OBJEKTNO PROGRAMIRANJE 2

Oznaka predmeta: OP2

Predavanje broj: 06

Nastavna jedinica: JAVA

Nastavne teme:

Problem 5 filozofa. Serijalizacija. Kloniranje. GUI, AWT, java.awt, događaji, izvori, osluškivači, adapteri, anonimna klasa. Obrada događaja koji potiču od miša. MouseWheelListener.

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

Literatura:

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

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

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

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

Problem 5 filozofa

- Za okruglim stolom sedi 5 filozofa a između njihovih tanjira nalazi se pet štapića za jelo. Za uzimanje hrane potrebna su dva štapića tako da svi filozofi ne mogu istovremeno jesti.

```
/** Stapic za jelo. */
public class Stick {
        /** Status stapica. */
        private boolean free;
        /** Stapic je inicijalno slobodan. */
        public Stick() { free = true; }
        /** Stapic se koristi. */
        public synchronized void take() {
                try {
                        while (!free) wait();
                } catch (Exception ex) {}
                free = false;
        }
        /** Stapic se oslobadja. */
        public synchronized void release() {
                free = true;
                notify();
        }
}
```

Problem 5 filozofa

```
public class Philosopher extends Thread {
   private int index;
   private Stick left;
   private Stick right;
   private int statistikajela;
   public Philosopher(int index, Stick left, Stick right) {
      this.index = index;
      this.left = left;
      this.right = right;
      statistikajela = 0;
   }
   public void run() {
      for (int i = 0; i < 1000; i++) {
         left.take();          right.take();
         statistikajela++;
         left.release();       right.release();
         yield();
      }
   }
   public String toString() {
      return ("[" + index + "] jeo [" + statistikajela + "]");
   }
}
```

Problem 5 filozofa

```
public class FivePhilosophers {
   public static void main(String[] args) {
      Stick[] sticks = new Stick[5];
      for (int i = 0; i < 5; i++) sticks[i] = new Stick();
      Philosopher a = new Philosopher(1, sticks[4], sticks[0]);
      Philosopher b = new Philosopher(2, sticks[0], sticks[1]);
      Philosopher c = new Philosopher(3, sticks[1], sticks[2]);
      Philosopher d = new Philosopher(4, sticks[2], sticks[3]);
      Philosopher e = new Philosopher(5, sticks[3], sticks[4]);
      a.start(); b.start(); c.start(); d.start(); e.start();
      try {
          //a.join();b.join();c.join();d.join();e.join();
          Thread.sleep(2000);
      } catch (InterruptedException e1) { e1.printStackTrace(); }
      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
      System.out.println(d);
      System.out.println(e);
   }
}
```

```
[1] jeo [333]
[2] jeo [176]
[3] jeo [115]
[4] jeo [226]
[5] jeo [266]
```

Serijalizacija

- Postupak serijalizacije je konvertovanje stanja objekta u niz bajtova koji se snima u datoteku.
  - Modifikator transient određuje da promenljiva kojoj je pridružen ovaj modifikator **neće** biti sačuvana u postupku serijalizacije.
- Sve promenljive objekta podrazumevano nisu transient, tako da ako se u procesu serijalizacije želi izbeći snimanje neke od ovih promenljivih mora se ispred date promenljive eksplicitno navesti modifikator transient.
- Primer serijalizacije u kojoj će se snimiti atributi objekta koji se odnose na tip vozila i kubikažu, dok se atribut koji se odnosi na marku vozila neće snimiti u datoteku (automobili). Interfejs Serializable ne deklariše niti jednu metodu (marker inteface).

```
import java.io.*;
class Serijalizacija implements Serializable {
  private transient String markaVozila;
  private String tipVozila;
  private int kubikazaVozila;
  public Serijalizacija(String markaVozila, String tipVozila,
    int kubikazaVozila) {
       this.markaVozila = markaVozila;
       this.tipVozila = tipVozila;
       this.kubikazaVozila = kubikazaVozila;}
```

