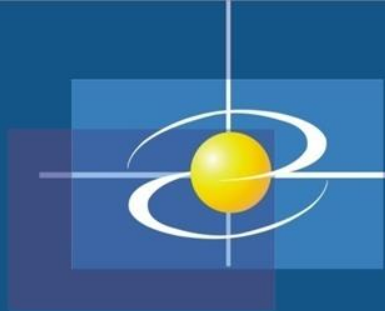
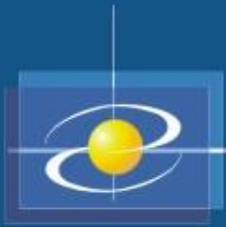




Programski jezik JAVA

PREDAVANJE 7

2020



Window Events

- Ako želimo učiniti nešto složenije od samog prekidanja programa kada korisnik zatvara prozor onda moramo reagovati na događaje koje prozor generiše.
- Kada na primjer zatvaramo prozor, `JFrame` generiše `WindowEvent`. Ako želimo da reagujemo na taj događaj trebamo da registrujemo `WindowListener`. Interfejs `WindowListener` izgleda ovako:

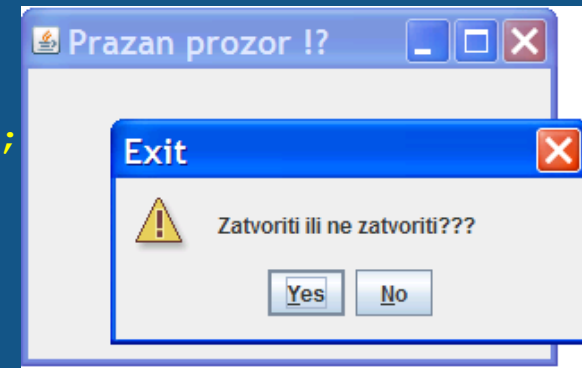
```
1  package java.awt.event;
2
3  import java.util.EventListener;
4
5  public interface WindowListener extends EventListener {
6      public void windowOpened(WindowEvent e);
7      public void windowClosing(WindowEvent e);
8      public void windowClosed(WindowEvent e);
9      public void windowIconified(WindowEvent e);
10     public void windowDeiconified(WindowEvent e);
11     public void windowActivated(WindowEvent e);
12     public void windowDeactivated(WindowEvent e);
13 }
```



Window Events

- Tako bismo u klasi (npr. `SmartFrame`) koja proširuje `JFrame` imali:

```
class SmartFrame extends JFrame
{
    public SmartFrame()
    {
        setTitle("Prazan prozor !?");
        setSize(300,200);
        WindowListener wl = new Terminator();
        addWindowListener(wl);
    }
}
```



- Naš se `WindowListener` ovdje naziva `Terminator`. Prilikom zatvaranja prozora ponuditi ćemo korisniku *confirmation dialog*, prozor u kojem očekujemo da potvrdi svoju odluku klikom na OK dugme.



Window Events

Pri tome koristimo statičke metode klase `javax.swing.JOptionPane`.
Kod u klasi `Terminator` je sljedeći:

```
class Terminator implements WindowListener
{
    public void windowOpened(WindowEvent e){}
    public void windowClosing(WindowEvent e){
        int i=JOptionPane.showConfirmDialog(null, "Zatvoriti ili ne zatvoriti???",
                                                "Exit", JOptionPane.YES_NO_OPTION,
                                                JOptionPane.WARNING_MESSAGE);

        if(i == JOptionPane.OK_OPTION)
            System.exit(0);
    }
    public void windowClosed(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
}
```

- Uočimo ovdje da smo morali implementirati sve metode iz interfejsa `WindowListener` premda nam je bila potrebna samo jedna metoda. Sve su druge implementirane trivijalno. Da bi se to izbjeglo, Java nudi *Adapter* klase pridružene interfejsima s više metoda. Takva adapter klasa implementira na trivijalan način (dakle, ne rade ništa) sve metode iz interfejsa pa korisnik treba umjesto implementacije interfejsa proširiti pripadnu adapter klasu i preraditi samo onu metodu koja ga zanima. U slučaju `WindowListener` pripadna adapter klasa se zove `WindowAdapter`.



