Laérian B. - Logiciel 1
Si j'ai fait des fautes la team dites-le moi pour que je corrige

# Exercice 1 - Problème du barbier

## Activités

- 1 barbier
- N clients (N inconnu)

## Code du client

```
boucle
    - entrer dans la salle
    - s'asseoir dans le fauteuil
    - se lever et partir
fin de la voucle
```

## Code du barbier

```
boucle
    - débuter le rasage
    - terminer le rasage
fin de la voucle
```

# Méthodologie de construction du moniteur

| 1) Interface | 2) Conditions d'acceptation | 3)CA avec variables |
|---|---|---|
| entrer en salle | places libres en salle d'attente | $nbPlacesLibres > 0$ |
| s'asseoir | le fauteuil est libre | $fauteuilLibre$ |
| se lever et partir | barbe absente | $\neg barbePresente$ |
| débuter le rasage | client barbu dans le fauteil | $(\neg fauteuilLibre) \wedge barbePresente$ |
| terminer le rasage | aucunes | $true$ |

#### 4) Variables d'état

- `nbPlacesLibres : int := l`
- `fauteuilLibre : bool`
- `barbePresente : bool`

#### 4.5) Invariant

$$(0 \leqslant nbPlacesLibres \leqslant l) \wedge (barbePresente \implies \neg fauteuilLibre)$$

#### 5) Variables conditions

- Salle
- Fauteuil
- ClientBarbu
- Rasé

# Programmation

## Entrer dans la salle

```
si not (nbPlacesLibres > 0) alors
    Salle.wait
finsi
{nbPlacesLibres > 0}
nbPlacesLibres -= 1
{nbPlacesLibres >= 0}
```

## S'asseoir dans le fauteuil

```
si not fauteuilLibre alors
    Fauteuil.wait
finsi
{fauteuilLibre}
fauteuilLibre = false
nbPlacesLibres += 1
barbePresente = true
{not fauteuilLibre          }
    and nbPlacesLibres > 0
    and barbePresente
[ ClientBarbu.signal ]
[ Salle.signal ]
```

## Se lever et partir

```
si not barbePresente alors
    Rasé.wait
finsi
{not barbePresente}
fauteuilLibre = true
{(not barbePresente) and fauteuilLibre}
[ Fauteuil.signal ]
```

## Débuter rasage

```
si not barbePresente alors
    ClientBarbu.wait
finsi
{barbePresente}
```

## Terminer rasage

```
barbePresente = false
{not barbePresente}
[ Rasé.signal ]
```

# Exercice 2 - Lecteurs/Rédacteurs

Un objet qui peut être lu concurrement mais dont l'écriture est exclusive des autres écritures et des lectures Stratégie "priorité aux rédacteurs"

| 1 ) Interface | 2) Conditions d'acceptation | 4) Prédicats |
|---|---|---|
| DE `Débuter_Écriture` | pas d'écriture en cours et pas de lecture en cours | $nl = 0 \ \wedge ne = 0$ |
| TE `Terminer_Écriture` | aucunes | $true$ |
| DL `Débuter_Lecture` | pas d'écriture en cours et pas d'écriture en attente | $ne = 0 \ \wedge$ $neatt = 0$ |
| TL `Terminer_Lecture` | aucunes | $true$ |

#### Bon comportement des activités

$$((DL;TL) + (DE;TE))*$$

## 3) Variables d'état

- `nl : int` (nombre de lecteurs)
- `ne : int` (nombre d'écrivains)
  - (entier pour démontrer ne $\leqslant$ 1)
- `neatt : int`

#### Invariant

$$(0 \leqslant ne \leqslant 1) \land (nl = 0 \lor ne = 0)$$

## 5) Variables conditions

- AccèsLec

- AccèsÉcr

# Programmation

## DL

```
si not (ne = 0 and neatt = 0) alors
    AccèsLec.wait
finsi
{ne = 0}
nl += 1
{ne = 0 and nl > 0}
AccèsLec.signal # Réveil en chaîne
```

## DE

```
si not (nl = 0 and ne = 0) alors
    neatt += 1
    AccèsÉcr.wait
    neatt -= 1
finsi
{nl = 0 and ne = 0}
ne += 1
{nl = 0 and ne = 1}
```

## TL

```
{nl > 0 and ne = 0} # bon comportement + invariant
nl -= 1
{nl >= 0 and ne = 0}
si (nl = 0) alors
    {nl = 0 and ne = 0}
    AccèsÉcr.signal
finsi
```

## TE

```
{nl = 0 and ne = 1} # bon comportement + invariant
ne -= 1
{nl = 0 and ne = 0}
si (neatt > 0) alors
    AccèsÉcr.signal
sinon
    AccèsLec.signal
finsi
```