(varianta pdf)

SwRTc – supliment proiect

Interfete grafice Swing si applet-uri Java

GUI.1. Descrierea laboratorului

In aceasta lucrare de laborator vor fi acoperite urmatoarele probleme:

- **crearea interfetelor grafice Java folosind Java Swing**
- **tratarea evenimentelor (interactivitatea) in interfetele grafice Swing**
- **crearea si instalarea *applet-urilor* (miniaplicatiilor) Java**

GUI.2. Introducere in interfete grafice Swing

GUI.2.1. Elementele unei aplicatii grafice Swing

Programul `ElementeAplicatieSwing` ilustreaza **elementele unei aplicatii grafice Swing** (**script pentru compilarea si executia lui**):

1. Importul pachetelor Swing si AWT necesare

```
import javax.swing.*;          // Numele actual al pachetului Swing
import java.awt.*;             // Numele pachetului AWT (necesar uneori)
import java.awt.event.*;       // Numele pachetului pentru tratarea
                               // evenimentelor, pentru interactivitate
public class ElementeAplicatieSwing {
  public static void main(String[] args) {
    // codul metodei principale
  }
}
```

2. Optional: stabilirea aspectului (Java, Windows, CDE/Motif)

```
try {
    UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName());
} catch (Exception e) { }
```

3. Stabilirea containerului de nivel maxim (fereasta principala - `JFrame`, fereastra secundara - `JDialog`, applet - `JApplet`)

```
JFrame frame = new JFrame("Elementele unei aplicatii Swing");
```

4. Obtinerea panoului de continut (*content pane*) intern cadrului (containerul in care vor fi plasate componentele ferestrei)

```
Container container = frame.getContentPane(); //
```

5. Optional: Stabilirea modului de asezare (*layout-ului*) a panoului (care implicit e `FlowLayout`)

```
container.setLayout(new BorderLayout());
```

6. Crearea si configurarea componentelor grafice (controalelor)

In cazul programului `ElementeAplicatieSwing`:

- crearea unei etichete:

```
final String textEticheta = "Numarul de actionari ale butonului: ";
final JLabel eticheta = new JLabel(textEticheta + "0    ");
```

- crearea unui buton, si atasarea unei combinatii de taste echivalente, [Alt] + I:

```
JButton buton = new JButton("Sunt un buton Swing!");
buton.setMnemonic(KeyEvent.VK_I);
```

- crearea si configurarea unui panou (componenta care grupeaza spatial alte componente):

```
JPanel panou = new JPanel();

// Stabilirea spatiului gol care inconjoara continutul panoului
panou.setBorder(BorderFactory.createEmptyBorder(
                                30,    // sus
                                30,    // in stanga
                                10,    // jos
                                30));  // in dreapta
// Stabilirea modului de asezare (layout-ului) a componentelor
panou.setLayout(new GridLayout(0, 1));

// Adaugarea componentelor in panou
panou.add(buton);
panou.add(eticheta);
```

7. Adaugarea in fereastra principala a componentelor grafice (controalelor)

```
container.add(panou, BorderLayout.CENTER);
```

8. Crearea codului pentru tratarea evenimentelor (interactivitatii)

Atasarea unui obiect al unei clase anonime care implementeaza interfata `ActionListener`, a carui metoda **`actionPerformed()` trateaza actionarea butonului:**

```
buton.addActionListener(new ActionListener() {
  int numActionari = 0;

  public void actionPerformed(ActionEvent e) {
    numActionari++;
    eticheta.setText(textEticheta + numActionari);
  }
});
```

Stabilirea iesirii din program la inchiderea ferestrei, prin **atasarea unui obiect al unei clase anonime care extinde clasa `ActionListener`**, a carui metoda **`windowClosing()` trateaza inchiderea ferestrei:**

```
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```

Alternativa pentru stabilirea iesirii din program la inchiderea ferestrei, incepand cu versiunea 1.3 a Java 2 SE:

```
// Alternativa, incepand cu Java 2, versiunea 1.3:
//      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

9. Stabilirea dimensiunii ferestrei

Stabilirea dimensiunii ferestrei in functie de cea a componentelor:

```
frame.pack();
```

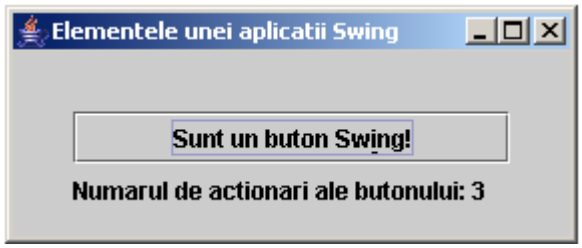
Impunerea dimensiunii ferestrei:

```
// Alternativa pentru cazul ca se doreste impunerea unei dimensiuni
//      frame.setSize(400,    // latime
//                    300);   // inaltime
```

10. Prezentarea ferestrei pe ecran

```
frame.setVisible(true);
```

Fereastra obtinuta prin executia programului **ElementeAplicatieSwing.java**:



Iata si codul unui program, **TestFrame**, care ilustreaza **crearea unei ferestre**, fara componente grafice, **a carei inchidere duce la iesirea din program** (**script pentru compilarea si executia lui**).

```
 1   // 1. Importul pachetelor Swing si AWT necesare
 2   import javax.swing.*;
 3
 4   public class TestFrame {
 5     public static void main(String[] args) {
 6
 7       // 2. Stabilirea containerului de nivel maxim
 8       JFrame frame = new JFrame(
 9                        "Fereastra cu iesire din program la inchidere");
10
11       // 3. Stabilirea dimensiunii ferestrei
12       frame.setSize(400,    // latime
13                     300);   // inaltime
14
15       // 4. Stabilirea iesirii din program la inchiderea ferestrei
16       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17
18       // 5. Prezentarea ferestrei pe ecran
19       frame.setVisible(true);
20     }
21   }
```

In lipsa codului din linia 16 (ca in programul **TestFrame2** - **script pentru compilarea si executia lui**) **inchiderea ferestrei nu conduce la iesirea din program**, ceea ce poate fi neplacut pentru utilizator.

Desigur, **exista si posibilitatea de a crea coduri mai simple**, de exemplu prin **compactarea etapelor 4 si 7**:

```
frame.getContentPane().add(panou, BorderLayout.CENTER);
```

GUI.2.2. Modalitati de a crea containerul de nivel maxim

Exista urmatoarele modalitati de a crea containere de nivel maxim:

1. Includerea unui obiect de tip `Frame` (fereastra principala in programele Java de sine statatoare)

Programul **IncludereJFrame** ilustreaza crearea unei ferestre principale prin **includerea unui obiect** de tip **`JFrame`**, fereastra in care **sunt asezate 5 butoane, folosind asezarea relativ la margini** - **`BorderLayout`** (**script pentru compilarea si executia lui**).

```
 1  import java.awt.*;
 2  import javax.swing.*;
 3  /**
 4   * Demonstreaza includerea unui obiect JFrame pentru a crea o fereastra pe ecran.
 5   */
 6  public class IncludereJFrame {
 7      public static void main(String[] args) {
 8        // Crearea obiectului cadru, cu titlu specificat
 9        JFrame cadru = new JFrame("Demo includere JFrame si asezare BorderLayout");
10
11        // Obtinerea panoului de continut intern cadrului (container de componente)
12        Container container = cadru.getContentPane();
13
14        // Asezarea componentelor in panou (la 10 pixeli de marginea panoului)
15        container.setLayout(new BorderLayout(10, 10));
16
17        // Adaugarea a 5 butoane la panoul cadrului (ferestrei)
18        container.add(new JButton("Est (Dreapta)"), BorderLayout.EAST);
19        container.add(new JButton("Sud (Jos)"), BorderLayout.SOUTH);
20        container.add(new JButton("Vest (Stanga)"), BorderLayout.WEST);
21        container.add(new JButton("Nord (Sus)"), BorderLayout.NORTH);
22        container.add(new JButton("Centru"), BorderLayout.CENTER);
23
24        // Din jdk1.3, pentru terminarea programului la inchiderea ferestrei
25        // cadru.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26
27        // Stabilirea dimensiunii ferestrei
28        // - impunand dimensiunile ferestrei (nepotrivita la BorderLayout):
29        //      cadru.setSize(100, 100);
30        // - compactand componentele adaugate:
31        cadru.pack();
32
33        // Stabilirea vizibilitatii ferestrei (Atentie: implicit e false!)
34        cadru.setVisible(true);
35      }
36  }
```

