UNIVERZITET METROPOLITAN BEOGRAD

CS202 - OBJEKTNO-ORIJENTISANO PROGRAMIRANJE 2

Programiranje sa događajima

Lekcija 02

PRIRUČNIK ZA STUDENTE

CS202 - OBJEKTNO-ORIJENTISANO PROGRAMIRANJE 2

Lekcija 02

PROGRAMIRANJE SA DOGAĐAJIMA

UVOD

Ciljevi ove lekcije

Cilj ove lekcije je da:

- student razume specifičnost programiranje vođeno događajima
- opiše događaje, njihove izvore i klase koje definišu događaje
- definiše klase za obradu događaja upotrebom unutrašnjih klasa,
- definiše klase za obradu događaja upotrebom anonimnih unutrašnjih klasa
- prikaže uprošćenje obrade događaja primenom lambda izraza
- razvije GUI aplikaciju za obračun duga
- pokaže pisanje programa koji rade sa događajima koje generiše miš (MouseEvents)
- pokaže pisanje programa koji rade sa dođađajima koje generiše tastaura (KeyEvents)
- pokaže kako se kreiraju osluškivač događaja koji nastaju promenama vrednosti u posmatranom objektu,
- kako se koriste klase Animation, PathTransition, FadeTransition, i Timeline pri razvoju animacija
- Nauči studenta da rsyvije animaciju ya simulaciju lopte koja skakuće

Kako se rad sa dođajima u programiranju najčešće koristi pri radi pri interakciji korisnika i aplikacije, preko grafičkog korisničkog interfejsa (GUI), to ćemo na početku lekcije da ti i jedan mali video klip o kreiranju GUI primenom JavaFX Java paketa. To je noviji pkaet razvijen s namerom da zameni Java swing paket..

Referenca: Y. Daniel Liang, INTRODUCTION TO JAVA PROGRAMMING (COMPREHENSIVE VERSION), Tenth Edition, Pearson, ISBN 10: 0-13-376131-2, ISBN 13: 978-0-13-376131-3

Ovo je osnovni udžbenik za ovaj predmet i preporučuje se studentima da ga koriste,

Poglavlje 1

Šta je programiranje sa događajima?

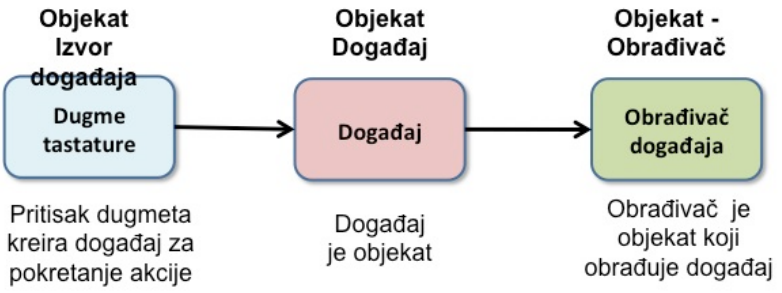
USLOVI ZA OBRADU DOGAĐAJA

Potrebno je da napišete program koji obrađuje događaje, kao što je na primer, klik dugmeta menija, pokret mišem ili pritisak tastera na tastaturi.

Najveći broj aplikacija zahtevaju komunikaciju sa korisnikom aplikacije, jer on mora da izabere šta želi od aplikacije, i dr. Ove aplikacije za tu namenu obezbeđuju grafički korisnički interfejs (Graphic User Interface – GUI). Tipičan scenario rada se sastoji u sledećem:

- Korisnik klikne određeno dugme na grafičkom interfejsu
- Kao rezultat, program generiše odgovarajući signal – tj. određeni tip događaja (event).
- Softverski objekat – obrađivač događaja (event handler) – obrađuje događaj time što inicira odgovarajuću akciju programa.

Ovo je tipičan scenario rada i programiranja sa događajima (slika 1).



Slika 1.1 .1: Obrađivač je objekat koji obrađuje događaj kreiran u svom izvornom objektu

Da bi neki objekat mogao da ima ulogu obrađivača događaja, mora da zadovolji dva uslova:

1. Objekat mora da bude primerak klase koja primenjuje intrefejs***EventHandler<T extends Event>***. Ovaj interfejs definiše zajedničko ponašanje svih obrađivača događaja, ***<T extends Event>,*** gde je generički tip koji je podtip od ***Event*** .
2. Objekat ***EventHandler*** vrši obradu događaja za koji je prethodno registrovan kod objekta koji je izvora događaja, korišćenjem njegovog metoda source.***setOnAction(handler***).

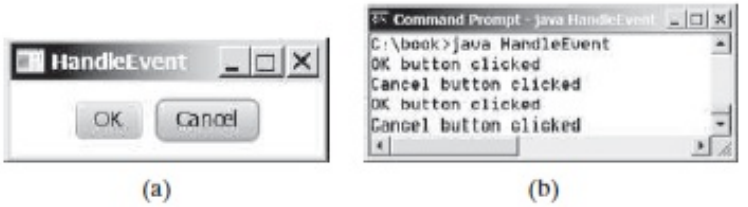
Interfejs ***EventHandler<ActionEvent>*** sadrži metod ***handle(ActionEvent)*** za obradu događaja акcije. Vaša klasa za obradu događaja mora da predefiniše ovaj metod koji treba da odgovori na događaj.

PRIMER KLASE HANDLEEVENT

Klikom dugmeta OK ili Cancel dobijaju se dve odgovarajuće poruke na monitoru računara.

Ovde se daje primer koda koji obrađuje događaj akcije – ***ActionEvent*** - koji nastaje klikom na dva dugmeta menija (slika 2). Kada se pritisne dugme OK, prikazuje se na monitoru poruka:

“OK button clicked” . Kada se pritisne dugme Canceč, prikazuje se poruka: “Cancel button clicked”.



Slika 1.2 .2: (a) Program prikazuje dva dugmeta (b) Prikaz poruke na monitoru po pritisku ovih dugmeta [1]

U linijama 32-44 prikazane su dve klase obrađvača . *EventHandler<ActionEvent>* koji obrađuju ActionEvent . Obrađivač handler 1 obrađuje generisan događaj klika dugmeta OK (linija 18) , a handle 2 – dugmeta Cancel (linija 20).

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
 import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
 import javafx.event.EventHandler;

public class HandleEvent extends Application {
  @Override // Predefinisanje start metod klase Application
  public void start(Stage primaryStage) {
  // Kreiranje okvira za unos njegivih svojstava
 HBox pane = new HBox(10);
  pane.setAlignment(Pos.CENTER);
 Button btOK = new Button("OK");
 Button btCancel = new Button("Cancel");
 OKHandlerClass handler1 = new OKHandlerClass();
 btOK.setOnAction(handler1);
  CancelHandlerClass handler2 = new CancelHandlerClass();
  btCancel.setOnAction(handler2);
```

```
pane.getChildren().addAll(btOK, btCancel);

// Kreiranje scene i postavljanje scene na pozornicu
  Scene scene = new Scene(pane);
 primaryStage.setTitle("HandleEvent"); // Set the stage title
primaryStage.setScene(scene); // Place the scene in the stage
primaryStage.show(); // Display the stage
   }
 }

class OKHandlerClass implements EventHandler<ActionEvent> {
   @Override
  public void handle(ActionEvent e) {
 System.out.println("OK button clicked");
  }
}

class CancelHandlerClass implements EventHandler<ActionEvent> {
  @Override
  public void handle(ActionEvent e) {
 System.out.println("Cancel button clicked");
   }
}
```

VIDEO - UVODNE NAPOMENE O JAVAFX

Upoznavanje sa JavaFX paketom za kreiranje GUI-a, jer će se rad sa događajima najviše objašnjavati primenim razvoja GUI-a uz pomoć JavaFX (u 2. i 3. lekcije koje traju 3 nedelje)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Događaji i izvori događaja

DOGAĐAJI

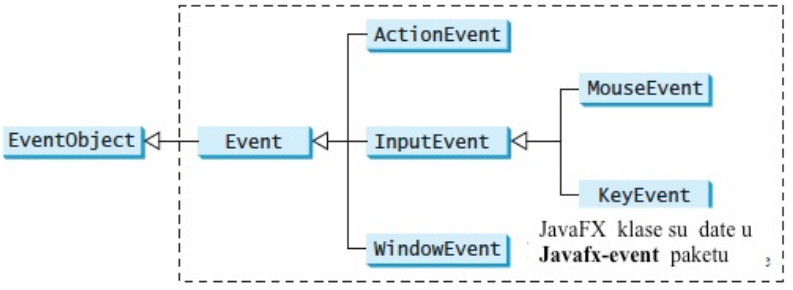
Događaj je objekat koji je kreirao izvorni objekat. "Ispaljivanje" događaja znači da je kreiran neki događaj i da je delegiran obrađivač da obradi taj događaj.

Pri radu interaktivnih aplikacija, program je u interakciji sa korisnikom i generisani događaji usmeravaju izvršenje programa. To se zove programiranje sa događajima. (event-driven programming).

Događaj (event) se može definisati kao signal programu da se nešto desilo. Oni se stvaraju ("ispaljuju") spoljnim korisnika , kao što je pokretanje i rad sa mišom, i kucanjem po tastaturi. Program može da reaguje na ove događaje, a može da se programira da na njih ne reaguje.

Objekat koji kreira i "ispaljuje" neki događaj naziva se objekat – izvor događaja (event source object) , ili kraće, izvorni objekat ili izvorna komponenta. Na primer, dugme menija je izvorna komponenta, jer klikom dugmeta (mouse) kreira se akcioni događaj buttonclicking.

Događaj je objekat klase Event. Osnovna klasa Java klasa događaja je ***java.util.EventObject*** . Osnovna klasa svih klasa događaja u JavaFX ***javafx.event.Event***. Hijerarhijska veza između klasa događaja je prikazana na slici 1.



Slika 2.1 .1: Događaj u JavaFX je objekat klase javafx.event [1]

Objekat događaja, sadrži sva svojstva koja su specifična za određeni događaj. Može se utvrditi izvorni objekat događaja upotrebom metoda ***getSource()*** klase ***EventObject.***

Podklase klase ***EventObject*** definišu specifične tipove događaja, kao što su događaji akcija (***ActionEvent***), događaju prozora, (***WindowEvent***) događaji miša (***MouseEvent***) i događaji tastature ***(KeyEvent***)

AKCIJE KOJE IZAZIVAJU DOGAĐAJE

Određene interakcije korisnika, preko GUI, izazivaju kreiranje i ispaljivanje određenih tipova događaja od strane izvornih objekata.

Prve tri kolone prikazane tabele pokazuju akcije korisnika koje dovode da izvorni objekt kreira ("ispali") navedeni tip događaja .

Na primer, kada se klikne dugme da GUI, ono kreira i ispaljuje tip događaj ***ActionEvent*** .

Akcija korisnika	Izvorni objekat	Tip događaja	Metodi interfejsa osluškivača
Klik dugmeta	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Pritisnut ENTER	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Čekiraj ili Nečekiraj	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Čekiraj ili Nečekiraj	CheckBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Izbor nove stavke	ComboBox	ActionEvent	setOnAction(EventHandler<ActionEvent>
Pritisak na mišu	Node, Scene	MouseEvent	setOnMousePressed(EventHandler<MouseEvent>)
Oslobodi miš			setOnMousePressed(EventHandler<MouseEvent>)
Klik mišom			setOnMousePressed(EventHandler<MouseEven
Unos mišom			setOnMousePressed(EventHandler<MouseEvent>)
Izlaz miša			setOnMousePressed(EventHandler<MouseEvent>)
Kretanje mišom			setOnMousePressed(EventHandler<MouseEvent>)
Vučenje mišom			setOnMousePressed(EventHandler<MouseEvent>)
Dugme pritisnuto	Node, Scene	KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Dugme oslobođeno			setOnKeyPressed(EventHandler<KeyEvent>)
Otkucaj na dugmentu			setOnKeyPressed(EventHandler<KeyEvent>)

Slika 2.2 .2: Akcije korisnika, izvorni objekat, tip događaja i metod registracije događaja [1]

VIDEO - OBRADA KORISNIČKIH DOGAĐAJA

JavaFX Java GUI Tutorial - 2 - Handle User Events (6,16 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 3

Registracija obrađivača i obrada događaja

OBRAĐIVAČ DOGAĐAJA

Obrađivač događaja je objekat koji mora da se registruje kod izvornog objekta događaja i mora da bude primerak odgovarajućeg interfejsa za obrađivanje događaja.

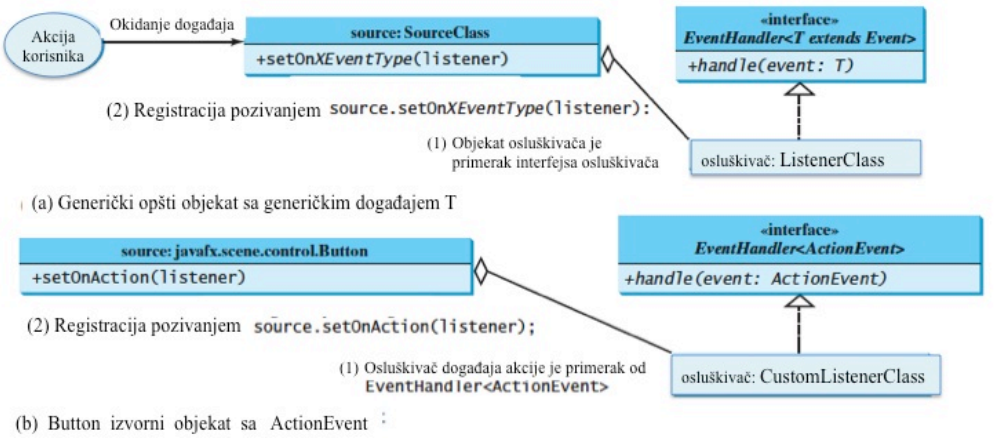
Java koristi model delegiranja za obrađivanje događaja: Izvorni objekat "ispaljuje" događaj, a objekat koji je za njega zainteresovan ga onda obrađuje. Objekat koji to radi je obrađivač događaja ili osluškivač događaja.

Da bi neki objekat bio obrađivač događaja,mora da zadovolji dva preduslova (slika 1):

1. Osluškivač mora da bude primerak odgovarajućeg interfejsa za obradu događaja, da bi time obezbedio odgovarajući metod za obradu događaja. JavaFX definiše jedinstveni interfejs za obradu događaja ***EventHandler<T extends Event>*** za neki događaj ***T***. Ovaj interfejs sadrži metod ***handle(T e)*** za obradu događaja. Ako je događaj ***e*** tipa ***ActionEvent*** onda umesto T treba koristiti ***Action Event***.

2. Obrađivač mora da se registruje kod izvornog objekta. Metod za registraciju zavisi od od tipa događaja. Za ***ActionEvent*** tip događaja metod je ***setOnAction().*** Za događaj koji nastaje pristiskom dugmeta miša, koristi se metod ***setOnMousePressed().*** Za dođađaj koji nastaje pritiskom dugmeta na tastaturi, metod za obradu je ***setOnKeyPressed().***

.



Slika 3.1 .1: Osluškivač je primerak interfejsa osluškivala i mora se registrovati kod izvornog objekta [1]

PRIMER OBRADE DOGAĐAJA

Kada kliknete na dugme, objekat Button ispaljuje ActionEvent događaj koji prebacuje obrađivaču na obradu metodom handle(ActionEvent)

U donjem listingu, dat je program (tj. klasa) HandleEvent.java

```
import javafx.application.Application;
 import javafx.geometry.Pos;
 import javafx.scene.Scene;
 import javafx.scene.control.Button;
 import javafx.scene.layout.HBox;
 import javafx.stage.Stage;
 import javafx.event.ActionEvent;
 import javafx.event.EventHandler;

 public class HandleEvent extends Application {
   @Override // Predefinisanje start metod klase Application
   public void start(Stage primaryStage) {
  // Kreiranje okvira za unos njegivih svojstava
  HBox pane = new HBox(10);
  pane.setAlignment(Pos.CENTER);
  Button btOK = new Button("OK");
  Button btCancel = new Button("Cancel");
  OKHandlerClass handler1 = new OKHandlerClass();
  btOK.setOnAction(handler1);
  CancelHandlerClass handler2 = new CancelHandlerClass();
  btCancel.setOnAction(handler2);
  pane.getChildren().addAll(btOK, btCancel);

  // Kreiranje scene i postavljanje scene na pozornicu
  Scene scene = new Scene(pane);
  primaryStage.setTitle("HandleEvent"); // Unos naziva poyornice
  primaryStage.setScene(scene); // Postavljanje scene na pozornicu
  primaryStage.show(); // Prikaz pozornice
   }
 }

 class OKHandlerClass implements EventHandler<ActionEvent> {
   @Override
   public void handle(ActionEvent e) {
  System.out.println("OK button clicked");
   }
 }

 class CancelHandlerClass implements EventHandler<ActionEvent> {
   @Override
   public void handle(ActionEvent e) {
```

```
 System.out.println("Cancel button clicked");
  }
}
```

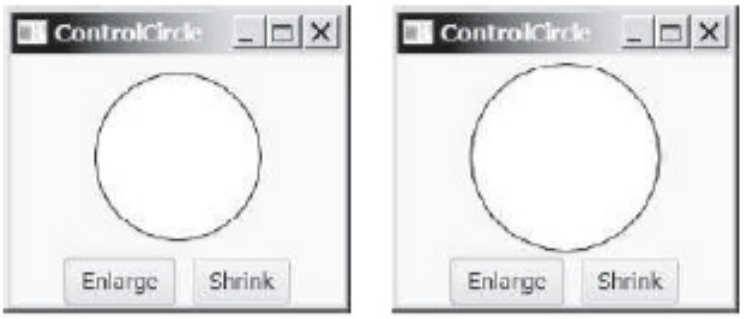
Kada Button objekat ispali ***ActionEvent***, odgovarajući obrađivač ***ActionEvent*** događaja je primerak, tj. objekat klase ***EventHandler<ActionEvent>*** (linija 34). Izvorni objekat poziva metod ***setOnAction(handler)*** koji registruje obrađivač ***(handler),*** kao što je to prikazano u linijama 16, 18, i 19.

Kada kliknete na dugme, objekat ***Button*** ispaljuje ***ActionEvent*** događaj koji prebacuje obrađivaču na obradu metodom ***handle(ActionEvent)***. Objekat događaja (***ActionEvent***) sadrži informacije o događaju, a do kojih se može doći pozivanjem njegovih metoda. Na primer, pozivom metoda; ***e.getSource()*** se dobija izvorni objekat koji je ispalio događaj.

PRIMER KRUGA BEZ OBRAĐIVAČA DOGAĐAJA

Program crta krug u okviru interfejsa i dva dugmeta za povećanje (Enlarge) i smanjivanje (Shrink) kruga.

Ovde se prikazuje primer korišćenja dva dugmeta za povećanje i smanjenje prečnika kruga (slika 2). Program za crtanje kruga i prikazanog korisničkog interfejsa razvićemo inkrementalno, u dve faze. Prvo ćemo razviti program koji pokazuje korisnički interfejs sa krugom u centru prozora (linije 15-19) i sa dva dugmeta (linije 21-27).



Slika 3.2 .2: Korisnik likom na dugme Enlarge povećava krug, a na dugme Shrink, smanjuje krug [1]

