



Interfejsat grafikë të shfrytëzuesit dhe programimi sipas ngjarjeve

Interfejsat grafikë të shfrytëzuesit dhe programimi sipas ngjarjeve

Objektivat:

- ⑥ Të mësohet si të zbatohet java korniza AWT/Swing për të disenjuar interfejsa grafikë të shfrytëzuesit.
- ⑥ Të shkruhen programe kontrollerët e të cilëve janë të distribuuar dhe të udhëhequr nga ngjarje.
- ⑥ Të mësohet të përdoret shabllon me disenj observuesi për të drejtuar kolaborimet ndërmjet komponenteve të një programi i cili zbaton GUI.

Sërish mbi Model-View-Controller

Interfejs grafik i shfrytëzuesit (Graphical User Interface, GUI): pamje hyrëse/dalëse e cila i lejon shfrytëzuesit të jep hyrje duke trusur butone, duke zgjedhur opzione menysh, duke rradhitur tekst etj., dhe i afishon shfrytëzuesit dalje në formë grafike.

Sërish mbi Model-View-Controller – Vazhdim

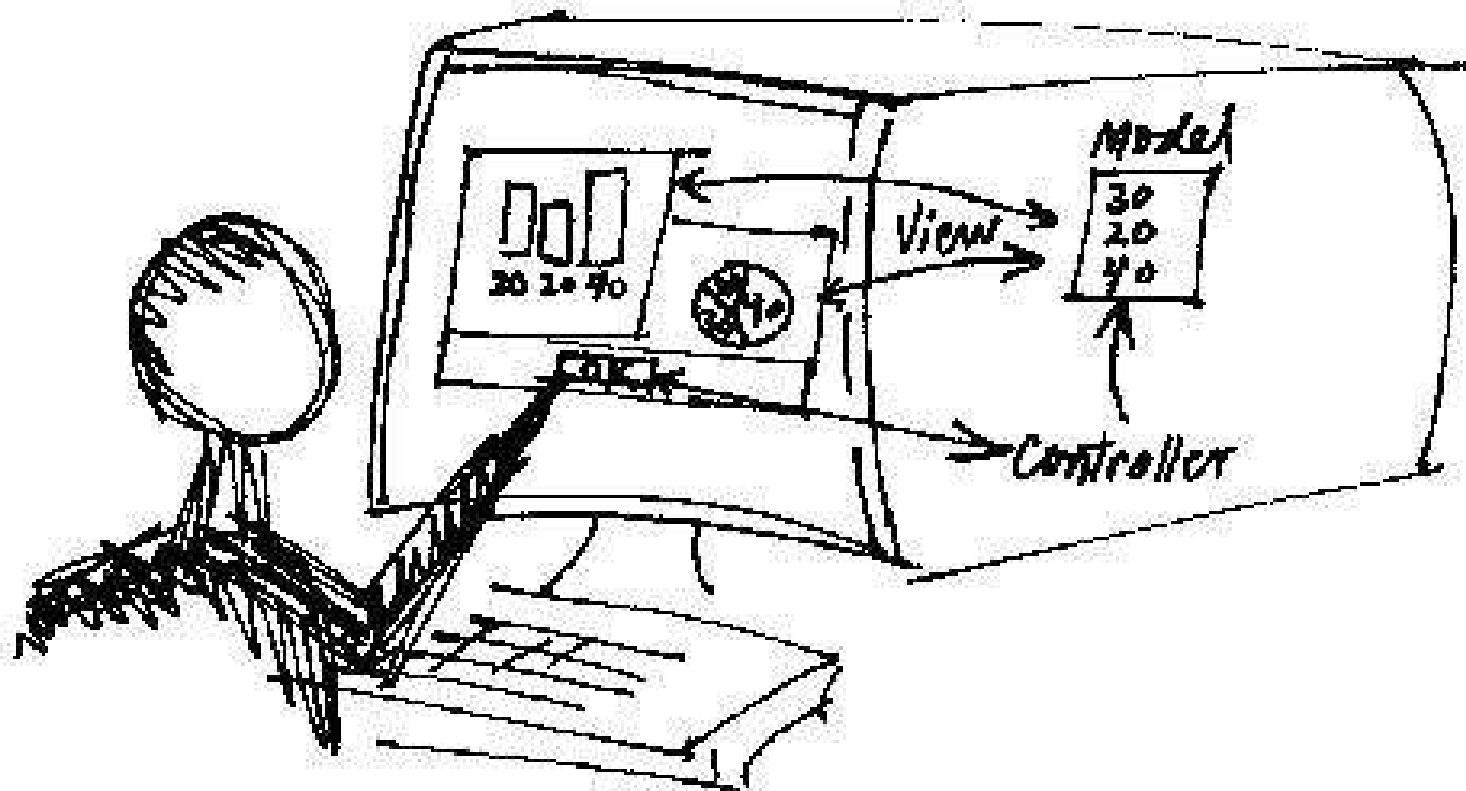


Figura 1. MVC arkitektura

Ngjarjet

Ngjarje (event): input i programit – mund të jetë një trusje butoni, rradhitje teksti, zgjedhje opcionesh të menysë etj. Komponenta grafike (butoni, fusha e tekstit ose opcioni i menysë) e cila përdoret për të shkaktuar ngjarjen quhet *burim i ngjarjes*.

Programimi i udhëhequr nga ngjarje (event-driven programming): stili i programimit i cili zbatohet për të pranuar ngjarje dhe reaguat në to, tipikisht duke shfrytëzuar një koleksion komponentesh kontrolluese, i cili kryen llogaritjen si reaksion në çdo formë ngjarjeje.

Ngjarjet – Vazhdim

Event hanlder (event listener) (përpunues ngjarjesh):

komponentë kontrollues e cila pranon dhe reagon në një ngjarje; një gjë e tillë quhet përpunim ngjarjeje.

Ngjarje aksion (action event): ngjarje e shkaktuar nga trusje butoni ose selektim menyje. Një ngjarje aksion përpunohet nga përpunues ngjarjesh i quajtur *action listener*.

AWT/Swing korniza: Java pakot `java.awt` dhe `javax.swing`, të cilat përmbajnë klasë që i ndihmojnë programuesit të shkruajë GUI.

Kierarkia e klasëve të AWT/Swing

Komponentë (component): termi në Java për një entitet (objekt grafik) i cili ka pozitë, madhësi dhe mund të ketë ngjarje të cilat ndodhin përbrenda tij.

Konteiner (container): komponentë e cila mund të përmbajë komponente tjera.

Panelë (panel): forma e zakonshme e konteinerit.

Kornizë (frame): dritare me title bar dhe meny. Janë me destinim „permanent“.

Dialog: dritare „e përkohshme“.

Kierarkia e klasëve të AWT/Swing – Vazhdim

Labelë (label): komponentë e cila afishon tekst të cilin shfrytëzuesi mund ta lexojë por nuk mund ta ndryshojë.

Komponentë tekst (text component): komponentë në të cilën shfrytëzuesi mund të rradhisë tekst; mund të jetë *text field (fushë teksti)*, në të cilën rradhitet një rresht tekst, ose *text area (hapësirë teksti)*, në të cilën mund të rradhiten më tepër rreshta.

Buton (button): komponentë e cila mund të truset, duke shkaktuar një ngjarje aksion.

Kierarkia e klasëve të AWT/Swing – Vazhdim



Listë (list): komponentë e cila afishon opzione të cilat mund të zgjedhen („selektohen“).

Meny (menu): komponentë e cila, kur të selektohet, afishon një varg *opcionesh të menysë (menu items)*, çdonjëri nga të cilët mund të selektohet, duke shkaktuar një ngjarje aksion.