Window Events

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

public class TestWindowListener {
    public static void main(String[] args){
        SmartFrame frame = new SmartFrame();
        frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE );
        frame.setVisible(true);
    }
}

class SmartFrame extends JFrame
{
    public SmartFrame()
    {
        setTitle("Prazan prozor !?");
        setSize(300,200);
        WindowListener wl = new Terminator();
        addWindowListener(wl);
    }
}

class Terminator extends WindowAdapter
{
    public void windowClosing(WindowEvent e){
        int i=JOptionPane.showConfirmDialog(null, "Zatvoriti ili ne zatvoriti???",
            "Exit", JOptionPane.YES_NO_OPTION,
            JOptionPane.WARNING_MESSAGE);
        if(i == JOptionPane.OK_OPTION)
            System.exit(0);
    }
}
```



JOptionPane

- U prethodnom primjeru smo koristili klasu `JOptionPane` i njenu statičku metodu `showConfirmDialog`. Ova nam klasa nudi četiri takve statičke metode koje otvaraju jednostavne prozore za komunikaciju sa korisnikom. To su:
 - `showMessageDialog` -- prikaži poruku i čekaj da korisnik klikne OK
 - `showConfirmDialog` -- prikaži poruku i čekaj odobrenje (OK/Cancel)
 - `showOptionDialog` -- prikaži poruku i čekaj da korisnik selektuje jednu od ponuđenih opcija
 - `showInputDialog` -- prikaži poruku i čekaj korisnikov unos
- Preciznije ćemo opisati `showMessageDialog` i `showConfirmDialog`. Metode su preopterećene pa ćemo pokazati samo dvije tipične:

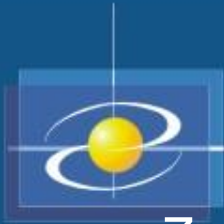
```
public static void showMessageDialog(Component parentComponent,  
                                   Object message, String title, int messageType)  
  
public static int showConfirmDialog(Component parentComponent,  
                                   Object message, String title, int optionType, int messageType)
```

- Za prvi argument se može uvijek staviti `null`. Treći argument `title` je naslov koji dolazi na prozor.



JOptionPane

- Ovi se prozori sastoje od ikone, poruke i jednog ili više dugmadi. Ikona zavisi od tipa poruke (argument `messageType`) koji može biti:
 - `JOptionPane.ERROR_MESSAGE`
 - `JOptionPane.INFORMATION_MESSAGE`
 - `JOptionPane.WARNING_MESSAGE`
 - `JOptionPane.QUESTION_MESSAGE`
 - `JOptionPane.PLAIN_MESSAGE`
- U slučaju `PLAIN_MESSAGE` nema ikone.
- Poruka (argument `message`) zadaje se najčešće kao `String`, ali može biti i ikona ili čak niz objekata. Zbog toga se zadaje kao promjenljiva tipa `Object`.
- Broj dugmadi u prozoru zavisi od promjenljive koja se zove `optionType` i može imati ove vrijednosti:
 - `JOptionPane.DEFAULT_OPTION`
 - `JOptionPane.YES_NO_OPTION`
 - `JOptionPane.YES_NO_CANCEL_OPTION`
 - `JOptionPane.OK_CANCEL_OPTION`



JOptionPane

- Značenje svake pojedine opcije dato je imenom. *MessageDialog* ima samo OK dugme, pa ovog argumenta u *showMessageDialog* nema.
- Isto tako, *showMessageDialog* ne vraća ništa, dok *showConfirmDialog* vraća cijeli broj koji označava izabranu opciju. Vrijednosti su ponovo date simboličkim imenima:
 - `JOptionPane.YES_OPTION`
 - `JOptionPane.NO_OPTION`
 - `JOptionPane.CANCEL_OPTION`
 - `JOptionPane.OK_OPTION`
 - `JOptionPane.CLOSED_OPTION`
- Zadnja vrijednost se vraća onda kada korisnik zatvori prozor bez potvrde.



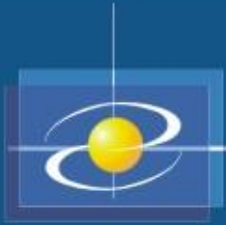
Događaji koji dolaze od miša

- Događaji vezani uz miša distribuiraju se osluškivačima koji su podijeljeni u tri interfejsa: **MouseListener**, **MouseMotionListener** i **MouseWheelListener**. Razlog je uglavnom efikasnost jer događaja vezanih za kretanje miša ima jako mnogo i većina aplikacija ih ne želi slušati.
- Interfejs **MouseListener**: Kada se pritisne neka od tipki na mišu poziva se metoda `mousePressed`; kada se tipka otpusti zove se `mouseReleased` i zatim `mouseClicked`. Pomoću `MouseEvent` objekata koji ove metode dobijaju moguće je naći koordinate događaja pomoću metoda:

