Un exemple de développement d'un *gem* : `minised`, une version simplifiée de `sed`

Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay_gu

INF600A
20 novembre 2018

1. Le *gem* `minised`
2. Création du squelette de `minised` avec `gli`
3. Spécification des tests d'acceptation
4. Architecture de la solution
5. Mise en œuvre des commandes `substitute` et `delete`
6. Les étapes de l'exécution d'une commande
7. Un peu de *Refactoring* dans `bin/minised`
8. Création d'un *gem* pour distribution
9. Conclusion sur l'exemple `minised`

1. Le *gem* `minised`

L'exemple de *gem* qui suit est une version simplifiée de `sed`, avec des commandes et options «à la `git`»

Ce que présentent les diapos

- Les principales étapes pour la création d'une application avec une interface personne–machine (IPM) en ligne de commandes avec `gli`.
- Le logiciel (presque) final — et non pas toutes les étapes ayant mené à sa forme finale.

L'exemple de *gem* qui suit est une version simplifiée de `sed`, avec des commandes et options «à la `git`»

Ce que présentent les diapos

- Les principales étapes pour la création d'une application avec une interface personne–machine (IPM) en ligne de commandes avec `gli`.
- Le logiciel (presque) final — et non pas toutes les étapes ayant mené à sa forme finale.

Grandes lignes de la mise en œuvre (architecture)

- L'IPM (ligne de commandes) est clairement séparée de la logique métier (traitement de flux de lignes de texte).
- Dans l'IPM, l'analyse et la répartition (*dispatch*) des commandes et options se fait en utilisant le *gem* `gli`.

Voici à quoi ressemblera l'IPM de cette version simplifiée de `sed` «à la `git`»

Le commentaire indique l'équivalent en «vrai» `sed`

```
# sed '/m1/d' foo.txt
$ minised delete m1 foo.txt
...

# sed 's/m1/c1/' f1.txt f2.txt
$ minised substitute m1 c1 f1.txt f2.txt
...

# sed 's/m1/c1/g' <f.txt
$ minised substitute -g m1 c1 <f.txt
...

# sed --in-place=.bak 's/m1/c1/g' f.txt
$ minised --in-place=.bak substitute -g m1 c1 f.txt
```

2. Création du squelette de `minised` avec `gli`

On doit tout d'abord installer le *gem* `gli`


```
$ gem install gli
Fetching: gli-2.13.2.gem (100%)
Successfully installed gli-2.13.2
1 gem installed
```

Remarque importante : Si vous travaillez sur le serveur `java.labunix.uqam.ca`, vous n'avez pas besoin d'installer `gli` car il est déjà installé et disponible.

On crée le squelette de l'application `minised`

Les arguments après le nom de l'application (`minised`) sont les commandes de l'application : `substitute`, `delete` et `print`

```
$ gli scaffold minised substitute delete print
Creating dir ./minised/lib...
Creating dir ./minised/bin...
Creating dir ./minised/test...
Created ./minised/bin/minised
Created ./minised/README.rdoc
Created ./minised/minised.rdoc
Created ./minised/minised.gemspec
Created ./minised/test/default_test.rb
Created ./minised/test/test_helper.rb
Created ./minised/Rakefile
Created ./minised/Gemfile
Created ./minised/features
Created ./minised/lib/minised/version.rb
Created ./minised/lib/minised.rb
```

Note : `gli scaffold` = `gli init`

Structure des répertoires pour le squelette d'application créé par `gli`


```
$ tree minised
minised
|-- bin
|   `-- minised
|-- features
|   |-- minised.feature
|
|-- step_definitions
|   |
`-- minised_steps.rb
|   `-- support
|       `-- env.rb
|-- Gemfile
|-- lib
|   |-- minised
|   |   `-- version.rb
|   `-- minised.rb
|-- minised.gemspec
|-- minised.rdoc
|-- Rakefile
|-- README.rdoc
`-- test
    |-- default_test.rb
    `-- test_helper.rb

7 directories, 13 files
```

Contenu du programme principal : Importation du *gem* `gli` et inclusion de `GLI::App`


```
$ cat minised/bin/minised
#!/usr/bin/env ruby
require 'gli'
begin # XXX: Remove this begin/rescue before distributin
require 'minised'
rescue LoadError
  STDERR.puts "In development, you need to use `bundle e
  STDERR.puts "At install-time, RubyGems will make sure
  STDERR.puts "Feel free to remove this message from bin
  exit 64
end

include GLI::App

program_desc 'Describe your application here'

version MiniSed::VERSION
```

Contenu du programme principal (suite) : Spécification des options globales

Options par défaut, pour illustrer les différentes formes

```
# Use argument validation
arguments :strict

desc 'Describe some switch here'
switch [:s,:switch]

desc 'Describe some flag here'
default_value 'the default'
arg_name 'The name of the argument'
flag [:f,:flagname]
```

Contenu du programme principal (suite) : Squelette de la commande `substitute`

Commandes partiellement définies, illustrant différents éléments à compléter

```
desc 'Describe substitute here'
arg_name 'Describe arguments to substitute here'
command :substitute do |c|
  c.desc 'Describe a switch to substitute'
  c.switch :s

  c.desc 'Describe a flag to substitute'
  c.default_value 'default'
  c.flag :f
  c.action do |global_options,options,args|
    # Your command logic here
    # If you have any errors, just raise them
    # raise "that command made no sense"

    puts "substitute command ran"
  end
end
```

Contenu du programme principal (suite) : Squelettes des commandes `delete` et `print`

Commandes partiellement définies, illustrant différents éléments à compléter

```
desc 'Describe delete here'
arg_name 'Describe arguments to delete here'
command :delete do |c|
  c.action do |global_options,options,args|
    puts "delete command ran"
  end
end

desc 'Describe print here'
arg_name 'Describe arguments to print here'
command :print do |c|
  c.action do |global_options,options,args|
    puts "print command ran"
  end
end
```

Contenu du programme principal (suite) : Traitement des erreurs et lancement de l'exécution

On rédefinira ultérieurement ce traitement par défaut des erreurs

```
...

on_error do |exception|
  # Error logic here
  # return false to skip default error handling
  true
end

exit run(ARGV)
```

On met les fichiers sous contrôle de `git`


```
$ cd minised

$ git init .
Initialized empty Git repository in /home/tremblay_gu/minised/

$ git add .

$ git commit -am "Creation initiale du projet"
[master (root-commit) 81c9f08] Creation initiale
 13 files changed, 225 insertions(+)
 create mode 100644 Gemfile
 create mode 100644 README.rdoc
 create mode 100644 Rakefile
 create mode 100755 bin/minised
 ...
 create mode 100644 test/default_test.rb
 create mode 100644 test/test_helper.rb
```

On installe les *gems* requis avec `bundle`

Sur `java.labunix.uqam.ca` ou sur votre machine personnelle

Sur `java.labunix.uqam.ca` ou votre machine personnelle

```
$ emacs minised.gemspec

#  On supprime la ligne <<s.has_rdoc = true>>
...

$ bundle install --path vendor/bundle
Fetching gem metadata from https://rubygems.org/.........
Resolving dependencies...
Using rake 11.1.1
Using ffi 1.9.10
...
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is inst
```

