

# Quelques points concernant OCaml

## Génie Logiciel - MPRI

GROSSHANS Nathan  
`nathan.grosshans@lsv.ens-cachan.fr`

23 octobre 2014

# Introduction

## Objectifs de ce tutoriel

Apporter quelques bases concernant les fonctionnalités “objet” d’OCaml, ainsi qu’évoquer des outils facilitant la documentation et les tests de code OCaml.

## Points abordés

1. Bases de la POO en OCaml.
2. Documentation avec ocamlldoc.
3. Tests unitaires avec OUnit.
4. Intégration continue avec Travis CI.

## Manuel de référence pour OCaml

<http://caml.inria.fr/pub/docs/manual-ocaml/index.html>

# Bases de la POO en OCaml

## Déclaration simple

```
class <class_name> <parameters> =  
object  
  val mutable <attribute_1> = <init_value_1>  
  <...>  
  val mutable <attribute_n> = <init_value_n>  
  method <method_1> <parameters_1> = <code_1>  
  <...>  
  method <method_m> <parameters_m> = <code_m>  
end;;
```

## Instanciation d'un nouvel objet

```
new <class_name>;;
```

# Bases de la POO en OCaml

## Appel de méthodes

```
<object_name>#<method_name> <parameters >;
```

## Remarques importantes

- ▶ Le **type d'un objet** en OCaml n'est pas donné par un nom de classe comme en C++ ou en Java, mais par **une signature formée de la signature des méthodes publiques de l'objet**.
- ▶ Le nom de la classe ayant servi à la construction d'un objet est simplement une **abréviation pour son type**.
- ▶ Il est donc parfaitement possible de **construire un objet directement, sans passer par une classe**.
- ▶ Il n'est pas possible d'accéder directement aux attributs d'un objet en OCaml.

# Bases de la POO en OCaml

## Méthodes privées

```
method private <method_name> <parameters> =  
  <code>
```

**Attention** : les méthodes privées peuvent être uniquement appelées par d'autres méthodes du **même objet**.

## L'auto-référence

```
class <class_name> =  
object (<self >)  
  <class_body>  
end ; ;
```

- ▶ On peut faire référence à l'objet courant en utilisant <self> dans le corps de la classe.
- ▶ L'utilisation de l'auto-référencement est obligatoire pour appeler des méthodes de l'objet courant.

# Bases de la POO en OCaml

## Classes paramétrées

```
class [<types>] <class_name> =  
  object (<self> : <class_type>)  
    <class_body>  
  end;;
```

- ▶ Lorsque la classe contient des **méthodes polymorphes**, cette première doit obligatoirement être **paramétrique**. Le typage de certains paramètres ou méthodes polymorphes doit ensuite être imposé en utilisant les variables de type en paramètre.
- ▶ <class\_type> permet d'utiliser le type défini par la classe dans son propre corps.
- ▶ Il est possible de faire des choses beaucoup plus fines et subtiles quant aux contraintes sur les types donnés en paramètres, voir notamment l'utilisation de `constraint`.

# Bases de la POO en OCaml

## Héritage

Dans le corps de la classe concernée :

```
inherit <class_name> <parameters>
```

ou

```
inherit [<types>] <class_name> <parameters>
```

au cas où l'on veut hériter d'une classe paramétrée.

- ▶ Tout est bien évidemment hérité, autant les attributs que les méthodes, qu'elles soient publiques ou privées.
- ▶ OCaml permet l'utilisation de l'**héritage multiple**. Il suffit pour cela de répéter la construction ci-dessous pour chaque classe de laquelle l'on souhaiterait hériter.

# Bases de la POO en OCaml

## Classes virtuelles

```
class virtual [<types>] <class_name> =  
object (<self> : <class_type>)  
  <class_body>  
end;;
```

## Méthodes virtuelles

```
method virtual <method_name> : <method_type>
```

**Attention** : une classe contenant des méthodes virtuelles doit elle aussi l'être.



# Documentation avec ocaml doc

## Quoi ?

Outil de **génération de documentation** présent dans l'écosystème de base d'OCaml.

## Pourquoi ?

- ▶ Pour celui qui développe le bout de code concerné : en plus de le forcer à s'interroger sur l'utilité et la pertinence de ce qu'il fait, permet de donner une **spécification explicite** à **respecter**.
- ▶ Pour les autres : permet de **réutiliser directement** le bout de code concerné sans avoir à le lire et à le comprendre ou à demander des explications au développeur concerné.

## Comment ?

- ▶ Exemple.
- ▶ <http://caml.inria.fr/pub/docs/manual-ocaml/ocaml doc.html>.

# Tests unitaires avec OUnit

## Quoi ?

Environnement permettant de faciliter la **mise en place et l'exploitation de tests unitaires** pour du code OCaml.

## Pourquoi ?

**Parce qu'il faut faire des tests !**

## Comment ?

<http://ounit.forge.ocamlcore.org/>.

# Intégration continue avec Travis CI

## Quoi ?

Service d'intégration continue intégré à GitHub, permettant de compiler tout un projet et de lancer une batterie de tests automatiques dans un environnement frais à chaque modification du dépôt Git.

## Pourquoi ?

Pour rendre le processus de test encore plus **automatique** et **efficace**, afin de **détecter plus rapidement les problèmes**.

## Comment ?

- ▶ `http://docs.travis-ci.com/user/getting-started/`.
- ▶ `http://anil.recoil.org/2013/09/30/travis-and-ocaml.html`.

# Fin

Des questions ?