```
import javafx.application.Application;
 import javafx.geometry.Pos;
 import javafx.scene.Scene;
 import javafx.scene.control.Button
 import javafx.scene.layout.StackPane;
 import javafx.scene.layout.HBox;
 import javafx.scene.layout.BorderPane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Circle;
 import javafx.stage.Stage;

 public class ControlCircleWithoutEventHandling extends Application {
```

```
  @Override // Predefinisanje metoda start klase Application
  public void start(Stage primaryStage) {
 StackPane pane = new StackPane();
 Circle circle = new Circle(50);
 circle.setStroke(Color.BLACK);
 circle.setFill(Color.WHITE);
 pane.getChildren().add(circle);

 HBox hBox = new HBox();
 hBox.setSpacing(10);
 hBox.setAlignment(Pos.CENTER);
 Button btEnlarge = new Button("Enlarge");
 Button btShrink = new Button("Shrink");
 hBox.getChildren().add(btEnlarge);
 hBox.getChildren().add(btShrink);

 BorderPane borderPane = new BorderPane();
 borderPane.setCenter(pane);
 borderPane.setBottom(hBox);
 BorderPane.setAlignment(hBox, Pos.CENTER);

 // Kreiranje scene i njeno postavljanje na pozornicu
 Scene scene = new Scene(borderPane, 200, 150);
 primaryStage.setTitle("ControlCircle"); // Unis nayiva pozornicee
 primaryStage.setScene(scene); // Postavljanje scene na poyornicu
 primaryStage.show(); // Prikaz pozornice
  }
}
```

PRIMER KRUGA SA OBRAĐIVAČEM DOGAĐAJA

Ovde se daje proširenje prethodnog programa sa metodom koji uvećava poluprečnik kruga klikom da dugme Enlarge.

Ovde ćemo prikazati novu verziju prethodnog programa, koja omogućava povećanje ili smanjivanje kruga zavisno od upotrebe dugmeta Enlarge i Shrink. To je sprovedeno sledećim dopunama prethodnog programa:

1. Definisana je nova klasa CirclePane za prikaz okna sa krugom (linije 51-68). Pored prikaza kruga, ova klasa omogućava sa svojim metodima, povećanje i smanjenje poluprečnika kruga (linije 60-62, 64-67). Preporučljivo je koristiti jednu klasu za prikaz okna sa krugom zajedno sa odgovrajućim metodima.
2. Kreiranje objekta klase CirclePane i deklarisanje circlePane kao polja podataka koji daje referencu ka ovom objektu (linija 15) u klasi ControlCircle. Metodi ove klase sada pristupaju objektu klase CirclePane preko ovog polja podataka (atributa).
3. Definisanje klase obrađivača EnlargeHandler koji primenjuje interfejs ***EventHandler<ActionEvent>*** (linije 43-48). Da bi se formirala promenljiva koja

predstavlja referencu objekta circlePane kome se pristup aiz metoda obrađivača, definisan je EnlargeHandler kao unutrašnja klasa klase ControlCicle.
4. Registracija obrađivača za Enlarge dugme (linija 29) i.

primena metoda handle() u EnlargeHandler da bi s euvećao krug.

```
import javafx.application.Application;
 import javafx.event.ActionEvent;
 import javafx.event.EventHandler;
 import javafx.geometry.Pos;
 import javafx.scene.Scene;
 import javafx.scene.control.Button;
 import javafx.scene.layout.StackPane;
 import javafx.scene.layout.HBox;
 import javafx.scene.layout.BorderPane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Circle;
 import javafx.stage.Stage;

 public class ControlCircle extends Application {
   private CirclePane circlePane = new CirclePane();

   @Override // Redefinisanje metoda start klase Application
   public void start(Stage primaryStage) {
  // Postavljanje dva dugmeta u objekat HBox
  HBox hBox = new HBox();
  hBox.setSpacing(10);
  hBox.setAlignment(Pos.CENTER);
  Button btEnlarge = new Button("Enlarge");
  Button btShrink = new Button("Shrink");
  hBox.getChildren().add(btEnlarge);
  hBox.getChildren().add(btShrink);

  // Kreiranje i registracija obrađivača događaja
  btEnlarge.setOnAction(new EnlargeHandler());

  BorderPane borderPane = new BorderPane();
  borderPane.setCenter(circlePane);
  borderPane.setBottom(hBox);
  BorderPane.setAlignment(hBox, Pos.CENTER);

  // Kreiranje scene i njeno postavljanje na pozornicu
  Scene scene = new Scene(borderPane, 200, 150);
  primaryStage.setTitle("ControlCircle"); // Unos naziva pozornice
  primaryStage.setScene(scene); // Postavljanje scene na pozornicu
  primaryStage.show(); // Prikaz pozornice
   }

   class EnlargeHandler implements EventHandler<ActionEvent> {
  @Override // Redefinisanje metode handle()
  public void handle(ActionEvent e) {
    circlePane.enlarge();
  }
```

```
  }
}

class CirclePane extends StackPane {
  private Circle circle = new Circle(50);

  public CirclePane() {
 getChildren().add(circle);
 circle.setStroke(Color.BLACK);
 circle.setFill(Color.WHITE);
  }

  public void enlarge() {
 circle.setRadius(circle.getRadius() + 2);
  }

  public void shrink() {
 circle.setRadius(circle.getRadius() > 2 ?
 circle.getRadius() - 2 : circle.getRadius());
  }
}
```

ZADACI ZA SAMOSTALNI RAD

Proverite vaše razumevanje stečenog znanja rešavanjem sledećih zadataka

Dodajte klasi ControlCicle kod koji obrađuje događaj koji se generiše klikom na dugme Shrink, čime se smanjuje poluprečnik kruga. Pored ovoga, odgovorite (sebi) na sledeća pitanja:

1. Zašto je obrađivač događaja primerak odgovarajućeg interfejsa obrađivača?
2. Objasni kako se registruje objekat obrađivača i kako se primenjuje interfejs obrađivača.
3. Šta je metod obrađivača za interfejs EventHandler<ActionEvent>?
4. Šta je metod regitracije za dugme koje registruje obrađivač zaa ActionEvent događaj?

Poglavlje 4

Unutrašnje klase

ŠTA JE UNUTRAŠNJA KLASA?

Unutrašnja klasa, ili povezana klasa, je klasa koja je definisana unutar neke druge klase. Unutrašnje klase su korisne za definisanje klase obrađivača događaja.

Slika 1.a definiše dve posebne klase, klasa Test i klasa A. Kod na slici 1.b definiše klasu A kao unutrašnju klasu klase Test.

Klasa **InnerClass** je definisana unutar klase **OuterClass** na slici 1.c. Unutrašnja klasa se može koristiti kao i normalna klasa. Klasa se definiše kao unutrašnja ako se isključivo koristi od svoje spoljnje klase. Unutrašnja klasa ima sledeća svojstva:

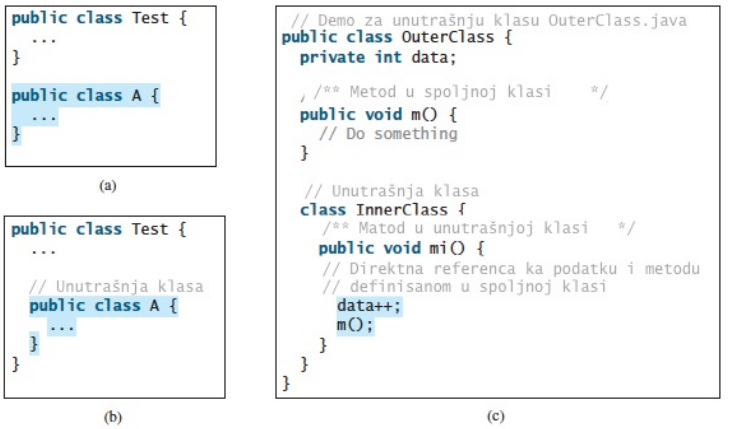
- Unutrašnja klasa se kompilira u klasi ***OuterClassName$InnerClassName***. Na primer, unutrašnja klasa A u klasa Test se kompilira u klasu **Test$A** na slici 1.b.
- Unutrašnja klasa može da pristupa podacima i metodima definisane u spoljnoj klasi sa kojom je povezana, te ne morate da prenesete referencu na objekat spoljnje klase u konstruktoru unutrašnje klase. Zbog toga, unutrašnje klase mogu da učine programe jednostavnijim i kraćim. Na primer, objekat **circlePane**, je definisan u programu **ControlCircle** (linija 15). Može joj se pristupati u unutrašnjoj klasi **EnlargeHandler** u liniji 46.
- Unutrašnja klasa se definiše sa modifikatorom vidljivosti sa pravilima vidljivosti koja su ista kao i za ostale članove klase.
- Unutrašnja klasa se može definisati kao static. Tada joj se pristupa upotrebom imena spoljnje klase. Statička unutrašnja klasa ne može da pristupa nestatičke članove spoljnje klase. sss

- Objekti unutrašnje klase se često kreiraju u spolnjoj klasi. Međutim, možete da kreirate objekat unutrašnje klase iz neke druge klase. Ako je unutrašnja klasa nestatička, morate prvo da kreirate primerak (objekat) spoljnje klase, a onda upotrebite sledeću sintaksu da bi kreirali objekat unutrašnje klase:

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

Ako je unutrašnja klasa statička, objekat se stvara sledećom sintaksom:

```
OuterClass.InnerClass innerObject = new OuterClass.InnerClass();
```



Slika 4.1 .1: Unutrašnje klase predstavljaju zavisne klase jedne primarne klase [1]

UPOTREBA UNUTRAŠNJIH KLASA

Unutrašnje klase se koriste za obuhvatanje zavisnih klasa u primarnoj klasi, kao i kod izbegavanja konfliktnih situacija u vezi imena klasa.

Unutrašnje klase se koriste za obuhvatanje zavisnih klasa u primarnoj klasi. Na ovaj način se smanjuje broj fajlova sa izvornim klasama. To olakšava i organizaciju datoteka sa klasama, jer dobijaju naziv primarne klase. Na primer, umesto da se kreiraju dve datoteke (fajla) sa izvornim kodom za klase **A** i **Test** (na slici 1.a), možete da spojite klasu **A** sa klasom **Test** i da dobijete samo jednu datoteku **Test.java** kao što je pokazano na slici 1.b. Tako se dobijaju datoteke sa klasama pod nazivom: **Test.class** i **Test$A.class**

Druga praktična primena unutrašnjih klasa je kod izbegavanja konfliknih situacija u vezi imena klasa. Na primer, u ranije prikazanim listinzima programa ***ControlCircleWithoutEventHandling*** i ControlCicre koristile su se dve verzije klase ***CirclePane***. Da bi izbegli sukob imena, možete da ih definišete kao unutrašnje klase

Klasa obrađivača se koristi za kreiranje objekta obrađivača za GUI komponente. (npr. za dugme). Klasa obrađivač se ne deli sa drugim aplikacijama te se može definisati unutar glavne klase kao unutrašnja klasa.

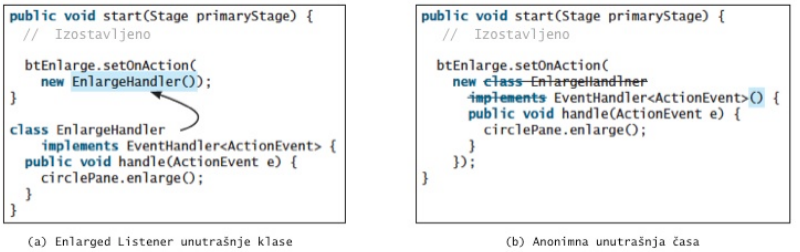
Poglavlje 5

Obrađivači anonimnih unutrašnjih klasa

ŠTA JE ANONIMNA UNUTRAŠNJA KLASA?

Anonimna unutrašnja klasa je unutrašnja klasa bez naziva. Ona kombinuje definisanje unutrašnje klase i kreiranje primerka (objekta) te klase, u jednom koraku.

Unutrašnja klasa za obradu događaja se može skratiti i pojednostaviti upotrebom anonimnih unutrašnjih klasa, kao što je prikazano na slici 1.



Slika 5.1 .1: Razlika unutrašnje klase i anonime unutrašnje klase [1]

Sintaksa za kreiranje anonimne unutrašnje klase je sledeći:

```
new SuperClassName/InterfaceName() {
   // Implement or override methods in superclass or interface
   // Other methods if necessary
 }
```

Anonimna unutrašnja klasa ima sledeća svojstva:

- mora da proširi (***extends***) neku superklasu ili da primeni neki interfejs (***implements***);
- mora da primeni sve apstraktne metode super klase;
- uvek koristi konstruktor bez argumenata Object();
- kompilira se u klasu sa nazivom OuterClassName**$**n.class . Na primer, spoljnja klasa Test ima dve unutrašnje klase koje posle kompilcije se prikazuju u dve datoteke pod nazivima: Test**$**1.class i Test**$**2.class .

PRIMER KORIŠĆENJA ANONIMNE UNUTRAŠNJE KLASE

Potrebno je razviti program koji obrađuje događaje koji se kreiraju klikom na četiri dugmeta korisničkog interfejsa primenom anonimne unutrašnje klase.

Ovde je prikazan listing programa, tj. klase **AnonymousHandlerDemo**. Program omogućava korišćenje korisničkog interfejsa prikazanog na slici 2, a koji sadrži četiri dugmeta. Klikom na neko od ovih dugmeta, stvara se odgovarajuži događaj koji program onda obrađuje i pretvara u željenu akciju.

Slika 5.2 .2: Program obrađuje događaje nastale dejstvom četiri dugmadi [1]

Program kreira četiri obrađivača događaje, tj. procedura za njihovu obradu upotrebom anonimne unutrašnje klase (linije 24-50). Ako se ne bi koristila anonimna unutrašnja klasa, bilo bi potrebno formirati četiri nezavisne klase. Anonimni obrađivač radi na isti način kao što radi obrađivač unutrašnje klase. Program je kondenzovan upotrebom anonimne unutrašnje klase. Posle kompilacije, dobijaju se sledeće datoteke:

AnonymousHandlerDemo$1.class , AnonymousHandlerDemo$2.class ,

AnonymousHandlerDemo$3.class , i **AnonymousHandlerDemo$4.class** .

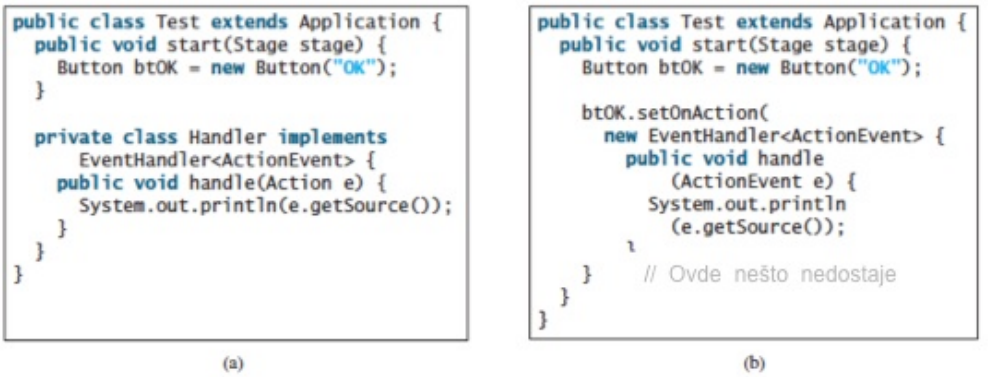
```
import javafx.application.Application;
 import javafx.event.ActionEvent;
 import javafx.event.EventHandler;
 import javafx.geometry.Pos;
 import javafx.scene.Scene;
 import javafx.scene.control.Button;
 import javafx.scene.layout.HBox;
 import javafx.stage.Stage;

 public class AnonymousHandlerDemo extends Application {
   @Override // Predefinisanje metoda start() klase Application
   public void start(Stage primaryStage) {
  // Sadrži dva dugmeta u HBox
  HBox hBox = new HBox();
  hBox.setSpacing(10);
  hBox.setAlignment(Pos.CENTER);
  Button btNew = new Button("New");
  Button btOpen = new Button("Open");
  Button btSave = new Button("Save");
  Button btPrint = new Button("Print");
  hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
```

```
// Kreira i registruje obrađivač događaja
btNew.setOnAction(new EventHandler<ActionEvent>() {
  @Override // Predefinisanje metoda handle()
  public void handle(ActionEvent e) {
    System.out.println("Process New");
  }
});

btOpen.setOnAction(new EventHandler<ActionEvent>() {
  @Override // Predefinisanje metoda handle()
  public void handle(ActionEvent e) {
    System.out.println("Process Open");
  }
});

btSave.setOnAction(new EventHandler<ActionEvent>() {
@Override // Predefinisanje metoda handle()
  public void handle(ActionEvent e) {
    System.out.println("Process Save");
  }
});

btPrint.setOnAction(new EventHandler<ActionEvent>() {
  @Override // Predefinisanje metoda handle()
  public void handle(ActionEvent e) {
    System.out.println("Process Print");
  }
});

// Kreiranje scene i njeno postavljenje na pozornicu
Scene scene = new Scene(hBox, 300, 50);
primaryStage.setTitle("AnonymousHandlerDemo"); Unos naziva
primaryStage.setScene(scene); // Postavljanje scene na pozornicu
primaryStage.show(); // Prikaz pozornice
 }
}
```

ZADACI ZA SAMOSTALAN RAD

Proverite razumevanje primene anonimnih unutrašnjih klasa u obradi događaja kreiranih korišćenjem korisničkog interfejsa.

1. Ako je klasa A unutrašnja klasa u klasi B, šta je datoteka (fajl) za klasu A? Ako klasa B sdrži dve anonimne unutrašnje klase, koja su imena datoteka sa ovim klasama?
2. Šta ne valja u sledećem kodu?



```
public class Test extends Application {
  public void start(Stage stage) {
    Button btOK = new Button("OK");
  }

  private class Handler implements
      EventHandler<ActionEvent> {
    public void handle(Action e) {
      System.out.println(e.getSource());
    }
  }
}
```

(a)

```
public class Test extends Application {
  public void start(Stage stage) {
    Button btOK = new Button("OK");

    btOK.setOnAction(
      new EventHandler<ActionEvent> {
        public void handle
            (ActionEvent e) {
          System.out.println
            (e.getSource());
        }
    }     // Ovde nešto nedostaje
  }
}
```

(b)

Slika 5.3 .3: Primer progograma sa greškama [1]

Poglavlje 6

Primena Lambda izraza

ŠTA JE LAMBDA ISKAZ?

Lambda izraz je anonimna klasa sa pojednostavljenom sintaksom. Primenom lambda izraza može se znatno uprostiti programiranje postupka rada sa događajima.

Lamda izraz je novo svojstvo predstavljeno sa Java 8. Lambda izraz je anonimna klasa sa pojednostavljenom sintaksom. Na primer, dat kod na slici 1.a se može znatno uprostiti primenom lambda izraza u (b) gde je program sveden na tri linije.

