Ingénierie de l'IHM

durée indicative : **2h**
documents autorisés : **1 feuille A4 recto-verso manuscrite de votre main**

Commencez par lire l'intégralité du sujet en détail.
Le barème est **indicatif**. La qualité de la **présentation**, de l'**expression**, ainsi que la **précision** et la **concision** seront prises en compte dans la notation de **manière significative**. Si le sujet présente des ambiguïtés, précisez vos choix. Lorsqu'il est demandé de produire du code, ne vous focalisez pas sur la correction de la syntaxe.

Généralités (7 points)

Question 1
Il existe deux sortes de boîtes à outils de construction d'interface en Java : celles qui ont une approche minimaliste et celles qui suivent l'approche maximaliste.
a) Expliquez en une phrase ce qui les distingue.
b) Donnez les avantages et les inconvénients de ces deux approches.
c) Donnez le nom d'une boîte à outils de chaque sorte.

Question 2
Situez dans le temps (donnez la décennie de) :
• l'invention de la souris ;
• l'invention des interfaces graphiques utilisant la métaphore du bureau.

Question 3
a) Comment distinguer ce qui fait partie du noyau fonctionnel de ce qui fait partie de l'interface d'une application ?
b) Expliquez pourquoi il faut séparer le code du noyau fonctionnel de celui de l'interface.
c) Expliquez pourquoi le mécanisme des *callbacks* (fonctions de rappel) est important dans cette séparation.
d) Selon Fekete [1996], le noyau fonctionnel doit fournir les trois services suivants : notification, prévention des erreurs, et annulation. Dans une application de client mail, donnez un exemple d'interaction rendue possible pour chacun de ces services.

Programmation par événements (7 points)

Question 4
Soient les deux interactions suivantes d'une application de dessin qui commencent toutes deux sur l'enfoncement du bouton gauche de la souris (↓), se poursuivent lors de son déplacement, bouton enfoncé (D pour *drag*), et se terminent sur le relâchement du bouton (↑) :
• la sélection d'objets par une zone rectangulaire (gérée par les fonctions données : `start_selection(start_point), update_selection(point), end_selection(end_point)`) ;
• le déplacement de la zone de travail (petite main gérée par : `start_drag(), drag(dx, dy), end_drag()`).

On souhaite combiner ces deux techniques de la manière suivante : lorsque le bouton gauche de la souris est enfoncé, c'est l'interaction de déplacement qui s'enclenche si la touche espace est enfoncée, sinon c'est l'interaction de sélection.
La touche espace envoie les événements P (*press*) pour l'enfoncement et R (*release*) pour le relâchement. Elle peut être enfoncée ou relâchée à tout moment pendant qu'on interagit.

a) **Dessinez la machine à états qui spécifie cette interaction.**

On supposera que les évènements souris (↓, D et ↑) ont une méthode `getPoint()` qui retourne un `Point2D` avec la position du curseur et que les évènements clavier (P et R) ont une méthode `getCode()` qui retourne un `KeyCode` que l'ont peut comparer à la constante `KeyCode.SPACE` pour savoir si c'est bien la touche *espace* qui a été actionnée.

b) **Réalisez en java cette machine à états**.

Pour cela, vous est donnée une classe `Surface` qui utilisera votre classe `Interaction`, et qui fournit les différentes fonctions que cette dernière doit appeler.
C'est la classe `Interaction` que vous devez réaliser.

```
package exam.ui;

import javafx.scene.canvas.Canvas;
import javafx.geometry.Point2D;

public class Surface extends Canvas {
      Interaction interaction;
      public Canvas() {
            interaction = new Interaction(this);
      }
      public void start_selection(Point2D start_point) { ... }
      public void update_selection(Point2D point) { ... }
      public void end_selection(Point2D end_point) { ... }

      public void start_drag() { ... }
      public void drag(float dx, float dy) { ... }
      public void end_drag() { ... }
}
```

Votre classe `Interaction` pourra commencer ainsi :

```
package exam.ui;

public class Interaction ... {
      protected Surface surface;
      public Interaction(Surface surface) {
            this.surface = surface;
            ...
      }
      ...
}
```

On rappelle qu'on peut s'abonner à une instance de `Canvas` via la méthode `addEventHandler` qui prend en premier paramètre le type d'évènement attendu, et en second paramètre une instance spécialisant l'interface fournie par `javafx.event` donnée ci-dessous :

```
public interface EventHandler<T> {
      public void handle(T e);
}
```

Les évènements qui nous intéressent sont ici de cinq types répartis dans deux classes :

- `MouseEvent` pour la souris pour les types `MouseEvent.MOUSE_PRESSED` (↓), `MouseEvent.MOUSE_RELEASED` (↑) et `MouseEvent.MOUSE_DRAGGED` (D).
- `KeyEvent` pour le clavier pour les types `KeyEvent.KEY_PRESSED` (P) et `KeyEvent.KEY_RELEASED` (R).

`InputEvent` est une superclasse de `MouseEvent` et `KeyEvent`. Pour simplifier, on supposera qu'elle dispose de la méthode `getPoint()` qui retourne un `Point2D` avec la position du curseur et de la méthode `getCode()` qui retourne un `KeyCode` (en réalité il faudrait transtyper les `InputEvent` en `MouseEvent` ou en `KeyEvent` suivant les cas pour accéder à ces méthodes).

Pour tout évènement `e`, on peut accéder à son type grâce à `e.getEventType().getName()` qui renvoie une `String` que l'on peut comparer à `"MOUSE_PRESSED"`, `"MOUSE_RELEASED"`, `"MOUSE_DRAGGED"`, `"KEY_PRESSED"`, `"KEY_RELEASED"`.

Composants interactifs (6 points)

Question 5

Les figures ci-dessous donnent la fenêtre de l'application CyberDuck (transfert FTP) sous Mac OS X avant et après un redimensionnement. Donnez l'arbre des widgets que vous utiliseriez pour coder cette interface avec JavaFX.
Certains aspects ne peuvent être rendus simplement avec JavaFX : les boutons collés dans la barre de boutons pourront être considérés comme n'étant pas collés.
Ne détaillez pas le contenu de la liste des serveurs.