10. Događaji (events)

Programi koje smo dosada pisali izvršavali su se tako su započinjali svoje izvršavanje od jedne točke i onda predvidljivim tokom prolazili kroz niz naredbi. Eventualno bi korisnik trebao unijeti neku informaciju, a na kraju rezultat su bili neki obrađeni podaci ispisani na neki od izlaza.
Međutim komercijalne aplikacije najčešće se na izvršavaju na opisani način. Pogledajte NotePad ili Word aplikaciju. Oba programa reagiraju na široki opseg korisničkih akcija, poput klikanja mišem po raznim elementima sučelja, pritiskanje neke od tipaka na tipkovnici, ...
U svakom slučaju aplikacija u kratkom vremenu odgovara na svaku od ovih akcija. Odgovor može biti npr. zatvaranje prozora, snimanje datoteke, dodavanje karaktera u tekst,...
Nakon što je izvršio određenu akciju program čeka da korisnik zada slijedeću. Ova vrsta programa izgleda poput sluge koji čeka korisnikove naredbe.

Ovo poglavlje je uvod u pisanje "programa sluge" korištenjem Java mehanizama koji omogućavaju ovakav rad. Posebno ćemo baviti vrijeme kako napisati interaktivni program koji odgovara na akcije miša (kretanje, klikanje) po ekranu. Ova vrsta programiranja zove se **programiranje na osnovu događaja(event-driven).**

Također ćemo vidjeti kako se piše program koji se izvršava po satu (sat driven). Ovaj program kreira objekt nazvan Timer . Timer generira pravilan niz događaja tj. otkucaja. Ostatak programa odgovara na svaki otkucaj nekom akcijom.

SADRŽAJ

---

1. Događaji

Java programi mogu reagirati na široki niz korisničkih akcija, npr. klikanje miša po ekranu, promjenu veličine prozora, pritisak tipke na tipkovnici i mnoge druge. Svaka od tih akcija naziva se **događaj** (**event**). Kada se dogodi neki od događaja, Java kreira objekt **događaja** (**event object**) koji sadržava reprezentaciju događaja.

Postoje različite vrste događaja. Npr. događaj koji se generira pritiskanjem klikanjem miša po ekranu pripada klasi MouseEvent. Postoji niz drugih akcija koje proizvode objekt klase MouseEvent: pritiskanje (lijevog) dugmeta miša, otpuštanje dugmeta miša, dvostruki klik, itd.

Pritiskanjem tipke na tipkovnici kreira se KeyEvent objekt, pritiskanje dugmeta kreira ActionEvent objekt, itd.

Jednom kad je objekt događaja kreiran , Java će ga proslijediti metodi koja je izabrana da obradi određenu vrstu događaja. Takve metode nazivamo **metoda slušač (listener method).** Takva metoda mora za određeni događaj izvršiti odgovarajuću akciju. Npr. ako pišete metodu slušač za koja odgovara na klik mišem , **morate** je nazvati `mouseClicked`. Ona koja odgovara na pritisak tipke na tastaturi mora se zvati `keyTyped`, itd. Kompletna lista naziva dana je u odjeljku 3.
Svaka metoda slušač ima jedan parametar koji odgovara objektu događaja za koji metoda treba izvršiti određenu akciju. Metoda slušač može iz tog objekta dobiti detaljnije informacije o određenom događaju.

Tijek izvršavanja interaktivnog programa tj. programa čije je izvršavanje zasnovano na događajima može se podijeliti u dvije faze:

1. Faza uspostave. Ova faza može sadržavati kreiranje objekata potrebnih za slijedeću fazu, inicijalizaciju varijabli, itd.

2. Interaktivna faza. Tijekom ove faze program čeka na događaje koje inicira korisnik, sat ili operativni sustav. Čim se događaj dogodi izvršava se odgovarajuća metoda slušač. Često odgovor na događaj uključuje promjene na području prikaza aplikacije tj. ekranu. Ako program ne osigura odgovarajuću metodu slušača, događaj se jednostavno ignorira. Ako nema događaja program jednostavno čeka da se pojavi neki događaj.

