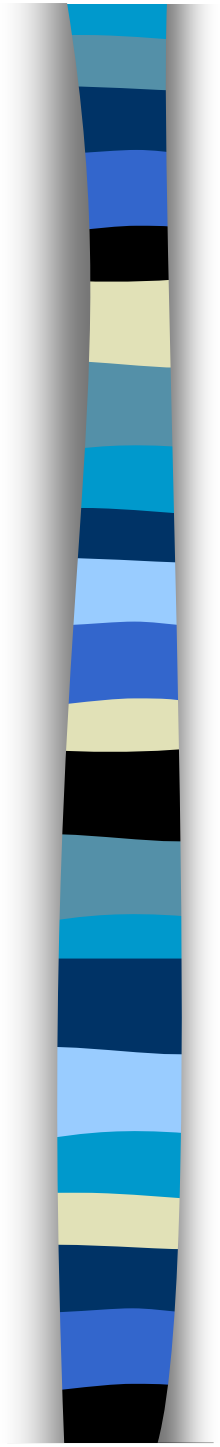


JavaFX

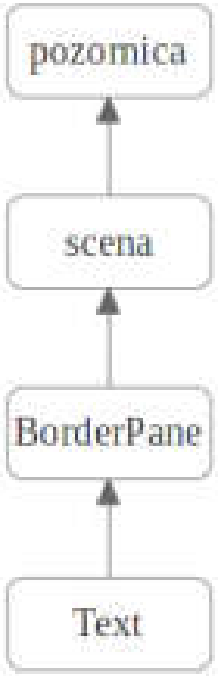
Struktura JavaFX aplikacije

- JavaFX aplikacija ima stukturu stabla, u čijem korenu se nalazi **pozornica** (eng. **stage**).
- U praksi, ako je aplikacija pokrenuta na desktop operativnom sistemu, pozornica će biti prozor; ukoliko je aplikacija pokrenuta u okviru veb stranice, pozornica će biti aplet.
- Dakle, pozornica zavisi od konkretnog operativnog sistema, odnosno uređaja na kom se aplikacija izvršava.
- Bitno je naglasiti da programer veći deo vremena ne mora voditi računa o tome gde se njegova aplikacija izvršava, što je prednost koju JavaFX nudi u odnosu na mnoge druge grafičke biblioteke.

Struktura JavaFX aplikacije

- Pozornica sadrži tačno jednu **scenu** (eng. ***scene***), koja, dalje, sadrži **čvorove** (eng. ***nodes***).
- Čvor je najčešće vizuelna komponenta, poput dugmeta ili polja za unos teksta, ali može biti i slika, multimedijalni plejer, itd.
- Glavna klasa nasleđuje klasu Application i implementira njen metod ***start*** koji prihvata jedan parametar tipa Stage - pozornicu aplikacije. Inicijalizacija JavaFX aplikacija se uvek vrši u ovom metodu.
- U narednom primeru scena sadrži jedan čvor tipa BorderPane, koji, dalje, sadrži čvor tipa Label, što je vizuelna komponenta za prikaz statičkog teksta.

Struktura JavaFX aplikacije



Struktura JavaFX aplikacije

Primer 12.1: Primer jednostavne JavaFX aplikacije.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
  @Override
  public void start(Stage stage) throws Exception {
    Label txt = new Label("Hello, JavaFX!");
    BorderPane pane = new BorderPane(txt);
    Scene scene = new Scene(pane, 320, 240);
    stage.setScene(scene);
    stage.show();
  }

