Programmation Fonctionnelle – Travaux Dirigés

Séance 8: Labels et objets

L3 – Université Nice Sophia Antipolis

Exercice 1 (Range, ⋆)

Définissez la fonction `range` *inspirée de Python avec un argument obligatoire et deux arguments optionnels qui renvoie une liste de nombres. Par exemple*

- `range 5` *renvoie* `[0;1;2;3;4]`
- `range 5 ~from:3` *renvoie* `[3;4]`
- `range 5 ~from:2 ~step:2` *renvoie* `[2;4]`

Pour simplifier, on supposera que l'argument `step` *est toujours un entier strictement positif.* □

Exercice 2 (Compteur avec historique, ⋆⋆)

1. *Définissez une classe* `compteur` *avec les méthodes* `get`, `set`, *et* `incr`.
2. *Définissez la classe* `compteur2` *qui hérite de* `compteur` *et qui ajoute une méthode* `undo`. *La méthode* `undo` *annule le dernier* `set` *ou* `incr`. *Indication : vous conserverez l'historique des valeurs antérieures du compteur dans une liste.*

□

Exercice 3 (Pile, ⋆⋆)

1. *Définissez une classe* `pile` *qui contienne les méthodes suivantes :* `empile x`, `depile` *(qui renvoie le sommet de pile), et* `est_vide` *(qui renvoie vrai si la pile est vide). Indication : la principale difficulté est le typage. Il faudra démarrer la déclaration par* `class ['a] pile` *et utiliser le type* `'a` *au moins une fois dans une annotation de type.*
2. *Définissez une classe* `pile_etendue` *sous-classe de la classe pile et qui ajoute les méthodes suivantes :*
   - `copy` *qui renvoie un objet copie de* `self`
   - *une méthode* `sommet` *qui renvoie le sommet de pile (sans le dépiler)*
   - `empile_persistant x` *qui renvoie un objet copie de* `self` *dans lequel* `x` *a été empilé, sans l'empiler dans* `self`
   - `depile_persistant` *qui renvoie un objet copie de* `self` *dans lequel le sommet a été dépilé.*

□

Exercice 4 (Expressions arithmétiques, ⋆⋆)

OCaml comme Java permet de définir des classes abstraites. Par exemple, une classe abstraite pour des expressions arithmétiques est

```
class virtual expr = object
  method copy = {<>}
  method virtual eval : float
  method virtual print : unit
end
```

1. *Définissez une classe concrète* `constante` *sous-classe de* `expr`.
2. *Définissez une classe abstraite* `bin_op` *qui implémente* `eval` *et* `print` *à l'aide de deux nouvelles méthodes virtuelles* `oper:(float * float ) -> float` *et* `symbol:string`.
3. *Définissez des sous-classes concrètes* `add` *et* `mult` *de* `bin_op` *qui implémentent* `oper` *et* `symbol`.

□

Exercice 5 (Méthode binaire, ⋆⋆)
On souhaite représenter deux classes `reel` *et* `complexe` *avec une notion de sous-classe. Une tentative est la suivante*

```
class reel (x:float) = object
val x = x
method reel = x
method eq (y:reel) = x=y#reel
end

class complexe (x:float) (y:float) = object
inherit reel x
val y = y
method im = y
method eq (z:complexe) = x=z#reel && y=z#im
end
```

1. *Quel est le problème (et pourquoi ne se pose-t-il pas en JAVA) ?*
2. *Quelle est la solution ?*

□

Exercice 6 (Constructeur surchargé, ⋆)
Comment feriez-vous la classe suivante en OCaml ?

```
class date {
  private final int year;
  private final int month;
  date(int year){ self.year=year; self.month=1;}
  date(int year, int month){self.year=year; self.month=month;}
  String toString(){return month+"/"+year;}
}
```

□