MGL7460 — Automne 2016

Le patron «*Repository*»

Guy Tremblay
Professeur

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

20 octobre 2016

Contenu

1. Introduction/motivation

Comment peut-on manipuler des données persistentes, mais en faisant abstraction de leur représentation ?

Une solution possible, proposée par E. Evans = *Repository*

A Repository represents all objects of a certain type as a conceptual set. It acts like a collection, except with more elaborate querying capability.

«Domain-Driven Design—Tackling Complexity in the Heart of Software», E. Evans

Une solution possible, proposée par E. Evans = *Repository*

*A Repository represents all objects of a certain type as a conceptual set. **It acts like a collection**, except with more elaborate querying capability.*

«Domain-Driven Design—Tackling Complexity in the Heart of Software», E. Evans

Une solution possible, proposée par E. Evans = *Repository*

A Repository mediates between the domain and data mapping layers, acting like an in-memory domain object collection. [. . .] Objects can be added to and removed from the Repository, as they can from a simple collection of objects, and the mapping code encapsulated by the Repository will carry out the appropriate operations behind the scenes.

Source: *M. Fowler,* `http://martinfowler.com/eaaCatalog/repository.html`

Un *repository* devrait permettre de changer facilement le mécanisme de persistence

Repository

Methods for retrieving domain objects should delegate to a specialized Repository object such that alternative storage implementations may be easily interchanged.

Source: `https://en.wikipedia.org/wiki/Domain-driven_design`

Pourquoi la présentation de ce patron avec un exemple ?

- Parce qu'on utilisera cet exemples (classes/module) la semaine prochaine pour un laboratoire sur les tests d'acceptation.

Pourquoi la présentation de ce patron avec un exemple ?

- Parce qu'on utilisera cet exemples (classes/module) la semaine prochaine pour un laboratoire sur les tests d'acceptation.

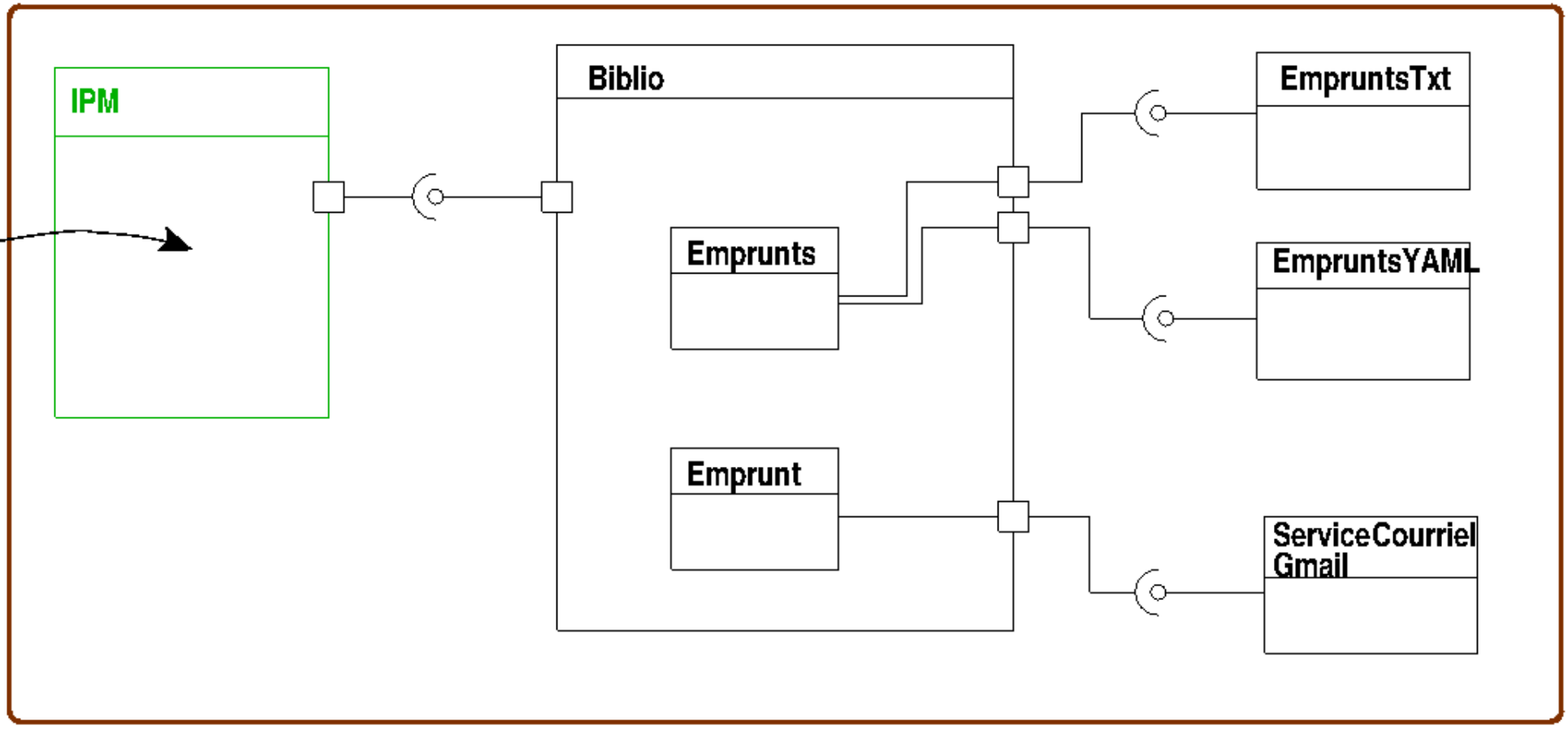
- Parce que cela pourrait vous être utile pour le projet #1 !