Kierarkia e klasëve të AWT/Swing – Vazhdim



Figura 2. Kornizë me komponente themelore

Kierarkia e klasëve të AWT/Swing – Vazhdim

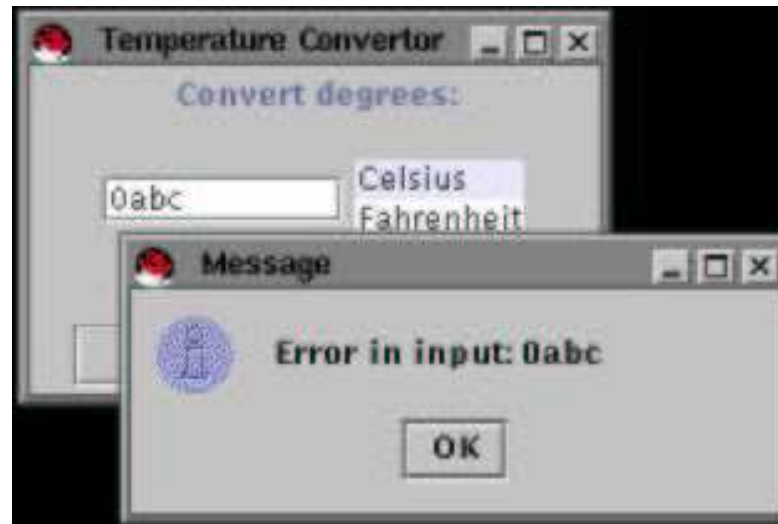


Figura 3. Dialog

Kierarkia e klasëve të AWT/Swing – Vazhdim



Figura 4. Kornizë me meny

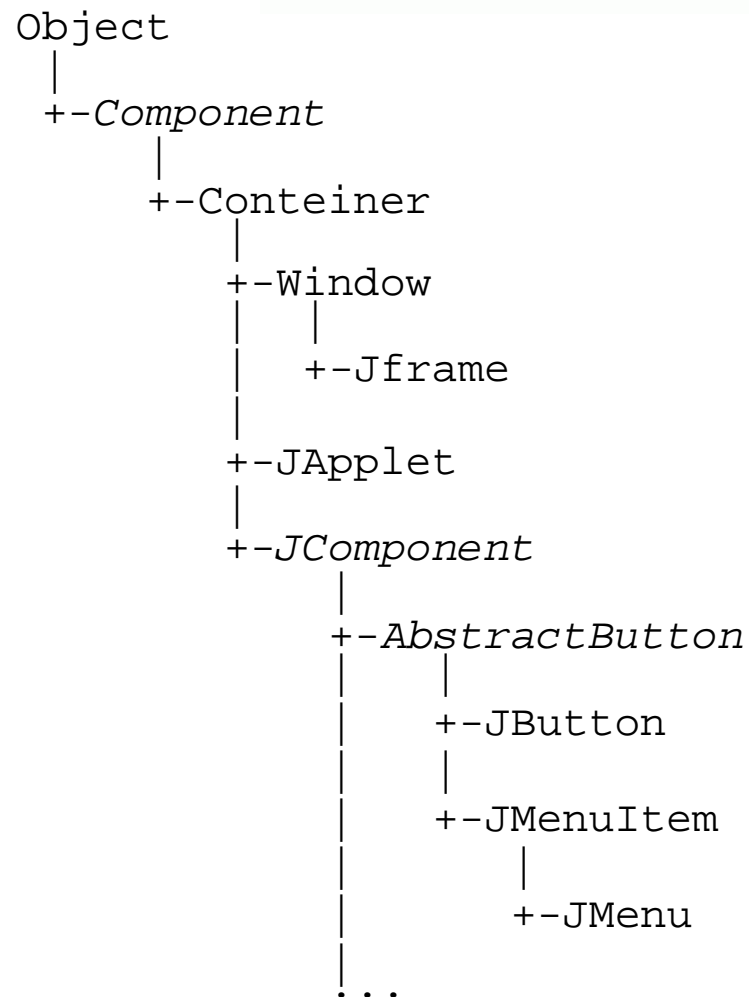
Kierarkia e klasëve të AWT/Swing – Vazhdim

Shtrirje (layout): mënyra në të cilën një koleksion komponentesh është aranzhuar për afishim në një GUI. Disa forma shtrirjesh janë:

- ⑥ *flow layout:* komponentet aranzhohen në renditje lineare.
- ⑥ *border layout:* komponentet u shoqërohen në mënyrë eksplicite regjioneve: „north“, „east“, „south“, „west“ ose „center“ të konteinerit.
- ⑥ *grid layout:* komponentet aranzhohen si elemente të madhësisë së njëjtë në rreshta dhe shtylla.

Pano përmbajtjeje (content pane): pjesa e një kornize ku komponentet futen për afishim.

Kierarkia e klasëve të AWT/Swing – Vazhdim



Kierarkia e klasëve të AWT/Swing – Vazhdim

```
...
|
+-JList
|
+-JMenuBar
|
+-JOptionPane
|
+-JPanel
|
+-JScrollPane
|
+-JTextComponent
|
|   +-JTextField
|   |
|   +-JTextArea
```

Dritare të thjeshta: labelat dhe butonët

```
import java.awt.*;
import javax.swing.*;
/** Kornizë me labelë dhe buton. */
public class Frame1 extends JFrame
{ /** Krijon kornizën. */
    public Frame1()
    { JLabel label = new JLabel("Truse këtë:");
      JButton button = new JButton("OK");
      Container c = this.getContentPane();
      c.setLayout(new FlowLayout());
      c.add(label);
      c.add(button);
      setTitle("Shembulli 1");
      setSize(200, 80);
      setVisible(true);
    }
}
```


Dritare të thjeshta: labelat dhe butonët – Vazhdim

```
public static void main(String[] args)
{ new Frame1(); }
}
```

Dritare të thjeshta: labelat dhe butonët – Vazhdim

Vizatimi në „xhamin“ e sipërm të kornizës (nuk preferohet).

```
public void paint(Graphics g)
{ g.setColor(Color.red);
  g.fillRect(0, 0, 100, 80);
}
```

Dritare të thjeshta: labelat dhe butonët – Vazhdim

```
public Frame2()  
{ Container c = getContentPane();  
  JLabel l = new JLabel("Truse Universitetin:");  
  ImageIcon i = new ImageIcon("uni-pr.gif");  
  JButton b = new JButton(i);  
  c.setBackground(Color.yellow);  
  l.setForeground(Color.red);  
  c.setLayout(new FlowLayout());  
  c.add(l);  
  c.add(b);  
  setTitle("Buton me ikonë");  
  pack();  
  setVisible(true);  
}
```

Përpunimi i një ngjarjeje

```
/** Emërton metodën që nevojitet nga një action listener. */
public interface ActionListener
{ /** Përpunon një ngjarje aksion.
    * @param e - informata mbi ngjarjen */
    public void actionPerformed(ActionEvent e);
}

public class C implements ActionListener
{ // ...
    public void actionPerformed(ActionEvent e)
    { // ... instruksionet të cilat përpunojnë ngjarjen aksion
    }
}
```

View si action listener

```
/** Modelon numëruesin. */
public class Counter
{ private int count;
  /** Inicializon numëruesin.
   * @param start - vlera fillestare e numëruesit */
  public Counter(int start)
  { count = start; }
  /** Azhuron numëruesin. */
  public void increment()
  { count++; }
  /** I qaset numëruesit.
   * @return vlera e numëruesit count */
  public int countOf()
  { return count; }
}
```

View si action listener – Vazhdim

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

/** Një kornizë labela e së cilës afishon
 *  sa herë është trusur butoni i saj. */
public class Frame2a extends JFrame implements ActionListener
{ private Counter count;
  private JLabel label = new JLabel("Numri = 0");
```

View si action listener – Vazhdim

```
/** Krijon kornizë me labelë dhe buton.  
 * @param c - objekti model, numërues */  
public Frame2a(Counter c)  
{ count = c;  
  Container cp = getContentPane();  
  cp.setLayout(new FlowLayout());  
  cp.add(label);  
  JButton button = new JButton("OK");  
  cp.add(button);  
  button.addActionListener(this);  
  setTitle("Korniza 2a");  
  setSize(200, 80);  
  setVisible(true);  
}
```

View si action listener – Vazhdim

```
/** Përpunon një ngjarje aksion -- trusje butoni.  
 * @param e - ngjarja aksion */  
public void actionPerformed(ActionEvent e)  
{ count.increment();  
  label.setText("Numri = " + count.countOf());  
}  
}
```


View si action listener – Vazhdim

```
/** Starton aplikacionin. */  
public class Test2a  
{ public static void main(String[] args)  
  { Counter model = new Counter(0);  
    Frame2a view = new Frame2a(model);  
  }  
}
```

Controller i veçantë

```
/** Specifikon një pamje e cila mund të azhurohet. */  
public interface UpdatableView  
{ /** Azhuron pamjen për të paraqitur gjendjen vijuese. */  
    public void update();  
}  
  
import java.awt.*;  
import javax.swing.*;  
/** Një kornizë labela e së cilës afishon  
    * sa herë është trusur butoni i saj. */  
public class Frame2b extends JFrame implements UpdatableView  
{ private Counter count;  
    private JLabel label = new JLabel("Numri = 0");
```

Controller i veçantë – Vazhdim

```
/** Krijon kornizë me labelë dhe buton.
 * @param c - objekti model, numërues */
public Frame2b(Counter c)
{ count = c;
  Container cp = getContentPane();
  cp.setLayout(new FlowLayout());
  cp.add(label);
  JButton button = new JButton("OK");
  button.addActionListener(new CountController(count, this));
  cp.add(button);
  setTitle("Korniza 2b");
  setSize(200, 80);
  setVisible(true);
}

public void update()
{ label.setText("Numri = " + count.countOf()); }
}
```

Controller i veçantë – Vazhdim

```
import java.awt.event.*;
/** Përpunon ngjarjet e trusjes së butonit. */
public class CountController implements ActionListener
{ private Counter model;
  private UpdatableView view;
  /** Konstrukton kontrolluesin
   * @param m - objekti model
   * @param v - objekti view */
  public CountController(Counter m, UpdatableView v)
  { model = m;
    view = v;
  }
  /** Përpunon një ngjarje aksion -- trusje butoni. */
  public void actionPerformed(ActionEvent e)
  { model.increment();
    view.update();
  }
}
```

Buton-kontrollues