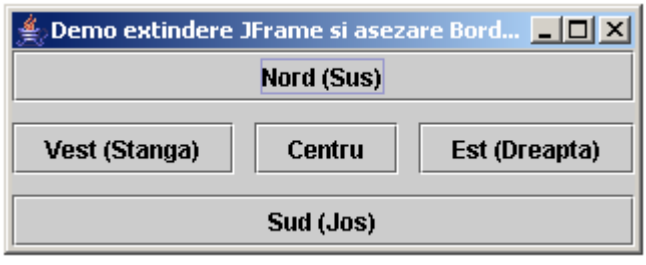
Fereastra obtinuta prin executia programului **`IncludereJFrame.java`**:

2. Extinderea clasei Frame (fereastra principala in programele Java de sine statatoare)

Programul **`ExtensieJFrame`** ilustreaza crearea unei ferestre principale prin **extinderea clasei `JFrame`**, si **asezarea relativ la margini** - **`BorderLayout`** (**script pentru compilarea si executia lui**).

```
 1  import java.awt.*;
 2  import javax.swing.*;
 3  /**
 4   * Demonstreaza extinderea JFrame pentru a crea o fereastra pe ecran.
 5   */
 6  public class ExtensieJFrame extends JFrame {
 7      public ExtensieJFrame() {
 8        // Obtinerea panoului de continut intern cadrului (container de componente)
 9        Container container = getContentPane();
10
11        // Asezarea componentelor in panou (la 10 pixeli de marginea panoului)
12        container.setLayout(new BorderLayout(10, 10));
13        // Adaugarea a 5 butoane la panoul cadrului (ferestrei)
14        container.add(new JButton("Est (Dreapta)"), BorderLayout.EAST);
15        container.add(new JButton("Sud (Jos)"), BorderLayout.SOUTH);
16        container.add(new JButton("Vest (Stanga)"), BorderLayout.WEST);
17        container.add(new JButton("Nord (Sus)"), BorderLayout.NORTH);
18        container.add(new JButton("Centru"), BorderLayout.CENTER);
19      }
20
21      public static void main(String[] args) {
22        // Crearea obiectului cadru
23        ExtensieJFrame cadru = new ExtensieJFrame();
24
25        // Adaugarea titlului ferestrei
26        cadru.setTitle("Demo extindere JFrame si asezare BorderLayout");
27
28        // Compactarea componentelor
29        cadru.pack();
30        // Stabilirea vizibilitatii ferestrei (Atentie: implicit e false!)
31        cadru.setVisible(true);
32      }
33  }
```

Fereastra obtinuta prin executia programului **`ExtensieJFrame.java`**:



3. Extinderea clasei JApplet (miniaplicatie (applet) Java)

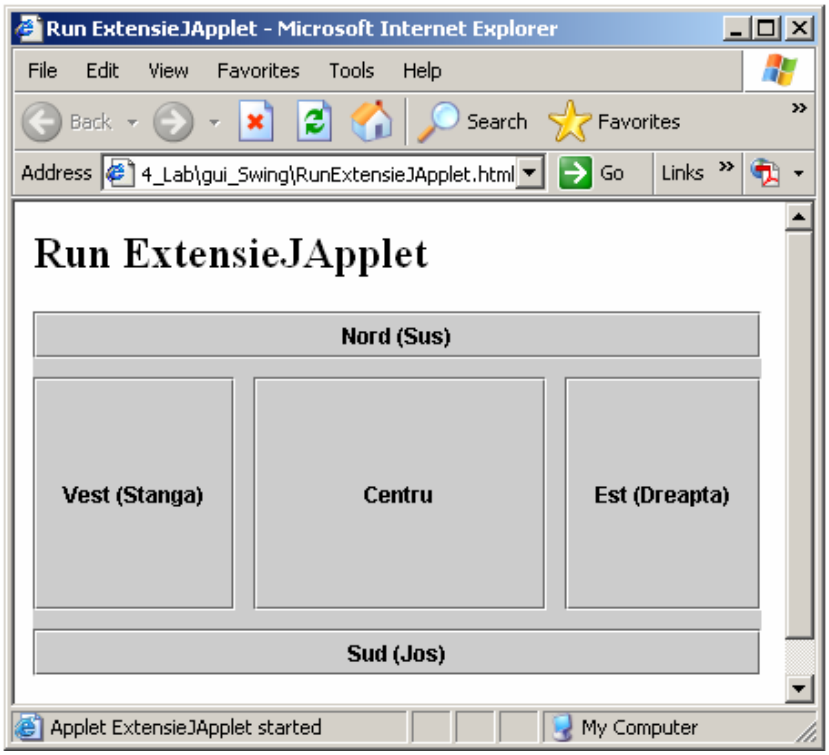
Programul **`ExtensieJApplet`** ilustreaza **crearea unei miniaplicatii (*applet*)** prin **extinderea clasei `JApplet`**, si **asezarea relativ la margini** – **`BorderLayout`** (**script pentru compilarea lui**).

Pentru includerea unei miniaplicatii intr-o pagina HTML in vederea vizualizarii ei se va adauga in corpul fisierului .html urmatorul cod:

```
<APPLET  CODE = "ExtensieJApplet.class" WIDTH = 400 HEIGHT = 200 >
</APPLET>
```

```
1  import java.awt.*;
2  import javax.swing.*;
3  /**
4   * Demonstreaza extinderea JApplet pentru a o miniaplicatie Java.
5   *
6   * Se executa prin includerea intr-o pagina Web a unui tag HTML de genul:
7   *    <APPLET  CODE = "ExtensieJApplet.class" WIDTH = 400 HEIGHT = 200 >
8   *    </APPLET>
9   */
10 public class ExtensieJApplet extends JApplet {
11     /**
12      * Metoda de initializare a appletului. Apelata de browser la prima
13      * utilizare a appletului, stabileste layout-ul (modul de dispunere a
14      * componentelor in panoul de continut) si adauga componentele in panou.
15      */
16     public void init() {
17       // Obtinerea panoului de continut intern cadrului (container de componente)
18       Container container = getContentPane();
19
20       // Asezarea componentelor in panou (la 10 pixeli de marginea panoului)
21       container.setLayout(new BorderLayout(10, 10));
22
23       // Adaugarea a 5 butoane la panoul appletului
24       container.add(new JButton("Est (Dreapta)"), BorderLayout.EAST);
25       container.add(new JButton("Sud (Jos)"), BorderLayout.SOUTH);
26       container.add(new JButton("Vest (Stanga)"), BorderLayout.WEST);
27       container.add(new JButton("Nord (Sus)"), BorderLayout.NORTH);
28       container.add(new JButton("Centru"), BorderLayout.CENTER);
29     }
30 }
```

Pagina HTML **RunExtensieJApplet.html** permite vizualizarea appletului **`ExtensieJApplet`**. Iata cum poate arata pagina **RunExtensieJApplet.html** vizualizata intr-un browser:



4. Crearea unui obiect de tip `JDialog`, pentru a crea o fereastra secundara

5. Crearea ferestrelor de dialog predefinite utilizand metodele clasei `JOptionPane`

Metoda **`showInputDialog()`** a clasei **`JOptionPane`** este folosita pentru a crea **dialoguri de intrare**, metoda **`showMessageDialog()`** pentru a crea **dialoguri de informare a utilizatorului**, etc.

GUI.2.3. Crearea interactivitatii aplicatiilor si miniaplicatiilor grafice Swing

Programele din sectiunea anterioara creau butoane care nu reactioneaza la actionarea lor de catre utilizator.

Pentru introducerea interactivitatii, trebuie tratate evenimentele din interfata grafica. In Java exista mai multe moduri de tratare a evenimentelor.