Serijalizacija

```
  public String toString() {
    StringBuffer sb = new StringBuffer(40);
    sb.append("\nMarka vozila: ");
    sb.append(this.markaVozila);
    sb.append("\nTip vozila: ");
    sb.append(this.tipVozila);
    sb.append("\nKubikaza vozila: ");
    sb.append(this.kubikazaVozila);
    sb.append(" kubika");
    return sb.toString();
  }
  public static void main(String args[]) throws Exception {
    Serijalizacija vozilo = new Serijalizacija("Ford","Mustang", 4000);
    ObjectOutputStream oos = new ObjectOutputStream(
       new FileOutputStream("automobili"));
    // upisivanje objekta
    oos.writeObject(vozilo); oos.close();
    // citanje objekta
    ObjectInputStream ois = new ObjectInputStream(
       new FileInputStream("automobili"));
    Serijalizacija vozilo2 = (Serijalizacija) ois.readObject();
    System.out.println(vozilo2);
}}
```

```
Marka vozila: null
Tip vozila: Mustang
Kubikaza vozila: 4000 kubika
```

Kloniranje

- Operator dodele samo će napraviti duplikat reference na objekat.
- Ono što se želi je da se napravi duplikat objekta.
- Metoda Object.clone() omogućuje dupliciranje objekta:
  - a.clone() != a
  - a.clone().getClass() == a.getClass() , klon_a.equals(a) je true
- Metod clone():
  - proveri da li klasa datog objekta implementira interfejs Cloneable (spada u takozvani marker ili tag interfejs koji nema članove, informacija za JVM). Ako prethodno nije zadovoljeno generiše se izuzetak CloneNotSupportedException
  - ako je prehodna provera pozitivna onda će se kreirati plitka kopija objekta (shallow copy, bit-wise kopija objekta) i vratiti kao rezultat.
- Shallow copy kreira kopije atributa što je korektno ako su u pitanju primitive:
  - Problem je pri kopiranju atributa koji predstavljaju reference na objekte (pri kopiranju original i kopija će se odnositi na isti objekat na heap-u).
- Rešenje je da se uradi duboko kloniranje (deep cloning) tako da će i 'drugi' nivo kloniranja biti uključen.

Primer plitke kopije 1/2

```
class Course {
    String subject1;
    String subject2;
    String subject3;
    public Course(String sub1, String sub2, String sub3)    {
        this.subject1 = sub1;
        this.subject2 = sub2;
        this.subject3 = sub3;
    }
}
class Student implements Cloneable{
    int id;
    String name;
    Course course;
    public Student(int id, String name, Course course)  {
        this.id = id;
        this.name = name;
        this.course = course;
    }
    //Default version of clone() method, shallow copy of an object.
    protected Object clone() throws CloneNotSupportedException    {
        return super.clone();
    }}
```

Primer plitke kopije 2/2

```
public class ShallowCopyInJava {
    public static void main(String[] args)    {
        Course science = new Course("Physics", "Chemistry", "Biology");
        Student student1 = new Student(111, "John", science);
        Student student2 = null;
        try
        {
            //Creating a clone of student1 and assigning it to student2
             student2 = (Student) student1.clone();
        }
        catch (CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
        //Printing the subject3 of 'student1'
        System.out.println(student1.course.subject3); //Output : Biology
        //Changing the subject3 of 'student2'
        student2.course.subject3 = "Maths";
        //This change will be reflected in original student 'student1'
        System.out.println(student1.course.subject3); //Output : Maths
    }
}
```

Primer duboke kopije 1/2

```
class Course2 implements Cloneable{
    String subject1;
    String subject2;
    String subject3;
    public Course2(String sub1, String sub2, String sub3)    {
        this.subject1 = sub1;
        this.subject2 = sub2;
        this.subject3 = sub3;
    }
    protected Object clone() throws CloneNotSupportedException    {
        return super.clone();
    }
}
class Student2 implements Cloneable{
    int id;
    String name;
    Course2 course2;
    public Student2(int id, String name, Course2 course2)    {
        this.id = id;
        this.name = name;
        this.course2 = course2;
    }
```

