# Programmation – TD8 : Des objets qui ont la classe*

{ Jeremie.Detrey, Emmanuel.Jeandel } @ens-lyon.fr

1er/2 décembre 2004

## 1 A tribute to Julien Musset™

### 1.1 Modélisation

Une banque est une entreprise dont l'activité principale consiste à consentir des crédits et à collecter l'épargne, ce qui lui donne la fonction bien particulière d'émettre de la monnaie. En outre, elle propose à sa clientèle des services divers (conseil aux épargnants, réalisation d'opérations financières, location de coffres...). Enfin, elle peut investir directement dans le capital des sociétés à condition de respecter des règles de sécurité qu'édictent les pouvoirs publics dans le but d'éviter le krach (faillite bancaire) des établissements qui auraient trop misé sur des affaires en difficulté (on s'exposerait alors à l'écroulement d'un système frappé par la défiance puis la panique des déposants). Les banques sont répertoriées parmi les "établissements de crédit" dans la nomenclature des secteurs institutionnels de la comptabilité nationale. On dit aussi "banque commerciale" et "banque de second rang".

Une *action* est un titre représentatif de la propriété d'une partie du capital d'une société anonyme ou en commandite par actions (sociétés de capitaux). Elle donne droit à participer aux assemblées générales d'actionnaires, qui élisent les dirigeants et prennent les décisions fondamentales (une voie par action le plus souvent), et à une fraction des bénéfices ("dividende") proportionnelle à la part du capital de l'entreprise que représente l'action. C'est fou comme ça peut être chiant ce truc, hein. Celle-ci ne fait pas mention de l'identité du propriétaire (titre "anonyme" ou "au porteur").

Une dévaluation a pour objectif premier le rétablissement de la compétitivité des productions nationales sur les marchés mondiaux grâce à la diminution du prix des exportations exprimé en devises et à l'augmentation de celui des importations converti en monnaie domestique. On espère ainsi restaurer l'équilibre de la balance des opérations courantes. Fondamentalement, il s'agit de desserrer les contraintes extérieures en donnant un avantage à OCaml de façon à pouvoir éventuellement mettre en œuvre une expansion anti-chômage, malgré l'inflation intérieure et l'accroissement de la demande d'importations qui en découleront. Parallèlement, en diminuant le prix de sa monnaie, on la rend attractive pour la spéculation et on pourra pratiquer un taux d'intérêt intérieur modéré pour stimuler les investissements, sans risquer une fuite des capitaux vers l'étranger. En outre, la dévaluation, satisfaisant les attentes des spéculateurs, doit renverser la tendance à la baisse du taux de change, donc éviter à la banque centrale de dilapider ses réserves en devises par ses interventions sur le marché.

Le triangle d'impossibilité (ou d'incompatibilité) des bermudas est la représentation graphique du fait qu'on ne peut avoir à la fois la fixité des taux de change, la liberté des mouvements de capitaux et l'autonomie de la politique monétaire ; les gouvernements doivent choisir deux de ces trois propositions ; on dit aussi triangle "d'Aglietta", "de Tinbergen" ou "de Mundell", du nom des auteurs ayant développé cette question. Voir le schéma ci-dessous :