Ako metoda slušač ne obavi zadatak dovoljno brzo može se dogoditi da se slijedeći događaj dogodi prije kraja njenog izvršavanja. Da bi se omogućila obrada ovako generiranih događaja svaki generirani objekt događaja sprema se u red. Ovaj red naziva se **red otpreme događaja** (**event dispatching queue**). Čim metoda slušač koja obrađuje prethodni događaj završi s obradom, slijedeći događaj (ako ga ima u redu) šalje se na obradu.

Jedna od posljedica korištenja reda otpreme događaja je da ako određena metoda zaplete u vremenski zahtjevnu obradu, svi ostali događaji moraju čekati. Stoga je potrebno da metode slušači obave svoje zadatke što je brže moguće. Inače će izgledati kao da je program blokirao i ne odgovara na korisnikove akcije.

Pretpostavimo da smo napisali `mouseClicked` metodu za koju želimo da se izvrši svaki put kad se klikne mišem u područje prikaza. Potrebno je učiniti slijedeće stvari:

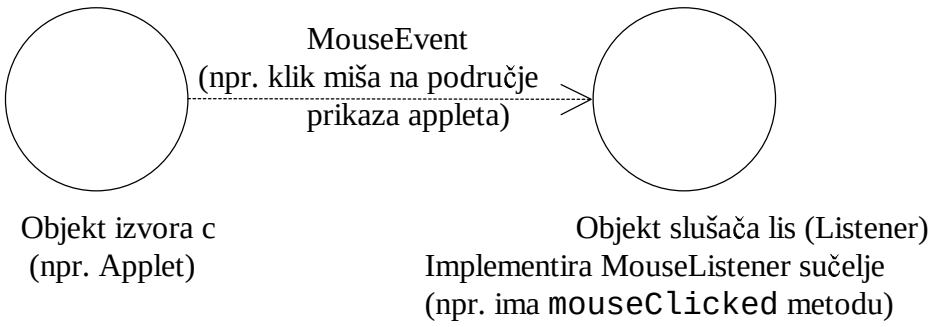
1. Metoda slušač ne može biti samostalna u memoriji. Ona mora biti pridružena s objektom koji nazivamo objekt slušač (**listener object)**. Možemo odabrati bilo koji implementira *MouseListener* sučelje(interface). To znači da taj objekt mora imati definirano slijedećih pet metoda: `mouseClicked`, `mouseEntered`, `mouseExited`, `mousePressed` i `mouseReleased`.

2. Potrebno je identificirati *izvor* događaja. To će biti objekt koji upravlja s komponentom sučelja na koju smo kliknuli. Ako smo kliknuli na područje prikaza appleta (kao u primjerima koji slijede) objekt izvor događaja je Applet objekt.

3. U fazi uspostave programa potrebno je uključiti izraz kojim se povezuje objekt slušača s objektom izvora. Taj postupak nazivamo **registracija** objekta slušača. Ako je c izvor događaja (npr. Applet objekt) tada će taj objekt imati niz metoda za registriranje slušača događaja (event listeners). Npr. ako je lis objekt slušača i želimo slušati klikanje miša na c onda se registracija ostvaruje slijedećim izrazom:

```
c.addMouseListener(lis);
```

Nakon izvršavanja prethodnog izraza svaki događaj miša uzrokovan klikanjem mišem po komponenti `c` bit će proslijeđen objektu `lis` , i bit će izvršena njegova `mouseClicked` metoda. Prevodilac će dopustiti da se lis koristi kao parametar `addMouseListener` metode samo ako lis objekt implementira MouseListener sučelje. To znači da lis **mora** imati `mouseClicked` metodu.



Moguće je registrirati više objekata slušača za neki od izvora tj. za svaku vrstu događaja posebno. Također je moguće odregistrirati objekte slušače kad više nisu potrebni.

Kad smo završili s registracijom potrebno je napisati kod koji će izvršiti određene akcije na osnovu generiranog događaja. Kod pišemo unutar metode slušača.

