

# OBJEKTNO PROGRAMIRANJE 2

Oznaka predmeta: OP2  
Predavanje broj: 07  
Nastavna jedinica: JAVA  
Nastavne teme:

Standardni AWT osluškivači. Grupe AWT događaja. Događaji koje generišu komponente. Meniji. Kaskadni meni, CheckboxMenuItem, enumeracija. PopupMenu. Dijalog. Panel. Rasporedi (FlowLayout, cardLayout, BorderLayout, grid Layout, GridBagLayout).

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

## Literatura:

Eckel B., *Thinking in Java*, 2nd edition, Prentice-Hall, New Jersey 2000.

Cay S. Horstmann and Gary Cornell: *"Core Java, Advanced Features", Vol. 2, Prantice Hall, 2013.*

*The Java Tutorial*, Sun Microsystems 2001. <http://java.sun.com>

Branko Milosavljević, Vidaković M, *Java i Internet programiranje*, GInT, Novi Sad 2002.

# Grupe AWT događaja

- AWT događaji se mogu podeliti u 2 grupe:
  - događaje niskog nivoa
    - reprezentuju elementarna zbivanja u sistemu prozora ili elementarne ulaze
  - semantičke događaje
    - rezultat su korisničkih akcija koje su specifične za komponente
- Događaji koje generišu komponente, kontejneri, fokusi i prozori:
  - događaji su niskog nivoa
  - događaji komponenata se generišu:
    - pri promeni pozicije, veličine i vidljivosti
  - događaji kontejnera se generišu:
    - kada se komponenta dodaje ili uklanja iz kontejnera
  - događaji fokusa se generišu:
    - kada komponenta dobija ili gubi fokus tastature (fokus tastature je sposobnost da se prihvate karakteri koji se unose preko tastature)
  - događaji prozora daju informaciju:
    - o bazičnom stanju proizvoljne vrste prozora

# Grupe AWT događaja

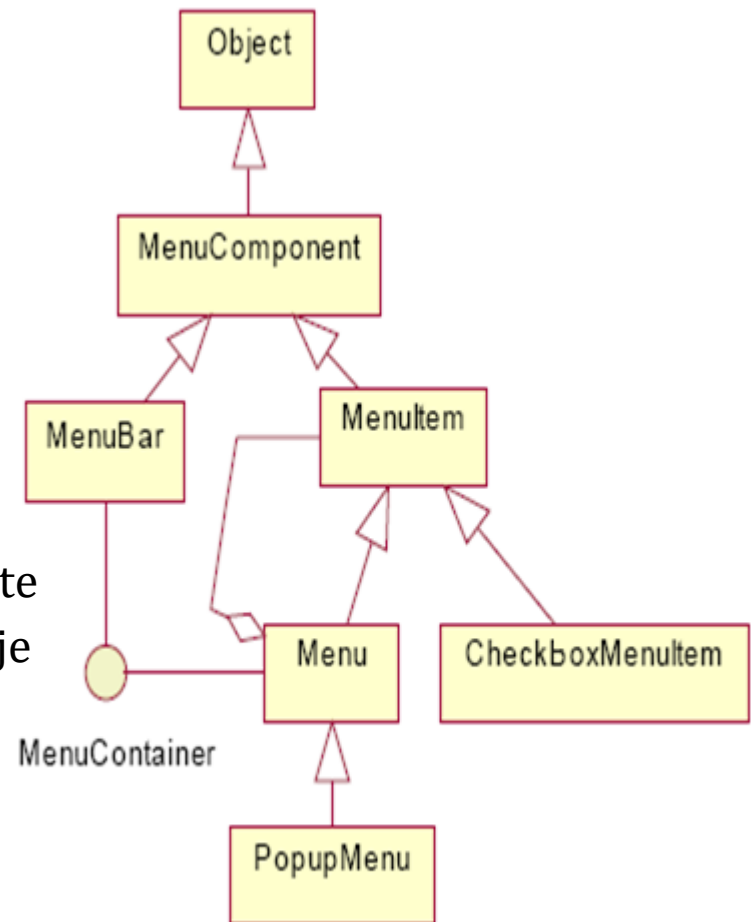
- Elementarni događaji koji potiču od ulaza miša ili tastature:
  - događaji su niskog nivoa
  - događaji miša:
    - događaje kretanja miša
    - pomeranja točkića miša
    - klik, pritisnut/otpušten taster, ušao/izašao u/iz prozora komponente
- Semantički događaji uključuju događaje akcije, prilagođenja, članske i tekstualne događaje:
  - događaje akcije generišu:
    - ekranski tasteri (pritisak), stavke menija, liste i tekst polja
  - događaji prilagođenja se generišu:
    - kada korisnik promeni vrednost klizača ( *scrollbar* )
  - članske događaje generiše
    - izbor jedne od stavki iz liste
  - tekstualni događaji se generišu
    - kada se menja tekst u prostoru za tekst ili u polju za tekst

# Meniji – klase i interfejsi

- Klasa **MenuComponent** je bazna klasa koja sadrži metode za rad sa menijima.
- Klasa **MenuBar** implementira traku menija (pridružuje se prozoru aplikacije)
  - klasa MenuBar se izvodi iz klase MenuComponent
  - objekat klase MenuBar se pridružuje objektu klase Frame
    - metodom setMenuBar() klase Frame
- Klasa **MenuItem** implementira pojedinačne stavke menija
  - klasa MenuItem se izvodi iz klase MenuComponent
  - sadrži metode za o(ne)mogućavanje kao i postavljanje i čitanje labela svojih objekata
- Klasa **Menu** implementira padajuće menije
  - izvedena je iz klase MenuItem
  - objekat klase Menu može sadržati druge MenuItem objekte i tako formirati kaskadne menije
  - klasa Menu sadrži metode za dodavanje objekata klase MenuItem i separatora u objekte klase Menu
  - klasa Menu sadrži i metode za pristup objektima MenuItem unutar objekta Menu
  - objekat klase MenuBar sadrži jedan ili više objekata klase Menu