```
btEnlarge.setOnAction(
  new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      // Programski kod za obradu događaja e
    }
  }
});
```

(a) Anonimna unutrašnja klasa za obradu događaja

```
btEnlarge.setOnAction(e -> {
  // Programski kod za obradu događaja e
});
```

(b) Lamda izraz za obradu događaja

Slika 6.1 .1: Upoređenje rešenja sa anonimnom unutrašnjom klasom i Lamda izrazom [1]

Osnovna sintaksa Lamda izraza ima sledeći oblik::

```
(type1 param1, type2 param2, ...) -> expression
```

ili:

```
(type1 param1, type2 param2, ...) -> { statements; }
```

Tip podataka za parametar Lambda izraza može da se eksplicitno deklariše ili implicitno od strane kompajlera. Zagrade se se mogu izostaviti ako postoji samo jedan parametar bez eksplicitnog tipa podatka.U prethodnom primeru, lamda izkaz dobija sledeći oblik:

```
e -> {
 // Programski kod za obradu događaja e
 }
```

Računar radi sa lambda izrazom kao da je objekat koji je kreiran od strane anonimne unutrašnje klase. Taj objekat treba da bude primerak (objekat) klase ***EventHandler<ActionEvent>.***.

Kako interfejs ***EventHandler*** definiše metod obrade događaja tipa ***Action Event***, računar automatski prepoznaje da je **e** parametar tipa ***ActionEvent,*** i da su iskazi u telu metoda obrađivača događaja. Interfejs ***EventHandler*** sadrži samo jedan metod. Naredbe u lambda iskazu pripadaju tom metodu. Ako bi sadržao više metoda, računar ne bi mogao da kompilira lambda iskaz.. Zato, da bi računar razumeo lamda iskaze, interfejs može da sadrži tačno jedan abstraktni metod. Takav interfejs je poznat kao funkcionalni interfejs ili kao ***Single Abstract Method (SAM)*** interfejs.

PRIMER PRIMENE LAMBDA ISKAZA

Program kreira četiri obrađivača događaja upotrebom lamda iskaza, koji omogućavaju dobijanje kraćeg, čistijeg i jasnijeg programskog koda.

Program kreira četiri obrađivača događaja upotrebom lamda iskaza (linija 23-35). Upotrebom lambda iskaza, dobija sa kraći i čistiji kod. Kao što se može videti u ovom primeru, lambda iskazi mogu da imaju više varijanti. Linija 23 upotrebljava deklarisan tip. Linija 27 upotrebljava izveden tip jer se tip može odrediti automatski od strane računara. Linija 31 izostavlja zagrade jer telo iskaza ima samo jednu naredbu.

Možete obrađivati događaje definisanjem klase za obradu upotrebom unutrašnjih klasa, anonimnih unutrašnjih klasa, ili lambda iskaze. Preporučujemo da koristite lambda iskaze jer se na taj način dobija kraći, čistiji i jasniji programski kod.

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

 public class LambdaHandlerDemo extends Application {
   @Override // Predefiniše metod start() klase Application
   public void start(Stage primaryStage) {
  // Sadrži dva dugmeta u HBox
  HBox hBox = new HBox();
  hBox.setSpacing(10);
  hBox.setAlignment(Pos.CENTER);
  Button btNew = new Button("New");
  Button btOpen = new Button("Open");
  Button btSave = new Button("Save");
  Button btPrint = new Button("Print");
  hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);

  // Kreira i registrira obrađivač
  btNew.setOnAction((ActionEvent e) -> {
    System.out.println("Process New");
  });
```

```
  btOpen.setOnAction((e) -> {
    System.out.println("Process Open");
  });

  btSave.setOnAction(e -> {
    System.out.println("Process Save");
  });

  btPrint.setOnAction(e -> System.out.println("Process Print"));

  // Kreira scenu i postavlja je na pozornicu
  Scene scene = new Scene(hBox, 300, 50);
  primaryStage.setTitle("LambdaHandlerDemo"); // Unos naslova pozornice
  primaryStage.setScene(scene); // Postavljanje scene na pozornicu
  primaryStage.show(); // Prikaz pozornice
   }
}
```

VIDEO - ANONIMNE UNUTRAŠNJE KLASE I LAMBDA ISKAZ

JavaFX Java GUI Tutorial - 3 - Anonymous Inner Classes and Lambda

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADACI ZA SAMOSTALNI RAD

Proverite vaše razumevanje funkcije i primene lambda iskaza.

1. Šta je lamda iskaz? Koje koristi donosi upotreba lambda iskaza u obradi događaja? Koja je sintaksa lambda iskaza?
2. Šta je funkcionalni iskaz? Zašto je potreban funkcionalni iskaz za lambda iskaz?
3. Prikažite rezultat izvršenja sledećeg programa.

```
public class Test {
 public static void main(String[] args) {
   Test test = new Test();
    test.setAction1(() -> System.out.print("Action 1! "));
    test.setAction2(e -> System.out.print(e + " "));
    System.out.println(test.setAction3(e -> e * 2));
   }
   public void setAction1(T1 t) {
    t.m();
   }
   public void setAction2(T2 t) {
```

```
    t.m(4.5);
  }
  public double setAction3(T3 t) {
    return t.m(5.5);
  }
}
interface T1 {
   public void m();
}
interface T2 {
   public void m(Double d);
}
interface T3 {
   public double m(Double d);
}
```

Poglavlje 7

Studija slučaja: Kalkulator kredita

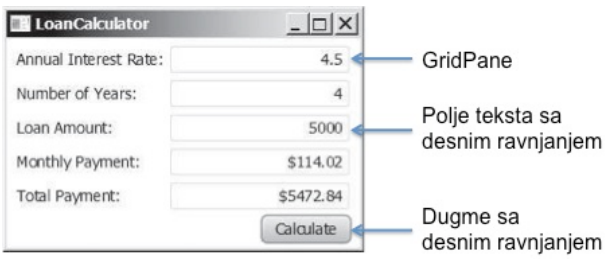
LISTING PROGRAMA LOANCALCULATOR

Ova studija slučaja razvija kalkulator kredita upotrebom programiranja uz korišćenje događaja sa kontrolom GUI.

Potrebno je razviti program koji računa ukupnu i mesečnu otplatu kredita, kada su poznati uslovi dobijanja kredita (kamata, iznos kredita i period vraćanja kredit). S tim ciljem treba uraditi sledeće:

1. Kreirati korisnički interfejsa u skladu sa slikom1.
2. Kreiranje ***GridPane*** . Dodavanje natpisa, i drugmeta u oknu.
3. Postaviti desno ravnjanje za dugme.
4. Obraditi događaj

Kreirajte i registrujte obrađivač za obradu akcionog događaja do koga dolazi klikom dugmenta. Obrađivač dobija korisnički unos visine kredite, kamate i broj godina za vraćanje kredita.. Na osnovo toga kompjuter treba da izračuna visinu mesečne otplate kredita i ukupni iznos novca koji mora da se plata povraćajem kredita. Rezultat se prikazuje u poljima za prikaz teksta GUI.



Slika 7.1 .1: GUI programa koji treba razviti [1]

```
import javafx.application.Application;
 import javafx.geometry.Pos;
 import javafx.geometry.HPos;
 import javafx.scene.Scene;
 import javafx.scene.control.Button;
 import javafx.scene.control.Label;
 import javafx.scene.control.TextField;
 import javafx.scene.layout.GridPane;
 import javafx.stage.Stage;

 public class LoanCalculator extends Application {
   private TextField tfAnnualInterestRate = new TextField();
```

```
 private TextField tfNumberOfYears = new TextField();
 private TextField tfLoanAmount = new TextField();
 private TextField tfMonthlyPayment = new TextField();
 private TextField tfTotalPayment = new TextField();
 private Button btCalculate = new Button("Calculate");

 @Override // Predefinisanje metoda start() klase Application
 public void start(Stage primaryStage) {
// Kreiranje korisničkog interfejsa (KI)
GridPane gridPane = new GridPane();
gridPane.setHgap(5);
gridPane.setVgap(5);
gridPane.add(new Label("Annual Interest Rate:"), 0, 0);
gridPane.add(tfAnnualInterestRate, 1, 0);
gridPane.add(new Label("Number of Years:"), 0, 1);
gridPane.add(tfNumberOfYears, 1, 1);
gridPane.add(new Label("Loan Amount:"), 0, 2);
gridPane.add(tfLoanAmount, 1, 2);
gridPane.add(new Label("Monthly Payment:"), 0, 3);
gridPane.add(tfMonthlyPayment, 1, 3);
gridPane.add(new Label("Total Payment:"), 0, 4);
gridPane.add(tfTotalPayment, 1, 4);
gridPane.add(btCalculate, 1, 5);

// Unos svojstava KI
gridPane.setAlignment(Pos.CENTER);
tfAnnualInterestRate.setAlignment(Pos.BOTTOM_RIGHT);
tfNumberOfYears.setAlignment(Pos.BOTTOM_RIGHT);
tfLoanAmount.setAlignment(Pos.BOTTOM_RIGHT);
tfMonthlyPayment.setAlignment(Pos.BOTTOM_RIGHT);
tfTotalPayment.setAlignment(Pos.BOTTOM_RIGHT);
tfMonthlyPayment.setEditable(false);
tfTotalPayment.setEditable(false);
GridPane.setHalignment(btCalculate, HPos.RIGHT);

// Događaji procesa
btCalculate.setOnAction(e -> calculateLoanPayment());

// Kreiranje scene i njeno postavljanje na pozornicu
Scene scene = new Scene(gridPane, 400, 250);
primaryStage.setTitle("LoanCalculator"); // Set title
primaryStage.setScene(scene); // Place the scene in the stage
primaryStage.show(); // Display the stage
 }

 private void calculateLoanPayment() {
// Dobijanje vrednosti iz polja sa tekstom
double interest =
Double.parseDouble(tfAnnualInterestRate.getText());
int year = Integer.parseInt(tfNumberOfYears.getText());
double loanAmount =
  Double.parseDouble(tfLoanAmount.getText());
```

```
  // Kreiranje objekta kredita. Listing klase Loan je definisan u posebno
  Loan loan = new Loan(interest, year, loanAmount);

  // Prikaz mesečne otplate i ukupne otplate
  tfMonthlyPayment.setText(String.format("$%.2f",
  loan.getMonthlyPayment()));
  tfTotalPayment.setText(String.format("$%.2f",
  loan.getTotalPayment()));
   }
 }
```

LISTING KLASE LOAN

Klasa Loan izračunava mesečnu ratu otplate kredita i ukupnu visinu kredita koji treba vratiti.

Korisnički interfejs se kreira u metodu **start()** . Dugme je izvor događaja. Obrađivač događaje se kreira i registruje kod izvornog objekta – dugmeta. Obrađivač događaja kreiran od strane dugmeta poziva metod **calculateLoanPayment()** da bi se dobila kamata (linija 60), broj godina (62) i visina kredita (linija 63). Pozivanjem **tfAnnualInterestRate.getText()** dobija se string tekst u polju teksta tfAnnualInterestRate text field.

Klasa **Loan** se koristi za proračun otplate kredita. Pozivanjem metoda loan**.getMonthlyPayment()** koji vraća visinu mesečne otplate kredita (linija 54). Pozivom metoda **setText()** polja teksta postavlja vrednost stringa u polje teksta. .

```
public class Loan {
   private double annualInterestRate;
   private int numberOfYears;
   private double loanAmount;
   private java.util.Date loanDate;

   /** Početni konstruktor */
   public Loan() {
   this(2.5, 1, 1000);
   }

   /** Konstruiše kredit sa unetom kamatom,
   rojem godina otplate i visinom kredita
   */
   public Loan(double annualInterestRate, int numberOfYears,
  double loanAmount) {
  this.annualInterestRate = annualInterestRate;
  this.numberOfYears = numberOfYears;
  this.loanAmount = loanAmount;
  loanDate = new java.util.Date();
   }

   /** Vraća godišnju kamatu */
```

```
  public double getAnnualInterestRate() {
 return annualInterestRate;
  }

  /** Unosi novu godišnju kamatu */
  public void setAnnualInterestRate(double annualInterestRate) {
 this.annualInterestRate = annualInterestRate;
  }

  /** Vraća broj godina otplate kredita */
  public int getNumberOfYears() {
 return numberOfYears;
  }

  /** Unosi broj otplate kredita  */
  public void setNumberOfYears(int numberOfYears) {
 this.numberOfYears = numberOfYears;
  }

  /** Vraća visinu kredita */
  public double getLoanAmount() {
 return loanAmount;
  }

  /** Unos visine kredita  */
  public void setLoanAmount(double loanAmount) {
 this.loanAmount = loanAmount;
  }

  /** Nalaženje mesečne rate otplate kredita */
  public double getMonthlyPayment() {
 double monthlyInterestRate = annualInterestRate / 1200;
 double monthlyPayment = loanAmount * monthlyInterestRate / (1 -
   (1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12)));
 return monthlyPayment;
  }

  /** Određivanje ukupnog duga */
  public double getTotalPayment() {
 double totalPayment = getMonthlyPayment() * numberOfYears * 12;
 return totalPayment;
  }

  /** Vraća datum odobrenja kredita  */
  public java.util.Date getLoanDate() {
 return loanDate;
  }
}
```

Poglavlje 8

Događaji miša

DOGAĐAJ MIŠA - MOUSEEVENT

MouseEvent se javlja uvek kada se pritisne, otpusti i klikne dugme miša, kao i kada se miš kreće ili kada sa vrši vučenje nekog čvora ili scene mišem

Događaj miša: Objekat ***MouseEvent*** sadrži informacije o događaju miša, kao što su broj klikova, lokaciju miša (x i y koordinate), kao i koje je dugme pritisnuto, kao što se može videti na slici 1.

Četiri konstante: ***PRIMARY*** ,***SECONDARY*** ,***MIDDLE*** i ***NONE*** koje su definisane u klasi ***MouseButton***, označavaju levo, desno i srednje dugme miša, odnosno odsustvo dugmeta miša (NONE). Možete koristiti metod ***getButton()*** radi utvrđivanja koje je dugme pritisnuto. Na primer, *getButton() == MouseButton.SECONDARY* označava da pritisnuto desno dugme miša.

javafx.scene.input.MouseEvent	
+getButton(): MouseButton	Ukazuje na dugme mi[a koje je kori[ćeno
+getClickCount(): int	Vraća broj klikova miša povezanih sa ovim događajem.
+getX(): double	Vraća x-koordinatu tačke miša u izvornom čvoru miša.
+getY(): double	Vraća y-koordinatu tačke miša u izvornom čvoru miša.
+getSceneX(): double	Vraća x-koordinatu tačke mišau sceni
+getSceneY(): double	Vraća y-koordinatu tačke miša u sceni
+getScreenX(): double	Vraća x-koordinatu tačke miša na ekranu.
+getScreenY(): double	Vraća y-koordinatu tačke miša na ekranu.
+isAltDown(): boolean	Vraća true ako je pritisnut taster Altna na ovaj događaj
+isControlDown(): boolean	Vraća true ako je pritisnut taster Control na ovaj događaj
+isMetaDown(): boolean	Vraća true ako je pritisnuto Meta dugme miša na ovaj događaj
+isShiftDown(): boolean	Vraća true ako je pritisnuta taster Shift na ovaj događaj

Slika 8.1 .1: Klasa MouseEvent [1]

PRIMER UPOTREBE MIŠA

Prikaz poruke u oknu i njegovo pomeranje vučenjem miša preko njega.

U ovom primeru se daje program koji omogućava prikaz poruke u oknu i njeno pomeranje pomoću miša. Poruka se pomera kada se vrši vučenje okna pomožu miša (miš se dovede do okna, pritisne dugme i pomera miš u nekom pravcu).

Na slici 2 prikazuje se rezultat koji se dobija izvršenjem programa MouseEventDemo.

Slika 8.2 .2: Okno sa porukom koje se može pomerati pomožu miša [1]

Svali čvor ili scenamože da proizvede događaj. Program kreira objekat Text (linija 12) i registruje obrađivač događaja koji se javlja kada se vrši vučenje okna pomoću miša. (linija 14). Uvek kada dođe do pomeranja miša, tekst se pomera u koordinate x- i z-koordinate definisane položajem miša (linije 15 i 16).

```
import javafx.application.Application;
 import javafx.scene.Scene;
 import javafx.scene.layout.Pane;
 import javafx.scene.text.Text;
 import javafx.stage.Stage;

 public class MouseEventDemo extends Application {
   @Override // Redefinisanje metoda start() klase Application
   public void start(Stage primaryStage) {
  // Kreiranje okna i postavljanje njegovih svojstava
  Pane pane = new Pane();
  Text text = new Text(20, 20, "Programming is fun");
  pane.getChildren().addAll(text);
  text.setOnMouseDragged(e -> {
    text.setX(e.getX());
    text.setY(e.getY());
  });

  // Kreiranje scene i njeno postavljanje na pozornicu
  Scene scene = new Scene(pane, 300, 100);
  primaryStage.setTitle("MouseEventDemo"); // Unos naziva pozornice
  primaryStage.setScene(scene); // Postavljanje scene na pozornicu e
  primaryStage.show(); // Prikaz pozornice
   }
 }
```

Poglavlje 9

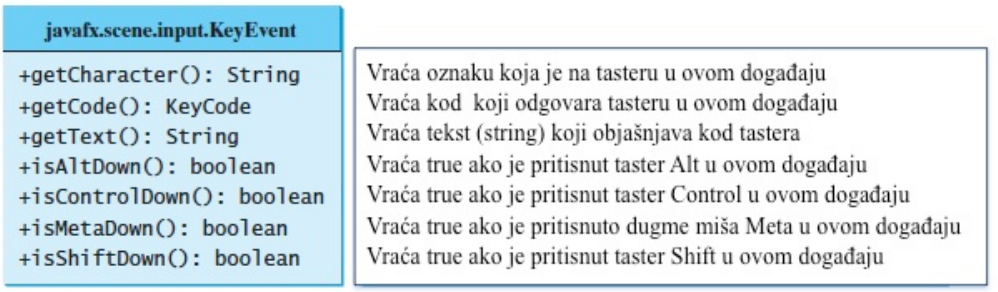
Događaji tastature

DOGAĐAJ TASTATURE - KEYEVENT

KeyEvent se javlja uvek kada dođe do pritiska, opuštanja i kucanja tastera nekog čvora ili scene.

Događaji tastature omogućavaju korisnicima da upotrebljavaju tastere da bi kontrolisali i izvršavali akcije ili dobili ulaz sa tastature. Objekat ***KeyEvent*** opisuje prirodu događaja (označava koji je taster pritisnut, opušten ili otkucan) i vrednost koda tastature (slika1).

Svaki događaj tastera ima pridružen kod koji vraća metod ***getCode()*** i klasi ***KeyEvent.*** Kodovi tastera su konstante u klasi ***KeyCode.*** Tabela na slici 2 prikazuje kodove nekih tastera. ***KeyCode*** je ***enum*** tip. Pri pritisku i opuštanju tastera metod ***getCode()*** vraća vrednost definisanu u tabeli. Metod ***getText()*** vraća string koji opisuje kod tastera, a metod ***getCharacter()*** vraća string koji opisuje kod tastera, i ***getCharacter()*** vraća prazan string Za događaj tastera, ***getCode()*** vraća***UNDEFINED***, a ***getCharacter()*** vraća Unicode oznaku ili niz oznaka povezanih sa događajem.



