



Príručka Java

Tvorba GUI a SWING



IT ACADEMY

IT Academy s.r.o., Tomášikova 50/A, 831 04 Bratislava, tel.: 0917/095 406, 0907/375 543

IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

Obsah:

I. Balík (package)	3
II. Rozhrania (interfaces)	12
III. Modifikátory prístupu	16
IV. Použitie rozhrania ako typu.....	22
V. Vytvorenie hlavného okna GUI aplikácie	29
VI. Správcovia rozmiestnenia (layout managers)	33
VII. Kombinácia správcov rozmiestnenia	50
VIII. GUI – spracovanie udalostí.....	53
IX. Trieda udalostí EventObject	58
X. Trieda udalostí AWTEvent.....	59
XI. Spracovanie udalostí	61
XII. JMenuitem	103
XIII. Vytvorenie toolbaru	109
XIV. Spracovanie udalosti menu a toolBaru.....	112
XV. Popup menu	122
XVI. Trieda JInfernalFrame.....	129
XVII. Trieda JTabbedPane	131
XVIII. Odporúčaná literatúra a zdroje	134

Túto príručku môžete využiť ako pomôcku pri práci s programom Java. **Príručka podlieha autorským právam a jej vlastníkom je spoločnosť IT Academy s.r.o.**

I. Balík (package)

Balík (package) je zoskupenie súvisiacich typov (triedy, rozhrania, enum), ktoré poskytuje ochranu prístupu a zabránenie konfliktov názvov.

Príklady balíkov:

- Základné triedy sa nachádzajú v balíku **java.lang**
- Triedy pre čítanie a zápis (pre vstup a výstup) sú v balíku **java.io**.

Výhody zoskupenia typov do balíka

- typy v jednom balíku spolu **súvisia**
- ak napr. majú dve triedy rovnaký názov, dajú sa **rozlíšiť** podľa balíka, ktorého sú súčasťou
- prístupové práva k typom vo vnútri vlastného balíka môžu byť **väčšie** ako prístupové práva k typom iného balíka

Vytvorenie balíka

1. Najprv treba určiť **názov balíka**.
2. Na začiatok každého zdrojového súboru patriaceho do určitého balíka treba napísať **príkaz package** nazovbalíka;

Príkaz package musí byť prvým príkazom v zdrojovom súbore. V každom súbore môže byť len jeden príkaz package.

Prístupové práva

Ak je v jednom zdrojovom súbore umiestnených viacero typov (nie vnorených), potom iba jeden z nich môže byť označený ako **public** a musí mať rovnaký názov ako je názov súboru.

Typy označené ako **public** sú prístupné z ľubovoľného balíka, typy bez označenia public t.j. **package-private** sú prístupné iba z vnútra balíku.

Balík bez názvu

Ak nie je použité kľúčové slovo **package**, potom je typ umiestnený do balíka bez názvu (default package). Balík bez názvu je vhodný iba pre malé, alebo dočasné aplikácie.

Príklad:

- vytvorenie balíka s názvom grafika, ktorý obsahuje triedy Grafika, Obdĺžnik, Kruh a rozhranie Presúvateľný:

súbor Grafika.java:

```
package grafika;  
public abstract class Grafika {  
    // ....  
}
```

súbor Presuvatelny.java:

```
package grafika;  
public interface Presuvatelny {  
    // ....  
}
```

súbor Obdlznik.java:

```
package grafika;  
public class Obdlznik extends Grafika implements  
Presuvatelny {  
    // ....  
}
```

súbor Kruh.java: package grafika;

```
public class Kruh extends Grafika implements  
Presuvatelny {  
    // ....  
}
```

Pomenovanie typov

Jednoduchý názov typu je názov typu ktorý je umiestnený za **class**, **interface**, **enum**.

Napríklad:

- **Object**
- **String**
- **Math**
- **BufferedReader**
- **FileInputStream**

Plný názov (úplný názov) typu sa skladá z názvu balíka, za ktorým nasleduje bodka a jednoduchý názov typu v balíku. Napríklad:

- java.lang.Object
- java.lang.String
- java.lang.Math
- java.io.BufferedReader
- java.io.FileInputStream

Konvencia pomenovania balíka

- názvy balíkov sa píšu iba **malými** písmenami, aby sa predišlo konfliktom s názvami tried, alebo rozhraní
- spoločnosti uvádzajú na začiatku názvov balíkov svoj obrátený **doménový názov**
- balíky v samotnom jazyku **java** začínajú reťazcami **java.** alebo **javax.**
- doménový názov z internetu **nemusí** byť platný, ak obsahuje **špeciálne znaky** (napr. pomlčku), alebo sa začína číslicou. Vtedy sa pomlčka odporúča nahradiť podčiarkovníkom, pred začiatočnú číslicu doplniť podčiarkovník a pod.

Typy, ktoré tvoria balík sa označujú ako **členy balíka**.

Použitie členov balíka vo vnútri balíka

Stačí používať jednoduché meno ako doteraz.

Napríklad: Grafika, Obdĺžnik, Kružnica.

Použitie členov balíka v inom balíku

Pri používaní členov balíku mimo balíku v ktorom sú definovaný je potrebné:

- odkazovať sa na člen pomocou úplného názvu,
- alebo importovať člen balíka,
- alebo importovať celý balík

Na importovanie balíkov, alebo členov balíka sa používa kľúčové slovo **import**.

Kľúčové slovo **import** treba v súbore umiestniť **za package** (ak existuje) a pred definíciu akéhokoľvek typu.

Príklad:

– odkazovanie pomocou úplného názvu

```
package inybalik;
```

```
public class TestovaciaTrieda {  
    public static void main(String[] args) {  
        grafika.Obdlznik o = new  
        grafika.Obdlznik();  
        // .....  
    }  
}
```

Príklad:

– importovanie člena balíka

```
package inybalik;

import grafika.Obdlznik;
import grafika.Kruh;

public class TestovaciaTrieda {
    public static void main(String[] args) {
        Obdlznik o = new Obdlznik();
        Kruh k = new Kruh();
        // .....
    }
}
```

Príklad:

– import celého balíka

```
package inybalik;

import grafika.*;

public class TestovaciaTrieda {
    public static void main(String[] args) {
        Obdlznik o = new Obdlznik();
        Kruh k = new Kruh();
        // .....
    }
}
```

Importovanie verejných vnorených typov

Predpokladajme že trieda **Obdĺžnik** ma verejný vnorené typy **Zväčšovatel.** a **Vyfarbovatel.** Tieto verejné typy môžeme importovať napr. takto: `import grafika.Obdlznik.*;`

Automatické importovanie

Prekladač do každého súboru automaticky importuje

- balík bez názvu
- balík `java.lang`
- aktuálny balík

Zdanlivá hierarchia balíkov

V java existujú napr. takéto balíky:

- `java.awt,`
- `java.awt.color,`
- `java.awt.font`

Názvy týchto balíkov začínajú rovnakým reťazcom **java.awt**, aby bolo vidno že spolu súvisia. Avšak balíky **java.awt.color** a **java.awt.font** nie sú súčasťou balíka **java.awt**.

Príkaz `import java.awt.*;` **nainportuje** všetky členy balíka **java.awt**, ale **nenainportuje** balíky **java.awt.color** a **java.awt.font**.

Pre naimportovanie všetkých troch balíkov treba zadať:

```
import java.awt.*;  
import java.awt.color.*;  
import java.awt.font.*;
```

Nejednoznačné názvy

Ak sú naimportované dva balíky, ktoré obsahujú rovnako pomenované typy, potom sa na tieto typy treba odvolávať plným menom.

Príkaz statického importu

Slúži na importovanie statických finálnych atribútov (konštánt) a statických metód pridaním slova **static**.

Napr. kód: `double r = Math.cos(Math.PI*theta);`

možno upraviť nasledovne:

```
import static java.lang.Math.PI;  
import static java.lang.Math.cos;
```

```
// neskor v kode  
double r = cos(PI*theta);
```

alebo

```
import static java.lang.Math.*;
```

```
// neskor v kode  
double r = cos(PI*theta);
```

Príkaz statického importu treba používať **opatrne**. Niekedy môže **zvyšovať čitateľnosť**, pretože netreba stále uvádzať názov triedy (dlhý matematický vzorec). Ak je ale tento príkaz použitý príliš často, potom znižuje čitateľnosť, pretože sťažuje určovanie, ku ktorej triede statický člen patrí.

Správa zdrojových a výstupných súborov

Mnohé **implementácie** platformy Java, **ukladajú** súbory v systéme súborov podľa **názvu balíka a názvu typu**.

Oddelenie zdrojových a skompilovaných súborov umožňuje ľahké zverejnenie skompilovaných súborov bez zdrojových súborov.

Cesta k skompilovaným súborom sa nachádza v systémovej premennej **CLASSPATH** (windows aj unix). Táto premenná môže obsahovať aj viacero ciest oddelených bodkočiarkou (windows), alebo dvojbodkou (unix).

Pri spustení programu sa hľadá skompilovaná trieda v aktuálnom adresári, v súbore **JAR**, ktorý obsahuje triedy platformy Java a v adresároch uvedených v premennej **CLASSPATH**.

II. Rozhrania (interfaces)

- Rozhrania slúžia na definíciu rozhrania objektov.
- Definujú **funkcie**, ktoré objekt musí mať implementované.

Napr. dvaja výrobcovia sa môžu dohodnúť na spoločnom rozhraní. Jeden výrobca bude vytvárať **triedy**, ktorých inštancie budú vedieť vykonávať funkcie definované v rozhraní. Druhý výrobca bude vedieť tieto **inštancie** používať vďaka definovanému rozhraniu.

Rozhranie je **referenčným typom**, podobne ako trieda.

Môže obsahovať:

- **konštanty**
- **signatúry metód**
- **vnorené typy**

Neobsahuje telá metód, nemožno vytvoriť inštanciu rozhrania. Rozhranie je možné implementovať v triede, alebo ho rozšíriť v inom rozhraní. Rozhranie sa uvádza kľúčovým slovom **interface**.

Príklad definície rozhrania:

```
public interface NazovRozhrania {  
    // konstanty  konstanta1 = 10;  
    String konstanta2 = "konstanta2";  
    eger kostanta3 = new eger(100);  
  
    // funkcie  
    double funkcia1(parameter1, double parameter2);  
    String funkcia2();  
    void funkcia3();  
  
    // vnorene typy enum VnorenyEnum  
    { /*.....*/ } interface Vnorenyinterface {  
    /*.....*/ }  
    class VnutornaTrieda { /*.....*/ }  
}
```

Za signatúrou každej metódy je **bodkočiarka**.

Použitie rozhrania = implementácia rozhrania triedou:

Ak trieda implementuje rozhranie, vtedy sa názov rozhrania uvádza za kľúčovým slovom **implements**. Tá trieda, ktorá implementuje rozhranie, musí implementovať aj **všetky metódy rozhrania**.

Príklad:

```
public class TriedaImplRozhranie implements
NazovRozhrania {
    public double funkcia1(parameter1, double
parameter2) {
        return 1;
    }
    public String funkcia2() {
        return konstanta2; // return
NazovRozhrania.konstanta2;
    }
    public void funkcia3() {
        // .....
    }
}
```

Trieda môže implementovať viacero rozhraní. Vtedy sú za kľúčovým slovom **implements** uvedené názvy implementovaných rozhraní **oddelené čiarkou**.

