Programski jezik JAVA
PREDAVANJE 8
2009

www.etf.ac.me

Grafičko Interfejs I - Događaji

- Programiranje grafičkog interfejsa - određeno događajima (*events*) izvan njega. Pri tome *događaji* dolaze od tastature (pritisci na tipke) i miša (kretanje miša i *klik* određenom tipkom miša).
- Svaki GUI (*graphical user interface*) program strukturiran je kao jedna beskonačna petlja u kojoj se:
  - skupljaju informacije o događajima koji su se desili
  - obavještavaju svi zainetresirani objekti.
- Objekt koji je zainteresovan za neki tip događaja reguje na njega izvršavanjem određenog dijela koda (neke svoje metode).
- U Javinom programu koji implementira grafičko sučelje imat ćemo tri vrste objekata: *izvore* događaja, o*sluškivače* i same *događaje*:
- Izvori događaja su komponente grafičkog interfejsa (npr. paneli, dugmad, slideri, itd.).
- Osluškivači (*listeners*) događaja su objekti koji reaguju na neku vrstu događaja.
- Sami događaju (*events*) su, naravno, instance određenih klasa.

Osluškivači – Registracija i Implementacija

- Komponenta koja je izvor događaja nekog tipa ima metodu kojom registruje osluškivače.
- Na taj način se ostvaruje veza između izvora i osluškivača i samo će osluškivači koji su registrovani biti obavješteni o događaju. Opšti oblik metode ovog tipa je:

```
izvorDogađaja.addDogađajListener(objektSlušač)
```

- Klasa koja modeluje osluškivač nekog događaja mora da implementira interfejs listener odgovarajućeg tipa.
- Interfejs deklariše metodu koja će automatski biti pozvana kad se događaj dogodi.
- Pošto je generiranje događaja u potpunosti pod kontrolom korisnika programa ne možemo znati kada će tačno metoda koju objekt-osluškivač definiše biti pozvana.
- Ono što znamo je da će automatski biti pozvane metode svih registrovanih osluškivača.

Primjer 1: JButton

- Klasa `javax.swing.JButton` modeluje grafičku komponentu koja predstavlja dugme.
- Ta komponenta generiše `ActionEvent` (klasa `java.awt.event.ActionEvent`) kojim se definiše *akcija* koju komponenta realizuje.
- U slučaju `JButton` komponente `ActionEvent` će biti generisan svaki put kada kliknemo na dugme.
- Osluškivač za `ActionEvent` mora da implementira interfejs `ActionListener` (u paketu `java.awt.event`) koje definiše smo jednu metodu: `actionPerformed(ActionEvent e)`.
- Ta će metoda automatski biti pozvana kad se klikne na dugme.
- Registracija osluškivača:

```
ActionListener listener=....;        //osluškivč ActionEvent-a
JButton button = new JButton("OK"); // izvor ActionEvent-a
button.addActionListener(listener); // registracija osluš.
```

Nastavak...

- Implementacija osluškivača

```
class MyListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        // reakcija na klik na dugme ide ovdje
    }
}
```

- Sada će svaki puta kad korisnik klikne na dugme s labelom "OK" biti pozvana metoda `listener.actionPerformed`.
- Metoda kao argument automatski dobija objekt klase `ActionEvent` koji reprezentuje događaj.
- Programer samo kreira i registruje osluškivač, dok se poziv odgovarajućoj metodi (`actionPerformed`) dešava automatski.

Kompletan Primjer

- Napravimo sada kompletan primjer: program će otvoriti prozor u kome se nalaze tri dugmeta, označena jednom bojom. Klikom na dugme mijenja se boja pozadine prozora.
- Konstruktor za `JButton` uzima kao argument labelu u obliku stringa ili ikonu ili oboje. Na primjer:

```
JButton yellow = new JButton("Yellow");
```

- Nakon toga button se mora smjestiti na panel pomoću metode add koju JPanel nasljeđuje iz klase java.awt.Container. Na primjer:

```
class ButtonPanel extends JPanel
{
       public ButtonPanel()
       {
           JButton yellow = new JButton("Yellow");
           add(yellow);
       }
 }
```

Nastavak...