  public static void main(String[] args) {
    launch(args); // staticki metod klase Application
  }
}
```

Struktura JavaFX aplikacije

- Scena se postavlja pozivom metoda **setScene** nad objektom klase **Stage**.
- Da bi pozornica bila vidljiva na ekranu, potrebno je pozvati njen metod **show**.
- Kao što je prikazano u primeru, scena najčešće ne sadrži vizuelne komponente direktno.
- Uobičajeni pristup je da se u scenu najpre postavi okno (eng. **pane**), a zatim da se komponente postavljaju u njega.

Dizajniranje korisničkog interfejsa

- Veoma bitan aspekt programiranja grafičkih korisničkih interfejsa predstavlja raspoređivanje komponenti (tj. čvorova) na ekranu, odnosno određivanje njihovih veličina i pozicija.
- Prilikom dodavanja čvora možemo sami zadati tačnu poziciju i veličinu svakog. Iako pogodan u nekim slučajevima, ovaj pristup se ne preporučuje u realnim aplikacijma.
- Naime, problem je što su ovako postavljeni čvorovi statični - neće se pomerati niti će menjati svoju veličinu ako korisnik menja veličinu prozora. Stoga, JavaFX nudi okna, objekte koji vrše automatsko raspoređivanje čvorova po sceni.
- **Okna** se prilagođavaju dimenzijama scene, tako da se veličine i raspored čvorova menjaju sa promenom veličine same scene. Na ovaj način se značajno olakšava razvoj interaktivnih grafičkih aplikacija.

Dizajniranje korisničkog interfejsa

- JavaFX čvorovi nemaju samo jedno polje za veličinu, već tri. Tj. prilikom rada sa čvorovima moguće je zadati:
  - **minimalnu,**
  - **željenu (eng. preferred), i**
  - **maksimalnu veličinu.**
- Iako tačno ponašanje zavisi od konkretnog okna, željena veličina je veličina koju će čvor imati ukoliko na sceni ima dovoljno mesta. Kada korisnik smanjuje prozor, okno će smanjivati čvorove, ali ne ispod zadatih minimalnih vrednosti.
- Ako su minimalne veličine takve da nema dovoljno mesta za sve čvorove, može doći do njihovog preklapanja.
- Slično, kada korisnik povećava prozor, neka okna će povećavati čvorove, ali ne iznad maksimalnih vrednosti.

Dizajniranje korisničkog interfejsa

- Generalno, kada kreiramo čvor, sve tri veličine se automatski inicijalizuju na određene vrednosti.
- Na primer, ako kreiramo dugme, njegova željena i maksimalna veličina će biti jednake, tolike da je unutar dugmeta moguće prikazati ceo tekst.
- Minimalna veličina će biti tolika da je u dugmetu moguće prikazati tekst "…" i okno ne sme smanjiti dugme ispod toga.
- Sa druge strane, maksimalna veličina polja za unos teksta će automatski biti postavljena na neograničenu vrednost, te će ono zauzimati maksimalan dostupan prostor.

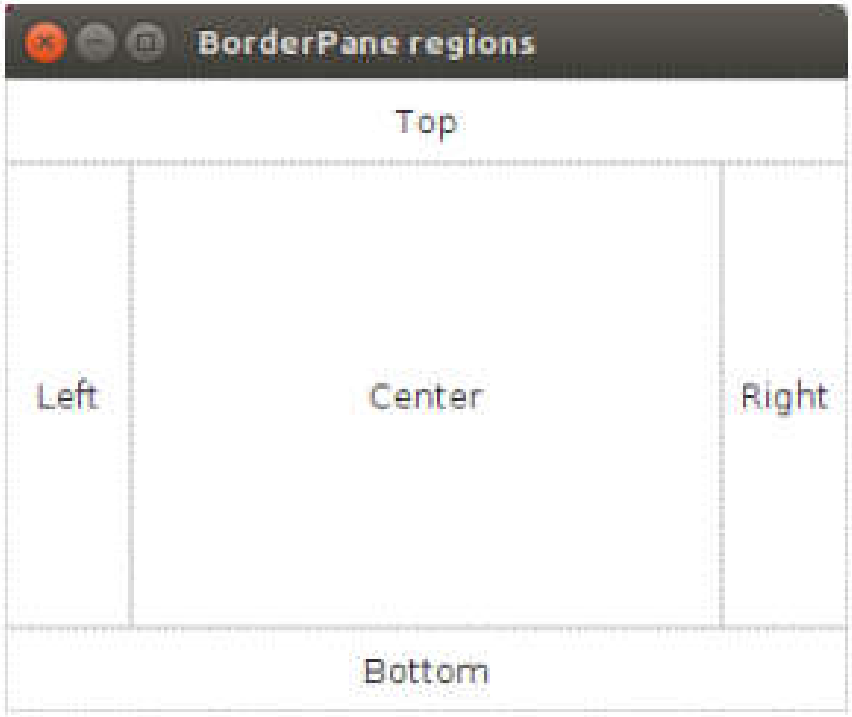
Dizajniranje korisničkog interfejsa

- Vrednosti za veličine čvorova možemo postaviti pozivom metoda
  - **`set*Width,`**
  - **`set*Height i`**
  - **`set*Size,`**

  gde * može biti **`Min`**, **`Pref`** i **`Max`** za, redom, minimalnu, željenu i maksimalnu vrednost.
- Veličine su tipa Double; za neograničenu vrednost koristimo konstantu **`Double.MAX_VALUE`**.

BorderPane

- BorderPane deli scenu na 5 regiona, od čega su 4 ivična (u oznakama Top, Bottom, Left i Right), a 1 centralni (u oznaci Center). Svaki region može sadržati najviše jedan čvor.



BorderPane

- Paket: javafx.scene.layout
- Konstruktori
  - BorderPane()
  - BorderPane(Node center)
  - BorderPane(Node c, Node t, Node r, Node b, Node l)
- Metode
  - void setCenter(Node n)
  - void setLeft(Node n)
  - void setRight(Node n)
  - void setTop(Node n)
  - void setBottom(Node n)
  - void setPadding(Insets value)
    - Postavlja marginu (prazan prostor) oko okna na vrednost definisanu Insets objetkom *value*
    - Konstruktori klase Insets
      - Insets(double topRightBottomLeft) – sve margine identicne
      - Insets(double top, double right, double bottom, double left)

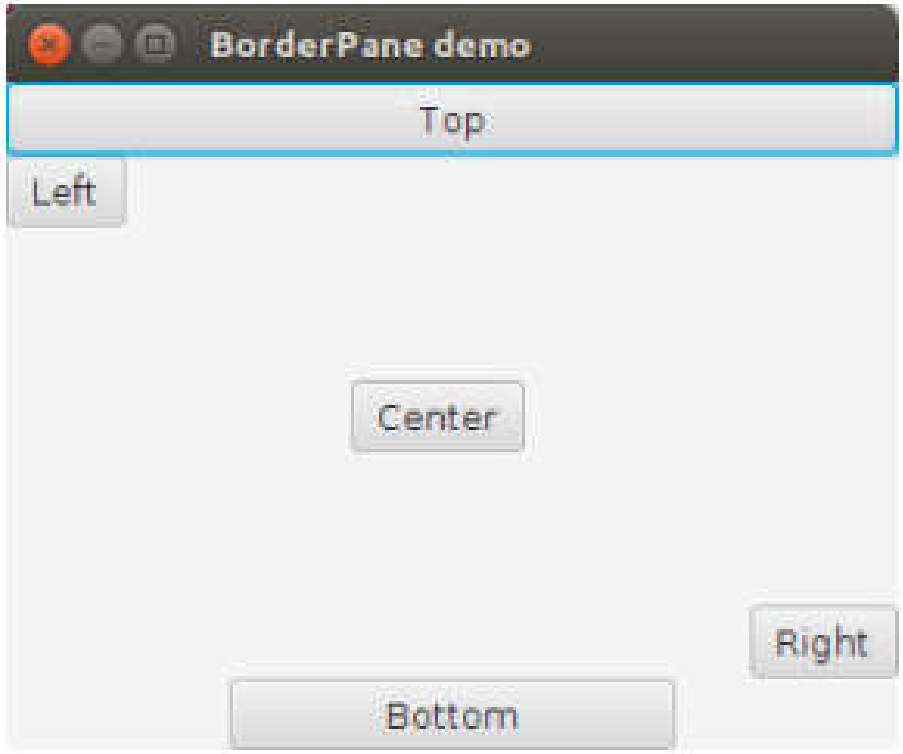
BorderPane

- Veličine ivičnih regiona BorderPane-a zavise od veličina čvorova.
- Čvor u donjem i gornjem regionu će imati visinu jednaku željenoj vrednosti, dok će mu širina biti maksimalna.
- Slično, čvor u levom i desnom regionu će imati širinu jednaku željenoj vrednosti, dok će mu visina biti maksimalna.
- Ivični regioni koji ne sadrže komponente nisu vidljivi.
- Centralni region zauzima sav preostali prostor, i njegov čvor ima maksimalnu dozvoljenu veličinu.

Statički metodi klase BorderPane

- **static void setMargin(Node child, Insets value)**
  - Postavlja marginu (prazan prostor) oko deteta *child* unutar dela okna u kome se child nalazi na vrednost definisanu Insets objektom *value*
- **static void setAlignment(Node child, Pos value)**
  - Postavlja poravnanje deteta *child* u delu okna u kome se nalazi
  - Pos (enum) konstante
    - Pos.CENTER – u centru dela okna i vertikalno i horizontalno
    - Ostale konstante su oblika Pos.XXX_YYY
      XXX = {TOP, BOTTOM, CENTER} → vertikalno poravnanje
      YYY = {LEFT, RIGHT, CENTER} → horizontalno poravnanje
    - Na primer: Pos.TOP_RIGHT, Pos.BOTTOM_LEFT, itd.

Primer - Dodavanja dugmića u BorderPane