# Meniji – klase i interfejsi

- Klasa **CheckboxMenuItem** implementira stavke koje mogu biti obeležene potvrdom
    - klasa **CheckboxMenuItem** se izvodi iz klase **MenuItem**
    - sadrži metode koje postavljaju znak potvrde i očitavaju status potvrđenosti
  - Klasa **PopupMenu** implementira “iskačuće” menije
    - izvodi se iz klase **Menu**
    - pojavljuju se na zadatoj poziciji u prostoru iznad odgovarajuće komponente
  - Interfejs **MenuContainer** definiše metode koje moraju implementirati klase koje sadrže objekte vezane za menije
    - Klase
      - **Frame**,
      - **MenuBar**,
      - **Menu**
- implementiraju interfejs **MenuContainer**



# Kreiranje menija i obrada događaja

- Meni aplikacije se kreira tako što se:
  - kreira objekat **MenuBar** i kreiraju objekti **Menu**
  - dodaju objekti **Menu** u objekat **MenuBar** pozivom metoda *theMenuBar.add( theMenu )*
  - dodaju objekti **MenuItem** objektu klase **Menu** pozivom metoda npr. *theMenu.add( String )*
  - postavi meni prozora aplikacije pozivom *theFrame.setMenuBar( theMenuBar )*
- **Stara** tehnika (mala je verovatnoća da se naiđe na ovaj slučaj) obrade događaja iz menija
  - zasniva se na prepoznavanju **ACTION\_EVENT**
  - u metodi *handleEvent ()* se najpre testira se da li je u pitanju događaj **ACTION\_EVENT**
  - zatim se testira da li je cilj događaja (komponenta nad kojom se dogodio) tipa **MenuItem** :
    - *theEvent.target instanceof MenuItem*

# Primer menija

- **Nova tehnika** obrade događaja iz menija
  - zasniva se na interfejsu ActionListener
  - klasa koja hvata događaje iz menija treba da implementira interfejs ActionListener
  - klasa se registruje kao slušalac objekta određenog menija
    - *theMeni.addActionListener( this );*
  - piše se metod **public void actionPerformed (ActionEvent e)**
  - u metodi actionPerformed ime aktivirane stavke menija se dobija kao:
    - *e.getActionCommand()*

```
import java.awt.*;
import java.awt.event.*;
public class PrimerMenija extends Frame implements ActionListener{
    String izborIzMenija = "Izaberite stavku iz menija...";
    public PrimerMenija() {
        super("Primer Menija");
        setSize(300,200);
        dodajMenije();
        setVisible(true);
    }
}
```

# Primer menija

```
void dodajMenije() {  
    MenuBar trakaMenija = new MenuBar();  
    Menu prviMeni = new Menu("Prvi meni");  
    Menu drugiMeni = new Menu("Drugi meni");  
    prviMeni.add("Prvi meni, prva stavka");  
    prviMeni.add("Prvi meni, druga stavka");  
    prviMeni.add("Kraj");  
    prviMeni.addActionListener(this);  
    drugiMeni.add("Drugi meni, prva stavka");  
    drugiMeni.add("Drugi meni, druga stavka");  
    drugiMeni.addActionListener(this);  
    trakaMenija.add(prviMeni);  
    trakaMenija.add(drugiMeni);  
    setMenuBar(trakaMenija);  
}  
  
public void paint(Graphics g) {  
    g.drawString(izborIzMenija, 50, 100);  
}
```



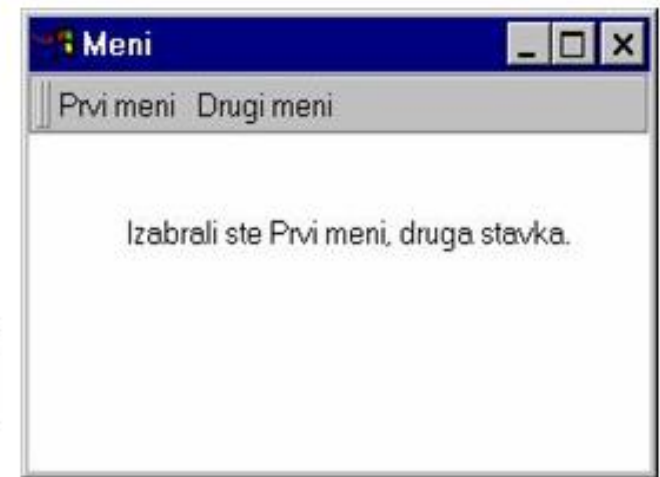
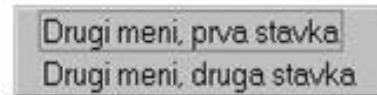
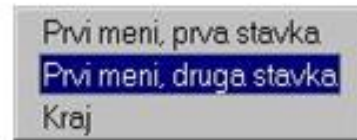
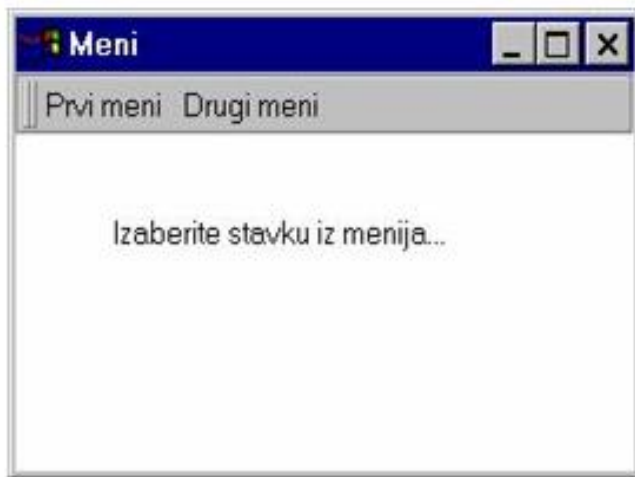
# Primer menija

```
public void actionPerformed (ActionEvent e) {  
    String komanda=e.getActionCommand();  
    if(komanda.equals("Kraj"))  
        System.exit(0);  
    else{  
        izborIzMenija = "Izabrali ste "+komanda+".";   
        repaint();  
    }  
}  
  
public static void main(String args[]){  
    PrimerMenija prozor = new PrimerMenija();  
}
```

- Metod **repaint()** se koristi da se prozor ponovo iscrta
  - ovaj metod izaziva poziv metoda **paint()**
  - metod **paint()** se poziva kada se koriste metodi:
    - **setVisible(true)**
    - **show()** - zastareo,
    - **repaint()** - otvara nit koja zove **update**
    - **update()** - dobra za animaciju, briše stari sadržaj, a onda zove **paint**.