Primer duboke kopije 2/2

```
    //Overriding clone() method to create a deep copy of an object.
    protected Object clone() throws CloneNotSupportedException    {
        Student2 student2 = (Student2) super.clone();
        student2.course2 = (Course2) course2.clone();
        return student2;
    }
}
public class DeepCopyInJava2{
    public static void main(String[] args)    {
        Course2 science = new Course2("Physics","Chemistry", "Biology");
        Student2 student11 = new Student2(111, "John", science);
        Student2 student22 = null;
        try{
            student22 = (Student2) student11.clone();
        }
        catch (CloneNotSupportedException e)  {
            e.printStackTrace();
        }
        System.out.println(student11.course2.subject3); //Output:Biology
        student22.course2.subject3 = "Maths";
        System.out.println(student11.course2.subject3); //Output:Biology
    }
}
```

GUI

- Grafički korisnički interfejs je karakteristika gotovo svih savremenih aplikacija
- Termin na engleskom:

  Graphical User Interface (GUI)
- GUI je vezan za rad sa prozorima:
  - ulazni podaci generišu događaje u prozorima
  - izlazni podaci se prikazuju u prozorima
- Aplikacije koje imaju samo tekstualni ulaz i izlaz
  - nazivaju se konzolnim aplikacijama
  - upravljaju celim ekranom (odnosno prozorom koji simulira ceo ekran konzole)
- Aplikacija sa GUI ne upravlja celim ekranom, već prozorima koje kreira.
- Java je prvi široko rasprostranjeni programski jezik koji na standardan način podržava programiranje GUI.

Paket AWT

- Podrška za programiranje GUI nalazi se u java.awt paketu
- Sledeći paket za programiranje GUI: javax.swing
  - sadrži neke nove i neke poboljšane komponente u odnosu na java.awt
  - pisan je u Javi pa je prenosivost potpuna
- AWT je skraćenica od *Abstract Windowing Toolkit*
  - apstraktni alati za rad sa prozorima
  - apstraktni: ne zavise od konkretne platforme
- Paket java.awt sadrži klase i interfejse koji podržavaju izlazne i ulazne aspekte GUIa.
- Paket java.awt se koristi za programiranje
  - samostalnih aplikacija
  - apleta
- Komponente koje se pojavljuju na ekranu nazivaju se i "kontrole" ili "vidžiti“

  primeri: ekranski tasteri (*button*), radio-dugmad (*radio-button*), polja za potvrde (*checkbox*), klizači (*scrollbar*), polja za tekst (*text box*), liste (*list*), padajuće liste (*combo-box, choice*).

