturer
  [string] $BIOSversion
  [string] $Domain
  [int] $NumberOfProcessors
  [int] $NumberOfCores
  [double] $TotalPhysicalMemory
```

```
  [string] $OperatingSystemName
  [string] $OperatingSystemArchitecture
  [string] $TimeZone
  [double] $SizeOfCdrive
  [double] $CdriveFreeSpace

  ## default constructor
  SystemInfo(){}

  ## constructor with arguments
  SystemInfo ([string]$ComputerName, [string] $Domain ){

     if (($ComputerName -eq '') -or ($ComputerName -eq $null)) {
       throw [InvalidOperationException]::new(
       "ComputerName is empty or null")
     }

     if (($Domain -eq '') -or ($Domain -eq $null)) {
       throw [InvalidOperationException]::new(
       "Domain is empty or null")
     }

     $this.ComputerName = $ComputerName
     $this.Domain = $Domain
  }

  [void] GetComputerInfo() {
     $compsys = Get-CimInstance -ClassName Win32_ComputerSystem

     $this.ComputerName = $compsys.Name
     $this.TotalPhysicalMemory = [math]::Ceiling($compsys.TotalPhysicalMemory / 1GB)

     $this.Domain = $compsys.Domain
     $this.NumberOfProcessors = $compsys.NumberOfProcessors
  }

  [void] GetCoresInfo() {
     $proc = Get-CimInstance -ClassName Win32_processor
     $this.NumberOfCores = $proc.NumberOfCores
  }

  [void] GetBIOSInfo() {
     $bios = Get-CimInstance -ClassName Win32_Bios

     $this.BIOSmanufacturer = $bios.Manufacturer
     $this.BIOSversion = $bios.Version
  }

  [void] GetOSInfo() {
     $os = Get-CimInstance -ClassName Win32_OperatingSystem

     $this.OperatingSystemName = $os.Caption
     $this.OperatingSystemArchitecture = $os.OSArchitecture
  }

  [void] GetTimeZoneInfo() {
     $this.TimeZone = (Get-TimeZone).DisplayName
  }

  [void] GetDiskInfo() {
     $disk = Get-CimInstance -ClassName Win32_LogicalDisk -Filter "DeviceID='C:'"

     $this.SizeOfCdrive = [math]::Round(($disk.Size / 1GB), 2)
     $this.CdriveFreeSpace = [math]::Round(($disk.FreeSpace / 1GB), 2)
  }

  [void] GetAllInfo() {
     $this.GetComputerInfo()
     $this.GetCoresInfo()
     $this.GetBIOSInfo()
     $this.GetOSInfo()
     $this.GetTimeZoneInfo()
     $this.GetDiskInfo()
  }

}

$sysinfo = [SystemInfo]::new()

$sysinfo.GetComputerInfo()
$sysinfo.GetCoresInfo()
$sysinfo.GetBIOSInfo()
$sysinfo.GetOSInfo()
$sysinfo.GetTimeZoneInfo()
$sysinfo.GetDiskInfo()
```

```
$sysinfo
```

The classes all start like this:

```
[void] GetTimeZoneInfo()
```

This is return type – [void] in this case as I'm not returning anything and the method name. Any arguments would go in the (). The code to get the property information is the same as you saw earlier. Once you've created an instance of the class you can call the methods and populate the properties.

```
ComputerName                : W510W10
BIOSmanufacturer            : LENOVO
BIOSversion                 : LENOVO - 1240
Domain                      : WORKGROUP
NumberOfProcessors          : 1
NumberOfCores               : 4
TotalPhysicalMemory         : 16
OperatingSystemName         : Microsoft Windows 10 Enterprise
OperatingSystemArchitecture : 64-bit
TimeZone                    : (UTC+00:00) Dublin, Edinburgh, Lisbon, London
SizeOfCdrive                : 476.1
CdriveFreeSpace             : 107.29
```

Using the single method to get all data is much simpler:

```
PS> $sysinfo2 = [SystemInfo]::new()

PS> $sysinfo2.GetAllInfo()

PS> $sysinfo2


ComputerName                : W510W10
BIOSmanufacturer            : LENOVO
BIOSversion                 : LENOVO - 1240
Domain                      : WORKGROUP
NumberOfProcessors          : 1
NumberOfCores               : 4
TotalPhysicalMemory         : 16
OperatingSystemName         : Microsoft Windows 10 Enterprise
OperatingSystemArchitecture : 64-bit
TimeZone                    : (UTC+00:00) Dublin, Edinburgh, Lisbon, London
SizeOfCdrive                : 476.1
CdriveFreeSpace             : 107.29
```

You were also asked to create a method that would calculate the percentage of free disk space.