# Primer menija

- Izlaz



- Sledeći primer ilustruje korišćenje:
  - klase `CheckboxMenuItem`,
  - enumeracije,
  - kaskadnog menija,te interfejsa:
  - `ActionListener`
  - `ItemListener`koji su implementirani u istoj klasi.

# Kaskadni meni, CheckboxMenuItem, enumeracija

```
import java.awt.*;
import java.awt.event.*;
public class PrimerMenija2
    extends Frame
    implements ActionListener, ItemListener{
    private String izbor = "Izaberite stavku menija...";
    private CheckboxMenuItem cbmi;
    public enum MyMenu{ //malo vezbanja enumeracije
        IME,PREZIME,KRAJ,BROJ_INDEXA,BONUS_1,BONUS_2,BONUS_3;
    }
    public PrimerMenija2() {
        super("Primer Menija");
        setSize(600,250);
        addWindowListener(new WindowAdapter()
            {
                public void windowClosing(WindowEvent we){
                    System.exit(0);
                }
            }
        );
        dodajMenije();
        setVisible(true);
    }
}
```

# Kaskadni meni, CheckboxMenuItem, enumeracija

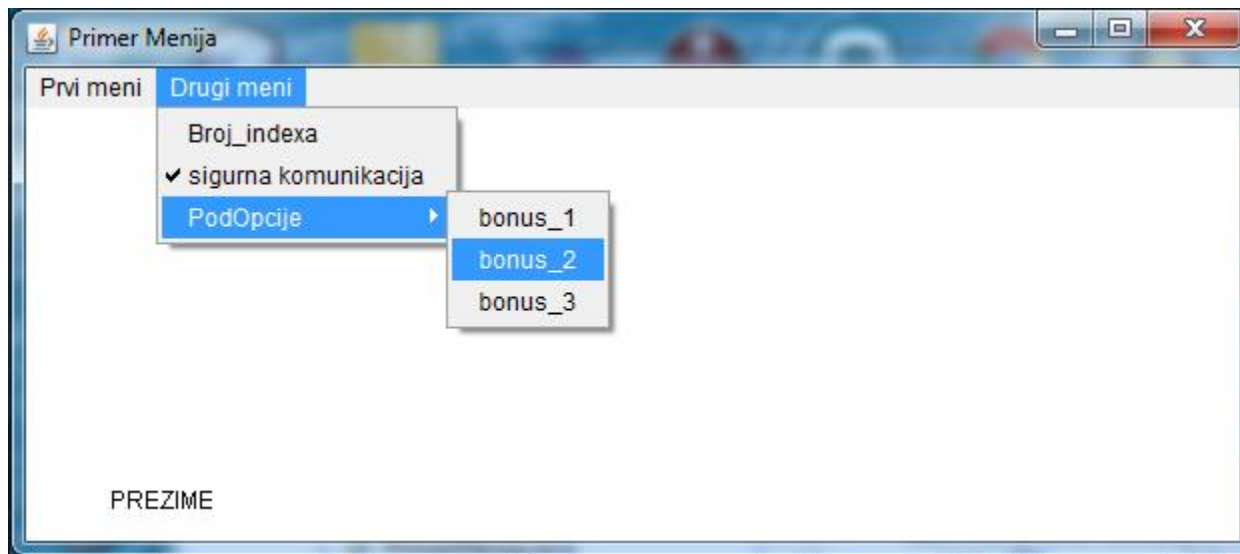
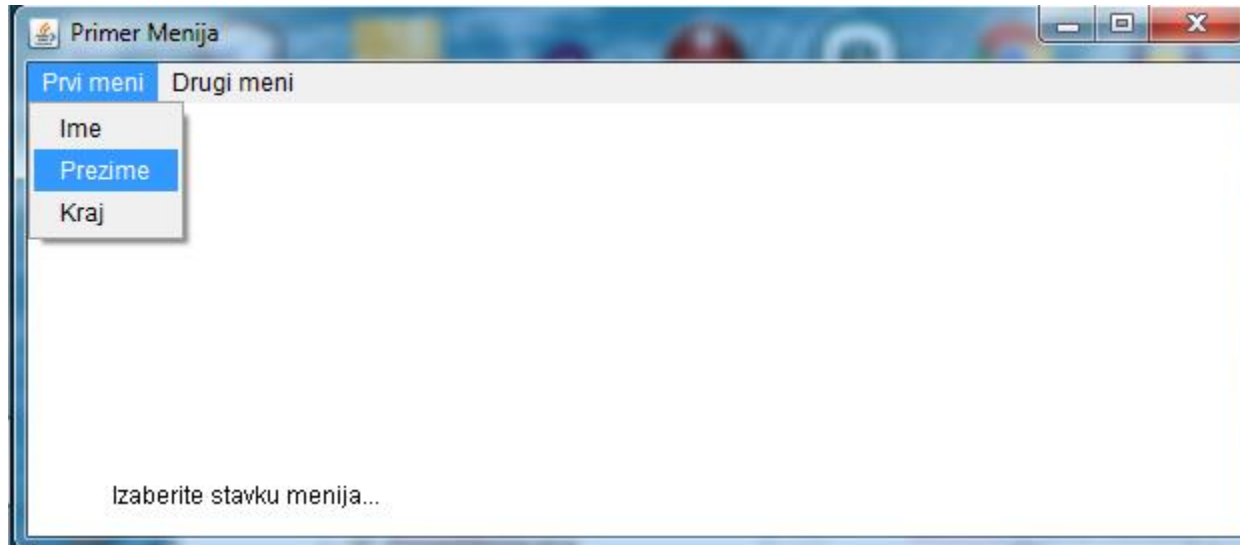
```
void dodajMenije() {  
    MenuBar trakaMenija = new MenuBar();  
    Menu prviMeni = new Menu("Prvi meni");  
    Menu drugiMeni = new Menu("Drugi meni");  
    prviMeni.add("Ime");  
    prviMeni.add("Prezime");  
    prviMeni.add("Kraj");  
    prviMeni.addActionListener(this);  
    drugiMeni.add("Broj_indexa");  
    cbmi=new CheckboxMenuItem("sigurna komunikacija");  
    cbmi.setState(true);  
    cbmi.addItemListener(this);  
    drugiMeni.add(cbmi);  
    Menu podmeni = new Menu("PodOpcije"); //kaskadni meni, podopcije  
    podmeni.add("bonus_1");  
    podmeni.add("bonus_2");  
    podmeni.add(new MenuItem("bonus_3")); //preko objekta  
    podmeni.addActionListener(this);  
    drugiMeni.add(podmeni);  
    drugiMeni.addActionListener(this);  
    trakaMenija.add(prviMeni);      trakaMenija.add(drugiMeni);  
    setMenuBar(trakaMenija);  
}
```