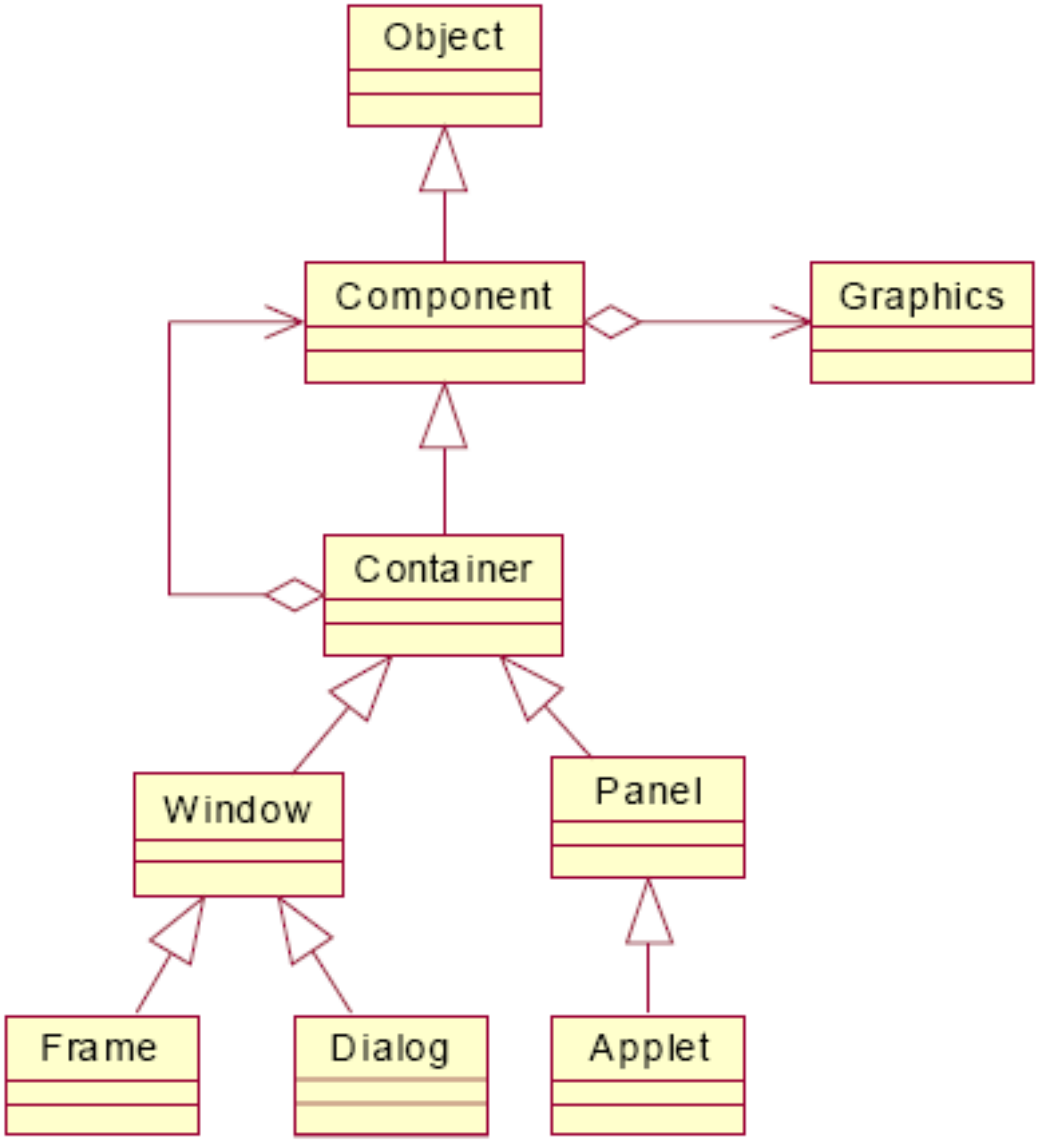
Važnije klase iz paketa java.awt

- Klasa Component je zajednička osnovna klasa za sve GUI kontrole
- Klasa Component reprezentuje nešto što:
  - ima poziciju, veličinu, može se iscrtati na ekranu i prihvata ulazne događaje
- Klasa Container je izvedena iz Component
  - objekat Container može da sadrži druge AWT komponente
- Klasa Window je izvedena iz Container (sadrži komponente)
  - njeni objekti su prozori najvišeg nivoa
  - obuhvata metode za rad sa prozorima
- Klasa Frame je izvedena iz Window
  - koristi se za kreiranje glavnog prozora aplikacije
  - objekat tipa Frame može da sadrži traku menija i da prikaže naslov
- Klasa Dialog je izvedena iz Window
  - koristi se za kreiranje prozora dijaloga
  - dijalog ima roditeljski prozor i ne sadrži meni

Važnije klase iz paketa java.awt

- Klasa Graphics omogućava crtanje/pisanje na komponentama
  - crtanje i pisanje se obavlja nad objektom klase (ili izvedene iz klase) Graphics
  - grafičke operacije modifikuju bite samo unutar odsecajućeg (*clipping*) regiona
  - crtanje ili pisanje se obavlja u tekućoj boji, koristeći tekući režim slikanja i tekući font
  - objekat Component sadrži objekat Graphics do kojeg se može stići pomoću getGraphics()
- Klasa Event je posvećena centralizovanoj obradi događaja (zastareo koncept)
  - aplikacije sa GUI su "vođene događajima"
  - definiše kompletnu listu događaja u obliku klasnih (statičkih) celobrojnih konstanti
  - ove konstante se koriste u obradi događaja da se detektuje događaj
  - atribut objekta događaja id određuje vrstu događaja – vrednost se poredi sa konstantama

