Interfețe Grafice cu Utilizatorul (GUI) II

1. Scopul lucrării

Obiectivele de învățare ala acestei sesiuni de laborator sunt:

- Înțelegerea modului de folosire a mouse pentru interacțiunea cu utilizatorul
- Acumularea cunoștințelor privind utilizarea claselor și interfețelor importante care sunt necesare în gestionarea interacțiunii cu mouse
- Acumularea de experiență de programare în gestiunea evenimentelor generate de mouse.

Mouse este tratat automat de către majoritatea componentelor, astfel că, în general, nu trebuie să ştiţi de el. Spre exemplu, dacă cineva dă clic pe un buton (**JButton**), veţi recepţiona un **ActionEvent**, dar nu este nevoie să ştiţi (şi n-ar trebui să vă pese) dacă aceasta s-a datorat unui clic cu mouse pe buton sau a fost cauzat de o apăsare de tastă "rapidă" (shortcut).

Grafica. Dacă desenaţi grafică proprie (d.e., într-un **JPanel**) şi aveţi nevoie să ştiţi daca utilizatorul face clic, atunci trebuie să ştiţi despre evenimentele legate de mouse. Puteţi dăuga cu uşurinţă un "ascultător" pentru mouse la un **JPanel**.

2. Clase şi interfeţe importante

Clasele care urmează sunt definite în **java.awt.event**. Primele trei sunt cele mai folosite.

- **MouseEvent** – se trimite un obiect de tipul MouseEvent object tuturor ascultătorilor de mouse. Cele mai folositoare informaţii într-un MouseEvent sunt coordonatele x şi y ale cursorului mouse-ului.
- **MouseListener** – Interfaţă pentru apăsări şi eliberări de butoane de mouse, clic-uri, intrări şi ieşiri din zone.
- **MouseMotionListener** – Interfaţă pentru deplasări şi trageri (drag).
- **MouseInputListener** – Combinaţie de interfaţă între MouseListener şi MouseMotionListener.
- **MouseAdapter** – Clasă utilă pentru scrierea unui ascultător anonim pentru apăsări de butoane ale mouse, intrări în zone, …
- **MouseMotionAdapter** – Clasă utilă pentru scrierea unui ascultător anonim pentru deplasarea mouse.

2.1. MouseListener – tratează apăsări, eliberări, clicuri, intrări şi ieşiri.

Acest tip de ascultător de mouse este pentru evenimente care nu se întâmplă de obicei prea des – se apasă sau eliberează un buton al mouse sau mouse intră sau iese din zona componentei cu ascultător. Iată care sunt acţiunile pe care **MouseListener** le interceptează.

press	Unul dintre butoanele mouse a fost apăsat.
release	Unul dintre butoanele mouse a fost eliberat.
click	Un buton al mouse a fost apăsat şi eliberat fără a mişca mouse. Acesta este poate cel mai folosit.
enter	Cursorul mouse intră pe componentă. Folosit adesea pentru a schimba cursorul.
exit	Cursorul mouse iese de pe componentă. Adesea folosit pentru a restaura cursorul.

Pentru a asculta astfel de evenimente folosiţi **addMouseListener**.

2.3.1. Interfaţa MouseListener

Pentru a implementa interfaţa **MouseListener** trebuie să definiţi următoarele metode. Puteţi copia aceste definiţii în program şi construi un corp cu înţeles pentru metodele care sunt de interes.

```
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

Metoda este apelată	Atunci când utilizatorul efectuează acţiunea aceasta
mouseClicked	Un clic este rezultatul unei apăsări şi eliberări. Probabil cea mai comună metodă de scris.
mousePressed	A fost apăsat un buton al mouse (oricare dintre cele trei posibile)
mouseReleased	A fost eliberat un buton al mouse.
mouseEntered	Cursorul mouse intră pe o componentă. Aţi putea scrie aceasta pentru a schimba cursorul.
mouseExited	Cursorul mouse iese de pe o componentă. Aţi putea scrie aceasta pentru a restaura cursorul.

Pentru a obţine coordonatele mouse. Toate coordonatele sunt relative le colţul din stânga sus al componentei care are ascultătorul pentru mouse. Folosiţi următoarele metode din **MouseEvent** pentru a obţine coordonatele x şi y ale locului în care a apărut evenimentul mouse.

```
int getX() // returns the x coordinate of the event.
int getY() // returns the y coordinate of the event.
```

Pentru a testa clicuri duble. Folosiţi următoarea metodă din **MouseEvent** pentru a obţine numărul de clicuri.