# Kaskadni meni, CheckboxMenuItem, enumeracija

```
public void paint(Graphics g) {
    g.drawString(izbor, 50, 250);
}
public void actionPerformed (ActionEvent e) {
    String komanda=e.getActionCommand();
    switch(MyMenu.valueOf(komanda.toUpperCase())){
        case IME          : izbor = "IME"; break;
        case PREZIME      : izbor = "PREZIME"; break;
        case KRAJ         : System.exit(0); break;
        case BROJ_INDEXA  : izbor = "BROJ INDEXA"; break;
        default           : izbor = komanda;
    } //mozete promeniti kod
    repaint();
}
public void itemStateChanged(ItemEvent e) {
    izbor = "sigurna komunikacija = " + cbmi.getState();
    repaint();
}
public static void main(String args[]){
    PrimerMenija2 prozor = new PrimerMenija2();
}
}
```

# Izlaz

- Izlaz:



# PopupMenu (iskaćući meni)

- Ideja je da se implementiraju intefejsi `MouseListener` i `ActionListener` koji su potrebni za događaje klik mišem suprotnim tasterom i odabiranje stavke iz padajućeg menija respektivno.
  - Dodate su dve stavke u iskaćući meni (popup option 1 i popup option 2) kome je dodat osluškivač `ActionListener`.
  - U metodi `mouseClicked` ispituje se da li je događaj potekao od suprotnog tastera miša i ako jeste aktivira se iskaćući meni na poziciji miša gde je nastupio ovaj događaj. Metoda `equalsIgnoreCase` klase `String` vrši poređenje dva stringa pri čemu se ignoriše veličina znakova.

```
import java.awt.*;
import java.awt.event.*;
public class Primer_PopuMenu
    extends Frame
    implements MouseListener, ActionListener {

    private PopuMenu popup = new PopuMenu();
    private String t="Čeka se na događaj ...";

    public Primer_PopuMenu(){
        super("Osluskivac misa");
        setSize(300,100);
    }
}
```

# PopupMenu (iskaćući meni)

```
MenuItem mi = new MenuItem("popup option 1");
popup.add(mi);
mi = new MenuItem("popup option 2");
popup.add(mi);
popup.addActionListener(this);
add(popup);
addMouseListener(this);
setVisible(true);
}

public void paint(Graphics g){
    g.drawString(t,50,50);
}

public void mouseClicked (MouseEvent d){
    if (d.getButton()==MouseEvent.BUTTON3)
        popup.show(this, d.getX(), d.getY());

    t="clicked"; repaint(); }

public void mouseEntered (MouseEvent d){
    t="entered"; repaint(); }

public void mouseExited (MouseEvent d){
    t="exited" ; repaint(); }

public void mousePressed (MouseEvent d){
    t="pressed"; repaint(); }

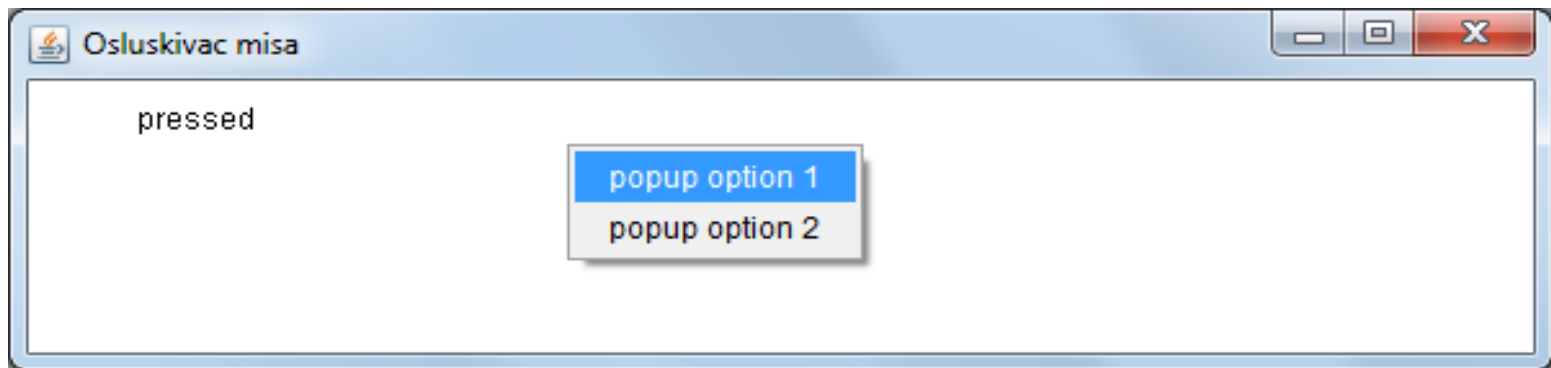
public void mouseReleased(MouseEvent d){
    t="released";repaint(); }
```