Un exemple d'exécution directe du programme squelette

Les *gems* installés pour ce projet ne sont pas ceux disponibles par défaut ☹

```
$ bin/minised
In development, you need to use `bundle exec bin/minised`\
    to run your app
At install-time, RubyGems will make sure lib, etc. are in the
Feel free to remove this message from bin/minised now
```

Un exemple d'exécution indirecte du programme squelette : Aucun argument ⇒ `help`

L'appel avec `bundle exec` assure que les *gems* utilisés sont bien ceux du projet

```
$ bundle exec bin/minised

NAME
    minised - Describe your application here
SYNOPSIS
    minised [global options] command [command options] [argume
VERSION
    0.0.1
GLOBAL OPTIONS
    -f, --flagname=The name of the argument - Describe some fl
    --help                                  - Show this messag
    -s, --[no-]switch                       - Describe some sw
    --version                               - Display the prog
COMMANDS
    delete     - Describe delete here
    help       - Shows a list of commands or help for one comm
    print      - Describe print here
    substitute - Describe substitute here
```

Un autre exemple d'exécution : la commande `help substitute`

Affiche un squelette d'aide... évidemment à compléter

```
$ bundle exec bin/minised help substitute
NAME
    substitute - Describe substitute here

SYNOPSIS
    minised [global options] substitute
            [command options] Describe arguments to sub

COMMAND OPTIONS
    -f arg - Describe a flag to substitute (default: def
    -s     - Describe a switch to substitute
```

Un autre exemple d'exécution : la commande `substitute`

Exécution de la commande bidon, qui affiche simplement une trace

```
$ bundle exec bin/minised substitute
substitute command ran
```

Un autre exemple d'exécution : la commande `substitute`

Exécution de la commande bidon, qui affiche simplement une trace

```
$ bundle exec bin/minised substitute
substitute command ran
```

```
desc 'Describe substitute here'
arg_name 'Describe arguments to substitute here'
command :substitute do |c|
  c.desc 'Describe a switch to substitute'
  c.switch :s

  c.desc 'Describe a flag to substitute'
  c.default_value 'default'
  c.flag :f
  c.action do |global_options,options,args|
    puts "substitute command ran"
  end
end
```

Jusqu'à présent, tous les fichiers présentés sont ceux générés par `gli scaffold`

- L'opération `gli scaffold` crée donc un squelette, relativement détaillé mais «bidon», de l'application et des fonctions requises pour mettre en œuvre les diverses commandes, avec des exemples d'options (*switches* et *flags*).

- Les prochaines étapes = compléter/raffiner les squelettes des diverses commandes de l'application...

Jusqu'à présent, tous les fichiers présentés sont ceux générés par `gli scaffold`

- L'opération `gli scaffold` crée donc un squelette, relativement détaillé mais «bidon», de l'application et des fonctions requises pour mettre en œuvre les diverses commandes, avec des exemples d'options (*switches* et *flags*).

- Les prochaines étapes = compléter/raffiner les squelettes des diverses commandes de l'application... possiblement en introduisant des modules, classes et/ou méthodes auxiliaires.

Jusqu'à présent, tous les fichiers présentés sont ceux générés par `gli scaffold`

- L'opération `gli scaffold` crée donc un squelette, relativement détaillé mais «bidon», de l'application et des fonctions requises pour mettre en œuvre les diverses commandes, avec des exemples d'options (*switches* et *flags*).

- Les prochaines étapes = compléter/raffiner les squelettes des diverses commandes de l'application... possiblement en introduisant des modules, classes et/ou méthodes auxiliaires.

- Mais auparavant, on regarde les tests d'acceptation

3. Spécification des tests d’acceptation

Par défaut, les tests d'acception sont configurés pour être décrits avec `cucumber` et `aruba`

On y reviendra peut-être (?), ultérieurement,. . . si le temps le permet (?)

```
$ cat minised/features/minised.feature
Feature: My bootstrapped app kinda works
  In order to get going on coding my awesome app
  I want to have aruba and cucumber setup
  So I don't have to do it myself

  Scenario: App just runs
    When I get help for "minised"
    Then the exit status should be 0

$ cat minised/features/step_definitions/minised_steps.rb
When /^I get help for "([^"]*)"$/ do |app_name|
  @app_name = app_name
  step %(I run `#app_name help`)
end

# Add more step definitions here
```

Les tests d'acceptation qui seront utilisés ont plutôt été écrits avec l'approche des devoirs #1 et #2

- On utilise le cadre de tests (unitaires) `MiniTest` pour spécifier les tests d'acceptation

- Un fichier `test_helper.rb` définit diverses méthodes auxiliaires permettant de manipuler des fichiers et lancer l'exécution de `minised` (via `bundle exec`) :
  - `avec_fichier`
  - `contenu_fichier`
  - `genere_sortie`
  - `execute_sans_sortie_ou_erreur`

  - `run_minised`

Or, dans le squelette généré par `gli`, les tests unitaires utilisent `Test::Unit` ☹

Sauf qu'on veut plutôt utiliser `MiniTest`

```
$ cat test/test_helper.rb
require 'test/unit'
# Add test libraries you want to use here, e.g. mocha

class Test::Unit::TestCase
  # Add global extensions to the test case class here
end

$ cat test/default_test.rb
require 'test_helper'

class DefaultTest < Test::Unit::TestCase
  def setup
  end
  ...
  def test_the_truth
    assert true
  end
end
```

Voici donc le contenu de ces deux fichiers après modification pour plutôt utiliser `MiniTest`


```
$ emacs test/test_helper.rb
gem 'minitest'
require 'minitest/autorun'
require 'minitest/spec'
require 'minitest/mock'

class Object
  def _it( test ) ... end
  ...
end

$ emacs test/default_test.rb
require 'test_helper'
require 'minised'

describe "A COMPLETER" do
  it "a completer" do
  end
end
```

On lance l'exécution des tests unitaires...


```
$ rake test
/home/tremblay_gu/.rvm/gems/ruby-2.2.5/gems/bundler-1.12
    minitest is not part of the bundle.
    Add it to Gemfile. (Gem::LoadError)
         from [...]
         [...]
         from [...]
rake aborted!
Command failed with status (1)
[...]
Tasks: TOP => test
(See full trace by running task with --trace)
```

On lance l'exécution des tests unitaires... Oops ! Il manque le *gem* `MiniTest` ☹


```
$ rake test
/home/tremblay_gu/.rvm/gems/ruby-2.2.5/gems/bundler-1.12
    minitest is not part of the bundle.
    Add it to Gemfile. (Gem::LoadError)
         from [...]
         [...]
         from [...]
rake aborted!
Command failed with status (1)
[...]
Tasks: TOP => test
(See full trace by running task with --trace)
```

On ajoute la ligne appropriée dans `minised.gemspec` et on réinstalle


```
$ emacs minised.gemspec
  ...
  s.add_development_dependency('minitest') # AJOUT
end

$ bundle install --path vendor/bundle # OU: bundle install
Fetching gem metadata from https://rubygems.org/.........
...
```

On ajoute la ligne appropriée dans `minised.gemspec` et on réinstalle


```
$ emacs minised.gemspec
  ...
  s.add_development_dependency('minitest') # AJOUT
end

$ bundle install --path vendor/bundle # OU: bundle install
Fetching gem metadata from https://rubygems.org/.........
...
```