Príklad

```
public class Trieda implements Rozhranie1,
Rozhranie2 {
    public double funkcia1(parameter1, double
parameter2) {
        // .....
    }
}
```

Rozhranie rozširujúce iné rozhrania

Rozhranie môže byť doplnené (resp. rozšírené) iným rozhraním. V definícii rozhrania, ktoré rozširuje iné rozhranie sa rozširované rozhranie uvádza za kľúčovým slovom **extends**.

Ak rozhranie rozširuje viacero rozhraní, tak sú tiež vymenované za kľúčovým slovom **extends**. Názvy rozhraní za slovom **extends** sú oddelené čiarkou.

Príklad:

```
public interface NoveRozhranie extends Rozhranie1,
Rozhranie2 {
    // konstanty
    double PI = 3.14159265358979323846;
    // signatury funkcií
    double dalsiaFunkcia1(double parameter);
    dalsiaFunkcia2(String parameter);
}
```


III. Modifikátory prístupu

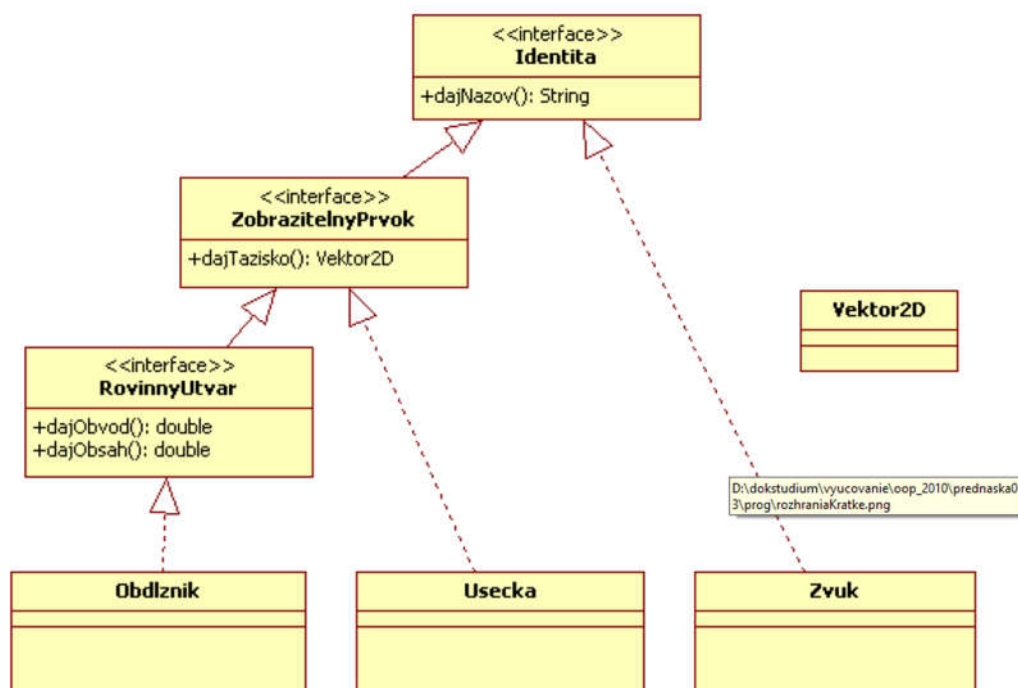
Modifikátory prístupu pred definíciou rozhrania (pred kľúčovým slovom interface)

- **public** – rozhranie možno používať na ľubovoľnom mieste
- **bez uvedenia modifikátora** – rozhranie je možné používať iba v tom balíku, v ktorom je uvedené

Modifikátory prístupu v tele rozhrania

- Všetky konštanty definované v rozhraní sú implicitne typu **public static final**. Preto možno tieto modifikátory vynechať.
- Všetky metódy definované v rozhraní sú implicitne typu **public**. Takže možno tento modifikátor vynechať.

(Iné modifikátory (**private**, **protected**) nemožno uviesť)



Obr. 1 Príklad

Identita.java

```
public interface Identita {  
    String dajNazov();  
}
```

ZobrazitelnyPrvok.java

```
public interface ZobrazitelnyPrvok extends Identita  
{  
    Vektor2D dajTazisko();  
}
```

RovinnnyUtvar.java

```
public interface RovinnnyUtvar extends  
ZobrazitelnyPrvok {  
    double dajObvod();  
    double dajObsah();  
}
```

Obdlznik.java

```
public class Obdlznik implements RovinnnyUtvar {  
    private String nazov;  
    private Vektor2D lavyHorny;  
    private Vektor2D velkost;  
    public Obdlznik(String nazov, double lavy,  
double horny,  
double sirka, double  
vyska) {  
        this.nazov = nazov;  
        lavyHorny = new Vektor2D(lavy, horny);  
        velkost = new Vektor2D(sirka, vyska);  
    }  
}
```

```
public String dajNazov() {
    return nazov;
}
public Vektor2D dajTazisko() {
    double x =
lavyHorny.dajX()+(velkost.dajX()/ 2);
    double y =
lavyHorny.dajY()+(velkost.dajY()/ 2);
    return new Vektor2D(x, y);
}
public double dajObvod() {
    return 2 *(velkost.dajX()+ velkost.dajY());
}
public double dajObsah() {
    return velkost.dajX()* velkost.dajY();
}
public boolean jeStvorec() {
    return velkost.dajX() == velkost.dajY();
}
}
```

Usecka.java

```
Usecka implements ZobrazitelnyPrvok {
    private String nazov;
    private Vektor2D zaciatok;
    private Vektor2D koniec;
    public Usecka(String nazov, double x1, double
y1,
                                double x2, double y2) {
        this.nazov = nazov;
        zaciatok = new Vektor2D(x1, y1);
        koniec = new Vektor2D(x2, y2);
    }
    public String dajNazov() {
        return nazov;
    }

    public Vektor2D dajTazisko() {
        double x = priemer(zaciatok.dajX(),
koniec.dajX());
        double y = priemer(zaciatok.dajY(),
koniec.dajY());
        return new Vektor2D(x, y);
    }
    private double priemer(double a, double b) {
        return (a + b)/ 2;
    }
}
```

Zvuk.java

```
public class Zvuk implements Identita {  
    private String nazov;  
    public Zvuk(String nazov) {  
        this.nazov = nazov;  
    }  
    public String dajNazov() {  
        return nazov;  
    }  
    public void start() {  
        //.....  
    }  
    public void stop() {  
        //.....  
    }  
}
```

Vektor2D.java

```
public class Vektor2D {
    private double x;
    private double y;
    public Vektor2D(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double dajX() {
        return x;
    }
    public void nastavX(double x) {
        this.x = x; }
    public double dajY() {
        return y;
    }
    public void nastavY(double y) {
        this.y = y;
    }
}
```

IV. Použitie rozhrania ako typu

Rozhrania patria medzi **referenčné typy**. Názov rozhrania možno použiť všade tam, kde je možné použiť názov ľubovoľného typu.

Možno definovať referenčnú **premennú** typu rozhranie. Takejto premennej môžeme priradiť objekt, ktorý je inštanciou triedy implementujúcou príslušné rozhranie.

Pomocou premennej typu rozhranie možno nad objektom volať **funkcie**, ktoré sú v rozhraní definované (iné funkcie sa volať nedajú).

Príklad (pokračovanie):

```
public static void main(String[] args) {  
    Obdlznik o = new Obdlznik("obdlznik1", 40,50,  
100,110);  
    Usecka u = new Usecka("usecka1", 10,15,20,25);  
    Zvuk z = new Zvuk("zvuk1");  
  
    Identita io = o; // OK  
    Identita iu = u; // OK  
    Identita iz = z; // OK  
  
    ZobrazitelnyPrvok zo = o; // OK  
    ZobrazitelnyPrvok zu = u; // OK  
    //    ZobrazitelnyPrvok    zz    =    z;    CHYBA  
(neimplementovane rozhranie)  
  
    RovinnyUtvor ro = o; // OK  
    // RovinnyUtvor ru = u; CHYBA (neimplementovane  
rozhranie)  
    // RovinnyUtvor rz = z; CHYBA (neimplementovane  
rozhranie)
```

```
io.dajNazov(); // OK
// io.dajTazisko(); CHYBA(funkcia nie je v
rozhrani definovana)
// io.dajObsah(); CHYBA(funkcia nie je v
rozhrani definovana)
// io.jeStvorec(); CHYBA(funkcia nie je v
rozhrani definovana)

zo.dajNazov(); // OK
zo.dajTazisko(); // OK
// zo.dajObsah(); CHYBA(funkcia nie je v
rozhrani definovana)
// zo.jeStvorec(); CHYBA(funkcia nie je v
rozhrani definovana)

ro.dajNazov(); // OK
ro.dajTazisko(); // OK
ro.dajObsah(); // OK
// ro.jeStvorec(); CHYBA(funkcia nie je v
rozhrani definovana)
o.dajNazov(); // OK
o.dajTazisko(); // OK
o.dajObsah(); // OK
o.jeStvorec(); // OK
}
```


Zoznámenie s knižnicou Swing

Aplikácie s grafickým používateľským rozhraním

GUI - graphical user interface

Knižnice

- AWT
- pre zobrazenie využíva natívne **API** operačného systému
- na **rôznych** platformách rôzny vzhľad
- **rýchlejšie**

Swing

- vlastné zobrazenie **komponent**
- **rovnaký** vzhľad na rôznych platformách
- **pomalšie** ako AWT
- **novšia** ako AWT
- Swing využíva niektoré **triedy** z AWT, ale nemá triedy zobrazujúce prvky z oboch knižníc

Balíky



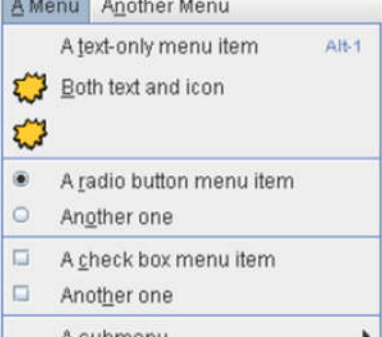
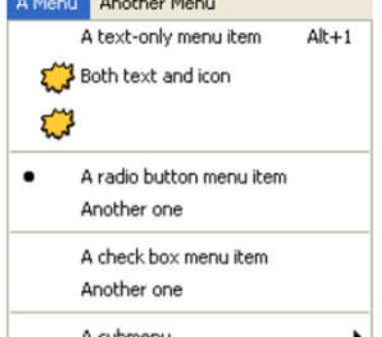
- java.awt
- java.awt.event
- javax.swing
- javax.swing.event