# PopupMenu (iskaćući meni)

```
public void actionPerformed(ActionEvent ae){  
    if(ae.getActionCommand().equalsIgnoreCase("popup option 2"))  
        System.exit(0);  
    if(ae.getActionCommand().equalsIgnoreCase("popup option 1"))  
        t="odabrana opcija \"popup option 1\" iskacuceg menija";  
        repaint();  
}  
public static void main(String[] args){  
    Primer_PopupMenu d=new Primer_PopupMenu();  
}  
}
```

- Izlaz:



# Kreiranje dijaloga

- Klasa Dialog implementira prozore dijaloga kroz koje se obavlja komunikacija sa korisnikom.
- Dva tipa dijaloga se mogu kreirati:
  - modalni dijalozi
    - dok su otvoreni, fokus se ne može preneti na druge prozore aplikacije
  - nemodalni dijalozi
    - fokus se može preneti i na druge prozore aplikacije dok su otvoreni
- Dijalog se kreira sledećim konstruktorom:

```
public Dialog ( Frame roditelj,  
               String naslov,  
               boolean modalni )
```
- Roditelj je glavni prozor aplikacije.
- Nakon kreiranja dijaloga, ovaj se može otvarati i zatvarati pozivom metode setVisible(Boolean).
- U sledećem primeru dat je dijalog čiji je roditelj glavni prozor aplikacije, naslov je Dijalog, dijalog je nemodalni a klasa Dijalog realizovana je kao unutrašnja klasa klase PrimerDijaloga.

# Primer dijaloga

```
import java.awt.*;
import java.awt.event.*;
public class PrimerDijaloga
    extends Frame
    implements ActionListener{

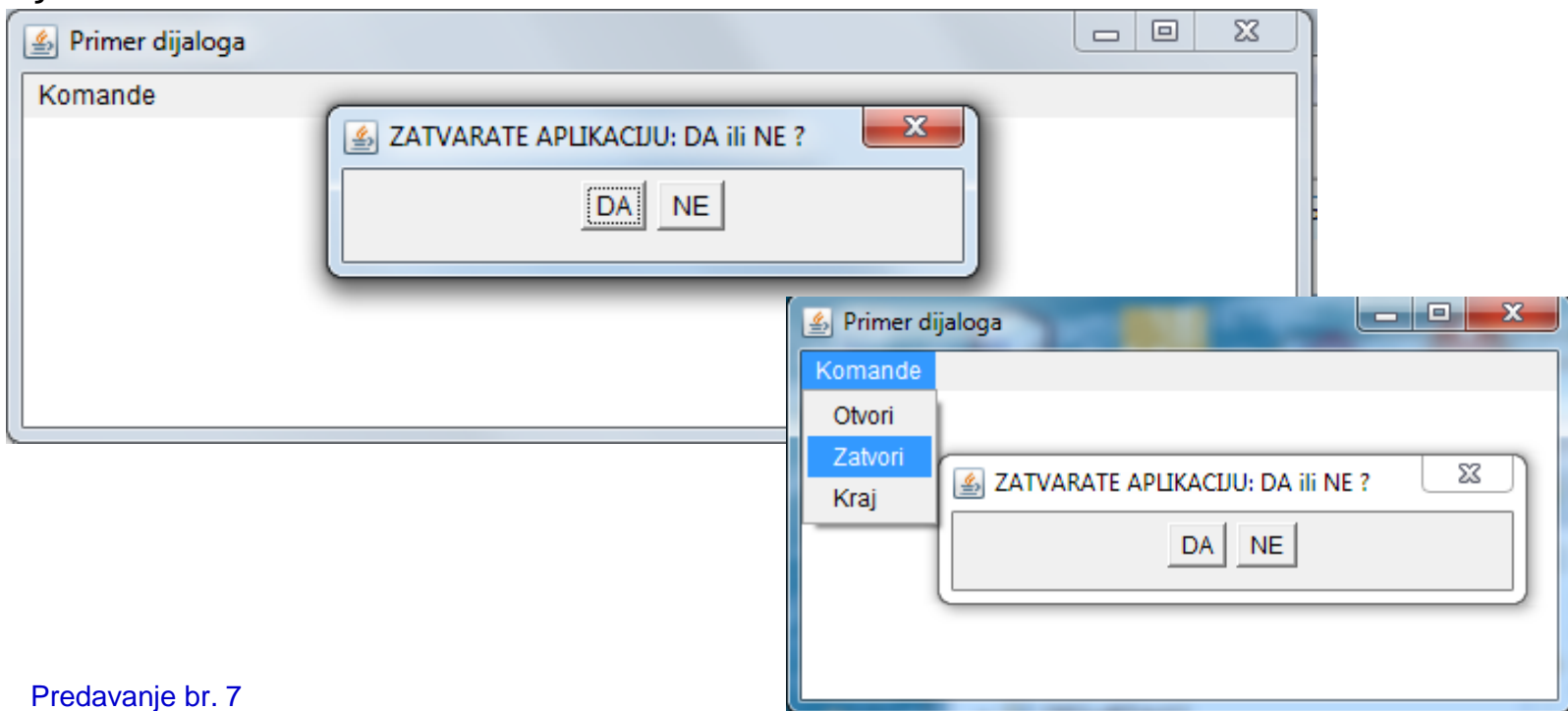
    class Dijalog extends Dialog {
        private Button da = new Button("DA");
        private Button ne = new Button("NE");
        Dijalog(Frame roditelj) {
            super(roditelj, "ZATVARATE APLIKACIJU: DA ili NE ?", false);
            setSize(300,80);
            addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent we){ setVisible(false); }
            });
            this.setLayout(new FlowLayout()); //rasporedjivanje komponenti
            da.setActionCommand("Zatvori aplikaciju");
            add(da);
            da.addActionListener((PrimerDijaloga)roditelj);
            add(ne);
            ne.addActionListener((PrimerDijaloga)roditelj);
        }
    }
}
```