Postoji jedno nezgodno svojstvo predložene sheme. Naime **objekti slušači moraju definirati sve metode slušača** određenog sučelja, iako nam u nekom programu npr. treba samo jedna od njih.
Recimo da želimo definirati MouseListener objekt koji će imati `mouseClicked` metodu koja odgovara na svaki klik miša. Unutar te metode napisat ćemo kod koji izvršava određenu akciju. Iako nam ostale metode sučelja MouseListener nisu potrebne bit će ih potrebno definirati za ovaj objekt. Tijelo metoda ostavit ćemo praznim. Klasa će izgledati ovako:

```
public class MyMouseListener implements MouseListener
```

```
{  public void mouseClicked(MouseEvent e)
   {  Odgovori na klik mišem
   }

   public void mouseEntered(MouseEvent e) { }

   public void mouseExited(MouseEvent e) { }

   public void mousePressed(MouseEvent e) { }

   public void mouseReleased(MouseEvent e) { }
}
```

Ako želite izbjeći definiciju metoda koje ništa ne rade postoji jedan način. Java biblioteka posjeduje klasu koja se zove MouseAdapter. Klasa MouseAdapter samo definira prazne metode iz MouseListener sučelja (pet metoda).
To ćemo iskoristiti tako da ćemo klasu slušač definirati kao nadogradnju klase MouseAdapter te unutar klase ponovo definirati stvarno potrebne metode (overriding). Ostale metode ne definiramo ponovo. Slijedeći ovaj pristup definicija klase MyMouseListener je slijedeća:

```
public class MyMouseListener extends MouseAdapter

{  public void mouseClicked(MouseEvent e)
   {  Odgovori na klik mišem predstavljen objektom e
   }
}
```

Primijetite da u gornjoj definiciji jedini naziv koji je po volji odabran je naziv klase slušača tj. `MyMouseListener.`

Za svako sučelje slušača koji ima dvije ili više metoda slušača u Java biblioteci nalazi se odgovarajuća adapter klasa.

Prije implementacije MouseListener sučelja ili nadogradnje MouseAdapter klase potrebno je uzeti u obzir jednu već spomenutu osobinu Jave. Java klasa može naslijediti samo jednu klasu. Stoga nije moguće napisati applet koji nasljeđuje MouseAdapter zato što applet uvijek nasljeđuje klasu Applet. S druge strane klasa može implementirati bilo koji broj sučelja.

PRIMJER 1:

Prvi interaktivni program koji ćemo napisati bit će applet. Njegov zadatak je vrlo jednostavan. Korisnik će klikanjem na dva mjesta na području prikaza appleta definirati liniju. Prvo će korisnik kliknuti na jednu točku i program će markirati poziciju crtanjem malog kružića. Onda će korisnik kliknuti na drugu točku i prvi kružić će biti izbrisan te će biti nacrtana linija definirana s dvije kliknute točke. Svaki daljnji klik bit će ignoriran.

Za funkcioniranje ovog appleta potrebno je definirati skup vrijednosti (varijabli) koje definiraju prikaz na ekranu. U ovom primjeru koristimo slijedeće vrijednosti:

1. Cjelobrojnu vrijednost 'faza' koja pokazuje u kojoj smo fazi izvršavanja programa. faza=0 pokazuje da mišem dosada nismo uopće kliknuli. faza=1 znači da je mišem kliknuto samo jednom i da je izabrana prva krajnja točka linije. faza=2 znači da je mišem kliknuto dva puta i da su poznate obje krajnje točke.

2. Koordinate x0, y0 prve točke. (Ove vrijednosti su poznate samo u fazi 1 i fazi 2)

3. Koordinate x1, y1 druge točke. (Ove vrijednosti su poznate samo u fazi 2)

Kako će metoda slušač (`mouseClicked`) odgovoriti na klik mišem ovisit će o trenutnoj fazi. Ako je faza=0, postavit će koordinate x0 i y0 na trenutne koordinate miša i fazu postaviti na 1. Ako je faza = 1, postavit će koordinate klika u x1 i y1 i postaviti fazu u 2. U fazi 2 neće raditi ništa.

Kako metoda `mouseClicked` dolazi do koordinata klika ? Koristi se vrijednosti koje su proslijeđene preko parametra `e`. Svaka metoda slušača ima jedan parametar koji reprezentira događaj koji se dogodio. U slučaju klika mišem, događaj će biti tipa MouseEvent te će imati dvije asocirane metode koje se nazivaju `getX` i `getY` i koje vraćaju vrijednosti koordinata pozicije na koju se kliknulo.