Incepand cu versiunea initiala, JDK 1.0, interfetele grafice realizate cu biblioteca AWT au 2 moduri de tratare a evenimentelor:

1. Implementand metoda `action()`, care:

- primeste ca parametri un obiect de tip **`Event`** care **incapsuleaza evenimentul produs,** si **un obiect** de tip **`Object`** care **incapsuleaza parametrii** acestuia,

- testeaza atributele `target` si `id` ale obiectului de tip **`Event`** pentru a **identifica obiectul tinta (in care s-a produs evenimentul) si tipul de actiune** produsa, si **trateaza apoi evenimentul respectiv**

2. Implementand metoda `handleEvent()`, care:

- primeste ca parametru un obiect de tip **`Event`** care **incapsuleaza evenimentul produs,**

- testeaza atributele `target` si `id` ale obiectului de tip **`Event`** pentru a **identifica obiectul tinta (in care s-a produs evenimentul) si tipul de actiune** produsa, si **trateaza apoi evenimentul respectiv**

Incepand cu versiunea JDK 1.1, interfetele grafice realizate cu biblioteca AWT au:

3. Un nou mod de tratare a evenimentelor, utilizat si de interfetele grafice Swing, care presupune trei operatii:

- (a) declararea unei clase care implementeaza o interfata « ascultator de evenimente », (care contine metode ce trebuie implementate de utilizator pentru tratarea evenimentului respectiv), sau **(b) declararea unei clase care extinde o clasa predefinita care implementeaza o interfata « ascultator de evenimente »**

- (a) implementarea tuturor metodelor definite in interfata « ascultator de evenimente », sau **(b) re-implementarea metodelor dorite din clasa extinsa care implementeaza interfata**

- inregistrarea unui obiect din clasa « ascultator de evenimente » de catre fiecare dintre componentele grafice (numite tinta sau sursa) pentru care se doreste tratarea evenimentului respectiv

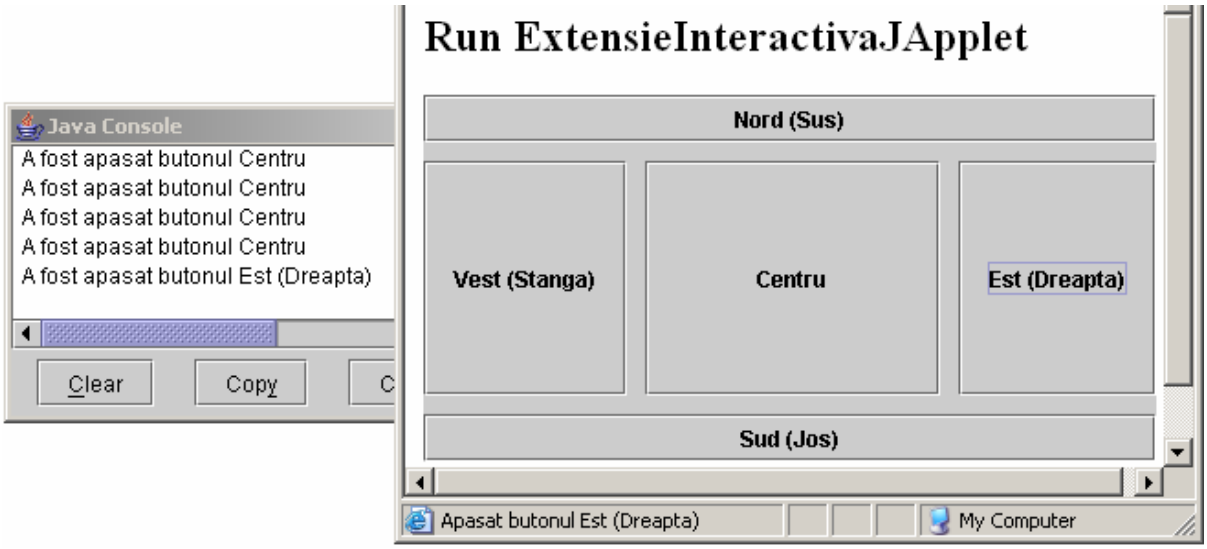
Programul **`ExtensieInteractivaJFrame`** ilustreaza crearea unei ferestre principale prin **extinderea clasei `JFrame`**, **asezarea relativ la margini - `BorderLayout`**, si **tratarea evenimentului « actionare » pentru componentele buton** (**script pentru compilarea si executia lui**).

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4  /**
 5   * Extinderea JFrame pentru a crea o fereastra cu componente interactive pe ecran.
 6   */
 7  public class ExtensieInteractivaJFrame extends JFrame {
 8      public ExtensieInteractivaJFrame() {
 9        // Obtinerea panoului de continut intern cadrului (container de componente)
10        Container container = getContentPane();
11
12        // Asezarea componentelor in panou (la 10 pixeli de marginea panoului)
13        container.setLayout(new BorderLayout(10, 10));
14
15        // Adaugarea a 5 butoane la panoul cadrului (ferestrei)
16        JButton b1 = new JButton("Est (Dreapta)");
17        JButton b2 = new JButton("Sud (Jos)");
18        JButton b3 = new JButton("Vest (Stanga)");
19        JButton b4 = new JButton("Nord (Sus)");
20        JButton b5 = new JButton("Centru");
21        container.add(b1, BorderLayout.EAST);
22        container.add(b2, BorderLayout.SOUTH);
23        container.add(b3, BorderLayout.WEST);
24        container.add(b4, BorderLayout.NORTH);
25        container.add(b5, BorderLayout.CENTER);
26
27        // Inregistrarea "ascultatorului" de "evenim.actionare" la "sursele" evenim.
28        b1.addActionListener(obiectAscultatorActionare);
29        b2.addActionListener(obiectAscultatorActionare);
30        b3.addActionListener(obiectAscultatorActionare);
31        b4.addActionListener(obiectAscultatorActionare);
32        b5.addActionListener(obiectAscultatorActionare);
33
34        // Inregistrarea "ascultatorului" de "evenimente fereastra "
35        this.addWindowListener(ascultatorInchidereFereastra);
36      }
37
38      // Crearea unui obiect "ascultator" de "evenimente actionare"
39      ActionListener obiectAscultatorActionare = new ActionListener() {
40
41        // Tratarea actionarii unui buton
42        public void actionPerformed(ActionEvent ev) {
43          // Mesaj informare
44          System.out.println("A fost apasat un buton " + ev.getActionCommand());
45        }
46      };
47
48      // Crearea unui "adaptor pentru ascultator" de "evenimente fereastra"
49      WindowAdapter ascultatorInchidereFereastra = new WindowAdapter() {
50
51        // Tratarea inchiderii ferestrei curente
52        public void windowClosing(WindowEvent ev) {
53          // Terminarea programului
54          System.exit(0);
55        }
56      };
57
58      public static void main(String[] args) {
59        // Crearea obiectului cadru
60        ExtensieInteractivaJFrame cadru = new ExtensieInteractivaJFrame();
61
62        // Adaugarea titlului ferestrei
63        cadru.setTitle("Demo extindere JFrame interactiva");
64        // Compactarea componentelor
65        cadru.pack();
66        // Stabilirea vizibilitatii ferestrei (Atentie: implicit e false!)
67        cadru.setVisible(true);
68      }
69  }
```

Programul **ExtensieInteractivaJApplet** ilustreaza **crearea unei miniaplicatii (*applet*)** prin **extinderea clasei `JApplet`**, si **tratarea evenimentului « actionare » pentru componentele buton** (**script pentru compilarea lui**).