Le test (bidon) s'exécute

```
$ rake test
Run options: --seed 4468

# Running:
.

Finished in 0.000656s, 1524.4576 runs/s, 0.0000 assertions/s.

1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
```

Contenu du répertoire des tests d'acceptation

État du répertoire **après** l'écriture des tests et l'ajout de divers fichiers

```
$ tree test_acceptation
test_acceptation
|-- delete_test.rb
|-- insert_test.rb
|-- substitute_test.rb
`-- test_helper.rb

0 directories, 4 files
```

```
$ grep "def " test_acceptation/test_helper.rb
def run_minised( *cmds )
  def _describe( test )
  def _it_( test, niveau =~:base )
  def it_( test, niveau =~:base, &bloc )
def avec_fichier( nom_fichier, lignes = [], conserver = nil )
def contenu_fichier( nom_fichier )
def genere_erreur( erreur )
def execute_sans_sortie_ou_erreur
def genere_sortie( attendu, strict = nil  )
def genere_sortie_et_erreur( attendu, erreur, strict = nil )
```

Un premier exemple de test d'acceptation pour la commande `substitute`


```
it "change seulement la 1ere occurrence sans l'option -g" do
  avec_fichier 'f.txt', ["0abc!", "==", "abcdef9abc"] do
    genere_sortie ["0XX!", "==", "XXdef9abc"] do
      run_minised( 'substitute "abc" "XX" <f.txt' )
    end
  end
end
```

Un deuxième exemple de test d'acceptation pour la commande `substitute`


```
it "change toutes les occurrences, modifie le fichier\
    et cree une copie" do
  entree = ["0abc!", "==", "abcdef9abc"]
  sortie = ["0XX!", "==", "XXdef9XX"]

  avec_fichier 'f.txt', entree do
    execute_sans_sortie_ou_erreur do
      run_minised( '-i .bk substitute -g "abc" "XX" f.txt' )
    end

    contenu_fichier( 'f.txt' ).must_equal sortie
    contenu_fichier( 'f.txt.bk' ).must_equal entree
  end

  FileUtils.rm_f 'f.txt.bk'
end
```

Un exemple de test d'acceptation pour la commande `delete`


```
it "supprime les lignes qui matchent et ce avec plusieurs fich
  entree = ["aaax2xaax2", "foo", "abcaaabc12"]

  avec_fichier 'foo.txt', entree do
    avec_fichier 'vide.txt', [] do
      genere_sortie  ["foo", "foo"] do
        run_minised( 'delete "aa" foo.txt vide.txt foo.txt' )
      end
    end
  end
end
```

4. Architecture de la solution

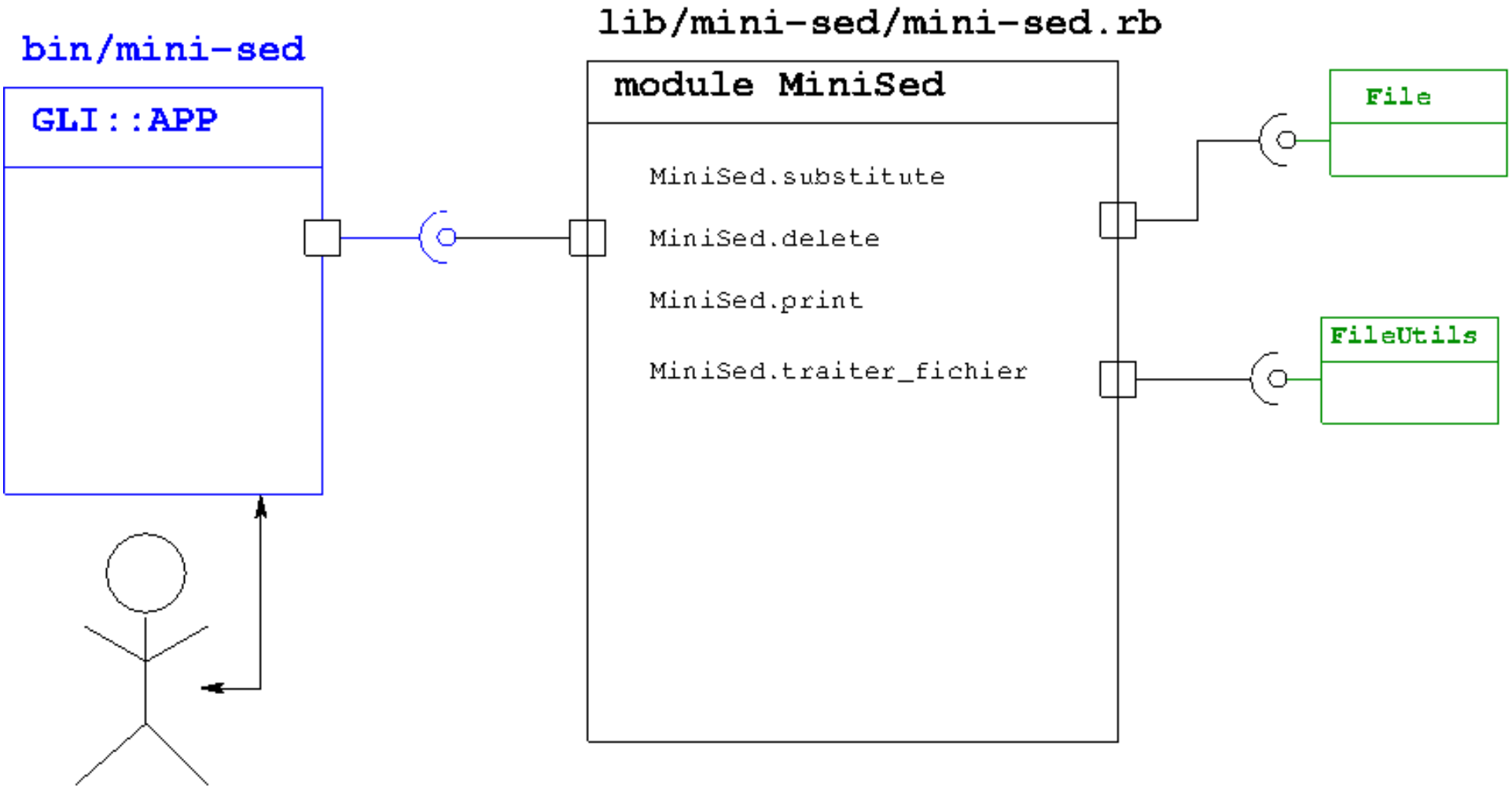
Architecture de l'application `minised`

Forme finale… ou presque

- Les diapositives qui suivent présentent la forme finale de l'architecture de `minised`