Swing

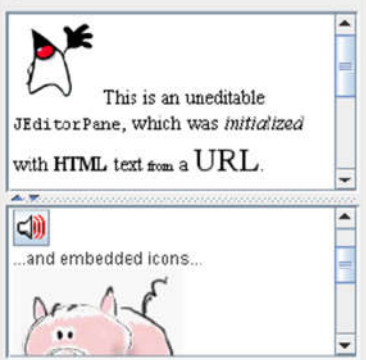

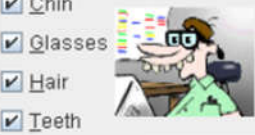



JPanel – pre zobrazenie sady súvisiacich komponent




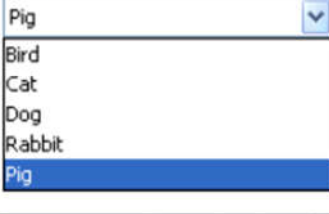








JFrame – používame pre vytvorenie hlavného okna





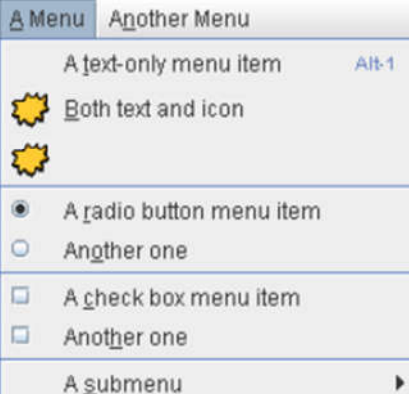

Príklady Komponent a vzhľad motívu


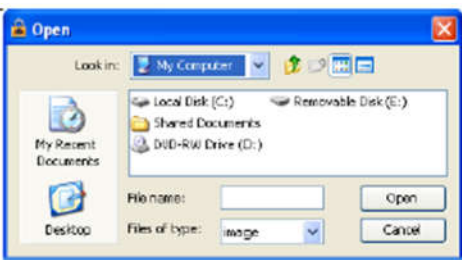
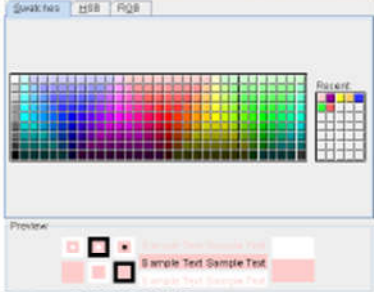



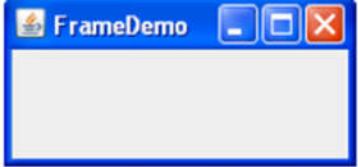
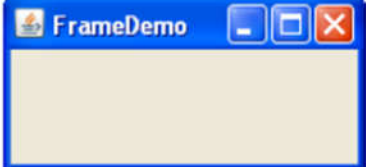
JTable	<table><tr><th>Host</th><th>User</th><th>Password</th><th>Last Modified</th></tr><tr><td>Bucca Games</td><td>Freddy</td><td>123456789</td><td>Mar 15, 2006</td></tr><tr><td>zabie</td><td>chabod</td><td>123456789</td><td>Mar 6, 2006</td></tr><tr><td>Sun Developer</td><td>fraz@hotmail.co</td><td>AsaM541fBz</td><td>Feb 22, 2006</td></tr><tr><td>Hulkam Seeds</td><td>shams@gmail</td><td>blz/ACF78!</td><td>Jul 29, 2005</td></tr><tr><td>Pacific Zoo Shop</td><td>seal@hotmail.c...</td><td>vbl/124%z</td><td>Feb 22, 2006</td></tr></table>	Host	User	Password	Last Modified	Bucca Games	Freddy	123456789	Mar 15, 2006	zabie	chabod	123456789	Mar 6, 2006	Sun Developer	fraz@hotmail.co	AsaM541fBz	Feb 22, 2006	Hulkam Seeds	shams@gmail	blz/ACF78!	Jul 29, 2005	Pacific Zoo Shop	seal@hotmail.c...	vbl/124%z	Feb 22, 2006	<table><tr><th>Host</th><th>User</th><th>Password</th><th>Last Modified</th></tr><tr><td>Bucca Games</td><td>Freddy</td><td>123456789</td><td>Mar 15, 2006</td></tr><tr><td>zabie</td><td>chabod</td><td>123456789</td><td>Mar 6, 2006</td></tr><tr><td>Sun Developer</td><td>fraz@hotmail.co</td><td>AsaM541fBz</td><td>Feb 22, 2006</td></tr><tr><td>Hulkam Seeds</td><td>shams@gmail.com</td><td>blz/ACF78!</td><td>Jul 29, 2005</td></tr><tr><td>Pacific Zoo Shop</td><td>seal@hotmail.com</td><td>vbl/124%z</td><td>Feb 22, 2006</td></tr></table>	Host	User	Password	Last Modified	Bucca Games	Freddy	123456789	Mar 15, 2006	zabie	chabod	123456789	Mar 6, 2006	Sun Developer	fraz@hotmail.co	AsaM541fBz	Feb 22, 2006	Hulkam Seeds	shams@gmail.com	blz/ACF78!	Jul 29, 2005	Pacific Zoo Shop	seal@hotmail.com	vbl/124%z	Feb 22, 2006
Host	User	Password	Last Modified																																															
Bucca Games	Freddy	123456789	Mar 15, 2006																																															
zabie	chabod	123456789	Mar 6, 2006																																															
Sun Developer	fraz@hotmail.co	AsaM541fBz	Feb 22, 2006																																															
Hulkam Seeds	shams@gmail	blz/ACF78!	Jul 29, 2005																																															
Pacific Zoo Shop	seal@hotmail.c...	vbl/124%z	Feb 22, 2006																																															
Host	User	Password	Last Modified																																															
Bucca Games	Freddy	123456789	Mar 15, 2006																																															
zabie	chabod	123456789	Mar 6, 2006																																															
Sun Developer	fraz@hotmail.co	AsaM541fBz	Feb 22, 2006																																															
Hulkam Seeds	shams@gmail.com	blz/ACF78!	Jul 29, 2005																																															
Pacific Zoo Shop	seal@hotmail.com	vbl/124%z	Feb 22, 2006																																															
JTree																																																		
JMenu																																																		

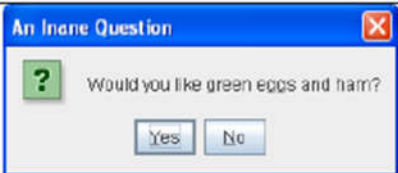





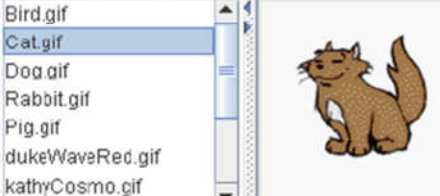
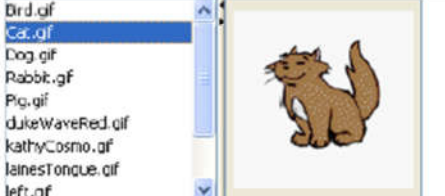


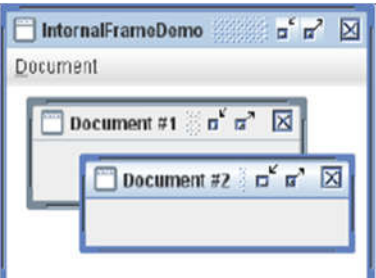
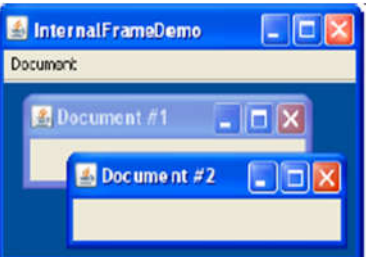
Obr. 2 Komponent a vzhľad motívu

JEditorPane JTextPane		
JCheckBox		
JRadioButton		

JList		
JComboBox		
JSlider		
JSpinner		
JProgressBar		
JToolTip		

JTable		
JTree		
JMenu		

JFileChooser		
JColorChooser		
JSeparator		
JFrame		

JDialog		
JApplet		
JPanel		
JScrollPane		
JToolBar		
JSplitPane		
JTabbedPane		
JInternalFrame		

V. Vytvorenie hlavného okna GUI aplikácie

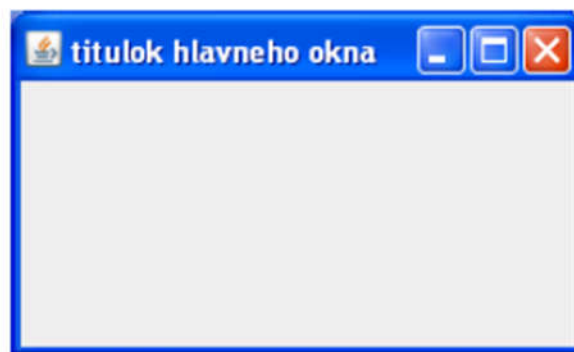
```
import javax.swing.JFrame;  
import javax.swing.SwingUtilities;  
  
public class PrveOkno {  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(new Runnable() {  
            public void run() {  
                JFrame okno = new JFrame();  
                okno.setDefaultCloseOperation(JFrame.  
e.EXIT_ON_CLOSE);  
                okno.setSize(200,100);  
                okno.setVisible();  
            }  
        });  
    }  
}
```



Obr. 3 Vytvorenie hlavné okna GUI aplikácie

```
import javax.swing.JFrame;  
import javax.swing.SwingUtilities;
```

```
public class HlavneOkno extends JFrame {  
    public HlavneOkno() {  
        setTitle("titulok hlavneho okna");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(new Runnable() {  
            public void run() {  
                HlavneOkno okno = new HlavneOkno();  
                okno.setSize(250, 150);  
                okno.setVisible();  
            }  
        });  
    }  
}
```



Obr. 4 Titulok hlavného okna

Rozmiestnenie prvkov bez použitia správcu rozmiestnenia – väčšinou nevhodný spôsob

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

public class BezPouzitiaSpravcovRozmiestnenia
extends JFrame {

    public BezPouzitiaSpravcovRozmiestnenia() {
        setTitle("priklad rozmiestnenia bez spravcu
rozmiestnenia");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);

        Container pane = getContentPane();
        pane.setLayout(null);    // odstranenie
defaultneho spravcu

        JButton b1 = new JButton("aaaa");
        b1.setLocation(10, 20);
        b1.setSize(70, 25);
        pane.add(b1);

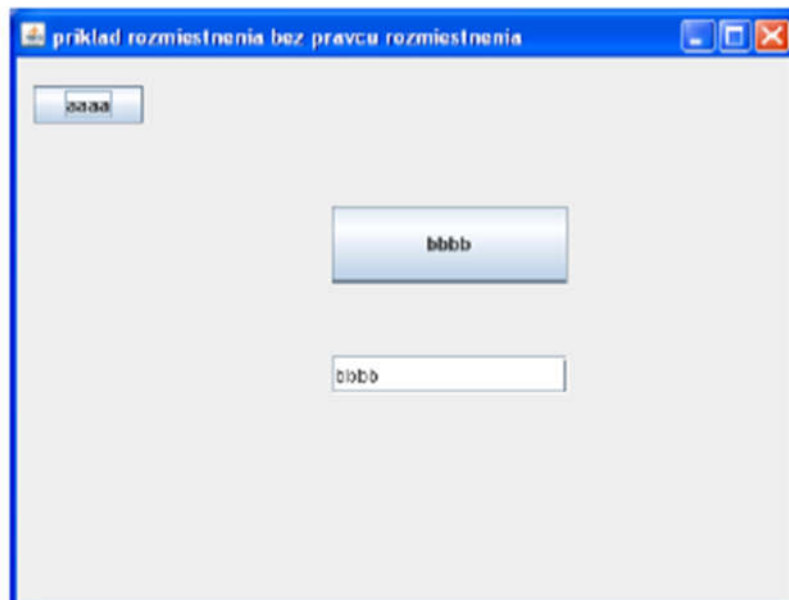
        JButton b2 = new JButton("bbbb");
        b2.setBounds(200, 100, 150, 50);
        pane.add(b2);
    }
}
```



```

        JTextField text = new JTextField("bbbb");
        text.setBounds(200, 200, 150, 25);
        pane.add(text);
        setSize(500, 400);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater (new Runnable()
        {
            public void run() {
                BezPouzitiaSpravcovRozmiestnenia
                frame =
                    new
                    BezPouzitiaSpravcovRozmiestnenia();
                frame.setVisible();
            }
        });
    }
}

```



Obr. 5 Príklad rozmiestnenia bez správcu rozmiestnenia

VI. Správcovia rozmiestnenia (layout managers)

Správcovia rozmiestnení sú **triedy** používané pre riadenia **rozmiestnenia** a **veľkosti** jednotlivých komponent pridávaných do kontajneru.

Správcovia rozmiestnenia určujú **polohu a veľkosť** komponent v kontajneri podľa minimálnej, preferovanej a maximálnej veľkosti komponent. Tiež väčšinou určujú veľkosť minimálnej a preferovanej veľkosti kontajneru podľa určitých veľkostí komponent.

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

public class VelkostiSwingPrvkov {
    public static void main(String[] args) {
        JTextField component = new
        JTextField("vstup");
        // JButton component = new JButton("stlac");
        // JLabel component = new JLabel("nazov");
        Dimension min = component.getMinimumSize();
        System.out.println("min      :      "+min.width+",
        "+min.height);

        Dimension pref = component.getPreferredSize();
        System.out.println("pref:      "+pref.width+",
        "+pref.height);

        Dimension max = component.getMaximumSize();
        System.out.println("max      :      "+max.width+",
        "+max.height);
    }
}
```

Správcovia rozmiestení umožňujú určiť **vzdialenosť** (voľné miesto) medzi komponentmi navzájom a tiež medzi komponentmi a okrajom kontajneru.