```
public int getX()  
public int getY()  
public Point getPoint()
```

Unutar metode `mouseClicked` možemo koristiti metodu

public int `getClickCount()` koja se takodje nalazi u `MouseEvent` klasi koja daje broj klikanja.



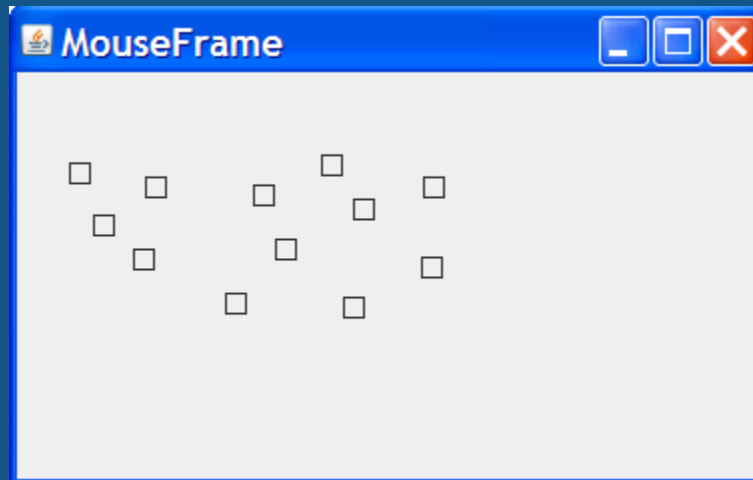
Nastavak...

- Ponekad se koriste kombinacije pritiska na tipku miša i neku od specijalnih tipki (Ctrl, Alt Shift). U tom slučaju treba koristiti metodu `public int getModifiersEx()` iz klase `InputEvent`, koja je nadklasa klase `MouseEvent`. Ona će vratiti niz bita koji treba testirati u odnosu na *maske* definisane u `InputEvent` klasi.
- Interfejs **MouseListener** definiše dvije metode:
`public void mouseDragged(MouseEvent e)`
`public void mouseMoved(MouseEvent e)`
- Prva se odnosi na situaciju kada se miš pomjera sa pritisnutom tipkom, a druga na kretanje bez pritisnute tipke (vidi klasu `java.awt.Cursor`).



Nastavak...

- Procesiranje događaja koji dolaze od miša može se vidjeti na primjeru koji se nalazi na sljedećem slajdu.
- Napisan je program koji otvara prozor u kome se jednim klikom miša iscrtava pravougaonik na onom mjestu na kome se klik dogodio.
- Dva klika na nekom pravougaoniku isti brišu. Pored toga, prelaskom miša preko pravougaonika mijenja se izgled kursora i ako na pravougaoniku pritisnemo tipku miša, a miš nastavimo povlačiti, pravougaonik će se kretati za mišem (*dragging*).