```
import java.awt.event.*;
import javax.swing.*;
/** Definon një kontrollues butoni. */
public class CountButton extends JButton implements ActionListener
{ private UpdatableView view;
  private Counter model;
  /** Ndërton kontrolluesin
   * @param l - mbishkrimi në butonin
   * @param m - objekti model
   * @param v - objekti view */
  public CountButton(String l, Counter m, UpdatableView v)
  { super(l);
    view = v;
    model = m;
    addActionListener(this);
  }
}
```

Buton-kontrollues – Vazhdim

```
/** Përpunon një ngjarje aksion -- trusje butoni.  
 * @param e - ngjarja aksion */  
public void actionPerformed(ActionEvent e)  
{ model.increment();  
  view.update();  
}  
}
```

Buton-kontrollues – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Një kornizë labela e së cilës afishon
 *  sa herë është trusur butoni i saj. */
public class Frame2c extends JFrame implements UpdatableView
{ private Counter count;
  private JLabel label = new JLabel("Numri = 0");
```

Buton-kontrollues – Vazhdim

```
/** Krijon kornizë me labelë dhe buton.
 * @param c - objekti model, numërues */
public Frame2c(Counter c)
{ count = c;
  Container cp = getContentPane();
  cp.setLayout(new FlowLayout());
  cp.add(label);
  cp.add(new CountButton("OK", count, this));
  setTitle("Korniza 2c");
  setSize(200, 80);
  setVisible(true);
}

public void update()
{ label.setText("Numri = " + count.countOf()); }
}
```


Buton-kontrollues – Vazhdim

```
/** Starton aplikacionin. */  
public class Test2c  
{ public static void main(String[] args)  
  { Counter model = new Counter(0);  
    Frame2c view = new Frame2c(model);  
  }  
}
```

Buton-kontrollues – Vazhdim

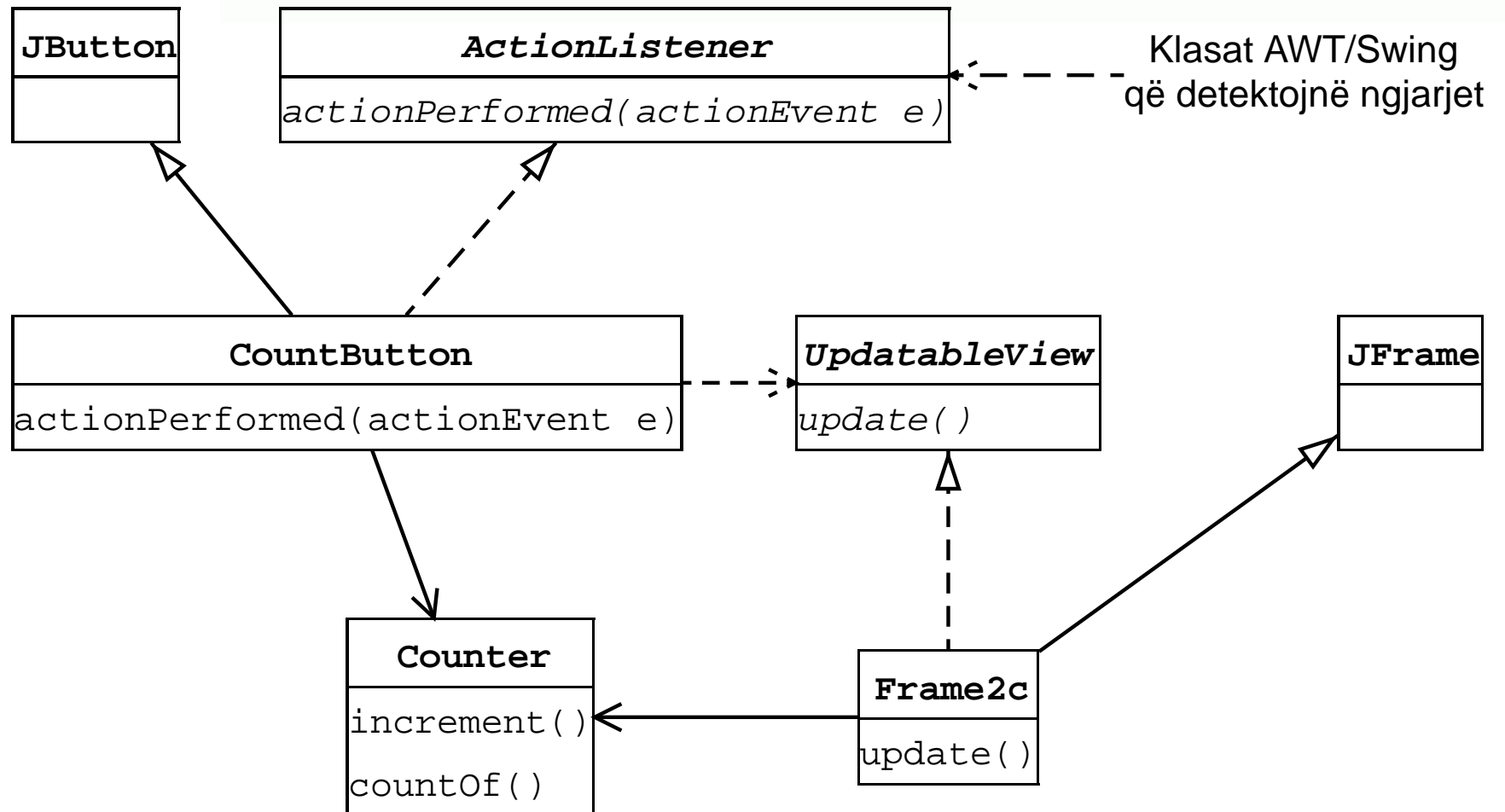


Figura 5. Arkitektura e Test2c

Buton-kontrollues – Vazhdim

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

/** Definon një kontrollues i cili terminon një aplikacion. */
public class ExitButton extends JButton implements ActionListener
{ /** Ndërtton kontrolluesin.
    * @param l - mbishkrimi në butonin */
    public ExitButton(String l)
    { super(l);
      addActionListener(this);
    }
    /** Përpunon një ngjarje aksion -- trusje butoni.
    * @param e - ngjarja aksion */
    public void actionPerformed(ActionEvent e)
    { System.exit(0); }
}
```

Buton-kontrollues – Vazhdim

```
import java.awt.event.*;
/** Definon një kontrollues dritareje. */
public class ExitController extends WindowAdapter
{ /** Përpunon një ngjarje -- trusje butoni "X" në dritare.
    * @param e - ngjarja aksion */
    public void windowClosing(WindowEvent e)
    { System.exit(0); }
}
```

Buton-kontrollues – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Një kornizë labela e së cilës afishon
 *  sa herë është trusur butoni i saj. */
public class Frame3 extends Frame2c
{ /** Krijon kornizë me labelë dhe 2 butona.
 *   @param c - objekti model, numërues */
  public Frame3(Counter c)
  { super(c);
    Container cp = getContentPane();
    cp.add(new ExitButton("Exit"));
    addWindowListener(new ExitController());
    setTitle("Korniza 3");
    setSize(250, 80);
    setVisible(true);
  }
}
```

Panelat dhe kornizat



Figura 6. Shtrirja BorderLayout

Panelat dhe kornizat – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Një kornizë me vizatim, labelë e 2 butona. */
public class Frame4 extends JFrame implements UpdatableView
{ private Counter count;
  private JLabel label = new JLabel("Numri = 0");
  private JPanel drawing;
```

Panelat dhe kornizat – Vazhdim

```
/** Krijon kornizë me vizatim, labelë e 2 butona.
 * @param c - objekti model, numërues
 * @param panel - panela e cila afishon vizatimin */
public Frame4(Counter c, JPanel panel)
{
    count = c;
    drawing = panel;
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    JPanel p1 = new JPanel();
    p1.add(label);
    cp.add(p1, BorderLayout.NORTH);
    cp.add(drawing, BorderLayout.CENTER);
    JPanel p2 = new JPanel(new FlowLayout());
    p2.add(new CountButton("Numëro", count, this));
    p2.add(new ExitButton("Mjaft"));
    cp.add(p2, BorderLayout.SOUTH);
}
```


Panelat dhe kornizat – Vazhdim



```
setTitle("Korniza 4");  
setSize(200, 180);  
setVisible(true);  
}  
  
public void update()  
{ label.setText("Numri = " + count.countOf());  
  drawing.repaint();  
}  
}
```

Panelat dhe kornizat – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Panelë e cila afishon një vizatim. */
public class Drawing extends JPanel
{ private Counter count;
  /** Krijon panelën
   * @param m - objekti model, numërues */
  public Drawing(Counter model)
  { count = model;
    setSize(200, 80);
  }
}
```

Panelat dhe kornizat – Vazhdim

```
/** Vizaton vizatimin.
 * @param g - penda grafike */
public void paintComponent(Graphics g)
{ g.setColor(Color.white);
  g.fillRect(0, 0, 150, 80);
  g.setColor(Color.red);
  for ( int i = 0; i < count.countOf(); i++ )
  { g.fillOval((i % 6) * 25, (i / 6) * 25, 20, 20); }
}
```