Primijetite u definiciji metode `mouseClicked` da se u dva slučaja kad metoda mijenja vrijednosti varijabli u kojima je definiran prikaz ne ekranu, nakon promjene poziva metoda `repaint()`čiji je zadatak osvježavanje sadržaja ekrana. Ovo osigurava da slika odmah prati klikove miša po ekranu.

Metoda `paint` mora također uzeti u obzir vrijednost varijable `faza`. Ako je ta vrijednost 0, nema ničega za ispisati. Ako je faza 1 metoda treba nacrtati mali kružić na poziciji (x0,y0). Ako je faza 2, treba nacrtati liniju od (x0,y0) do (x1,y1).

Metoda `init` treba učiniti samo jednu stvar. U njoj se registrira metoda slušač za Applet objekt. Drugim riječima tu se registrira Applet objekt (slušač) sa objektom izvorom tj. sa samim sobom. Koristi se slijedeći izraz:

```
addMouseListener(this);
```

Primijetite da applet referira na samog sebe s ključnom riječi `this`.

Koko smo prije spomenuli, klasa Applet ne može nadograditi klasu MouseAdapter jer već nadograđuje klasu Applet. Zbog toga mora implementirati MouseListener sučelje te zbog toga definirati svih pet metoda primijenjenog sučelja. Samo će metoda `mouseClicked` nešto raditi, a ostale definiramo kao prazne metode.

PRIMJER 1

(verzija u kojoj Applet objekt osluškuje klikove mišem.)

```
// Interaktivni applet koji crta jednu liniju.
// Korisnik klikne u jednu točku.
```

```
// Zatim u slijedeću točku.
// Nakon toga se crta linija koja spaja točke.

import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;
import java.applet.Applet;

public class LineApplet1 extends Applet
   implements MouseListener

{  private int faza = 0;
   // Faza = 0 prije nego što korisnik klikne prvu točku.
   // Faza = 1 nakon prvog klika.
   // Faza = 2 nakon drugog klika.

   private int x0, y0;
   // Koordinate početka linije.

   private int x1, y1;
   // Koordinate kraja linije.

   public void init()
   {  addMouseListener(this);
   }

   public void paint(Graphics g)
   {  if (faza == 0) return;

      Graphics2D g2 = (Graphics2D) g;
      if (faza == 1)
      {  /* Crtaj malu kružnicu s centrom u(x0,y0). */
         double radius = 5;
         Shape circle =
            new Ellipse2D.Double
               (x0-radius, y0-radius, 2*radius, 2*radius);
         g2.draw(circle);
         return;
      }

      /* (Pretpostavi faza == 2)
         Crtaj liniju od (x0,y0) to (x1,y1). */
      Shape line = new Line2D.Double(x0,y0,x1,y1);
      g2.draw(line);
   }

   /* MouseListener metode. */

   public void mouseClicked(MouseEvent e)
   {  if (faza == 0)
      {  x0 = e.getX();
         y0 = e.getY();
         faza = 1;
         repaint();
      }
```

```
        else if (faza == 1)
        {  x1 = e.getX();
           y1 = e.getY();
           faza = 2;
           repaint();
        }
        /* (Za faza == 2 ne čini ništa) */
    }

    public void mouseEntered(MouseEvent e) {}

    public void mouseExited(MouseEvent e) {}

    public void mousePressed(MouseEvent e) {}

    public void mouseReleased(MouseEvent e) {}

}
```

---

2. Korištenje unutarnjih klasa (Inner Classes)

Kako smo vidjeli kod pisanja appleta koji osluškuje klikove miša, bilo je potrebno napisati prazne (dummy) definicije nekorištenih metoda MouseListener sučelja.

Bilo bi zgodno definirati odvojene objekte čiji će zadatak biti slušanje klikova mišem te da budu nadogradnja klase MouseAdapter. Tako bi izbjegli definiranje neželjenih metoda sučelja.

Na prvi pogled ovaj pristup ima nedostatak koji se očituje u tome da i applet i slušač (listener) trebaju imati pristup varijablama koje određuju prikaz na ekranu (npr. polja `faza`, `x0`, `y0`, `x1` i `y1` u prethodnom primjeru).