```
 1  Import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4
 5  public class ExtensieInteractivaJApplet extends JApplet {
 6
 7      public void init() {
 8        // Obtinerea panoului de continut (content pane) creat de browser pentru
 9        // executia appletului (container in care vor fi plasate componentele)
10        Container container = getContentPane();
11
12        // Stabilirea layout-ului panoului, BorderLayout cu spatiu 10 pixeli
13        container.setLayout(new BorderLayout(10, 10));
14        // Adaugarea a 5 butoane la panoul appletului
15        JButton b1 = new JButton("Est (Dreapta)");
16        JButton b2 = new JButton("Sud (Jos)");
17        JButton b3 = new JButton("Vest (Stanga)");
18        JButton b4 = new JButton("Nord (Sus)");
19        JButton b5 = new JButton("Centru");
20        container.add(b1, BorderLayout.EAST);
21        container.add(b2, BorderLayout.SOUTH);
22        container.add(b3, BorderLayout.WEST);
23        container.add(b4, BorderLayout.NORTH);
24        container.add(b5, BorderLayout.CENTER);
25
26        // Crearea unui obiect "ascultator" de "evenimente actionare"
27        // (pe care le trateaza)
28        ActionListener obiectAscultatorActionare = new ActionListener() {
29
30          // Tratarea actionarii unui buton
31          public void actionPerformed(ActionEvent ev) {
32
33            // Mesaj informare in consola Java
34            System.out.println("A fost apasat butonul " + ev.getActionCommand());
35
36            // Mesaj informare in bara de stare
37            showStatus("Apasat butonul " + ev.getActionCommand());
38          }
39        };
40
41        // Inregistrarea "ascultatorului" de "evenimente actionare" la "sursele"
42        // de evenimente
43        b1.addActionListener(obiectAscultatorActionare);
44        b2.addActionListener(obiectAscultatorActionare);
45        b3.addActionListener(obiectAscultatorActionare);
46        b4.addActionListener(obiectAscultatorActionare);
47        b5.addActionListener(obiectAscultatorActionare);
48      }
49  }
```

Pagina **RunExtensieInteractivaJApplet.html** permite vizualizarea *applet-ului*. Iata cum poate arata pagina **RunExtensieJApplet.html** vizualizata intr-un browser:



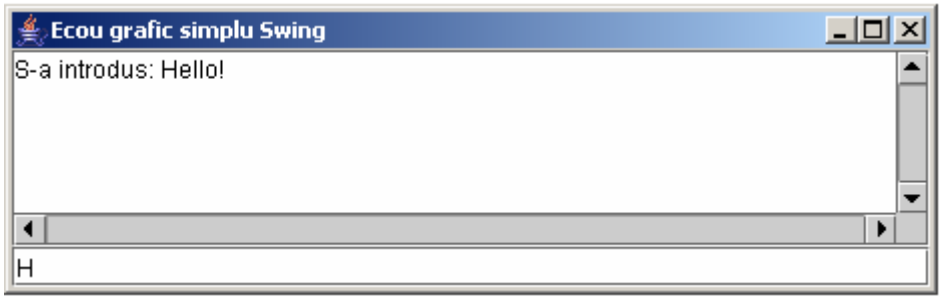
GUI.2.4. Utilizarea componentelor grafice Swing pentru lucrul cu text

Programul **EcouGrafic_Swing** ilustreaza utilizarea componentelor grafice Swing pentru lucrul cu text, de tip **JTextField** si **JTextArea** (**script pentru compilarea si executia lui**).

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4  /**
 5    * Ecou grafic folosind JtextField si JTextArea
 6    */
 7  public class EcouGrafic_Swing extends JFrame {
 8    /**
 9      * Intrare - linie de text grafica (JtextField)
10      */
11    private static JTextField inTextGrafic;
12    /**
13      * Iesire - zona de text grafica (JtextArea)
14      */
15    private static JTextArea outTextGrafic;
16
17    /**
18      * Initializari grafice
19      */
20    public EcouGrafic_Swing() {
21
22      // Stabilire titlu fereastra (JFrame)
23      super("Ecou grafic simplu Swing");
24      Container containerCurent = this.getContentPane();
25      containerCurent.setLayout(new BorderLayout());
26
27      // Zona de text non-editabila de iesire (cu posibilitati de defilare)
28
29      outTextGrafic = new JTextArea(5, 40);
30      JScrollPane scrollPane = new JScrollPane(outTextGrafic,
31                      JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
32                      JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
33      containerCurent.add("Center", scrollPane);
34      outTextGrafic.setEditable(false);
35
36      // Camp de text editabil de intrare
37
38      inTextGrafic = new JTextField(40);
39      containerCurent.add("South", inTextGrafic);
40
41      System.out.println("\nPentru oprire introduceti '.' si <Enter>\n");
42
43      // Inregistrarea "ascultatorului" de "evenimente actionare" la
```

```
44      // "obiectul sursa" intrare de text
45      inTextGrafic.addActionListener(ascultatorInText);
46
47      // Inregistrarea "ascultatorului" de "evenimente fereastra" la
48      // "sursa" (fereastra curenta)
49      this.addWindowListener(ascultatorInchidere);
50
51      // Impachetarea (compactarea) componentelor in container
52      pack();
53      // Fereastra devine vizibila - echivalent cu frame.setVisible(true)
54      show();
55      // Cerere focus pe intrarea de text din fereastra curenta
56      inTextGrafic.requestFocus();
57    }
58
59    /**
60      * Crearea unui "ascultator" de "evenimente actionare", obiect al unei
61      * clase "anonime" care implementeaza interfata ActionListener
62      */
63    ActionListener ascultatorInText = new ActionListener() {
64      /**
65        * Tratarea actionarii intrarii de text (introducerii unui "Enter")
66        */
67      public void actionPerformed(ActionEvent ev) {
68        // Citirea unei linii de text din intrarea de text grafica
69        String sirCitit = inTextGrafic.getText();
70        // Pregatirea intrarii de text pentru noua intrare (golirea ei)
71        inTextGrafic.setText("");
72
73        // Scrierea liniei de text in zona de text grafica
74        outTextGrafic.append("S-a introdus: " + sirCitit + "\n");
75        // Conditie terminare program
76        if (sirCitit.equals(new String(".")))   System.exit(0);
77      }
78    };
79
80    /**
81      * Crearea unui "adaptor pentru ascultator" de "evenimente fereastra"
82      * Obiect al unei clase "anonime" care extinde clasa WindowAdapter
83      */
84    WindowAdapter ascultatorInchidere = new WindowAdapter() {
85
86      /**
87        * Tratarea inchiderii ferestrei curente
88        */
89      public void windowClosing(WindowEvent ev) {
90        // Terminarea programului
91        System.exit(0);
92      }
93    };
94
95    /**
96      * Punctul de intrare in program
97      */
98    public static void main (String args[]) {
99      EcouGrafic_Swing ecouGraficJTFJTA = new EcouGrafic_Swing();
00    }
01  }
```

Fereastra obtinuta prin executia programului **`EcouGrafic_Swing.java`**:



GUI.2.5. Surse multiple de evenimente tratate de un singur ascultator

Programul **SurseMultiple** ilustreaza tratarea evenimentelor generate de surse (componente grafice Swing) multiple (**script pentru compilarea si executia lui**).

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4  /**
 5   *  Ilustreaza surse multiple de evenimente
 6   */
 7  public class SurseMultiple extends JFrame {
 8    private JTextField intrareNume;
 9    private JButton butonConfirmareNume;
10    private String nume;
11    private boolean numeIntrodus = false;
12    private JTextField intrarePrenume;
13    private JButton butonConfirmarePrenume;
14    private String prenume;
15    private boolean prenumeIntrodus = false;
16
17    private JTextField intrareNickname;
18    private JButton butonConfirmareNickname;
19    private String nickname;
20    private boolean nicknameIntrodus = false;
21    /**
22     * Initializeaza obiectul de tip SurseMultiple
23     */
24    public SurseSiAscultatoriMultipli() {
25      // Apel al constructorului supraclasei (JFrame) pentru stabilirea titlului
26      super ("Va rugam sa va prezentati!");
27
28      // Containerul ferestrei grafice
29      Container containerFereastra = this.getContentPane();
30
31      // Stabilirea modului de asezare a componentelor in fereastra (layoutului)
32      containerFereastra.setLayout(new GridLayout(3,2));
33
34      // Camp de text pentru introducerea numelui
35      intrareNume = new JTextField(20);
36      containerFereastra.add(intrareNume);
37
38      // Buton pentru confirmarea introducerii numelui
39      butonConfirmareNume = new JButton("Numele dvs.");
40      containerFereastra.add(butonConfirmareNume);
41
42      // Camp de text pentru introducerea prenumelui
43      intrarePrenume = new JTextField(20);
44      containerFereastra.add(intrarePrenume);
45
46      // Buton pentru confirmarea introducerii prenumelui
47      butonConfirmarePrenume = new JButton("Prenumele dvs.");
48      containerFereastra.add(butonConfirmarePrenume);
49
50      // Camp de text pentru introducerea numelui utilizatorului
51      intrareNickname = new JTextField(20);
52      containerFereastra.add(intrareNickname);
53
54      // Buton pentru confirmarea introducerii numelui utilizatorului
55      butonConfirmareNickname = new JButton("Nume dvs. de utilizator");
56      containerFereastra.add(butonConfirmareNickname);
57
58      // Inregistrarea "ascultatorilor" de "evenimente actionare"
59      // Butoane (actionare = click pe buton)
60      butonConfirmareNume.addActionListener(ascultatorEvenimenteActionare);
61      butonConfirmarePrenume.addActionListener(ascultatorEvenimenteActionare);
62      butonConfirmareNickname.addActionListener(ascultatorEvenimenteActionare);
63
64      // Intrari grafice de text (actionare = apasare <Enter>)
65      intrareNume.addActionListener(ascultatorEvenimenteActionare);
66      intrarePrenume.addActionListener(ascultatorEvenimenteActionare);
```

