



D02 - Formation Ruby on Rails

Classes Héritage et Exceptions

Stéphane Ballet balletstephane.pro@gmail.com
42 Staff pedago@staff.42.fr

Résumé: Dans ce jour D02, vous allez apprendre à faire des classes et aborder le coeur de la P.O.O., si vos exercices vous semblent trop verbeux c'est que votre code n'est pas assez D.R.Y. !

Table des matières

I	Préambule	2
II	Consignes	3
III	Règles spécifiques de la journée	5
IV	Exercice 00 : HTML	6
V	Exercice 01 : Raise HTML	8
VI	Exercice 02 : Rescue HTML	9
VII	Exercice 03 : Elem	11
VIII	Exercice 04 : Dejavu	13
IX	Exercice 05 : Validation	15

Chapitre I

Préambule

En plus d'être génial, Ruby est beau !

Ce guide (écrit par Bozhidar Batsov) a vu le jour en tant que convention de programmation Ruby interne à son entreprise. Au bout d'un certain temps, il lui a semblé que ce travail pourrait s'avérer intéressant pour les membres de la communauté ruby en général et que le monde n'avait pas vraiment besoin d'une convention de programmation interne de plus. Mais le monde pourrait certainement tirer profit d'un ensemble de pratiques, d'idiomes et de conseils de style pour la programmation Ruby.

Dès le lancement de ce guide, il a reçu énormément de retours des membres de l'exceptionnelle communauté Ruby à travers le monde. Merci pour toutes les suggestions et votre soutien ! Ensemble, nous pouvons créer une ressource bénéfique à tous les développeurs Ruby ici présents.

- [Le style est ce qui sépare le bon de l'excellent.](#)
- Le très largement utilisé [Rubocop](#)

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes :
 - Ruby `>= 2.3.0`
 - pour d09 Rails `> 5`
 - mais pour tous les autres jours Rails `4.2.7`
 - HTML `5`
 - CSS `3`
- Nous vous interdisons FORMELLEMENT d'utiliser les mots clés `while`, `for`, `redo`, `break`, `retry` et `until` dans les codes sources Ruby que vous rendrez. Toute utilisation de ces mots clés est considérée comme triche (et/ou impropre), vous donnant la note de -42.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas, nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépôt GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google` / `man` / `Internet` /
- Pensez à discuter sur le forum Piscine de votre Intra, ou Slack, ou IRC...
- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne

sont pas autrement précisées dans le sujet...

- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapitre III

Règles spécifiques de la journée

- Tous les fichiers rendus seront dotés d'un shebang approprié ET du flag de warning.
- Aucun code dans le scope global. Faites des fonctions ou des classes !
- Chaque fichier rendu doit comporter une série de tests qui prouvent son bon fonctionnement :

```
$> cat <FICHIER>.rb
[...]
if $PROGRAM_NAME == __FILE__
  ## votre code ici
end
```


- Chaque fichier doit être testé dans une console interactive (irb ou pry au choix) comme ceci :

```
require_relative "nom_de_fichier.rb"
>
```

- Aucun import autorisé, à l'exception de ceux explicitement mentionnés dans la section "Fonctions Autorisées" du cartouche de chaque exercice.

Chapitre IV

Exercice 00 : HTML

	Exercice : 00
Exercice 00 : HTML	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : ex00.rb	
Fonctions Autorisées : n/a	
Remarques : n/a	

Faites une classe `Html` capable de créer et de remplir un fichier HTML. Pour ce faire, vous devez implémenter :

- Un constructeur qui prend en paramètre un titre de fichier (sans extension) qui :
 - appelle une méthode `head`
 - attribue le titre de fichier à la variable d'instance `@page_name`
- Un `attr_reader` pour `@page_name`
- Une méthode `Head` doit apposer en tête de fichier un header `html` valide suivi d'une balise ouvrante `body`.
- Une méthode "dump" qui prend en paramètre une string et qui met à la suite de la balise `<body>` notre string entourée de balises `<p>`.
- Une méthode "finish" qui termine le fichier par `</body>`



Toutes les insertions doivent être sous forme de lignes.

Dans une console **ruby**, on doit pouvoir obtenir ceci :

```
> require_relative "ex00.rb"
=> true
> a = Html.new("test")
=> #< Html:0x00000001d71580 @page_name="test" >
> 10.times{|x| a.dump("titi_number#{x}")}
=> 12
> a.finish
=> 7
```


Et dans notre shell :

```
$> cat -e test.html
<!DOCTYPE html>$
<html>$
<head>$
<title>test</title>$
</head>$
<body>$
  <p>titi_number0</p>$
  <p>titi_number1</p>$
  <p>titi_number2</p>$
  <p>titi_number3</p>$
  <p>titi_number4</p>$
  <p>titi_number5</p>$
  <p>titi_number6</p>$
  <p>titi_number7</p>$
  <p>titi_number8</p>$
  <p>titi_number9</p>$
</body>$
```

Utiliser comme dans l'exemple une boucle pour remplir son fichier **DOIT** fonctionner. Les valeurs de retour dans irb ou pry peuvent varier.

