Contrôle d'accès

1 Enoncé

Un bâtiment hautement sécurisé dispose d'un unique accès, sur lequel deux opérations seulement sont possibles :

— Entrer(), qui permet d'entrer dans le bâtiment ;

— Sortir(), qui permet de sortir du bâtiment ;

Les usagers du bâtiment sont représentés par des processus ayant le comportement suivant :

```
répéter
Entrer() ;
activité dans le bâtiment ;
Sortir() ;
activité hors bâtiment ;
sans fin
```

L'accès au bâtiment est en outre soumis à une unique règle, stricte :

> une personne ne doit jamais se trouver seule dans le bâtiment.

Question

Donner le code des deux opérations de manière à assurer la contrainte précédente (et à garantir que dès lors qu'une ou plusieurs opérations sont possibles au regard de la contrainte, au moins l'une de ces opérations est exécutée) en supposant que l'on dispose de sémaphores (généraux), définis par une classe Sémaphore, fournissant les méthodes P(), V(), et un constructeur permettant de fixer la valeur initiale du sémaphore.

2 Solution 1 (standard)

```
Semaphore PE := new Semaphore(0); // événement ''Peut Entrer''
Semaphore PS := new Semaphore(0); // événement ''Peut Sortir''
Semaphore mutex := new Semaphore(1);
integer nbAttE := 0; // nb d'usagers en attente pour entrer
integer nbAttS := 0; // nb d'usagers en attente pour sortir
integer nbDans := 0; // nb d'usagers dans le bâtiment

Entrer() {
    mutex.P();
    si nbDans = 0   ∧   nbAttE + nbAttS = 0 alors
        nbAttE++; mutex.V(); PE.P();
    sinon
        si nbAttS = 1 alors // priorité aux sorties, pour la vivacité
            nbAttS := 0 ; PS.V() ;
        sinon
            si nbAttE = 1 alors
               nbAttE := 0 ; nbDans := 2 ; PE.V();
            sinon
               nbDans++  ;
            fsi ;
        fsi ;
        mutex.V();
    fsi
}

Sortir() {
    mutex.P();
    si nbDans = 2   ∧   nbAttE + nbAttS = 0 alors
        nbAttS++; mutex.V(); PS.P();
    sinon
        si nbAttS = 1 alors // priorité aux sorties, pour la vivacité
            nbAttS := 0 ; nbDans := 0 ; PS.V() ;
        sinon
         // cas nbAttE= 1 impossible : si le batiment est non vide, on peut entrer directement
            nbDans--  ;
        fsi ;
        mutex.V();
    fsi
}
```

3 Solution 2 (astucieuse, marchant... presque)

Des étudiants ont proposé une solution plus élégante (mais qui demande un peu plus de réflexion), exploitant la structure du problème, lequel s'avère symétrique pour les entrants et les sortants : lorsque l'un quelconque attend, un arrivant quelconque peut le débloquer : ce sera un échange ou un passage en bloc, mais dans tous les cas, l'usager en attente sera débloqué par l'arrivant.

```
Semaphore PP := new Semaphore(0); // événement ''Peut Passser''
Semaphore mutex := new Semaphore(1);
integer nbAtt := 0; // nb d'usagers en attente
integer nbDans := 0; // nb d'usagers dans le bâtiment

Entrer() {
    mutex.P();
    si nbDans = 0   ∧   nbAtt = 0 alors
        nbAtt++; mutex.V(); PP.P();
    sinon
        si  nbAtt = 1 alors
          nbAtt := 0 ;
          nbDans := 2 ; //échange ou entrée, il y aura 2 usagers dans le bâtiment
          PP.V() ;
       sinon
          nbDans++  ;
       fsi;
       mutex.V();
    fsi
}

Sortir() {
    mutex.P();
    si nbDans = 2   ∧   nbAtt = 0 alors
        nbAtt++ ; mutex.V(); PP.P();
    sinon
        si nbAtt = 1 alors
           nbAtt := 0 ;
           si nbDans = 2 alors //un sortant attend (un entrant ne peut attendre si nbDans = 2)
               nbDans := 0 ;
            fsi ; // sinon, échange, et alors nbDans ne change pas
            PP.V()
        sinon
            nbDans--  ;
        fsi ;
        mutex.V();
    fsi
}
```

Cette solution souffre cependant d'un gros défaut, dû au fait que mutex doit être libéré avant `PP.P()`. Le scénario suivant met en défaut l'invariant :

- premier entrant, libère mutex et n'exécute pas immédiatement `PP.P()`
- second entrant, passe ($\rightarrow$ exécute `PP.V()`)
- sortie du second entrant, qui passe `PP.P()`
- le premier entrant se bloque

$\rightarrow$ un usager s'est trouvé seul et est ressorti.