Slika 9.1 .1: Klasa KeyEvent [1]

Konstanta	Opis	Konstanta	Opis
HOME	Taster Home	CONTROL	Taster Control
END	Taster End	SWIFT	Taster Swift
PAGE_UP	Taster Page Up	BACK_SPACE	Taster Backspace
PAGE_DOWN	Taster Page Down	CAPS	Taster velikih slova
UP	Taster srelica gore	NUM_LOCK	Zaključavanje brojčanih tastera
DOWN	Taster sterica dole	ENTER	Taster Enter
LEFT	Taster strelica levo	UNDEFIED	Nepoznat kod tastera
RIGHT	Taster strelica desno	F1 – F12	Funkcionalni tasteri F1-F12
ESCAPE	Taster Esc	0 – 9	Tasteri brojeva 0-9
TAB	Taster Tab	A do Z	Tasteri slova A - Z

Slika 9.2 .2: Konstante koda tastature [1]

PRIMER RADA SA DOGAĐAJIMA TASTATURE

Program prikazuje oznaku u oknu, a može da ga pomera gore, dole, levo ili desno.

Na slici 3 prikazano je okno sa prikazom jedne oznake. Ovde je dat program koji omogućava ovaj prikaz i pomeranje oznaku levo, desno, gore i dole, upotrebom tastera sa strelicom levo, desno, gore i dole.

Slika 9.3 .3: Program reaguje na događaje tastera, prikazom oznake, i pomeranjem oznake gore, dole, levo ili dole [1]

Program kreira okono (linija 11), kreira tekst (liija 12), i postavlja tekst u okno (linij 14). Tekst registruje obrađivač događaja tastera u liniama 15-25. Kada se taster pritisne, poziva se obrađivač događaja. Program koristi e.getCode() (linija 16) d se dobije kod tastera i e.getText() (linija 23) da se dobije oznaka za taster. Kada se pritisne taster bez strelice, oznaka se prikazuje (linija 22 i 23). Kada se pritisne taster sa strelicom, oznaka se pomera u pravcu strelice (linije 17-20). Naredva switch za tip enum, slučajevi za enum konstante (linije 16-24). Konstante su nekvalifikovane..

.

Samo fokusiran čvor može da prime KeyEvent, pozivom requestFocus() na tekstu, omogućava tekst da primi ulaz preko tastera (linija 33). Ovaj metod mora da se pozove posle prikaza pozornice.

```
import javafx.application.Application;
 import javafx.scene.Scene;
 import javafx.scene.layout.Pane;
 import javafx.scene.text.Text;
 import javafx.stage.Stage;

 public class KeyEventDemo extends Application {
   @Override // Redefinisanje metoda start() klase Application
   public void start(Stage primaryStage) {
  // Kreiranje okvira i unos njegovih svojstava
  Pane pane = new Pane();
  Text text = new Text(20, 20, "A");

  pane.getChildren().add(text);
  text.setOnKeyPressed(e -> {
    switch (e.getCode()) {
      case DOWN: text.setY(text.getY() + 10); break;
      case UP: text.setY(text.getY() - 10); break;
      case LEFT: text.setX(text.getX() - 10); break;
      case RIGHT: text.setX(text.getX() + 10); break;
      default:
        if (Character.isLetterOrDigit(e.getText().charAt(0)))
          text.setText(e.getText());
    }
  });

  // Kreiranje scene i njeno postavljanje na pozornicu
  Scene scene = new Scene(pane);
  primaryStage.setTitle("KeyEventDemo"); // Unos naziva pozornice
  primaryStage.setScene(scene); // Postavljanje scene na pozornicu
  primaryStage.show(); // Prikaz scene

  text.requestFocus(); // tekst je usmeren ka primanju teksta
   }
 }
```

PRIMER SA PROMENOM VELIČINE KRUGA

Krug se povećava ili smanjuje pritiskom levog ili desnog dugmeta miša, odnosno, tastera U ili D.

Može se dodati veću kontrolu u programu ***ControlCircle***, da bi se povećao ili smanjio poluprečnik klikom na levo ili desno dugme miša ili pritiskom tastera U i D. Novi program je ovde prikazan, ***ControlCircleWithMouseAndKey***. Obrađivač događaja tastera se kreira u linijama 29-36. Ako se klikne levo dugme miša, krug se povećava (linije 30-32); ako klikne desno dugme miša, krig se smanjuje (linije 33-35).

Obrađivač događaja atstera je kreiran u linijama 38-45. Ako se pritisne dugme U, krug se povećava, (linije 39-41); ako se taster D pritisne , krug se smanjuje (linije 42-44). Pozivom

requestFocus() na ***circlePane*** (lina 58) čini da ***circlePane*** prima događaje tastera. Ako se pritisne neko dugme, ***circlePane*** nije više fokusiran. Da bi se problem rešio, ponovnim pozivom ***requestFocus()*** na ***circlePane*** posle klika svakog dugmeta.

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

 public class ControlCircleWithMouseAndKey extends Application {
   private CirclePane circlePane = new CirclePane();

   @Override // Redefinisanje metoda start() klase Application
  // Sadrži dva dugmeta u HBox
  HBox hBox = new HBox();
  hBox.setSpacing(10);
  hBox.setAlignment(Pos.CENTER);
  Button btEnlarge = new Button("Enlarge");
  Button btShrink = new Button("Shrink");
  hBox.getChildren().add(btEnlarge);
  hBox.getChildren().add(btShrink);

  // Kreira i registruje obrađivač
  btEnlarge.setOnAction(e -> circlePane.enlarge());
  btShrink.setOnAction(e -> circlePane.shrink());

  circlePane.setOnMouseClicked(e -> {
    if (e.getButton() == MouseButton.PRIMARY) {
      circlePane.enlarge();
    }
    else if (e.getButton() == MouseButton.SECONDARY) {
          circlePane.shrink();
    }
  });

  circlePane.setOnKeyPressed(e -> {
    if (e.getCode() == KeyCode.U) {
       circlePane.enlarge();
    }
    else if (e.getCode() == KeyCode.D) {
      circlePane.shrink();
    }
  });

  BorderPane borderPane = new BorderPane();
  borderPane.setCenter(circlePane);
```

```
      9   borderPane.setBottom(hBox);
 BorderPane.setAlignment(hBox, Pos.CENTER);

 // Kreiranje scene i njeno postavljanje na pozornicu
 Scene scene = new Scene(borderPane, 200, 150);
 primaryStage.setTitle("ControlCircle"); // Unos naziva pozornice
 primaryStage.setScene(scene); // Postavljanje scene na pozornicu
 primaryStage.show(); // Prikaz pozornice

 circlePane.requestFocus(); // Zahtevani fokus na circlePane
  }
}
```

Poglavlje 10

Osluškivači osmatranih objekata

OBJEKTI OSLUŠKIVAČI

Možete dodati osluškivače (listeners) procesu menjanja vrednosti u nekom objektu koji je osmatran.

Primerak klase ***Observable*** je objekat osmatranja, koji sadrži metod ***addListener(InvalidationListener listener)*** koji dodaje slušaoca. Klasa slušaoca mora da primeni interfejs **InvalidationListener** da bi redefinisao metod ***invalidated(Observable o)*** za obradu događaja promene vrednosti kod objekta osmatranja. Kada dođe do promene vrednosti kod objekta osmatranja, tj. objekta klase Observable, osluškivač se obaveštava pozivom njegovog metoda i***nvalidated(Observable o)***. Svako povezano svojstvo je primerak klase ***Observable.*** Ovde prikazan listing programa **ObservablePropertyDemo** daje primer osmatranja i obrade događaja koji se javlja kada dođe do promene ravnoteže objekata klase DoubleProperty.

Kada se izvrši linija 16, dolazi do promene ravnoteže, što obaveštava osluškivač pozivom metod **invalidated()** osluškivača. Kao što se vidi, anonimna unutrašnja klasa u linijama 9-14 može da se pojednostavi upotrebom lamda iskaza:

```
balance.addListener(ov -> {
System.out.println("The new value is " +
balance.doubleValue());
});
```

```
import javafx.beans.InvalidationListener;
 import javafx.beans.Observable;
 import javafx.beans.property.DoubleProperty;
 import javafx.beans.property.SimpleDoubleProperty;

 public class ObservablePropertyDemo {
   public static void main(String[] args) {
   DoubleProperty balance = new SimpleDoubleProperty();
     balance.addListener(new InvalidationListener() {
      public void invalidated(Observable ov) {
        System.out.println("The new value is " +
            balance.doubleValue());
      }
  });

  balance.set(4.5);
   }
```

```
}
```

PRIMER: PRIKAZ SATA BEZ MENJANJA NJEGOVE VELIČINE

Pre prikaza programa koji omogućava promenu veličine analognog sata, ovde se navode listinzi klasa koje crtaju analogni sat bez mogućnosti promene njegove veličine.

U lekciji br. 3 predmeta CS202, šta je studija slučaja u kome se radi program za prikaz analognog sata. Ovde se taj program ponavlja (klase ClockPane i DisplayPane). Ovi programi ne omogućavaju promenu veličine sata promenom veličine okna za prikaz sata.

.

```
import java.util.Calendar;
 import java.util.GregorianCalendar;
 import javafx.scene.layout.Pane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Circle;
 import javafx.scene.shape.Line;
 import javafx.scene.text.Text;

 public class ClockPane extends Pane {
   private int hour;
   private int minute;
   private int second;

   // Širina i visina okna sata
   private double w = 250, h = 250;

   /** Konstruisanje početnog sata sata sa trenutnim vremenom */
   public ClockPane() {
  setCurrentTime();
   }

   /** Konstruisanje sata sa specificiranim satim, minutima i sekundama  */
   public ClockPane(int hour, int minute, int second) {
  this.hour = hour;
  this.minute = minute;
  this.second = second;
  paintClock();
   }

   /** Vraća hour */
   public int getHour() {
  return hour;
   }
```

```
  /** Unisi novu vrednost za  hour */
  public void setHour(int hour) {
 this.hour = hour;
 paintClock();
  }

/** vraća minute */
  public int getMinute() {
 return minute;
  }

  /** Unosi novu vrednost za minute */
  public void setMinute(int minute) {
 this.minute = minute;
 paintClock();
  }

  /** Vraća second */
  public int getSecond() {
 return second;
  }

  /** Unosi novu vrednost za second */
  public void setSecond(int second) {
 this.second = second;
 paintClock();
  }

  /** Vraća širinu okna za sat  */
  public double getW() {
 return w;
  }

  /** Unosi širinu okna za sat */
  public void setW(double w) {
 this.w = w;
 paintClock();
  }

  /** Vraća visinu okna za sat  */
 public double getH() {
 return h;
  }

  /** Podešava visinu okna sata  */
  public void setH(double h) {
 this.h = h;
 paintClock();
  }

  /* Unosi sadašnje vreme u sat  */
  public void setCurrentTime() {
```

```
// Konstruiše kalendar ya sadašnji datum i vreme
Calendar calendar = new GregorianCalendar();

// Unos trenutnog vremena sata, minuta i sekundi
this.hour = calendar.get(Calendar.HOUR_OF_DAY);
this.minute = calendar.get(Calendar.MINUTE);
this.second = calendar.get(Calendar.SECOND);

paintClock(); // nacrtaj sat
 }

 /** Crtanje sata  */
 protected void paintClock() {
   // Inicijalizacija parametra sata
   double clockRadius = Math.min(w, h) * 0.8 * 0.5;
   double centerX = w / 2;
   double centerY = h / 2;

   // Crtanje kruga
   Circle circle = new Circle(centerX, centerY, clockRadius);
   circle.setFill(Color.WHITE);
   circle.setStroke(Color.BLACK);
   Text t1 = new Text(centerX - 5, centerY - clockRadius + 12, "12");
   Text t2 = new Text(centerX - clockRadius + 3, centerY + 5, "9");
   Text t3 = new Text(centerX + clockRadius - 10, centerY + 3, "3");
   Text t4 = new Text(centerX - 3, centerY + clockRadius - 3, "6");

   // Crtanje skazaljke za prikaz sekundi
   double sLength = clockRadius * 0.8;
   double secondX = centerX + sLength *
   Math.sin(second * (2 * Math.PI / 60));
   double secondY = centerY - sLength *
   Math.cos(second * (2 * Math.PI / 60));
   Line sLine = new Line(centerX, centerY, secondX, secondY);
   sLine.setStroke(Color.RED);

   // Crtanje velike skazalje za prikaz minuta
   double mLength = clockRadius * 0.65;
   double xMinute = centerX + mLength *
   Math.sin(minute * (2 * Math.PI / 60));
   double minuteY = centerY - mLength *
   Math.cos(minute * (2 * Math.PI / 60));
   Line mLine = new Line(centerX, centerY, xMinute, minuteY);
   mLine.setStroke(Color.BLUE);

   // Crtanje sklayalje ya prikaz sati
   double hLength = clockRadius * 0.5;
   double hourX = centerX + hLength *
       Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
   double hourY = centerY - hLength *
       Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
   Line hLine = new Line(centerX, centerY, hourX, hourY);
   hLine.setStroke(Color.GREEN);
```

```
    getChildren().clear();
    getChildren().addAll(circle, t1, t2, t3, t4, sLine, mLine, hLine);
  }
}
```

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

 public class DisplayClock extends Application {
   @Override // Predefinisanje metoda start u klasi Application
   public void start(Stage primaryStage) {
  // Kreiranje sata i naslova
  ClockPane clock = new ClockPane();
  String timeString = clock.getHour() + ":" + clock.getMinute()
      + ":" + clock.getSecond();
  Label lblCurrentTime = new Label(timeString);

  // Postavljanje sata i nalepnice nu okno  BorderPane
  BorderPane pane = new BorderPane();
  pane.setCenter(clock);
  pane.setBottom(lblCurrentTime);
  BorderPane.setAlignment(lblCurrentTime, Pos.TOP_CENTER);

  // Kreiranje scene i postavljanje je na pozornicu
  Scene scene = new Scene(pane, 250, 250);
  primaryStage.setTitle("DisplayClock"); // Unos imena pozornice
  primaryStage.setScene(scene); // STavljanje scene na pozornicu
  primaryStage.show(); // Prikaz pozornice
   }
 }
```

PRIMER: PRIKAZ SATA SA MENJANJEM NJEGOVE VELIČINE

Ovde se prikazuje listing klase DisplayResizableClock koja omogućava promenu veličina analognog sata kada se menja veličina okna u kome je prikazan

Ovde se prikazuje program koji omogućuje promenu veličine analognog sata kada se menja veličina okna u kome je prikazan sat. To se postiže tako što se dodaje osluškivač promene veličine okna. Osluškivač se registruje da prati širinu i visinu prozora okna. Kada se promene

ove dimenzije, osluškivač to primećuje i kreira događaj koji onda menja veličinu sata, u skladu sa promenama dimenzija okna u kome se sat nalazi.

Program se ne razlikuje mnogo u odnosu na prethodno prikazan program (ClockPane i DisplayClock) sem što su dodate linij 29-35 u kojima se vrši registracija osluškivača zapraćenje promene dimenzija širine ili visine scene, odn. okna. Program obezbeđuje sinhronizaciju veličine okna sa veličinom scene.

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

 public class DisplayResizableClock extends Application {
   @Override // Redefinisanje metoda start() klase Animation
   public void start(Stage primaryStage) {
    // Kreiranje sata i natpisa
    ClockPane clock = new ClockPane();
    String timeString = clock.getHour() + ":" + clock.getMinute()
       + ":" + clock.getSecond();
    Label lblCurrentTime = new Label(timeString);

    // Postavljanje sata i natpisa u okno sa granicom
    BorderPane pane = new BorderPane();
    pane.setCenter(clock);
    pane.setBottom(lblCurrentTime);
    BorderPane.setAlignment(lblCurrentTime, Pos.TOP_CENTER);

    // Kreirane scene i njeno postavljanje na pozornicu
    Scene scene = new Scene(pane, 250, 250);
    primaryStage.setTitle("DisplayClock"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage

    pane.widthProperty().addListener(ov ->
    clock.setW(pane.getWidth())
  );

  pane.heightProperty().addListener(ov ->
     clock.setH(pane.getHeight())
  );
    }
 }
```

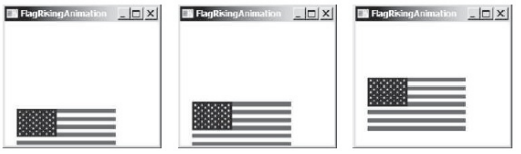
Poglavlje 11

Animacija

KLASA ANIMATION

JavaFX obezbeđuje klasu Animation sa osnovnim funkcijama potrebnim za sve vrste animacija.

Animacija unosi novu dimenziu u grafiku: vreme. Scena sa grafičkim objektima, se menja sa vremenom. Na primer, zastava na slici 1 može da menja svoj položaj tokom vremena. Postavlja se pitanje: Kako to programirati? Da bi se to postiglo,postoji nekoliko načina. Mi ćemo ovde prikazati jedan od njih. Potrebno je da kreirate podklasu apstraktne klase ***Animation***, koju sadrži ***JavaFX*** paket.



Slika 11.1 .1: Animacija dizanja američke zastave [1]

Na slici 2 dat je UML dijagram klase ***Animation***, sa osnovnim svojom funkcionalnošću. Pored ove klase, prikazaćemo još tri ***JavaFX*** klase: ***PathTransition***, ***FadeTransition*** i***Timeline***.

Slika 11.2 .2: Klasa Animation [1]

Svojstvo ***autoReverse*** je logička (Boolean) promenljiva koja označava da li se animacija odvija u suprotnom smeru u sledećem ciklusu. Svojstvo (atribut) ***cycleCount*** označava broj ciklusa animacije. Možete koristiti konstantu ***Timeline.INDEFINITE*** da bi označili beskonačan broj ciklusa. Svojstvo ***rate*** definiše brzinu animacije. Negativna vrednost označava suprotni smer animacije. Svojstvo status je svojstvo koje označava status animacije. Moguće konstante: ***Animation.Status.PAUSED***, ***Animation.Status.RUNNING,***

i ***Animation.Status.STOPPED***. Metodi ***pause(), play(),*** i ***stop()*** čina da se animacija prvremeno ili stalno zaustavi, i da se animacija vrši. .

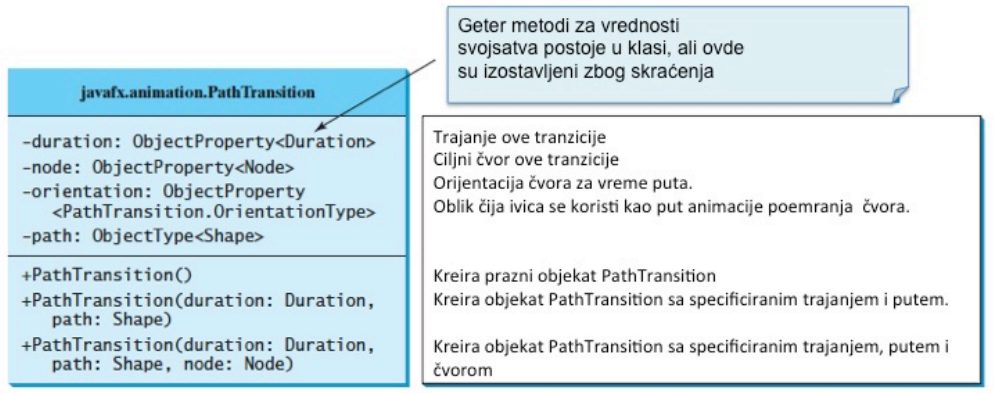
KLASA PATHTRANSITION

Klasa PathTransition animira kretanje čvora duž nekog puta od jednog do drugog kraja u određenom vremenskom periodu

Klasa ***PathTransition*** animira kretanje čvora duž nekog puta od jednog do drugog kraja u određenom vremenskom periodu. Klasa ***PathTransition*** je podtip tipa ***Animation***. UML dijagram klase je dat na slici 3.

Klasa ***Duration*** definiše vremensko trajanje. To je nepromenljiva klasa. Klasa definiše konstante***: INDEFINTE, ONE, UNKNOWN,*** i ***ZERO*** da bi definisale beskonačno trajanje, trajanje od 1 milisekunde, nepoznato trajanje i trajanje 0.

Sa ***new Duration(double millis)*** se kreira primerak (objekat) klase ***Duration***, metodi za dodavanje, oduzimanje, množenje i delenje vremenskih vrednosti. Metodi koji te vrednosti pretvaraju u sate,minute, sekunde i milisekunde su ***toHours(), toMinutes(), toSeconds()***, i ***toMillis()***



Slika 11.3 .3: Klasa PathTransition definiše animaciju duž određenog puta [1]

Metod ***compareTo()*** služi za upoređivanje vremenskih trajanja animacija.

Konstante ***NONE ORTHOGONAL_TO_TANGENT*** se definišu u klasi ***PathTransition***

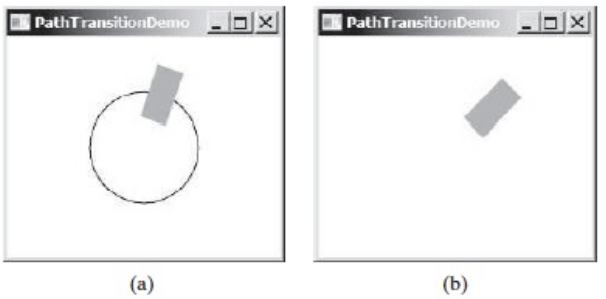
.OrientationType. Ona specificira da je položaj čvora normalan u dnosu na tangentu gemoetrijskog puta.

.

PRIMER ANIMACIJE KRETANJA PRAVOUGAONIKA

Klasa PathTransition animira kretanje pravougaonika po kružnici

Na slici 4.a prikazan je prikazan je pravougaonik i kružna putanja duž koje treba da se kreće, zadržavajući svoj normalni položaj u odnosu na tu putanju. Ovde je dat listing klase ***PathTransitionDemo*** koja realizuje ovu animaciju.



Slika 11.4 .4: Klasa PathTransition animira kretanje pravougaonika po kružnici [1]

Program kreira okvir (linija 16), pravougaonik (linija 19= i krug (linija 23). Krug i pravougaonik se postavljaju u okvir (linije 28 i 29.. Slika 4.b prikazuje slučaj ako krug nije ubačen u okvir. Program kreira put kretanja (linija 32), definiše njegovo trajanje od 4 sekunde u jednom ciklusu. Animacije (linija 23), postavlja krug da bude put kretanja (linija 34), postavlja pravougaonik kao čvor (linija 35, i postavlja ga da bude normalan na kružnicu (linija 36)

Broj ciklusa je podešen na beskonačnu vrednost (linija 38) tako da se animacije nikad ne zaustavlja. Svojstvo autorevers je postavljeno na vrednost true (istinito) tako da se pravougaonik u svakom ciklusu (jedan krug kretanja) kreće u suprotnom smeru. Pozivom metoda play() počinje animacija (linija 40). Ako se metod pause() zameni sa metodom stop() u liniji 42, animacija će se posle zaustavljanja početi sledeći put iz položaja u kome je zaustavljena.

.