2. Un exemple : Des données persistentes pour des emprunts de documents

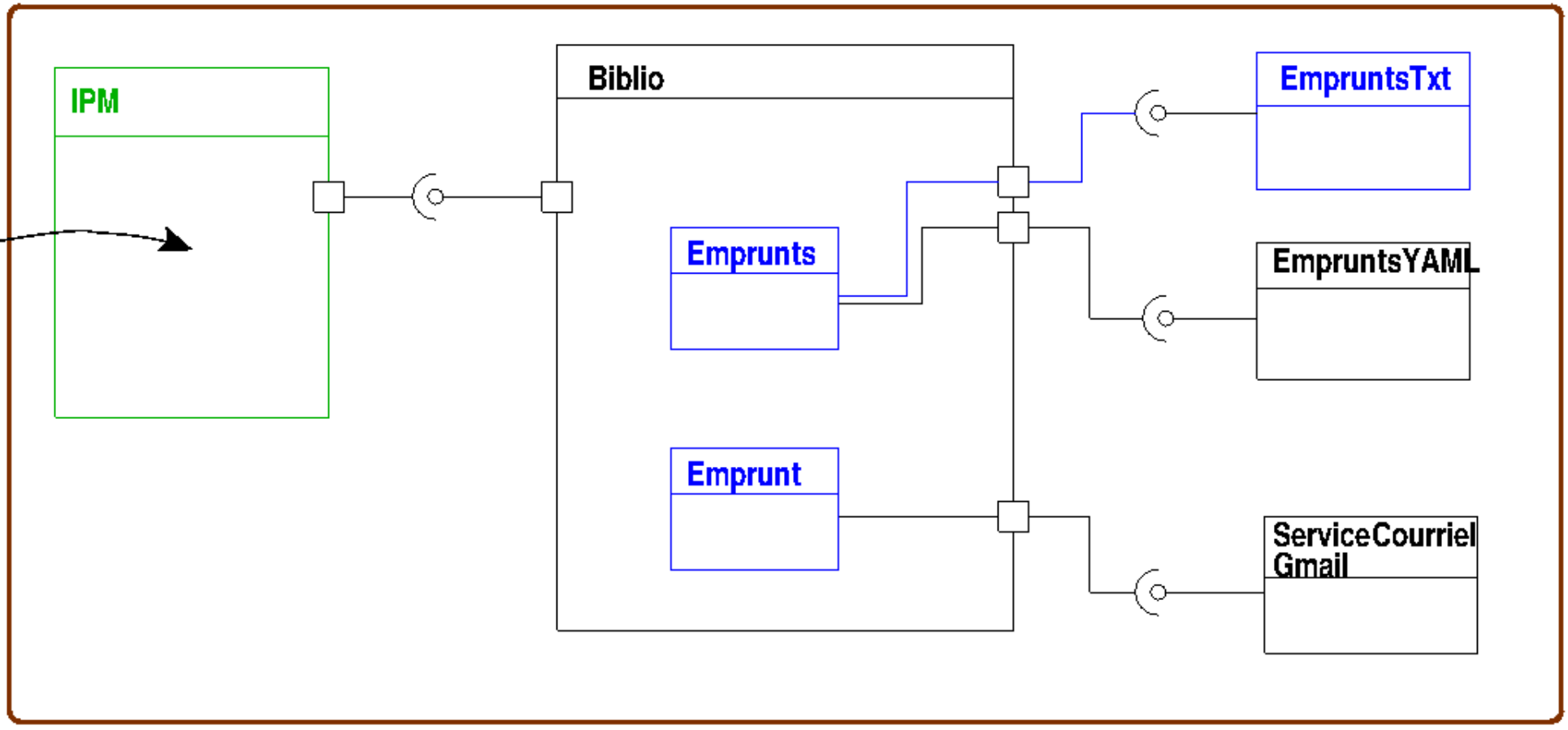
Le contexte = l'application

`biblio`

