

# PCO Laboratoire 5

## Modélisation par moniteur de Mesa

Benoît Delay, Eva Ray

December 19, 2023

### Description des fonctionnalités du logiciel

Le programme modélise une boutique de barbier. La boutique est constituée d'un salon dans lequel on trouve une salle d'attente avec un nombre de siège fixes et une chaise de travail sur laquelle un client peut se faire couper les cheveux. Un certain nombre de clients vont tenter de se faire couper les cheveux par le barbier. Le programme est doté d'une interface graphique, qui est animée grâce aux animations présentes dans la classe `PcoSalon` qui représentent les actions du barbier et des clients.

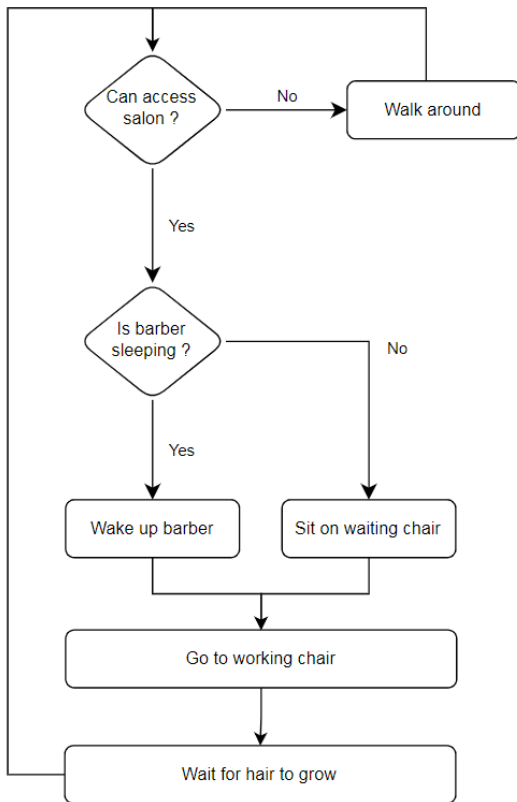
Les fonctionnalités du salon sont assez représentatives d'un salon de coiffure classique. Le salon est géré par un barbier qui s'occupe de couper les cheveux des clients sur la chaise de travail. Lorsqu'il a fini de s'occuper d'un client, le barbier appelle le prochain client parmi ceux qui sont en train d'attendre dans la salle d'attente. Le barbier fait bien attention à appeler les clients dans leur ordre d'arrivée. S'il n'y a plus aucun client dans la salle d'attente, le barbier va faire une petite sieste et ne revient que lorsqu'un nouveau client arrive à la porte de la boutique. Le comportement du barbier est simulé par la méthode `Barber::run()`.

Les clients, quant à eux, ont aussi un comportement prédéfini. Les clients qui souhaitent se faire couper les cheveux ne peuvent entrer dans le salon que si celui-ci a assez de place pour les accueillir, c'est-à-dire s'il y a de la place dans la salle d'attente ou si aucun client n'est présent dans le salon. Dans le second cas, le barbier est en train de faire une petite sieste, comme mentionné ci-dessus. L'arrivée du nouveau client à la porte va réveiller le barbier qui se déplace jusqu'à la chaise de travail, où le nouveau client va le rejoindre. Si le salon est plein, le client va faire un petit tour avant de revenir voir si une place s'est libérée. Si le client accède à la salle d'attente, il va gentiment s'asseoir sur une chaise libre et attend que le barbier l'appelle. Lorsque c'est le tour du client, il se dirige vers la chaise de travail, où le barbier lui coupe les cheveux. Une fois la coupe finie, le client quitte le salon et attend que ses cheveux repoussent avant de revenir se faire faire une nouvelle coupe. Le comportement d'un client est simulé par la méthode `Client::run()`.

