Labo 3 Programmation Concurrente.

# Introduction

Le but de ce laboratoire est de simuler un aéroport en utilisant un thread pour chaque avion. Ceci doit être fait de deux manières différentes, en utilisant les blocking queu et en utilisant un système de tampon avec des blocks synchronized. Une fois les deux implémentations faites nous avons dû tester leurs temps d’exécutions.

# Mode d’emploi

Afin de pouvoir lancer le programme en modifiant les paramètres il est nécessaire de les passer par la ligne de commande (dans eclipse : Run / Run configurations / arguments / Program arguments). Les arguments sont dans l’ordre suivant : nombres d’avions, nombre de pistes d’atterrissages, nombre de pistes de départ, nombre de places sur le tarmac et usage de blocking queu (true ou false). Donc si nous rentrons « 20 2 2 4 true » notre programme utilisera les blocking queu, aura 20 avions, 2 pistes de décollage, 2 pistes d’atterrissages et 4 places sur le tarmac.

# Explications du code

Dans AirportFrame.java nous avons juste ajouté quelques fonctions permettant a l’affichage de se rafraichir en fonction du code de nos threads.

Dans Main.java on récupère la paramètres de la ligne de commande et en fonction de si on doit utiliser blocking queu ou pas on instancie les avions avec 2 constructeurs différents (un ou on passe les blocking queu et l’autres de simples listes) ensuite on start nos threads et on implémente ce que doivent faire les boutons stop et start (on utilise un sémaphore pour gérer cela)

Dans Avion.java on a 2 constructeur, un utilisant les blocking queu et l’autre de simples listes. Ensuite dans le run en fonction on a un if permettant de choisir entre blocking queu ou simple liste. Finalement les autres fonctions implémentent le la logique de chaque avion. Chose à noter la fonction isPaused() nous permet de gérer le bouton stop en utilisant un sémaphore.

# Test