Panelat dhe kornizat – Vazhdim

```
/** Starton aplikacionin. */  
public class Test4  
{ public static void main(String[] args)  
  { Counter model = new Counter(0);  
    Drawing drawing = new Drawing(model);  
    Frame4 view = new Frame4(model, drawing);  
  }  
}
```

Animimi në panelë

```
/** Modelon një top i cili pulson duke ndryshuar madhësinë. */  
public class ThrobbingBall  
{ private boolean isCurrentlyLarge;  
  /** Konstrukton topin */  
  public ThrobbingBall()  
  { isCurrentlyLarge = true; }  
  /** Kthen gendjen vijuese të topit */  
  public boolean isLarge()  
  { return isCurrentlyLarge; }  
  /** Bën topin që të pulsojë */  
  public void throb()  
  { isCurrentlyLarge = !isCurrentlyLarge; }  
}
```

Animimi në panelë – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Afishon topin pulsues. */
public class ThrobPanel extends JPanel
{ private int panelSize;
  private int location;
  private int ballSize;
  private Color ballColor = Color.red;
  private ThrobbingBall ball;
```

Animimi në panelë – Vazhdim

```
/** Ndërton panelën.
 * @param size - madhësia e panelës
 * @param b - topi */
public ThrobPanel(int size, ThrobblingBall b)
{
    panelSize = size;
    location = panelSize / 2;
    ballSize = panelSize / 3;
    ball = b;
    setSize(panelSize, panelSize);
}
/** Kthen ngjyrën vijuese të topit. */
public Color getColor()
{
    return ballColor;
}
```

Animimi në panelë – Vazhdim

```
/** Vë ngjyrën vijuese të topit.  
 * @param newColor - ngjyra e re */  
public void setColor(Color newColor)  
{ ballColor = newColor; }  
/** Vizaton topin.  
 * @param g - penda grafike */  
public void paintComponent(Graphics g)  
{ g.setColor(Color.white);  
  g.fillRect(0, 0, panelSize, panelSize);  
  g.setColor(ballColor);  
  if ( ball.isLarge() )  
  { g.fillOval(location, location, ballSize, ballSize); }  
  else  
  { g.fillOval(location, location, ballSize/2, ballSize/2); }  
}  
}
```


Animimi në panelë – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Afishon panelën e topit dhe butonin për ngjyrë. */
public class ThrobFrame extends JFrame
{ /** Ndërton kornizën.
    * @param size - gjerësia e kornizës
    * @param p - panela e cila afishon topin
    * @param b - butoni për ndryshim ngjyre */
    public ThrobFrame(int size, ThrobPanel p, ColorButton b)
    { Container cp = getContentPane();
      cp.setLayout(new BorderLayout());
      cp.add(p, BorderLayout.CENTER);
      cp.add(b, BorderLayout.SOUTH);
      setTitle("Pulsuesi");
      setSize(size, size + 40);
      setVisible(true);
    }
}
```

Animimi në panelë – Vazhdim

```
/** Ekzekuton animimin e topit pulsues. */
public class ThrobController
{ private ThrobPanel writer;
  private ThrobbingBall ball;
  private int time;
  /** Inicializon kontrolluesin.
   * @param w - panela që kontrollohet
   * @param b - topi që kontrollohet
   * @param delayTime - koha ndërmjet rivizatimit të animimit */
  public ThrobController(ThrobPanel w, ThrobbingBall b,
                        int delayTime)
  { writer = w;
    ball = b;
    time = delayTime;
  }
```

Animimi në panelë – Vazhdim

```
/** Ekzekuton animimin. */
public void run()
{ while ( true )
  { ball.throb();
    writer.repaint();
    delay();
  }
}

private void delay()
{ try { Thread.sleep(time); }
  catch (InterruptedException e) { }
}
```

Animimi në panelë – Vazhdim

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/** Definon një kontrollues butoni. */
public class ColorButton extends JButton implements ActionListener
{ private ThrobPanel view;
  /** Ndërton kontrolluesin.
   * @param v - objekti view */
  public ColorButton(ThrobPanel v)
  { super("Ngjyra");
    view = v;
    addActionListener(this);
  }
}
```

Animimi në panelë – Vazhdim

```
/** Përpunon një klik.  
 * @param e - ngjarja aksion */  
public void actionPerformed(ActionEvent e)  
{ Color c = view.getColor();  
  if ( c == Color.red )  
  { view.setColor(Color.blue); }  
  else { view.setColor(Color.red); }  
}  
}
```

Animimi në panelë – Vazhdim

```
/** Vë së bashku objektet e animimit. */  
public class ThrobTest  
{  
    public static void main(String[] args)  
    {  
        int frameSize = 180;  
        int pauseTime = 200;  
        ThrobbingBall b = new ThrobbingBall();  
        ThrobPanel p = new ThrobPanel(frameSize, b);  
        ThrobFrame f =  
            new ThrobFrame(frameSize, p, new ColorButton(p));  
        new ThrobController(p, b, pauseTime).run();  
    }  
}
```

Animimi në panelë – Vazhdim

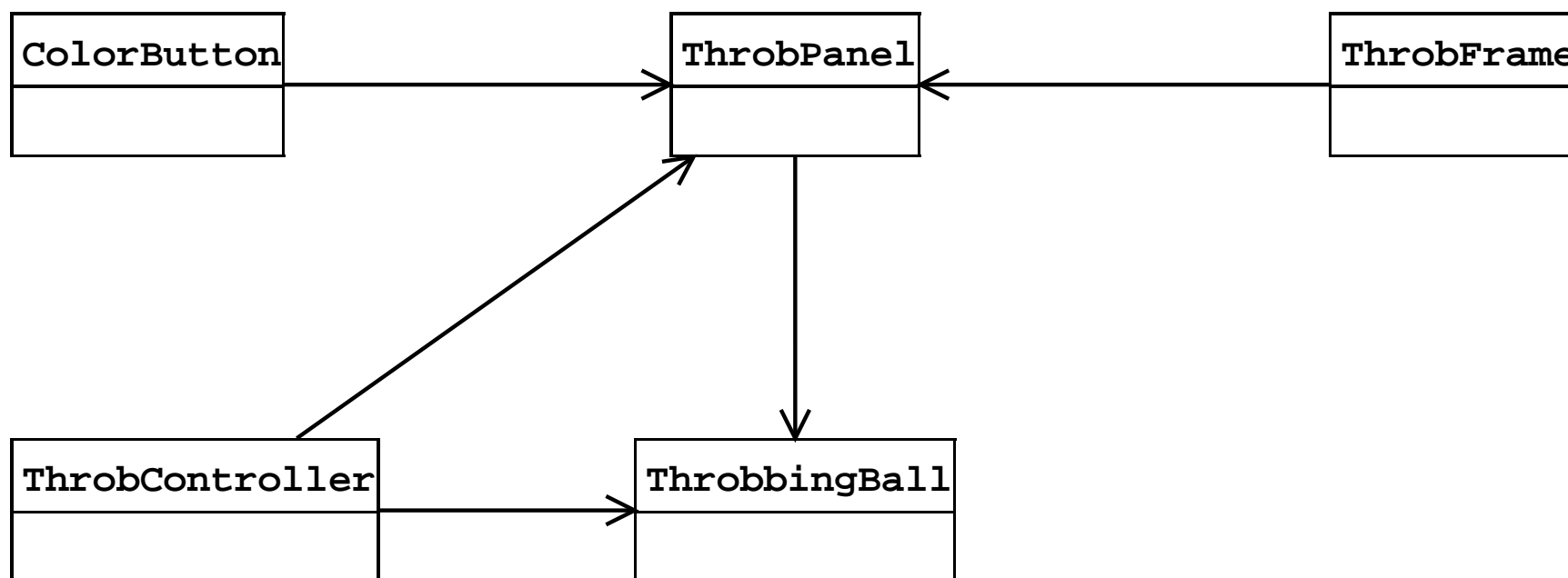


Figura 7. Arkitektura e animimit të topit pulsues

Shtrierja grid layout

```
/** Definon një copëzë të lojës së rebusit rrëshqitës. */
public class PuzzlePiece
{ private int faceValue;
  /** Krijon copëzën.
   * @param value - vlera e cila paraqitet në copëzën */
  public PuzzlePiece(int value)
  { faceValue = value; }
  /** Kthen vlerën e copëzës. */
  public int valueOf()
  { return faceValue; }
}
```


Shtrierja grid layout – Vazhdim

```
/** Modelon një rebus rrëshqitës. */  
public class SlidePuzzleBoard  
{ private int size;  
  private PuzzlePiece[][] board;  
  private int emptyRow;  
  private int emptyCol;
```

Shtrierja grid layout – Vazhdim

```
/** Konstrukton rebusin inicial me copëza
 * në renditje të anasjelltë.
 * @param s - madhësia e rebusit (s x s) */
public SlidePuzzleBoard(int s)
{
    size = s;
    board = new PuzzlePiece[size][size];
    for ( int num = 1; num != size * size; num++ )
    {
        PuzzlePiece p = new PuzzlePiece(num);
        int row = num / size;
        int col = num % size;
        board[size - 1 - row][size - 1 - col] = p;
    }
    emptyRow = size - 1;
    emptyCol = size - 1;
}
```

Shtrierja grid layout – Vazhdim

