```
y = x;
```

En revanche en Caml rien n'empêche d'écrire

```
let y = ref 4
in let x = ref 5
in let x = ref 6
in y := !x
```

et ce programme affecte la valeur `6` *à la variable* `y`*, car c'est la dernière déclaration de* `x` *qui prime. On dit que la première déclaration de* `x` *est* cachée *par la seconde.*

Java, Caml et C permettent de déclarer une variable avec une valeur initiale en indiquant que l'on ne va pas affecter cette variable par la suite. Une telle variable est appelée *finale* et une variable qui n'est pas finale est appelée *mutable*. En Java on fait précéder le type de la variable du mot clé `final`, par exemple

```
final int x = 4;
y = x + 1;
```

pour indiquer qu'elle est finale, et elle est mutable sinon. L'instruction

```
final int x = 4;
x = 5;
```

dans laquelle on tente d'affecter une variable finale, est incorrecte.

En Caml, pour indiquer que la variable `x` *est finale, on écrit* `let x = t in p` *au lieu d'écrire* `let x = ref t in p`*. Quand une variable est finale, on n'écrit pas* `!x` *pour exprimer sa valeur, mais simplement* `x`*. Ainsi, on écrit* `let x = 4 in y := x + 1` *et l'instruction* `let x = 4 in x := 5` *est incorrecte. En C, on indique qu'une variable est finale en faisant précéder son type du mot clé* `const`*.*

1.1.3 La séquence

La *séquence* est une construction qui permet de former une instruction à partir de deux instructions p_1 et p_2. En Java, cette instruction s'écrit {p_1 p_2}. Par convention, l'instruction {p_1 {p_2 { ... p_n} ...}} s'écrit aussi {p_1 p_2 ... p_n}.

Pour exécuter l'instruction {p_1 p_2} dans un état `s`, on exécute d'abord p_1 dans l'état `s`, ce qui produit un état `s'` puis on exécute p_2 dans l'état `s'`.

En Caml, cette instruction s'écrit p_1 ; p_2*. En C, elle s'écrit comme en Java.*