7. Une approche des *threads*

Le but des exercices qui suivent est de présenter la notion de *thread* à travers une application très modeste qui affiche constamment la date et l'heure courantes :

Exercice 7.1. Écrivez la première version de ce programme. Il comporte un cadre (classe `Version1`, sous-classe de `Frame`) auquel on a ajouté deux panneaux (classe `Panel`) :

- au-dessus (placement avec `BorderLayout.CENTER`), un panneau dans lequel on se propose d'afficher la date et l'heure par des appels de la méthode `drawString`, voir ci-après,
- au-dessous (placement avec `BorderLayout.SOUTH`), un panneau dans lequel on aura mis une boite à cocher (classe `Checkbox`) représentée par une variable d'instance nommée `court`. Lorsque cette case est cochée, seule l'heure doit être affichée.

En fait, le panneau supérieur est instance d'une classe `Afficheur`, une sous-classe de `Panel` munie d'une méthode `paint` ainsi conçue :

```
public void paint(Graphics g) {
    Date d = new Date();
    String s = court.getState() ? sansDate.format(d) : avecDate.format(d);
    g.drawString(s, 20, 35);
}
```

où `sansDate` et `avecDate` sont des variables d'instance ayant pour valeurs des objets de la classe `DateFormat` obtenus par des appels de `getDateTimeInstance` avec des arguments idoines.

La méthode principale de ce programme (et de ceux des questions suivantes, en remplaçant `Version1` par `Version2`, `Version3`, etc.) est

```
public static void main(String[] args) {
    new Version1();
}
```

Constatez que ce programme n'est pas très dynamique : il faut qu'un événement extérieur, comme le masquage par une autre fenêtre, provoque la réfection de la fenêtre pour que la date soit mise à jour.

Exercice 7.2. Écrivez les instructions formant la méthode `paint` (obtention, mise en forme et affichage de la date et l'heure) à l'intérieur d'une boucle infinie (genre « `for(;;)` ») qui les répète sans cesse. Le résultat est-il très satisfaisant ?

Exercice 7.3. Dans la version précédente les écritures se superposent et au bout d'un moment on ne peut plus lire l'heure. Pour corriger cela, ajoutez dans la boucle l'effacement de l'intérieur du cadre (méthode `clearRect`). Est-ce vraiment mieux ?

En particulier, est-il facile de changer le format de l'affichage ?

Exercice 7.4. Dans la version précédente nous avons obtenu un affichage qui permet de lire, dans un scintillement plus ou moins désagréable, une date et une heure constamment correctes. Mais il faut savoir que ce programme est une *très mauvaise* solution du problème posé car, pour un travail minime, afficher une information qui ne change que toutes les secondes, il consomme un temps de calcul considérable.

En fait *tout* le temps de calcul que l'application réussit à se faire allouer est employé à faire tourner la boucle infinie, qui ne rend jamais la main pour que d'autres tâches puissent être effectuées, par exemple la détection et le traitement des changements d'état de la case à cocher.

Pour rendre votre programme beaucoup moins consommateur de temps de calcul insérez dans la boucle de la méthode `paint` un appel de la méthode `Thread.sleep` (voyez la documentation en ligne) de manière à « endormir » le *thread* qui exécute votre programme pendant une seconde à chaque tour de la boucle.

Exercice 7.5. On s'approche de la bonne solution, mais on n'y est pas encore. Comme vous pouvez le constater, pendant que le thread est endormi, « personne ne garde la boutique » : les différents événements qui peuvent survenir, comme les clics dans la boîte à cocher, ne sont pas traités ; la date change, mais les autres composants sont bloqués.

Nouvelle version (la bonne, enfin) : écrivez un programme organisé en deux *threads* : le premier est identique à celui de la version 1, le second est réalisé par une instance de la classe `Agitateur`, sous-classe de `Thread`, dont la méthode `run` consiste en une boucle qui répète inlassablement :

- dormir pendant une seconde,
- envoyer au panneau qui affiche la date le « message » `repaint` (ce qui provoquera l'appel de `paint`).

C'est le constructeur du cadre qui crée et lance le deuxième *thread*[1].

Exercice 7.6. Pour terminer, écrivez encore une version pour constater qu'il n'est pas nécessaire d'avoir deux objets pour avoir deux *threads* : le cadre peut être son propre « agitateur » ; il suffit pour cela qu'il implémente l'interface `Runnable`.

• • •

1. En fait, le constructeur du cadre lance *deux* threads : l'appel `setVisible(true)` crée et lance le thread qui fait vivre l'interface graphique ; l'appel du constructeur de `Agitateur` puis de la méthode `start` de l'objet construit créent et lancent le thread qui rafraîchit l'affichage chaque seconde. Notez qu'il y avait un troisième thread dans cette histoire, celui dans lequel a été appelée la méthode `main` ; mais celui-là aura vécu très peu de temps, car il exécute une méthode, `main`, comportant une unique instruction, l'appel du constructeur du cadre.