```
/** Kthen gjendjen vijuese të rebusit.  
 * @return matricë me adresa të copëzave */  
public PuzzlePiece[][] contents()  
{ PuzzlePiece[][] answer = new PuzzlePiece[size][size];  
  for ( int i = 0; i != size; i++ )  
  { for ( int j = 0; j != size; j++ )  
    { answer[i][j] = board[i][j]; }  
  }  
  return answer;  
}
```

Shtrierja grid layout – Vazhdim

```
/** Lëviz një copëz në hapësirën e lirë nëse është e mundur.  
 * @param w - vlera e copëzës që do të lëvizet  
 * @return suksesin e lëvizjes */  
public boolean move(int w)  
{ int NOT_FOUND = -1;  
  int row = NOT_FOUND;  
  int col = NOT_FOUND;  
  if ( found(w, emptyRow - 1, emptyCol) )  
  { row = emptyRow - 1;  
    col = emptyCol;  
  }  
  else if ( found(w, emptyRow + 1, emptyCol) )  
  { row = emptyRow + 1;  
    col = emptyCol;  
  }  
}
```

Shtrierja grid layout – Vazhdim

```
else if ( found(w, emptyRow, emptyCol - 1) )
{
    row = emptyRow;
    col = emptyCol - 1;
}
else if ( found(w, emptyRow, emptyCol + 1) )
{
    row = emptyRow;
    col = emptyCol + 1;
}
if ( row != NOT_FOUND )
{
    board[emptyRow][emptyCol] = board[row][col];
    emptyRow = row;
    emptyCol = col;
    board[emptyRow][emptyCol] = null;
}
return row != NOT_FOUND;
}
```

Shtrierja grid layout – Vazhdim

```
private boolean found(int v, int row, int col)
{
    boolean answer = false;
    if ( row >= 0 && row < size && col >= 0 && col < size )
    {
        answer = ( board[row][col].valueOf() == v );
    }
    return answer;
}
```

Shtrierja grid layout – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Afishon një rebus rrëshqitës. */
public class PuzzleFrame extends JFrame
{ private SlidePuzzleBoard board;
  private int size;
  private int buttonSize = 60;
  private PuzzleButton[][] button;
  /** Ndërton komponentën view.
   * @param boardSize - gjerësia dhe lartësia e rebusit
   * @param b - modeli, tabelë rebusi rrëshqitës */
  public PuzzleFrame(int boardSize, SlidePuzzleBoard b)
  { size = boardSize;
    board = b;
    button = new PuzzleButton[size][size];
    Container cp = getContentPane();
    cp.setLayout(new GridLayout(size, size));
```

Shtrierja grid layout – Vazhdim

```
for ( int i = 0; i != size; i++ )
{ for ( int j = 0; j != size; j++ )
    { button[i][j] = new PuzzleButton(board, this);
      cp.add(button[i][j]);
    }
}
update();
addWindowListener(new ExitController());
setTitle("Korniza e rebusit");
setSize(size * buttonSize + 10, size * buttonSize + 20);
setVisible(true);
}
```


Shtrierja grid layout – Vazhdim

```
/** Konsulton modelin dhe rivizaton secilin buton. */
public void update()
{
    PuzzlePiece[][] r = board.contents();
    for ( int i = 0; i != size; i++ )
    {
        for ( int j = 0; j != size; j++ )
        {
            if ( r[i][j] != null )
            {
                button[i][j].setBackground(Color.white);
                button[i][j].setText(" " + r[i][j].valueOf());
            }
            else { button[i][j].setBackground(Color.black);
                button[i][j].setText( " " );
            }
        }
    }
}
```

Shtrierja grid layout – Vazhdim

```
/** Starton rebusin rrëshqitës. */  
public class Puzzle  
{ public static void main(String[] args)  
  { int size = 4;  
    SlidePuzzleBoard board = new SlidePuzzleBoard(size);  
    PuzzleFrame frame = new PuzzleFrame(size, board);  
  }  
}
```

Shtrierja grid layout – Vazhdim

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

/** Implementon kontrollues butoni për lojë rebusi. */
public class PuzzleButton extends JButton implements ActionListener
{ private SlidePuzzleBoard puzzle;
  private PuzzleFrame view;
  /** Ndërton butonin.
   * @param p - objekti model i rebusit
   * @param v - komponenta view e rebusit */
  public PuzzleButton(SlidePuzzleBoard p, PuzzleFrame v)
  { super("");
    puzzle = p;
    view = v;
    addActionListener(this);
  }
}
```

Shtrierja grid layout – Vazhdim

```
/** Përpunon një lëvizje të rebusit. */  
public void actionPerformed(ActionEvent e)  
{ String s = getText();  
  if ( !s.equals("") )  
  { boolean ok = puzzle.move(new Integer(s).intValue());  
    if ( ok ) { view.update(); }  
  }  
}
```

Listat rimbështjellëse (scrolling lists)

```
String[] listLabels = new String[numItems];  
JList items = new JList(listLabels);  
JScrollPane sp = new JScrollPane(items)
```

```
Container cp = getContentPane();  
cp.add(sp);
```

```
for ( int i = 0; i != listLabels.length; i++ )  
{ listLabels[i] = "Numëruesi " + i + " është 0"; }
```

Listat rimbështjellëse – Vazhdim

```
/** Afishon një varg numëruesish si scrolling list. */
public class TestList
{ public static void main(String[] a)
  { int numCounters = 8;
    Counter2[] counters = new Counter2[numCounters];
    for ( int i = 0; i != numCounters; i++ )
      { counters[i] = new Counter2(0, i); }
    new ListFrame(counters);
  }
}
```

Listat rimbështjellëse – Vazhdim

```
/** Numërues i cili identifikon vetveten me toString metodë. */  
public class Counter2 extends Counter  
{ private int index;  
  public Counter2(int start, int i)  
  { super(start);  
    index = i;  
  }  
  public String toString()  
  { return "Numëruesi " + index + " është " + countOf(); }  
}
```

Listat rimbështjellëse – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Afishon një scrolling list. */
public class ListFrame extends JFrame
{ private Counter2[] counters;
  private JList items;
```


Listat rimbështjellëse – Vazhdim

```
/** Gjeneron kornizën së bashku me listën.  
 * @param model - objekti model që do të afishohet në listën */  
public ListFrame(Counter2[] model)  
{ counters = model;  
  items = new JList(counters);  
  JScrollPane sp = new JScrollPane(items);  
  Container cp = getContentPane();  
  cp.setLayout(new GridLayout(2,1));  
  cp.add(sp);  
  JPanel p = new JPanel(new GridLayout(2,1));  
  p.add(new ListButton("Rrite", counters, this));  
  p.add(new ExitButton("Mjaft"));  
  cp.add(p);  
  update();  
  setTitle("Shembull liste");  
  setSize(200,200);  
  setVisible(true);  
}
```

Listat rimbështjellëse – Vazhdim

```
/** Kthen hyrjen e listës të selektuar nga shfrytëzuesi.  
 * @return indeksi i elementit, ose -1 nëse nuk ka */  
public int getSelection()  
{ return items.getSelectedIndex(); }  
/** Azhuron paraqitjen e listës. */  
public void update()  
{ items.clearSelection(); }  
}
```

Listat rimbështjellëse – Vazhdim

```
import java.awt.event.*;
import javax.swing.*;
/** Implementon buton i cili ndryshon një scrolling list. */
public class ListButton extends JButton implements ActionListener
{ private Counter2[] counters;
  private ListFrame view;
  /** Konstrukton kontrolluesin. */
  public ListButton(String label, Counter2[] c, ListFrame v)
  { super(label);
    counters = c;
    view = v;
    addActionListener(this);
  }
}
```

Listat rimbështjellëse – Vazhdim

```
/** Përpunon ngjarjen e klikimit. */  
public void actionPerformed(ActionEvent evt)  
{ int choice = view.getSelection();  
  if ( choice != -1 )  
  { counters[choice].increment();  
    view.update();  
  }  
}  
}
```

Fushat e tekstit

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

/** Kornizë abstrakte e cila afishon konverzime temperaturash. */
public abstract class AbTempFrame extends JFrame
{ private String START_TEXT = "0";
  private String BLANKS = "          ";
  private JTextField inputText = new JTextField(START_TEXT, 8);
  private JLabel answer = new JLabel(BLANKS);
  private String[] choices = {"Fahrenheit", "Celsius"};
  private JList scales = new JList(choices);
```

Fushat e tekstit – Vazhdim