Pridávanie komponentov do kontajneru – preťažená metóda
Container.add

```
Component add(Component comp)
Component add(Component comp, index)
void add(Component comp, Object constras)
void add(Component comp, Object constras, index)
Component add(String name, Component comp)
```

Správca rozmiestnenia FlowLayout

Usporiada komponenty do riadkov **zľava doprava** a **zhora dole**.

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

public class FlowLayoutFrame extends JFrame {
    public FlowLayoutFrame() {
        setTitle("priklad na pouzitie FlowLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);

        Container pane = getContentPane();
        pane.setLayout(new FlowLayout());
    }
}
```

```
//                               pane.setLayout(new
FlowLayout(FlowLayout.LEFT,10,5));

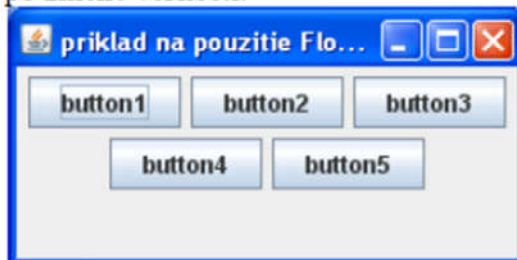
pane.add(new JButton("button1"));
pane.add(new JButton("button2"));
pane.add(new JButton("button3"));
pane.add(new JButton("button4"));
pane.add(new JButton("button5"));
pack(); // nastaví preferovanú veľkosť
// setSize(400,300);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            FlowLayoutFrame frame = new
            FlowLayoutFrame();
            frame.setVisible();
        }
    });
}
}
```

po spustení:



po zmene veľkosti:



Obr. 6 Príklad na použitie FlowLayout

Obmedzujúce pravidlá

FlowLayout **nepoužíva** obmedzujúce pravidlá.

Rozmery komponent

Komponenty sú zobrazované v preferovaných veľkostiach bez ohľadu na veľkosť kontajneru.

Umiestnenie komponent

Zľava doprava, zhora dole. **Poradie** komponent je dané poradím ich **vkladania** do kontajneru. Do riadku sa vloží toľko komponentov, koľko dovoľuje šírka kontajneru.

Veľkosť kontajneru

Preferovaná šírka je daná **súčtom**:

- šírka ľavej a pravej výplne kontajneru
- súčet šírky vodorovných medzier
- súčet uprednostňovaných širok všetkých komponentov

Preferovaná výška je daná výškou najvyššej komponenty + horná a dolná výplň kontajneru. **Minimálna veľkosť** sa vypočíta podobne ako preferovaná, ale namiesto preferovaných rozmerov komponentov sa používajú **minimálne rozmery**.

Správca rozmiestnenia GridLayout

- rozdeľuje priestor na mriežku „**buniek**“, rovnomerne rozmiestnenej po celej ploche kontajneru. Každá bunka má **rovnakú** veľkosť.

Príklad:

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

public class GridLayoutFrame extends JFrame {
    public GridLayoutFrame() {
        setTitle("príklad na pouzitie GridLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    Container pane = getContentPane();
    pane.setLayout(new GridLayout(0,2 /*2,2*/
/*,5,20*/));
    pane.add(new JButton("prvy"));
    pane.add(new JButton("druhy"));
    pane.add(new JButton("treti je dlhy -----
dlhy "));
    pane.add(new JButton("stvrty"));
    pane.add(new JButton("piaty"));
    pane.add(new JButton("siesty"));
    pane.add(new JButton("siedmy"));
    pane.add(new JButton("osmy"));
    pack();
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            GridLayoutFrame frame = new
            GridLayoutFrame();
            frame.setVisible();
        }
    });
}
```



Obr. 7 Príklad na použitie GridLayout

Obmedzujúce pravidlá

GridLayout nepoužíva obmedzujúce pravidlá.

Rozmery komponent

Každý komponent má rovnakú veľkosť. Rozmery komponent závisia od **veľkosti kontajnera** a **veľkosti medzier** medzi komponentmi a veľkosti okrajov. **GridLayout** ignoruje minimálnu, preferovanú aj maximálnu veľkosť komponent pri určovaní ich rozmerov.

Umiestnenie komponent

Komponenty sú umiestňované zľava doprava a zhora dole, v poradí v ktorom sú do kontajneru pridávané.

Veľkosť kontajneru

Pri výpočte minimálnej (a preferovanej) výšky kontajneru sa nájde najväčšia výška a šírka zo všetkých minimálnych (preferovaných) výšok a šírok komponentov.

Táto výška a šírka sa vynásobí počtom riadkov, resp. stĺpcov, pripočítajú sa medzery medzi komponentmi a okraje.

Správca rozmiestnenia BorderLayout

Má **5 oblastí**: horný, dolný, ľavý a pravý okraj, stred

Príklad:

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

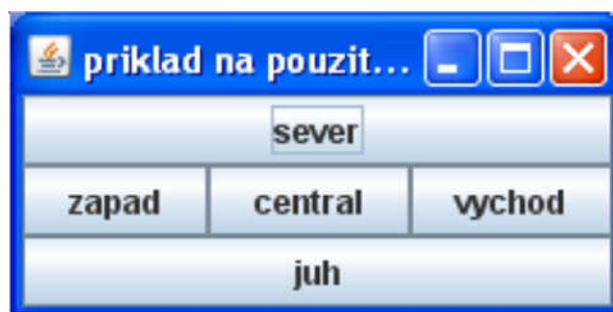
public class BorderLayoutFrame extends JFrame {
    public BorderLayoutFrame() {
        setTitle("príklad na použitie
BorderLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        pane.add(new JButton("sever"),
BorderLayout.NORTH);
        pane.add(new JButton("zapad"),
BorderLayout.WEST);
        pane.add(new JButton("central"),
BorderLayout.CENTER);
        pane.add(new JButton("vychod"),
BorderLayout.EAST);
        pane.add(new JButton("juh"),
BorderLayout.SOUTH);
        pack();
    }
}
```



```

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            BorderLayoutFrame frame=new
            BorderLayoutFrame();
            frame.setVisible();
        }
    });
}

```



Obr. 8 Po spustení



Obr. 9 Po zväčšení

Obmedzujúce pravidlá

-určujú **jednu** z piatich oblastí. Bez použitia obmedzujúceho pravidla je komponent pridaný do **strednej** oblasti (v ktorej sa zobrazí naposledy pridaný komponent).

Rozmery komponent

- **horná a dolná oblasť**
 - ✓ výška – použitá je preferovaná výška komponenty
 - ✓ šírka – použitá je šírka kontajneru mínus okraje
- **pravá a ľavá oblasť**
 - ✓ výška = výška kontajneru mínus okraje mínus výška hornej a dolnej časti
 - ✓ šírka = preferovaná šírka komponenty
- **stredná časť**

Pre komponent je použitá celá zvyšná oblasť.

Ak sú rozmery kontajnera príliš malé, tak sa komponenty prekrývajú.

Umiestnenie komponent

podľa obmedzujúceho pravidla: **NORTH** – horná časť, **SOUTH**- dolná časť, **EAST** – pravá časť, **WEST** – ľavá časť, **CENTER** – stredná časť

Veľkosť kontajneru

Minimálna šírka je šírka okrajov plus maximum z:

- minimálna šírka horného komponentu
- súčet minimálnych širok komponentov v pravej, ľavej a strednej časti a medzier
- minimálna šírka dolného komponentu

Analogicky je vypočítaná minimálna výška, preferovaná šírka a výška.

Správca rozmiestnenia CardLayout

- umožňuje pridať **mnoho komponent**, ale v jednom čase vždy zobrazuje iba **jednu**. Zobrazovaný komponent je možné zmeniť volaním metód `firs()`, `last()`, `next()`, `previous()`, `show()` triedy **CardLayout**.

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class CardLayoutFrame extends JFrame {  
    CardLayout layout;  
    Container pane;
```

```
public CardLayoutFrame() {
    setTitle("priklad na pouzitie
CardLayout");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
    pane = getContentPane();
    layout = new CardLayout();
    setLayout(layout);
    pane.add(new JLabel("label"), "label" );
    pane.add(new JTextField("textField"),
"text" );
    pane.add(new JRadioButton()
,
"radiobutton");
    pane.add(new JButton("button"), "button");
    pack();
}

public void startCycle() {
    new Timer(1000, new ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            layout.next(pane);
        }
    }).start();
}
```

```

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            CardLayoutFrame frame = new
            CardLayoutFrame();
            frame.setVisible();
            frame.startCycle();
        }
    });
}
}

```



Obr. 10 Správca rozmiestnenia CardLayout

Obmedzujúce pravidlá

-pre **identifikovanie** komponentov v metóde show(), treba každému komponentu priradiť názov typu **String**.

Rozmery komponent

Veľkosť komponentov je daná veľkosťou kontajneru mínus okraje.

Umiestnenie komponent

V celom kontajneri (okrem okrajov).

Veľkosť kontajneru

Minimálna (preferovaná) veľkosť kontajneru je daná **najväčšou minimálnou** (preferovanou) šírkou a najväčšou minimálnou (preferovanou) výškou všetkých komponentov.

Správca rozmiestnenia GridBagLayout

- najflexibilnejší a najzložitejší **správca**. Je dostatočne flexibilný. Delí kontajner do mriežky buniek, ale omnoho flexibilnejšie ako **GridLayout**. Komponenty môžu obsadzovať viacero buniek. **Riadky** môžu mať rôznu výšku, **stĺpce** rôznu šírku.

Obmedzujúce pravidlá umožňujú prispôsobiť výšku a šírku každého komponentu. Ku každému komponentu je pridružená inštancia obsahujúca obmedzujúce pravidlá (tie sú dané 11 hodnotami).