Hijerarhija važnijih klasa paketa AWT



Primer "Zdravo“ u centralizovanoj obradi događaja

```
import java.awt.*;
public class Prozor extends Frame {
   public Prozor() {
        super("Prozor");
        setSize(180,80); // ili setSize(new Dimension(180,80));
        setVisible (true);
   }

   public void paint(Graphics g) {
        g.drawString("Zdravo!",50,50);
   }

   public boolean handleEvent(Event e) {
        if(e.id==Event.WINDOW_DESTROY){
                System.exit(0);
                return true;
        }
        else return false;
   }

   public static void main(String args[]){
        Prozor prozor = new Prozor();
}
```

O primeru “Zdravo”

- Klasa Prozor realizuje jednostavnu samostalnu aplikaciju sa GUI
- Klasa Prozor se izvodi iz klase Frame i sadrži:
  - konstruktor
  - metode: paint, handleEvent i main
- Konstruktor
  - poziva konstruktor klase Frame prosleđujući mu naslov
  - poziva metod setSize() klase Component koji određuje dimenzije prozora
  - poziva metod setVisible() klase Window koji prikaže prozor
- Metod handleEvent()
  - metod klase Component koji se automatski poziva kada se desi događaj unutar komponente
  - ako metod handleEvent() uspešno obradi događaj vraća se true
  - ako metod ne obradi događaj uspešno, vraća se false ,

    te se događaj prosleđuje roditeljskoj komponenti u hijerarhiji objekata
  - ovde handleEvent() metod detektuje zatvaranje prozora aplikacije kao WINDOW_DESTROY događaj

O primeru “Zdravo”

