

Interfețe Grafice cu Utilizatorul (GUI) II

1. Scopul lucrării

Obiectivele de învățare ale 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-ul
- Acumularea de experiență de programare în gestiunea evenimentelor generate de mouse.

Mouse-ul este tratat automat de către majoritatea componentelor, astfel că, în general, nu trebuie să știți de el. Spre exemplu, dacă cineva dă clic pe un buton (**JButton**), veți recepționa un **ActionEvent**, dar nu este nevoie să știți (și 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 desenați grafică proprie (d.e., într-un **JPanel**) și aveți nevoie să știți unde a dat clic utilizatorul, atunci trebuie să știți despre evenimentele legate de mouse. Puteți adăuga cu ușurință un "ascultător" pentru mouse la un **JPanel**.

2. Clase și interfețe importante

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

- **MouseEvent** – se trimite un obiect de tipul **MouseEvent** tuturor ascultătorilor de mouse. Cele mai folosite informații într-un **MouseEvent** sunt coordonatele x și y ale cursorului mouse-ului.
- **MouseListener** – Interfață pentru apăsări și eliberări de butoane de mouse, clic-uri, intrări și ieșiri din zone.
- **MouseMotionListener** – Interfață pentru deplasări și trageri (drag).
- **MouseInputListener** – Combinație de interfață între **MouseListener** și **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 și ieșiri.

Acest tip de ascultător de mouse este pentru evenimente care nu se întâmplă de obicei prea des – se apasă sau se eliberează un buton al mouse sau mouse intră sau iese din zona componentei cu ascultător. Tabelul 1 prezintă acțiunile pe care **MouseListener** le interceptează.

Acțiune	Semnificație
press	Unul dintre butoanele mouse a fost apăsat.
release	Unul dintre butoanele mouse a fost eliberat.
click	Un buton al mouse a fost apăsat și eliberat fără a mișca mouse. Această acțiune este, probabil, cea mai folosită.
enter	Cursorul mouse intră pe componentă. Folosită adesea pentru a schimba cursorul.
exit	Cursorul mouse iese de pe componentă. Adesea folosită pentru a restaura cursorul.

Tabelul 1. Acțiunile utilizatorului comunicate **MouseListener**

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

2.1.1. Interfața **MouseListener**

Pentru a implementa interfața **MouseListener** trebuie să definiți următoarele metode. Puteți copia aceste definiții în program și construi un corp cu înțeles 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 acțiunea aceasta
mouseClicked	Un clic este rezultatul unei apăsări și eliberări. Probabil cea mai comună metodă de scris.
mousePressed	A fost apăsător un buton al mouse (oricare dintre cele trei posibile)
mouseReleased	A fost eliberat un buton al mouse.
mouseEntered	Cursorul mouse intră pe o componentă. Ați putea scrie aceasta pentru a schimba cursorul.
mouseExited	Cursorul mouse iese de pe o componentă. Ați putea scrie aceasta pentru a restaura cursorul.

Pentru a obține coordonatele mouse. Toate coordonatele sunt relative le colțul din stânga sus al componentei care are ascultătorul pentru mouse. Folosiți următoarele metode din **MouseEvent** pentru a obține coordonatele *x* și *y* ale locului în care a apărut evenimentul mouse.

```
int getX() // întoarce coordonata x a evenimentului.
int getY() // întoarce coordonata y a evenimentului.
```

Pentru a testa clicuri duble. Folosiți următoarea metodă din **MouseEvent** pentru a obține numărul de clicuri.

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

2.1.2. **MouseMotionListener** – tratează deplasările și tragerile (*drags*)

Deplasări și trageri (târări – *drags*). La deplasarea mouse, interfața generează foarte rapid evenimente. Dacă un buton al mouse este apăsător pe timpul deplasării, aceasta se numește târare (*drag*). Evenimentele dintr-o deplasare sau târare sunt generate foarte repede și pot fi ascultate dacă adăugăm un ascultător de mișcare a mouse (*mouse motion listener*).

Metodele **MouseMotionListener**

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

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

Metoda apelată	Atunci când utilizatorul efectuează următoarea acțiune
mouseMoved(...)	Mouse este mișcat în timp ce se află deasupra componentei.
mouseDragged(...)	Mouse este tras (deplasat cu un buton apăsător).