Príklad:

```
import java.awt.*;
import javax.swing.*;
public class GridBagLayoutFrame extends JFrame {
    public GridBagLayoutFrame() {
        setTitle("príklad na použitie
GridBagLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();
        pane.setLayout(new GridBagLayout());

        GridBagConstras c;
        c=new GridBagConstras();
```

```
c.gridx = 0;
c.gridy = 0;
c.fill=GridBagConstraints.BOTH;                                //
NONE,HORIZONTAL,VERTICAL
pane.add(new JButton("button1"), c);

c.gridx = 1;
// c.gridy = 0;
pane.add(new JButton("button2"), c);

c.gridx = 2;
c.insets = new Insets(10,10,10,20);
pane.add(new JButton("button3"), c);

c.gridx = 0;
c.gridy = 1;
c.gridwidth = 3;
c.insets = new Insets(0,0,0,0);
pane.add(new JButton("button4"), c);

c.gridx = 0;
c.gridy = 2;
c.gridwidth = GridBagConstraints.REMAINDER;
pane.add(new JButton("button5"), c);

c.gridx = GridBagConstraints.RELATIVE;
c.gridy = 3;
c.gridwidth = 1;
pane.add(new JButton("buttonA"), c);
pane.add(new JButton("buttonB"), c);
pane.add(new JButton("buttonC"), c);
```

// vahy sa prejavia pri zmene veľkosti
hlavneho okna

```
c.gridy = 4;
c.weightx = 0.5;
pane.add(new JButton("aaaa"), c);
c.weightx = 0;
pane.add(new JButton("bbbb"), c);
c.weightx = 1;
pane.add(new JButton("cccc"), c);

c.gridx=2;
c.gridy=6;
c.insets = new Insets(10,0,0,0);
c.fill=GridBagConstras.BOTH;
pane.add(new JButton("okraj"), c);
pack();
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            GridBagLayoutFrame      frame=new
            GridBagLayoutFrame();
            frame.setVisible();
        }
    });
}
}
```




Obr. 11 Správca rozmiestnenia GridBagLayout

Obmedzujúce pravidlá

Nepodporuje obmedzujúce pravidlá. Používa iba **zarovnanie** (alignment).

Trieda Box

- poskytuje metódy pre **BoxLayout**.

Niektoré **statické výrobné metódy**:

- `createRigidArea()` - pevná oblasť – zaberá pevnú plochu v kontajneri
- `createHorizontalGlue()`, `createVerticalGlue()` – tmel (názov nevystihuje význam).

Objekty typu **Glue** sa používajú na vyplnenie nadbytočného priestoru. Oddeluje komponenty od seba tak ďaleko ako je to možné. Jeho minimálna veľkosť je **nulová**.

-
- `createHorizontalStrut()`, `createVerticalStrut()` – rozperry.

Rozperry sú podobné pevným oblastiam. Definujú ale iba **jeden** rozmer. Druhý – nedefinovaný rozmer má hodnotu nula pre minimálnu a preferovanú veľkosť a pre maximálnu veľkosť je použitá veľmi veľká hodnota. To môže niekedy spôsobiť **problémy**. Vtedy je lepšie použiť pevnú oblasť.

VII. Kombinácia správcov rozmiestnenia

Príklad:

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

public class KombinaciaSpravcovRozhrani extends
JFrame {
    public KombinaciaSpravcovRozhrani() {
        setTitle("kombinacia spravcov rozhrani");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        pane.add(createHeader(),
BorderLayout.NORTH);
        pane.add(createLeftMenu(),
BorderLayout.WEST);
        pane.add(createTextArea(),
BorderLayout.CENTER);
        pane.add(createButtonPanel(),
BorderLayout.SOUTH);
        pack();
    }
}
```

```
private JComponent createHeader() {
    JLabel label =
        new JLabel("Kombinacia viacerych
spravcov rozhrani");
    label.setFont(new Font("Times New Roman",
Font.ITALIC, 24));
    label.setForeground(Color.blue);
    return label;
}

private JComponent createLeftMenu() {
    JPanel menu = new JPanel();
    menu.setLayout(new GridLayout(0,1));
    menu.add(new JButton("kopiruj"));
    menu.add(new JButton("hrube pismo"));
    menu.add(new JButton("vyfarbi"));
    menu.add(new JButton("vymaz"));

    JPanel leftPanel = new JPanel();
    leftPanel.setLayout(new BorderLayout());
    leftPanel.add(menu, BorderLayout.NORTH);
    return leftPanel;
}

private JComponent createTextArea() {
    return new JScrollPane(new JTextArea(10,50)
);
}

private JComponent createButtonPanel() {
    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new
```

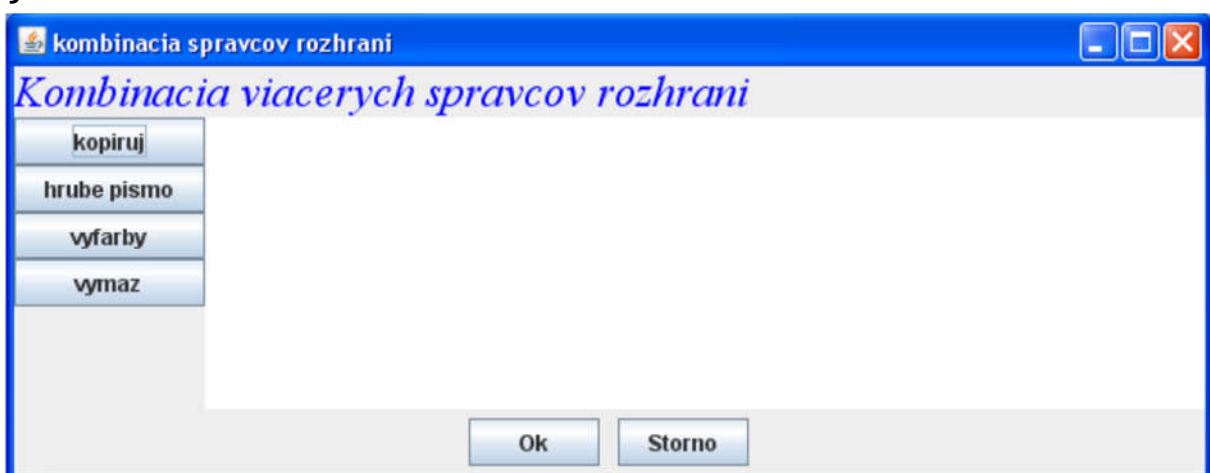
```

GridLayout(1,2,10,0));
    buttonPanel.add(new JButton("Ok"));
    buttonPanel.add(new JButton("Storno"));

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new FlowLayout());
    bottomPanel.add(buttonPanel);
    return bottomPanel;
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            KombinaciaSpravcovRozhrani frame =
                new
                    KombinaciaSpravcovRozhrani(
                );
            frame.setVisible();
        }
    });
}
}

```



Obr. 12 Kombinácia viacerých správcov rozhraní

VIII. GUI – spracovanie udalostí

GUI aplikácie využívajú **model** udalosťami riadeného programovania.

Udalosti vznikajú napr. pri stlačení klávesy, posunom myši, voľbou príkazu v menu. Po vzniku udalosti aplikácia reaguje určitou činnosťou. Medzi tým je aplikácia väčšinou nečinná.

GUI aplikácia väčšinou na začiatku (po spustení) vykoná svoju **inicializáciu**, potom čaká na výskyt udalostí, na ktoré bude reagovať.

Zdrojom udalostí v našich programoch budú **objekty**. Tieto objekty generujú **udalosti** (napr. pri stlačení klávesu).

Reakcie na vzniknuté udalosti budú vykonávané metódami **iných** objektov.

Objekty ktoré generujú udalosti, obsahujú zoznam objektov, ktorým budú oznamovať, že nastala určitá udalosť. Oznámenie prebieha zavolaním určitej metódy nad každým objektom v zozname. Vykonávaním týchto metód aplikácia reaguje na vzniknuté udalosti.

Objekty generujúce udalosti, obsahujú metódy pre pridávanie a odoberanie objektov do/zo zoznamu, ktorým budú oznamovať vznik udalosti.

Pre súvisiace typy udalostí sú definované rozhrania. Tieto rozhrania definujú metódy, ktoré sú volané pri vzniku udalosti. Objekt ktorý implementuje niektoré z týchto rozhraní nazývame **poslucháč** (listener).

Rozhranie **KeyListener** definuje **3 metódy**, ktoré súvisia s udalosťami vstupu z klávesnice:

```
public interface KeyListener {  
    void keyPressed(KeyEvent e)  
    void keyReleased(KeyEvent e)  
    void keyTyped(KeyEvent e)  
}
```

Metódy definované v týchto rozhraniach majú jeden **vstupný parameter**, ktorý obsahuje **informácie o udalosti** (napr. referenciu na zdroj udalosti, ktorý kláves bol stlačený).

Ak chceme reagovať na udalosti súvisiace so vstupom z klávesnice, musíme vytvoriť objekt implementujúci rozhranie **KeyListener**. Potom takýto objekt musíme pridať napríklad do zoznamu inštancie triedy **JTextField** pomocou metódy **addKeyListener()**.

Napríklad:

```
instanciaTextField.addKeyListener(instanciaKeyListener);
```

Inštancia triedy **JTextField** bude pre nás **zdroj udalostí**. Tieto udalosti budú vznikať vtedy, ak bude táto inštancia prijímať vstup z klávesnice.

Po vzniku udalosti, inštancia **JTextField** zavolá nad objektom pridaným pomocou **addKeyListener()** príslušnú metódu definovanú v rozhraní **KeyListener**.

Rozhrania definujúce metódy pre spracovanie udalostí, často definujú viacero metód. Ak potrebujeme obsluhovať (reagovať na) iba niektoré udalosti, potom nám stačí zmysluplne definovať **kód** iba v niektorých metódach.

Kompilátor však vyžaduje **implementáciu** všetkých metód definovaných v rozhraní.

Aby sme kvôli kompilátoru nemuseli definovať prázdne metódy, existujú triedy ktoré sa nazývajú **adaptéry**.

Adaptéry existujú pre tie rozhrania, ktoré definujú **viacero** metód. Adaptér implementuje **všetky** metódy príslušného rozhrania tak, že tieto metódy nič nevykonávajú.

Ak potrebujeme **zmysluplne** implementovať iba **niektoré** z metód definovaných v rozhraní, potom je výhodné dediť od adaptéru a v podtriede prekryť iba potrebné metódy.

Príklad – implementácia rozhrania:

```
public class ProcessKeyEvent implements KeyListener
{
    // metóda definovaná iba preto, aby prekladač
    // preložil túto triedu
    public void keyPressed(KeyEvent e) {
    }

    // metóda definovaná iba preto, aby prekladač
    // preložil túto triedu
    public void keyTyped(KeyEvent e) {
    }
}
```