```
67      intrareNickname.addActionListener(ascultatorEvenimenteActionare);
68
69      // Inregistrarea "ascultatorului" de "evenimente fereastra" la "sursa"
70      // (fereastra curenta)
71      this.addWindowListener(ascultatorInchidere);
72      // Impachetarea (compactarea) componentelor in container
73      pack();
74      // Fereastra devine vizibila - echivalent cu frame.setVisible(true)
75      show();
76    }
77    /** Crearea unui "ascultator" de "evenimente actionare", obiect al unei
78      * clase "anonime" care implementeaza interfata ActionListener    */
79    WindowAdapter ascultatorInchidere = new WindowAdapter() {
80      public void windowClosing(WindowEvent ev) {
81        // Terminarea programului
82        System.exit(0);
83      }
84    };
85    /**
86      * Crearea unui "ascultator" de "evenimente actionare"
87      * Obiect al unei clase "anonime" care implementeaza interfata ActionListener
88      */
89    ActionListener ascultatorEvenimenteActionare = new ActionListener() {
90
91      /**
92        * Tratarea evenimentului actionare pentru ascultatorii inregistrati
93        */
94      public void actionPerformed(ActionEvent ev) {
95
96        // Eveniment legat de introducerea numelui
97        if ((ev.getSource() == butonConfirmareNume) ||
98            (ev.getSource() == intrareNume)) {
99
00          // Culegerea informatiilor din intrarea text
 1          nume = intrareNume.getText();
 2
 3          // Tratarea actionarilor asociate introducerii numelui
 4          System.out.println("A fost introdus numele: " + nume);
 5
 6          // Dezactivarea posibilitatii de modificare a numelui
 7          intrareNume.setEditable(false);
 8          butonConfirmareNume.setEnabled(false);
 9
10          numeIntrodus = true;
11        }
12
13        // Eveniment legat de introducerea prenumelui
14        else if ((ev.getSource() == butonConfirmarePrenume) ||
15                 (ev.getSource() == intrarePrenume)) {
16
17          // Culegerea informatiilor din intrarea text
18          prenume = intrarePrenume.getText();
19
20          // Tratarea actionarilor asociate introducerii numelui
21          System.out.println("A fost introdus prenumele: " + prenume);
22
23          // Dezactivarea posibilitatii de modificare a prenumelui
24          intrarePrenume.setEditable(false);
25          butonConfirmarePrenume.setEnabled(false);
26
27          prenumeIntrodus = true;
28        }
29        // Eveniment legat de introducerea numelui de utilizator
30        else if ((ev.getSource() == butonConfirmareNickname) ||
31                 (ev.getSource() == intrareNickname)) {
32
33          // Culegerea informatiilor din intrarea text
34          nickname = intrareNickname.getText();
35
36          // Tratarea actionarilor asociate introducerii numelui
37          System.out.println("A fost introdus numele utilizatorului: " + nickname);
38
39          // Dezactivarea posibilitatii de modificare a numelui de utilizator
```

```
40          intrareNickname.setEditable(false);
41          butonConfirmareNickname.setEnabled(false);
42
43          nicknameIntrodus = true;
44        }
45
46        if (numeIntrodus && prenumeIntrodus && nicknameIntrodus) {
47          setTitle("Va numiti " + nume + prenume + " (alias " + nickname + ")");
48        }
49      }
50    };
51    /**
52     * Metoda principala
53     */
54    public static void main(java.lang.String[] args) {
55      new SurseMultiple();
56    }
57  }
```

GUI.2.6. Surse multiple de evenimente multiple

Programul **SurseSiAscultatoriMultipli** ilustreaza tratarea evenimentelor generate de surse (componente grafice Swing) multiple catre mai multi ascultatori de evenimente (**script pentru compilarea si executia lui**).

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4  import javax.swing.event.*;
 5  import javax.swing.text.*;
 6
 7  /**
 8   *  Ilustreaza surse multiple de evenimente si ascultatori multipli ai
 9   *  evenimentelor
10   */
11  public class SurseSiAscultatoriMultipli extends JFrame {
12    private JTextField intrareInformatie;
13    private JButton butonConfirmareInformatie;
14    private String informatie;
15
16    /**
17     * Initializeaza obiectul de tip SurseSiAscultatoriMultipli
18     */
19    public SurseSiAscultatoriMultipli() {
20      // Apel al constructorului supraclasei (JFrame) pentru stabilirea titlului
21      super ("Surse multiple si ascultatori multipli");
22
23      // Containerul ferestrei grafice
24      Container containerFereastra = this.getContentPane();
25
26      // Stabilirea modului de asezare a componentelor in fereastra (layoutului)
27      containerFereastra.setLayout(new FlowLayout());
28
29      // Camp de text pentru introducerea informatiei
30      intrareInformatie = new JTextField(20);
31      containerFereastra.add(intrareInformatie);
32
33      // Buton pentru confirmarea introducerii informatiei
34      butonConfirmareInformatie = new JButton("Informatie");
35      containerFereastra.add(butonConfirmareInformatie);
36
37      // Inregistrarea "ascultatorilor" de "evenimente actionare"
38      // Buton (actionare = click pe buton)
39      butonConfirmareInformatie.addActionListener(ascultatorEvenimenteActionare);
40
41      // Intrare grafica de text (actionare = apasare <Enter>)
42      intrareInformatie.addActionListener(ascultatorEvenimenteActionare);
```

```
43
44      // Inregistrarea "ascultatorilor" de "evenimente document"
45      // Intrare grafica de text (inserare, eliminare)
46      intrareInformatie.getDocument().addDocumentListener(ascultatorDocument);
47
48      // Inregistrarea "ascultatorului" de "evenimente fereastra"
49      this.addWindowListener(ascultatorInchidere);
50
51      // Impachetarea (compactarea) componentelor in container
52      pack();
53      // Fereastra devine vizibila - echivalent cu frame.setVisible(true)
54      show();
55    }
56
57
58
59
60
61    /**
62      * Crearea unui "ascultator" de "evenimente document"
63      * Obiect al unei clase "anonime" care implementeaza interfata DocumentListener
64      */
65    DocumentListener ascultatorDocument = new DocumentListener() {
66      // Tratarea evenimentului inserare in document
67      public void insertUpdate(DocumentEvent ev) {
68        Document doc = (Document)ev.getDocument();
69        int numarCaractere = ev.getLength();
70        System.out.println(numarCaractere + " caracter" +
71              ((numarCaractere == 1) ? " a " : "e au ") +
72              "fost adaugat" + ((numarCaractere == 1) ? " " : "e ") +
73              "la document.\n" +
74              "  Lungimea textului = " + doc.getLength() + "\n");
75      }
76      // Tratarea evenimentului eliminare din document
77      public void removeUpdate(DocumentEvent ev) {
78        Document doc = (Document)ev.getDocument();
79        int numarCaractere = ev.getLength();
80        System.out.println(numarCaractere + " caracter" +
81              ((numarCaractere == 1) ? " a " : "e au ") +
82              "fost eliminat" + ((numarCaractere == 1) ? " " : "e ") +
83              "din document.\n" +
84              "  Lungimea textului = " + doc.getLength() + "\n");
85      }
86      //Documentele text nu pot genera aceste evenimente
87      public void changedUpdate(DocumentEvent ev) {
88      }
89    };
90    /**
91      * Crearea unui "adaptor pentru ascultator" de "evenimente fereastra"
92      * Obiect al unei clase "anonime" care extinde clasa WindowAdapter
93      */
94    WindowAdapter ascultatorInchidere = new WindowAdapter() {
95      /**
96        * Tratarea inchiderii ferestrei curente
97        */
98      public void windowClosing(WindowEvent ev) {
99        // Terminarea programului
00        System.exit(0);
 1      }
 2    };
 3    /**
 4      * Crearea unui "ascultator" de "evenimente actionare"
 5      * Obiect al unei clase "anonime" care implementeaza interfata ActionListener
 6      */
 7    ActionListener ascultatorEvenimenteActionare = new ActionListener() {
 8      // Tratarea evenimentului actionare pentru ascultatorii inregistrati
 9      public void actionPerformed(ActionEvent ev) {
10
11        // Eveniment legat de introducerea informatiei
12        if ((ev.getSource() == butonConfirmareInformatie) ||
13            (ev.getSource() == intrareInformatie)) {
14
15          // Culegerea informatiilor din intrarea text
```

```
16          informatie = intrareInformatie.getText();
17
18          // Tratarea actionarilor asociate introducerii numelui
19          System.out.println("A fost introdusa informatia: " + informatie);
20
21          // Dezactivarea posibilitatii de modificare a numelui
22          intrareInformatie.setEditable(false);
23          butonConfirmareInformatie.setEnabled(false);
24        }
25      }
26    };
27    /**
28     * Metoda principala
29     */
30    public static void main(java.lang.String[] args) {
31      new SurseSiAscultatoriMultipli();
32    }
33  }
```