# Primer dijaloga

```
private Dialog dijalog;  
public PrimerDijaloga() {  
    super("Primer dijaloga"); setSize(400,400);  
    dodajMenije();  
    dijalog = new Dialog(this);  
    setVisible(true);  
}  
void dodajMenije() {  
    MenuBar trakaMenija = new MenuBar();  
    Menu meni = new Menu("Komande");  
    meni.add("Otvori");  
    meni.add("Zatvori");  
    meni.add("Kraj");  
    meni.addActionListener(this);  
    trakaMenija.add(meni);  
    setMenuBar(trakaMenija);  
}  
public void actionPerformed (ActionEvent e) {  
    String komanda=e.getActionCommand();  
    if ( (komanda.equals("Zatvori aplikaciju")) ||  
        (komanda.equals("Kraj") ) )  
        System.exit(0);  
}
```

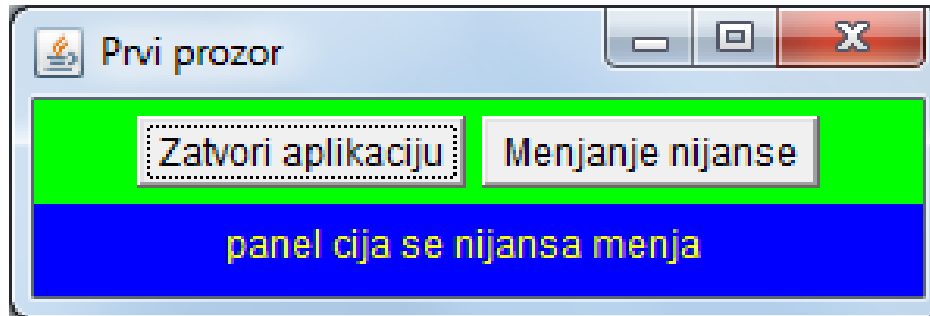
# Primer dijaloga

```
else
    if ( (komanda.equals("NE")) ||
        (komanda.equals("Zatvori")))
        dijalog.setVisible(false);
    else dijalog.setVisible(true);
}
public static void main(String args[]){
    PrimerDijaloga prozor = new PrimerDijaloga();
}
}
```



# Panel

- Klasa Panel služi za organizovanje komponenti u prozoru:
  - izvedena je iz klase Container
  - predstavlja najjednostavniju kontejnersku komponentu
  - predstavlja prostor u koji se mogu smeštati druge komponente
  - komponente koje se smeštaju na panele uključuju i druge panele
  - prima događaje prouzrokovane:
    - mišem,
    - tastaturom i
    - promenom fokusa
- Paneli se definišu, postavlja im se pozadina, dodaju im se komponente (npr. ekranski tasteri).
- Sledi primer panela koji sadrži dva panela:



# Primer panela

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

public class PrimerPanela extends Frame implements ActionListener {
    private Label labela = new Label("panel cija se nijansa menja");
    private Button kraj = new Button("Zatvori aplikaciju");
    private Button menjajnijansu = new Button("Menjanje nijanse");
    private Panel gornjipanel = new Panel();
    private Panel donjipanel = new Panel();

    public PrimerPanela(String naziv) {
        super(naziv);
        setSize(300, 100);
        dodajPanele();
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { dispose(); }
        });
        kraj.setActionCommand("KRAJ");
        kraj.addActionListener(this);
        menjajnijansu.setActionCommand("MENJAJ");
        menjajnijansu.addActionListener(this);
        setVisible(true);
    }
}
```

# Primer panela

```
void dodajPanele() {
    gornjipanel.setBackground(Color.green);
    gornjipanel.add(kraj);
    gornjipanel.add(menjajnijansu);
    add("North", gornjipanel);
    donjipanel.setBackground(new Color(0, 0, 255));
    labela.setForeground(Color.yellow);
    donjipanel.add(labela);
    add("South", donjipanel);
}

public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("KRAJ")) System.exit(0);
    else if (e.getActionCommand().equals("MENJAJ")) {
        Color nijansa = new Color(0, 0, (int) (Math.random() * 256));
        donjipanel.setBackground(nijansa);
        labela.setBackground(nijansa);
    }
}

public static void main(String args[]) {
    PrimerPanela prozor1 = new PrimerPanela("Prvi prozor");
    PrimerPanela prozor2 = new PrimerPanela("Drugi prozor");
}}
```



# Rasporedi

- Komponente u kontejneru raspoređuje odgovarajući upravljač rasporeda ( *layout manager* )
- Svaka klasa upravljača rasporeda mora implementirati interfejs
  - `LayoutManager`
- Klase upravljača rasporeda su:
  - **FlowLayout** koja komponente u kontejneru raspoređuje po redovima u nizovima sleva-udesno
  - **BorderLayout** koja komponente u kontejneru raspoređuje po ivicama i u sredini kontejnera
  - **CardLayout** koja komponente u kontejneru raspoređuje kao jednu iza druge (kao špil karata)
  - **GridLayout** koja komponente u kontejneru raspoređuje matrično
  - **GridBagLayout** koja komponente u kontejneru raspoređuje prema skupu objekata `GridBagConstraints`
  - **null-layout** koja omogućuje da programer eksplicitno odredi poziciju i veličinu komponente

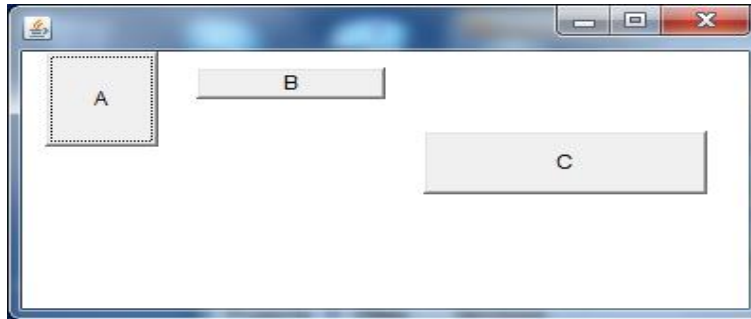
# Raspored nullLayout

- Primer null-layouta u kome se kreiraju tri programska dugmeta kojima se određuje pozicija i veličina na kanvasu:

```
import java.awt.*;
import java.awt.event.*;
public class NullLayoutPrimer extends Frame{
    private Button b1, b2, b3;
    NullLayoutPrimer(){
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e){ System.exit(0); }
        });
        setSize(400,200);
        setLayout(null);
        b1 = new Button("A"); b2 = new Button("B"); b3 = new Button("C");
        add(b1);          add(b2);          add(b3);
        b1.setLocation( 20,30); b1.setSize( 60,60);
        b2.setLocation(100,40); b2.setSize(100,20);
        b3.setLocation(220,80); b3.setSize(150,40);
        //b3.reshape(220, 80, 150, 40); //zastareo nacin
        setVisible(true);
    }
    public static void main(String str[]) {
        NullLayoutPrimer nlp = new NullLayoutPrimer(); } }
```

# Rasporedi: flow, card, border, grid

Izlaz:



- U primeru koji sledi koriste se rasporedi: flowLayout, card Layout, border Layout i grid Layout.