```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class BorderPaneDemo extends Application {
  @Override
  public void start(Stage stage) throws Exception {
    BorderPane pane = new BorderPane();
    // levi i desni region
    Button left = new Button("Left");
    pane.setLeft(left);
    Button right = new Button("Right");
    pane.setRight(right);
    // gornje dugme ce imati neogranicenu sirinu
    Button top = new Button("Top");
    top.setMaxWidth(Double.MAX_VALUE);
    pane.setTop(top);
    // maks. sirina donjeg dugmeta je 160 tacaka
    Button bottom = new Button("Bottom");
    bottom.setMaxWidth(160);
    pane.setBottom(bottom);
    // centralni region
    Button center = new Button("Center");
    pane.setCenter(center);
    // poravnanja levog, desnog i dognjeg dugmeta
    BorderPane.setAlignment(left, Pos.TOP_LEFT);
    BorderPane.setAlignment(right, Pos.BOTTOM_RIGHT);
    BorderPane.setAlignment(bottom, Pos.CENTER);
    // postavi scenu i prikazi pozornicu
    Scene scene = new Scene(pane, 320, 240);
    stage.setScene(scene);
    stage.setTitle("BorderPane demo");
    stage.show();
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

HBox, VBox i FlowPane

- **HBox** i **VBox** raspoređuju čvorove, redom, po horizontalnoj, odnosno vertikalnoj liniji.
- Veličine svojih čvorova će inicijalno postaviti na željene vrednosti, a smanjivaće ih ukoliko nema dovoljno mesta, poštujući pri tome minimalne veličine.
- **FlowPane** može da raspoređuje čvorove i horizontalno i vertikalno, što se zadaje pomoću konstante tipa **Orientation**. Ovo okno će takođe postaviti veličine svojih čvorova na željene vrednosti.
- Ukoliko, pak, nema dovoljno mesta, FlowPane će prelamanjem redova i kolona pokušati da prikaže sve čvorove, ali im veličine neće menjati.

FlowPane

- Paket: javafx.scene.layout (kao i sve Pane klase)
- Konstruktori
  - FlowPane()
  - FlowPane(double hgap, double vgap)
  - FlowPane(Orientation orient)
  - FlowPane(Orientation orient, double hgap, double vgap)

  - Orientation (enum)
    - Orientation.HORIZONTAL → ređa komponente horizontalno sa leva na desno, pa prelazi u “sledeći red”
    - Orientation.VERTICAL → ređa komponente vertikalno odozgo na dole, pa prelazi u “sledeću kolonu”

FlowPane

- Metoda koja vraća listu čvorova koji se nalaze u oknu

  ObservableList<Node> getChildren()

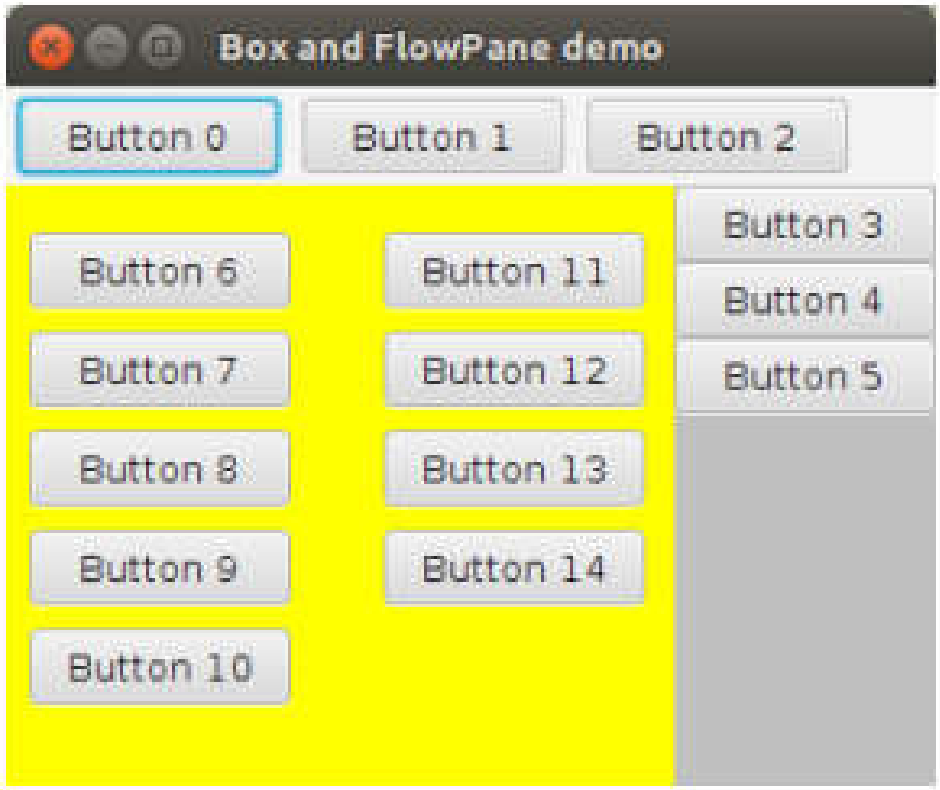
- Pozivajući metod **add(Node n)** nad tom listom dodajemo čvor u okno