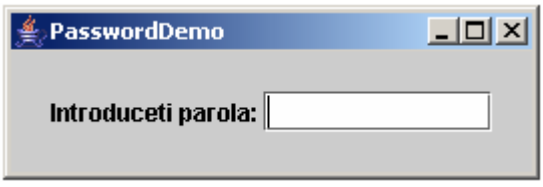
GUI.2.7. Alte componente grafice Swing

1. Intrare text pentru parole

Programul **PasswordDemo** si un **script pentru compilarea si executia lui**.

```
1    import javax.swing.*;
2    import java.awt.*;
3    import java.awt.event.*;
4
5    public class PasswordDemo {
6      private static boolean isPasswordCorrect(char[] input) {
7        char[] correctPassword = { 'g', 'h', 'i', 'c', 'i' };
8        if (input.length != correctPassword.length)
9          return false;
10       for (int i = 0;  i < input.length; i ++)
11         if (input[i] != correctPassword[i])
12           return false;
13       return true;
14     }
15
16     public static void main(String[] argv) {
17       final JFrame f = new JFrame("PasswordDemo");
18       JLabel label = new JLabel("Introduceti parola: ");
19
20       JPasswordField passwordField = new JPasswordField(10);
21
22       passwordField.setEchoChar('#');
23
24       // Tratarea actionarii intrarii
25       passwordField.addActionListener(new ActionListener() {
26         public void actionPerformed(ActionEvent e) {
27
28           JPasswordField input = (JPasswordField)e.getSource();
29
30           char[] password = input.getPassword();
31
32           if (isPasswordCorrect(password)) {
33             JOptionPane.showMessageDialog(f, "Succes! Password corect.");
34           }
35           else {
36             JOptionPane.showMessageDialog(f, "Password incorect. Mai incearca.",
37                         "Error Message", JOptionPane.ERROR_MESSAGE);
38           }
39         }
40       });
41
42       JPanel contentPane = new JPanel(new BorderLayout());
43       contentPane.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
44       contentPane.add(label, BorderLayout.WEST);
45       contentPane.add(passwordField, BorderLayout.CENTER);
46
47       f.setContentPane(contentPane);
48       f.addWindowListener(new WindowAdapter() {
49         public void windowClosing(WindowEvent e) { System.exit(0); }
50       });
51       f.pack();
52       f.setVisible(true);
53     }
54   }
```

Fereastrele obtinute prin executia programului **PasswordDemo.java**:



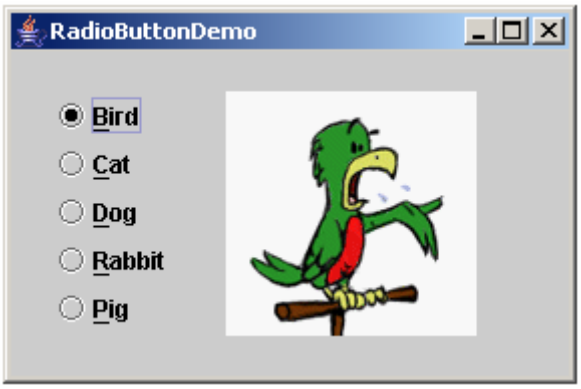
2. Butoane radio (`RadioButton`)

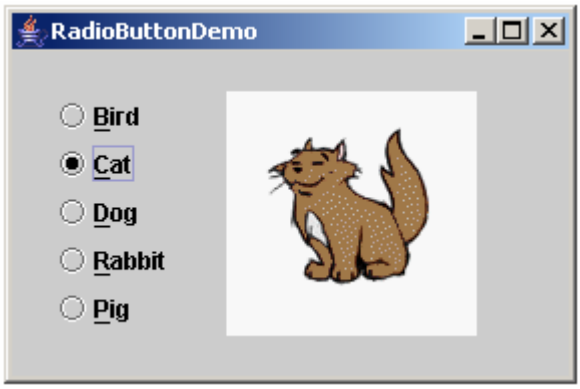
Programul **`RadioButtonDemo`** si un **script pentru compilarea si executia lui**.

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4
 5  public class RadioButtonDemo extends JPanel {
 6      static JFrame frame;
 7
 8      static String birdString = "Bird";
 9      static String catString = "Cat";
10      static String dogString = "Dog";
11      static String rabbitString = "Rabbit";
12      static String pigString = "Pig";
13
14      JLabel picture;
15
16      public RadioButtonDemo() {
17          // Create the radio buttons.
18          JRadioButton birdButton = new JRadioButton(birdString);
19          birdButton.setMnemonic(KeyEvent.VK_B);
20          birdButton.setActionCommand(birdString);
21          birdButton.setSelected(true);
22
23          JRadioButton catButton = new JRadioButton(catString);
24          catButton.setMnemonic(KeyEvent.VK_C);
25          catButton.setActionCommand(catString);
26
27          JRadioButton dogButton = new JRadioButton(dogString);
28          dogButton.setMnemonic(KeyEvent.VK_D);
29          dogButton.setActionCommand(dogString);
30
31          JRadioButton rabbitButton = new JRadioButton(rabbitString);
32          rabbitButton.setMnemonic(KeyEvent.VK_R);
33          rabbitButton.setActionCommand(rabbitString);
34
35          JRadioButton pigButton = new JRadioButton(pigString);
36          pigButton.setMnemonic(KeyEvent.VK_P);
37          pigButton.setActionCommand(pigString);
38
39          // Group the radio buttons.
40          ButtonGroup group = new ButtonGroup();
41          group.add(birdButton);
42          group.add(catButton);
43          group.add(dogButton);
44          group.add(rabbitButton);
45          group.add(pigButton);
46
47          // Register a listener for the radio buttons.
48          RadioListener myListener = new RadioListener();
49          birdButton.addActionListener(myListener);
50          catButton.addActionListener(myListener);
51          dogButton.addActionListener(myListener);
52          rabbitButton.addActionListener(myListener);
53          pigButton.addActionListener(myListener);
54
55          // Set up the picture label
56          picture = new JLabel(new ImageIcon("images/"
57                                             + birdString
58                                             + ".gif"));
```