```
int getClickCount() // numarul de clicuri pe mouse
```

2.3.2. MouseMotionListener – tratează deplasările şi tragerile (drags).

Deplasări şi trageri (târâri – drags). La deplasarea mouse, interfața generează foarte rapid evenimente. Dacă un buton al mouse este apăsat pe timpoul deplasării, aceasta se numește târâre (drag). Evenimentele dintyr-o deplasare sau târâre sunt generate foarte repede și pot fi ascultate dacă adăugăm un ascultător de mișcare a mouse (mouse motion listener).

MouseMotionListener methods

Pentru a implementa un MouseMotionListener, trebuie definite metodele care urmează:

```
public void mouseMoved(MouseEvent e) {}
public void mouseDragged(MouseEvent e) {}
```

This method is called	When the user does this action
mouseMoved(…)	The mouse is moved while over the component.
mouseDragged(…)	The mouse is dragged (moved with a button pressed).

2.2. Ascultători (listeners) pentru mouse - cum și unde se scriu

Există câteva stiluri de folosire a ascultătorilor de mouse. Ascultătorii sunt, de obicei, adăugați la un panou grafic cu metoda paintComponent.

Ascultarea în panoul însuși

Este uzual ca un panou să asculte propriile evenimente. De exemplu,

```
class DrawingPanel extends JPanel implements MouseListener
{

   public DrawingPanel()
   { // Constructor
      this.addMouseListener(this);
      . . .
   }
```

```
    public void paintComponent(Graphics g)
    {
       . . .
    }
    . . .
    public void mousePressed(MouseEvent e) {. . .}
    public void mouseReleased(MouseEvent e) {. . .}
    public void mouseClicked(MouseEvent e) {. . .}
    . . .
}
```

Panoul poate comunica schimbările cu exteriorul dacă: (1) îl face o subclasă, (2) furnizează metode de acces, sau (3) furnizează constructorului un obiect "model" (din MVC).

Ascultarea din afara panoului

Se poate crea un panou și să se dorească ca ascultătorii să fie exteriori acestuia, din rațiuni de convenabilitate a interacțiunii cu ascultătorii. Dacă există doar un singur asemenea panou, atunci se pot implementa interfețele ascultător de mouse în clasa care nu e panou și se pot scrie toate metodele de ascultare. De exemplu,

```
public class MyClass implements MouseListener {
    . . .
    DrawingPanel drawing = new DrawingPanel();
    drawing.addMouseListener(this);
    . . .

    public void mousePressed(MouseEvent e) {. . .}
    public void mouseReleased(MouseEvent e) {. . .}
    public void mouseClicked(MouseEvent e) {. . .}
    . . .
}

class DrawingPanel extends JPanel {
    public void paintComponent(Graphics g) {
       . . .
    }
    . . .
}
```

Aceasta necesită să existe metode mutatoare în clasa DrawingPanel astfel încât să se poată schimba ce se desenează. Sau se poate transmite unui constructor pentru DrawingPanel un obiect pentru "model" care i-ar permite să obțină valorile de care are nevoie metoda paintComponent.

Ca mai sus cu ascultători anonimi

Dacă doriți să ascultați doar un fel de eveniment, atunci este ușor de folosit clasa MouseAdapter sau clasa MouseMotionAdapter pentru a crea un ascultătore anonim. Spre exemplu, pentru a asculta clicuri pe mouse,

```
p.addMouseListener(new MouseAdapter()
{
    public void mouseClicked(MouseEvent e)
    {
       x = e.getX();
       y = e.getY();
    }
});
```

2.3. Butoane ale mouse, taste modificatoare - cum să verificăm ce butoane s-au apăsat

Butoane ale mouse. Java suportă până la trei butoane ale mouse. Chiar dacă perifericul nu are trei butoane separate, unele pot fi simulate prin apăsarea de taste modificatoare concomitent cu butoanele mouse.

Obiectul MouseEvent care este transmis ascultătorului conține informații care permit să interogați ce combinații de butoane s-au apăsat la apariția evenimentului. Controalele de defilare (scroll) ale mouse au fost suportate prima dată în Java 2 SDK 1.4.

Există două căi de testare a butoanelor mouse și a tastelor modificatoare, folosind:

- *Metode* pentru a afla starea butoanelor mouse și a tastelor modificatoare.
- *Măști pe biți* care sunt definite în clasa InputEvent pentru a examina biții modificatori ai MouseEvent. Această metodă este una de verificare *foarte rapidă*, mai ales în cazul combinațiilor complexe de modificatori și butoane -- adr e nevoie să înțelegeți operatorii pe biți (| & ^ ~ >> >>> <<).

2.3.1. Pentru a folosi metode de verificare a butoanelor mouse

Pentru a verifica ce buton s-a apăsat, apelați una dintre metodele statice din SwingUtilities. Aceste metode t=returnează true dacă s-a apăsat butonul corespunzător. Observați că mai mult de o metodă va returna true dacă este apăsat mai mult de un buton simultan.

- boolean SwingUtilities.isLeftMouseButton(MouseEvent anEvent)
- boolean SwingUtilities.isMiddleMouseButton(MouseEvent anEvent)
- boolean SwingUtilities.isRightMouseButton(MouseEvent anEvent)

2.3.2. Pentru a folosi metode de verificare a tastelor modificatoare

Pentru a verifica ce taste modificatoare sunt apăsate folosiți următoarele metode din clasa MouseEvent:

```
    boolean isAltDown()     // true if Alt key middle mouse button
    boolean isControlDown() // true if Control key is pressed
    boolean isShiftDown()   // true if Shift key is pressed
    boolean isAltGraphDown()// true if Alt Graphics key (found on some keyboards)
  is pressed
    boolean isMetaDown()    // true if Meta key or right mouse button
```

De exemplu, înlăuntrul unui ascultător de mouse am putea face un test precum cel care urmează pentru a vedea ce buton s-a apăsat simultan cu tasta Shift. Presupuneți că e este un obiect de clasa MouseEvent .

```
      if (SwingUtilities.isRightMouseButton(e) && e.isShiftDown())
         ...
```

2.3.3. Pentru a folosi măști pe biți pentru verificarea butoanelor mouse și a tastelor modificatoare apăsate

Folosiți metoda getModifiers() din MouseEvent pentru a obține o mască pe biți care spune ce butoane au fost apăsate la apariția evenimentului. Măștile pentru fiecare buton și tastele modificatoare sunt date în tabelul următor::

Mask	Meaning
InputEvent.BUTTON1_MASK	mouse button1
InputEvent.BUTTON2_MASK	mouse button2
InputEvent.BUTTON3_MASK	mouse button3
InputEvent.ALT_MASK	alt key
InputEvent.CTRL_MASK	control key
InputEvent.SHIFT_MASK	shift key

InputEvent.META_MASK	meta key
InputEvent.ALT_GRAPH_MASK	alt-graph key

Pentru a rescrie exemplul anterior folosind măști pe biți în scopul testării dacă s-a apăsat butonul din dreapta al mouse simultan cu tasta Shift, am putea face următoarele:

```
int RIGHT_SHIFT_MASK = InputEvent.BUTTON3_MASK + InputEvent.SHIFT_MASK;
. . .
if ((e.getModifiers() & RIGHT_SHIFT_MASK) == RIGHT_SHIFT_MASK) {
   ...
```

2.4. Exemplu - DragDemo.java - Exemplul arată târârea cu mouse.

Târâți mingea in aplicatie spre stângat. Programul sursa este dat mai jos. Cele două fișiere sursă sunt date mai jos.
Imaginea se desenează folosind metoda (fillOval) din Graphics, dar se poate ușor afișa o imagine în loc de minge.
Cadrul principal

```
/** DragDemo.java - Mouse drag example dual application
   @author Fred Swartz
   @version 2004-04-15
*/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
//////////////////////////////////////////////////////////////// class DragDemo

public class DragDemo{
   //============================================================ method main
   public static void main(String[] args) {
      JFrame window = new JFrame();
      window.setTitle("Drag Demo");
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setContentPane(new DragBallPanel());
      window.pack();
      window.show();
   }//end main

}//endclass DragDemo
```

Panoul folosit pentru grafică

```
/** DragBallPanel.java - Panel that allows dragging a ball around.
   @author Fred Swartz
   @version 2004-04-15
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

//////////////////////////////////////////////////////// class DragBallPanel
/** La apelul asculktatorului mousePressed se testeaza pozitia to
   pentru a vedea daca este in zona mingii. Daca este,
   (1) _canDrag is set true meaning pay attention to the MouseDragged events.
   (2) Record where in the ball (relative to the upper left coordinates)
      the mouse was clicked, because it looks best if we drag from there.
*/
public class DragBallPanel extends JPanel implements MouseListener, MouseMotionListener {

   private static final int BALL_DIAMETER = 40; // Diameter of ball
   //--- instance variables
   /** Ball coords.  Changed by mouse listeners.  Used by paintComponent. */
```

```
   private int _ballX     = 50;   // x coord - set from drag
   private int _ballY     = 50;   // y coord - set from drag

   /** Position in ball of mouse press to make dragging look better. */
   private int _dragFromX = 0;    // pressed this far inside ball's
   private int _dragFromY = 0;    // bounding box.