Ako te varijable držimo u appletu i označimo ih kao private što se obično čini onda bi trebali definirati metode kojima bi im mogli pristupiti iz metoda slušača tj. metode `mouseClicked` u prethodnom primjeru (sad u drugoj klasi).

Ako ih držimo u klasi slušača onda trebamo u toj klasi definirati metode s kojima im možemo pristupiti iz metode `paint` koja se nalazi u appletu.

Kako uzeli program postaje kompliciraniji.

Java omogućava rješenje ovog problema. Moguće je definirati dvije klase, recimo A i B, tako da B objekti mogu pristupati poljima objekta A, iako su ta polja označena kao private.

Potrebno je napisati definiciju klase B *unutar* definicije klase A. tako definirana klasa B se naziva **ugniježđena klasa** (**nested class**).

Ako B nije definirana kao statička klasa kažemo da je **unutarnja klasa (inner class)** klase A.

Već prije smo vidjeli da svaka varijabla klase A koja nije statička (polje) pripada objektu klase A. Svaka metoda klase A, ako nije statička, asocirana je s objektom klase A i može pristupiti poljima i metodama objekta.

Isto vrijedi i za objekte koji su tipa unutarnje (inner) klase B. Svaki B objekt je asociran s objektom A, i metode objekta B mogu pristupati privatnim poljima i metodama A objekta.

Ponekad se B objekt naziva pomoćnim objektom (helper) koji asistira asociranom A objektu.

U slučajevima kad je jedan objekt u potpunosti ovisan od objekta drugog tipa to može biti slučaj kad je potrebno da taj objekt definiramo kao unutarnju (inner) klasu.

To vrijedi za objekte slušača (listener) i često ih definiramo na navedeni način. Možemo promatrati objekt slušača kao pomoćni objekt koji asistira objektu koji je izvor događaja.

Postoji još jedna tehnika za dobivanje još kompaktnijeg koda. To je da definiramo klasu slušača unutar metode klase izvora. Tu tehniku u kojoj se kreiraju anonimne klase nećemo obraditi u ovim predavanjima.

Slijedi nova verzija programa iz primjera 1. Radi se o programu s istom funkcijom, ali drukčije strukture. Metode slušača uklonjene su iz appleta i stavljene u unutarnji (inner) objekt asociran s appletom. Kako je unutarnji objekt slušača nadogradnja MouseAdapter klase u njemu nije potrebno definirati sve metode MouseListener sučelja, već samo potrebne.

Izmjene u odnosu na prethodni program su podebljane. Unutarnja klasa nazvana je ClickListener.

PRIMJER 1 B
(verzija gdje je objekt slušača definiran unutarnjom klasom)

```
public class LineApplet2 extends Applet

{  private int faza = 0;
   // Faza = 0 prije nego što korisnik klikne prvu točku.
   // Faza = 1 nakon prvog klika.
   // Faza = 2 nakon drugog klika.

   private int x0, y0;
   // Koordinate početka linije.

   private int x1, y1;
   // Koordinate kraja linije.


   public void init()
   {  addMouseListener(new ClickListener());
   }
```

```
        public void paint(Graphics g)
        {  if (faza == 0) return;

           Graphics2D g2 = (Graphics2D) g;
           if (faza == 1)
           {  /* Crtaj malu kružnicu s centrom u(x0,y0). */
              double radius = 5;
              Shape circle =
                 new Ellipse2D.Double
                    (x0-radius, y0-radius, 2*radius, 2*radius);
              g2.draw(circle);
              return;
           }

           /* (Pretpostavi faza == 2)
              Crtaj liniju od (x0,y0) do (x1,y1). */
           Shape line = new Line2D.Double(x0,y0,x1,y1);
           g2.draw(line);
        }
        /******************Unutarnja (Inner) klasa
***************/

        public class ClickListener extends MouseAdapter

        {  public void mouseClicked(MouseEvent e)
           {  if (faza == 0)
              {  x0 = e.getX();
                 y0 = e.getY();
                 faza = 1;
                 repaint();
              }
              else if (faza == 1)
              {  x1 = e.getX();
                 y1 = e.getY();
                 faza = 2;
                 repaint();
              }
              /* (Ne čini ništa ako je faza == 2.) */
           }
        }
    }
```

Primijetite da se poljima `faza, x0`, `y0`, `x1`, `y1` u LineApplet objektu može pristupiti iz metode asociranog listener objekta.