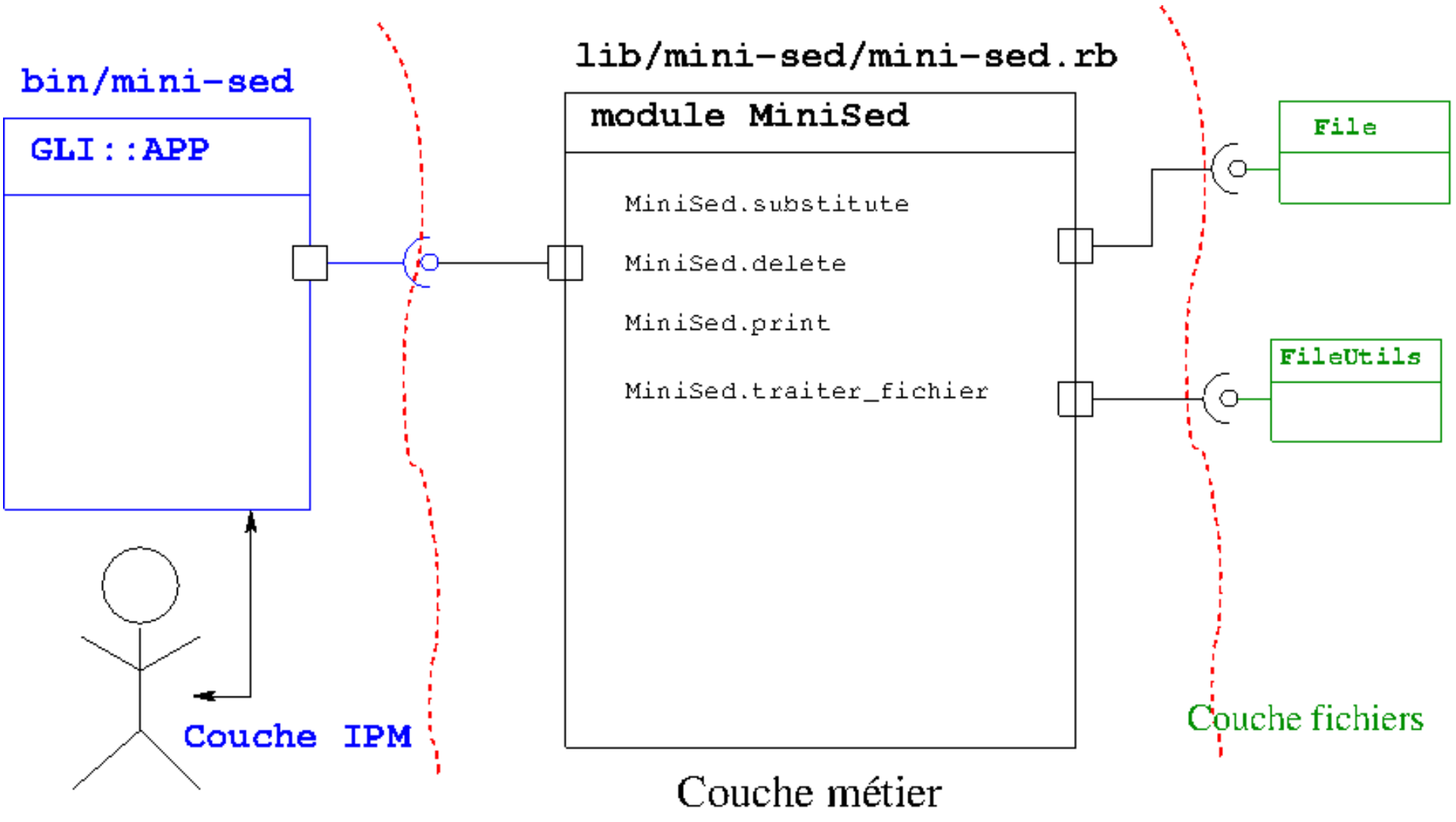
- Les étapes pour y arriver ne sont pas décrites — et elles ont été nombreuses, impliquant plusieurs *refactoring*

Architecture de l'application `minised`



Architecture de l'application `minised`

Architecture multi-couches (*multi-tiers*)



Pour arriver à cette architecture. . .

J'ai ajouté et modifié divers fichiers :

- `lib/dbc.rb` : Des méthodes pour DBC (*Design By Contract*)
- `lib/debug.rb` : Des méthodes pour aider au débogage (trace dans un fichier de journalisation)

- `lib/minised.rb` : Les divers `require` — voir plus loin.
- `lib/minised/minised.rb` : Les méthodes auxiliaires pour la couche métier, définies dans un `module MiniSed`.

On regarde maintenant la structure et l'état de ces fichiers. . .

Note : Dans ce qui suit, on suppose la définition suivante — pour réduire l'espace sur les diapositives :

```
MS = MiniSed
```

Organisation des fichiers du répertoire `lib`

Le répertoire `lib` contient tous les fichiers associés à la mise en œuvre de `minised`, sauf le programme principal (= l'application)

```
$ tree bin
bin
`-- minised

0 directories, 1 file


$ tree lib
lib
|-- dbc.rb
|-- debug.rb
|-- minised
|   |-- minised.rb
|   `-- version.rb
`-- minised.rb

1 directory, 5 files
```

Le fichier `lib/minised.rb` ne contient que des `require`


```
$ cat lib/minised.rb
# Add requires for other files you add to your project h
# you just need to require this one file in your bin fil

require 'debug'
require 'dbc'

require 'minised/version'
require 'minised/minised'
```

- Inclut (avec `require`) tous les fichiers (modules et classes) définis par l'application.
- Donc : on regroupe tous les `require`, pour éviter d'en avoir à plusieurs endroits, pour mieux identifier les fichiers/*gems* utilisés, etc.

Le fichier `lib/minised/version.rb` définit le numéro de version du *gem*


```
$ cat lib/minised/version.rb
module MiniSed
  VERSION = '0.0.1'
end
```

Le fichier `lib/minised/minised.rb` définit les méthodes auxiliaires, qui font «le vrai travail»

Donc des méthodes qui encapsulent la «logique métier»

```
$ cat lib/minised/minised.rb
module MiniSed
  def self.substitute( flux_in, flux_out, motif, chaine, global )
    ...
  end

  def self.delete( flux_in, flux_out, motif )
    ...
  end

  def self.print( flux_in, flux_out, motif )
    ...
  end

  def self.traiter_fichier( nom_fichier, in_place: nil )
    ...
  end
end
```

«le vrai travail»

= comment fait-on pour émettre sur `flux_out` les lignes de `flux_in` avec `motif` substitué par `chaine` (partout si `global`)...

Le fichier `lib/minised/minised.rb` définit les méthodes auxiliaires, qui font «le vrai travail»

Donc des méthodes qui encapsulent la «logique métier»

```
$ cat lib/minised/minised.rb
module MiniSed
  def self.substitute( flux_in, flux_out, motif, chaine, global )
    ...
  end

  def self.delete( flux_in, flux_out, motif )
    ...
  end

  def self.print( flux_in, flux_out, motif )
    ...
  end

  def self.traiter_fichier( nom_fichier, in_place: nil )
    ...
  end
end
```

«le vrai travail»

= comment fait-on pour émettre sur `flux_out` les lignes de `flux_in` avec `motif` substitué par `chaine` (partout si `global`)... peu importe la source des lignes (`stdin`, fichier), `in-place` ou non, etc.

Le fichier `lib/minised/minised.rb` définit les méthodes auxiliaires, qui font «le vrai travail»

Donc des méthodes qui encapsulent la «logique métier»

```
$ cat lib/minised/minised.rb
module MiniSed
  def self.substitute( flux_in, flux_out, motif, chaine, global )
    ...
  end

  def self.delete( flux_in, flux_out, motif )
    ...
  end

  def self.print( flux_in, flux_out, motif )
    ...
  end

  def self.traiter_fichier( nom_fichier, in_place: nil )
    ...
  end
end
```

«le vrai travail»

= comment fait-on pour émettre sur `flux_out` les lignes de `flux_in` avec `motif` substitué par `chaine` (partout si `global`)... peu importe la source des lignes (`stdin`, fichier), `in-place` ou non, etc.