- ObservableList – liste uz koje možemo vezati specijalne objekte (funkcijske objekte) koji će biti izvršeni svaki put kada se lista promeni

HBox, VBox i FlowPane

- U osnovnim podešavanjima, čvorovi su međusobno "slepljeni".
- Razmak se u slučaju HBox-a i VBox-a postavlja pozivom metoda **setSpacing**. U slučaju FlowPane-a, moguće je zasebno zadati horizontalni i vertikalni razmak, pozivom metoda **setHgap** i **setVgap**, redom.
- Dodatno, da bi se čvorovi "odlepili" od ivica okna, potrebno je pozvati metod **setPadding**, koji prihvata objekat klase Insets.
- Upotreba ova tri okna je prikazana u narednom primeru. Aplikacija sadrži 15 dugmića, od kojih su prva tri u HBox-u uz gornju ivicu prozora, druga tri u VBox-u uz desnu ivicu, a poslednjih 9 u FlowPane-u u centralnom delu prozora.
- Pošto su i sama okna čvorovi, i njih možemo stavljati u druge okna (u datom primeru, u BorderPane). Ovakvo kombinovanje okna se najčešće i koristi u praksi kako bi se dizajnirali kompleksniji korisnički interfejsi.

Primer - HBox, VBox i FlowPane



Primer - HBox, VBox i FlowPane

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class BoxFlowDemo extends Application {
  @Override
  public void start(Stage stage) throws Exception {
    // niz od 15 dugmica, svaki zeljene duzine od 90 tacaka
    Button[] buttons = new Button[15];
    for (int i = 0; i < buttons.length; i++) {
      buttons[i] = new Button("Button " + i);
      buttons[i].setPrefWidth(90);
    }

    HBox hbox = new HBox();
    // rastojanje cvorova od ivica okna je 4 piksela
    hbox.setPadding(new Insets(4));
    // medjusobno rastojanje komponenti
    hbox.setSpacing(8);
    for (int i = 0; i < 3; i++)
      hbox.getChildren().add(buttons[i]);

    // u ovom oknu ce komponente biti slepljene medjusobno,
    // ali i uz ivicu okna
    VBox vbox = new VBox();
    for (int i = 3; i < 6; i++)
      vbox.getChildren().add(buttons[i]);
```

Primer - HBox, VBox i FlowPane

```
    // vertikalni FlowPane
    FlowPane flow = new FlowPane(Orientation.VERTICAL);
    // rastojanja cvorova od, redom, gornje, leve, donje i desne ivice
    flow.setPadding(new Insets(16, 8, 16, 8));
    // horizontalno i vertikalno rastojanje komponenti
    flow.setHgap(32);
    flow.setVgap(8);
    for (int i = 6; i < buttons.length; i++)
      flow.getChildren().add(buttons[i]);

    // gornja tri okna stavljamo u BorderPane
    BorderPane bp = new BorderPane();
    bp.setTop(hbox);
    bp.setRight(vbox);
    bp.setCenter(flow);

    Scene scene = new Scene(bp, 320, 240);
    stage.setScene(scene);
    stage.setTitle("Box and FlowPane demo");
    stage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

GridPane

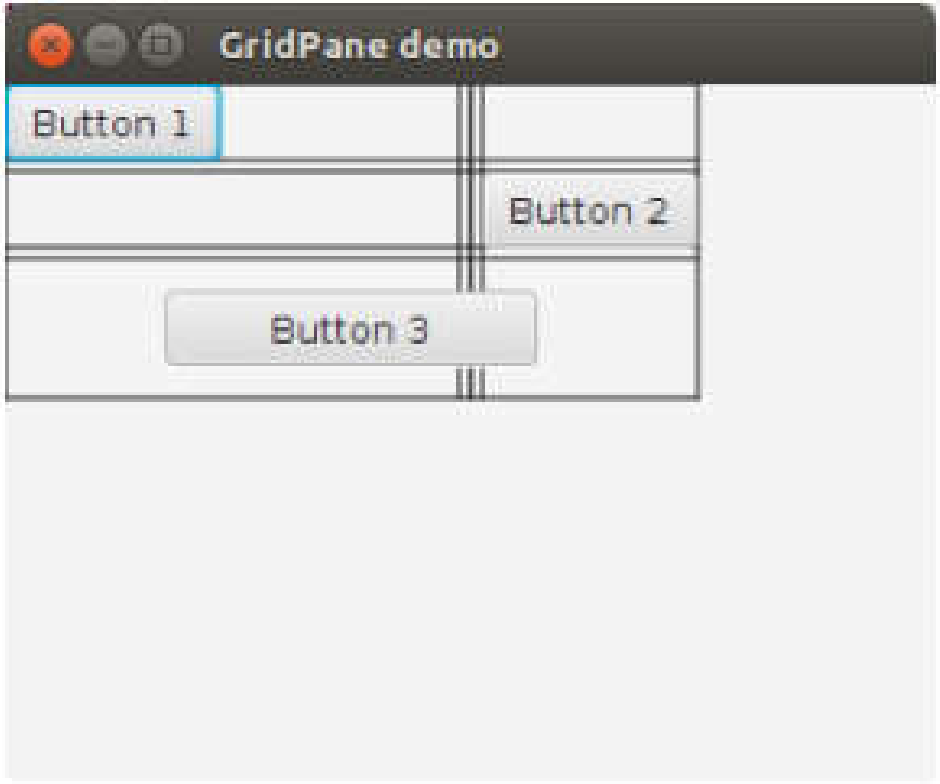
- **GridPane** predstavlja tabelu. Čvor se može postaviti u proizvoljnu ćeliju tabele, i može se protezati proizvoljan broj kolona i redova.
- Pri tome, tabela raste automatski kako dodajemo čvorove u nju. Na primer, ako već prvi čvor postavimo u treću kolonu i drugi red, tabela će automatski postati dimenzija 3x2, pri čemu će ćelije u prve dve kolone kao i prvom redu biti prazne.
- Slično, ako čvor postavimo u prvu kolonu i prvi red, a kažemo da se proteže dve kolone, tabela će biti dimenzija 2x1. Jedna ćelija može sadržati i više čvorova, ali tada može doći do njihovog preklapanja.

> Indeksi kolona i redova počinju od 0, te se pod "drugom kolonom" podrazumeva kolona sa indeksom 1.