- Nakon što smo button stavili na panel moramo registrovati njegov osluškivač (`ActionListener`).
- Taj će osluškivač promijeniti boju panela za šta treba imati pristup metodi `setBackground` iz `JPanel` klase.
- Prema tome, treba staviti klasu koja implementira `ActionListener` interfejs unutar klase `ButtonPanel`, jer će ona tada moći dohvatiti sve njene metode.
- Kako klasu nećemo koristiti izvan `ButtonPanel` klase slobodno ju možemo deklarisati privatnom.
- Konačno, registracija osluškivača ima ovaj oblik:

```
// ColorAction implementira ActionListener
 ColorAction yellowAction = new ColorAction(Color.YELLOW);
 yellow.addActionListener(yellowAction);
```

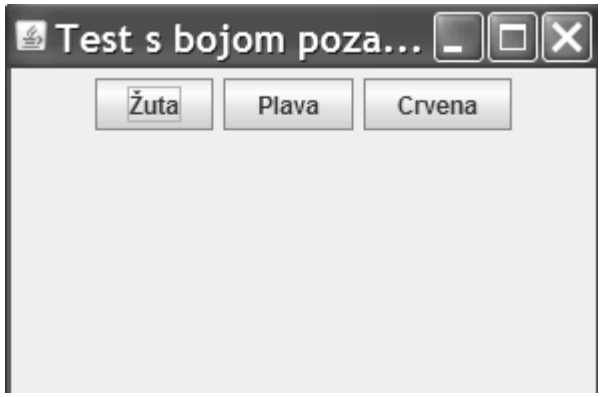
- Ovdje smo iskoristili klasu `java.awt.Color` koja implemetira neke temeljne boje u obliku konstanti (vidite dokumentaciju).

Program

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4
 5
 6  public class TestButton {
 7      public static void main(String[] args){
 8          ButtonFrame frame = new ButtonFrame();
 9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10          frame.setVisible(true);
11      }
12  }
13
14  class ButtonFrame extends JFrame
15  {
16      public ButtonFrame()
17      {
18          setTitle("Test s bojom pozadine");
19          setSize(300,200);
20          Container cp = getContentPane();
21          ButtonPanel panel = new ButtonPanel();
22          cp.add(panel);
23      }
24  }
25
26  class ButtonPanel extends JPanel
27  {
28      public ButtonPanel()
29      {
30          // Tri buttona
31          JButton yellow = new JButton(“Žuta");
32          JButton blue   = new JButton(“Plava");
33          JButton red    = new JButton(“Crvena");
34
35          // Dodajemo ih na panel
36          add(yellow);
37          add(blue);
38          add(red);
39
```

Primjer..

```
40          // Kreiramo osluškivače ...
41          ColorAction yellowAction = new ColorAction(Color.YELLOW);
42          ColorAction blueAction   = new ColorAction(Color.BLUE);
43          ColorAction redAction    = new ColorAction(Color.RED);
44
45          // registrujemo osluškivače
46          yellow.addActionListener(yellowAction);
47          blue.addActionListener(blueAction);
48          red.addActionListener(redAction);
49      }
50
51      // Privatna unutrašnja klasa
52      // Na taj način ColorAction() konstruktor ne mora dobiti
53      // referencu na ButtonPanel koja bi joj trebala da dohvati
54      // ButtonPanel.setBackground(backgroundColor)
55
56      private class ColorAction implements ActionListener
57      {
58          public ColorAction(Color c) { backgroundColor=c; }
59
60          public void actionPerformed(ActionEvent e) {
61              // metoda iz JComponent klase
62              setBackground(backgroundColor);
63          }
64
65          private Color backgroundColor;
66      }
67  }
68
69
```



Nastavak...

- Gornji kod možemo pojednostaviti jer se kreiranje svakog dugmeta sastoji od četiri akcije:
  - Instanciranje JButtona;
  - Dodavanje na panel (add);
  - Konstrukcija ActionListener objekta;
  - Registracija ActionListener objekta.
- Sve to možemo obaviti u jednoj metodi. Štoviše, klasa `ColorAction` može postati anonimna unutrašnja klasa. Nova metoda neka se zove `makeButton` :