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 `MouseListenerAdapter` sau clasa `MouseMotionAdapter` pentru a crea un ascultător 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 apăsate

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 apăsate 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 modificali ai `MouseEvent`. Această metodă este una de verificare *foarte rapidă*, mai ales în cazul combinațiilor complexe de modificali și butoane – aici 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 apăsate, apelați una dintre metodele statice din `SwingUtilities`. Aceste metode returnează `true` dacă s-a apăsate butonul corespunzător. Observați că mai mult de o metodă va returna `true` dacă este apăsate 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 is pressed

```

De exemplu, înlăuntru unui ascultător de mouse am putea face un test precum cel care urmează pentru a vedea ce buton s-a apăsate 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 apăstate

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

Mask	Meaning
<code>InputEvent.BUTTON1_MASK</code>	mouse button1
<code>InputEvent.BUTTON2_MASK</code>	mouse button2
<code>InputEvent.BUTTON3_MASK</code>	mouse button3
<code>InputEvent.ALT_MASK</code>	alt key
<code>InputEvent.CTRL_MASK</code>	control key
<code>InputEvent.SHIFT_MASK</code>	shift key
<code>InputEvent.META_MASK</code>	meta key
<code>InputEvent.ALT_GRAPH_MASK</code>	alt-graph key

Pentru a rescrie exemplul anterior folosind măști pe biți în scopul testării dacă s-a apăsător 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 în aplicație spre stânga. 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 - Aplicatie duala de târâre a mouse
    @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{
    //===== metoda 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 - Panou care permite târârea mingii.
    @author Fred Swartz
    @version 2004-04-15
*/

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

////////////////////////////////////// clasa DragBallPanel
/** La apelul ascultatorului mousePressed se testeaza pozitia
    pentru a vedea daca este in zona mingii. Daca este,
    (1) _canDrag este setat la true cu semnificatia "atentie la
    evenimentele MouseDragged".
    (2) se înregistreaza unde în minge (relative la coordonatele din
    stânga sus) a fost dat clic, fiindca arata cel mai bine daca o târâm
    de acolo.
*/
public class DragBallPanel extends JPanel implements MouseListener,
MouseMotionListener {

    private static final int BALL_DIAMETER = 40; // Diametrul mingii
    //--- variabile instanta
    /** Coordonatele mingii. Modificate de ascultatori. Folosite de
        paintComponent. */
    private int _ballX = 50; // x coord - setata din târâre
    private int _ballY = 50; // y coord - setata din târâre

    /** Pozitia în minge a unei apasari pe mouse pentru a afce târârea sa
        arate mai bine */
    private int _dragFromX = 0; // apasat atât in interiorul
    private int _dragFromY = 0; // dreptunghiului mingii.

    /** true inseamna ca mouse a fost apasat in minge si înca in panou.*/
    private boolean _canDrag = false;

    //=====
    constructor
    /** Constructorul seteaza marimea, culorile si adauga ascultatori de
        mouse.*/
    public DragBallPanel() {
        setPreferredSize(new Dimension(300, 300));
        setBackground(Color.blue);
        setForeground(Color.yellow);
        //--- Add the mouse listeners.
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
    } //endconstructor

    //===== metoda paintComponent
    /** Mingea este desinata la ultimele coordonate înregistrate de
        ascultator */
    public void paintComponent(Graphics g) {
        super.paintComponent(g); // Required for background.
        g.fillOval(_ballX, _ballY, BALL_DIAMETER, BALL_DIAMETER);
    } //end paintComponent

```

```

//===== method mousePressed
/** Seteaza _canDrag daca clic este în minge (sau în dreptunghiul care
    o margineste, ceea ce este destul de bun pentru acest program).
    Pastrati deplasamentul (dragFromX si Y) în minge ca sa-l folositi
    ca punct relativ de afisare în timpul tâtării.
 */
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

//===== metoda mouseDragged
/** Seteaza pozitia x,y a mouse si redeseneaza. */
public void mouseDragged(MouseEvent e) {
    if (_canDrag) {    // True doar la apasare in interiorul mingii.
        //--- Pozitia mingii din mouse si deplasament click original
        _ballX = e.getX() - _dragFromX;
        _ballY = e.getY() - _dragFromY;

        //--- Nu muta mingea in afara marginilor ecranului
        _ballX = Math.max(_ballX, 0);
        _ballX = Math.min(_ballX, getWidth() - BALL_DIAMETER);

        //--- Nu muta mingea pest marginile de sus sau jos
        _ballY = Math.max(_ballY, 0);
        _ballY = Math.min(_ballY, getHeight() - BALL_DIAMETER);

        this.repaint();    // Redeseneaza - s-a schimbat pozitia
    }
}
} //end mouseDragged

//===== metoda mouseExited
/** Dezactiveaza târârea daca mouse iese din panou. */
public void mouseExited(MouseEvent e) {
    _canDrag = false;
}
} //end mouseExited

//===== Ignora alte evenimente mouse.
public void mouseMoved (MouseEvent e) {}
public void mouseEntered (MouseEvent e) {}
public void mouseClicked (MouseEvent e) {}
public void mouseReleased (MouseEvent e) {}
} //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.start();
    }

    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, generați coordonate aleatoare ale mingii la un interval de timp dat
- Afișați în colțul stânga sus al panoului coordonatele curente ale mingii. 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 direcția ar trebui să se modifice astfel încât bila să nu iasă din panou)

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