# Théorie et pratique de la concurrence – Master 1 Informatique
# TP 7 : Sémaphores en Java (suite)

**Exercice** 1: *La traversée des babouins*

Il y a un canyon quelque part dans le parc national Kruger, Afrique du Sud, et une seule corde qui traverse le canyon. Les babouins peuvent traverser le canyon sur la corde, mais si deux babouins allant dans des directions opposées se rencontrent au milieu, ils vont se battre et tomber. En outre, la corde est assez forte pour tenir seulement cinq babouins. Programmez une solution à ce problème telle que :

— une fois qu'un babouin a commencé à traverser, il est garanti d'arriver de l'autre côté sans croiser un babouin dans l'autre sens.
— il n'y a jamais plus de 5 babouins sur la corde.
— un flux continu de babouins allant dans un sens ne devrait pas empêcher des babouins de faire la traversée dans l'autre sens indéfiniment (pas de famine).

**Exercice** 2: *Chimie amusante*

Dans cet exercice, il existe deux types de threads, **Oxygène** et **Hydrogène**. Pour assembler ces threads dans des molécules d'eau ($H_2O$), on va créer une *barrière* qui est attendue par chaque thread tant qu'il n'y a pas suffisamment de threads pour créer une molecule d'eau (composée de deux threads **Hydrogène** et un thread **Oxygène**). Juste avant qu'un thread passe la barrière, il va afficher un message. Il faut garantir que les messages des threads qui forment une molecule sont affichés avant les messages des threads d'une autre molecule. Plus précisément, le programme devra respecter les conditions suivantes :

— si un thread **Oxygène** arrive à la barrière quand aucun thread **Hydrogène** n'est présent, il doit attendre deux threads **Hydrogène**,
— si un thread **Hydrogène** arrive à la barrière quand aucun autre thread n'est présent, il doit attendre un thread **Oxygène** et un autre thread **Hydrogène**.

Les threads doivent passer la barrière dans des ensembles complets ; ainsi, en examinant la séquence de messages affichés et en les divisant en groupe de trois, chaque groupe doit contenir un thread **Oxygène** et deux threads **Hydrogène**.

On souhaite developper un programme concurrent simulant le comportement décrit ci-dessus :

1. Définir la méthode `waitBarrier()` qui bloque jusqu'au moment où trois threads rentrent dans cette méthode *(on ne se préoccupe pas de la nature des threads)*.
2. Écrire un programme concurrent qui modélise le comportement des atomes **Oxygène** et **Hydrogène** décrit ci-dessus.
3. Modifier votre programme pour traiter le cas de l'eau oxygénée $H_2O_2$ (deux atomes **Oxygène** et deux atomes **Hydrogène**).
4. Modifier votre programme pour traiter le cas de l'eau **et** du méthane $CH_4$ (on ajoute un troisième type d'atome, l'atome **Carbone**). La barrière retiendra les threads jusqu'à former une molécule d'eau ou de méthane :
   — si un thread **Oxygène** arrive à la barrière quand aucun thread **Hydrogène** n'est présent, il doit attendre deux threads **Hydrogène**,
   — si un thread **Carbone** arrive à la barrière quand aucun autre thread n'est présent, il doit attendre quatres threads **Hydrogène**,
   — si un thread **Hydrogène** arrive à la barrière quand aucun autre thread n'est présent, il doit attendre un thread **Oxygène** et un autre thread **Hydrogène**, ou un thread **Carbone** et un trois autres thread **Hydrogène** (selon ceux qui arrivent en premier).

   Les atomes de la molécule formée quittent la barrière laissant éventuellement un thread **Carbone** ou **Oxygène**, qui continue donc à attendre.