Slijedi još jedan program koji se zasniva na crtanju linija. Program omogućava korisniku da nacrta linija koliko želi. Program predstavlja primitivnu formu programa za crtanja. Svaki put kad dodamo liniju pozvat će se `repaint` metoda što će izazvati poziv `paint` metode. Paint metoda obnavlja cijeli ekran i bit će potrebno imati pohranjene podatke za sve dotada nacrtane linije.

Prema tome bit će potrebno držati zapisano sve dotada nacrtane linije. Prirodan način je da ih pohranimo u ArrayList. Prilikom crtanja proći ćemo po svim elementima liste i nacrtati ih na ekranu.

Novi program će imati slijedeće varijable:

1. Cjelobrojnu vrijednost 'faza' koja pokazuje u kojoj smo fazi izvršavanja programa. faza=0 znači da je program spreman za prihvat koordinata prve točke linije. faza=2 znači da je program učitao prvu točku i da je sprema za drugu točku.

2. Koordinate `x0`, `y0` za prvu točku linije. (Ove vrijednosti bit će poznate kada faza bude 1)

3. Lista `lineList sa svim dosada definiranim linijama`.

Ideja programa je u biti slična kao iz prvog primjera. Korisnik klikne mišem. Odgovor je pozivanje metode slušača. Metoda slušača ažurira koordinate linija i poziva metodu `repaint`, koja onda poziva `paint`. `paint` koristi ažurirane sadržaje za obnavljanje sadržaja ekrana. Već prije smo vidjeli da se metoda `paint` poziva i nakon prekrivanja, maksimiziranja, minimiziranja, itd. prozora ekrana..

PRIMJER 2

```
// Interaktivni applet koji crta višestruke linije.
// Korisnik klika na prvi pa onda na drugi kraj.
// Zatim se crta linija koja spaja zadane točke.
// Postupak se ponavlja.

import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;
import java.util.*;
import java.applet.Applet;

public class ManyLinesApplet extends Applet

{  private int faza = 0;
   // If faza=1, (x0,y0) = početak slijedeće linije.
   // If faza=0, početak slijedeće linije nije kliknut.

   private int x0, y0;
   //Koordinate starta slijedeće linije.

   java.util.List lineList = new ArrayList();
   // Lista svih definiranih linija.

   public void init()
   {  addMouseListener(new ClickListener());
   }

   public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;
      if (faza == 1)
      {  /* Crtaj malu kružnicu s centrom u (x0,y0). */
         double radius = 5;
         Shape circle =
            new Ellipse2D.Double
               (x0-radius, y0-radius, 2*radius, 2*radius);
```

```
            g2.draw(circle);
         }

         /*Prikaži sve linije pohranjene u listi. */
         for (int i = 0; i < lineList.size(); i++)
         {  Shape nextLine = (Shape) lineList.get(i);
            g2.draw(nextLine);
         }
      }

      /******************Unutarnja (Inner) klasa ***************/

      public class ClickListener extends MouseAdapter

      {  public void mouseClicked(MouseEvent e)
         {  if (faza == 0)
            {  x0 = e.getX();
               y0 = e.getY();
               faza = 1;
            }
            else
            {  // pretpostavi da je faza == 1.
               int x = e.getX();
               int y = e.getY();
               Shape line = new Line2D.Double(x0,y0,x,y);
               lineList.add(line);
               faza = 0;
            }
            repaint();
         }
      }
   }
```

---

3. Tablica klasa događaja (event classes) i tablica sučelja slušača (listener interfaces)

Prva kolona sadrži nazive različitih tipova događaja i odgovarajuća sučelja slušača. Primijetite da se naziv sučelja slušača i tipa događaja imaju početnu zajedničku osnovu te prvi ima nastavak "Listener" a drugi "Event".

Jedina iznimka je za tip događaja MouseEvents. To je zato što postoje dva sučelja za MouseEvents, nazvana MouseListener i MouseMotionListener.