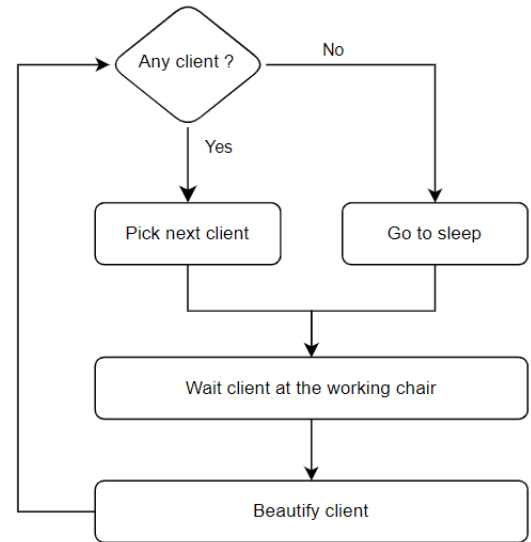
Les comportements décrits ci-dessus sont représentés par les méthodes de la classe `PcoSalon` et correspondent aux machines d'états illustrées ci-dessous.

La terminaison du programme est gérée grâce aux méthodes `PcoSalon::endService()` et `PcoSalon::isInService()`. La fin du programme est initiée dès qu'une entrée utilisateur est déclenchée dans le terminal. Le salon est alors fermé. Les clients qui ne se trouvent pas actuellement dans le salon rentrent chez eux. Le barbier finit de s'occuper de tous les clients qui sont encore dans le salon mais n'en accepte pas de nouveau. Une fois qu'il a terminé son travail, ferme le salon et termine sa journée.

Le logiciel simule le comportement du barbier et des clients par des threads différents, il est donc multi-threadé. Par conséquent, il doit assurer une bonne gestion de la concurrence pour les ressources partagées entre plusieurs threads. Les méthodes du salon sont ainsi implémentées à l'aide d'un moniteur de Mesa, qui permet de gérer la concurrence et les animations.



(a) Machine d'état du client



(b) Machine d'état du barbier

## Choix d'implémentation

### Variables de condition

La synchronisation entre les différents threads du programme se fait via des variables de condition qui sont des attributs protégés de la classe `PcoSalon`.

- **canGoForHaircut**: La variable de condition `canGoForHaircut` représente l'attente d'un client sur une des chaises de la salle d'attente avant de pouvoir aller se faire couper les cheveux. Les threads des clients attendent sur cette variable de condition dans la méthode `PcoSalon::accessSalon` et le barbier réveille les threads des clients en attente depuis la méthode `PcoSalon::pickNextClient`.
- **waitClient**: La variable de condition `waitClient` représente le fait que le barbier attende qu'un client se déplace de la salle d'attente à la chaise de travail. Le thread du barbier attend sur cette variable de condition dans la méthode `PcoSalon::waitClientAtChair` et le client signifie qu'il a atteint la chaise de travail dans la méthode `PcoSalon::goForHaircut`.
- **doneBeautified**: La variable de condition `doneBeautified` représente le fait qu'un client attende que sa coupe de cheveux se termine. Le thread du client attend sur cette variable de condition dans la méthode `PcoSalon::goForHaircut` et le barbier signifie au client qu'il a fini la coupe dans la méthode `PcoSalon::beautifyClient`.
- **barberSleeping**: La variable de condition `barberSleeping` représente le sommeil du barbier. Le thread du barbier attend sur cette variable de condition dans la méthode `PcoSalon::goToSleep` pour représenter la fait que le barbier s'endort. Le client signifie que le barbier doit se réveiller dans la méthode `PcoSalon::accessSalon`.

## Gestion des chaises de la salle d'attente

Pour gérer les chaises de la salle d'attente, nous avons ajouté un attribut `chairs` dans la classe `PcoSalon`. C'est simplement un vecteur de booléens qui a pour taille le nombre de chaises de la salle d'attente. Si le booléen contenu à l'indice `i` vaut `false` cela signifie que la chaise numéro `i` est libre, s'il vaut `true` cela signifie que la chaise est occupée. Dans la méthode `PcoSalon::accessSalon`, on assigne une chaise libre au client qui doit aller s'asseoir dans la salle d'attente, elle devient alors occupée. Lorsque le client se déplace sur la chaise de travail, on libère la chaise sur laquelle il était assis.