```
/** Konstrukton kornizën. */  
public AbsTempFrame()  
{ ComputeTempButton computeController =  
    new ComputeTempButton("Llogarit", this);  
    Container cp = getContentPane();  
    cp.setLayout(new GridLayout(4, 1));  
    JPanel p1 = new JPanel(new FlowLayout());  
    p1.add(new JLabel("Konverto shkallë:"));  
    cp.add(p1);  
    JPanel p2 = new JPanel(new FlowLayout());  
    p2.add(inputText);  
    p2.add(scales);  
    cp.add(p2);  
    JPanel p3 = new JPanel(new FlowLayout());  
    p3.add(answer);  
    cp.add(p3);
```

Fushat e tekstit – Vazhdim

```
JPanel p4 = new JPanel(new FlowLayout());
p4.add(computeController);
p4.add(new ResetButton("Reseto", this));
p4.add(new ExitButton("Mjaft"));
cp.add(p4);
resetFields();
setSize(240, 180);
setTitle("Konvertuesi i temperaturës");
setVisible(true);
}

/** Kthen hyrjet e selektuara dhe të rradhitura nga shfrytëzuesi.
 * @return (1) stringu i rradhitur; (2) hyrja e selektuar */
public String[] getInputs()
{ String[] input = new String[2];
  input[0] = inputText.getText();
  input[1] = choices[scales.getSelectedIndex()];
  return input;
}
```

Fushat e tekstit – Vazhdim

```
/** Ndryshon labelën.  
 * @param s - stringu që reseton labelën */  
public void displayAnswer(String s)  
{ answer.setText(s); }  
/** Afishon mesazh gabimi.  
 * @param s - mesazhi */  
public abstract void displayError(String s);  
/** Reseton fushat e pamjes. */  
public void resetFields()  
{ inputText.setText(START_TEXT);  
  answer.setText(BLANKS);  
  scales.setSelectedIndex(0);  
}  
}
```


Fushat e tekstit – Vazhdim

```
/** Ndërton pamje të kompletuar për konvertorin e temperaturës. */  
public class TempFrame extends AbsTempFrame  
{  
    public TempFrame()  
    { super(); }  
  
    public void displayError(String s)  
    { displayAnswer("Gabim: " + s); }  
}
```

Fushat e tekstit – Vazhdim

```
import javax.swing.*;
import java.awt.event.*;
/** Implementon buton që konverton temperatura. */
public class ComputeTempButton extends JButton
    implements ActionListener
{
    private TemperatureConvertor calc = new TemperatureConvertor();
    private AbsTempFrame view;
    /** Konstrukton butonin.
     * @param label - mbishkrimi në butonin
     * @param v - adresa e objektit view */
    public ComputeTempButton(String label, AbsTempFrame v)
    {
        super(label);
        view = v;
        addActionListener(this);
    }
}
```

Fushat e tekstit – Vazhdim

```
/** Llogarit temperaturën. */
public void actionPerformed(ActionEvent evt)
{ try
  { String[] s = view.getInputs();
    double temp = new Double(s[0].trim()).doubleValue();
    String answer = "ështëë ";
    if ( s[1].equals("Fahrenheit") )
    { answer = answer + calc.fahrenheitToCelsius(temp)
      + " Celsius"; }
    else { answer = answer + calc.celsiusToFahrenheit(temp)
      + " Fahrenheit"; }
    view.displayAnswer(answer);
  }
  catch(RuntimeException e)
  { view.displayError(e.getMessage()); }
}
```

Fushat e tekstit – Vazhdim

```
import javax.swing.*;
import java.awt.event.*;
/** Reseton fushat e GUI. */
public class ResetButton extends JButton implements ActionListener
{ private AbsTempFrame view;
  /** Konstrukton butonin.
   * @param v - objekti view */
  public ResetButton(String label, AbsTempFrame v)
  { super(label);
    view = v;
    addActionListener(this);
  }
  /** Përpunon klikun. */
  public void actionPerformed(ActionEvent evt)
  { view.resetFields(); }
}
```

Fushat e tekstit – Vazhdim

```
/** Konverton vlera temperaturash. */
public class TemperatureConvertor
{ /** Konverton shkallë Fahrenheit në Celsius.
    * @param f -- shkallët në Fahrenheit
    * @return ekuivalenti në shkallë Celsius */
    public double fahrenheitToCelsius(double f)
    { return (5.0/9.0) * (f - 32); }
    /** Konverton shkallë Celsius në Fahrenheit.
    * @param c - shkallët në Celsius
    * @return ekuivalenti në shkallë Fahrenheit */
    public double celsiusToFahrenheit(double c)
    { return (9.0/5.0) * c + 32; }
}
```

Fushat e tekstit – Vazhdim

```
public class Temp
{ public static void main(String[] args)
  { new TempFrame(); }
}
```

Fushat e tekstit – Vazhdim

Klasat AWT/Swing që detektojnë ngjarjet

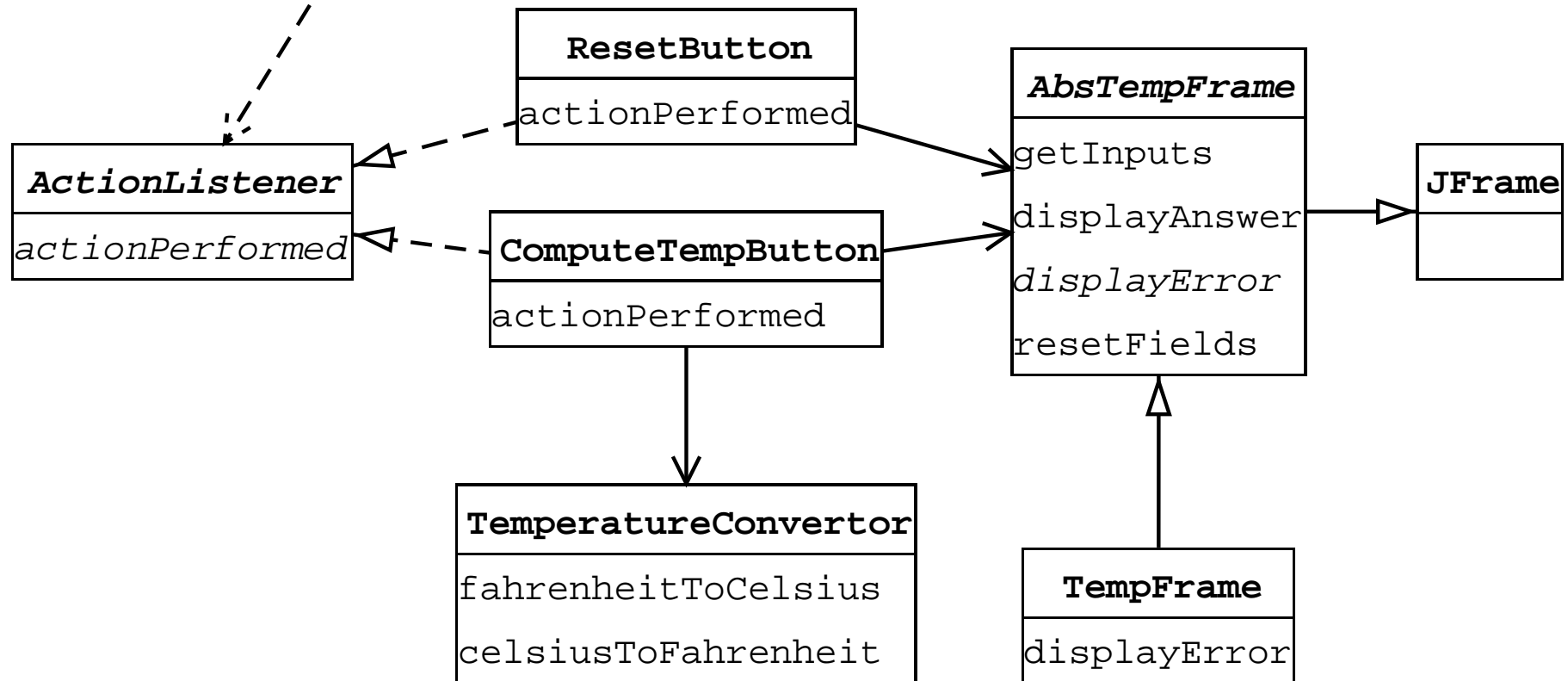


Figura 8. Arkitektura e konverorit të temperaturës

Raportimi i gabimeve: Dialogjet

```
import javax.swing.*;