L'application pour la gestion de prêts de documents…



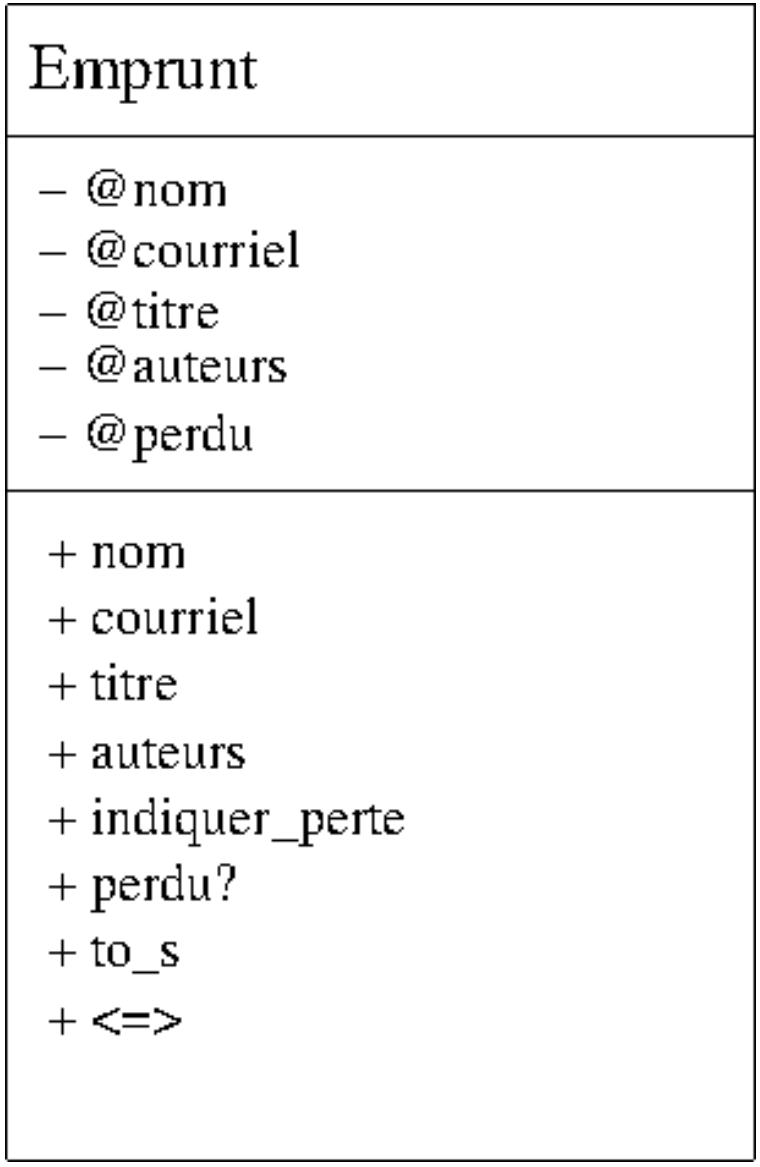
L'application pour la gestion de prêts de documents… mais pour trois classes seulement