GridPane

- GridPane postavlja veličine čvorova na zadate željene vrednosti.
- Ukoliko su minimalne dimenzije prevelike, čvor može izaći van okvira ćelije, odnosno tabele.
- U podrazumevanim podešavanjima, ćelija tabele ima visinu najvišeg čvora u redu, i širinu najšireg čvora u koloni. Ova podešavanja se, naravno, mogu menjati.
- Kao što je rečeno, dimenzije ćelija tabele zavise od dimenzija čvorova.
- Moguće je i ručno zadati širine kolona i visine redova, popunjavanjem, redom, kolekcija ***columnConstraints*** i ***rowConstraints*** GridPane-a.

Primer - upotreba GridPane-a



Primer - upotreba GridPane-a

```
import java.util.ArrayList;
import java.util.List;
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.RowConstraints;
import javafx.stage.Stage;

public class GridPaneDemo extends Application {
  @Override
  public void start(Stage stage) throws Exception {
    GridPane grid = new GridPane();
    grid.setHgap(4);
    grid.setVgap(4);
    // prikazi linije tabele (korisno prilikom debug-ovanja)
    grid.setGridLinesVisible(true);

    // dodaj dugme u celiju (0, 0)
    Button b1 = new Button("Button 1");
    grid.add(b1, 0, 0);
    // dodaj dugme u celiju (2, 1)
    Button b2 = new Button("Button 2");
    grid.add(b2, 2, 1);
    // dodaj dugme sirine 128 tacaka u celiju (0, 2),
    // tako da se proteze 3 kolone i 1 red i bude centralno poravnato
    Button b3 = new Button("Button 3");
```