⇒ permettra de tester uniquement la logique de la substitution !

La méthode `MiniSed.traiter_fichier`

Encapsule les détails liés au choix des flux — émission sur `stdout` vs. dans un fichier, création ou non d'une copie, etc.

```ruby
def self.traiter_fichier( nom_fichier, in_place: nil )
  if nom_fichier == :stdin
    flux_in = $stdin
    flux_out = $stdout
  elsif in_place.nil?
    flux_in = File.open(nom_fichier)
    flux_out = $stdout
  else
    in_place = "#{$$}" if in_place.empty?
    nom_copie = "#{nom_fichier}#{in_place}"
    FileUtils.mv nom_fichier, nom_copie, force: true
    flux_in = File.open(nom_copie);
    flux_out = File.open(nom_fichier, "w")
  end

  yield( flux_in, flux_out ) # Le <vrai> traitement des flux!

  FileUtils.rm_f nom_copie if nom_copie && in_place == "#{$$}"
  flux_out.close if flux_out != $stdout
end
```

Des tests peuvent être définis pour la méthode `MiniSed.traiter_fichier`


```
it "modifie le fichier lorsqu'on traite en place et fait\
    une copie de sauvegarde sans rien emettre sur stdout" do
  lignes = ["abc", "def"]
  attendu = ["abc", "abc", "def", "def", "FIN"]
  res_stdout = []
  avec_fichier 'f.txt', lignes, :conserver do
    $stdout.stub :<<, ->(l) { res_stdout << l } do
      MS.traiter_fichier( 'f.txt', in_place: '.bk' ) do |f_in,
        f_in.each { |x| f_out << x << x }
        f_out << "FIN"
      end
    end
  end

  res_stdout.must_be_empty
  contenu_fichier('f.txt').must_equal attendu
  contenu_fichier('f.txt' + '.bk').must_equal lignes

  FileUtils.rm_f ['f.txt','f.txt.bk']
end
```

5. Mise en œuvre des commandes `substitute` et `delete`

Le programme principal `bin/minised`

Spécification des options globales

```
$ cat bin/minised
...
program_desc 'Programme qui emule sed (de facon simplifi

version MiniSed::VERSION

# Use argument validation
arguments :strict

desc 'Execute en modifiant directement le fichier traite
arg_name 'extension'
flag [:i, :'in-place']
...
```

Note : Remarquez la façon dont est défini le symbole... pcq. «–» n'est pas un caractère valide pour un identificateur :

```
>> :"in_place"
=> :in_place
```

5.1 Commande `substitute`

Le programme principal `bin/minised`

Mise en œuvre de la commande `substitute`


```
desc "Substitue un motif par une chaine dans les lignes qui ma
arg_name 'motif chaine [fichier...]'
command :substitute do |c|
  c.desc 'Substitution globale'
  c.switch :g

  c.action do |glob_opts,options,args|
    motif, chaine, *fichiers = args
    fichiers = [:stdin] if fichiers.empty?

    fichiers.each do |fichier|
      MS.traiter_fichier( fichier, in_place: glob_opts[:i] )
                                do |flux_in, flux_out|
        MS.substitute( flux_in, flux_out,
                       motif, chaine, options[:g] )
      end
    end
  end
end
```

Exécution de la commande `help` après les modifications


```
$ bundle exec bin/minised help
NAME
    minised - Programme qui emule sed (de facon simplifiee)
SYNOPSIS
    minised [global options] command [command options] [argume
VERSION
    0.0.1
GLOBAL OPTIONS
    --help                          - Show this message
    -i, --in-place=extension - Execute en modifiant directemen
    --version                       - Display the program version
COMMANDS
    delete      - Describe delete here
    help        - Shows a list of commands or help for one comm
    print       - Describe print here
    substitute - Substitue un motif par une chaine dans les li
```

Note : On aurait le même effet avec l'option `help` :

```
bundle exec bin/minised --help
```

Exécution de la commande `substitute --help` après les modifications


```
$ bundle exec bin/minised substitute --help
NAME
    substitute - Substitue un motif par une chaine\
                 dans les lignes qui matchent

SYNOPSIS
    minised [global options] substitute [command options]\
                                    motif chaine [fichier...]

COMMAND OPTIONS
    -g - Substitution globale
```

Le fichier `lib/minised/minised.rb`, commande `substitute`

Méthode de classe de `MiniSed`, qui met en œuvre le traitement sur les flux

```
$ cat lib/minised/minised.rb
module MiniSed
  def self.substitute( flux_in, flux_out,
                       motif, chaine, global )
    motif_er = /#{motif}/

    flux_in.each do |ligne|
      if motif_er =~ ligne
        flux_out << (global ? ligne.gsub(motif_er, chaine)
                            : ligne.sub(motif_er, chaine))
      else
        flux_out << ligne
      end
    end
  end
end
```

Note : `sub` et `gsub` sont définies dans `String` et ne modifient pas `ligne` :

`https://ruby-doc.org/core-2.5.1/String.html`

La documentation en format YARD de la méthode `MiniSed.substitute`


```
# Effectue des substitutions sur les lignes du flux d'entree
# et emet les lignes resultantes sur le flux de sortie.
#
# @param [#each] flux_in le flux a traiter
# @param [#<<] flux_out le flux sur lequel emettre les resultat
# @param [String] motif la chaine qui decrit le motif a cherch
# @param [String] chaine la chaine de remplacement
# @param [Bool] global substitution globale
#                      (toute la ligne) ou non (1ere occurrenc
#
# @return [void]
#
def self.substitute( flux_in, flux_out,
                     motif, chaine, global )
  ...
end
```

La documentation en format YARD de la méthode `MiniSed.substitute` une fois générée


```
.substitute(flux_in, flux_out, motif, chaine, global) ⇒ void
```

> This method returns an undefined value.

Effectue des substitutions sur les lignes du flux d'entree et emet les lignes resultantes sur le flux de sortie.

Parameters:

- **`flux_in`** (#each) — le flux a traiter
- **`flux_out`** (#<<) — le flux sur lequel emettre les resultats
- **`motif`** (String) — la chaine qui decrit le motif a chercher
- **`chaine`** (String) — la chaine de remplacement
- **`global`** (Bool) — substitution globale (toute la ligne) ou non (1ere occurrence)

[View source]

Les attentes sur le flux d'entrée vs. le flux de sortie sont limitées, mais précises et spécifiques

- `flux_in` : On doit pouvoir obtenir les divers éléments du flux avec `each`.
- `flux_out` : On doit pouvoir ajouter un élément au flux avec `<<`.

Les attentes sur le flux d'entrée vs. le flux de sortie sont limitées, mais précises et spécifiques

- `flux_in` : On doit pouvoir obtenir les divers éléments du flux avec `each`.
- `flux_out` : On doit pouvoir ajouter un élément au flux avec `<<`.

= *Duck Typing* : Ce qui compte est uniquement les messages auxquels répond un objet, pas la classe de cet objet.