Druga kolona sadrži nazive metoda sučelja slušača (listener interface). U svakom slučaju kad sučelje sadrži dvije ili više metoda, postoji i odgovarajuća klasa adaptera. Naziv adapter klase ima istu osnovu kao i naziv sučelja. (Listener zamijenjeno s Adapter)

Za registraciju objekta slušača koristi se metoda čiji je naziv "add"+naziv sučelja. Za uklanjanje se koristi "remove"+naziv sučelja

Event Class and Listener Interface	Listener Methods
ActionEvent ActionListener	actionPerformed
AdjustmentEvent AdjustmentListener	adjustmentValueChanged
ComponentEvent ComponentListener	componentHidden componentMoved componentResized componentShown
ContainerEvent ContainerListener	componentAdded componentRemoved
FocusEvent FocusListener	focusGained focusLost
ItemEvent ItemListener	itemStateChanged
KeyEvent KeyListener	keyPressed keyRepeased keyTyped
MouseEvent MouseListener	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
MouseEvent MouseMotionListener	mouseDragged mouseMoved
TextEvent TextListener	textValueChanged
WindowEvent WindowListener	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified

windowOpened

---

4. Korištenje sata (timer)

U prethodnim odjeljcima koristili smo događaje generirane mišem. Ovdje ćemo događaje generirati na drugačiji način.

Dosadašnji programi koje smo dosada vidjeli izvršavali su samo jednu sekvencu akcija. U slučaju aplikacije sekvenca je započinjala i bila određena `main` metodom.
Na njen tijek se moglo djelovati tako da se ovisno o unosu korisnika izvršavanja određeni niz naredbi. Međutim uvijek se radilo o jednoj sekvenci naredbi. Međutim Java može više od toga. Java nam omogućava pisanje programa u kojima se metode izvršavaju neovisno i paralelno svaka sa svojom sekvencom naredbi. Ako jedna sekvenca čeka na unos podataka od strane korisnika druga može neovisno o tome obavljati nekakav posao.

Takve različite sekvence izvršavanja naredbi nazivaju se **niti (threads).** Ovdje nećemo detaljnije objašnjavati niti tj. kako se kreiraju , izvršavaju , sinkroniziraju i zaustavljaju već ćemo u programu koristiti objekt koji će se izvršavati u neovisnoj niti.

Java biblioteka definira klasu objekata nazvanu Timer. Timer objekt posjeduje metodu `start`. Ako pozovemo metodu `start` Timer objekta, ona nastavlja svoje izvršavanje u novoj niti. Sve što ta neovisna nit radi je generiranje objekta događaja klase ActionEvent. U našem primjeri ti će se objekti kreirati u regularnom intervalu koji se zadaje prilikom kreacije Timer objekta.

Timer možemo zaustaviti na određeno vrijeme, nakon toga opet pokrenuti , itd.

Svaki događaj koji objekt Timer kreira stavlja se u red i bit će obrađen kada dođe na red. Događaji će biti proslijeđeni ActionListener objektu koji se registrirao na Timer. ActionListener sučelje ima samo jednu metodu koja odgovara na događaje i naziva se `actionPerformed`.

koristit ćemo tri metode objekta Timer i jedan konstruktor.

`start()`
Pokreni Timer. tj. pokreni nit koja generira ActionEvent događaje.

`stop()`
Zaustavi Timer.

`isRunning()`
Vrati true ako je Timer pokrenut. Inače vrati false.

`Timer(delay,listener)`
Kreiraj novi objekt Timer. Jednom kad je Timer pokrenut, počet će s generiranjem sekvence događaja (ActionEvent). Cjelobrojna vrijednost `delay` je vrijeme između dva sukcesivna događaja tipa ActionEvent. `listener` je ActionListener objekt koji će biti registriran od strane objekta Timer.

Koristit ćemo Timer da bismo proizveli animiranu sliku. Ponovo ćemo definirati Applet s pripadnim poljima koja definiraju prikaz. Osim toga definirat ćemo Timer koji će generirati seriju događaja (ActionEvent). Definirat ćemo ActionListener čija će metoda `actionPerformed` odgovarati na svaki od ActionEvent događaja. ActionListener će stalno u malim izmjenama modificirati sadržaj polja koja definiraju prikaz što će kao rezultat imati sliku koja se stalno mijenja. Ako interval objekta Timer (delay) smanjimo na dovoljno malu vrijednost netko tko gleda prikaz imat će utisak glatke animacije.