```
class SystemInfo {

  ## class properties
  [string] $ComputerName
  [string] $BIOSmanufacturer
  [string] $BIOSversion
  [string] $Domain
  [int] $NumberOfProcessors
  [int] $NumberOfCores
  [double] $TotalPhysicalMemory
  [string] $OperatingSystemName
  [string] $OperatingSystemArchitecture
  [string] $TimeZone
  [double] $SizeOfCdrive
  [double] $CdriveFreeSpace

  ## default constructor
  SystemInfo(){}

  ## constructor with arguments
  SystemInfo ([string]$ComputerName, [string] $Domain ){

     if (($ComputerName -eq '') -or ($ComputerName -eq $null)) {
       throw [InvalidOperationException]::new(
       "ComputerName is empty or null")
     }

     if (($Domain -eq '') -or ($Domain -eq $null)) {
       throw [InvalidOperationException]::new(
       "Domain is empty or null")
     }
```

```
        $this.ComputerName = $ComputerName
        $this.Domain = $Domain
    }

    [void] GetComputerInfo() {
        $compsys = Get-CimInstance -ClassName Win32_ComputerSystem

        $this.ComputerName = $compsys.Name
        $this.TotalPhysicalMemory = [math]::Ceiling($compsys.TotalPhysicalMemory / 1GB)

        $this.Domain = $compsys.Domain
        $this.NumberOfProcessors = $compsys.NumberOfProcessors
    }

    [void] GetCoresInfo() {
        $proc = Get-CimInstance -ClassName Win32_processor
        $this.NumberOfCores = $proc.NumberOfCores
    }

    [void] GetBIOSInfo() {
        $bios = Get-CimInstance -ClassName Win32_Bios

        $this.BIOSmanufacturer = $bios.Manufacturer
        $this.BIOSversion = $bios.Version
    }

    [void] GetOSInfo() {
        $os = Get-CimInstance -ClassName Win32_OperatingSystem

        $this.OperatingSystemName = $os.Caption
        $this.OperatingSystemArchitecture = $os.OSArchitecture
    }

    [void] GetTimeZoneInfo() {
        $this.TimeZone = (Get-TimeZone).DisplayName
    }

    [void] GetDiskInfo() {
        $disk = Get-CimInstance -ClassName Win32_LogicalDisk -Filter "DeviceID='C:'"

        $this.SizeOfCdrive = [math]::Round(($disk.Size / 1GB), 2)
        $this.CdriveFreeSpace = [math]::Round(($disk.FreeSpace / 1GB), 2)
    }

    [void] GetAllInfo() {
        $this.GetComputerInfo()
        $this.GetCoresInfo()
        $this.GetBIOSInfo()
        $this.GetOSInfo()
        $this.GetTimeZoneInfo()
        $this.GetDiskInfo()
    }

    [double] CalcFreeSpacePerc() {
        if (($this.SizeOfCdrive -eq 0) -or ($this.CdriveFreeSpace -eq 0)) {
            $this.GetDiskInfo()
        }

        $perc = ($this.CdriveFreeSpace / $this.SizeOfCdrive) * 100

        return [math]::Round($perc, 3)
    }

}

$sysinfo = [SystemInfo]::new()
$sysinfo.GetAllInfo()

$sysinfo

$sysinfo.CalcFreeSpacePerc()
```

The method returns a double – you do want accuracy. If the properties haven't been populated the GetDiskInfo() method is called. This prevents those nasty divide by zero errors. Calculate the percentage free space. I've rounded the result on return. You HAVE to use return to return the results of the method back to the pipeline. Its one of the major differences between a class method and a function.

```
ComputerName                : W510W10
BIOSmanufacturer            : LENOVO
BIOSversion                 : LENOVO - 1240
Domain                      : WORKGROUP
```

```
NumberOfProcessors          : 1
NumberOfCores               : 4
TotalPhysicalMemory         : 16
OperatingSystemName         : Microsoft Windows 10 Enterprise
OperatingSystemArchitecture : 64-bit
TimeZone                    : (UTC+00:00) Dublin, Edinburgh, Lisbon, London
SizeOfCdrive                : 476.1
CdriveFreeSpace             : 107.29

22.535
```

The last question was can you create the class such that all properties are automatically populated when its instantiated. You could change the default constructor so it called GetAllInfo()

```
  ## default constructor
SystemInfo(){
  $this.GetAllInfo()
}
```

But I prefer to leave the default constructor as that – it creates an empty object – and create another constructor:

```
  SystemInfo([string] $ComputerName){
    if (($ComputerName -eq '') -or ($ComputerName -eq $null)) {
      throw [InvalidOperationException]::new(
      "ComputerName is empty or null")
    }

    if ($ComputerName -ne $env:COMPUTERNAME) {
      throw "$ComputerName NOT equal to local machine name "
    }

    $this.GetAllInfo()
}
```

This takes a string with the local machine name and as long as the string isn't empty or null and actually matches the local machine name calls the GetAllInfo() method.

The code also works on PowerShell v6 on Windows – Linux doesn't have the CIM classes so you'll have to get the data another way if you want to include Linux systems.

Remember classes were introduced in PowerShell v5 so you can't run this code in earlier versions of PowerShell.

What you have here is probably enough for Battle faction – maybe even a bit too much.

Daybreak faction will want to format the code so that it looks good. Maybe add it to one of the modules created in an earlier puzzle.

Flawless faction will want to add all the bells and whistles - anything that ensures the code executes flawlessly.

Enjoy! Puzzle 8 will be available around the time you're reading this.