```
void makeButton(String labela, final Color bojaPozadine)
    {
        JButton botun = new JButton(labela);
        add(botun);
        botun.addActionListener(new
                ActionListener()
                {
                     public void actionPerformed(ActionEvent e) {
                     // metoda iz JComponent klase
                     setBackground(bojaPozadine);
                }
                });
        // Referenca botun sada nestaje, ali to nije bitno jer ju čuva ButtonPanel
    }
```

Nastavak...

- Konstruktor klase `ButtoPanel` sada je vrlo jednostavan:

```
public ButtonPanel()
    {
        // Tri buttona
        makeButton("Yellow", Color.YELLOW);
        makeButton("Blue", Color.BLUE);
        makeButton("Red", Color.RED);
    }
```

Window Events

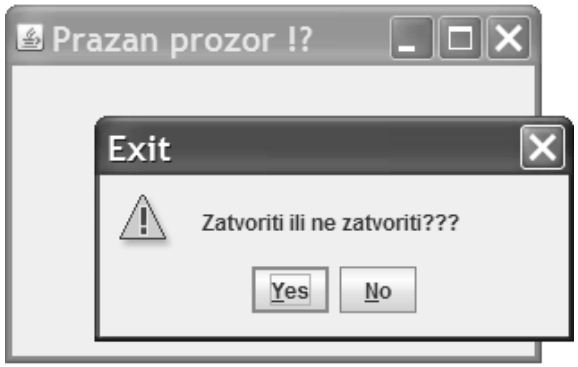
- Ako želimo učiniti nešto složenije od samog prekidanja programa kada korisnik zatvara prozor onda moramo reagovati na događaje koje prozor generiše.
- Kada na primjer zatvaramo prozor, `JFrame` generiše `WindowEvent`. Ako želimo da reagujemo na taj događaj trebamo registrujemo `WindowListener`. Interfejs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

Nastavak...

- 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 ponudit ćemo korisniku *confirmation dialog*, prozor u kojem očekujemo da potvrdi svoju odluku klikom na OK dugme.

Nastavak...

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

```
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`.

