

## **Exercice I.1 Question d'ordonnancement**

```
class Test extends Thread {
    String msg;

public Test(String s) {
    msg = s;
}

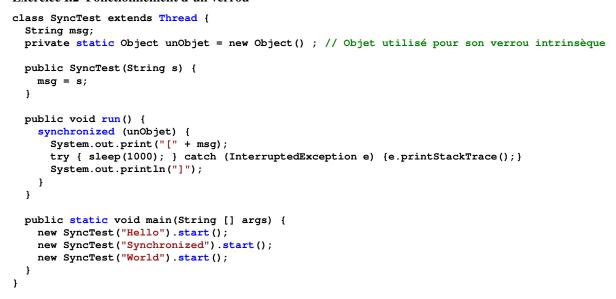
public void run() {
    try { sleep(1000); } catch (InterruptedException e) {e.printStackTrace();}
    System.out.print(msg + "_");
}

public static void main(String [] args) {
    new Test("Hello").start();
    new Test("World").start();
}
```

Question 1. Que va afficher ce programme lors de son exécution?

Question 2. Comment ajouter un retour à la ligne à la fin?

## Exercice I.2 Fonctionnement d'un verrou



Question 1. Donner une sortie écran possible de ce programme.

Question 2. Donner une sortie écran possible de ce programme lorsque l'on supprime le mot-clef static de la définition de l'objet unObjet.

Exercice I.3 Evaluation de  $\pi/4$  par la méthode de Monte-Carlo La méthode dite « de Monte-Carlo » consiste à calculer une valeur numérique en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Considérons par exemple un point M de coordonnées (x,y) avec 0 < x < 1 et 0 < y < 1 tirées aléatoirement. Le point M appartient au disque de centre (0,0) de rayon 1 si, et seulement si,  $x^2 + y^2 \leqslant 1$ . La probabilité que le point M appartienne au disque est donc de  $\pi/4$ . Si l'on effectue un grand nombre de tirages de points, le rapport du nombre de points dans le disque au nombre de tirages total fournit une approximation du nombre  $\pi/4$ . Cette idée conduit au code du programme Java PiSurQuatre de la figure 1 qui évalue et affiche une estimation de la valeur de  $\pi/4$ .

Pour accélérer ce calcul sur une machine multi-coeur, on souhaite le paralléliser sur 10 threads. Écrire un programme Java qui crée les 10 threads et leur affecte une part équitable de la tâche globale à effectuer; attend que tous les threads aient terminé leur tâche; puis affiche la valeur approchée de  $\pi/4$ . Ce programme ne doit définir qu'une seule classe de threads (ou de runnables).









(P)

2017