```
import java.awt.*;
import java.awt.event.*;
public class PrimerRasporeda extends Frame {
    Button menjajkarte = new Button("sledeca karta");
    CardLayout cardlayout = new CardLayout();
    int indekskarte = 0;
    String nazivkarte[] = {"prva karta", "druga karta",
                           "treca karta", "cetvrta karta", "peta karta"};

    Panel flow    = new Panel();
    Panel card    = new Panel();
    Panel border  = new Panel();
    Panel grid    = new Panel();
```

# Raspoledi: flow, card, border, grid

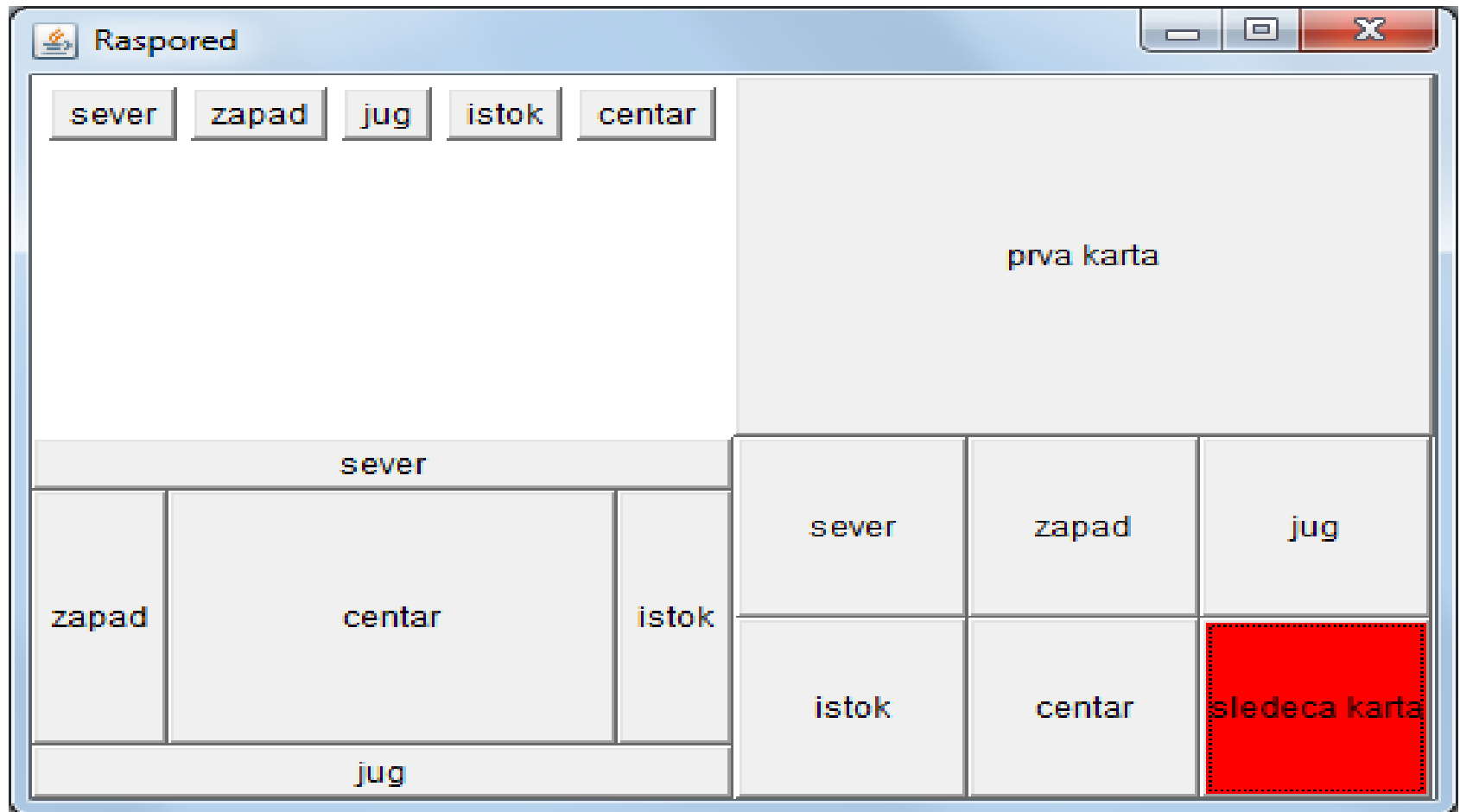
```
public PrimerRasporedi() {  
    super("Rasporedi");  
    dodajPanel();  
    setSize(500,350);  
    addWindowListener(new WindowAdapter() {  
        public void windowClosing(WindowEvent e){ System.exit(0); }  
    });  
    setVisible(true);  
}  
  
void dodajPanel() {  
    setLayout(new GridLayout(2,2));  
    dodajTastere(flow);  
    card.setLayout(cardlayout);  
    for (int i = 0; i < nazivkarte.length; i++)  
        card.add(nazivkarte[i],new Button(nazivkarte[i]));  
    cardlayout.show(card, nazivkarte[0]);  
    flow.setLayout(new FlowLayout());  
    border.setLayout(new BorderLayout());  
    grid.setLayout(new GridLayout(2,3));  
    dodajTastere(border);  
    dodajTastere(grid);  
}
```

# Rasporedi: flowLayout, cardLayout, BorderLayout, grid Layout