```

public void keyReleased(KeyEvent e) {
    // tu bude kód pre obsluhu udalosti
}

```

Príklad – použitie adaptéra:

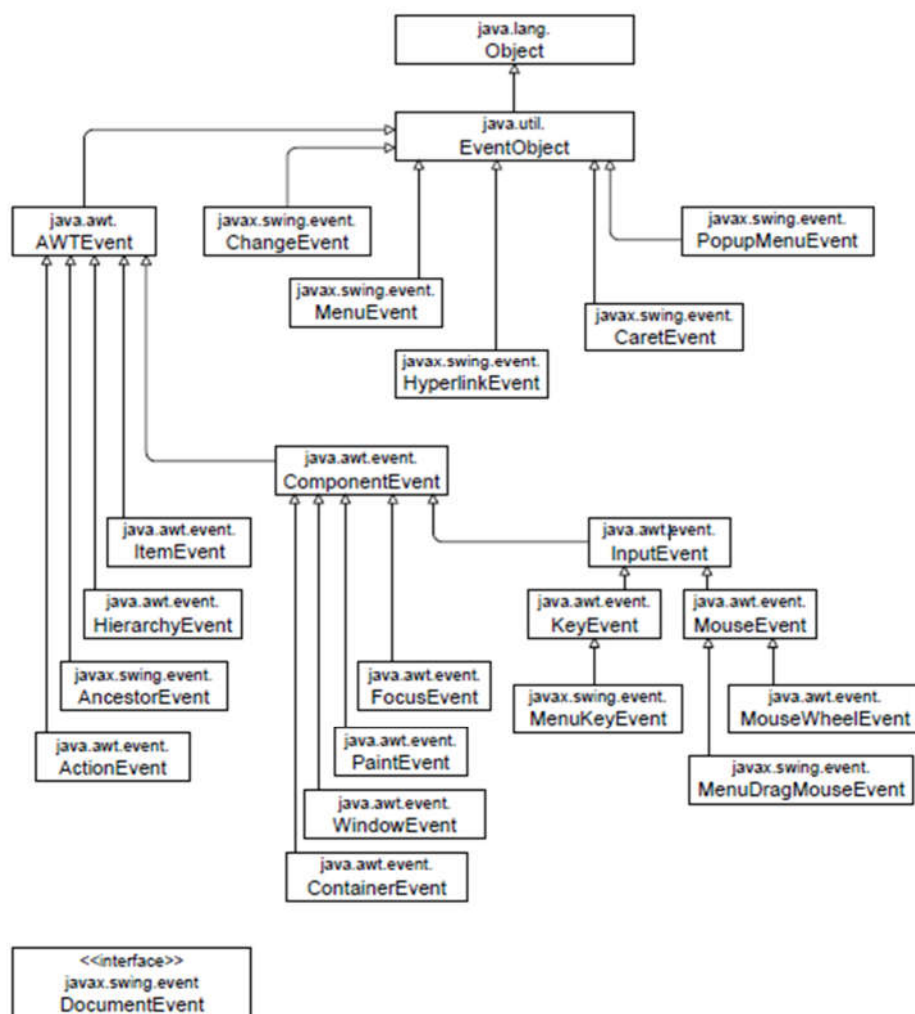
```

public class ProcessKeyEvent extends KeyAdapter {
    public void keyReleased(KeyEvent e) {
        // tu bude kód pre obsluhu udalosti
    }
}

```

Rozhranie (poslucháč)	Adaptér	Trieda udalosti	Metóda pre pridanie poslucháča ku zdroju udalostí
ActionListener	neexistuje	ActionEvent	addActionListener
FocusListener	FocusAdapter	FocusEvent	addFocusListener
ChangeListener	neexistuje	ChangeEvent	addChangeListener
ComponentListener	ComponentAdapter	ComponentEvent	addComponentListener
ContainerListener	ContainerAdapter	ContainerEvent	addContainerEvent
KeyListener	KeyAdapter	KeyEvent	addKeyListener
MouseListener	MouseAdapter	MouseEvent	addMouseListener
MouseMotionListener	MouseMotionAdapter	MouseEvent	addMouseMotionListener
WindowListener	WindowAdapter	WindowEvent	addWindowListener

Obr. 13 Príklady rozhraní



Obr. 14 Základné triedy udalostí

IX. Trieda udalostí EventObject

Vstupným parametrom metód volaných pri vzniku udalosti je **objekt** nesúci informácie o udalosti. Triedy týchto objektov sú priamymi, alebo nepriamymi potomkami triedy **java.util.EventObject**.

Táto trieda definuje **dve metódy**:

- **Object getSource()** – metóda Object getSource vracia referenciu na objekt, ktorý je zdrojom udalosti
- **String toString()** - zobrazí nie len názov triedy, ale aj textovú reprezentáciu zdroja udalosti.

X. Trieda udalostí AWTEvent

Trieda `java.awt.AWTEvent` je potomkom triedy **EventObject**. Táto trieda obsahuje atribút `id`, ktorý možno získať pomocou metódy `getID()`. Atribút `id` obsahuje typ udalosti (napr. **KEY_RELEASE** definovaný v triede `KeyEvent`).

Tento atribút umožňuje filtrovanie udalostí. Filtrovanie udalostí slúži pre zníženie zaťaženia systému aplikáciou. Každá udalosť podporovaná triedou **java.awt.Component** je odosielaná prostredníctvom metódy `dispatchEvent()`. Táto metóda podľa atribútu `id` určí kategóriu udalosti. Napríklad pre udalosť **KEY_RELEASE** určí kategóriu udalostí `KeyEvent`.

Po určení kategórie udalosti sa kontroluje **maska udalostí**, ktorá určuje či je daná kategória udalostí povolená alebo nie. Ak je **kategória udalostí** zakázaná, tak sa udalosť ďalej nespracováva. **Maska udalostí** je pre každú kategóriu udalosti implicitne zakázaná, ale pri pripojení poslucháča sa maska pre príslušnú kategóriu udalostí automaticky povolí.

Pre **explicitné povolenie**, alebo **zakázanie kategórie** udalosti slúžia **metódy**:

- `protected final void enableEvents(long eventsToEnable)`
- `protected final void disableEvents(long eventsToDisable)`

definované v triede **java.awt.Component**.

Metódy triedy AWTEvent:

- **protected void consume()** - po zavolaní tejto metódy, nenastane implicitné spracovanie udalosti (ak to udalosť umožňuje).
- **getID()** - vráti typ udalosti.
- **protected boolean isConsumed()** - vráti ak bola udalosť „zničená“
- **String paramString()** - vráti reťazec obsahujúci stav udalosti (vhodné pre ladenie)
- **void setSource(Object newSource)** - nastaví nový zdroj udalosti
- **String toString()** - vráti reťazec reprezentujúci tento objekt

XI. Spracovanie udalostí

Trieda udalosti ChangeEvent a rozhranie ChangeListener.

Používa sa pre oznámenie rôznych typov zmien stavu. Neobsahuje žiadne informácie o stave udalosti (povrchné oznamovanie). Poslucháč musí zistiť stav sám.

Rozhranie ChangeListener definuje jedinú metódu:

- void stateChanged(ChangeEvent e)

Táto metóda je volaná vždy pri zmene stavu zdrojového objektu (toho objektu, ktorý je zdrojom udalosti).

Príklad:



Obr. 15 ChangeEventTest

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

```
public class ChangeEventTest extends JFrame {

    public ChangeEventTest() {
        setTitle("ChangeEventTest");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();
        pane.setLayout(new FlowLayout());

        JSlider slider = new JSlider();
        pane.add(slider);

        pack();

        slider.addChangeListener(new
ChangeListener() {
            public void stateChanged(ChangeEvent e)
            {
                JSlider slider = (JSlider)
e.getSource();
                System.out.println(slider.getValue());
            }
});
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                ChangeEventTest win = new
ChangeEventTest();
                win.setVisible();
            }
        });
    }
}
```

Výstup:

50

51

52

53

54

55

55

Udalosti generované komponentom Component

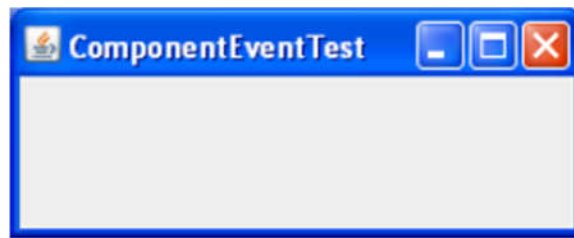
Trieda **java.awt.Component** je nadtriedou väčšiny vizuálnych komponentov definovaných v balíkoch java.awt a java.swing. Táto trieda poskytuje podporu podstatnej väčšine udalostí. Všetci jej potomkovia dedia schopnosť generovať tieto udalosti.

Trieda udalostí **ComponentEvent** a rozhranie **ComponentListener**

- Trieda **ComponentEvent** slúži pre zasielanie oznámení o zmenách viditeľnosti, veľkosti alebo pozície potomka triedy **Component**.
- Je podtriedou **AWTEvent**.
- Pridáva **metódu**: **Component** `getComponent()`, ktorá podobne ako `getSource` vracia **zdroj udalosti**. Rozdielom ale je že typ návratovej hodnoty je **Component**.

Volaná metóda v rozhraní ComponentListener	Hodnota atribútu <code>id</code> v triede ComponentEvent	popis udalosti
<code>componentHidden()</code>	<code>COMPONENT_HIDDEN</code>	skrytie komponentov
<code>componentShown()</code>	<code>COMPONENT_SHOWN</code>	zobrazenie komponentov
<code>componentMoved()</code>	<code>COMPONENT_MOVED</code>	zmena pozície
<code>componentResized()</code>	<code>COMPONENT_RESIZED</code>	Zmena veľkosti

*Tab. 1 Tabuľka typov udalostí definovaných rozhraním **ComponentListener***

Príklad:

Obr. 16 ComponentEventTest

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

public class ComponentEventTest extends JFrame {
    public ComponentEventTest() {
        setTitle("ComponentEventTest");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ;
        setSize(250,100);

        this.addComponentListener(new ComponentListener()
        {
            public void componentResized(ComponentEvent e)
            {
                System.out.println(e paramString());
            }
            public void componentMoved(ComponentEvent e) {
                System.out.println(e paramString());
            }
            public void componentShown(ComponentEvent e) {
                System.out.println(e paramString());
            }
        }
    }
}
```

```
        public void componentHidden(ComponentEvent e)
        {
            System.out.println(e paramString());
        }
    });
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            ComponentEventTest win = new
            ComponentEventTest();
            win.setVisible();
        }
    });
}
}
```

Príklad výstupu:

COMPONENT_RESIZED (0,0 250x100)
COMPONENT_RESIZED (0,0 250x100)
COMPONENT_MOVED (0,0 250x100)
COMPONENT_SHOWN
COMPONENT_MOVED (4,0 250x100)
COMPONENT_MOVED (6,2 250x100)
COMPONENT_MOVED (7,3 250x100)
COMPONENT_MOVED (9,5 250x100)
COMPONENT_MOVED (11,6 250x100)
COMPONENT_MOVED (12,8 250x100)
COMPONENT_MOVED (13,8 250x100)
COMPONENT_RESIZED (13,8 397x175)
COMPONENT_MOVED (13,9 397x175)
COMPONENT_MOVED (17,9 397x175)
COMPONENT_MOVED (19,10 397x175)
COMPONENT_MOVED (20,10 397x175)

Trieda udalostí FocusEvent a rozhranie FocusListener

Ak má komponent **focus**, tak môže prijímať vstup od používateľa. Ak nemá focus, tak komponent neprijíma vstup od používateľa. Napr. ak má textový komponent focus, tak v ňom bliká kurzor.

FocusEvent je potomkom triedy ComponentEvent.

Pridáva **metódy**:

boolean isTemporary(), ktorá určuje či je stav trvalý (focus získal iný komponent), alebo dočasný (pri určitých operáciách, po dokončení ktorých sa focus vráti napr. minimalizovanie celého okna aplikácie).

Component getOppositeComponent(), ktorá vracia referenciu na komponent ktorý fokus stratil, alebo získal, alebo vracia **null**.