```
59          // The preferred size is hard-coded to be the width of the
60          // widest image and the height of the tallest image.
61          // A real program would compute this.
62          picture.setPreferredSize(new Dimension(177, 122));
63
64
65          // Put the radio buttons in a column in a panel
66          JPanel radioPanel = new JPanel();
67          radioPanel.setLayout(new GridLayout(0, 1));
68          radioPanel.add(birdButton);
69          radioPanel.add(catButton);
70          radioPanel.add(dogButton);
71          radioPanel.add(rabbitButton);
72          radioPanel.add(pigButton);
73
74          setLayout(new BorderLayout());
75          add(radioPanel, BorderLayout.WEST);
76          add(picture, BorderLayout.CENTER);
77          setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
78      }
79
80      /** Listens to the radio buttons. */
81      class RadioListener implements ActionListener {
82          public void actionPerformed(ActionEvent e) {
83              picture.setIcon(new ImageIcon("images/"
84                                            + e.getActionCommand()
85                                            + ".gif"));
86          }
87      }
88      public static void main(String s[]) {
89           frame = new JFrame("RadioButtonDemo");
90           frame.addWindowListener(new WindowAdapter() {
91               public void windowClosing(WindowEvent e) {System.exit(0);}
92           });
93
94           frame.getContentPane().add(new RadioButtonDemo(), BorderLayout.CENTER);
95           frame.pack();
96           frame.setVisible(true);
97      }
98  }
```

Fereastrele obtinute prin executia programului **`RadioButtonDemo.java`**:





3. Cutie de optiuni (`CheckBox`)

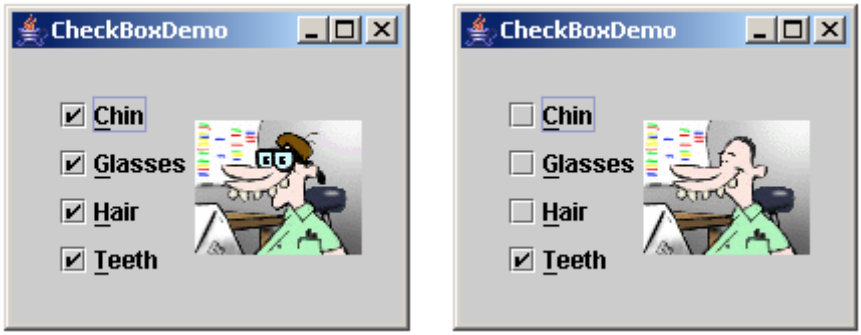
Programul **`CheckBoxDemo`** si un **script pentru compilarea si executia lui**.

```
1   import java.awt.*;
2   import java.awt.event.*;
3   import javax.swing.*;
4
5   public class CheckBoxDemo extends JPanel {
6       JCheckBox chinButton;
7       JCheckBox glassesButton;
8       JCheckBox hairButton;
9       JCheckBox teethButton;
```

```
    /*
     * Four accessory choices provide for 16 different
     * combinations. The image for each combination is
     * contained in a separate image file whose name indicates
     * the accessories. The filenames are "geek-XXXX.gif"
     * where XXXX can be one of the following 16 choices.
     * The "choices" StringBuffer contains the string that
     * indicates the current selection and is used to generate
     * the file name of the image to display.

       ----             // zero accessories

       c---             // one accessory
       -g--
       --h-
       ---t

       cg--             // two accessories
       c-h-
       c--t
       -gh-
       -g-t
       --ht

       -ght             // three accessories
       c-ht
       cg-t
       cgh-

       cght             // all accessories
     */

    StringBuffer choices;
    JLabel pictureLabel;

    public CheckBoxDemo() {
        // Create the check boxes
        chinButton = new JCheckBox("Chin");
        chinButton.setMnemonic(KeyEvent.VK_C);
        chinButton.setSelected(true);

        glassesButton = new JCheckBox("Glasses");
        glassesButton.setMnemonic(KeyEvent.VK_G);
        glassesButton.setSelected(true);

        hairButton = new JCheckBox("Hair");
        hairButton.setMnemonic(KeyEvent.VK_H);
        hairButton.setSelected(true);

        teethButton = new JCheckBox("Teeth");
        teethButton.setMnemonic(KeyEvent.VK_T);
        teethButton.setSelected(true);

        // Register a listener for the check boxes.
        CheckBoxListener myListener = new CheckBoxListener();
        chinButton.addItemListener(myListener);
        glassesButton.addItemListener(myListener);
        hairButton.addItemListener(myListener);
        teethButton.addItemListener(myListener);

        // Indicates what's on the geek.
        choices = new StringBuffer("cght");

        // Set up the picture label
        pictureLabel = new JLabel(new ImageIcon(
                                        "images/geek/geek-"
                                         + choices.toString()
                                         + ".gif"));
        pictureLabel.setToolTipText(choices.toString());

        // Put the check boxes in a column in a panel
        JPanel checkPanel = new JPanel();
        checkPanel.setLayout(new GridLayout(0, 1));
```

```
83          checkPanel.add(chinButton);
84          checkPanel.add(glassesButton);
85          checkPanel.add(hairButton);
86          checkPanel.add(teethButton);
87
88          setLayout(new BorderLayout());
89          add(checkPanel, BorderLayout.WEST);
90          add(pictureLabel, BorderLayout.CENTER);
91          setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
92      }
93
94      /** Listens to the check boxes. */
95      class CheckBoxListener implements ItemListener {
96          public void itemStateChanged(ItemEvent e) {
97              int index = 0;
98              char c = '-';
99              Object source = e.getItemSelectable();
00
01              if (source == chinButton) {
02                  index = 0;
03                  c = 'c';
04              } else if (source == glassesButton) {
05                  index = 1;
06                  c = 'g';
07              } else if (source == hairButton) {
08                  index = 2;
09                  c = 'h';
10              } else if (source == teethButton) {
11                  index = 3;
12                  c = 't';
13              }
14
15              if (e.getStateChange() == ItemEvent.DESELECTED)
16                  c = '-';
17              choices.setCharAt(index, c);
18              pictureLabel.setIcon(new ImageIcon(
19                                      "images/geek/geek-"
20                                      + choices.toString()
21                                      + ".gif"));
22              pictureLabel.setToolTipText(choices.toString());
23          }
24      }
25
26      public static void main(String s[]) {
27           JFrame frame = new JFrame("CheckBoxDemo");
28           frame.addWindowListener(new WindowAdapter() {
29               public void windowClosing(WindowEvent e) {
30                   System.exit(0);
31               }
32           });
33
34           frame.setContentPane(new CheckBoxDemo());
35           frame.pack();
36           frame.setVisible(true);
37      }
38  }
```

Fereastrele obtinute prin executia programului **`CheckBoxDemo.java`**:



GUI.3. Introducere in applet-uri Java

GUI.3.1. Caracteristicile *applet-urilor* Java

Applet-urile sau miniaplicatiile Java sunt **portiuni de cod Java care mostenesc clasa `Applet`**.

Prin plasarea lor in *browser-e*, *applet-urile* devin **panouri frontale ale serviciilor distribuite oferite de *Web***.

Applet-urile sunt **mai intai incarcate in *browser-e***, fiind **apoi executate in mediul de executie oferit de acesta**.

Applet-urile **nu sunt aplicatii complete**, ci **componente care ruleaza in mediul *browser-ului***.

Browser-ul* actioneaza ca un *framework pentru executia *applet-urile* (componentelor Java).

Browser-ul* informeaza *applet-ul asupra evenimentelor care se petrec pe durata de viata a *applet-ului*.

Serviciile oferite de *browser* sunt:

- **controlul total al ciclului de viata** al *applet-ului*,
- **furnizarea informatiilor privind atributele** din *tag-ul* APPLET,
- **functia de program/proces principal** din care se executa *applet-urile* (ofera functia **main()**).

GUI.3.2. Ciclul de viata al *applet-urilor* Java

Clasa `Applet` interfata `Runnable` definesc metode pe care un *browser* le poate invoca pe durata ciclului de viata al unui *applet*.

Browser-ul* invoca*:

- **`init()`** cand ***incarca applet-ul prima oara***;
- **`start()`** cand ***un utilizator intra*** sau ***reintra in pagina care contine applet-ul***;
- **`stop()`** cand ***utilizatorul iese din pagina***;
- **`destroy()`** ***inaintea terminarii normale.***

Invocarea ultimelor doua metode conduce la **"omorarea" tuturor firelor de executie** ale *applet-ului* si la **eliberarea tuturor resurselor** *applet-ului*.

Urmatorul *applet* simplu:

```
import java.applet.Applet;
import java.awt.Graphics;

public class Simple extends Applet {

    StringBuffer buffer;

    public void init() {
        buffer = new StringBuffer();
```

```
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting... ");
    }

    public void stop() {
        addItem("stopping... ");
    }

    public void destroy() {
        addItem("preparing for unloading...");
    }

    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }

    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0, size().width - 1, size().height - 1);

        //Draw the current string inside the rectangle.
        g.drawString(buffer.toString(), 5, 15);
    }
}
```

permite, **prin vizualizarea lui**, urmarirea fazelor ciclului de viata ale unui *applet*.

Pentru a avea interactivitate avansata, *applet-ul* trebuie sa implementeze metoda `run()` a interfetei **`Runnable`** sau metoda `run()` a clasei `Thread`, **care se executa in interiorul unui fir de executie** (*thread*).

GUI.3.3. Crearea unui *applet* Java

Miniaplicatia **FirstApplet** foloseste componentele multimedia integrate ale limbajului Java pentru afisarea unei imagini si redarea unui fisier de sunet.