/** Ndërton pamje të kompletuar për konvertorin e temperaturës
 * me raportim gabimi me përmes dialogu. */
public class TempFrame2 extends AbsTempFrame
{
    public TempFrame2()
    { super(); }

    public void displayError(String s)
    { JOptionPane.showMessageDialog(this, "Gabim: " + s); }
}

public class Temp2
{ public static void main(String[] args)
  { new TempFrame2(); }
}
```


Hapësirat e tekstit dhe menytë: Një editor teksti

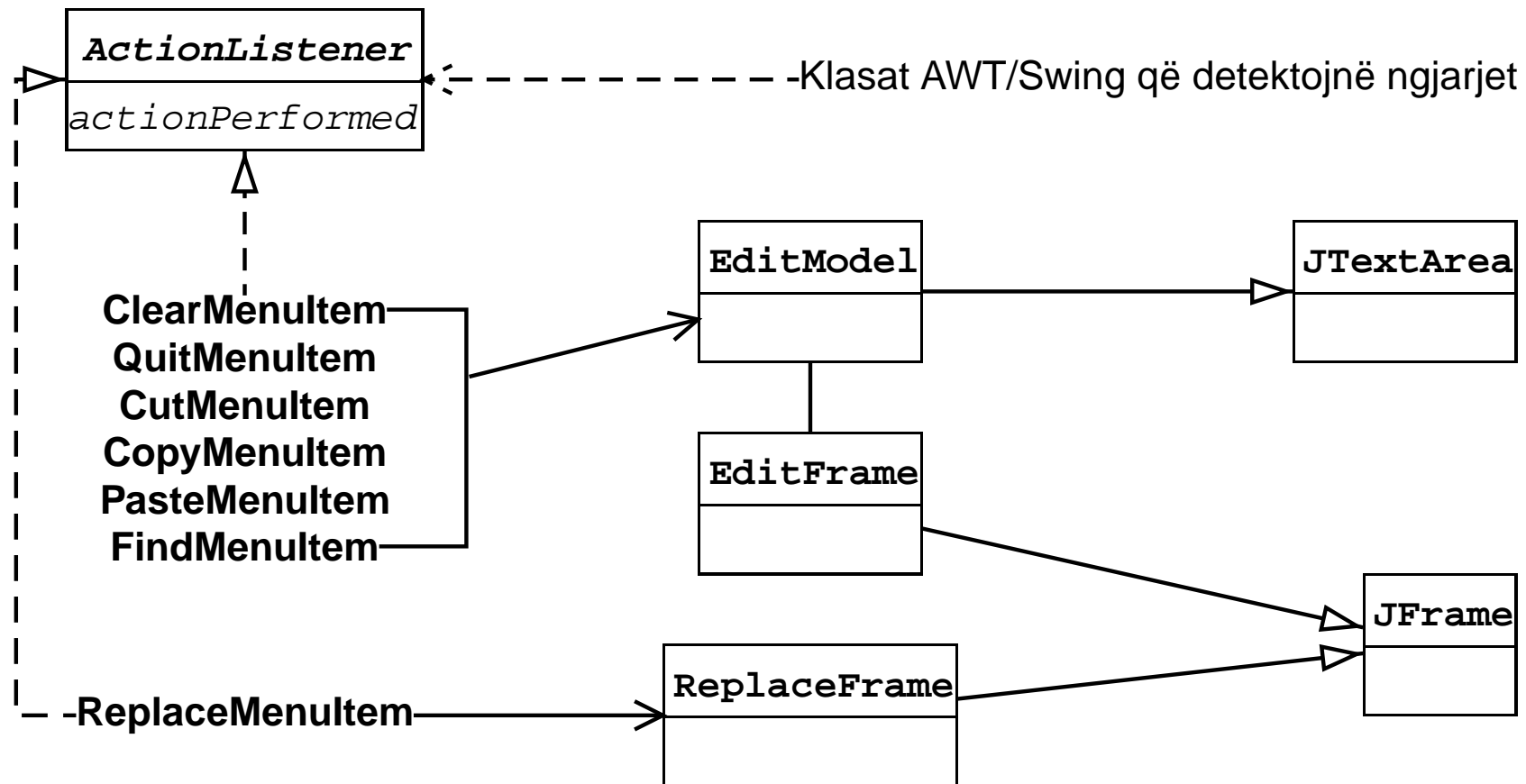


Figura 9. Diagrami i pjesërishëm i klasave të editorit të tekstit

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Afishon editor teksti me meny. */
public class EditFrame extends JFrame
{ private EditModel buffer = new EditModel("", 15, 50);
  public EditFrame()
  { ReplaceFrame secondFrame = new ReplaceFrame(buffer);
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    JMenuBar mbar = new JMenuBar();
    JMenu file = new JMenu("Fajl");
    file.add(new ClearMenuItem("I ri", buffer));
    file.add(new QuitMenuItem("Dalja"));
    mbar.add(file);
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
JMenu edit = new JMenu("Edito");
edit.add(new CutMenuItem("Prej", buffer));
edit.add(new CopyMenuItem("Kopjo", buffer));
edit.add(new PasteMenuItem("Ngjrit", buffer));
edit.addSeparator();
JMenu search = new JMenu("Kërko");
search.add(new FindMenuItem("Gjej", buffer));
search.add(new ReplaceMenuItem("Zëvendëso", secondFrame));
edit.add(search);
mbar.add(edit);
setJMenuBar(mbar);
JScrollPane sp = new JScrollPane(buffer);
cp.add(sp, BorderLayout.CENTER);
setTitle("Editor teksti");
pack();
setVisible(true);
}
}
```

Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

```
import java.awt.*;
import javax.swing.*;
/** Modelon hapësirë teksti. */
public class EditModel extends JTextArea
{ /** Ndërton hapësirën e tekstit.
    * @param initialText - teksti fillestar për hapësirën e tekstit
    * @param rows - numri i rreshtave
    * @param cols - numri i shtyllave */
    public EditModel(String initialText, int rows, int cols)
    { super(initialText, rows, cols);
      setLineWrap(true);
      setFont(new Font("Courier", Font.PLAIN, 12));
    }
    /** Reseton hapësirën e tekstit në boshe. */
    public void clear()
    { setText(""); }
}
```

Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

```
/** Alokon një string në fushën e tekstit
 * nga pozita fillestare. */
private int find(String s, int position)
{ String text = getText();
  int index = text.indexOf(s, position);
  if ( index != -1 )
  { setCaretPosition(index + s.length());
    moveCaretPosition(index);
  }
  return index;
}

/** Alokon një string në fushën e tekstit nga fillimi.
 * @param s - stringu që alokohet
 * @return pozita ku është gjetur s; -1 në të kundërtën */
public int findFromStart(String s)
{ return find(s, 0); }
```

Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

```
/** Alokon një string në fushën e tekstit nga pozita e karetit.  
 * @param s - stringu që alokohet  
 * @return pozita ku është gjetur s; -1 në të kundërtën */  
public int findFromCaret(String s)  
{ return find(s, getCaretPosition()); }  
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import javax.swing.*;
import java.awt.event.*;
/** Terminon editorin e tekstit. */
public class QuitMenuItem extends JMenuItem
    implements ActionListener
{
    public QuitMenuItem(String label)
    {
        super(label);
        addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }
}
```

Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

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

/** Definon hyrje menyje gjenerike për editorin e tekstit. */
public abstract class EditorMenuItem extends JMenuItem
    implements ActionListener
{
    private EditModel buffer;

    public EditorMenuItem(String label, EditModel model)
    {
        super(label);
        buffer = model;
        addActionListener(this);
    }

    /** Kthen adresën e modelit i cili manipulohet. */
    public EditModel getBuffer()
    {
        return buffer;
    }

    public abstract void actionPerformed(ActionEvent e);
}
```


Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import java.awt.event.*;
/** Pastron hapësirën e tekstit. */
public class ClearMenuItem extends EditorMenuItem
{ public ClearMenuItem(String label, EditModel model)
  { super(label, model); }

  public void actionPerformed(ActionEvent e)
  { getBuffer().clear(); }
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import java.awt.event.*;
/** Pret tekstin e selektuar nga hapësira e tekstit. */
public class CutMenuItem extends EditorMenuItem
{ public CutMenuItem(String label, EditModel model)
  { super(label, model); }

  public void actionPerformed(ActionEvent e)
  { getBuffer().cut(); }
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import java.awt.event.*;
/** Kopjon në clipboard tekstin e selektuar. */
public class CopyMenuItem extends EditorMenuItem
{ public CopyMenuItem(String label, EditModel model)
  { super(label, model); }

  public void actionPerformed(ActionEvent e)
  { getBuffer().copy(); }
}
```

Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

```
import java.awt.event.*;
/** Bart përmbajtjen e clipboard në fushën e tekstit. */
public class PasteMenuItem extends EditorMenuItem
{ public PasteMenuItem(String label, EditModel model)
  { super(label, model); }

  public void actionPerformed(ActionEvent e)
  { getBuffer().paste(); }
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import javax.swing.*;
import java.awt.event.*;
/** Gjeneron dialog për të gjetur një string. */
public class FindMenuItem extends EditorMenuItem
{ public FindMenuItem(String label, EditModel model)
  { super(label, model); }
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
public void actionPerformed(ActionEvent e)
{ String s = JOptionPane.showInputDialog(this,
    "Stringu që kërkohet:");
  if ( s != null )
  { int index = getBuffer().findFromCaret(s);
    if ( index == -1 )
    { int response = JOptionPane.showConfirmDialog(this,
        "Stringu nuk u gjet. Të fillohet kërkimi nga fillimi?");
      if ( response == JOptionPane.YES_OPTION )
      { index = getBuffer().findFromStart(s);
        if ( index == -1 )
        { JOptionPane.showMessageDialog(this,
            "Stringu nuk u gjet"); }
      }
    }
  }
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
import javax.swing.*;
import java.awt.event.*;
/** Nxjerr kornizën ndihmëse për zëvendësim stringu. */
public class ReplaceMenuItem extends JMenuItem
    implements ActionListener
{
    private ReplaceFrame view;
    public ReplaceMenuItem(String label, ReplaceFrame v)
    {
        super(label);
        view = v;
        addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        view.setVisible(true);
    }
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

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

/** Afishon kornizë ndihmëse për zëvendësim stringu. */
public class ReplaceFrame extends JFrame implements ActionListener
{ private EditModel model;
  private JButton replace = new JButton("Zëvendëso");
  private JButton close = new JButton("Mbylle");
  private JTextField findText = new JTextField("", 20);
  private JTextField replaceText = new JTextField("", 20);
```


Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

```
public ReplaceFrame(EditModel m)
{
    model = m;
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    JPanel p1 = new JPanel(new GridLayout(2, 1));
    JPanel p11 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    p11.add(new JLabel("Nga kareti, zëvendëso "));
    p11.add(findText);
    p1.add(p11);
    JPanel p12 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    p12.add(new JLabel("me "));
    p12.add(replaceText);
    p1.add(p12);
    cp.add(p1, BorderLayout.CENTER);
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
JPanel p2 = new JPanel(new FlowLayout());  
p2.add(replace);  
p2.add(close);  
cp.add(p2, BorderLayout.SOUTH);  
replace.addActionListener(this);  
close.addActionListener(this);  
setTitle("Korniza për zëvendësim");  
pack();  
setVisible(false);  
}
```

Hapësirat e tekstit dhe menyte: Një editor teksti – Vazhdim

```
/** Përpunon të gjitha shypjet e butonave në këtë kornizë.  
 * @param e - ngjarja aksion */  
public void actionPerformed(ActionEvent e)  
{ if ( e.getSource() == close )  
  { setVisible(false); }  
  else if ( e.getSource() == replace )  
    { String find = findText.getText();  
      int location = model.findFromCaret(find);  
      if ( location == -1 )  
        { JOptionPane.showMessageDialog(this,  
          "Stringu " + find + " nuk u gjet");  
        }  
      else { model.replaceRange(replaceText.getText(),  
        location, location + find.length());  
      }  
    }  
  }  
}
```

Hapësirat e tekstit dhe menytë: Një editor teksti – Vazhdim

```
public class TextEditor
{ public static void main(String[] args)
  { new EditFrame(); }
}
```

Programimi me observues i udhëhequr nga ngjarje

```
import java.util.*;
/** Modelon një numërues që mund të observohet. */
public class Counter3 extends Observable
{ private int count; // the count
  /** Inicializon numëruesin
   * @param start - vlera fillestare */
  public Counter3(int start)
  { count = start; }
  /** Rrit numeruesin dhe sinjalizon të gjithë observuesit */
  public void increment()
  { count = count + 1;
    setChanged();
    notifyObservers();
  }
  /** Kthen vlerën e count */
  public int countOf()
  { return count; }
}
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

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

public class Count3Button extends JButton implements ActionListener
{ private Counter3 model;
  public Count3Button(String label, Counter3 c)
  { super(label);
    model = c;
    addActionListener(this);
  }
  public void actionPerformed(ActionEvent evt)
  { model.increment(); }
}
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

```
import java.awt.*; import javax.swing.*; import java.util.*;
/** Kornizë me labelë, butona dhe panelë. */
public class Frame4a extends JFrame implements Observer
{ private Counter3 count;
  private JLabel lab = new JLabel("count = 0");
  /** Krijon panelën
   * @param c - objekti model, numëruesi
   * @param drawing - panela që afishon një vizatim */
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

```
public Frame4a(Counter3 c, JPanel drawing)
{
    count = c;
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    JPanel p1 = new JPanel(new FlowLayout());
    p1.add(lab);
    cp.add(p1, BorderLayout.NORTH);
    cp.add(drawing, BorderLayout.CENTER);
    JPanel p2 = new JPanel(new FlowLayout());
    p2.add(new Count3Button("Numëro", count));
    p2.add(new ExitButton("Mjaft"));
    cp.add(p2, BorderLayout.SOUTH);
    setTitle("Korniza 4a");
    setSize(200,150);
    setVisible(true);
}
```


Programimi me observues i udhëhequr nga ngjarje – Vazhdim

```
/** Azhuron labelën e cila afishon numrin */  
public void update(Observable model, Object o)  
{ lab.setText("count = " + count.countOf()); }  
}
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
/** Krijon panelë që afishon vizatim */
public class Panel4a extends JPanel implements Observer
{ private Counter3 count;
  public Panel4a(Counter3 model)
  { count = model;
    setSize(200, 80);
  }
  /** Rivizaton panelën */
  public void update(Observable model, Object o)
  { repaint(); }
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

```
public void paintComponent(Graphics g)
{
    g.setColor(Color.white);
    g.fillRect(0, 0, 150, 80);
    g.setColor(Color.red);
    for ( int i = 0; i < count.countOf(); i = i + 1 )
    {
        g.fillOval((i % 6) * 25, (i / 6) * 25, 20, 20);
    }
}
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

```
/** Starton aplikacionin */  
public class Test4a  
{ public static void main(String[] args)  
  { Counter3 model = new Counter3(0);  
    Panel4a panel = new Panel4a(model);  
    Frame4a frame = new Frame4a(model, panel);  
    model.addObserver(panel);  
    model.addObserver(frame);  
  }  
}
```

Programimi me observues i udhëhequr nga ngjarje – Vazhdim

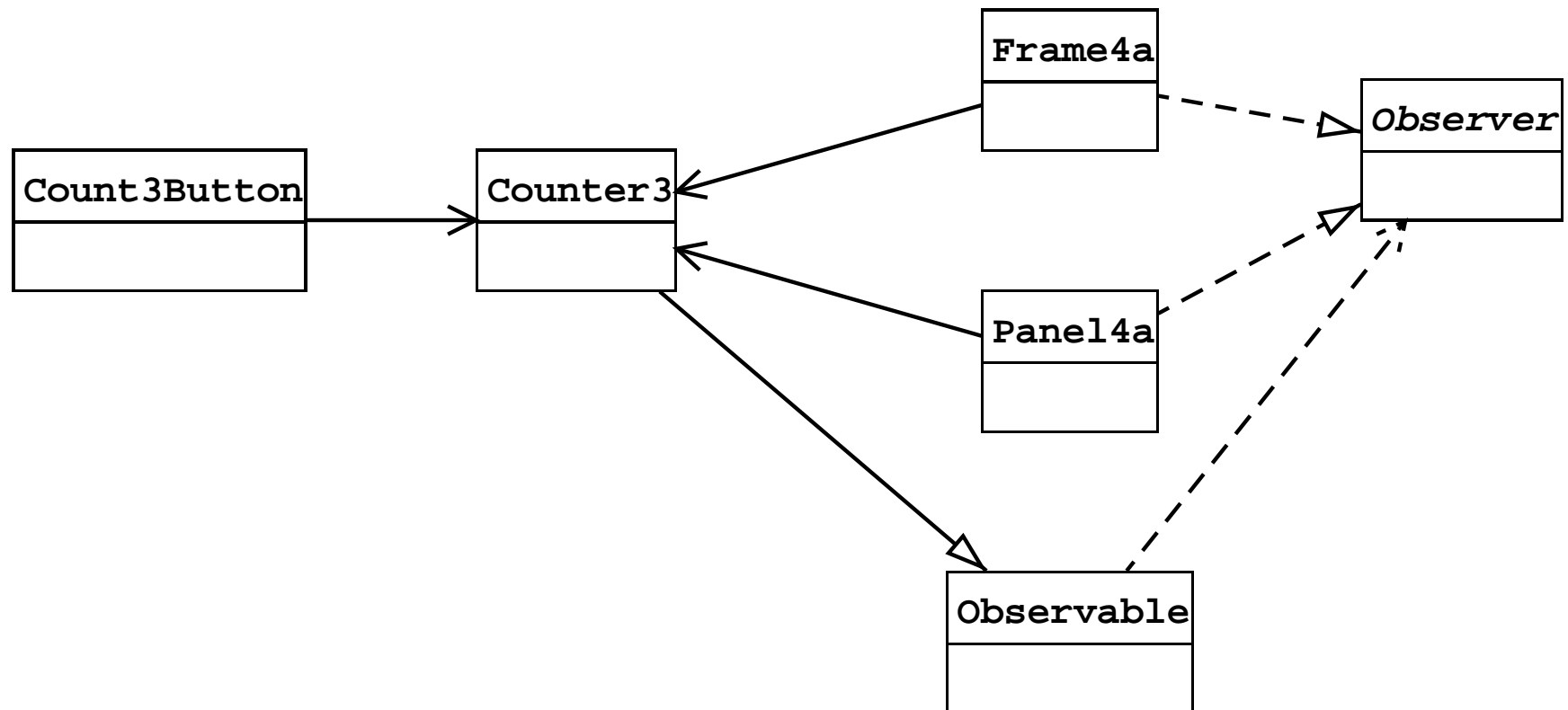


Figura 10. MVC arkitektura me observuesë dhe view të shumfishtë