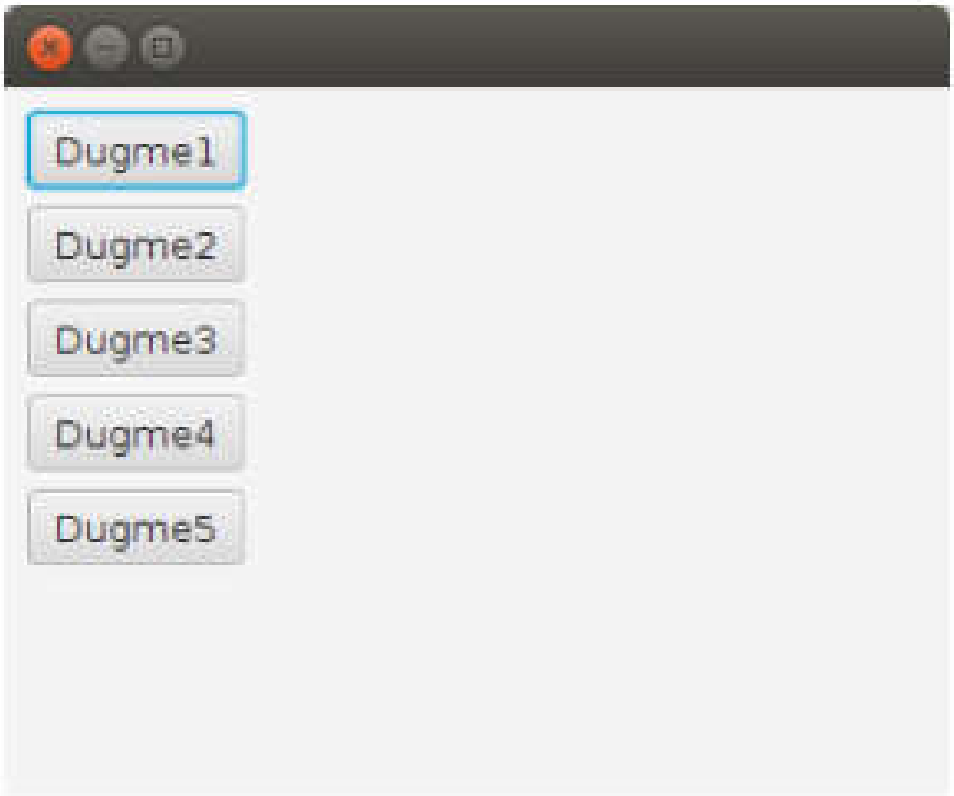
Primer - upotreba GridPane-a

```
        b3.setPrefWidth(128);
        grid.add(b3, 0, 2, 3, 1);
        GridPane.setHalignment(b3, HPos.CENTER);

        // postavljamo zeljene visine redova
        List<RowConstraints> rowc = new ArrayList<>();
        rowc.add(new RowConstraints()); // podrazumevana vrednost za 0. red
        rowc.add(new RowConstraints()); // podrazumevana vrednost za 1. red
        rowc.add(new RowConstraints(48)); // tacno 48 tacaka za 2. red
        // preostali redovi (ako postoje) ce imati podrazumevane visine
        grid.getRowConstraints().addAll(rowc);

        // postavljamo zeljene sirine kolona
        // 50% dostupne sirine za 1. kolonu
        ColumnConstraints cc = new ColumnConstraints();
        cc.setPercentWidth(50);
        // preostale kolone (ako postoje) ce imati podrazumevane sirine
        grid.getColumnConstraints().add(cc);

        stage.setScene(new Scene(grid, 320, 240));
        stage.setTitle("GridPane demo");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Pane

- Postoje situacije u kojima ne želimo automatsko raspoređivanje čvorova, već želimo sami da zadamo (fiksnu) poziciju i veličinu svakog čvora. Ovo se postiže korišćenjem okna **Pane**, koji ujedno predstavlja i osnovnu klasu za sva ostala okna.
- Prilikom dodavanja čvora, Pane će veličinu čvora postaviti na željenu vrednost i više je neće menjati. Pozicija svakog čvora će biti (0, 0), tj. nalaziće se u gornjem-levom uglu okna.
- Novu poziciju možemo postaviti ili pozivom metoda **relocate** koji prihvata x i y, ili pozivima metoda **setLayoutX** i **setLayoutY** za pojedinačno zadavanje x, odnosno y koordinate. Pri tome, x-osa „ide“ desno, dok y-osa „ide“ dole.
- Upotrebom okna Pane imamo veću fleksibilnost u raspoređivanju komponenti, ali je ujedno odgovornost za konačan izgled aplikacije u potpunosti na nama.

Primer - Upotreba okna Pane



Primer - Upotreba okna Pane

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

public class PaneDemo extends Application {
  @Override
  public void start(Stage stage) throws Exception {
    Pane pane = new Pane();

    double y = 8; // pocetna y-pozicija
    for (int i = 1; i <= 5; i++) {
      Button d = new Button("Dugme" + i);
      pane.getChildren().add(d);
      d.relocate(8, y);
      y += 32;
    }

    Scene scene = new Scene(pane, 320, 240);
    stage.setScene(scene);
    stage.show();
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

Programiranje vođeno događajima

- Za razliku od proceduralnih aplikacija u kojima se naredbe izvršavaju sekvencijalno ili paralelno, grafičke aplikacije se izvršavaju na osnovu događaja.
- To znači da nakon startovanja aplikacija „čeka“ na neki događaj, kao što je klik mišem ili pritisak tastera, a zatim izvršava skup naredbi kao odgovor na taj događaj.
- Ovaj model programiranja grafičkih aplikacija se naziva programiranje vođeno događajima (*eng. event-driven programming*) i u njemu je logika programskog koda „izvrnuta“.

Programiranje vođeno događajima

- Umesto jedinstvenog kontrolnog toka koji bi tekao od početka do kraja rada aplikacije, sistemski dodeljena nit se izvršava u petlji i čeka na akcije korisnika.
- Kada korisnik klikne mišem, operativni sistem o tome obaveštava pomenutu nit, koja ovu informaciju dalje prosleđuje prozoru, sceni, ili konkretnoj komponenti na koju je kliknuto.
- Tako se programiranje interaktivnih grafičkih aplikacija svodi na raspoređivanje čvorova u okviru prozora, a zatim i na definisanje metoda koji služe za obradu događaja.

Tipovi i obrada događaja

- JavaFX aplikacija može da obrađuje različite kategorije događaja, predstavljene odgovarajućim klasama.
- Klase su organizovane u obliku hijerarhije u čijem korenu se nalazi **javafx.event.Event**.
- Neke od najznačajnijih klasa su:
  - **KeyEvent**: koristi se kada korisnik pritisne ili otpusti taster na tastaturi;
  - **MouseEvent**: kada korisnik pritisne dugme ili pomeri miša;
  - **MouseDragEvent**: kada korisnik izvrši prevlačenje mišem;
  - **ActionEvent**: specijalizovan događaj, označava da je korisnik pritisnuo dugme, otvorio ili zatvorio padajuću listu (eng. combo box), ili odabrao stavku iz menija;
  - **WindowEvent**: prozor aplikacije je prikazan, sakriven, ili zatvoren.

Tipovi i obrada događaja

- Pored ovih, JavaFX ima ugrađenu podršku i za moderne uređaje sa ekranima osetljivim na dodir, kroz klase TouchEvent, SwipeEvent i ZoomEvent.
- Unutar svake kategorije, odnosno klase, postoji više tipova događaja.
- Na primer, korisnik može da pomeri miša ili da pritisne neko njegovo dugme - ovo su različiti tipovi događaja MouseEvent.
- Da bismo dobili informaciju o događaju, neophodno je da za odgovarajući čvor „zakačimo“ metod za obradu događaja. Na primer, ako nas interesuje kada je korisnik kliknuo na neko dugme, tada ćemo metod za obradu događaja „zakačiti“ za to konkretno dugme.

Tipovi i obrada događaja

- Kada korisnik klikne na dugme, JavaFX će prikupiti sve značajne informacije vezane za ovaj događaj i proslediće ih našem metodu.
- Prikupljene informacije će, u slučaju klika, obuhvatati poziciju kursora, ali i kojim dugmetom i koliko puta je kliknuto.
- Sličan pristup se koristi za sve klase događaja.

Rad sa tastaturom

- Za rad sa tastaturom JavaFX nudi tri osnovna tipa događaja:
  - KeyEvent.KEY_PRESSED: korisnik je pritisnuo (i drži) neki taster;
  - KeyEvent.KEY_RELEASED: korisnik je otpustio taster; i
  - KeyEvent.KEY_TYPED: korisnik je uneo neki karakter.
- Prva dva događaja su nižeg nivoa i obaveštavaju nas da je korisnik pritisnuo ili otpustio neki od dostupnih tastera.
- To uključuje i kontrolne tastere, kao što su shift, escape, alt, strelice, itd.
- Poslednji tip događaja se generiše samo ako je korisnik uneo neki karakter bilo pritiskom alfanumeričkog tastera, bilo držanjem tastera alt i navođenjem numeričkog koda.

Rad sa tastaturom

- Informacije o događaju se prikupljaju u objekat klase KeyEvent.
- Najznačajniji metod ovog objekta je **getCode()** koji vraća oznaku pritisnutog/otpuštenog tastera. Povratna vrednost je nabrojivi tip **KeyCode** sa konstantama koje predstavljaju tastere.
- Za događaj tipa **KEY_TYPED** se može koristiti i metod **getCharacter()** koji će vratiti uneti karakter u obliku stringa.
- U narednom primeru je dat deo aplikacije koja „osluškuje“ tastaturu, i informacije o pritisnutim i otpuštenim tasterima ispisuje u konzoli.
- Dodatno, ako korisnik pritisne escape, aplikacija završava sa radom. U ovom primeru, metod za obradu događaja je vezan za scenu, ali to može biti bilo koji čvor, odnosno komponenta.

Rad sa tastaturom

```
@Override
public void start(Stage stage) throws Exception {
  Scene scene = new Scene(new Pane(), 320, 240);
  // korisnik je pritisnuo (i drzi) neki taster
  scene.setOnKeyPressed(e -> {
    if (e.getCode() == KeyCode.ESCAPE)
      stage.close();
    else
      out.println(e.getCode() + " Pressed");
  });
  // korisnik je uneo neki karakter
  scene.setOnKeyTyped(e -> out.println(e.getCharacter() + " Typed"));
  // korisnik je otpustio neki taster
  scene.setOnKeyReleased(e -> out.println(e.getCode() + " Released"));

  stage.setScene(scene);
  stage.setTitle("Keyboard Demo");
  stage.show();
}
```

Ako pritisnemo kombinaciju 'shift+a', u konzoli će biti ispisano sledeće:

```
SHIFT Pressed
A Pressed
A Typed
A Released
SHIFT Released
```

Ako pritisnemo samo taster 'a', u konzoli ćemo videti sledeći ispis:

```
A Pressed
a Typed
A Released
```

Događaji **KEY_PRESSED** i **KEY_RELEASED** sadrže jedinstveni kod tastera na tastaturi (u konkretnom primeru, kod tastera A), te se ne mogu direktno koristiti za razlikovanje velikih i malih slova.

Rad sa mišem

- Pošto je pomoću miša moguće izvesti mnogo više akcija nego pomoću tastature, JavaFX nudi veći skup klasa za rad sa mišem nego sa tastaturom.
- Najpre, postoje četiri klase događaja:
  - **MouseEvent,**
  - **MouseDragEvent,**
  - **DragEvent i**
  - **ScrollEvent.**

Rad sa mišem

- Prva klasa **MouseEvent** pokriva standardne operacije mišem, poput kliktanja, pomeranja kursora i prevlačenja mišem unutar jednog čvora.
- Klasa **MouseDragEvent** se koristi kada prevlačenje mišem uključuje više čvorova; na primer, korisnik prevlači neki sadržaj iz jedne u drugu komponentu unutar prozora aplikacije.
- Događaj predstavljen klasom **DragEvent** se generiše kada korisnik iz druge (ne obavezno Java) aplikacije prevlači sadržaj u prozor JavaFX aplikacije.
- Konačno, **ScrollEvent** se generiše kada korisnik okreće točkić miša.

Rad sa mišem

- Klasa MouseEvent sadrži nekoliko različitih tipova događaja, uključujući i sledeće:
  - MOUSE_PRESSED: korisnik je pritisnuo (i drži) taster miša;
  - MOUSE_RELEASED: korisnik je otpustio taster;
  - MOUSE_CLICKED: korisnik je kliknuo tasterom miša;
  - MOUSE_ENTERED: kursor miša je ušao u čvor;
  - MOUSE_EXITED: kursor je izašao iz čvora;
  - MOUSE_DRAGGED: korisnik vrši prevlačenje mišem.
- Metod za obradu događaja miša prihvata brojne informacije, uključujući:
- poziciju kursora, kojim tasterom i koliko puta je kliknuto, da li je korisnik istovremeno pritisnuo i neki od kontrolnih tastera (shift, alt, ctrl), itd.

Rad sa mišem

- Tasteri miša su označeni nabrojivim tipom MouseButton sa konstantama PRIMARY, SECONDARY, MIDDLE i NONE.
- Primarni i sekundarni taster zavise od korisničkih podešavanja unutar operativnog sistema.
- Po podrazumevanim podešavanjima, primarni taster je levi, dok je sekundardni desni.
- Međutim, ukoliko korisnik, na primer, koristi podešavanja za levoruke osobe, primarni taster će biti desni, a sekundarni levi.
- Srednji taster je obično točkić, dok je NONE zgodna konstanta koja se može koristiti u metodu za obradu događaja kako bi se utvrdilo da nijedan taster nije pritisnut.

Rad sa mišem

- Postoje tri para brojeva za utvrđivanje pozicije miša. X i Y predstavljaju poziciju unutar čvora, u odnosu na njegovu gornju-levu tačku.
- SceneX i SceneY su koordinate kursora relativne u odnosu na scenu aplikacije (tj. njenu gornju levu tačku), dok su ScreenX i ScreenY definisane u odnosu na ekran u kom se aplikacija nalazi.
- U nastavku je data jednostavna aplikacija koja demonstrira crtanje linija u prozoru.

Rad sa mišem

- Ako korisnik drži pritisnut levi taster i prevlači miša, ispod kursora ostaje „trag“ u vidu crne linije. Dupli klik levim tasterom briše sadržaj prozora.
- Konačno, ako korisnik drži desni taster i prevlači miša, brišu se linije ispod kursora (tj. simulira se gumica).
- Za crtanje u prozoru koristimo objekat klase **Canvas**. Canvas je slika, odnosno vizuelna komponenta po kojoj je moguće crtati.
- Samo crtanje se vrši pomoću objekta klase **GraphicsContext**, koja nudi veliki skup metoda za rad sa geometrijskim figurama u ravni.

Rad sa mišem

```
public void start(Stage stage) throws Exception {
    Canvas canvas = new Canvas(320, 240);
    final GraphicsContext gc = canvas.getGraphicsContext2D();
    gc.setStroke(Color.BLACK);
    // postavljamo inicijalnu poziciju olovke kada korisnik
    // pritisne levi taster
    canvas.setOnMousePressed(e -> {
        if (e.getButton() == MouseButton.PRIMARY)
            gc.moveTo(e.getX(), e.getY());

    });
    // u zavisnosti od toga koji taster je pritisnut,
    // prevlacenje moze crtati liniju ili simulirati gumicu
    canvas.setOnMouseDragged(e -> {
        if (e.getButton() == MouseButton.PRIMARY) {
            gc.lineTo(e.getX(), e.getY());
            gc.stroke();
        }
        else if (e.getButton() == MouseButton.SECONDARY)
            gc.clearRect(e.getX() - 1, e.getY() - 1, 3, 3);
    });
    // dupli klik levim tasterom brise ceo Canvas
    canvas.setOnMouseClicked(e -> {
        if (e.getButton() == MouseButton.PRIMARY && e.getClickCount() == 2)
            gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
    });
    BorderPane pane = new BorderPane();
    pane.setCenter(canvas);
    Scene scene = new Scene(pane);
    stage.setScene(scene);
    stage.setTitle("Paint Demo");
    stage.show();
}
```

Rad sa događajima prozora i akcijama

- JavaFX aplikacija može reagovati i na događaje koje generiše sam prozor aplikacije. Klasa WindowEvent definiše nekoliko tipova događaja:
  - WINDOW_CLOSE_REQUEST: korisnik želi da zatvori prozor;
  - WINDOW_HIDING: generiše se pre nego što počne minimiziranje prozora;
  - WINDOW_HIDDEN: prozor je minimiziran;
  - WINDOW_SHOWING: generiše se pre nego što je prozor prikazan na ekranu;
  - WINDOW_SHOWN: prozor je prikazan na ekranu.

Rad sa događajima prozora i akcijama

- Primer dat u nastavku prikazuje prozor koji je nemoguće zatvoriti. Prozor sadrži jedno dugme koje bi zatvorilo prozor kada bi korisnik kliknuo na njega.
- Međutim, dugme se pomera gore/dole kada se nađe ispod kursora miša.
- Izgled prozora je postavljen na **StageStyle.UNDECORATED**, što znači da prozor nema ni uobičajene dugmiće za zatvaranje, minimiziranje i maksimiziranje.
- Aplikacija odbija da zatvori prozor tako što postavlja metod za obradu događaja **WINDOW_CLOSE_REQUEST** i unutar njega poziva consume(), uništavajući time događaj.
- Konačno, ako korisnik ipak nekako uspe da klikne na dugme primarnim tasterom, „bićemo pošteni“ i zatvorićemo prozor.

Primer - annoying windows