Les classes et module examinés : `Emprunt`, `Emprunts` et `EmpruntTxt`

La classe `Emprunt`

Modélise un (1) emprunt pour un document



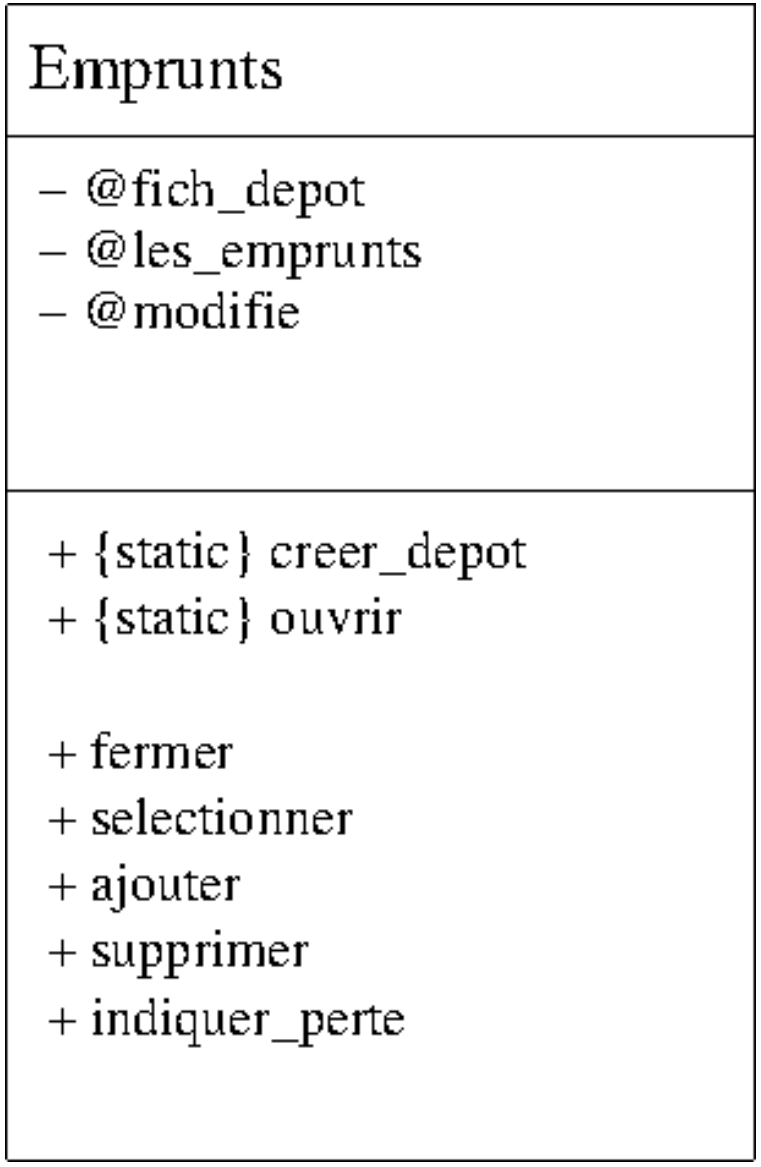
La classe `Emprunt`

Modélise un (1) emprunt pour un document

```
class Emprunt
  include Comparable

  attr_reader :nom, :courriel, :titre, :auteurs

  def initialize(nom, courriel,
                 titre, auteurs, perdu = false)
    ...
  end

  def to_s(le_format = "[ %N %C '%T' '%A' ]"); ...; end

  def <=>(autre); ...; end

  def indiquer_perte; @perdu = true; end

  def perdu?; @perdu; end
end
```