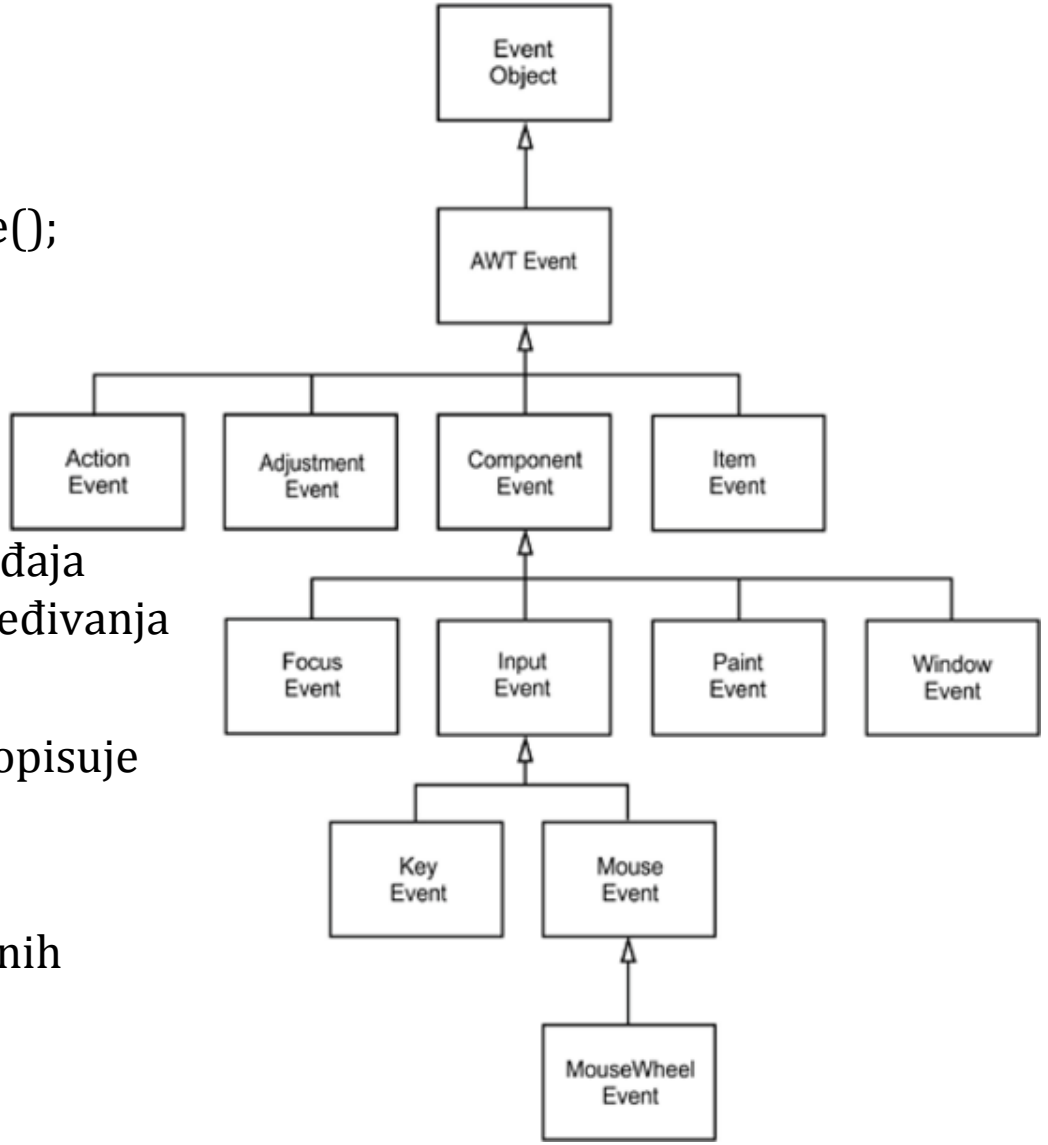
- Metod paint()
  - metod klase Component koji se automatski poziva iz izvršnog okruženja (iz AWT niti) da se iscta ta komponenta
  - metod se preklapa u izvedenim klasama da definiše kako će se iscrtati objekat (ovde prozor aplikacije)
  - poziva se za inicijalno iscrtavanje kao i prilikom pomeranja, promene veličine i pokrivanja prozora
  - kao parametar mu se prosleđuje objekat tipa Graphics
  - ovaj objekat se koristi da se ažurira prikaz komponente crtanjem u njegov podrazumevani kanvas
  - tačka (0,0) koordinatnog sistema grafičkog prikaza se poklapa sa gornjim levim uglom komponente
  - odsecajući region grafičkog prikaza je granični pravougaonik oko komponente
  - ovde paint() metod ispisuje tekst "Zdravo!“ u prozoru aplikacije, počevši od tačke (50,50)
- Metod main() kreira objekat tipa Prozor
  - startuje se AWT nit grafičkog okruženja
  - main po izvršenju tela čeka na završetak AWT niti

Obrada događaja korišćenjem osluškivača

- U Javi 1.0 obrada događaja je centralizovana
  - objekat komponente koja generiše događaj obrađuje događaj
  - centralni metod (handleEvent) obrade za sve događaje nameće obradu događaja u razgranatoj kontrolnoj strukturi - neobjektno
- Java 1.1 uvodi novi koncept događaja:
  - Događaje generišu izvori (*sources*) događaja
  - Događaje obrađuju objekti klasa osluškivača (*listeners*) događaja
    - Klase osluškivača obavezno implementiraju interfejs nekog osluškivača događaja
- Jedan ili više objekata osluškivača se može registrovati kod nekog izvora
  - registrovani osluškivači će biti obaveštavani od izvora o događajima pojedine vrste
- Metodi obrade događaja (rukovaoci - *handlers*)
  - "propisani" su interfejsom odgovarajućeg osluškivača
- Ovakav model obrade događaja se naziva "delegiranje" ili "prosleđivanje":
  - pravo obrade događaja se delegira svakom objektu koji implementira interfejs odgovarajućeg osluškivača