```
public void start(Stage stage) throws Exception {
  Button btnClose = new Button("Click me to Close");
  final double Y1 = 40, Y2 = 160;
  btnClose.setPrefSize(260, 40);
  btnClose.relocate(30, Y1);
  // ako korisnik ipak nekako klikne na dugme, zatvoricemo prozor
  btnClose.setOnMouseClicked(e -> {
    if (e.getButton() == MouseButton.PRIMARY && e.getClickCount() == 1)
      stage.close();
  });
  // sakrivamo dekoracije prozora, ukljucujuci i dugmice minimize,
  // maksimize i close
  stage.initStyle(StageStyle.UNDECORATED);
  // ako korisnik pokusa da zatvori prozor (npr. Alt+F4), odbicemo
     zahtev
  stage.setOnCloseRequest(e -> e.consume());
  // ako kursor misa udje u dugme sa gornje strane, dugme pomeramo
  // gore, i obrnuto
  btnClose.setOnMouseMoved(e -> {
    boolean gore = Math.abs(btnClose.getLayoutY() - Y1) < 0.001;
    double newY = gore ? Y2 : Y1;
    btnClose.setLayoutY(newY);
  });
  Pane root = new Pane(btnClose);
  stage.setScene(new Scene(root, 320, 240));
  stage.setTitle("Annoying Window");
  stage.setResizable(false);
  stage.show();
}
```

Primer – annoying windows

- Umesto klika mišem, kao što je dato u ovom primeru, prilikom rada sa dugmićima se češće koriste tzv. akcije.
- Akcija nad dugmetom označava da je korisnik pritisnuo to dugme: bilo kliktanjem miša, bilo pritiskom tastera na tastaturi (npr. space ili enter).
- Akcije su predstavljene klasom **ActionEvent** sa samo jednim tipom ACTION, koji označava da je akcija izvršena.
- Akcije se koriste i sa drugim komponentama, kao što su npr. padajuće liste.

Faze u obradi događaja

- Događaj se zatim šalje kroz lanac, u fazi koja se naziva ***hvatanje događaja*** (eng. event capturing phase). Tokom ove faze, bilo koji čvor u lancu, uključujući i sam ciljni objekat, može vršiti filtriranje događaja. Filter je funkcija koja procesira, i može da odbaci događaj.
- Recimo, ako se pozornici zada filter koji odbacuje sva kliktanja desnim tasterom miša, tada nijedan čvor neće biti obavešten o ovoj akciji korisnika.
- Slično, ukoliko npr. razvijamo kalkulator, želimo da ignorišemo sve tastere sem cifara i aritmetičkih operacija. Najlakši način za postizanje ovoga bi bilo zadavanje odgovarajućeg filtera pozornici.
- Faza hvatanja događaja se završava kada neki filter u lancu odbaci događaj, ili kada događaj prođe kroz ceo lanac.

Faze u obradi događaja

- Nakon toga sledi faza ***obrade događaja*** (eng. *event bubbling*). U ovoj fazi, događaj se prosleđuje metodama za obradu, najpre onom koji je zakačen za ciljni objekat, zatim metodu koji je zakačen za roditelja ciljnog objekta, i tako dalje, unazad sve do pozornice.
- Slično kao filtriranje prilikom faze hvatanja događaja, bilo koji metod za obradu događaja može prekinuti dalje napredovanje događaja uz hijerarhiju, pozivom metoda *consume*(), kao što je prikazano u prethodnom primeru.
- Posmatrajmo faze u obradi događaja za:
- **Stablo aplikacije**: u korenu je pozornica, ona sadrži scenu, scena sadrži jedno okno tipa BorderPane, koje, konačno, sadrži dva dugmeta, nazvana dugme1 i dugme2.
- Svi čvovorovi u stablu imaju definisane filtere i metode za obradu kliktanja mišem, koji ispisuju informacije na konzoli.

Faze u obradi događaja

- Kada korisnik klikne mišem na dugme1, najpre se konstruiše sledeći lanac prenosa događaja:

$$stage \rightarrow scene \rightarrow pane \rightarrow dugme1$$

- Tokom faze hvatanja događaja, u konzoli će biti ispisano sledeće (ECP je skraćenica od ***Event Capturing Phase***):

```
ECP stage
ECP scene
ECP pane
ECP dugme1
```

- Dakle, događaj polazi od pozornice i stiže do ciljnog dugmeta, jer ga nijedan filter nije odbacio.
- Nakon toga sledi faza obrade i događaj se prvo prosleđuje metodu za obradu zakačenom za ciljno dugme.
- Obrada klika mišem na dugme automatski odbacuje događaj (tj. automatski se poziva metod consume), te se faza obrade tu završava.

Faze u obradi događaja

- Sa druge strane, ako bismo kliknuli na okno, u konzoli bi bilo ispisano sledeće (EBP je skraćenica od ***Event Bubbling Phase***):

```
ECP stage
ECP scene
ECP pane
EBP pane
EBP scene
```

- Dakle, događaj putuje od pozornice do okna i prolazi sve filtere, a zatim se vraća nazad do pozornice. Međutim, metod za obradu događaja zakačen za scenu ga odbacuje, te je ovo i poslednji ispis koji se vidi u konzoli - događaj neće stići nazad do pozornice.