La classe `Emprunts` = le *repository*

Modélise une collection d'emprunts



La classe `Emprunts` = le *repository*

Méthodes de classe

```
class Emprunts
  def self.creer_depot(fich_depot, opts = {:detruire_si_existe
    ...
  end

  def self.ouvrir(fich_depot, opts = {:sauvegarder => false})
    ...
  end

  def initialize( fich_depot )
    ...
  end

  private_class_method :new

  ...
end
```

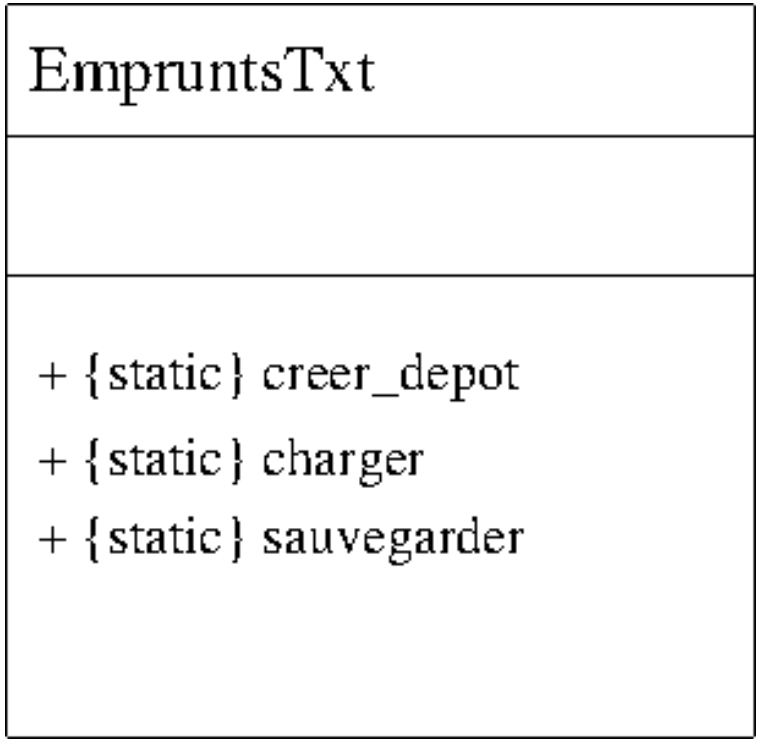
La classe `Emprunts` = le *repository*

Méthodes d'instance

```
class Emprunts
  ...
  def fermer
    ...
  end

  def selectionner( opts = {:unique => false}, &selecteur )
    ...
  end

  def ajouter( emp )
    ...
  end

  def supprimer( emp )
    ...
  end

  def indiquer_perte( emp ); ...; end
end
```

Le module `EmpruntsTxt`

Connait les détails du format du fichier texte utilisée pour les données persistentes



Le module `EmpruntsTxt`

Connait les détails du format du fichier texte utilisée pour les données persistentes

```
module EmpruntsTxt
  def self.creer_depot( fich_depot,
            opts = {:detruire_si_existe => false} )
    ...
  end

  def self.charger( fich_depot )
    ...
  end

  def self.sauvegarder( fich_depot, les_emprunts )
    ...
  end

  private
  SEPARATEUR = "%"

  # Autres methodes privees, liees a la representation choisie
  ...
end
```