```
import javafx.animation.PathTransition;
 import javafx.animation.Timeline;
 import javafx.application.Application;
 import javafx.scene.Scene;
 import javafx.scene.layout.Pane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Rectangle;
 import javafx.scene.shape.Circle;
 import javafx.stage.Stage;
 import javafx.util.Duration;

 public class PathTransitionDemo extends Application {
   @Override // Redefinisanje metoda start klase Application
   public void start(Stage primaryStage) {
  // Kreiranje okvira
  Pane pane = new Pane();

  // CKreiranje pravougaonika
  Rectangle rectangle = new Rectangle (0, 0, 25, 50);
  rectangle.setFill(Color.ORANGE);

  // Kreiranje kruga
```

```
  Circle circle = new Circle(125, 100, 50);
  circle.setFill(Color.WHITE);
  circle.setStroke(Color.BLACK);

  // Dodacanje kruga i pravougaonika u okno
  pane.getChildren().add(circle);
  pane.getChildren().add(rectangle);

  // Kreirane tranzicije puta
  PathTransition pt = new PathTransition();
  pt.setDuration(Duration.millis(4000));
  pt.setPath(circle);
  pt.setNode(rectangle);
  pt.setOrientation(
  PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
  pt.setCycleCount(Timeline.INDEFINITE);
  pt.setAutoReverse(true);
  pt.play(); // Početak animacije

  circle.setOnMousePressed(e -> pt.pause());
  circle.setOnMouseReleased(e -> pt.play());

  // Kreiranje scene i njeno postavljanje na pozornicu
  Scene scene = new Scene(pane, 250, 200);
  primaryStage.setTitle("PathTransitionDemo"); // Unos nayiva pozornice e
  primaryStage.setScene(scene); // Postavljanje scene na pozornicu
  primaryStage.show(); // Prikazivanje pozornice
   }
}
```

PRIMER DIZANJA ZASTAVE

Vrši se animacija dizanja zastave duž prave linije u trajanju od 10 sekundi, a sa ponavljanjem pet puta.

Ranije (slika 1) prikazana je slika zastava za koju će se vršiti animacija njenog podizanje. Ovde se daje listing klase ***FlagRisingAnimation*** koja realizuje animaciju dizanja zastave.

Program kreira okno (linija 14), kreira prikaz slike koristeći datoteku sa slikom zastave (linija 17) i postavlja prikaz slike u okno (linija 18). Put kretanja (tranzicije) se definiše da traje 10 sekundi pri čemu se za put kretanja zastave koristi prava linija, a prikaz slike se koristi kao čvor (linije 21 i 22).

Prikaz slike (zastava) se kreće duž linije. Kako linija nija postavljena u scenu, ne možete je videti u prozoru. Broj ciklusa dizanja je postavljen na 5 (linija 23) tako da se animacija ponavlja pet puta.

```
import javafx.animation.PathTransition;
 import javafx.application.Application;
```

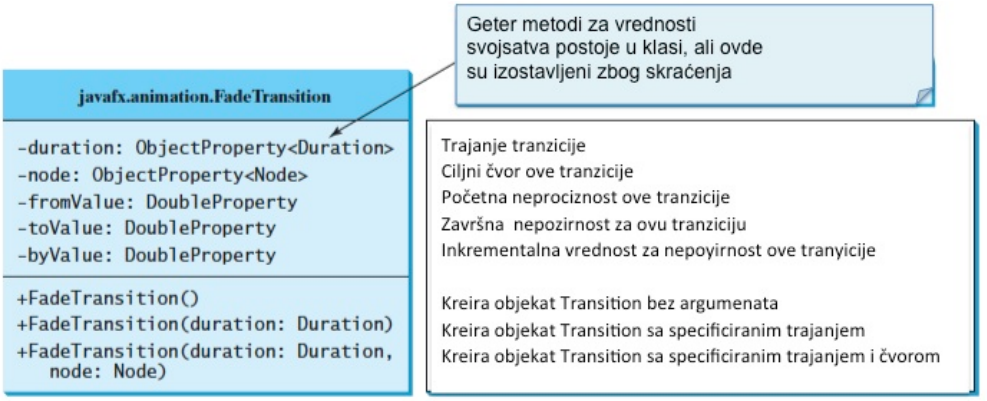
```
import javafx.scene.Scene;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
import javafx.util.Duration;

public class FlagRisingAnimation extends Application {
  @Override // Redefinisanje metoda start klase Animation
  public void start(Stage primaryStage) {
 // Kreiranje okna
 Pane pane = new Pane();

 // Dodavanje prikaza slike i njegovo dodavanje u okno
 ImageView imageView = new ImageView("image/us.gif");
 pane.getChildren().add(imageView);

 // Kreiranje puta tranzicije
 PathTransition pt = new PathTransition(Duration.millis(10000),
 new Line(100, 200, 100, 0), imageView);
 pt.setCycleCount(5);
 pt.play(); // Početak animacije

 // Kreiranje scene i njeno plasiranje na pozornicu
 Scene scene = new Scene(pane, 250, 200);
 primaryStage.setTitle("FlagRisingAnimation"); // Unos naslova pozornice
 primaryStage.setScene(scene); // Postavljanje scene na pozornicu
 primaryStage.show(); // Prikaz pozornice
}
```

KLASA FADETRANSITION

Klasa FadeTransition vrši animaciju promene neprozirnosti čvora u datom vremenskom periodu.

Klasa***FadeTransition*** vrši animaciju promene neprozirnosti čvora u datom vremenskom periodu.

Klasa ***FadeTransition*** je podklasa (podtip) klase (tipa) Animation. Na slici 5 je prikazan UML dijagram klase.



Slika 11.5 .5: Klasa FadeTransition animira promenu neprozirnost čvora [1]

PRIMER ANIMACIJE PROMENE NEPROZIRNOSTI

Primer animacije menjanja nivoa neprozirnosti elipse, od početne do krajnje, u datom vremenskom trajanju.

Na slici 6 je prikazana elipsa sa dva nivoa neprozirnosti. U ovom primeru je potrebno izvršiti animaciju promene stepena neprozirnosti elipse, tje promenu od početnog do krajnjeg nivoa neprozirnostu. Listing klase FadeTransition koja realizuje ovu animaciju je ovde prikazan. .

Slika 11.6 .6: Klasa FadeTransition vrši animaciju promene nivoa neprozirnosti elipse od početnog (a) do krajnjeg (b) nivoa. [1]

Program kreira okno (linija 15) a elipsu (linija 16) koju postavlja u okno (linija 25). Svojstva elipse (centerX, centerY, radiusX, i radiusY) su povezane sa veličinom okna (linije 19-24). Promena neprozirnosti elipse se vrši u trajanju od 3 sekunde (linija 29), nivoa 1,0 do nivoa 0,1 (linije 30 i 31). Broj ciklusa animacije je postavljen na beskonačan (linija 32). Kada se klikne mišem, animacija se privremeno zaustavlja (linija 37), a kada se dugme miša pusti, animacija se nastavlja od mesta zaustavljanja. (linija 38).

```
import javafx.animation.FadeTransition;
 import javafx.animation.Timeline;
 import javafx.application.Application;
 import javafx.scene.Scene;
 import javafx.scene.layout.Pane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Ellipse;
 import javafx.stage.Stage;
 import javafx.util.Duration;
```

```
public class FadeTransitionDemo extends Application {
  @Override // Redefinisanje metod start klase Animacije
  public void start(Stage primaryStage) {
 // Postavlja elipsu u okno
 Pane pane = new Pane();
 Ellipse ellipse = new Ellipse(10, 10, 100, 50);
 ellipse.setFill(Color.RED);
 ellipse.setStroke(Color.BLACK);
 ellipse.centerXProperty().bind(pane.widthProperty().divide(2));
 ellipse.centerYProperty().bind(pane.heightProperty().divide(2));
 ellipse.radiusXProperty().bind(
 pane.widthProperty().multiply(0.4));
 ellipse.radiusYProperty().bind(
 pane.heightProperty().multiply(0.4));
 pane.getChildren().add(ellipse);

 // Primena tranzicije izbleđivanja
 FadeTransition ft =
     new FadeTransition(Duration.millis(3000), ellipse);
 ft.setFromValue(1.0);
 ft.setToValue(0.1);
 ft.setCycleCount(Timeline.INDEFINITE);
 ft.setAutoReverse(true);
 ft.play(); // Start animation

 // Kontrola animacije
 ellipse.setOnMousePressed(e -> ft.pause());
 ellipse.setOnMouseReleased(e -> ft.play());

 // Kreiranje scene i njeno postavljanje na pozornicu
 Scene scene = new Scene(pane, 200, 150);
 primaryStage.setTitle("FadeTransitionDemo"); // Unos naziva pozornice
 primaryStage.setScene(scene); // postavljanje scene na pozprnicu
 primaryStage.show(); // Pikaz pozornice
  }
}
```

KLASA TIMELINE

Klasa Timeline se koristi u cilju programiranja bilo koje animacije upotrebom jednog ili više objekata klase KeyFrame.

Klase ***PathTransition*** i ***FadeTransition*** su specijalizovane klase za animaciju kretanja čvora i njegove neproizirnosti. Da li postoji klasa koja je opštijeg tipa, tj. koja može da podrži bilo kakvu animaciju? Da, postoji. To je klasa ***Timeline.***

Klasa ***Timeline*** se koristi u cilju programiranja bilo koje animacije upotrebom jednog ili više objekata klase ***KeyFrame.***

Svaki objekat klase ***KeyFrame*** se sekvencijalno izvršava u specificiranom vremenskom intervalu . Klasa ***Timeline*** je podklasa klase ***Animation***,. A kreira se konstruktorom:

new Timeline(KeyFrame keyframes)

Objekat klase KeyFrame se kreira iskazom:

```
new KeyFrame(Duration duration, EventHandler<ActionEvent> onFinished)
```

.

.

Obrađivač događaja ***onFinished*** se poziva po isteku trajanja jednog ***KeyFrame*** objekta. Ovde prikazan listing klase **TimelineDemo** daje primer prikaza teksta na slici 7. Tekst se javlja i gubi naizmenično

Slika 11.7 .7: Animacija naizmeničnog prikaza okna sa tekstom i bez teksta [1]

PRIMER PRIMENE KLASE TIMELINE

Program treba da omogući naizmenično javljanje i uklanjanje datog teksta u oknu

Program treba da omogući naizmenično javljanje i uklanjanje teksta u oknu, prikazanog na slici 7.

Program najpre kreira okno (linija 17) i tekst (linija 18) i postavlja tekst u okno (linija 20). Obrađivač je kreiran da bi promenio tekst u prazan prostor (linije 24-26), i da bi prikazao posle toga dati tekst (linije 27-29). Kreira se jedan objekat KeyFrame da bi se kreirao jedan događaj akcije na svaka pola sekunde (linija 34). Animacija sa Timeline objektom sadrži ključni okvir (objekat KeyFrame) (linije 33 i 34). Animacija je postavljena da beskonačno traje (linija 35). Postavljen je događaj kada se mišem pokaže na tekst (linije 39-46). Klikom miša preko teksta nastavlja izvršenje animacije , ako je bila u privremenom prekidu (linije 40-42) a ponovnim klikom miša preko teksta, dovodi do ponovnog privremenog zaustavljanja izvršenja animacije (linije 43-45).

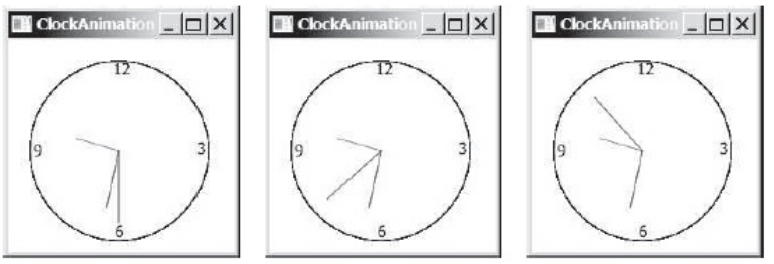
```
import javafx.animation.Animation;
 import javafx.application.Application;
 import javafx.stage.Stage;
 import javafx.animation.KeyFrame;
 import javafx.animation.Timeline;
 import javafx.event.ActionEvent;
 import javafx.event.EventHandler;
 import javafx.scene.Scene;
 import javafx.scene.layout.StackPane;
 import javafx.scene.paint.Color;
```

```
import javafx.scene.text.Text;
import javafx.util.Duration;

public class TimelineDemo extends Application {
  @Override // Redefinisanje metoda start() klase Application
  public void start(Stage primaryStage) {
 StackPane pane = new StackPane();
 Text text = new Text(20, 50, "Programming is fun");
 text.setFill(Color.RED);
 pane.getChildren().add(text); // Place text into the stack pane

 // Kreiranje obrađivača za promenu teksta
 EventHandler<ActionEvent> eventHandler = e -> {
   if (text.getText().length() != 0) {
     text.setText("");
   }
   else {
     text.setText("Programming is fun");
   }
 };

 // Kreiranje animacije za menjanje tekst
 Timeline animation = new Timeline(
 new KeyFrame(Duration.millis(500), eventHandler));
 animation.setCycleCount(Timeline.INDEFINITE);
 animation.play(); // Start animation

 // Prekid i nastavakanimacije
 text.setOnMouseClicked(e -> {
   if (animation.getStatus() == Animation.Status.PAUSED) {
     animation.play();
   }
   else {
     animation.pause();
   }
 });

 // Create a scene and place it in the stage
 Scene scene = new Scene(pane, 250, 250);
 primaryStage.setTitle("TimelineDemo"); // Set the stage title
 primaryStage.setScene(scene); // Place the scene in the stage
 primaryStage.show(); // Display the stage
  }
}
```

SIMULACIJA RADA ANALOGNOG SATA – FAZA 1

Pre animacije, mora se nacrtati analogni sat primenom klase ClockPane.

Na slici 8 prikazan je analogni sat u radu. Da bi se izvršila njegova animacija, prvo je potrebno nacrtati analogni sadt. To radi klasa ClockPane.



Slika 11.8 .8: . Analogni sat u radu prikazan u tri vremenska trenutka [1]

```
import java.util.Calendar;
 import java.util.GregorianCalendar;
 import javafx.scene.layout.Pane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Circle;
 import javafx.scene.shape.Line;
 import javafx.scene.text.Text;

 public class ClockPane extends Pane {
   private int hour;
   private int minute;
   private int second;

   // Širina i visina okna sata
   private double w = 250, h = 250;

   /** Konstruisanje početnog sata sata sa trenutnim vremenom */
   public ClockPane() {
  setCurrentTime();
   }

   /** Konstruisanje sata sa specificiranim satim, minutima i sekundama  */
   public ClockPane(int hour, int minute, int second) {
  this.hour = hour;
  this.minute = minute;
  this.second = second;
  paintClock();
   }

   /** Vraća hour */
   public int getHour() {
  return hour;
   }

   /** Unisi novu vrednost za  hour */
   public void setHour(int hour) {
  this.hour = hour;
  paintClock();
   }
```

```
/** vraća minute */
  public int getMinute() {
 return minute;
  }

  /** Unosi novu vrednost za minute */
  public void setMinute(int minute) {
 this.minute = minute;
 paintClock();
  }

  /** Vraća second */
  public int getSecond() {
 return second;
  }

  /** Unosi novu vrednost za second */
  public void setSecond(int second) {
 this.second = second;
 paintClock();
  }

  /** Vraća širinu okna za sat  */
  public double getW() {
 return w;
  }

  /** Unosi širinu okna za sat */
  public void setW(double w) {
 this.w = w;
 paintClock();
  }

  /** Vraća visinu okna za sat  */
 public double getH() {
 return h;
  }

  /** Podešava visinu okna sata  */
  public void setH(double h) {
 this.h = h;
 paintClock();
  }

  /* Unosi sadašnje vreme u sat  */
  public void setCurrentTime() {
 // Konstruiše kalendar ya sadašnji datum i vreme
 Calendar calendar = new GregorianCalendar();

 // Unos trenutnog vremena sata, minuta i sekundi
 this.hour = calendar.get(Calendar.HOUR_OF_DAY);
 this.minute = calendar.get(Calendar.MINUTE);
```