Volaná metóda v rozhraní FocusListener	Hodnota atribútu id v triede FocusEvent	Popis udalosti
componentGained()	FOCUS_GAINED	Komponent získal focus
componentLost ()	FOCUS_LOST	Komponent stratil focus

Tab. 2 Tabuľka typov udalostí definovaných rozhraním FocusListener

Príklad:



Obr.17 FocusEventTest

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

public class FocusEventTest extends JFrame {

    public FocusEventTest() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container pane = getContentPane();
        pane.setLayout(new FlowLayout());
        JTextField text1 = new JTextField(10);
        JTextField text2 = new JTextField(10);
        pane.add(text1, BorderLayout.NORTH);
        pane.add(text2, BorderLayout.SOUTH);
        pack();
    }
}
```

```
text1.addFocusListener(new FocusListener() {
    public void focusGained(FocusEvent e) {
        System.out.println("text1 získal focus
            (isTemporary="+e.isTemporary()+
            ")");
    }
    public void focusLost(FocusEvent e) {
        System.out.println("text1 stratil
            focus
            (isTemporary="+e.isTemporary()+
            ")");
    }
});

}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            FocusEventTest win = new
            FocusEventTest();
            win.setVisible();
        }
    });
}
}
```

Trieda InputEvent

Je to **abstraktná** trieda, ktorá nie je pridružená žiadnemu rozhraniu poslucháča. Tvorí nadtriedu pre ďalšie dve podtriedy: **KeyEvent** a **MouseEvent**.

Trieda **InputEvent** poskytuje príznaky modifikátorov, ktoré označujú aké klávesy, resp. tlačidlá myši boli stlačené. Medzi modifikátory patria napríklad klávesy **Shift**, **Control**, **Alt** a tlačidlá myši.

Trieda tiež poskytuje časové **razítko** označujúce presný **čas**, kedy udalosť nastala. Tento čas možno získať volaním metódy `getWhen()`. Metóda vracia hodnotu typu `long`, ktorú možno použiť pre konštrukciu objektu typu **java.util.Date**.

Metódy triedy InputEvent:

- `void consume()`
- `getModifiers()`
- `getModifiersEx()`
- `static String getModifiersExText(modifiers)`
- `long getWhen()`
- `boolean isAltDown()`
- `boolean isAltGraphDown()`
- `boolean isConsumed()`
- `boolean isControlDown()`
- `boolean isMetaDown()`
- `boolean isShiftDown()`

Trieda udalostí KeyEvent a rozhranie KeyListener

Slúžia pre udalosti generované pri stlačení, alebo uvoľnení klávesy na klávesnici. Rozlišujeme dve hlavné kategórie kláves:

- **znakové klávesy** – majú vizuálnu reprezentáciu
- **neznakové klávesy** – nemajú vizuálnu reprezentáciu (F1, Ctrl, Alt, Shift, Backspace, Delete)

Trieda definuje mnoho **konštánt** zodpovedajúcich znakom generovaných na klávesnici US, napr.: VK_A, VK_B, VK_F1, VK_F2, VK_CONTROL, VK_ALT, VK_SHIFT, VK_BACK_SPACE.

Volaná metóda v rozhraní KeyListener	Hodnota atribútu id v triede KeyEvent	Popis udalosti
keyPressed ()	KEY_PRESSED	stlačenie klávesy na klávesnici
keyReleased ()	KEY_RELEASED	uvoľnenie klávesy na klávesnici
keyTyped ()	KEY_TYPED	napísanie znaku

Tab. 3 Tabuľka typov udalostí definovaných rozhraním KeyListener

Udalosti **KEY_PRESSED** a **KEY_RELEASED** sú generované pre každé fyzické stlačenie klávesy. Udalosť **KEY_TYPED** je reprezentácia „vyššieho“ typu udalosti, ktorým je napísanie znaku („logické stlačenie klávesy“), pretože niektoré znaky sa dajú napísať iba stlačením viacerých kláves napr. dlhé ó, alebo hviezdička *.

Metody triedy KeyEvent:

`char` `getKeyChar()` - vracia napísaný znak
`getKeyCode()` - vracia kód fyzickej klávesy
`getKeyLocation()` - niektoré klávesy sa vyskytujú na viacerých miestach klávesnice
`static String` `getKeyModifiersText(modifiers)`
`static String` `getKeyText(keyCode)`
`boolean` `isActionKey()`
`String` `paramString()`
`void` `setKeyChar(char keyChar)`
`void` `setKeyCode(keyCode)`
`void` `setModifiers(modifiers)`

Príklad:



Obr. 18 KeyEventTest

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

```
public class KeyEventTest extends JFrame {
    public KeyEventTest() {
        setTitle("KeyEventTest");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();
        pane.setLayout(new FlowLayout());

        JTextField textField = new JTextField(10);
        add(textField);
        textField.addKeyListener(new KeyListener()
        {
            public void keyTyped(KeyEvent e) {
                System.out.println(e paramString());
            }
            public void keyPressed(KeyEvent e) {
                System.out.println(e paramString());
            }
            public void keyReleased(KeyEvent e) {
                System.out.println(e paramString());
            }
        });
        pack();
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                KeyEventTest win = new KeyEventTest();
                win.setVisible();
            }
        });
    }
}
```

Trieda udalostí **MouseEvent** a rozhrania **MouseListener** a **MouseMotionListener**. Trieda **MouseEvent** sa používa pre udalosti súvisiace s používaním myši. Implementácia rozhrania **MouseMotionListener** sa používa pre udalosti súvisiacich s pohybom myši, implementácia **MouseListener** sa používa pre ostatné udalosti súvisiace s myšou.

Udalosti súvisiace s pohybom myši sú oddelené od ostatných udalostí pretože týchto udalostí je generovaných najviac. Ak sa v aplikácii nevyžaduje spracovanie udalostí pohybu myši, tak možno tieto udalosti oddeliť a nespracovávať, čo zvyšuje výkon aplikácie.

Voľná metóda v rozhraní MouseListener	Hodnota atribútu id v triede MouseEvent	Popis udalosti
<code>mouseClicked ()</code>	<code>MOUSE_CLICKED</code>	kliknutie na tlačidlo myši
<code>mouseListener ()</code>	<code>MOUSE_LISTENER</code>	ukazovateľ myši sa presunul nad zobrazovaciu oblasť grafickej komponenty
<code>mouseExcited ()</code>	<code>MOUSE_EXCITED</code>	ukazovateľ myši sa presunul mimo zobrazovaciu oblasť grafickej komponenty
<code>mousePressed ()</code>	<code>MOUSE_PRESSED</code>	stlačenie tlačidla
<code>mouseReleased ()</code>	<code>MOUSE_RELEASED</code>	uvoľnenie tlačidla

*Tab. 4 Tabuľka typov udalostí definovaných rozhraním **MouseListener***

Tabuľka typov udalostí definovaných rozhraním MouseListener	Hodnota atribútu id v triede MouseEvent	popis udalosti
mouseMoved()	MOUSE_MOVED	ukazovateľ myši sa presunul nad zobrazovacou plochou grafickej komponenty, tlačidlo myši nie je stlačené
mouseDragged()	MOUSE_DRAGGED	ukazovateľ myši sa presunul, pričom je stlačené tlačidlo myši. Tlačidlo muselo byť stlačené keď bol ukazovateľ myši nad zobrazovacou plochou grafickej komponenty

*Tab. 5 Tabuľka typov udalostí definovaných rozhraním
MouseListener*

Základná trieda pre nasledujúce príklady:

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

public class MouseEventTestBase extends JFrame {
    private JPanel centerPanel;
```

```
public MouseEventTestBase() {  
    setTitle("MouseEventTestBase");  
  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    Container pane = getContentPane();  
  
    pane.add(createPanel(50,50),  
        BorderLayout.NORTH);  
    pane.add(createPanel(50,50),  
        BorderLayout.SOUTH);  
  
    pane.add(createPanel(50,50),  
        BorderLayout.WEST);  
    pane.add(createPanel(50,50),  
        BorderLayout.EAST);  
  
    centerPanel = createPanel(200, 150);  
    centerPanel.setBackground(Color.WHITE);  
    pane.add(centerPanel, BorderLayout.CENTER);  
  
    pack();  
  
    pack();  
}
```

```
protected JPanel getCenterPanel() {  
    return centerPanel;  
}  
private JPanel createPanel(preferredWidth,  
                           preferredHeight) {  
    JPanel panel = new JPanel();  
    panel.setPreferredSize(new  
Dimension(preferredWidth,  
                           preferredHeight));  
    return panel;  
}  
}
```

Príklad – použitie MouseListener:

```
import java.awt.*;  
import java.awt.event.*;  
  
import javax.swing.*;  
  
public class MouseEventTest1 extends  
MouseEventTestBase {
```

```
public MouseEventTest1() {
    setTitle("MouseEventTest - MouseListener");
    getCenterPanel().addMouseListener(new
MouseListener() {
        public void mouseClicked(MouseEvent e) {
            System.out.println(e paramString());
        }
        public void mousePressed(MouseEvent e) {
            System.out.println(e paramString());
        }
        public void mouseReleased(MouseEvent e) {
            System.out.println(e paramString());
        }
        public void mouseEntered(MouseEvent e) {
            System.out.println(e paramString());
        }
        public void mouseExited(MouseEvent e) {
            System.out.println(e paramString());
        }
    });
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            MouseEventTest1 win = new
MouseEventTest1();
            win.setVisible();
        }
    });
}
}
```



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MouseEventTest2 extends
MouseEventTestBase {
    public MouseEventTest2() {
        setTitle("MouseEventTest2 -
        MouseMotionListener");
        getCenterPanel().addMouseMotionListener(
            new
            MouseMotionListener() {
                public void mouseDragged(MouseEvent e)
                {
                    System.out.println(e paramString());
                }
                public void mouseMoved(MouseEvent e) {
                    System.out.println(e paramString());
                }
            });
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                MouseEventTest2 win = new
                MouseEventTest2();
                win.setVisible();
            }
        });
    }
}
```

Metódy triedy MouseEvent (dedí od triedy InputEvent)

- **getButton()** - vráti informáciu ktoré tlačidla myši zmenili stav.
- **getClickCount()** - vráti počet kliknutí súvisiacich s udalosťou
- **Po getLocationOnScreen()** - vráti absolútne ukazovateľa myši súradnice x, y, pri vzniku udalosti
- **static String getMouseModifiersText(modifiers)** - vráti reťazec popisujúci modifikátory a tlačidlá myši ktoré boli stlačené počas udalosti
- **Po getPo()** - vráti súradnice x, y pri vzniku udalosti. Tieto súradnice sú relatívne vzhľadom ku zdrojovému komponentu
- **getX()** - vráti relatívnu súradnicu x
- **getXOnScreen()** - vráti absolútnu súradnicu x
- **getY()** - vráti relatívnu súradnicu y
- **getYOnScreen()** - vráti absolútnu súradnicu y
- **boolean isPopupTrigger()** - vráti, či udalosť spúšťa popup menu
- **String paramString()** - vráti reťazec obsahujúci parametre o udalosti
- **void translatePo(x, y)** - zmení súradnice udalosť pripočítaním hodnôt x a y

Príklad rozpoznanie dvojklíku:

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

public class MouseEventTestDoubleClick extends
MouseEventTestBase {
    public MouseEventTestDoubleClick() {
        setTitle("MouseEventTestDoubleClick");

        getCenterPanel().addMouseListener(new
        MouseAdapter() {
            public void mouseClicked(MouseEvent e)
            {
                if(e.getClickCount() == 2) {
                    System.out.println("dvojklik");
                }
            }
        });
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                MouseEventTestDoubleClick win =
                    new
                    MouseEventTestDoubleClick()
                    ; win.setVisible();
            }
        });
    }
}
```

Príklad – tlačidlá a pozícia:

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

public class MouseEventTestButtonsAndPosition extends
MouseEventTestBase {
    public MouseEventTestButtonsAndPosition() {
        setTitle("MouseEventTestButtonsAndPos
ition");

        getCenterPanel().addMouseListener(new
MouseListener() {
            public void mouseClicked(MouseEvent e) {
                String buttonName = "";
                switch(e.getButton()) {
                    case MouseEvent.BUTTON1:
                        buttonName = "lave";
                        break;
                    case MouseEvent.BUTTON2:
                        buttonName = "stredne";
                        break;
                    case MouseEvent.BUTTON3:
                        buttonName = "prave";
                        break;
                    default:
                        buttonName = "????";
                }
                System.out.println("tlacidlo: " +
buttonName + ",
                    pozicia: " + e.getX() + ", " +
e.getY() );
            }
        });
    }
}
```

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            MouseEventTestButtonsAndPosition  
            win =  
                new  
                MouseEventTestButtonsAndPosition();  
            win.setVisible();  
        }  
    });  
}
```

Trieda udalostí InputMethodEvent a rozhranie InputMethodListener

Určená pre jazyky z rozsiahlymi **znakovými sadami** a pre zadávanie znakov z iných zariadení ako **klávesnica**.