```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.util.*; // ArrayList

public class TestMouseListener {
    public static void main(String[] args)    {
        MouseFrame mf = new MouseFrame();
        mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mf.setVisible(true);
    }
}

class MouseFrame extends JFrame {
    public MouseFrame()    {
        setTitle("MouseFrame");
        setSize(300,200);
        MousePanel mp = new MousePanel();
        Container contentPane = getContentPane();
        contentPane.add(mp);
    }
}

class MousePanel extends JPanel {
    // Privatni podaci
    private static final int DUZINA = 10; // duzina stranice kvadrata
    private ArrayList      kvadrati;      // lista kvadrata
    private Rectangle2D    trenutni;      // aktuelni kvadrat

    public MousePanel()    {
        kvadrati = new ArrayList();
        trenutni = null;

        addMouseListener(new MouseHandler());
        addMouseMotionListener(new MouseMotionHandler());
    }
}
```



```
// Iscrtavanje panela
public void paintComponent(Graphics g)      {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    for(int i=0; i<kvadrati.size(); ++i)
        g2.draw((Rectangle2D) kvadrati.get(i));
}

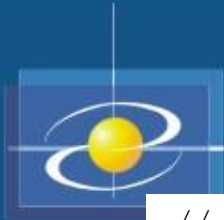
// Rutine za manipulaciju s listom kvadrata: add, find, remove
// Metode se jednostavno implemetiraju pomoću metoda klase ArrayList
// Dodaj novi kvadrat s centrom u tacki p.
public void add(Point2D p)      {
    double x = p.getX();
    double y = p.getY();

    trenutni = new Rectangle2D.Double(x-DUZINA/2, y-DUZINA/2,
                                      DUZINA, DUZINA);

    // Koristimo metodu add iz ArrayList
    kvadrati.add(trenutni);
    repaint();
}

// Pronađi element u listi koji sadrži tačku p. Vрати null ako
// takvog nema.
public Rectangle2D find(Point2D p)      {
    // Sav posao odrađuje metod contains iz Rectangle2D koja
    // ispituje je li tačka unutar pravougaonika.
    for(int i=0; i<kvadrati.size(); ++i){

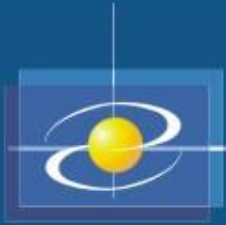
        Rectangle2D rec=(Rectangle2D) kvadrati.get(i);
        if(rec.contains(p)) return (Rectangle2D) kvadrati.get(i);
    }
    return null;
}
```



```
// Odstrani element iz liste
public void remove(Rectangle2D r)
{
    if(r == null) return;
    if(r == trenutni) trenutni = null;
    kvadrati.remove(r);
    repaint();
}

// Rutine za procesiranje događaja. Smještene su u dvije
//unutrašnje klase.
// Privatna unutrašnja klasa
private class MouseHandler extends MouseAdapter    {
    // Čim pritisnemo tipku miša kreiramo novi kvadrat
    public void mousePressed(MouseEvent e)          {
        // Da li se pritisak dogodio unutar nekog pravougaonik?
        trenutni = find(e.getPoint());
        if(trenutni == null)    // nije
            add(e.getPoint()); // dodaj novi pravougaonik
    }

    // Ako kliknemo dvaput u kvadratu brišemo ga
    public void mouseClicked(MouseEvent e)
    {
        // Da li se pritisak dogodio unutar nekog kvadrata?
        trenutni = find(e.getPoint());
        if(trenutni != null && e.getClickCount() >=2 )
            remove(trenutni);    // briši kvadrat
    }
}
```



```
// Privatna unutrašnja klasa
private class MouseMotionHandler implements MouseMotionListener
{
    public void mouseMoved(MouseEvent e)
    {
        // Ako se kretanje dešava unutar kvadrata promijeni kursor
        if(find(e.getPoint()) == null)
setCursor(Cursor.getDefaultCursor());
        else

setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
    }
    public void mouseDragged(MouseEvent e)
    {
        // Čim se stisne tipka unutar nekog kvadrata
        // biće postavljen trenutni
        if(trenutni != null)
        {
            int x = e.getX();
            int y = e.getY();

            // Vučemo kvadrat
            trenutni.setFrame(x-DUZINA/2, y-DUZINA/2, DUZINA, DUZINA);
            repaint();
        }
    }
}
}
```