```
 this.second = calendar.get(Calendar.SECOND);

 paintClock(); // nacrtaj sat
  }

  /** Crtanje sata  */
  protected void paintClock() {
    // Inicijalizacija parametra sata
    double clockRadius = Math.min(w, h) * 0.8 * 0.5;
    double centerX = w / 2;
    double centerY = h / 2;

    // Crtanje kruga
    Circle circle = new Circle(centerX, centerY, clockRadius);
    circle.setFill(Color.WHITE);
    circle.setStroke(Color.BLACK);
    Text t1 = new Text(centerX - 5, centerY - clockRadius + 12, "12");
    Text t2 = new Text(centerX - clockRadius + 3, centerY + 5, "9");
    Text t3 = new Text(centerX + clockRadius - 10, centerY + 3, "3");
    Text t4 = new Text(centerX - 3, centerY + clockRadius - 3, "6");

    // Crtanje skazaljke za prikaz sekundi
    double sLength = clockRadius * 0.8;
    double secondX = centerX + sLength *
    Math.sin(second * (2 * Math.PI / 60));
    double secondY = centerY - sLength *
    Math.cos(second * (2 * Math.PI / 60));
    Line sLine = new Line(centerX, centerY, secondX, secondY);
    sLine.setStroke(Color.RED);

    // Crtanje velike skazalje za prikaz minuta
    double mLength = clockRadius * 0.65;
    double xMinute = centerX + mLength *
    Math.sin(minute * (2 * Math.PI / 60));
    double minuteY = centerY - mLength *
    Math.cos(minute * (2 * Math.PI / 60));
    Line mLine = new Line(centerX, centerY, xMinute, minuteY);
    mLine.setStroke(Color.BLUE);

    // Crtanje sklayalje ya prikaz sati
    double hLength = clockRadius * 0.5;
    double hourX = centerX + hLength *
        Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
    double hourY = centerY - hLength *
        Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
    Line hLine = new Line(centerX, centerY, hourX, hourY);
    hLine.setStroke(Color.GREEN);

    getChildren().clear();
    getChildren().addAll(circle, t1, t2, t3, t4, sLine, mLine, hLine);
  }
}
```

SIMULACIJA RADA ANALOGNOG SATA – FAZA 2

Klasa Timeline omogućava crtanje skazaljki sata na svaki sekund.

Da bi se vršila animacija rada analognog sata, tj. da njegove skazaljke postepeno menjaju svoj položaj, u skladu sa trenutnim vremenom, potrebno je da se crtanje sata obnavlja svakog sekunde. Za tu svrhu se koristi klasa ***Timeline*** da bi se kontrolisalo crtanje sata svakog sekunda.

Listing klase **ClockAnimation** pokazuje program koji ovo omogućava. Analognog sata (linija 13). Objekat ***ClockPane*** se stavlja u scenu u liniji 27. Kreira se obrađivač događaja radi podešavanje trenutnog vremena na satu (linija 16-18). On se poziva svakog sekunda radi animacije (linija 21-24). Na taj način, vrši se animacija položaja skazaljki sata sa prikazom promene svake sekunde.

..

```
import javafx.application.Application;
 import javafx.stage.Stage;
 import javafx.animation.KeyFrame;
 import javafx.animation.Timeline;
 import javafx.event.ActionEvent;
 import javafx.event.EventHandler;
 import javafx.scene.Scene;
 import javafx.util.Duration;

 public class ClockAnimation extends Application {
   @Override // Redefinisanje metoda start klase Animation
   public void start(Stage primaryStage) {
  ClockPane clock = new ClockPane(); // Kreiranje sata

  // Kreiranje obrađivača za animaciju
  EventHandler<ActionEvent> eventHandler = e -> {
      clock.setCurrentTime(); // Postavljanje novog vremena u satu
  };

  // Kreiranje animacije sata koji radi
  Timeline animation = new Timeline(
      new KeyFrame(Duration.millis(1000), eventHandler));
  animation.setCycleCount(Timeline.INDEFINITE);
  animation.play(); // Sartuj animaciju

  // Kreiranje scene i njeno postavljanje na pozornicu
  Scene scene = new Scene(clock, 250, 50);
  primaryStage.setTitle("ClockAnimation"); // Unos naziva pozornice
  primaryStage.setScene(scene); // Postavljanje scene napozornicu
  primaryStage.show(); // Prikaz  pozornice
   }
}
```

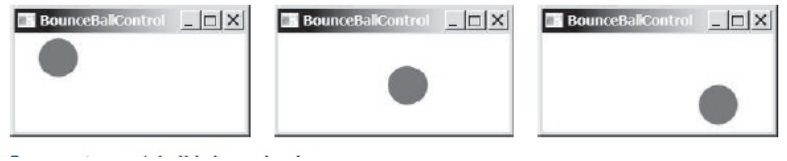
Poglavlje 12

Studija slučaja: Poskakivanje lopte

POSTUPAK PRIPREME PROGRAMA ZA ANIMACIJU

Ovde se prikazuje animacija kretanja lopte unutar granica okna, sa odbijanjem lopte na granicama.

Program koristi klasu Timeline za animaciju kretanja (poskakivanja) lopte prikazane na slici 1.



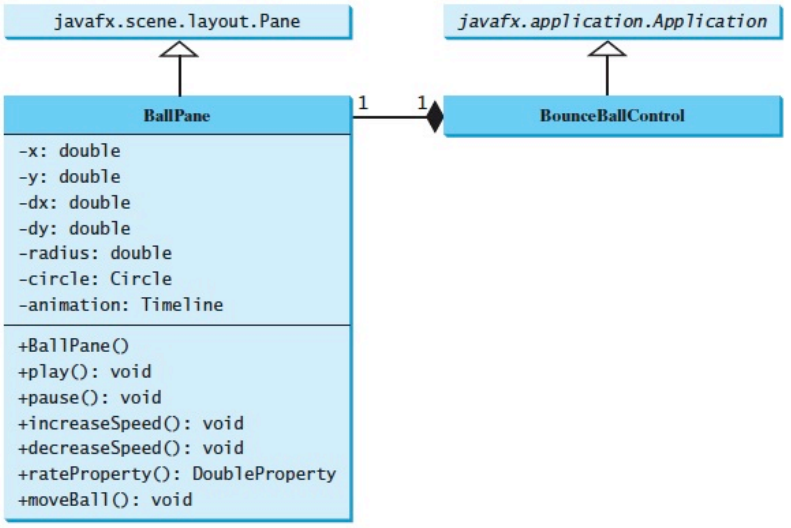
Slika 12.1 .1: Lopta poskakuje unutar okna [1]

Pri pisanju programa, za ovaj slučaj animacije, koriste se sledeće i postupak:

1. Definiše se podklasa klase Pane, sa nazivom **BallPane** radi prikaza poskivajuće lopte (videti listing klase **BallPane**).
2. Definiše se podklasa klase ***Annimation*** pod nazivom **BounceBallControl** koja kontroliše poskakivanje lopte sa akcijama miša, kao što je prikazano listing klase **BounceBallControl**. Animacija se privremeno zaustavlja pritiskom dugmeta miša, a novim pritiskom, nastavlja se animacija. Pritiskom tastera UP i DOWN povećava se ili smanjuje brzina animacije.

Na slici 2 prikazan je UML dijagram ovih klasa i sa vezama između ovih klasa.

.



Slika 12.2 .2: Klasa BounceBallControl sadrži klasu BallPane [1]

KLASA BALLPANE

Klasa BallPane vrši animaciju kretanja lopte u oknu koja se kreće i odbija od granica okna.

Klasa **BallPane** proširuje klasu ***Pane*** radi prikazivanja krećuće lopte (linija 9). Primerak klase ***Timeline*** se kreira radi kontrole animacije (linija 21 i 22). Ona sadrži ***KeyFrame*** objekat koji poziva metod moveBall() date brzine. Metod **moveBall()** pokreće loptu radi animacije. Centar lopte je u koordinatam (x,y) koji se pomera u novu lokaciju sa koordinatama (x+dx, y+dy) (linije 58-61). Kada lopta dođe do leve ili desne granica, dolazi do promene znaka priraštaja dx (linije 50-52). Ovo dovodi do kretanja lopte u suprotnom smeru duž horizontalne ose (x).

Kada lopta udari u gornju ili donju granicu, menja se znak ispred priraštaja dy (linija 53-55). Ovim se menja smer kretanja lopte duš vertikalne ose. Metodi pause() i play() (linije 27-33) zaustavljaju i nastavljaju animaciju. Metodi **increaseSpeed()** i **decreaseSpeed()** povećavaju i smanjuju brzinu animacije, Metod **rateProperty()** (linije 44-46) vraša povezano svojstvo brzine. Ono je korisno jer povezuje brzinu budućih animacija

```
import javafx.animation.KeyFrame;
 import javafx.animation.Timeline;
 import javafx.beans.property.DoubleProperty;
 import javafx.scene.layout.Pane;
 import javafx.scene.paint.Color;
 import javafx.scene.shape.Circle;
 import javafx.util.Duration;

 public class BallPane extends Pane {
   public final double radius = 20;
   private double x = radius, y = radius;
   private double dx = 1, dy = 1;
   private Circle circle = new Circle(x, y, radius);
```

```
  private Timeline animation;

  public BallPane() {
 circle.setFill(Color.GREEN); // Unos boje lopte
 getChildren().add(circle); // Postavljanje lopte u okno

 // Kreiranje animacije sa kretanjem lopte
 animation = new Timeline(
     new KeyFrame(Duration.millis(50), e -> moveBall()));
 animation.setCycleCount(Timeline.INDEFINITE);
 animation.play(); // Start animation
  }

  public void play() {
 animation.play();
  }

  public void pause() {
 animation.pause();
  }

  public void increaseSpeed() {
 animation.setRate(animation.getRate() + 0.1);
  }

  public void decreaseSpeed() {
 animation.setRate(
   animation.getRate() > 0 ? animation.getRate() - 0.1 : 0);
  }

  public DoubleProperty rateProperty() {
 return animation.rateProperty();
  }

  protected void moveBall() {
 // Provera granica
 if (x < radius || x > getWidth() - radius) {
   dx *= -1; // Change ball move direction
 }
 if (y < radius || y > getHeight() - radius) {
   dy *= -1; // Change ball move direction
 }

 // Adjust ball position
 x += dx;
 y += dy;
 circle.setCenterX(x);
 circle.setCenterY(y);
  }
}
```

KLASA BANCEBALLCONTROL

Klasa BounceBallControl omogućava prekid i nastavak animacije, ili ubrzanje i usporenje animacije.

Klasa **BounceBallContro**l proširuje JavaFX klasu ***Application*** da bi prikazio okvir sa loptom. Pritiskom i odpoštanjem dugmeta na mišu dovodi do događaja koje obrađuju njihovi obrađivači. Piriskom dugmenta aplikacija staje, a otpuštanjem dugmenta – aplikacija nastavlja animaciju. (linija 12 i 13).

Kada se pritisni taster UP na tastaturi (strelica gore) metod **increaseSpeed()** povećava brzinu kretanja lopte (linija 18).

Kada se pritisne taster DOWN (strelica nadole), aktivira se metod **decreaseSpeed()** koji smanjuje brzinu kretanja lopte (linija 21).

Pozivom metoda **ballPane.regestFocus()** u liniji 32, postavlja se ulazni fokus na metod **ballPane().**

```
import javafx.application.Application;
 import javafx.stage.Stage;
 import javafx.scene.Scene;
 import javafx.scene.input.KeyCode;

 public class BounceBallControl extends Application {
   @Override // Redefinisanje metoda start() u klasi Application
   public void start(Stage primaryStage) {
   BallPane ballPane = new BallPane(); // Kreiranje okna sa loptom

  // Pauziranje i nastavak animacija
  ballPane.setOnMousePressed(e -> ballPane.pause());
  ballPane.setOnMouseReleased(e -> ballPane.play());

  // Ubrzanje i usporenje animacije
  ballPane.setOnKeyPressed(e -> {
    if (e.getCode() == KeyCode.UP) {
      ballPane.increaseSpeed();
    }
    else if (e.getCode() == KeyCode.DOWN) {
      ballPane.decreaseSpeed();
    }
  });

  // Kreiranje scene i njeno postavljanje na pozornicu
  Scene scene = new Scene(ballPane, 250, 150);
  primaryStage.setTitle("BounceBallControl"); // Onos naziva pozornice
  primaryStage.setScene(scene); // Postavljanje scene na pozornice
  primaryStage.show(); // Priaz pozornice

  // MOra se yahtevati fokuc posle prikaza osnovne pozornice
  ballPane.requestFocus();
```

```
    }
}
```

ZADACI ZA SAMOSTALNI RAD

Proverite razumevanje rada ove aplikacije.

1. Kako program čini da se lopta kreće?
2. Kako kod u listingu klase BallPane menja smer kretanja lopte?
3. Šta program rad kada se pritiska dugme miša? Šta program radi kada se otpusti dugme u oknu lopte?
4. Ako se linija 32 u listingu klase BounceBallControl izbaci iz programa, šta će desiti kada pritisnite taster UP ili DOWN?
5. Ako se iz koda listinga BallPan izostaviti iz linije 23, šta će se desiti?

Poglavlje 13

Vežba – Pokazni primeri

PRIMER 1

Cilj ovog zadatka je provežbavanje animacija kroz JavuFX

Napraviti animaciju u JaviFX tako da iscrtani pun krug nestaje i opet se vraća (fade out, fade in). Za ovo je potrebno koristiti FadeTranslation i Cirlce objekte.

Potrebno vreme: 10 minuta

Slika 13.1 .1: Animacija sa krugom

PRIMER 1 - OBJAŠNJENJE I UPUTSTVO ZA REŠAVANJE

U nastavku je dato pojašnjenje rešenja zadatka 1

Objašnjenje i uputstva:

1. prilikom kreiranja projekta odabrati

File > New Project > JavaFX > JavaFX Application

2. pokretački main metod treba da sadrži sledeću naredbu
launch(args);

3. Override - ovati start metod

4. Postaviti panel na Frame I dati mu Layout

5. na panel dodati krug što se postiže naredbom
root.getChildren().add(circle);

6. postaviti na Scenu root panel naredbom
Scene scene = new Scene(root, 300, 250);
7. Kada je scena spremna treba je postaviti na Stage I učiniti je vidljivom
primaryStage.setScene(scene);
primaryStage.show();

8. Moguće je aplikaciji dati i Title naredbom
primaryStage.setTitle("Fade transition demo");

9. Postavljenom krugu treba dodati animaciju I aktivirati je
FadeTransition ft = new FadeTransition(Duration.millis(3000), circle);
ft.setFromValue(1.0); ft.setToValue(0.1);
ft.setCycleCount(Timeline.INDEFINITE);
ft.setAutoReverse(true);
ft.play();

PRIMER 1 - REŠENJE

Programiski kod koji predstavlja rešenje zadatka 1

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import javafx.animation.FadeTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 *
 * @author Aleksandra
 */
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        final Circle circle = new Circle(50);
```

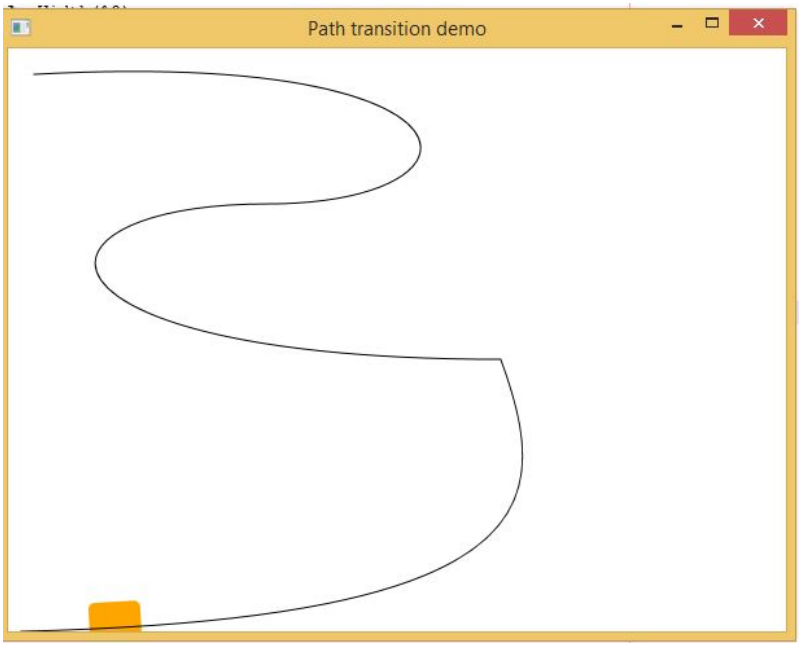
```
        circle.setFill(Color.BLUE);
        FadeTransition ft = new FadeTransition(Duration.millis(3000), circle);
        ft.setFromValue(1.0);
        ft.setToValue(0.1);
        ft.setCycleCount(Timeline.INDEFINITE);
        ft.setAutoReverse(true);
        ft.play();
        StackPane root = new StackPane();
        root.getChildren().add(circle);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Fade transition demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
        launch(args);
    }

}
```

PRIMER 2

Cilj ovog zadatka je provežbavanje Path animacija kroz JavuFX

Napraviti animaciju u JaviFX tako da kvadrat ide iscrtanom linijom kao na slici:

Potrebno vreme: 10 minuta



Slika 13.2 .2: Kretanje kvadrata po nacrtanoj liniji

PRIMER 2- OBJAŠNJENJE I UPUTSTVO ZA REŠAVANJE

U nastavku je dato pojašnjenje rešenja zadatka 2

Objašnjenje i uputstva:

1. prilikom kreiranja projekta odabrati

File > New Project > JavaFX > JavaFX Application

2. pokretački main metod treba da sadrži sledeću naredbu

launch(args);

3. Override - ovati start metod

4. Napraviti Scenu Scene scene = new Scene(root, 600, 450);

5. Dodati scenu na Stage
primaryStage.setScene(scene);
primaryStage.show();

6. Napraviti pravougaonik
final Rectangle rectPath = new Rectangle(0, 0, 40, 40);
rectPath.setArcHeight(10);
rectPath.setArcWidth(10);
rectPath.setFill(Color.ORANGE);

7. Napraviti putanju
Path path = new Path();
path.getElements().add(new MoveTo(20, 20));
path.getElements().add(new CubicCurveTo(380, 0, 380, 120, 200, 120));
path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240));
path.getElements().add(new CubicCurveTo(420, 350, 420, 440, 10, 450));

8. Dodati na scenu putanju I pravouganik
root.getChildren().add(rectPath);
root.getChildren().add(path);

9. Dodati animaciju

PRIMER 2 - REŠENJE

Programiski kod koji predstavlja rešenje zadatka 2

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
```

```
 * and open the template in the editor.
 */

import javafx.animation.PathTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.CubicCurveTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 *
 * @author Aleksandra
 */
public class FallowPath extends Application {

     @Override
    public void start(Stage primaryStage) {
        final Rectangle rectPath = new Rectangle(0, 0, 40, 40);
        rectPath.setArcHeight(10);
        rectPath.setArcWidth(10);
        rectPath.setFill(Color.ORANGE);

        Path path = new Path();
        path.getElements().add(new MoveTo(20, 20));
        path.getElements().add(new CubicCurveTo(380, 0, 380, 120, 200, 120));
        path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240));
        path.getElements().add(new CubicCurveTo(420, 350, 420, 440, 10, 450));
        PathTransition pathTransition = new PathTransition();
        pathTransition.setDuration(Duration.millis(4000));
        pathTransition.setPath(path);
        pathTransition.setNode(rectPath);

pathTransition.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
        pathTransition.setCycleCount(5);
        pathTransition.setAutoReverse(true);
        pathTransition.play();

        Group root = new Group();
        root.getChildren().add(rectPath);
        root.getChildren().add(path);

        Scene scene = new Scene(root, 600, 450);
```

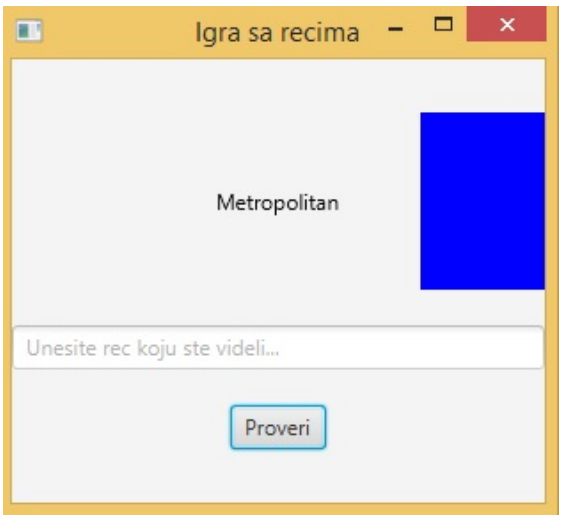
```
        primaryStage.setTitle("Path transition demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

PRIMER 3

Cilj ovog zadatka je pravljenje aplikacije koja koristi translacije i događaje

Napraviti igru za pogađanje reči. Korisnik treba da ima prekrivenu reč iza kvadrata a klikom na nju reč treba da se prikaže. Korisnik potom može da upiše reč a program treba da mu ispiše da li je pogodio reč ili ne. Nakon pogotka program treba da prikaže sledeću reč.

