Solution au devoir #2

1 Remarques et suggestions au niveau du style `Ruby`

a. Utilisez `attr_{reader,writer,accessor}` lorsqu'approprié :

```ruby
# Non!
def numero; @numero end
def date_achat; @date_achat end
...

# Ok
attr_reader :numero, :date_achat, ...
attr_accessor :note, :commentaire

# Mieux (pour gv.rb: + DRY)
attr_reader *READERS
attr_accessor *ACCESSORS
```

b. Utilisez le *pattern-matching* avec captures et groupes lorsqu'approprié :

```ruby
# Non!
if arg =~ /--format/
  le_format = arg.sub("--format=", "")

# Non! Non!
if arg =~ /--format/
  le_format = arg[9..-1] # 9 !?!?

# Ok -- et avec le Regexp *avant* l'operateur =~
if  /--format=(.*)/ =~ arg
  le_format = $1
```

c. Si un bloc ne contient qu'un appel direct à une méthode, utilisez la forme avec «`&:nom_methode`» :

```ruby
# Ok
les_vins.select { |v| v.bu? }

# Mieux
les_vins.select(&:bu?)
```

d. Évitez d'utiliser inutilement l'interpolation de chaine :

```
s = "..." # String

# Non!
ch = "#{s}"

# Ok
ch = s
# OU
ch = s.clone # Si on veut etre certain de ne pas modifier s
```

e. Utilisez les méthodes de `Enumerable`, par exemple, `Enumerable#find` :

```
# Non!
le_vin = nil
les_vins.each do |v|
  if v.numero == num then le_vin = v; break; end
end

# Ok
le_vin = les_vins.find { |v| v.numero == num }
```

f. N'utilisez `return` que pour retourner un résultat avant «la fin» — le résultat est toujours **la dernière expression évaluée.**

g. La méthode `shift` retourne l'élément retiré :

```
# Bof
a = ARGV[0]; ARGV.shift

# Ok
a = ARGV.shift
```

h. Utilisez les expressions court-circuitées lorsqu'approprié et évitez les comparaisons explicites avec `nil`:

```
# Non!
if le_format == nil # le_format.nil? preferable
  f = FORMAT_LONG
else
  f = le_format
end

# Ok
f = le_format || FORMAT_LONG
```

2 Extraits du fichier `gv.rb` illustrant le style fonctionnel (Enumerable)

```ruby
def lister( les_vins )
  if /^--(court|long|format=)/ =~ ARGV[0]
    le_format = case ARGV.shift
                when  /--court/ then FORMAT_COURT
                when  /--long/ then FORMAT_LONG
                when /^--format=(.*)$/ then $1
                else FORMAT_LONG
                end
  end

  verifier_arguments_en_trop( ARGV )

  les_vins
    .map { |e|  e.to_s(le_format) + "\n" }
    .join
end

def ajouter( les_vins )
  erreur_nb_arguments( ARGV ) unless ARGV.size == 0 || ARGV.size >= 4

  nouveaux_vins = les_vins_a_ajouter.flat_map do |qte, type, app, mil, nom,
    qte.times.map { Vin.creer(type, app, mil.to_i, nom, prix.to_f) }
  end

  les_vins + nouveaux_vins
end


# Identifie les vins a ajouter, soit le vin (unique) sur la ligne de
# commande, soit une serie de vins specifies via stdin.
#
def les_vins_a_ajouter
 ...
end
```

```ruby
def selectionner( les_vins )
  if /^--(bus|non-bus|tous)$/ =~ ARGV[0]
    option = ARGV.shift
    if /--bus/ =~ option
      les_vins = les_vins.select(&:bu?)
    elsif /--non-bus/ =~ option
      les_vins = les_vins.reject(&:bu?)
    end
  end

  motif = ARGV.shift

  verifier_arguments_en_trop( ARGV )

  les_vins.select { |v| motif ? /#{motif}/ =~ v.to_s : v }
end

def supprimer( les_vins )
  erreur_nb_arguments( ARGV ) unless ARGV.size == 0 || ARGV.size == 1

  vins_a_supprimer = les_numeros_a_supprimer.map do |numero|
    le_vin = vin_avec_numero(les_vins, numero)
    erreur( "supprimer: le vin numero '#{numero}' n'existe pas." ) unless l
    erreur( "supprimer: le vin numero '#{numero}' est deja note." ) if le_v

    le_vin
  end

  les_vins - vins_a_supprimer
end

# Retourne une liste des numeros de vin a supprime, soit un seul
# numero specifie via un argument recu sur la ligne de commande, soit
# 0, 1 ou plusieurs numeros specifies via stdin.
#
def les_numeros_a_supprimer
 ...
end
```

```ruby
def noter( les_vins )
  erreur_nb_arguments( ARGV ) unless ARGV.size >= 3

  numero, note, commentaire = ARGV.shift(3)

  verifier_arguments_en_trop( ARGV )

  le_vin = vin_avec_numero(les_vins, numero.to_i)

  erreur( "noter: le vin numero #{numero} n'existe pas." ) unless le_vin

  le_vin.noter( note.to_i, commentaire )

  les_vins
end


def vin_avec_numero( les_vins, numero )
  les_vins.find { |v| numero == v.numero }
end
```

```ruby
def trier( les_vins )
  champ_pour_cle = {
    'I' => :numero,
    'D' => :date_achat,
    'T' => :type,
    'A' => :appellation,
    'M' => :millesime,
    'N' => :nom,
    'P' => :prix,
    'n' => :note,
    'c' => :commentaire,
  }

  cles = []
  reverse = false
  while /^--(appellation|date-achat|millesime|nom|numero|prix|cle=.*|revers
          =~ ARGV[0]
    case option = ARGV.shift
    when /--cle=(.+)/
      cles = $1.each_char.reduce([]) do |cles, c|
        champ = champ_pour_cle[c]
        erreur "Cle invalide pour trier --cle: #{c}" unless champ
        cles << champ
      end
    when /--reverse/
      reverse = true
    else
      /--(.+)/ =~ option
      cles = [$1.to_sym]
    end
  end

  verifier_arguments_en_trop( ARGV )

  cles << :numero unless cles.include?(:numero)

  Vin.comparateurs = cles
  les_vins.sort { |v1, v2| reverse ? (v2 <=> v1) : (v1 <=> v2) }
end
```