Klase događaja

- U java.util:
  - klasa EventObject
    - koren hijerarhije
    - metod Object getSource();
- U java.awt:
  - klasa AWTEvent
    - potklasa klase EventObject
    - natklasa svih AWT događaja koji koriste model prosleđivanja
    - metod int getID();
      - vraća ceo broj koji opisuje vrstu događaja
- U java.awt.event:
  - razne klase događaja izvedenih iz AWTEvent



Osluškivači događaja

- Program koji obrađuje događaje ima karakteristične delove koda
- U zaglavlju klase osluškivača mora se naznačiti
  - da klasa implemetira interfejs nekog osluškivača ili
  - da klasa proširuje neku klasu koja implementira interfejs osluškivača
  - na primer:

```
public class Obradjivac implements ActionListener {
```

- Registrovanje instance klase osluškivača događaja kod izvora
  - na primer:

```
komponentaIzvor.addActionListener( obradjivac );
```

- Implementacija metoda koje propisuje interfejs osluškivača
  - na primer:

```
public void actionPerformed(ActionEvent e) {/*...*/}
```

Primer delegirane obrade događaja

```
import java.awt.*; import java.awt.event.*;
public class Dogadjaj extends Frame implements ActionListener {
   private Button dugme;
   private int broj=0;
   public Dogadjaj(){
        super("Dogadjaj"); setSize(200,200);
        kreirajDugme();
        setVisible(true);
   }
   private void kreirajDugme(){
        dugme = new Button("Pritisni");
        add(dugme);
        dugme.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
        dugme.setLabel("Pritisnuto "+ ++broj +". put");
   }
   public static void main(String[] args){
        Dogadjaj d = new Dogadjaj();  }}
```

Primer delegirane obrade - izlaz

Klase adaptera

- Problem
  - u klasi konkretnog osluškivača se moraju definisati svi metodi interfejsa
  - ostaće prazni neki metodi u klasi konkretnog osluškivača
  - kod postaje glomazan, nečitak i težak za održavanje
- AWT definiše klase adaptera da reši gornji problem
  - za sve osluškivače sa više od jednog metoda postoji klasa adaptera
  - adapter implementira sve metode interfejsa osluškivača kao prazne
- Korišćenje klasa adaptera
  - iz odgovarajućeg adaptera se izvodi klasa željenog osluškivača
  - izvedena klasa treba da redefiniše samo željene obrade događaja
  - umesto implementacije interfejsa radi se proširenje adaptera
- Nedostatak
  - nije moguće istovremeno izvesti klasu iz adaptera i iz neke druge klase

Primer proširenja adaptera

```
import java.awt.*;
import java.awt.event.*;
public class DogadjajProzora extends Frame {
   class AdapterProzora extends WindowAdapter{
        public void windowClosing(WindowEvent we){
                System.exit(0);
        }
   }
   public DogadjajProzora() {
        super("Dogadjaj Prozora");
        setSize(280,80);
        addWindowListener(new AdapterProzora());
        setVisible(true);
   }
   public void paint(Graphics g) {
        g.drawString("Zdravo!",50,50);
   }
   public static void main(String args[]){
        DogadjajProzora p = new DogadjajProzora();}}
```

Isti primer – anonimna klasa

```
import java.awt.*;
import java.awt.event.*;
public class DogadjajProzora extends Frame {
    public DogadjajProzora() {
         super("Anonimni Adapter");
         setSize(280,80);
         addWindowListener(new WindowAdapter(){
                  public void windowClosing(WindowEvent we){
                           System.exit(0);
                  }
         });
         setVisible(true);
    }
    public void paint(Graphics g) {
         g.drawString("Zdravo!",50,50);
    }
    public static void main(String args[]){
         DogadjajProzora p = new DogadjajProzora();
    }}
```

