Pour que la sémantique d'un terme soit la même en appel par valeur et en appel par nom, il faut éviter que l'interprétation des champs produise des effets secondaires — comme ici, créer une nouvelle référence. On réécrira donc ce terme en

```
let r = ref 0
in let x = {a = r}
in let inc = fun y -> (y.a := 1 + !(y.a))
in (inc x; !(x.a))
```

dont la valeur est `1`, que les enregistrements soient en en appel par valeur ou en appel par nom.

Exercice 8.5 *(Le type des enregistrements) Supposons que l'on ait un type* `personne` *des enregistrements qui comportent les trois champs :* `nom`, `prenom` *et* `telephone`. *Montrer que l'on peut programmer les trois fonctions de* `x` *et* `y` `x(nom <- y)`, `x(prenom <- y)` *et* `x(telephone <- y)` *sans utiliser le symbole* `<-`, *et que, de ce fait, ce symbole est superflu.*

Ce symbole reste-t-il superflu si on a un type `telephonable` *de tous les enregistrements qui comportent au moins le champ* `telephone` *?*

Si on a un type `personne` *et un type* `telephonable`, *a-t-on toujours l'unicité des types ?*

8.2 Les objets

Un programme traite en général des données assez diverses, souvent exprimées par des enregistrements. Par exemple, le système informatique d'une entreprise traite des bons de commandes, des fiches de paie, ... et un bon de commande est un enregistrement qui contient l'identification de l'objet commandé, la quantité commandée... Il y a plusieurs possibilités pour imprimer ces données. On peut écrire une fonction `imprime` unique qui commence par tester la nature de la donnée — bon de commande, fiche de paie, ... — et qui l'imprime selon un certain format fonction de cette nature. On peut écrire plusieurs fonctions : `imprime_bon_de_commande`, `imprime_fiche_de_paie`, ... On peut avoir un enregistrement `imprime` dont chacun des champs est une fonction d'impression. Une dernière solution est que la fonction d'impression d'un type de données fasse partie de la définition même de ce type de données. Un tel type de données s'appelle alors une classe, et ses éléments des *objets*.

Dans la forme la plus radicale de la programmation avec des objets, chaque objet, par exemple chaque bon de commande, a une fonction `imprime` différente. Un bon de commande est donc un enregistrement qui contient, en plus des champs habituels — identification de l'objet commandé, quantité commandée, ... — un champ `imprime` qui est une fonction d'impression que l'on peut appliquer à cet objet lui-même pour l'imprimer.

Certains langages, comme Java, associent une fonction `imprime`, non à chaque objet, mais à chaque classe d'objets. Ainsi, deux objets de la même classe partagent la même fonction d'impression — et ce que cette fonction soit statique ou