```
menjajkarte.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
        if (++indekskarte == nazivkarte.length)
            indekskarte = 0;
        cardlayout.show(card, nazivkarte[indekskarte]);
    }
});
menjajkarte.setBackground(Color.red);
grid.add(menjajkarte);
add(flow);    add(card);
add(border); add(grid);
}
void dodajTastere(Panl panel){
    panel.add("North" ,new Button("sever" ));
    panel.add("West"  ,new Button("zapad" ));
    panel.add("South" ,new Button("jug"   ));
    panel.add("East"  ,new Button("istok" ));
    panel.add("Center",new Button("centar"));
}
public static void main(String args[]){
    PrimerRasporeda prozor = new PrimerRasporeda();
}
}
```

# Rasporedi: flowLayout, cardLayout, BorderLayout, grid Layout

- Izlaz:



# Raspoređivač GridBagLayout

- Raspoređivačem GridBagLayout raspoređuje se npr. 5 programskih dugmadi tako da:
  - prva tri budu u nultom redu (gridy) redom u koloni (gridx) 0, 1 i 2 respektivno.
  - četvrto programsko dugme se nalazi u prvom redu i proteže se na sve tri kolone.
  - peto programsko dugme je postavljeno dole desno i proteže se na kolone 1 i 2.
  - Postavljanjem atributa za horizontalno popunjavanje na maksimalnu širinu sva programska dugmad će menjanjem veličine frejma menjati i svoju dužinu horizontalno popunjavajući pripadne kolone.

# Raspoređivač GridBagLayout

```
import java.awt.*;
import java.awt.event.*;
public class GridBagLayoutPrimer extends Frame {
    final boolean shouldFill = true;
    final boolean shouldWeightX = true;
    public GridBagLayoutPrimer() {
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
        Button button;
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setLayout(gridbag);
        if (shouldFill) {
            //horizontalno popunjavanje na maksimalnu sirinu
            c.fill = GridBagConstraints.HORIZONTAL;
        }
        button = new Button("Elvis");
        if (shouldWeightX) {
            //distribucija prostora izmedju kolona
            c.weightx = 0.5;
        }
    }
}
```



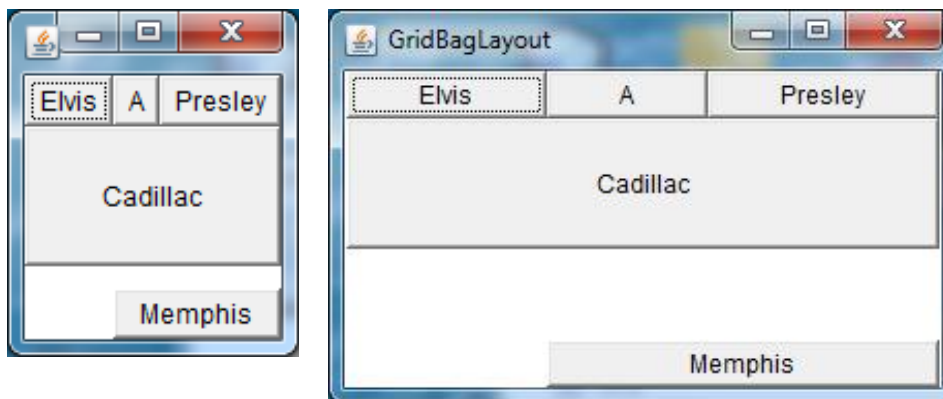
# Raspoređivač GridBagLayout

```
c.gridx = 0;
c.gridy = 0;
gridbag.setConstraints(button, c);
add(button);
button = new Button("A");
c.gridx = 1;
c.gridy = 0;
gridbag.setConstraints(button, c);
add(button);
button = new Button("Presley");
c.gridx = 2;
c.gridy = 0;
gridbag.setConstraints(button, c);
add(button);
button = new Button("Cadillac");
c.ipady = 40;           //povecanje visine
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
gridbag.setConstraints(button, c);
add(button);
```

# Raspoređivač GridBagLayout

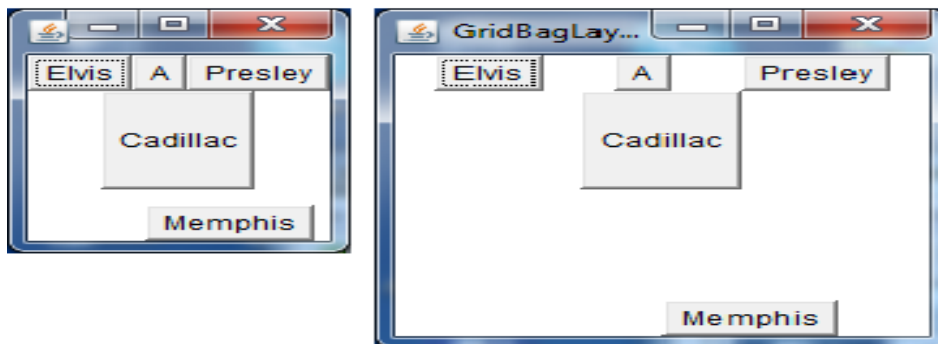
```
button = new Button("Memphis");
c.ipady = 0;           //podrazumevana visina
c.weighty = 1.0;       //dodavanje vertikalnog razmaka
c.anchor = GridBagConstraints.SOUTH; //na juznu oblast
c.insets = new Insets(10,0,0,0); //razmak oko komponente
c.gridx = 1;           //poravnanje sa dugmetom A
c.gridwidth = 2;       //sirina 2 kolone
c.gridy = 2;           //treci red
gridbag.setConstraints(button, c);
add(button);
}
public static void main(String args[]) {
    GridBagLayoutPrimer win = new GridBagLayoutPrimer();
    win.setTitle("GridBagLayout"); win.pack(); win.setVisible(true);
}}
```

Sa horizontalnim popunjavanjem, sa distribucijom prostora između kolona 0.5:



# Raspoređivač GridBagLayout

- Bez horizontalnog popunjavanja, sa distribucijom prostora između kolona 0.5:



- Sa horizontalnim popunjavanjem, sa podrazumevanom distribucijom prostora između kolona):



- Bez horizontalnog popunjavanja, sa podrazumevanom distribucijom prostora između kolona):