Svaka pojedinačna slika naziva se **okvir** (frame). Broj okvira prikazanih u jednoj sekundi naziva se brzina promjene okvira (frame rate). Za animaciju bit će nam dovoljno od 12 do 20 okvira u sekundi.

U primjeru koji slijedi animacija se sastoji od pravokutnika koji se pojavljuje na lijevoj strani, putuje na desnu stranu i na kraju iščezava na desnoj strani područja prikaza.

Metoda `actionPerformed` svaki put dodaje jedan na varijablu i onda poziva `repaint`. Varijabla je inicijalno postavljena na nulu. Zapravo radi se o brojaču koji broji broj okvira. Varijabla je nazvana `vrijeme` jer predstavlja i broj otkucaja Timer objekta.

Metoda `paint` računa položaj kvadrata na slijedeći način:

```
double x = startX + vrijeme*brzinaX;
double y = startY + vrijeme*brzinaY;
```

Kvadrat se kreira slijedećim izrazom.

```
Shape kvadrat =
   new Rectangle2D.Double(x,y,stranica,stranica);
```

Zadnji izraz je registracija MouseListener od strane appleta. Ako se klikne mišem na ekran slijedeća metoda će biti izvršena:

```
public void mouseClicked(MouseEvent e)
{  if (sat.isRunning())
      sat.stop();
   else
      sat.start();
}
```

Naredbe u ovoj metodi zaustavlja Timer ako je pokrenut i time se zaustavlja tok ActionEvent događaja. Kvadrat će stati na ekranu. Ako Timer nije bio pokrenut timer će se ponovo pokrenuti i animacija će se nastaviti. Slijedi kompletan applet.

PRIMJER 3

```
// Animirani applet.  Prikazuje mali kvadrat
// koji se lagano kreće preko područja prikaza.
// Klikni na područje prikaza za zaustavljanje kretanja,
// ili ako je zaustavljeno, klikni ponovo za pokretanje.

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.applet.Applet;
import javax.swing.*;    // (potrebno za Timer)

public class KvadratFilm extends Applet

{  Timer sat;

   private int vrijeme = 0;
   // Trenutno vrijeme (i.e. trenutni broj okvira).

   private int brzinaOkvira = 20;
   // Broj okvira u sekundi

   private final double brzinaX = 4, brzinaY = 0;
   // Koliko se piksela lik pomiče u svakom okviru

   private final double startX = -20, startY = 100;
   // Pozicija lika za okvir 0.

   private final double stranica = 20;
   // Duljina stranice kvadrata


   public void init()
   {  int delay = 1000/brzinaOkvira;
      // 'delay' =vrijeme između dva uzastopna okvira.

      sat = new Timer(delay, new ClockListener());
      // 'sat'inicijalno nije pokrenut.

      addMouseListener(new ClickListener());
   }

   public void paint(Graphics g)

   {  Graphics2D g2 = (Graphics2D) g;
      double x = startX + vrijeme*brzinaX;
      double y = startY + vrijeme*brzinaY;
      Shape kvadrat =
         new Rectangle2D.Double(x,y,stranica,stranica);
      g2.draw(kvadrat);
   }

   /* Unutarnja klasa (inner class). */

   public class ClockListener implements ActionListener

   {  public void actionPerformed(ActionEvent e)
      {  repaint();
```

```
            vrijeme++;
         }
      }

      public class ClickListener extends MouseAdapter

      {  public void mouseClicked(MouseEvent e)
         {  if (sat.isRunning())
               sat.stop();
            else
               sat.start();
         }
      }
}
```

---

5. Zadaci

1. Napiši applet koji će crtati kružnicu na slijedeći način. Korisnik prvo klikne u jednu točku koja sad predstavlja centar kružnice. Zatim klikne u drugu točku koja predstavlja jedno od točaka na kružnici. Nakon toga se nacrta kružnica.

2. Napiši drugu verziju programa iz primjera 3. Razlika neka bude u tome da klik miša mijenja smjer kretanja kvadrata.

Uputa: u metodi mouseClicked promijenite predznak koraka kretanja.

nazivi applet AmoTamo.