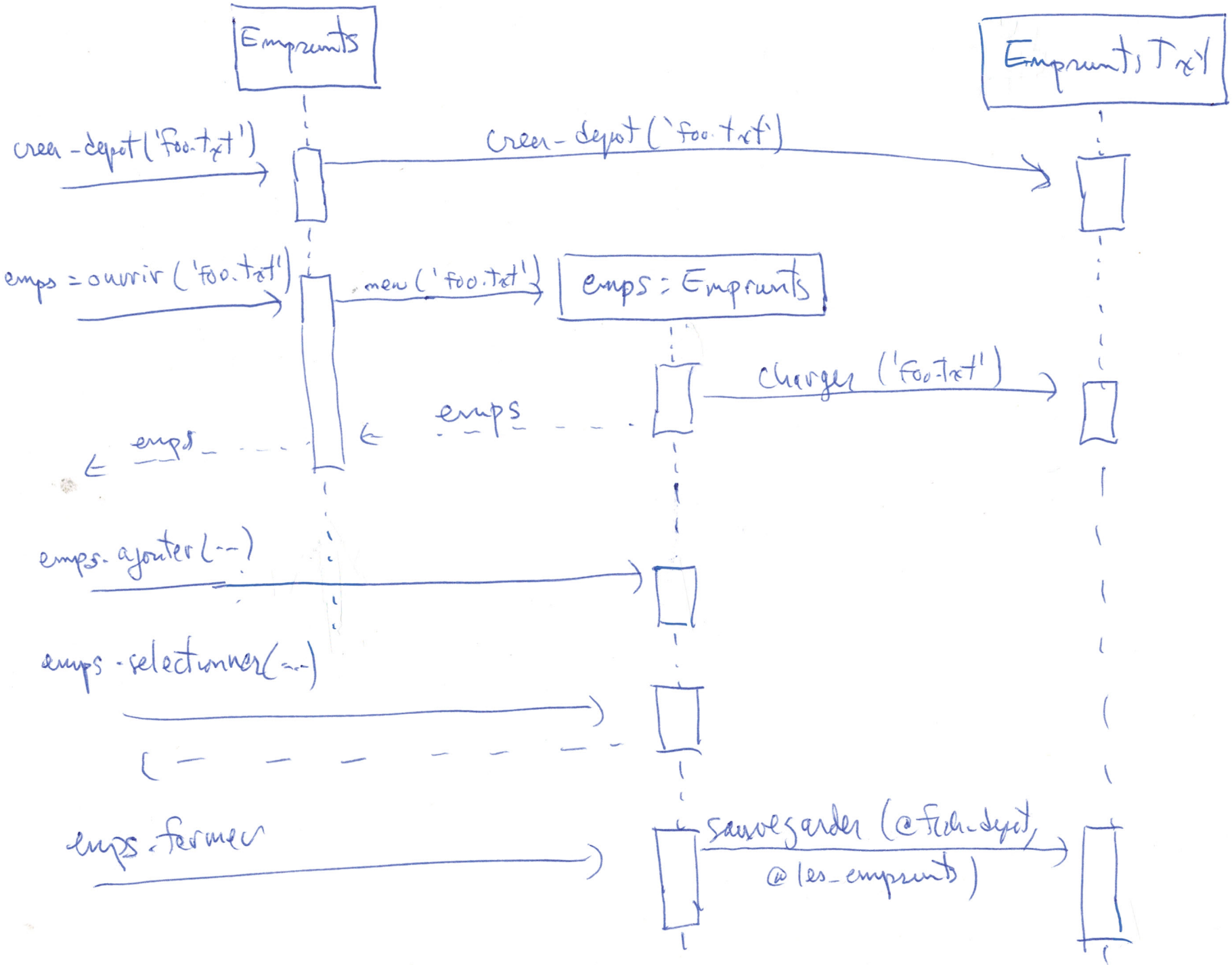
Le comportement dynamique

Un exemple illustrant les interactions entre ces objets

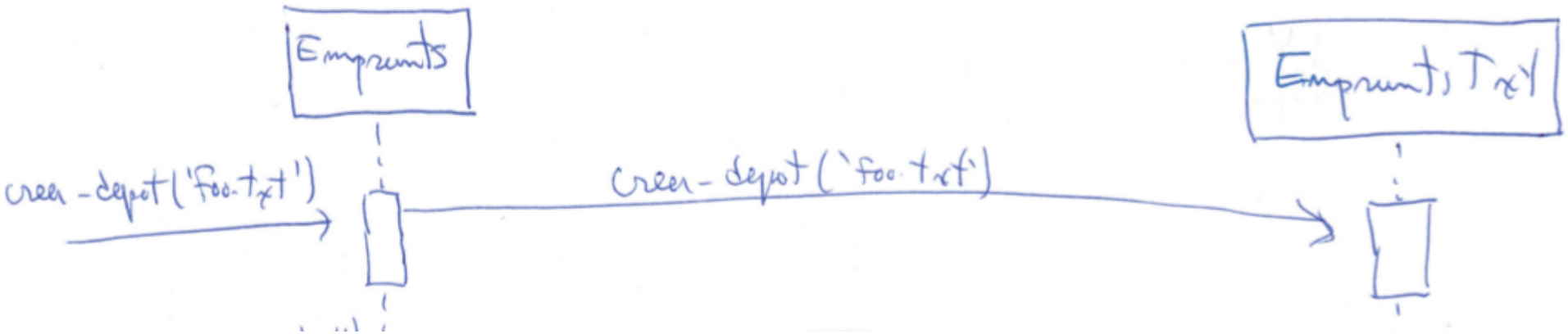
Voir les diagrammes UML d'interactions dans les diapositives suivantes qui illustrent les échanges de messages.

On veut exécuter les instructions ci-bas.