Cijeli kod TestWindowListener.java

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4
 5  public class TestWindowListener {
 6      public static void main(String[] args){
 7          SmartFrame frame = new SmartFrame();
 8          frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE );
 9          frame.setVisible(true);
10      }
11  }
12
13  class SmartFrame extends JFrame
14  {
15      public SmartFrame()
16      {
17          setTitle("Prazan prozor !?");
18          setSize(300,200);
19          WindowListener wl = new Terminator();
20          addWindowListener(wl);
21      }
22  }
23
24  class Terminator extends WindowAdapter
25  {
26      public void windowClosing(WindowEvent e){
27          int i=JOptionPane.showConfirmDialog(null, "Zatvoriti ili ne zatvoriti???",
28                                              "Exit", JOptionPane.YES_NO_OPTION,
29                                               JOptionPane.WARNING_MESSAGE);
30          if(i == JOptionPane.OK_OPTION)
31              System.exit(0);
32      }
33  }
```

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 s 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 selektira 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.

Nastavak...

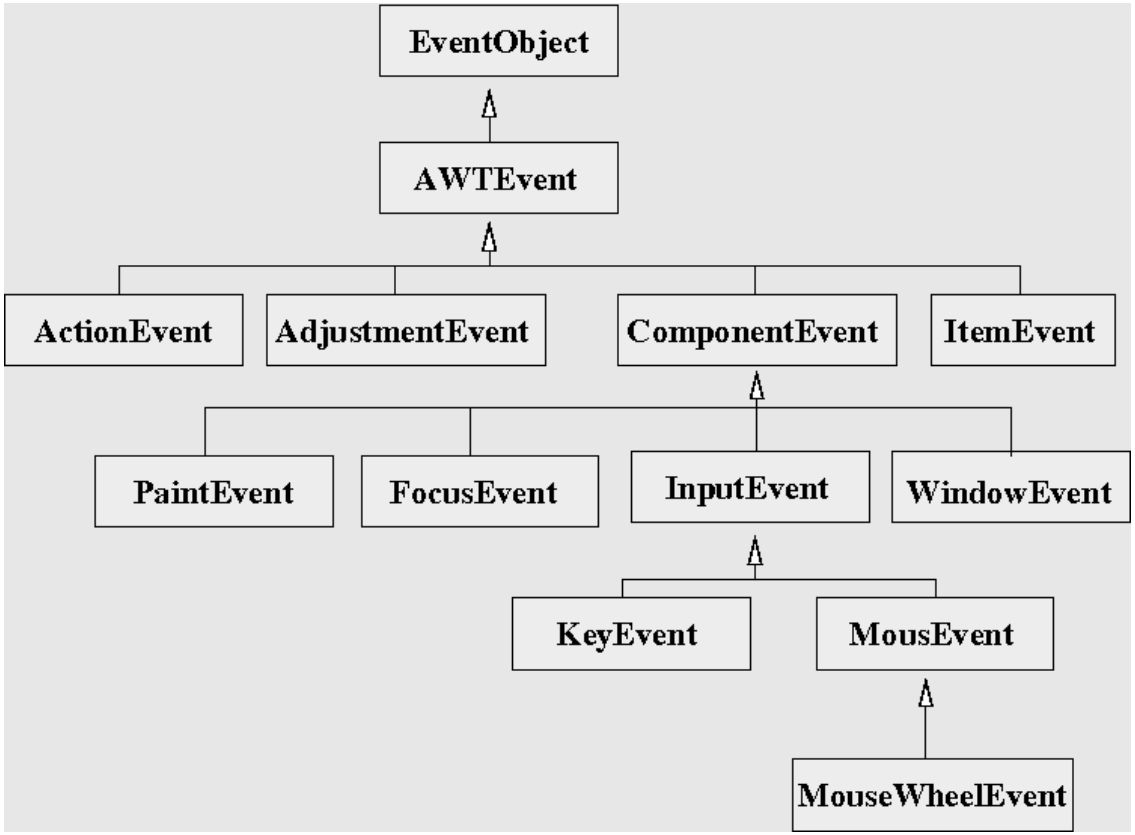
- Ovi se prozori sastoje od ikone, poruke i jednog ili više dugmadi. Ikona zavisi od tipa poruke (argument messageType) koji može biti:
  - JOptionPane.ERROR_MESSAGE
  - JOptionPane.NFORMATION_MESSAGE
  - JOptionPane.WARNING_MESSAGE
  - JOptionPane.QUESTION_MESSAGE
  - JOptionPane.PLAIN_MESSAGE
- U slučaju 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

Nastavak...

- 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 konfirmacije.

Stablo nasljeđivanja događaja

- Nadklasa svih događaja je `EventObject` klasa iz `java.util paketa`. Iz nje nastaje proširivanjem klasa `AWTEvent` iz `java.awt` paketa i većina *event*-a proširuje tu klasu. Većina korisnih *event*-klasa i *listener*-interfejsa je u paketu `java.awt.event`. Pored toga postoje događaji koje koristi `Swing` i koji su dobijeni proširivanjem direktno iz `EventObject` klase.



Nastavak...

- Sljedeća tabela sumira najvažniji dio interfejsa obrade događaja.

Interfejs	**Metode**	**Parametri/metode**	**Generatori**
ActionListener	actionPerformed	ActionEvent •getActionCommand •getModifiers	•AbstractButton •JComboBox •JTextField •Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent •getAdjustable •getAdjustmentType •getValue	•JScrollBar
ItemListener	itemStateChanged	ItemEvent •getItem •getItemSelectable •getStateChange	•AbstractButton •JComboBox
FocusListener	focusGained focusLost	FocusEvent •isTemporary	•Component
KeyListener	keyPressed keyReleased keyTyped	KeyEvent •getKeyChar •getKeyCode •getKeyModifiersText •getKeyText •isActionKey	•Component

Nastavak...

Interfejs	**Metode**	**Parametri/metode**	**Generatori**
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent •getClickCount •getX •getY •getPoint •translatePoint	•Component
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	•Component
MouseWheelListener	mouseWheelMoved	MouseWheelEvent •getWheelRotation •getScrollAmount	•Component
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	WindowEvent •getWindow	•Window
WindowFocusListener	windowGainedFocus windowLostFocus	WindowEvent •getOppositeWindow	•Window
WindowStateListener	windowStateChanged	WindowEvent •getOldState •getNewState	•Window

Događaji koji dolaze s tastature

- Jedan `KeyEvent` se generiše i kada se tipka tastature pritisne a drugi kad se otpusti. Kada se tipka pritisne bit će pozvana `keyPressed` metoda svakog registrovanog `KeyListener` objekta. Kada se tipka pusti poziva se `keyReleased` metoda. Nakon što se desi slijed pritiska i puštanja tipke poziva se metoda `keyTyped` svakog registrovanog `KeyListener` objekta.
- Ako želimo dobiti znak koji je unesen s tastature treba da koristimo metodu `keyTyped` iz interfejsa `KeyListener` i u njoj pozvati metodu `getKeyChar` iz `java.awt.event.KeyEvent klase`. Na primjer:

```
public void keyTyped(KeyEvent e)
    {
        char c = e.getKeyChar();
        System.out.println("keyTyped : znak = "+c);
    }
```

Nastavak...

- Metoda `keyTyped` će biti pozvana samo ako je pritisnuta tipka koja odgovara nekom Unicode znaku. Kada želimo pratiti ostale tipke (npr. Ctrl, Alt itd.) treba da koristimo metodu `keyPressed` iz interfejsa `KeyListener`. U njoj tada pozivamo metodu `getKeyCode` klase `KeyEvent` koja vraća numerički kod tipke. Na primjer:

```
public void keyPressed(KeyEvent e)
    {
        int kod = e.getKeyCode();
        System.out.println("keyPressed : kod = "+kod);

    }
```

- Numerički kod koji vraća `getKeyCode` metoda je tzv. *virtualni kod* tipke. Klasa `KeyEvent` definiše veliki broj simboličkih konstanti koje predstavljaju te kodove. Sve one počinju s VK_ :

```
VK_A , ...., VK_Z, VK_0, ..., VK_9, VK_COMMA , VK_SHIFT, ...
```

Nastavak...

- Pomoću tih simboličkih imena možemo kontrolisati je li neka specijalna tipka pritisnuta. Ako želimo ispitivati da li je neka kombinacija tipki pritisnuta (npr. SHIFT-CTRL-C) onda nam se nude metode is klase `java.awt.event.InputEvent` koja je nadklasa `KeyEvent klase`. To su metode:
  - `isAltDown()`
  - `isControlDown()`
  - `isMetaDown()`
  - `isShiftDown()`
- Na sledećem slajdu je jednostavan primjer koji ilustruje događaje s tastature i može služiti za njihovo bolje razumijevanje. Treba ipak upozoriti da je direktna obrada dagađaja koje generiše tastatura hardverski zavisno.
- Pošto se metoda `addKeyListener` nalazi u klasi `java.awt.Component` tastaturu može *slušati* svaka *komponenta*. Zato u ovom jednostavnom primjeru nemamo panela, već smo frame pretvorili u *KeyListener*.

Primer za *KeyListener*

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

public class TestKeyListener
{
    public static void main(String[] args)
    {
        MyFrame frame = new MyFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
        frame.setVisible(true);
    }
}
// Svaka komponenta može biti KeyListener pa stoga
// nećemo kreirati panel već će frame slušati tastaturu.
class MyFrame extends JFrame  implements KeyListener
{
    public MyFrame()
    {
        setTitle("Skupljam ulaz s tastature");
        setSize(300,200);
        addKeyListener(this);
    }

    public void keyTyped(KeyEvent e)
    {
        char c = e.getKeyChar();
        System.out.println("keyTyped : znak = "+c);
    }
    public void keyReleased(KeyEvent e)
    {
        int kod = e.getKeyCode();
        System.out.println("keyReleased: kod = "+kod);
    }
    public void keyPressed(KeyEvent e)
    {
        int kod = e.getKeyCode();
        System.out.println("keyPressed : kod = "+kod);
        if(kod == KeyEvent.VK_SHIFT)
            System.out.println("keyPressed : SHIFT pritisnut.");
        if(kod == KeyEvent.VK_C && e.isShiftDown() && e.isControlDown())
             System.out.println("keyPressed : SHIFT_CTRL_C pritisnut");
    }
}
```