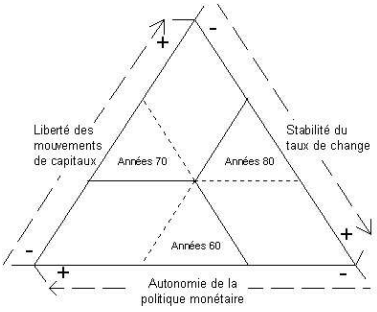


**FIG. 1:** Le triangle d'impossibilité.

Sous réserve de l'obtention du consentement du Surintendant des institutions financières, la Banque peut racheter au comptant les actions privilégiées série 15 à compter du 15 mai 2008, en totalité ou en partie, à tout moment, au gré de la Banque, à un prix correspondant à 26,00 euros l'action si elles sont rachetées avant le 15 mai 2009, à 25,75 euros si elles sont rachetées au cours de la période de 12 mois qui précède le 15 mai 2010, à 25,50 euros l'action si elles sont rachetées au cours de la période de 12 mois qui précède le 15 mai 2011, à 25,25 euros l'action si elles son rachetées au cours de la période de 12 mois qui précède le 15 mai 2012, et à 25,00 euros l'action si elles sont rachetées à compter du 15 mai 2012, dans chaque cas, majoré de tous les dividendes déclarés et impayés sur celles-ci jusqu'à la date fixée pour le rachat.

Les porteurs d'actions privilégiées série 15 auront le droit de recevoir un dividende au comptant privilégié non cumulatif trimestriel, s'il est déclaré par le conseil d'administration, le quinzième jour de février, de mai, d'août et de novembre de chaque année (la "date de versement d'un dividende"), à un taux trimestriel correspondant à 0,365625 euros par action. Le dividende initial, s'il est déclaré, sera payable le 15 mai 2003 et s'élèvera à 0,416712 par action, en fonction d'une date de clôture prévue du 31 janvier 2003.

*ou pas...

### 1.2 Questions

1. Mais où est Charlie ?
2. Quelle est la probabilité que Microsoft décide de faire une OPA hostile sur ST Microelectronics dans les 6 prochains mois ?
3. Sachant que votre TDman favori a acheté 20 actions France Télécom le 24 juillet 2002, profitant de son dividende sur une plusvalue de son stock d'options SCO, calculez en fonction de la variation différentielle du cours quand il devra revendre ces actions pour en tirer le meilleur rendement en terme de rapport bénéfices sur risque amorti.
4. Pour pouvoir répondre de manière plus rapide à ces questions, écrivez un simulateur de capitalisme$^{TM}$ en utilisant le paradigme de programmation par objets.
5. Débuggez une paire de feuilles de style Excel$^{TM}$, allez voir votre patron, et réclamez-lui votre bonus de \$100.000.

C'est chouette la finance en fait hein ?

…en fait non.

6. Étendez votre modèle avec une classe `prolétariat`, que vous munirez d'une méthode `molotov` bien choisie, et observez le résultat.

## 2 Oh, quel beau zob j'ai !

### 2.1 Dude, where's my `rec` ?

1. Écrivez une classe `toto` munie d'une fonction `fact` calculant récursivement la factorielle de son argument, le tout sans utiliser le mot-clé `rec`.

### 2.2 Les types OCaml à la sauce objets

Généralement, dans tout langage objet qui se respecte (comme Java par exemple[1]), même les types de base (entiers, ...) sont en fait des objets, et tous ces objets héritent d'une seule et même classe, ancêtre commun à toutes les autres.

En OCaml, nous représenterons cette classe ancêtre par la classe virtuelle `oo_base` définie comme suit :

```
class virtual oo_base =
object
  method virtual to_string : unit -> string
end;;
```

La méthode `to_string` de cette classe permet de convertir une instance de la classe en une chaîne de caractères pour pouvoir l'afficher par exemple.

1. Pourquoi cette classe est-elle virtuelle ? Que se passe-t-il si l'on instancie un objet de cette classe par un `new oo_base` ?

[1] Hum...

Nous pouvons alors représenter les entiers par la classe `oo_int` suivante :

```
class oo_int n0 =
object
  inherit oo_base
  val mutable n : int = n0
  method to_string () = string_of_int n
end;;
```

2. Étendez la classe `oo_int` de quelques méthodes permettant de manipuler les entiers. Par exemple, `add`, `neg`, `fact` (encore), etc...
3. Écrivez de même une classe `oo_string` pour les chaînes de caractères.
4. Idem pour `oo_list` et `oo_fun` représentant respectivement des listes et des fonctions.

Pour l'instant, votre méthode `to_string` renvoie une `string`. Pourtant on cherche à objectiser tous les types OCaml.