Chapitre V

Exercice 01 : Raise HTML

	Exercice : 01
Exercice 01 : Raise HTML	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : ex01.rb	
Fonctions Autorisées : n/a	
Remarques : n/a	


Pour cet exercice, vous allez réutiliser le code de l'ex00 et y incorporer une gestion d'erreur, lever des exceptions lorsque le comportement le requiert, évitant ainsi la production de pages html non valides et/ou farfelues. Toutes les exceptions suivantes doivent être reproduites exactement, en remplaçant `<filename>` par le nom de fichier qui a créé l'erreur :

- Créer un fichier qui existe déjà (nom identique) doit lever une exception : "A file named `<filename>` already exist!".
- Ecrire du texte avec `dump` dans un fichier ne comportant pas de balise `body` ouvrante doit lever une exception : "There is no body tag in `<filename>`".
- Ecrire du texte avec `dump` après une balise `body` fermante doit lever l'exception : "Body has already been closed in `<filename>`".
- Fermer le `body` avec `finish` alors que la balise de fermeture de `body` est déjà présente doit lever l'exception : "`<filename>` has already been closed."

```
> require_relative "ex01.rb"
=> true
> a = Html.new("test")
=> #<Html:0x0000000332b9c0 @page_name='test'>
> a = Html.new("test")
RuntimeError: test.html already exist!
from /ex01.rb:15:in "head"
> a.dump("Lorem_ipsum")
=> nil
> a.finish
=> nil
> a.finish
RuntimeError: test.html has already been closed
from/ex01.rb:39:in "finish"
```

Chapitre VI

Exercice 02 : Rescue HTML

	Exercice : 02
Exercice 02 : Rescue HTML	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : ex02.rb	
Fonctions Autorisées : n/a	
Remarques : n/a	

Maintenant que vous avez les comportements problématiques dans les mains, on va en profiter.

Cet exercice vous demandera de sauver l'exécution des processus en corrigeant les tirs avec un affichage spécifique aux erreurs, et des solutions pour garantir le fonctionnement.

Reprenez le code que vous avez créé lors de l'exercice précédent et créez deux nouvelles classes : **Dup_file** et **Body_closed**. Elles devront hériter de la classe **StandardError**.

Ces deux classes seront vos nouvelles Exceptions à implémenter comme il se doit dans votre classe HTML et devront contenir les méthodes "**show_state**", "**correct**" et "**explain**". Ces méthodes auront chacune un rôle bien défini :

- **show_state** affichera l'état avant correction
- **correct** corrigera l'erreur ayant provoqué la **raise**
- **explain** affiche l'état après correction

Bien sur, chacune de ces méthodes aura son propre comportement, que vous devrez implémenter. Ainsi, lorsque vous tentez de créer un fichier qui existe déjà, votre code devra lever l'exception `Dup_file` qui doit :

- afficher la liste des fichiers similaires (avec le PWD en entier)
- créer un nouveau fichier en ajoutant '.new' avant l'extension, plusieurs fois s'il le faut : test.html , test.new.html , test.new.new.html etc etc...

Exemple :

```
A file named <filename> was already there: /home/desktop/folder_2/<filename>.html
Appended .new in order to create requested file: /home/desktop/folder_2/<filename>.new.html
```

Aussi, si vous tentez d'écrire APRES un `</body>`, votre code devra lever l'exception `Body_closed` qui doit :


- afficher la ligne et son numéro dans le fichier
- supprimer ladite balise, insérer le texte et remettre une balise fermante après

Exemple :

```
In <filename> body was closed :
> ln :25 </body> : text has been inserted and tag moved at the end of it.
```

Chapitre VII

Exercice 03 : Elem

	Exercice : 03
Exercice 03 : Elem	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : ex03.rb	
Fonctions Autorisées : n/a	
Remarques : n/a	

Il est maintenant temps de changer de méthode. Votre premier essai de moteur de fichier HTML est concluant et prometteur mais il est question maintenant de pousser le paradigme de l'objet un peu plus loin.

Vous allez maintenant créer une classe pour représenter votre HTML de sorte qu'une méthode `to_s` sur ses instances affiche le code HTML généré.

Ainsi, avec sa méthode `add_content`, la classe `Elem` va être capable d'avoir pour contenu une autre instance d'`Elem`.

Cette architecture permettra une utilisation de ce genre :

```
html = Elem.new(.....)
head = Elem.new(.....)
body = Elem.new(...)
title = Elem.new(Text.new("blah blah"))
head.add_content(title)
html.add_content([head, title, body])
puts html
```


Pour garantir une bonne implémentation, nous fournissons un fichier de tests dans le dossier `ex02/` dans la tarball `d02.tar.gz` présente avec ce sujet.

On récapitule :

- Une classe **Elem** avec comme paramètre de construction, un type de balise, un Array de contenus, un type de tag (orphelin ou non), et un Hash nous permettant d'implémenter des infos 'in-tag' (src, style, data...).
- Une classe **Text** qui se construit avec une simple **String** en paramètre.
- Une **surcharge** de la méthode **to_s**.
- L'exécution du script de test doit passer **IN-TE-GRA-LEMENT**.

