Maitrise d'Informatique	Programmation Objet et	Université Pierre et Marie Curie
Module POD	Distribution	Année 1998–1999
	Documents autorisés	*Durée 3 heures*

Examen du 14 septembre 1999

1 Classes de listes en Java

Le but de cet exercice est d'implanter des listes sous deux formes, l'une classique, l'autre contenant l'information de taille des listes. Il y a plusieurs organisations possibles. Vous justifierez vos choix d'organisation.

1. Écrire une classe abstraite `list` en Java pour la manipulation de listes contenant les méthodes : `add`, `del`, `head`, `tail`, `size` et `append`, `equals` et `toString`. Le choix des signatures des méthodes est laissé à votre entière appréciation. Vous pouvez construire des listes contenant des `Object` ou des `int`.

2. Écrire la ou les sous-classes concrètes de `list` pour l'implantation de cette classe abstraite.

3. Écrire en Java un programme principal qui contruit des listes d'intervalles d'entiers (bornés par 2 entiers), qui concatène 2 intervalles pour ensuite afficher celui-ci ainsi que sa taille.

4. Définir la ou les sous-classes concrètes de `list` pour des listes qui contiennent leur taille en tant que nouveau champ. Indiquez, si c'est le cas, les méthodes à redéfinir et à surcharger par rapport à la version précédente.

5. Faut-il modifier le programme principal de la question 1.3 pour utiliser des listes avec taille? Si oui, indiquez les modifications.

6. Auriez-vous utilisé la même organisation des classes en O'Caml? Indiquez les changements à apporter à votre organisation pour adapter cette hiériarchie en O'Caml.

2 Applets Java pour O'Caml

Le but de cet exercice est de visualiser les ordres graphiques, de la bibliothèque `Graphics`, d'une application O'Caml dans une applet Java, et de récupérer les évènements produits dans cette applet dans l'application O'Caml. On ne s'intéressera qu'au tracé de lignes et au clic souris. La communication entre O'Caml et Java passe par des sockets en TCP/IP

L'application O'Caml ouvre un service sur le port 1409, puis lance l'applet Java par une des commandes suivantes `Sys.command "appletviewer monapplet.html &"` ou `Sys.command "netscape monapplet.html &"`. Cette applet va ouvrir une socket sur le service de l'application O'Caml. Une fois cette connexion réalisée les deux programmes vont communiquer en respectant le protocole texte décrit plus loin. L'applet Java envoie à l'application O'Caml les positions des clics souris. L'application O'Caml envoie à l'applet Java des ordres graphiques.
On définit le protocole suivant pour les échanges entre les deux applications :

- ouverture d'une fenêtre de taille w*h :

```
OPEN
w
h
```

- effacement d'une fenêtre : `CLEAR`

- fermeture d'une fenêtre : `CLOSE`

- positionnement du point courant en (x,y) :

```
MOVETO
x
y
```

- tracé d'une ligne à partir du point courant et jusqu'au point (x,y) :

```
LINETO
x
y
```

- clic souris au point (x,y) :

```
CLIC
x
y
```

Ce protocole texte simple permet d'indiquer une demande d'ouverture, d'effacement, de fermeture d'une fenêtre, de position du point courant, du tracé d'une ligne à partie du point courant et l'envoi d'un clic souris.

1. Écrire une sous-classe d'`Applet` qui ouvre une socket sur le service de port 1409 sur la machine locale (127.0.01). On prévoit deux champs `in` et `out` pour les canaux de la socket. La méthode `run` est en écoute sur `in`. Elle déclenche la méthode `treat` quand un texte arrive de la socket. Pour le moment la méthode `treat` ne fait rien.
2. Écrire le corps de la méthode `treat` pour pouvoir redimensionner la taille de la fenêtre graphique, l'effacer, la fermer, positionner le point courant, et dessiner une ligne à partir du point courant selon le protocole indiqué.
3. Écrire une sous-classe de `MouseAdaptor` en redéfinissant la méthode `mousePressed` qui envoie le texte suivant :

   ```
   CLIC
   x
   y
   ```

   au serveur O'Caml. Enregistrer une instance de cette classe sur l'applet.
4. Écrire une fonction O'Caml `init_server` qui ouvre un service sur le port 1409, lance l'applet Java et se met en écoute d'une connexion. Quand celle-ci survient, elle crée deux canaux d'entrées/sorties sur le client.
5. Écrire en O'Caml une classe paramétrée `'a queue` qui implante les files d'attente en définissant les méthodes `is_empty : unit -> bool`, `enter : 'a -> unit` et `exit : unit -> 'a`.
6. Écrire une fonction d'écoute bloquante sur le canal d'entrée. Quand un message survient, cette fonction ajoute cet élément à une queue des événements. Attention cette queue est en exclusion mutuelle avec les autres threads du programme.
7. Réécrire les fonctions `open_graph`, `close_graph`, `clear_graph`, `moveto` et `lineto` pour qu'à leur appel, soit envoyé sur le canal de sortie le texte correspondant.
8. Réécrire la fonction `wait_next_event` qui vérifie si un élément est dans la queue des événements, si oui retourne une valeur de type `status` adéquate, sinon se met en attente d'arrivée d'un événements dans cette queue. Attention cette queue est toujours en exclusion mutuelle.
9. Écrire l'application principale O'Caml qui consiste à tracer des lignes entre deux clics souris. À la connexion de l'applet Java, elle lance une thread en écoute bloquante du canal d'entrée. Cette thread ajoute un nouvel état à la queue des événements quand elle reçoit l'information d'un clic souris. Ensuite l'application attend un événement souris et envoie les ordres graphiques correspondants.
10. Pour que l'applet Java puisse redessiner, suite à une iconification, les différentes tracés effectués depuis le dernier `open_graph` ou `clear_graph`, il est nécessaire de conserver un historique des ordres graphiques envoyés. Implanter un tel mécanisme d'historique et modifier la méthode `paint` en conséquence.
11. Réfléchissez à l'intégration complète de la bibliothèque Graphics, en particulier les bitmaps qui sont de grosses données, et les polices de caractères. Proposez une solution sans l'implanter.