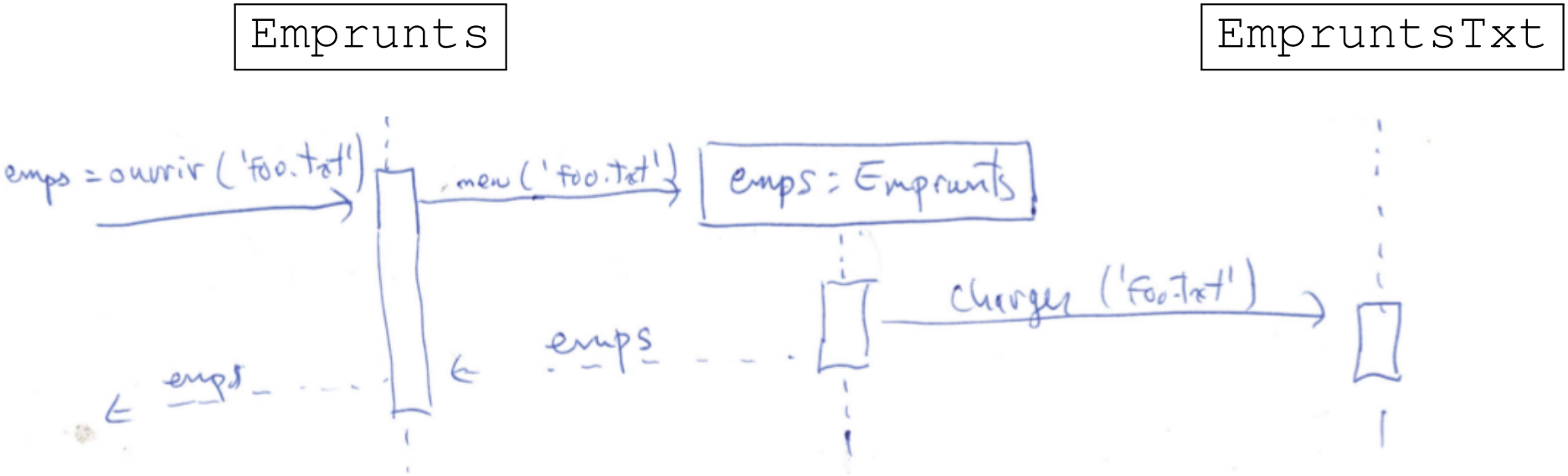
```
# a
Emprunts.creer_depot( 'foo.txt' )

# b
emps = Emprunts.ouvrir( 'foo.txt' )

# c
emps.ajouter( Emprunt.new("Guy T.", "@",
                  "Code Complete", "McConnell") )

# d
puts emps.selectionner { |e| e.titre =~ "Code" }

# e
emps.fermer
```