Događaji koji dolaze od miša

- Događaji vezani uz miša distribuiraju se osluškivačima koji su podijeljeni u 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` objekta koji ove metode dobijaju moguće je naći koordinate događaja pomoću metoda:

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

Koje se nalaze u `MouseEvent` klasi. Unutar metode `mouseClicked` možemo koristiti metodu **`public int`** `getClickCount()` koja se takodje nalazi u `MouseEvent` klasi koja daje broj klikanja.

Nastavak...

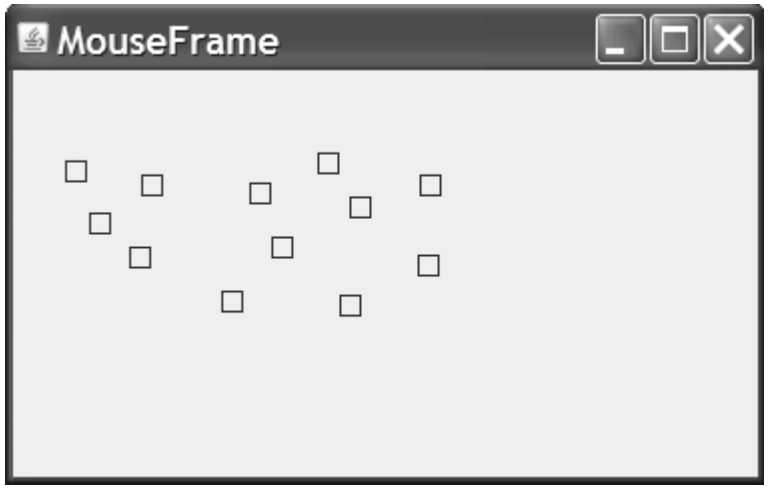
- Isto tako, možemo identifikovati tipku koju smo koristili pomoću metode `public int getButton()` koja vraća jednu od sljedećih konstanti: NOBUTTON, BUTTON1, BUTTON2 ili BUTTON3. To su simbolička imena definisana u `MouseEvent` klasi.
- Ponekad se koriste kombinacije pritiska na tipku miša i neku od 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 bitova koji treba testirati u odnosu na *maske* definisane u `InputEvent` klasi.
- Interfejs **MouseMotionListener** definiše dvije metode:

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

- Prva se odnosi na situaciju kada se miš pomjera s pritisnutom tipkom, a druga na kretanje bez pritisnute tipke. U našem primjeru koristit ćemo drugu metodu za promjenu kursora (vidi klasu `java.awt.Cursor`).

Nastavak...

- Procesiranje događaja koji dolaze od miša može se vidjeti na primjeru koji se nalazi na sledećem 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*).



Program

```
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 manipolaciju 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. Vrati null ako takvog nema.
  public Rectangle2D find(Point2D p)
  {
      // Sav posao odrađuje metod contains iz Rectangle2D koja ispituje je li
      // tačka unutar pravokutnika.
      for(int i=0; i<kvadrati.size(); ++i){
          // moramo castati -- u Java 5.0 mo¾e elegantnije ako koristimo
          // ArrayList<Rectangle2D>
          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 pravokutnika?
            trenutni = find(e.getPoint());
            if(trenutni == null)   // nije
                add(e.getPoint()); // dodaj novi pravokutnik
        }

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

    }
```

```
// 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 pravougaonika
            // bit će 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();
            }
        }
    }
}
```