Potrebno vreme: 10 minuta



Slika 13.3 .3: Kvadrat koji prekriva skrivenu reč

PRIMER 3 - OBJAŠNJENJE I UPUTSTVO ZA REŠAVANJE

U nastavku je dato pojašnjenje rešenja zadatka 3

Objašnjenje i uputstva:

1. Napraviti JavaFX Application

2. dodati niz reči koje se mogu pogadajti

private String[] words = new String[]{"CS202", "Programiranje", "Metropolitan", "Semestar", "Automobil", "Fakultet", "JavaFX", "Kuciste", "Tastatura", "Avion"};
3. kreiriati objekat klase Random za nasumični odabir reči iz niza
private Random rand = new Random();
4. Objekat rectange za sakrianje reči
private Rectangle rect;

5. U ovom slučaju root će biti centrirani layout sa vertikalnim prikazom elemenata

VBox root = new VBox(20);

root.setAlignment(Pos.CENTER);

6. Postavljamo jedan StackPane na koji ćemo dodati Text i preko njega Rectangle
StackPane sp = new StackPane();
final Text word = new Text(words[rand.nextInt(words.length)]);
rect = new Rectangle(160, 100, Color.BLUE);
sp.getChildren().add(word);
sp.getChildren().add(rect);

7. sp postaviti na root, zatim dodati I Text Field I button
final TextField tf = new TextField();
tf.setFocusTraversable(false);
tf.setPromptText("Unesite rec koju ste videli...");
root.getChildren().add(sp);
root.getChildren().add(tf);
root.getChildren().add(checkBtn);

8. Postaviti handler na klik miša
scene.setOnMouseClicked(mouseHandler);
scene.setOnMousePressed(mouseHandler);
scene.setOnMouseReleased(mouseHandler);

PRIMER 3 - REŠENJE

Programiski kod koji predstavlja rešenje zadatka 3

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import java.util.Random;
import javafx.application.Application;
import javafx.event.ActionEvent;
```

```
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javax.swing.JOptionPane;

/**
 *
 * @author Aleksandra
 */
public class CatchWord extends Application {
     private String[] words = new String[]{"CS202", "Programiranje",
"Metropolitan", "Semestar", "Automobil", "Fakultet", "JavaFX", "Kuciste",
"Tastatura", "Avion"};
    private Random rand = new Random();
    private Rectangle rect;

    @Override
    public void start(Stage primaryStage) {

        Button checkBtn = new Button("Proveri");

        VBox root = new VBox(20);
        root.setAlignment(Pos.CENTER);
        StackPane sp = new StackPane();

        Scene scene = new Scene(root, 300, 250);
        // dodavanje handlera za aktivnosti na misu
        scene.setOnMouseClicked(mouseHandler);
        scene.setOnMousePressed(mouseHandler);
        scene.setOnMouseReleased(mouseHandler);

        final Text word = new Text(words[rand.nextInt(words.length)]);
        rect = new Rectangle(160, 100, Color.BLUE);

        sp.getChildren().add(word);
        sp.getChildren().add(rect);

        final TextField tf = new TextField();
        tf.setFocusTraversable(false);
        tf.setPromptText("Unesite rec koju ste videli...");
        root.getChildren().add(sp);
        root.getChildren().add(tf);
        root.getChildren().add(checkBtn);
```

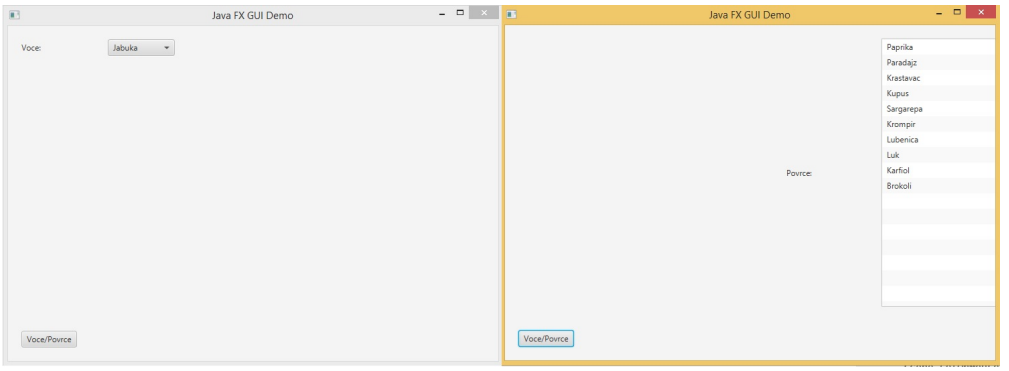
```
        primaryStage.setTitle("Igra sa recima");
        primaryStage.setScene(scene);
        primaryStage.show();

        // postavljanje eventHandler-a na prilikom klika na dugme checkBtn
        checkBtn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent t) {
                if (tf.getText().equals(word.getText())) {
                    JOptionPane.showMessageDialog(null, "Tacno! :)");
                } else {
                    JOptionPane.showMessageDialog(null, "Netacno! :(");
                }
                word.setText(words[rand.nextInt(words.length)]);
            }
        });
    }

    EventHandler<MouseEvent> mouseHandler = new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent t) {
            if (t.getEventType() == MouseEvent.MOUSE_PRESSED) {
                rect.getTransforms().add(new Translate(rect.getWidth(), 0));
            } else if (t.getEventType() == MouseEvent.MOUSE_RELEASED) {
                rect.getTransforms().add(new Translate(-rect.getWidth(), 0));
            }
        }
    };
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

PRIMER 4

Cilj ovog zadatka je pokaz primene ChoiseBox-a i ListView-a u JaviFX kao i sakrivanja istih na događaje

Napraviti ChoiceBox za voće kao i ListView za povrće. Napuniti ListView i ChoiceBox sa imenima voća i povrća. Napraviti dugme koje će menjati prikaz (jednom će prikazati ChoiceBox a drugi put ListView)

Potrebno vreme: 10 minuta



Slika 13.4 .4: Prikazivati ChoiceBox i ListView sa listama voće i povrža, naizmenično

PRIMER 4 - OBJAŠNJENJE I UPUTSTVO ZA REŠAVANJE

U nastavku je dato pojašnjenje rešenja zadatka 4

Objašnjenje i uputstva:

1. Napraviti JavaFX Application

2. Napraviti ChoiceBox
final ChoiceBox fruits = new ChoiceBox(FXCollections.observableArrayList("Jabuka", "Kruska", "Banana", "Pomorandza", "Mandarina", "Limun", "Kivi", "Grejpfrut", "Ananas", "Grozdje", "Breskva", "Visnja"));
fruits.setValue("Jabuka");
fruits.setTooltip(new Tooltip("Izaberi voce"));

3. Napraviti ListView
ListView vegetables = new ListView(FXCollections.observableArrayList("Paprika", "Paradajz", "Krastavac", "Kupus", "Sargarepa", "Krompir", "Lubenica", "Luk", "Karfiol", "Brokoli"));

4. Napraviti dugme I dodati akciju
```
Button toggle = new Button("Voce/Povrce");
//dodajemo listener na dugem
toggle.setOnAction(new EventHandler<ActionEvent>() {
@Override
public void handle(ActionEvent t) {
choicePane.setVisible(!choicePane.isVisible());
listPane.setVisible(!listPane.isVisible());
}
});
```

5. Postaviti sve na root, root na scenu, scenu na stage

6. dodati listener na checkbox koji reaguje na promenu
7. dodati listener na listview koji reaguje na promenu

PRIMER 4 - REŠENJE

Programiski kod koji predstavlja rešenje zadatka 4

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

/**
 *
 * @author Aleksandra
 */
public class JFXTest extends Application {

    @Override
    public void start(Stage primaryStage) {

        BorderPane root = new BorderPane();
        root.setPadding(new Insets(20, 50, 20, 20));

        final FlowPane choicePane = new FlowPane();
        choicePane.setHgap(100);

        Label choiceLbl = new Label("Voce:");

        final ChoiceBox fruits = new
ChoiceBox(FXCollections.observableArrayList("Jabuka", "Kruska", "Banana",
"Pomorandza", "Mandarina", "Limun", "Kivi", "Grejpfrut", "Ananas", "Grozdje",
"Breskva", "Visnja"));
        fruits.setValue("Jabuka");
        fruits.setTooltip(new Tooltip("Izaberi voce"));
```

```
        choicePane.getChildren().add(choiceLbl);
        choicePane.getChildren().add(fruits);

        root.setLeft(choicePane);

        final FlowPane listPane = new FlowPane();
        listPane.setHgap(100);

        Label listLbl = new Label("Povrce: ");
        ListView vegetables = new
ListView(FXCollections.observableArrayList("Paprika", "Paradajz", "Krastavac",
"Kupus", "Sargarepa", "Krompir", "Lubenica", "Luk", "Karfiol", "Brokoli"));

        listPane.getChildren().addAll(listLbl, vegetables);
        root.setRight(listPane);

        choicePane.setVisible(true);
        listPane.setVisible(false);

        Button toggle = new Button("Voce/Povrce");
        //dodajemo listener na dugem
        toggle.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent t) {
                choicePane.setVisible(!choicePane.isVisible());
                listPane.setVisible(!listPane.isVisible());
            }
        });
        // dodajemo listener na checkbox koji reaguje na promenu item-a checkbox-u
        fruits.getSelectionModel().selectedIndexProperty().addListener(new
ChangeListener<Number>() {

            @Override
            public void changed(ObservableValue<? extends Number> ov, Number t,
Number t1) {
                System.out.println(fruits.getItems().get(t1.intValue()));
            }

        });

        vegetables.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener() {

            @Override
            public void changed(ObservableValue ov, Object t, Object t1) {
                System.out.println(t1);
            }
        });

        root.setBottom(toggle);

        Scene scene = new Scene(root, 700, 500);
```

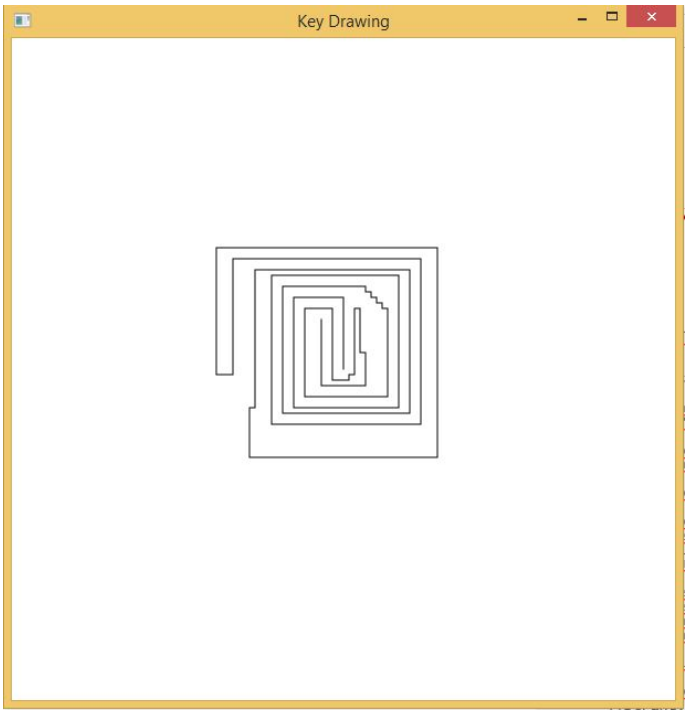
```
        primaryStage.setTitle("Java FX GUI Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

PRIMER 5

Cilj ovog zadatka je prikaz primene korišćenja događaja sa tastature.

Napraviti aplikaciju koja omogućava crtanje koristeći strelice na tastaturi. Početna kordinata treba da bude centar forme. Ukoliko korisnik drži strelicu na gore treba da se crta po X kordinati linija do god drži tu strelicu.

Potrebno vreme: 10 minuta



Slika 13.5 .5: Crtanje linija sa strelicama na tastaturu

PRIMER 5 - OBJAŠNJENJE I UPUTSTVO ZA REŠAVANJE

U nastavku je dato pojašnjenje rešenja zadatka 5

Objašnjenje i uputstva:
1. Napraviti JavaFX Application

2. Odrediti coordinate početne tačke za crtnaje
private int x = 300;
private int y = 300;
3. Napraviti dve putanje, jednu koja crta drugu koja briše
Path path;
Path erasePath;

4. Odrediti attribute putanje, debljinu linije, boju linije
path.getElements().add(new MoveTo(x, y));
path.setStrokeWidth(1);
path.setStroke(Color.BLACK);

5. Slično za putanju koja će crtati belom bojom.

6. Dodati na root obe putanje
7. postaviti listenere - metodama addKeyEvents();
addMouseEvents();

8. Kada je pritisnut taster, što proveravamo sa KeyEvent.KEY_PRESSED

handlovati KeyEvent t - u zavisnosti od tipa strelice koja je korišćena crtati putanju

provera da li je pritinuta strelica za levo:
if (t.getCode() == KeyCode.LEFT && x >= 0) {
x -= 5;
}

- slično je i za druge strelice

9. dodati akcije za miša na sceni
scene.setOnMouseClicked(mouseHandler);
scene.setOnMouseDragged(mouseHandler);
scene.setOnMousePressed(mouseHandler);
scene.setOnMouseReleased(mouseHandler);

PRIMER 5 - REŠENJE

Programiski kod koji predstavlja rešenje zadatka 5

```
/*
 * To change this license header, choose License Headers in Project Properties.
```

```
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.stage.Stage;

/**
 *
 * @author Aleksandra
 */
public class PaintingLavirint extends Application {

    private int x = 300;
    private int y = 300;
    Scene scene;
    Path path;
    Path erasePath;

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        scene = new Scene(root, 600, 600);
        scene.setFill(Color.WHITE);

        path = new Path();
        path.getElements().add(new MoveTo(x, y));
        path.setStrokeWidth(1);
        path.setStroke(Color.BLACK);

        erasePath = new Path();
        erasePath.setStrokeWidth(2);
        erasePath.setStroke(Color.WHITE);

        root.getChildren().add(path);
        root.getChildren().add(erasePath);
        addKeyEvents();
        addMouseEvents();

        primaryStage.setTitle("Key Drawing");
```

```
        primaryStage.setScene(scene);
        primaryStage.show();

    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

    private void addKeyEvents() {

        scene.addEventHandler(KeyEvent.KEY_PRESSED, new EventHandler<KeyEvent>() {

            @Override
            public void handle(KeyEvent t) {
                path.setStroke(Color.BLACK);
                if (t.getCode() == KeyCode.LEFT && x >= 0) {
                    x -= 5;
                } else if (t.getCode() == KeyCode.RIGHT && x <= scene.getWidth()) {
                    x += 5;
                } else if (t.getCode() == KeyCode.UP && y >= 0) {
                    y -= 5;
                } else if (t.getCode() == KeyCode.DOWN && y <= scene.getHeight()) {
                    y += 5;
                }
                path.getElements().add(new LineTo(x, y));
            }
        });
    }

    public void addMouseEvents() {
        scene.setOnMouseClicked(mouseHandler);
        scene.setOnMouseDragged(mouseHandler);
        scene.setOnMousePressed(mouseHandler);
        scene.setOnMouseReleased(mouseHandler);
    }

    public void wrapLine() {
        if (x < 0) {
            x = 0;
        }
        if (x > scene.getWidth()) {
            x = (int) scene.getWidth();
        }
        if (y < 0) {
            y = 0;
        }
        if (y > scene.getHeight()) {
            y = (int) scene.getHeight();
        }
```

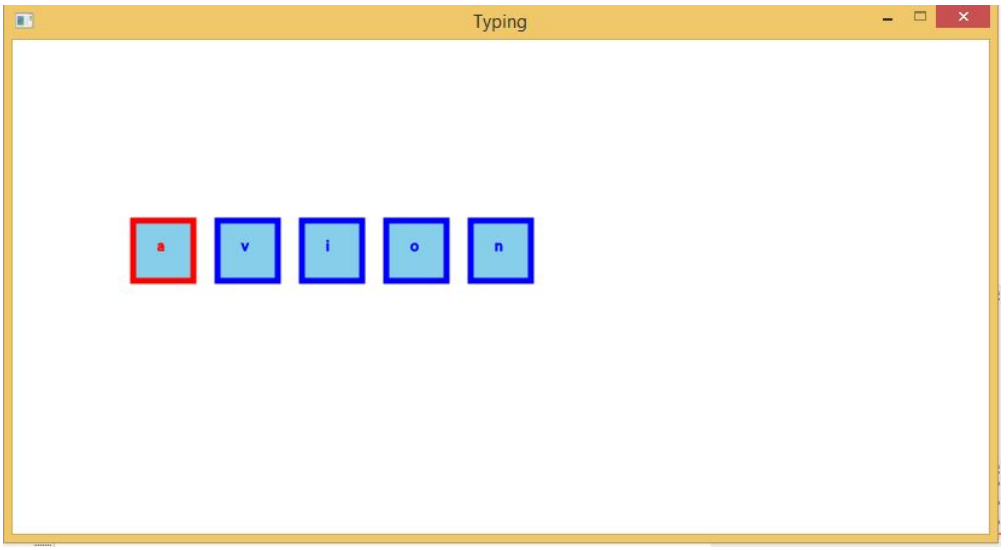
```
    }

    EventHandler<MouseEvent> mouseHandler = new EventHandler<MouseEvent>() {

        @Override
        public void handle(MouseEvent mouseEvent) {
            if (mouseEvent.getEventType() == MouseEvent.MOUSE_PRESSED) {
                erasePath.getElements().add(new MoveTo(mouseEvent.getX(),
mouseEvent.getY()));
            } else if (mouseEvent.getEventType() == MouseEvent.MOUSE_DRAGGED) {
                erasePath.getElements().add(new LineTo(mouseEvent.getX(),
mouseEvent.getY()));
            }

        }
    };
}
```

PRIMER 6

Cilj i ovog zadatka je prikaz primene korišćenja događaja sa tastature.