Emprunts
Emprunts.Txt
creer_depot('foo.txt')
creer_depot('foo.txt')
emps = ouvrir('foo.txt')
new('foo.txt')
emps : Emprunts
charger('foo.txt')
emps
emps
emps.ajouter(...)
emps.selectionner(...)
emps.fermer
sauvegarder(@fich_depot, @les_emprunts)

a. `Emprunts.creer_depot( 'foo.txt' )`

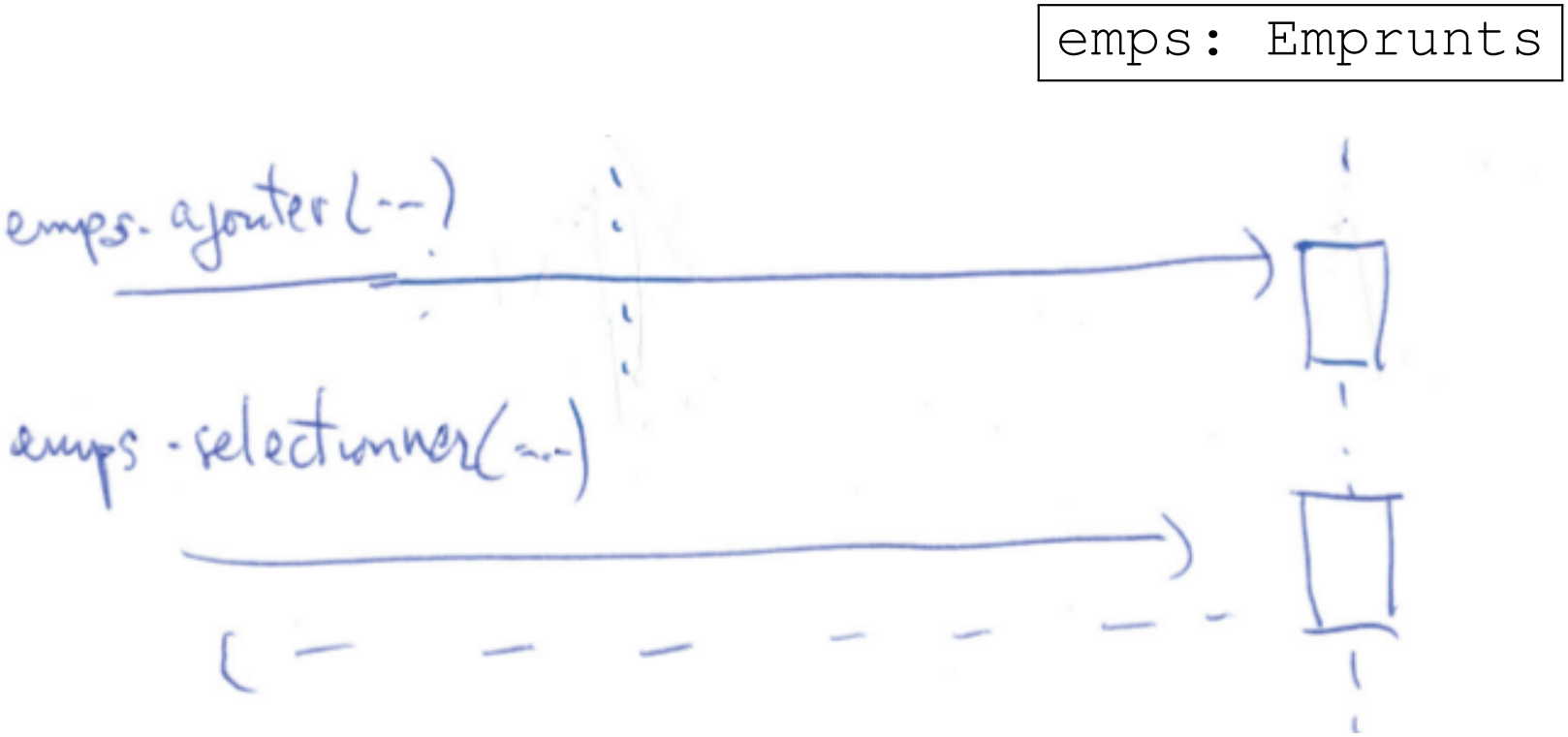


b. `emps = Emprunts.ouvrir( 'foo.txt' )`



c. `emps.ajouter(...)`

d. `puts emps.selectionner {...}`



e. emps.fermer