Les attentes sur le flux d'entrée vs. le flux de sortie sont limitées, mais précises et spécifiques

- `flux_in` : On doit pouvoir obtenir les divers éléments du flux avec `each`.
- `flux_out` : On doit pouvoir ajouter un élément au flux avec `<<`.

= *Duck Typing* : Ce qui compte est uniquement les messages auxquels répond un objet, pas la classe de cet objet.

Question : Comment obtiendrait-on quelque chose de semblable en Java ?

Les attentes sur le flux d'entrée vs. le flux de sortie sont limitées, mais précises et spécifiques

- `flux_in` : On doit pouvoir obtenir les divers éléments du flux avec `each`.
- `flux_out` : On doit pouvoir ajouter un élément au flux avec `<<`.

= *Duck Typing* : Ce qui compte est uniquement les messages auxquels répond un objet, pas la classe de cet objet.

Question : Comment obtiendrait-on quelque chose de semblable en Java ?

Avec des `interface` :

```
interface Eachable<T> { public T each(); }
interface Pushable<T> { public void push(T); }
```

Des tests unitaires pour `MiniSed.substitute`, qui n'accèdent à aucun fichier

Extraits du fichier `test/substitute_test.rb`

Fait : On peut itérer avec `each` sur un `Array` et on peut ajouter un élément avec `<<`

```
let(:flux_out) { [] }
let(:juste_des_a) { ["aaadef\n", "123aaa\n"] }

it "retourne [] lorsque recoit []" do
  MS.substitute( [], flux_out, nil, nil, false )

  flux_out.must_be_empty
end

it "effectue le remplacement partout lorsque global est true"
  MS.substitute( juste_des_a, flux_out, "a", "X", true )

  flux_out.must_equal ["XXXdef\n", "123XXX\n"]
end
...
```

Donc : Les tests unitaires n'ont pas besoin de manipuler des fichiers !

Mais on peut aussi définir des tests unitaires qui utilisent des flux qui sont des fichiers

Extraits du fichier `test/substitute_test.rb`

```
it "lit a partir d'un vrai fichier ouvert avec File.open" do
  avec_fichier 'f.txt', ["aaadef", "def", "123aaa"] do
    MS.substitute( File.open('f.txt'), flux_out,
                   "aaa", "X", false )

    flux_out.must_equal ["Xdef\n", "def\n", "123X\n"]
  end
end

it "ecrit dans un vrai fichier ouvert avec File.open" do
  flux_in = ["aaadef\n", "def\n", "123aaa\n"]

  File.open('f.txt', "w") do |flux_out|
    MS.substitute( flux_in, flux_out, "aaa", "X", false )
  end

  contenu_fichier('f.txt').must_equal ["Xdef", "def", "123X"]

  FileUtils.rm_f 'f.txt'
end
```

5.2 Commande `delete`

Le programme principal `bin/minised`

Mise en œuvre de `delete`

```
desc 'Supprime les lignes qui matchent un motif'
arg_name 'motif [fichier...]'
command :delete do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers = [:stdin] if fichiers.empty?

    fichiers.each do |fichier|
      MS.traiter_fichier( fichier, in_place: global_options[:i
                          do |flux_in, flux_out|
        MS.delete( flux_in, flux_out, motif )
      end
    end
  end
end
```

Le fichier `lib/minised/minised.rb`, commande `delete`

Méthode de classe de `MiniSed`, qui met en œuvre le traitement sur les flux

```
$ cat lib/minised/minised.rb
module MiniSed
  def self.delete( flux_in, flux_out, motif )
    flux_in.each do |ligne|
      flux_out << ligne unless /#{motif}/ =~ ligne
    end
  end
end
```

La documentation en format YARD de la méthode `MiniSed.delete`


```
# Effectue des suppressions de lignes du flux d'entree et emet
# lignes non supprimees sur le flux de sortie.
#
# @param [#each] flux_in le flux a traiter
# @param [#<<] flux_out le flux sur lequel emettre les resultat
# @param [String] motif la chaine qui decrit le motif a cherch
#
# @return [void]
#
def self.delete( flux_in, flux_out, motif )
  ...
end
```

6. Les étapes de l’exécution d’une commande

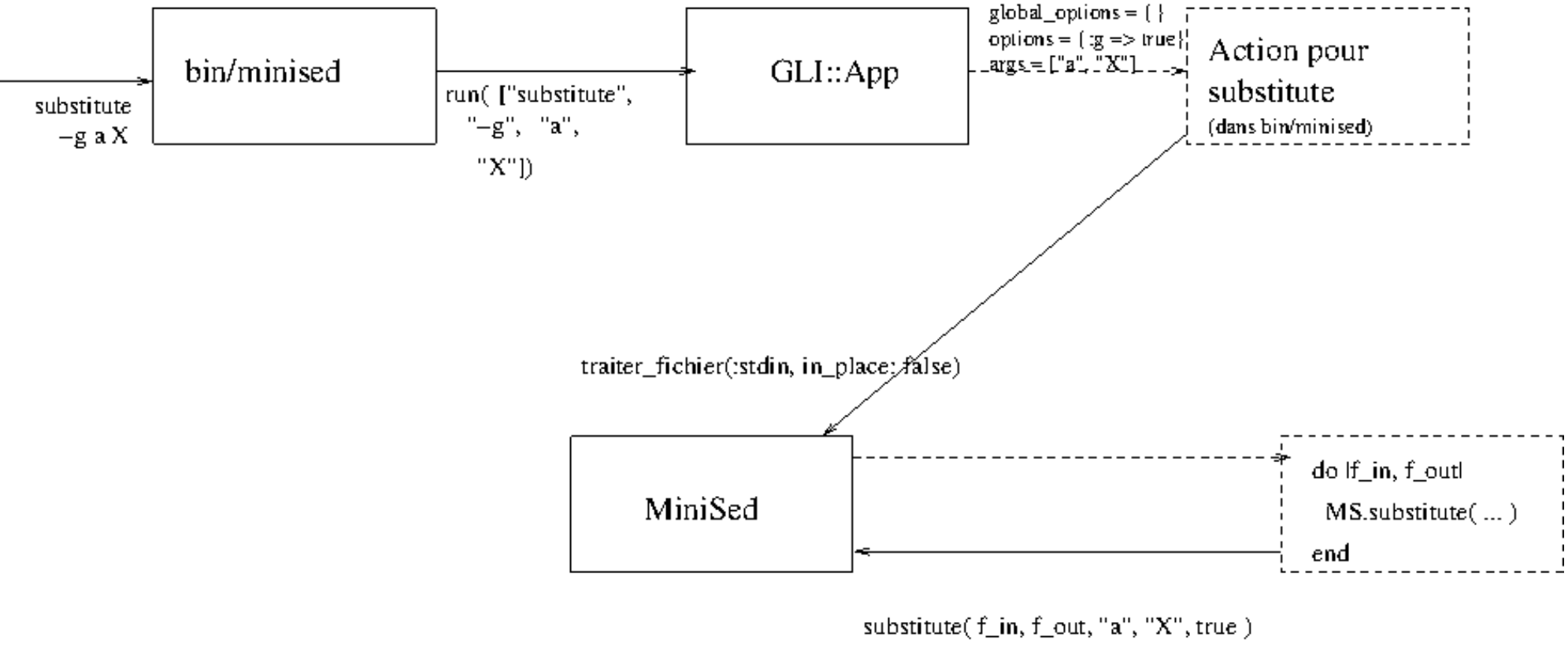
Les étapes de l'exécution d'une commande

- Les diapositives qui suivent illustrent les différentes étapes de l'exécution d'une commande avec `bundle` pour `minised`

- La commande exécutée dans cet exemple est :

```
$ bundle exec bin/minised substitute -g a X
```

Représentation graphique (simplifiée) des étapes de l'exécution d'une commande



Note : Les boites — resp. lignes — en pointillés représentent des blocs — resp. des appels à des blocs (`call/yield`).