5. *Bonus* : Modifiez donc tout ça pour que `to_string` renvoie un objet de type `oo_string`. Pour cela, prenez la définition suivante pour `oo_base` :

```
class virtual oo_base =
object
  method virtual to_string : unit -> (< to_string : unit -> 'a ;
                                         print : unit -> unit > as 'a)
end;;
```

### 2.3 Typage de `self`

Allez voir le manuel OCaml (`http://caml.inria.fr/ocaml/htmlman/index.html`), puis allez lire la page sur le module `Oo` de la librarie standard. Plus précisément, intéressez-vous à la fonction `Oo.copy`.

1. Regardez le type de `Oo.copy` et essayez de comprendre ce que signifie cette horreur. Ne vous privez pas pour faire plein d'essais avec cette fonction, histoire d'avoir les idées claires.
2. Ensuite, expliquez la différence (et mettez la en évidence par un exemple bien choisi) entre les deux classes suivantes :

```
class c1 = object (self) method c = Oo.copy self end;;
class c2 = object (self) method c = new c2 end;;
```

## 3 La sempiternelle GUI[2]

### 3.1 Un exemple de pattern : sujet-observateur

En programmation, il existe des idiomes standards, appelés *patterns*, qui sont par exemple destinés à faciliter la modularisation, l'extension ou la maintenance de gros logiciels. Ils défi-

[2]Vos TDmen vous présentent par avance leurs excuses les plus plates quant à ce sujet manquant cruellement d'intérêt et d'originalité. En plus, on a tout recopié du sujet de l'an dernier (merci Tom), nous sommes vraiment impardonnables.

nissent des conventions de programmation, de sorte qu'un programmeur extérieur peut rapidement étendre le logiciel en question en suivant le protocole imposé par le pattern.

Ici, nous considérons un gestionnaire de fenêtres programmé selon le pattern *sujet-observateur*. L'idée de ce pattern est que l'observateur reçoit de l'information de la part de ses sujets et décide des actions à accomplir. Il envoie ensuite des ordres à ses sujets pour exécuter ces actions. Un sujet peut de plus avoir plusieurs observateurs. Nous implantons ce pattern par deux classes paramétrées comme suit :

```
class ['observer] subject =
object (self : 'mytype)
  val mutable observers : 'observer list = []
  method add obs = observers <- obs :: observers
  method notify (message : 'observer -> 'mytype -> unit) =
    List.iter (fun obs -> message obs self) observers
end;;

class ['subject] observer = object end;;
```

### 3.2 Votre première fenêtre

Pour adapter le pattern à un cas concret, nous devons étendre `subject` avec les actions que les observateurs peuvent invoquer et `observer` avec les informations que les sujets peuvent envoyer. Par exemple, une classe `window` héritant de `subject` l'enrichira par une méthode `move` et préviendra ses observateurs en appelant leur méthode `moved`. Evidemment, la classe `manager` correspondante contiendra dans la méthode `moved` toutes les actions résultant du mouvement d'une fenêtre. Typiquement, elle appellera la méthode `draw` du sujet concerné.

1. Écrivez les classes et les méthodes décrites précédemment.
   *Remarque* : pas besoin de perdre du temps à faire quoi que ce soit de graphique. La méthode `draw` de la fenêtre pourra par exemple afficher un petit texte du genre :

   ```
   Drawing window "Example" at location x,y = 300,150.
   ```

### 3.3 Des widgets par milliers

Une fenêtre déplaçable c'est bien, mais ça serait encore mieux si on pouvait aussi la redimensionner. Et puis il faudrait aussi y mettre des trucs (des *widgets* comme on dit quand on se la pète) dedans : des boutons, des menus, des barres de défilement, ...

1. Réfléchissez bien à comment faire tout ça.
   *Remarque* : pensez à la réutilisation de code : moins de code à écrire, ça fait moins de bugs, et donc plus de temps pour aller picoler au foyer.
2. Faites-le !