## Gestion de la file FIFO

Une des fonctionnalités du programme est d'assurer le fait que le barbier s'occupe des clients dans leur ordre d'arrivée dans le salon. Pour ce faire, nous avons implémenter une file FIFO avec une méthode de ticketing. L'idée est simple; chaque client qui arrive dans la salle d'attente prend un ticket à la machine à tickets, représentée par l'attribut `ticketMachine` de la classe `PcoSalon`. Il attend ensuite sur la variable de condition `canGoForHaircut`. Lorsqu'il est prêt, le barbier va `notifyALL` sur cette variable, ce qui va avoir pour conséquence de réveiller tous les threads en attente. A ce moment, le client réveillé va comparer son numéro de ticket avec celui stocké dans l'attribut `nextClient` de la classe `PcoSalon` qui représente le numéro du prochain client à pouvoir aller se faire couper les cheveux. Si les numéros coïncident, le client peut aller se faire couper les cheveux et l'attribut `nextClient` est incrémenté, sinon, il se remet en attente jusqu'à ce que ce soit son tour.

## Gestion de l'interruption du programme

La terminaison du programme est amorcée lorsqu'une entrée utilisateur est déclenchée dans le terminal. Elle est gérée grâce aux méthodes `PcoSalon::endService()` et `PcoSalon::isInService()`. Nous avons ajouté un attribut booléen `isOpen` dans la classe `PcoSalon` qui vaut `true` lorsque le salon est ouvert et `false` sinon. La méthode `PcoSalon::isInService()` renvoie simplement cet attribut. Elle nous permet de connaître l'état d'ouverture du salon depuis l'extérieur de la classe. Dans la méthode `PcoSalon::endService()` on change la valeur de `isOpen` à `false` pour informer de la fermeture du salon. On s'assure aussi de gérer le cas dans lequel la fermeture du salon est déclenchée lorsque le barbier est en train de dormir. Dans ce cas, on réveille le barbier pour que son thread puisse se finir et ne pas rester en attente indéfiniment sur la variable de condition `barberSleeping`. On s'assure ensuite dans la méthode `Barber::run` du barbier qu'il s'occupe des clients qui sont étaient déjà présents dans la salon avant la fermeture avant que son thread ne s'arrête pour de bon. Cela est mis en place grâce aux méthodes `PcoSalon::isInService()` et `PcoSalon::getNbClient()`. Les clients attendent encore que leur cheveux repoussent avant de se rendre compte que le salon est fermé et rentrer chez eux pour de bon.

## Tests effectués

Le résultat des tests ci-dessus a notamment pu être vérifié car nous logons dans l'interface graphique des informations sur la plupart des actions entreprises par le barbier et les clients. En particulier, en plus de décrire leur comportement, un message s'affiche lorsque les threads des clients ou du barbier se finissent.

### Test: Fonctionnalités du salon

#### Paramètres par défaut

Les tests suivants ont été effectués avec les paramètres par défaut, c'est-à-dire avec 10 clients et 2 sièges dans la salle d'attente

- Lorsqu'un client souhaite accéder au salon et que le barbier est en train de dormir, le barbier se réveille et se déplace vers la chaise de travail. Le client attend sur le pas de la porte que le barbier soit arrivé à la chaise et l'y rejoint lorsque c'est le cas. C'est le comportement attendu.
- Lorsqu'un client souhaite accéder au salon alors que le barbier est déjà réveillé et qu'il y a de la place dans la salle d'attente, le client va s'asseoir sur une chaise libre de la salle d'attente et attend son tour. C'est le comportement attendu.