```
import java.awt.*;
import java.applet.*;

public class FirstApplet extends Applet {

Image NewImage;

      public void init() {
        resize(400,400);
        NewImage = getImage(getCodeBase(),"New.gif");
      }

      public void paint(Graphics g) {
        g.drawString("Hello!");
        g.drawImage(NewImage,0,0,this);
        play(getCodeBase(),"New.au");
      }

}
```

Instructiunea **import** permite miniaplicatiei sa foloseasca metode si clase din alte pachete:

```
import java.awt.*;
import java.applet.*;
```

In mod prestabilit, toate programele Java importa pachetul java.lang, care contine functiile de baza ale limbajului Java. Asteriscul de la sfarsitul instructiunii import permite importul dinamic al claselor Java. In acest exemplu, sunt importate dinamic clasele din pachetele java.awt si java.applet.

Linia urmatoare declara o clasa numita **FirstApplet** care extinde clasa **`Applet`**:

```
public class FirstApplet extends Applet {
```

Prin extinderea clasei **`Applet`**, **FirstApplet** mosteneste functionalitatea acestei clase. Acolada deschisa marcheaza inceputul clasei **FirstApplet**.

Linia urmatoare initializeaza variabila **NewImage** si o declara de a fi de tipul Image. In acest caz, **NewImage** are rolul unui substituent al imaginii care va fi afisata:

```
Image NewImage;
```

Linia urmatoare declara o metoda numita **`init()`**, care redefineste metoda **`init()`** a clasei **`Applet`**:

```
public void init() {
```

Metoda **`init()`** a clasei Applet este redefinita, astfel incat sa puteti redimensiona fereastra inainte de afisarea imaginii. Modificatorul public specifica faptul ca metoda este accesibila altor clase. Modificatorul void specifica faptul ca metoda nu returneaza nici o valoare. In mod normal, argumentele acceptate de o metoda sunt incadrate de paranteze rotunde. Deoarece metoda **`init()`** nu accepta argumente, intre paranteze nu apare nimic.

Folosind metoda **`resize()`**, puteti sa redimensionati zona de afisare a miniaplicatiei. In acest exemplu, dimensiunea zonei de afisare este stabilita la 400x400 pixeli:

```
resize(400,400);
```

Dupa ce ati declarat o variabila de un anumit tip, puteti sa o folositi. Linia urmatoare stabileste o valoare pentru variabila NewImage;

```
NewImage = getImage(getCodeBase(), "New.gif");
```

Pentru aceasta, este folosita metoda **getImage( )**. Primul argument al metodei este un apel al metodei `getCodeBase()`, care returneaza pozitia directorului de baza sau a directorului curent de pe hard-disc. Directorul de baza este directorul care contine fisierul clasei pe care o rulati. Al doilea argument este numele imaginii care poate fi gasita in pozitia specificata.

Urmatoarea linie de cod declara o metoda numita **paint( )**, care redefineste metoda **paint( )** din pachetul AWT:

```
public void paint (Graphics g) {
```

Metoda **paint( )** este redefinita pentru a permite miniaplicatiei sa afiseze imaginea intr-o anumita pozitie pe ecran. Modificatorul public specifica faptul ca metoda este accesibila altor clase. Modificatorul void specifica faptul ca metoda nu returneaza nici o valoare. La apelarea metodei **paint( )**, trebuie sa folositi ca parametru un obiect al clasei **Graphics**.

Graphics este o ***clasa de baza abstracta pentru toate obiectele grafice***. Elementul g reprezinta fereastra de tip **Graphics** specificata.

Linia urmatoare apeleaza obiectul g, de tip **Graphics**, pentru afisarea imaginii **NewImage:**

```
g.drawImage(NewImage,0,0,this);
```

Metoda care realizeaza de fapt operatiunea se numeste **drawImage( )**. Metoda **drawImage( )** accepta argumente prin care i se precizeaza ce imagine trebuie sa afiseze si unde. In acest exemplu, obiectul **NewImage** este afisat in punctul de coordonate 0,0. Ultimul argument al metodei se numeste observator. Scopul acestuia este sa verifice daca imaginea a fost afisata integral.

Asa cum sugereaza si numele sau metoda **play( )** este folosita pentru redarea fisierelor de sunet. Primul argument al metodei **play( )** este un apel al metodei `getCodeBase()`, care returneaza pozitia directorului de baza sau a directorului curent de pe *hard-disc*.

```
play(getCodeBase(), "New.au");
```

Directorul de baza este directorul care contine fisierul clasei pe care o rulati. Al doilea argument este numele fisierului de sunet care poate fi gasit in pozitia specificata.

Crearea miniaplicatiei

Trebuie sa stocati programul **FirstApplet** intr-un fisier numit **FirstApplet.java**. Acesta va fi salvat ca fisier de text ASCII standard.

Compilarea miniaplicatiei FirstApplet

Compilarea unei miniaplicatii se realizeaza la fel cu compilarea unei aplicatii. Pentru compilarea miniaplicatiei **FirstApplet**, folositi compilatorul Java, **javac**. La compilarea unui fisier sursa, compilatorul creeaza un fisier separat pentru fiecare clasa din program. Daca miniaplicatia are mai multe fisiere de clasa, trebuie sa apelati interpretorul Java cu numele clasei care contine metoda primara. Deoarece **FirstApplet** contine o singura declaratie de clasa, compilatorul Java va crea un singur fisier.

Crearea unui fisier HTML pentru miniaplicatie

Deorece miniaplicatiile pot fi vizualizate cu ajutorul unor programe *hypertext* specializate, cum ar fi browserele Web, trebuie sa creati un document HTML inainte de a putea utiliza miniaplicatia. In cadrul acestui document, pentru incarcarea si rularea miniaplicatiei specificate, folositi o eticheta de marcare numita APPLET. In eticheta <APPLET> se face referire la clasele Java, nu la fisierele de clasa care se termina cu extensia **.class**. Exemplul de document HTML de mai jos contine o eticheta <APPLET> care se refera la clasa **FirstApplet**, si nu la fisierul numit **FirstApplet.class**.

Cu ajutorul unui editor sau al unui procesor de texte, creati un fisier de text ASCII standard, cu urmatorul continut:

```
<HTML>
<HEAD>
<TITLE>First Java Applet</TITLE>
</HEAD>
<BODY>
<APPLET CODE="FirstApplet" width=400 height=400></APPLET>
</BODY>
</HTML>
```

Salvati acest fisier in acelasi director cu codul compilat al programului **FirstApplet**. Majoritatea documentelor HTML folosesc extensia **.html**; ar trebui sa salvati fisierul sub un nume corespunzator, cum ar fi **example.html**.

Rularea miniaplicatiei FirstApplet

Dupa crearea fisierelor necesare pentru programul **FirstApplet**, puteti rula miniaplicatia cu ajutorul unui program de vizualizare a *hypertextului*. Setul de dezvoltare Java contine un astfel de program, numit ***appletviewer***. In anumite sisteme, programul ***appletviewer*** este un instrument de lucru din linia de comanda si poate fi apelat cu numele clasei pe care vreti sa o rulati (**script pentru lansarea fisierului HTML si implicit a *applet-ului* in programul *appletviewer***).