   /** true means mouse was pressed in ball and still in panel.*/
   private boolean _canDrag  = false;

   //============================================================ constructor
   /** Constructor sets size, colors, and adds mouse listeners.*/
   public DragBallPanel() {
      setPreferredSize(new Dimension(300, 300));
      setBackground(Color.blue);
      setForeground(Color.yellow);
      //--- Add the mouse listeners.
      this.addMouseListener(this);
      this.addMouseMotionListener(this);
   }//endconstructor

   //=================================================== method paintComponent
   /** Ball is drawn at the last recorded mouse listener coordinates. */
   public void paintComponent(Graphics g) {
      super.paintComponent(g);   // Required for background.
      g.fillOval(_ballX, _ballY, BALL_DIAMETER, BALL_DIAMETER);
   }//end paintComponent

   //=================================================== method mousePressed
   /** Set _canDrag if the click is in the ball (or in the bounding
      box, which is lazy, but close enuf for this program).
      Remember displacement (dragFromX and Y) in the ball
      to use as relative point to display while dragging.
   */
   public void mousePressed(MouseEvent e) {
      int x = e.getX();   // Save the x coord of the click
      int y = e.getY();   // Save the y coord of the click

      if (x >= _ballX && x <= (_ballX + BALL_DIAMETER)
            && y >= _ballY && y <= (_ballY + BALL_DIAMETER)) {
         _canDrag = true;
         _dragFromX = x - _ballX;  // how far from left
         _dragFromY = y - _ballY;  // how far from top
      } else {
         _canDrag = false;
      }
   }//end mousePressed

   //============================================================ mouseDragged
   /** Set x,y  to mouse position and repaint. */
   public void mouseDragged(MouseEvent e) {
      if (_canDrag) {   // True only if button was pressed inside ball.
         //--- Ball pos from mouse and original click displacement
         _ballX = e.getX() - _dragFromX;
         _ballY = e.getY() - _dragFromY;

         //--- Don't move the ball off the screen sides
         _ballX = Math.max(_ballX, 0);
         _ballX = Math.min(_ballX, getWidth() - BALL_DIAMETER);

         //--- Don't move the ball off top or bottom
         _ballY = Math.max(_ballY, 0);
         _ballY = Math.min(_ballY, getHeight() - BALL_DIAMETER);

         this.repaint(); // Repaint because position changed.
```

```
        }
    }//end mouseDragged

    //================================================== method mouseExited
    /** Turn off dragging if mouse exits panel. */
    public void mouseExited(MouseEvent e) {
        _canDrag = false;
    }//end mouseExited

    //================================================ Ignore other mouse events.
    public void mouseMoved   (MouseEvent e) {}  // ignore these events
    public void mouseEntered (MouseEvent e) {}  // ignore these events
    public void mouseClicked (MouseEvent e) {}  // ignore these events
    public void mouseReleased(MouseEvent e) {}  // ignore these events
}//endclass DragBallPanel
```

2.5. Animație folosind clasa Timer

La fel ca și în cazul butoanelor sau a altor componente grafice, și pentru Timer trebuie implementată metoda actionPerformed() din interfața ActionListener. Pentru a porni/opri o animație se apelează metodele start() și stop() din Timer.

Un exemplu simplu de animație:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TimerEx extends JPanel implements ActionListener{
    JLabel l;
    Timer t;
    int x = 10;
    int y = 300;

   TimerEx(){
      ImageIcon img = new ImageIcon("Mario.gif");
      l = new JLabel(img);
      l.setLocation(x, y);
      this.add(l);
      setBackground(Color.white);
      t = new Timer(100, this);
      t.addActionListener(this);
      t.start();
   }

// @override
   public void actionPerformed(ActionEvent e){
      x+=20;
      if (x>800) x = 50;
      l.setLocation(x,y);
   }

   public static void main(){
      JFrame frame = new JFrame("Timer Example");
      frame.setDefaultCloseOperation(
```

```
                JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 800);
        TimerEx pane= new TimerEx();
        frame.setContentPane(pane);
        frame.setVisible(true);
    }
}
```

3. Mersul lucrării

3.1. Studiați și executați exemplele prezentate

3.2. Adăugați o parte de animație exemplului DragBall astfel:

- la click-ul mouse-ului în interiorul bilei, generati coordonate aleatoare bilei la un interval de timp dat.
- Afișați în colțul stânga sus al panoului coordonatele curente ale bilei. Indicații: folosiți metoda *drawString(String s, int x, int y)* din clasa Graphics. Codul se va completa în interiorul metodei *paintComponent(Graphics g).*
- O variantă mai avansată a acestei cerințe este de a genera o direcție de mers bilei pe care aceasta să se deplaseze până la întâlnirea marginilor panoului, de unde direția ar trebui să se modifice astfel încât bila să nu iasă din panou).

3.3. Faceți și modificări în cod (d.e. adăgați metode). Observați efectele modificărilor.