Les étapes de l'exécution d'une commande

Appel au niveau du *shell* :

```
$ bundle exec bin/minised substitute -g a X
```

- Crée un processus (Unix) pour exécuter…

  `bundle exec bin/minised substitute -g a X`

Note : Dans les diapositives qui suivent, l'en-tête du bloc indique les arguments traités par le programme ou la méthode indiquée.

Les étapes de l'exécution d'une commande

Dans `bundle` :

`exec bin/minised substitute -g a X`

- Configure les accès aux *gems* pour `bin/minised`
  (selon ce qui a été activé préalablement avec `bundle install`)

- Lance l'exécution (commande `exec` de `bundle`) de ...
  `ruby bin/minised substitute -g a X`

Les étapes de l'exécution d'une commande

Dans `ruby bin/minised` :

`"substitute", "-g", "a", "X"`

- Exécute `run(ARGV)`

  Où `run` est une méthode de classe du module `App` de `Gli` obtenue par `include GLI::App`

⇒ `GLI::App.run ["substitute", "-g", "a", "X"]`

Note : Les (4) arguments reçus par `bin/minised` sont dans `ARGV`.

Les étapes de l'exécution d'une commande

Dans `GLI::App.run` :

`["substitute", "-g", "a", "X"]`

- On obtient la description de la commande `substitute`
- On analyse les options et arguments :

```
global_options = {}
options = {:g => true}
args = ["a", "X"]
```

- On exécute l'action (le bloc) associée :

```
c.action do |global_options,options,args|
  motif, chaine, *fichiers = args
  fichiers = [:stdin] if fichiers.empty?
  in_p = global_options[:i] # Pas dans le vrai code

  fichiers.each do |fichier|
    MS.traiter_fichier( fichier, in_place: in_p ) do |flux_in, flux_out|
      MS.substitute( flux_in, flux_out, motif, chaine, options[:g] )
    end
  end
end
```

Les étapes de l'exécution d'une commande

Dans le bloc pour l'action associée à `substitute` :

```
global_options = {}
options = {:g => true}
args = ["a", "X"]
```

- On décompose les arguments
- On identifie les fichiers à traiter = `[:stdin]`
- On exécute...

```
# nil => Traitement qui n'est *pas* in-place
MS.traiter_fichier( :stdin, in_place: nil ) do |flux_in, f
  MS.substitute( flux_in, flux_out,
                 motif, chaine, options[:g] )
end
```

Les étapes de l'exécution d'une commande

Dans `MiniSed.traiter_fichier :`

`fichier = :stdin, in_place = nil`

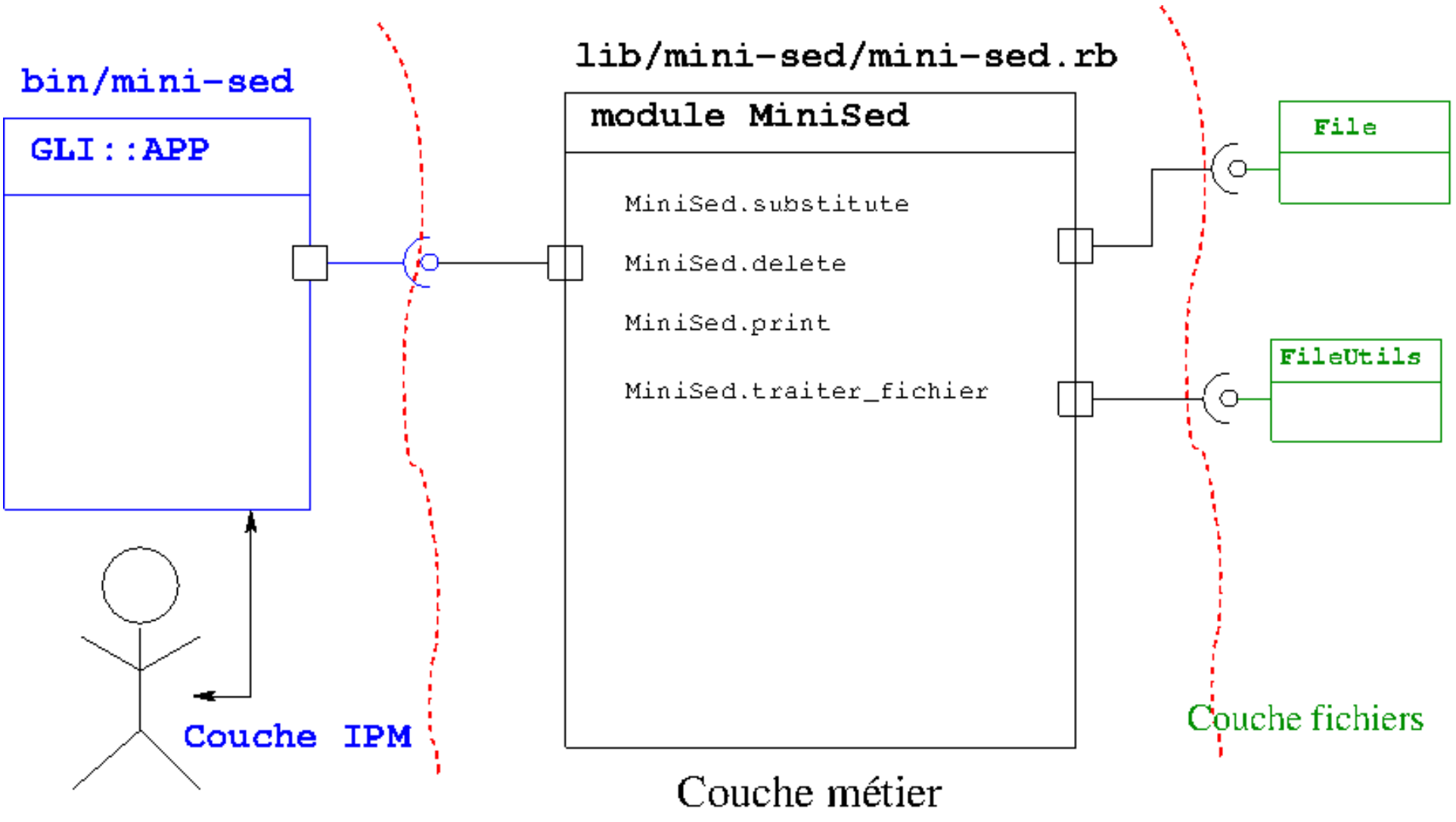
- On identifie les deux flux à utiliser :
  - Flux d'entrée = `$stdin`
  - Flux de sortie = `$stdout`
- On exécute `yield($stdin, $stdout)`