Chapitre VIII

Exercice 04 : Dejavu

	Exercice : 04
Exercice 04 : Dejavu	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : ex04.rb	
Fonctions Autorisées : n/a	
Remarques : n/a	

Félicitations ! Vous êtes maintenant capable de générer n'importe quel élément **HTML** et son contenu. C'est cependant un peu lourd de générer chaque élément en précisant à chaque fois chaque attribut à chaque instantiation. C'est l'occasion d'utiliser l'héritage pour faire d'autres petites classes plus simple d'utilisation.

Réalisez les classes suivantes en les dérivant d'**Elem** :

- **Html**, **Head**, **Body**
- **Title**
- **Meta**
- **Img**
- **Table**
- **Th**, **Tr**, **Td**
- **Ul**, **Ol**, **Li**
- **H1**, **H2**
- **P**
- **Div**
- **Span**
- **Hr**
- **Br**

Votre code doit exécuter ces commandes sans erreurs :

```
> puts Html.new([Head.new([Title.new("Hello ground!"))],  
> Body.new([H1.new("Oh no, not again!"),  
> Img.new([], {'src':'http://i.imgur.com/pfp3T.jpg'}) ] ) ])
```

Et afficher :


```
<Html>  
<Head>  
<Title>Hello ground!</Title>  
</Head>  
<Body>  
<H1>Oh no, not again!</H1>  
<Img src='http://i.imgur.com/pfp3T.jpg' />  
</Body>  
</Html>
```



Si vous avez l'impression que l'exercice est rébarbatif, c'est qu'il y a peut-être une autre solution.

Chapitre IX

Exercice 05 : Validation

	Exercice : 05
Exercice 05 : Validation	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : whereto.rb	
Fonctions Autorisées : n/a	
Remarques : n/a	

Malgré de réels progrès dans votre travail, vous aimeriez bien que tout soit un peu plus propre, un peu plus cadré, vous êtes comme ça : vous aimez les contraintes et les défis. Alors pourquoi ne pas imposer une norme à la structure de vos documents **HTML** ? Commencez par copier les classes des deux exercices précédents dans le dossier de cet exercice.

Créez une classe **Page** dont le constructeur doit prendre en paramètre une instance d'une classe héritant de **Elem**. Votre classe **Page** doit implémenter une méthode **isvalid()** qui doit renvoyer **True** si toutes les règles suivantes sont respectées, et **False** sinon :

- Si pendant le parcours de l'arbre, un nœud n'est pas de type **html**, **head**, **body**, **title**, **meta**, **img**, **table**, **th**, **tr**, **td**, **ul**, **ol**, **li**, **h1**, **h2**, **p**, **div**, **span**, **hr**, **br** ou **Text**, l'arbre est invalide.
- **Html** doit contenir exactement un **Head**, **puis** un **Body**.
- **Head** ne doit contenir qu'un unique **Title** et uniquement ce **Title**.
- **Body** et **Div** ne doivent contenir que des éléments des types suivants : **H1**, **H2**, **Div**, **Table**, **Ul**, **Ol**, **Span**, ou **Text**.
- **Title**, **H1**, **H2**, **Li**, **Th**, **Td** ne doivent contenir qu'un unique **Text** et uniquement ce **Text**.
- **P** ne doit contenir que des **Text**.
- **Span** ne doit contenir que des **Text** ou des **P**.
- **Ul** et **Ol** doivent contenir au moins un **Li** et uniquement des **Li**.

- **Tr** doit contenir au moins un **Th** ou **Td** et uniquement des **Th** ou des **Td**. Les **Th** et les **Td** doivent être mutuellement exclusifs.
- **Table** ne doit contenir que des **Tr** et uniquement des **Tr**.
- **Img** : doit contenir un champ **src** et sa valeur est un **Text**.

Démontrez le fonctionnement de votre classe **Page** par des tests de votre choix en nombre suffisant pour couvrir toutes les fonctionnalités. Par exemple, l'exécution de :

```
if $PROGRAM_NAME == __FILE__
  toto = Html.new([Head.new([Title.new(Text.new("Hello ground!"))]),
    Body.new([H1.new(Text.new("Oh no, not again!")), Img.new([],
      {'src': Text.new('http://i.imgur.com/pfp3T.jpg')} )] )])
  test = Page.new(toto)
  test.is_valid?
  tata = Html.new([Head.new([Title.new(Text.new("Hello ground!"))]),
    Body.new([H1.new(Text.new("Oh no, not again!")), Img.new([],
      {'src': Text.new('http://i.imgur.com/pfp3T.jpg')} )] )])
  test2 = Page.new(tata)
  test2.is_valid?
end
```

DOIT afficher :

```
Currently evaluating a Html :
- root element of type "html"
- Html -> Must contains a Head AND a Body after it
Head is OK
Evaluating a multiple node
Currently evaluating a Text :
-Text -> Must contains a simple string
Text content is OK
Evaluating a multiple node
Currently evaluating a Text :
-Text -> Must contains a simple string
Text content is OK
Currently evaluating a Img :
Img content is OK
      FILE IS OK
```