Obrada događaja koji potiču od miša

- Postoje dva interfejsa za osluškivanje događaja koji potiču od miša
  - MouseListener: osluškivač miša i
  - MouseMotionListener: osluškivač kretanja miša
- Interfejs osluškivača miša predviđa 5 metoda:

```
public void mouseClicked (MouseEvent d); //pritisnuto i otpusteno dug.
public void mouseEntered (MouseEvent d); //kursor usao u polje komp.
public void mouseExited  (MouseEvent d); //kursor izasao iz polja kom.
public void mousePressed (MouseEvent d); //pritisnuto dugme
public void mouseReleased(MouseEvent d); //otpusteno dugme
```

  - ako se dugme otpusti na istoj poziciji gde je pritisnuto – mouseClicked
  - ako se dugme otpusti na različitoj poziciji – mouseReleased

- Interfejs osluškivača kretanja miša predviđa 2 metoda:

```
public void mouseMoved (MouseEvent d); //pomeren bez pritiskanja dug.
public void mouseDragged (MouseEvent d); //pomeren sa pritisnutim dug.
```

Klasa MouseEvent

- Klasa MouseEvent je iz paketa java.awt.event
  - u hijerarhiji klasa na sledećoj poziciji:

    java.awt.event.MouseEvent->java.awt.event.InputEvent->
    java.awt.event.ComponentEvent->java.awt.AWTEvent->
    java.util.EventObject->java.lang.Object
- Konstruktor:

```
public MouseEvent(Component source, int id, long when, int
   modifiers, int x,int y, int clickCount, boolean popupTrigger )
```

  - **source** : komponenta koja je izazvala događaj
  - **id** : tip događaja (npr. MOUSE_CLICKED, MOUSE_DRAGGED, ...)
  - **when** : *timestamp* trenutak kada se događaj desio
  - **modifiers** : modifikatori koji određuju da li je pritisnut <ALT>, <SHIFT>, <MOUSE_BUTTONx>
  - **x,y** : koordinate tačke gde se nalazio pointer pri događaju
  - **clickCount** : broj "klikova" kojima je izazvan događaj
  - **popupTrigger**: informacija da li je događaj izazvao pojavu *pop-up* menija

Primer osluškivača miša

```
import java.awt.*;
import java.awt.event.*;
public class OsluskivacMisa extends Frame implements MouseListener {
   private String t="Ceka se na dogadjaj misa...";
   public OsluskivacMisa(){
        super("Osluskivac misa"); setSize(300,100);
        addMouseListener(this);
        setVisible(true);
   }
   public void paint(Graphics g){
        g.drawString(t,50,50);
   }
   public void mouseClicked (MouseEvent d){t="Dog: clicked";repaint();}
   public void mouseEntered (MouseEvent d){t="Dog: entered";repaint();}
   public void mouseExited  (MouseEvent d){t="Dog: exited"; repaint();}
   public void mousePressed (MouseEvent d){t="Dog: pressed";repaint();}
   public void mouseReleased(MouseEvent d){t="Dog: released";repaint();}
   public static void main(String[] args){
        OsluskivacMisa d=new OsluskivacMisa();
   }
}
```

MouseWheelListener

- Primer korišćenja MouseWheelListener-a, tako da se pomeranjem točkića miša menja vrednost brojača koja se prikazuje na koordinatama 50,50 datog frejma.

```
import java.awt.*; import java.awt.event.*;
public class MouseWheelTest extends Frame {
    int brojac;
    public MouseWheelTest() {
        super(); brojac = 0;
        addWindowListener(new WindowAdapter(){
           public void windowClosing(WindowEvent we){ System.exit(0); }
        });
        MouseWheelListener listener =  new MouseWheelListener() {
            private static final int UP = 1;
            private static final int DOWN = -1;
            public void mouseWheelMoved(MouseWheelEvent e) {
              int count = e.getWheelRotation();
              int direction = (count <= 0)?UP:DOWN;
              brojac+=direction;
              repaint();
            }
        };
        addMouseWheelListener(listener);
    }
```

MouseWheelListener

```
    public void paint(Graphics g){
        g.drawString( ""+brojac, 50,50);
    }
    public static void main(String args[]) {
        Frame frame = new MouseWheelTest();
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Izlaz (pomeranje točkića 1 korak dole a onda dva gore):