  Donc, on exécute (`true` pour substitution globale) :

  ```
  MS.substitute($stdin, $stdout, "a", "X", true)
  ```

7. Un peu de *Refactoring* dans `bin/minised`

Architecture de l'application `minised` : Les couches sont bien séparées ☺



Mais, dans `bin/minised`, du code se répète ☹


```
desc 'Supprime les lignes qui matchent un motif'
arg_name 'motif [fichier...]'
command :delete do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers = [:stdin] if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier, global_options[:"in-pl
        MiniSed.delete( flux_in, flux_out, motif )
      end
    end
  end
end
```

Mais, dans `bin/minised`, du code se répète ☹


```
desc 'Imprime les lignes qui matchent un motif'
arg_name 'motif [fichier...]'
command :print do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers = [:stdin] if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier, global_options[:"in-pl
        MiniSed.print( flux_in, flux_out, motif )
      end
    end
  end
end
```

On peut éliminer le code répétitif en introduisant, dans `bin/minised`, une méthode auxiliaire


```
def commande_sans_option( commande, nb_arguments: 1 )
  command commande do |c|
    c.action do |global_options,options,args|
    c.action do |global_options,options,args|
      les_args = args.shift(nb_arguments)
      fichiers = args.empty? ? [:stdin] : args
      inp = global_options[:"in-place"]

      fichiers.each do |fich|
        MiniSed.traiter_fichier(fich, inp) do |f_in,f_out|
          MiniSed.send commande, f_in, f_out, *les_args
        end
      end
    end
  end
end
```

On utilise ensuite cette méthode auxiliaire pour définir les commandes. . . qui n'ont pas d'option


```
desc 'Supprime les lignes qui matchent motif'
arg_name 'motif [fichier...]'
commande_sans_option :delete, nb_arguments: 1


desc 'Imprime les lignes qui matchent un motif'
arg_name 'motif [fichier...]'
commande_sans_option :print, nb_arguments: 1
```

C’est ensuite (très !) facile de définir de nouvelles commandes sans option


```
desc 'Supprime les lignes contenant un motif'
arg_name 'motif [fichier...]'
commande_sans_option :delete, nb_arguments: 1


desc 'Imprime les lignes contenant un motif'
arg_name 'motif [fichier...]'
commande_sans_option :print, nb_arguments: 1


desc 'Insere une ligne devant une qui matche le motif'
arg_name 'motif chaine_a_inserer [fichier...]'
commande_sans_option :insert, nb_arguments: 2
```

P.S. Et on pourrait faire de même pour des commandes avec options, si nécessaire. . .

8. Création d'un *gem* pour distribution

Pour créer un *gem* qu'on pourra distribuer, on doit tout d'abord finaliser le fichier `minised.gemspec`

On ajoute les informations sur l'auteur, la description détaillée du *gem*, etc.

```
$ emacs minised.gemspec
  ...
  s.author = 'Guy Tremblay'
  s.email = 'tremblay.guy@uqam.ca'
  s.homepage = 'http://www.labunix.uqam.ca'
  s.platform = Gem::Platform::RUBY
  s.summary = 'Une version simplifiee de sed'
  s.description = 'Une version simplifiee de sed,
                   avec des commandes a la git.
                   Utilise comme labo pour illustrer
                   les gems, dont gli'
  ...
  s.add_development_dependency(...)
  ...
```

Note : On raffine aussi certaines dépendances, pour éviter les messages d'avertissement. . .

On crée le *gem* avec la commande `gem build`

Il manque la licence d'utilisation... donc à compléter, mais autrement, le *gem* peut être distribué

```
$ minised
bash: minised: command not found

$ gem list minised

*** LOCAL GEMS ***


$ gem build minised.gemspec
WARNING:  [...]
  Successfully built RubyGem
  Name: minised
  Version: 0.0.1
  File: minised-0.0.1.gem

$ ls minised*gem
minised-0.0.1.gem
```

Le *gem* peut alors être distribué, installé, etc.

Pas de documentation `ri` pour ce *gem*, donc on utilise l'option «`--no-ri`»

```
$ gem install --no-ri minised-0.0.1.gem
Successfully installed minised-0.0.1
1 gem installed


$ gem list minised

*** LOCAL GEMS ***

minised (0.0.1)

$ minised
NAME
    minised - Programme qui emule sed (de facon simplifiee)

SYNOPSIS
    minised [global options] command [command options] [arguments...]

VERSION
    0.0.1
    ...
```

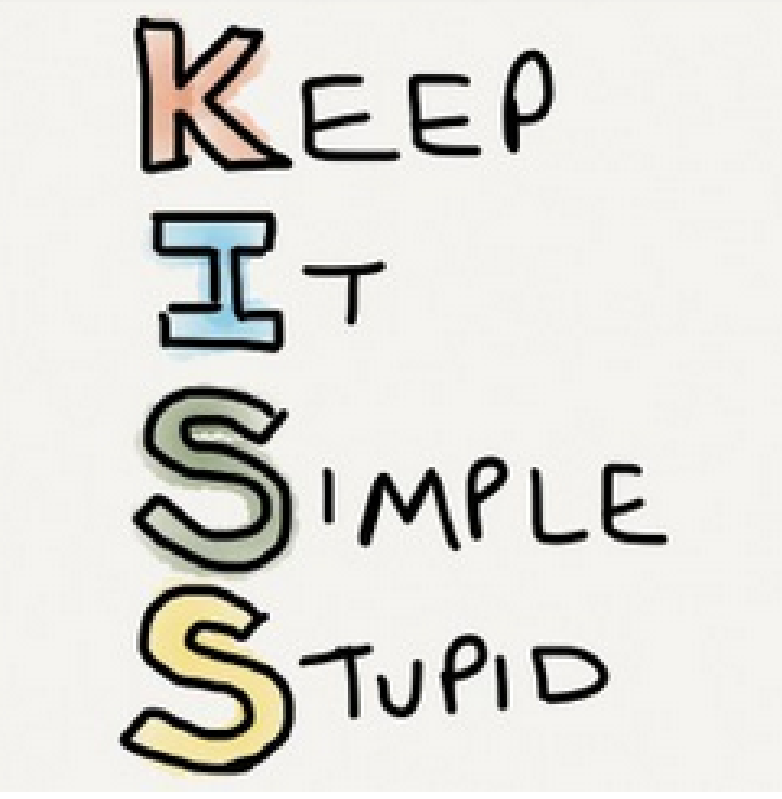
9. Conclusion sur l'exemple `minised`

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !



Source: `http://user47329.vs.easily.co.uk/tag/kiss-keep-it-simple-stupid/`

Source: `https://programmingenthusiast.wordpress.com/2015/05/08/`

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !

- Un des effets du *refactoring* est souvent d'introduire des méthodes auxiliaires.

- Si on définit des méthodes auxiliaires, alors on doit définir des tests unitaires pour ces méthodes, surtout si elles sont publiques !

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !

- Un des effets du *refactoring* est souvent d'introduire des méthodes auxiliaires.

- Si on définit des méthodes auxiliaires, alors on doit définir des tests unitaires pour ces méthodes, surtout si elles sont publiques !

- Et quand vous avez progressé, **faites un *commit* !**