Napraviti aplikaciju za učenje brzog kucanja. Korisniku izlaze reči na ekran a on mora da ih napiše na tastaturi.

Potrebno vreme: 10 minuta



Slika 13.6 .6: Prikaz reči na monitoru koje se onda ukucavaju preko tastature

PRIMER 6 - OBJAŠNJENJE I UPUTSTVO ZA REŠAVANJE

U nastavku je dato pojašnjenje rešenja zadatka 6

Objašnjenje i uputstva:

1. Napraviti JavaFX Application

2. Na scenu postaviti samo Canvas
TCanvas canvas = new TCanvas(800, 400);
root.getChildren().add(canvas);

3. TCanvas nasledjuje klasu Canvas

4. U konstruktoru TCancas postaviti attribute našeg kanvasa, odrediti dimenzije i pozvati metode za odabir reči koja će se na canvasu ispisati, iscrtati dugmad i dodati akcije
setRandomWord();
draw();
addEvents();

5. odabir reči se vrši naredbom
tmpWord = words[rand.nextInt(words.length)];

6. metod draw() treba na cleanuje panel I pozove metod drawRandomWord(); koji će da preračuna pozicije rectangle-a koje iscrtava i petljom ih postaviti na panel

7. dodati akcije za doga]aj KeyEvent.KEY_PRESSED

PRIMER 6 - REŠENJE MAIN KLASE

Programiski kod koji predstavlja rešenje Main klase zadatka 6

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

/**
 *
 * @author Aleksandra
 */
public class Main extends Application {
```

```
    @Override
    public void start(Stage primaryStage) {

        Group root = new Group();
        Scene s = new Scene(root, 800, 400);

        TCanvas canvas = new TCanvas(800, 400);

        root.getChildren().add(canvas);
        primaryStage.setTitle("Typing...");
        primaryStage.setScene(s);
        primaryStage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

PRIMER 6 - REŠENJE TCANVAS KLASE

Programski kod koji predstavlja rešenje TCanvas klase zadatka 6

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import java.util.Random;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

/**
 *
```

```
 * @author Aleksandra
 */
public class TCanvas extends Canvas {

    private final int L_SIZE = 50;
    private final GraphicsContext gc2d = this.getGraphicsContext2D();
    private String[] words = new String[]{"predmet", "ucenje", "zdravlje",
"semestar", "automobil", "fakultet", "desetka", "kuciste", "tastatura", "avion"};
    private String tmpWord;
    private int tmpInd = 0;
    private Random rand = new Random();
    private boolean hit = false;

    public TCanvas(int width, int height) {
        super();
        requestFocus();
        setFocusTraversable(true);
        setWidth(width);
        setHeight(height);
        setRandomWord();
        draw();
        addEvents();
    }

    public void draw() {
        gc2d.clearRect(0, 0, getWidth(), getHeight());
        gc2d.setFill(Color.SKYBLUE);
        gc2d.setStroke(Color.BLUE);
        gc2d.setLineWidth(5);
        gc2d.clearRect(0, 0, getWidth(), getHeight());
        drawRandomWord();
    }

    public void setRandomWord() {
        tmpWord = words[rand.nextInt(words.length)];
    }

    public void drawRandomWord() {
        for (int i = 0; i < tmpWord.length(); i++) {
            gc2d.fillRect(100 + i * (L_SIZE + 20), getHeight() / 2 - L_SIZE,
L_SIZE, L_SIZE);
            if (i == tmpInd) {
                gc2d.setStroke(Color.RED);
            }
            gc2d.strokeRect(100 + i * (L_SIZE + 20), getHeight() / 2 - L_SIZE,
L_SIZE, L_SIZE);
            gc2d.setLineWidth(1);
            gc2d.strokeText(tmpWord.charAt(i) + "", i * (L_SIZE + 20) + 120,
getHeight() / 2 - L_SIZE + 25);
            gc2d.setLineWidth(5);
            gc2d.setStroke(Color.BLUE);
        }
    }
```

```
    private void addEvents() {

        addEventHandler(KeyEvent.KEY_PRESSED, new EventHandler<KeyEvent>() {

            @Override
            public void handle(KeyEvent t) {
                hit = ((tmpWord.charAt(tmpInd) + "")).equals(t.getText());
                if (hit) {
                    if (tmpInd < tmpWord.length() - 1) {
                        tmpInd++;
                    } else {
                        tmpInd = 0;
                        hit = false;
                        setRandomWord();
                    }
                    draw();
                }
            }
        });

    }
}
```

PRIMER 7

Cilj ovog zadatka je prikaz animacije rotacije kroz Javu FX

Napraviti aplikaciju koja prikazuje rotiranje 8 lukova od celog kruga (po 45 stepeni) od početka do kraja prozora. Svaki luk treba da ima drugu boju.

Potrebno vreme: 10 minuta

Slika 13.7 .7: Animacija: Rotiranje lukova različitih boja

ZADATAK 7 - REŠENJE

Programski kod koji predstavlja rešenje zadatka 7

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
```

```
 */

import java.util.Random;
import javafx.animation.RotateTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 *
 * @author Aleksandra
 */

public class ParallerDemo extends Application {

    private Random rand;
    TranslateTransition translateTransition;
    RotateTransition rotateTransition;

    @Override
    public void start(Stage primaryStage) {

        rand = new Random();

        Arc[] arcs = new Arc[8];

        for (int i = 0; i < arcs.length; i++) {
            Arc tmpArc = new Arc(50, 250, 50, 50, i * 45, 45);
            tmpArc.setType(ArcType.ROUND);
            arcs[i] = tmpArc;
        }

        AnchorPane root = new AnchorPane();

        for (int i = 0; i < arcs.length; i++) {
            arcs[i].setFill(new Color(rand.nextDouble(), rand.nextDouble(),
rand.nextDouble(), 1));
            setTranslateAnimation(arcs[i]);
            setRotateAnimation(arcs[i]);
            root.getChildren().add(arcs[i]);
        }
```

```
        Scene scene = new Scene(root, 600, 550);

        primaryStage.setTitle("Parallel animation");
        primaryStage.setScene(scene);
        primaryStage.show();
    }


    public void setTranslateAnimation(Node node){
        translateTransition = new TranslateTransition(Duration.millis(4000), node);
        translateTransition.setFromX(0);
        translateTransition.setToX(500);
        translateTransition.setCycleCount(Timeline.INDEFINITE);
        translateTransition.setAutoReverse(true);
        translateTransition.play();
    }

    public void setRotateAnimation(Node node){
        rotateTransition = new RotateTransition(Duration.millis(4000), node);
        rotateTransition.setByAngle(360f);
        rotateTransition.setCycleCount(Timeline.INDEFINITE);
        rotateTransition.setAutoReverse(true);
        rotateTransition.play();
    }

    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
        launch(args);
    }

}
```

Poglavlje 14

Vežba – Zadaci za individualne vežbe

ZADACI ZA INDIVIDUALNE VEŽBE - OD 1. DO 4. ZADATKA

Prva 4 zadatka za individualni rad studenata

Zadatak 1 (15 minuta):

Napraviti program u JaviFX za računanje površine kvadrata, pravougaonika i trougla. Potreno je da program ima polja za unos svih potrebnih vrednosti za površinu pojedinačnog elementa. Potrebno je da posotji dugme za računanje površine. Ukoliko korisnik ne unese neki parametar treba ispisati validacionu poruku. Ukoliko unese sve parametre treba mu prikazati rezultat u labeli.

Zadatak 2 (15 minuta):

Napraviti program u JaviFX koij će generisati 8 random tački potom ih spojiti u putanju a potom napraviti kvadrat koji treba da idete po toj putanji koristeći animaciju.

Zadatak 3 (15 minuta):

Napraviti program u JaviFX koji svaki karakter otkucan na tastaturi pretvara u broj po (UTF-8) šifrovanju karaktera a potom sve brojeve sabira.

Zadatak 4 (15 minuta):

Napraviti program u JaviFX koji omogućava korisniku da računa kamatnu stopu po formuli sa sajta: http://hr.wikipedia.org/wiki/Slo%C5%BEeni_kamatni_ra%C4%8Dun

ZADACI ZA INDIVIDUALNE VEŽBE - OD 5. DO 8. ZADATKA

Dodatni zadaci za individualne vežbe

Zadatak 5 (15 minuta):

Napraviti program u JaviFX koji crta krugove tako što korisnik koristi strelice na tastaturi slično zadatku sa vežbi.

Zadatak 6 (15 minuta):

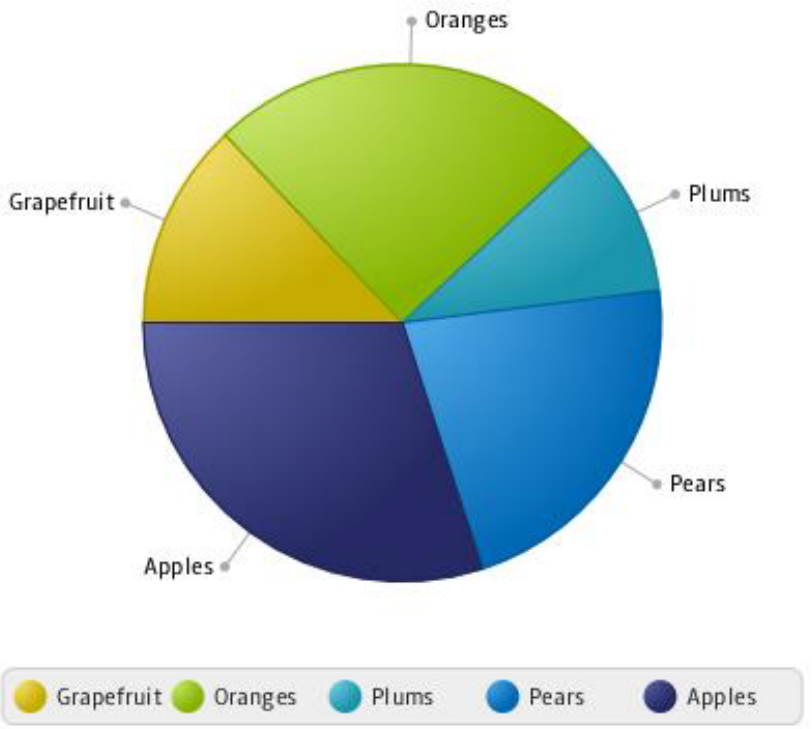
Napraviti program u JaviFX koji omogućuje korisniku da izabere FileChooser-om mp3 fajl a potom ga pušta.

Zadatak 7 (15 minuta):

Napraviti program koji animira objekat klase Circle tako što se nad objektom mogu vršiti tri vrste animacije pomoću klasa TranslateTransition, FadeTransition i ScaleTransition. Biranje animacije koja će se izvršiti se obavlja pomoću klase CheckBox i moguće je vršiti više animacija istovremeno.Z

Zadatak 8 (15 minuta):

Napraviti program u JaviFX-u koji prikazuje grafik kao na slici. Pored grafika, potrebno je kreirati 5 TextField-a gde korisnik može da unese vrednosti koje želi da prikaže. Promenu na dijagramu definisati pritiskom na taster Enter u okviru bilo kog tekstualnog polja.



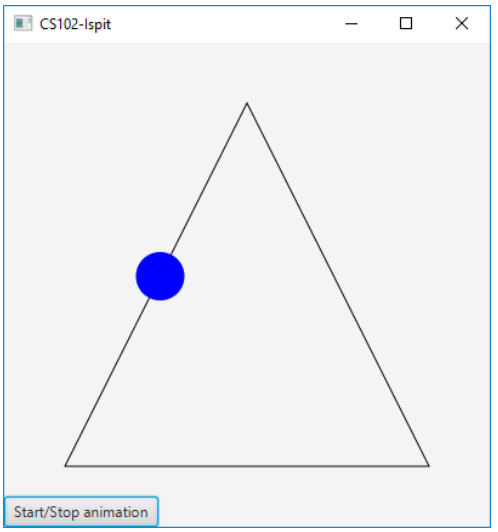
Slika 14.1 .1: - Prikaz dijagrama sa prikazom nayiva boja segmenata dijagrama

ZADACI ZA INDIVIDUALNE VEŽBE - OD 9. DO 11. ZADATKA

Još dva zadatka za individualni rad studenata.

Zadatak 9 (15 minuta):

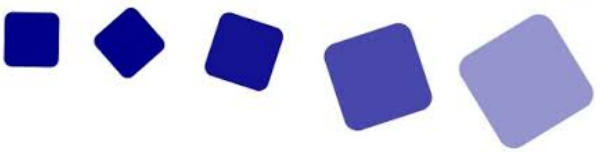
Napisati JavaFX aplikaciju u kojoj se izvršava animacija kruga po putanji trougla. Potrebno je da aplikacija ima dugme preko kog se animacija pokreće i zaustavlja.



Slika 14.2 .2: - Kretanje kruga po ivicama trougla

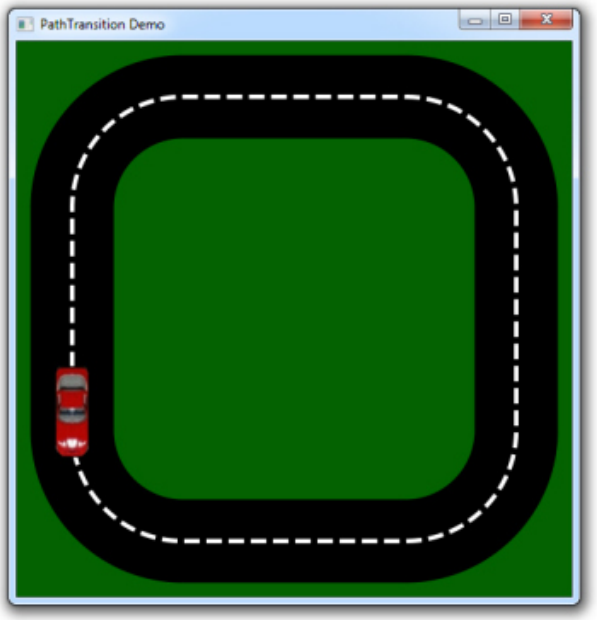
Zadatak 10 (15 minuta):

Napisati JavaFX aplikaciju u kojoj se izvršava animacija kvadrata na osnovu slike.



Slika 14.3 .3: - Animacij s akvadratima

Zadatak 11 (15 minuta):
Napisati JavaFX aplikaciju u kojoj se izvršava simulacija kretanja automobila na osnovu slike.



Slika 14.4 .4: - Simulacija kretanja automobila

DOMAĆI ZADATAK

UPUTSTVO ZA SLANJE DOMAĆEG ZADATKA

Svaki student će od svog asistenta dobiti poseban domaći zadatak za lekciju CS202-L02.

Student je dužan da asistentu pošalje rešenje zadatka u roku od 7 dana od dana prijema zadatka.

Posle tog roka, rešenje zadatka se ocenjuje sa 50% umanjenja poena.

Prilikom slanja domaćeg zadatka svom asistentu neophodno je da ispunite sledeće:
Subject mail-a mora biti CS202-DZbr (u slučaju kada šaljete domaći za drugu nedelju to je CS202-DZ01)
U prilogu mail-a treba da se nalazi projekat koji se ocenjuje imenovan na sledeći način **CS202-DZbr-ImePrezimeBrojIndeksa**.
Na primer, CS202-DZ01-AleksandraArsic123
Poželjno je uraditi i printscreen koda pre pokretanja programa. Telo maila treba da ima pozdravnu poruku

Studenti prve grupe u Beogradu zadatak na pregled šalju na mail lazar.mrkela@metropolitan.ac.rs

Studenti druge i treće grupe u Beogradu kao i internet studenti zadatak na pregled šalju na mail aleksandra.arsic@metropolitan.ac.rs.

Studenti iz Niša zadatak na pregled šalju na mail jovana.jovic@metropolitan.ac.rs.

Zaključak

REZIME

Pouke ove lekcije

1. Osnovna klasa događaja u JavaFX je ***javafx.event.Event,*** koja je podklasa klase ***java.util.EventObject***. Podklase klase ***Event*** definišu specifične tipove događaja , kao što su događaji akcije, događaji prozora, događaji miša, i događaji tastature. Ako neki čvor može da “ispali” neki događaj, svaka podklasa čvora može da “ispali” isti tip tip događaja.
2. Objekat klase obrađivača mora da primeni odgovarajući interfejsa obrađivača događaja JavaFX obezbeđuje interfejs obrađivača ***EventHandler<T extends Event>*** ta svaku klasu događaja T. Interfejs obrađivača sadrži metod ***handle(T e)*** za obradu događaja ***e.***
3. Objekat obrađivača se mora d aregistruje kod izvornog objekta. Metodi za registraciju zavise od tipa događaja. Za svaki događaj akcije koristi se metod ***setOnAction().*** Za događaje miša, koristi s emetod***setOnMousePress*** – slučaj pritiska dugmeta miša. Za događaje tastature, koristi se metod ***setOnKeyPressed.***
4. Unutrašnja ili povezana klasa, je klasa koja se definiše unutar druge klase. Unutrašnj aklasa može da poyiva podatke i metode koji su definisani u spoljnje klase sa kojom je povezana. Zato, ne treba da prebacite reference spoljnje klasa u konstruktor unutrašnje klase.
5. Anonimna unutrašnja klasa se može koristiti da se skrati kod koji obrađuje događaje. Pored toga, može se koristiti i lamda izraz koji značajno uprošćava kod za obradu događaja funkcionalnih interfejsa obrađivača.
6. Funkcionalni interfejs je interfejs sa tačno jednim apstraktnim metodom. Poznat je i pod nazivom: intefejs SAM – ***Single Abstract Method.***
7. Objekat ***MouseEvent*** se “ispaljuje” uvek kada se pritisne dugme miša, otpusti, pomera ili prektriva mišem neki čvor ili scena. Metod ***getButton()*** se koristi da bi se utvrdilo dugme koje je pritisnotio za generaciju događaja.
8. Događaj ***KeyEvent*** se ispaljuje uvek kada se pritiste ili otšusti taster nekog čvora ili scene. Metod ***getCode()*** vrača vrednost koda za to dugme. .
9. Primerak klase Observable je objekat pod posmtranjem, koji koristi metod addListener(InvalidationListener listener) radi dodavanja osluškivača. Kada se promeni vrednost svojstva, obaveštava se osluškivač. Klasa osluškivača primenjuje interfejs ***InvalidationListener***, koji koristi metod koji obrađuje slučaj kod promene vrednosti svojstva
10. Abstraktna klasa ***Animation*** obezbeđuje osnovnu funkcionalnost za animaciju JavaFX . Klase ***PathTransition***, ***FadeTransition*** i ***Timeline*** su specijalizovane klase za realizaciju animacije

REFERENCE

Korišćena literatura

1. Y. Daniel Liang, Introduction to Java Programming, Comprehensive Version, Chapter 15, 10th edition, Pierson – preporučeni udžbenik