Techniques des Logiciels Interactifs

durée : **1h30**
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 (6 points)

Question 1

a) Comment distinguer ce qui fait partie du noyau fonctionnel de ce qui fait partie de l'interface d'une application ?
b) Selon Fekete [1996], le noyau fonctionnel doit fournir les trois services suivants : notification, prévention des erreurs, et annulation. Expliquez pourquoi en donnant un exemple d'interaction impossible ou difficile à réaliser pour chacun de ces services lorsqu'il est absent.

Question 2

Dans le projet sur lequel votre équipe travaille apparaissent les codes suivants :

```
package fc;
import ui.View;

class Model {
   View view;
   public Model(View v) { view = v; }
   public void setValue(String value) { v.valueChange(value); }
   ...
}
```

```
package ui;
import fc.Model;

class View {
   JLabel label;
   public View() { label = new JLabel(); }
   public void valueChange(String value) { label.setText(value); }

   public static void main(String args[]) {
      view = new View();
      model = new Model(view);
   }
}
```

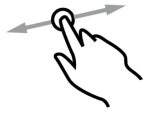
a) Citez les trois grands principes sur la séparation de l'interface utilisateur et indiquez le(s)quel(s) est(sont) respecté(s) ou non dans cet extrait de code.
b) Modifiez les deux classes (en ajoutant si nécessaire des éléments) pour que les trois principes soient respectés et que quand la méthode `setValue` du modèle `model` est appelée, le texte de l'étiquette `label` change bien pour refléter la nouvelle valeur.

Programmation par événements (9 points)

Question 3

Vous devez réaliser l'interface graphique d'un logiciel qui permet de manipuler des photos sur une surface interactive qui permet d'utiliser les doigts comme pointeurs.
Ceux-ci génèrent trois types d'événements : ↓ lorsqu'un doigt entre en contact avec la surface ; D lorsqu'il se déplace au contact de la surface ; et enfin ↑ lorsqu'il quitte cette dernière.
Ces événements contiennent la position (`e.getPoint()`) où ils ont eu lieu ainsi qu'un identifiant du doigt utilisé (`e.getFingerId()`).
On veut réaliser l'interaction suivante :

- avec 1 doigt, la photo est déplacée (déplacement géré par les fonctions données suivantes : `start_drag(Point start), drag(int dx, int dy), end_drag()`) ;



- avec 2 doigts, la photo est déplacée/tournée/zoomée (redimensionnement géré par la fonction donnée suivante : `pinch(Point old_p1, Point p1, Point old_p2, Point p2)` où old_p1 (resp. p1) désigne la position précédente (resp. courante) de l'un des doigt et old_p2 (resp. p2) celle de l'autre doigt).

Dessinez la machine à état qui réalise ces interactions (vous pouvez faire l'hypothèse qu'au plus 2 doigts seront utilisés en même temps).

Question 4

Réalisez en java la machine à état de la question précédente.
Pour cela, vous est donnée la classe `Surface` qui utilisera votre classe `Interaction` de la manière suivante :

```
package exam.ui;

import javax.swing.JComponent;
import java.awt.Point;

public class Surface extends JComponent {
   Interaction interaction;
   public Surface() {
      interaction = new Interaction(this);
  }

   public void start_drag(Point start) { ... }
   public void drag(int dx, int dy) { ... }
   public void end_drag() { ... }

   public void pinch(Point old_p1, Point p1, Point old_p2, Point p2) { ... }
   ...
}
```

Votre classe `Interaction` pourra commencer ainsi :

```
package exam.ui;

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

Les doigts génèrent des événements souris (`MouseEvent`) qui ont une méthode supplémentaire pour connaître l'identifiant du doigt correspondant.
Les `MouseEvent` ont ainsi les méthodes utiles suivantes :

- `Point getPoint()` qui retourne le lieu de l'événement ;
- `int getX()` et `int getY()` qui retournent la même information ;
- `int getFingerId()` qui retourne un entier qui sera le même pour tout événement entre un ↓ et un ↑ (inclus) si et seulement si c'est le même doigt qui l'a provoqué.

Les interfaces fournies par `java.awt.event` permettant de recevoir les événements sont données ci-dessous, elles disposent toutes d'un adaptateur associé (`XxxxAdapter` pour `XxxxListener`).

```
public interface MouseListener {
      public void mousePressed(MouseEvent e);
      public void mouseReleased(MouseEvent e);
      ...
}

public interface MouseMotionListener {
      public void mouseDragged(MouseEvent e);
      ...
}
```

L'interface `MouseInputListener` qui réunit ces deux interfaces est également disponible.

Composants interactifs (5 points)

Question 5

Les figures ci-dessous donnent la fenêtre principale de l'application Preview sous MacOS avant et après un redimensionnement. Sachant que la largeur de la colonne de gauche peut être modifiée par l'utilisateur, donnez l'arbre des composants que vous utiliseriez pour coder cette interface avec Java/SWING en spécifiant si nécessaire pour les conteneurs les gestionnaires de géométrie utilisés et la localisation de leurs enfants. Ne détaillez pas le contenu de la vue centrale.