- Lorsqu'un client souhaite accéder au salon alors que le barbier est déjà réveillé et qu'il n'y a pas de place dans la salle d'attente, le client ne peut pas accéder au salon et va faire un tour avant de rententer sa chance. C'est le comportement attendu.
- Lorsqu'il n'y a plus aucun client dans le salon, le barbier va faire une sieste. C'est le comportement attendu.
- Lorsqu'on lance le programme, le barbier commence toujours par aller faire une sieste. C'est le comportement attendu.
- Lorsque le barbier appelle le prochain client, il attend que le client arrive sur la chaise de travail avant de commencer la coupe. C'est le comportement attendu.
- Lorsque le barbier appelle le prochain client, c'est le client qui est arrivé le premier dans le salon qui se dirige vers la chaise de travail. C'est le comportement attendu.
- Lorsque le barbier est en train de faire une coupe, aucun autre client ne peut se diriger vers la chaise de travail tant qu'il n'a pas fini. C'est le comportement attendu.
- Lorsque le client a une nouvelle coupe, il quitte le salon et attend que ses cheveux repoussent avant d'essayer d'accéder à nouveau au salon.

### Variation des paramètres

Les tests suivants ont été réalisés en modifiant les paramètres par défaut pour tester la robustesse du programme.

- Lorsqu'il n'y a aucun siège dans la salle d'attente, le client va toujours devoir réveiller le barbier et attendre à la porte. Lorsque le barbier a fini la coupe du client, il repart tout de suite dormir avant de se faire rapidement réveiller par le prochain client qui attend à la porte. C'est le comportement attendu.
- Lorsqu'il n'y a qu'un seul client et 2 (ou un autre nombre non nul) de sièges dans la salle d'attente, le client réveille le barbier et attend à la porte. Le client ne va pas attendre dans la salle d'attente mais reste à la porte. Entre chaque coupe de l'unique client, le barbier va dormir. C'est le comportement attendu.
- Lorsqu'il y a 2 clients et 2 sièges dans la salle d'attente (c'est-à-dire autant de clients que de chaises), le programme se déroule conformément au cahier des charges. C'est le comportement attendu.
- Lorsqu'il n'y a aucun client, le barbier va dormir et ne se réveille jamais. C'est le comportement attendu.
- Lorsqu'il y a plus de 2 chaises dans la salle d'attente et 10 clients, les clients s'asseyent toujours sur une chaise libre et sont toujours appelés par le barbier dans leur ordre d'arrivée. C'est le comportement attendu.
- Lorsqu'on met 20 clients et 2 chaises dans la salle d'attente, le programme se déroule conformément au cahier des charges. C'est le comportement attendu.

### Test: Terminaison du programme

La fin des threads peut être vérifiée car un message est logé dans l'interface graphique dans les cases réservées aux clients et au barbier. Un message signifiant la terminaison du programme apparaît

- Lorsque le barbier est en train de travailler et une entrée utilisateur est déclenchée dans la console, plus aucun nouveau client ne rentre dans le salon, le barbier finit de s'occuper du client qui est actuellement sur la chaise de travail et de ceux qui sont en train d'attendre dans la salle d'attente avant que son thread se finisse. Le thread des clients qui n'étaient pas dans le salon au moment de la terminaison du programme se finit dès que les clients constatent que le salon est fermé. Les clients qui étaient dans le salon au moment de la fermeture quittent le salon lorsque leur coupe est terminée et attendent encore une fois que leur cheveux repoussent avant que leur thread s'arrête lorsqu'ils constatent que le salon est fermé. Le message "le programme est terminé" apparaît dans la console. C'est le comportement attendu.
- Lorsque le barbier est en train de dormir et une entrée utilisateur est déclenchée dans la console, il se réveille pour que son thread puisse se terminer. Le message "le programme est terminé" apparaît dans la console. C'est le comportement attendu.

## Conclusion

Nous avons implémenté un programme multi-threadé qui simule un salon de coiffure conformément au cahier des charges. Nous avons implémenté une gestion de la synchronisation grâce à des variables de condition d'un moniteur de Mesa qui semblent faire leur travail convenablement. L'accès des clients au salon, l'attente dans la salle d'attente, la prise en charge des clients, la coupe, la repouss des cheveux et le sommeil du barbier sont gérés correctement. L'initiation de la fin de programme par le déclenchement d'une entrée utilisateur dans le terminal fonctionne comme espéré et permet de mettre fin aux threads de la simulation correctement.