Trieda udalostí **HierarchyEvent** a rozhrania **HierarchyListener** a **HierarchyBoundsListener**. Sú určené pre **prijímanie** oznámení o zmene stavu v hierarchii **grafických** komponentov.

Metódy triedy HierarchyEvent:

- Component getChanged()
- Container getChangedParent()
- long getChangeFlags()
- Component getComponent()
- String paramString()

Voľná metóda v rozhraní HierarchyListener	Hodnota atribútu id v triede HierarchyEvent	popis udalosti
hierarchyChanged()	HIERARCHY_CHANGED	niektorá zo zmien: - zmena nadradenej komponenty - zobrazenie, alebo skrytie hierarchie - zmena viditeľnosti

*Tab.6 Tabuľka typov udalostí definovaných rozhraním
HierarchyListener*

Volaná metóda v rozhraní HierarchyBoundListener	Hodnota atribútu id v triede HierarchyEvent	popis udalosti
ancestorMoved()	ANCESTOR_MOVED	zmena pozície nadradenej komponenty
ancestorResized()	ANCESTOR_RESIZED	zmena veľkosti nadradenej komponenty

*Tab. 7 Tabuľka typov udalostí definovaných rozhraním
HierarchyBoundListener*

Udalosti generované kontajnerom Container

Trieda **java.awt.Container** je базovou triedou mnohých vizuálnych komponentov. Je nadradenou triedou triedy **javax.swing.JComponent**, ktorá je nadradenou triedou väčšiny vizuálnych komponentov knižnice **Swing**.

Trieda **Container** obsahuje metódy pre vkladanie ďalších komponentov na zobrazovaciu plochu kontajnera. Container tiež obsahuje metódy na **odoberanie** komponentov.

Trieda udalostí ContainerEvent a rozhranie ContainerListener

Metódy triedy ContainerEvent (dedí od ComponentEvent):

- **Component getChild()**
- **Container getContainer()**
- **String paramString()**

Volaná metóda v rozhraní ContainerListener	Hodnota atribútu id v triede ContainerEvent	popis udalosti
componentAdded()	COMPONENT_ADDED	do kontajneru bol pridaný komponent
componentRemoved()	COMPONENT_REMOVED	z kontajneru bol odobratý komponent

Tab. 8 Tabuľka typov udalostí definovaných *rozhraním ContainerListener*

Udalosti generované oknom Window

Trieda **java.awt.Window** je potomkom triedy **java.awt.Container**. Je priamou, alebo nepriamou nadtriedou všetkých oknových komponentov v knižniciach AWT aj **Swing** (napr. **JFrame**, **JDialog**).

Trieda udalostí **WindowEvent** a rozhrania **WindowListener**, **WindowStateListener** a **WindowFocusListener**.

Metódy triedy **WindowEvent** (dedí od **ComponentEvent**):

- **getNewState()** - pri výskyte udalosti **WINDOW_STATE_CHANGED** vráti nový stav okna
- **getOldState()** - pri výskyte udalosti **WINDOW_STATE_CHANGED** vráti predchádzajúci stav okna
- **Window getOppositeWindow()** - pri zmene focus-u vráti iné okno ktorému sa zmenil focus
- **Window getWindow()** - vracia zdroj udalosti ako inštanciu triedy **Window**
- **String paramString()** - vráti reťazec identifikujúci udalosť

Volaná metóda v rozhraní WindowListener	Hodnota atribútu id v triede WindowEvent	popis udalosti
windowActivated()	WINDOW_ACTIVATED	okno sa stane aktívnym
windowClosed()	WINDOW_CLOSED	okno bolo zavreté
windowClosing()	WINDOW_CLOSING	požiadavka na zavretie okna
windowDeactivated()	WINDOW_DEACTIVATED	okno už nie je aktívne
windowDeiconified()	WINDOW_DEICONIFIED	obnovenie z minimalizovaného stavu
windowIconified()	WINDOW_ICONIFIED	okno bolo minimalizované
windowOpened()	WINDOW_OPENED	prvé zobrazenie okna

*Tab. 9 Tabuľka typov udalostí definovaných rozhraním **WindowListener***

Volaná metóda v rozhraní WindowStateListener	Hodnota atribútu id v triede WindowEvent	Popis udalosti
windowStateChanged()	WINDOW_STATE_CHANGED	zmena stavu okna (minimalizácia, maximalizácia, ...)

*Tab. 10 Tabuľka typov udalostí definovaných rozhraním
WindowStateListener*

Volaná metóda v rozhraní WindowFocusListener	Hodnota atribútu id v triede WindowEvent	Popis udalosti
windowGainedFocus	WINDOW_GAINED_FOCUS	Získanie focus-u
windowLostFocus	WINDOW_LOST_FOCUS	Strata focus-u

*Tab. 11 Tabuľka typov udalostí definovaných rozhraním
WindowFocusListener*

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

public class WindowEventTest extends JFrame {
    public WindowEventTest() {
        setTitle("WindowEventTest");
        // vynechame setDefaultCloseOperation
        setSize(250,100);
    }
}
```

```
this.addListener(new WindowListener()
{
    public void windowOpened(WindowEvent e)
    {
        System.out.println(e paramString());
    }
    public void windowClosing(WindowEvent
e) {
        System.out.println(e paramString());
        System.exit(0);
        // dispose();
    }
    public void windowClosed(WindowEvent e)
    {
        System.out.println(e paramString());
    }
    public void windowIconified(WindowEvent
e) {
        System.out.println(e paramString());
    }
    public void windowDeiconified(WindowEvent e) {
        System.out.println(e paramString());
    }
    public void windowActivated(WindowEvent
e) {
        System.out.println(e paramString());
    }
}
```

```
public void
windowDeactivated(WindowEvent e) {
    System.out.println(e paramString());
}
});
this.addWindowStateListener(new
WindowStateListener() {
    public void windowStateChanged(WindowEvent
e) {
        System.out.println(e paramString());
    }
});
this.addWindowFocusListener(new
WindowFocusListener() {
    public void windowGainedFocus(WindowEvent
e) {
        System.out.println(e paramString());
    }
    public void windowLostFocus(WindowEvent e)
    {
        System.out.println(e paramString());
    }
});
}
```

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            WindowEventTest win = new  
            WindowEventTest();  
            win.setVisible();  
        }  
    });  
}
```

Udalosti generované triedou JComponent

Trieda **javax.swing.JComponent** je nadtriedou väčšiny komponent knižnice **swing**.

Trieda udalostí AncestorEvent a rozhrania AncestorListener

Trieda AncestorEvent je podobná triede ComponentEvent. Dedí od triedy AWTEvent. Poskytuje metódu `getComponent()` ktorá vracia zdroj udalosti ako komponent.

Metódy triedy AncestorEvent:

- Container `getAncestor()`
- Container `getAncestorParent()`
- JComponent `getComponent()`

Udalosti generované triedou **AbstractButton**

Trieda `javax.swing.AbstractButton` je nadtriedou množstva bežne používaných komponentov knižnice `swing`. Napríklad:

- **JButton**
- **JToggleButton**
- **JCheckBox**
- **JRadioButton**
- **JMenuItem**
- **JMenu**
- **JRadioButtonMenuItem**
- **JCheckBoxMenuItem**

Každý potomok generuje jednu alebo viacero z nasledujúcich udalostí `ActionEvent`, `ChangeEvent`, `ItemEvent`.

Inštancie triedy **AbstractButton** si udržujú určité **vlastnosti**:

- **pressed** – má hodnotu , ak je tlačidlo stlačené
- **armed** – ak má hodnotu , tak sa uvoľnením tlačidla myši aktivuje tlačidlo na obrazovke
- **enabled** – určuje, či je tlačidlo povolené
- **rollover** – ak má hodnotu , tak môže zobrazovať jednu ikonu keď je ukazovateľ myši nad tlačidlom a inú ikonu, keď je ukazovateľ myši mimo tlačidla
- **selected** – určuje, či je zvolená voľba (používa sa napr. v inštanciách `JRadioButton`, alebo `JCheckBox`)

Trieda udalostí **ActionEvent** a rozhrania **ActionListener**

Trieda `java.awt.event.ActionEvent` je určená pre udalosti ako napríklad stlačenie tlačidla.

Metódy triedy **ActionEvent**:

- **String getActionCommand()** - vracia príkazový reťazec, čo je väčšinou popis tlačidla
- **getModifiers()** - vráti modifikátory ktoré boli stlačené pri vzniku udalosti
- **long getWhen()** - vráti čas kedy sa udalosť vyskytla
- **String paramString()** - vráti reťazec identifikujúci udalosť

Volaná metóda v rozhraní ActionListener	Hodnota atribútu id v triede ActionEvent	popis udalosti
<code>actionPerformed()</code>	<code>ACTION_PERFORMED</code>	komponenta bola aktivovaná

*Tab. 12 Tabuľka typov udalostí definovaných rozhraním **ActionListener***

Príklad – aktivácia **JButton**:



*Obr.19 Aktivácia **JButton***

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

public class ActionEventTestButton extends JFrame {
    public ActionEventTestButton() {
        setTitle("ActionEventTestButton");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();
        pane.setLayout(new FlowLayout());

        JButton tlacidlo = new JButton("stlac");
        pane.add(tlacidlo);

        pack();

        tlacidlo.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent
            e) {
                System.out.println(e paramString());
            }
        });
    }
}
```



```

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            ActionEventTestButton win =
                new
                    ActionEventTestButton();
            win.setVisible();
        }
    });
}
}

```

Príklad – aktivácia JRadioButton:



Obr.20 Aktivácia JRadioButton

```

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

```

```
public class ActionEventTestRadioButton extends
JFrame {
    public ActionEventTestRadioButton() {
        setTitle("ActionEventTestRadioButton");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);

        Container pane = getContentPane();
        pane.setLayout(new BoxLayout(pane,
BoxLayout.Y_AXIS));

        ButtonGroup group = new ButtonGroup();
        Responder responder = new Responder();

        createRadionButton("prvy", KeyEvent.VK_P, ,
pane,
group, responder);
        createRadionButton("druhy", KeyEvent.VK_D,
, pane,
group, responder);
        createRadionButton("tretí", KeyEvent.VK_T,
, pane,
group, responder);
        createRadionButton("štvrtý", KeyEvent.VK_S,
, pane,
group, responder);
        createRadionButton("piaty", KeyEvent.VK_I,
, pane,
group, responder);

        pack();
    }
}
```

```
private void createRadioButton(String name,
mnemonic,
        boolean selected, Container
        container,
        ButtonGroup group, Responder
        responder) {
    JRadioButton radioButton = new
JRadioButton(name);
    radioButton.setMnemonic(mnemonic);
    radioButton.setSelected(selected);
    container.add(radioButton);
    group.add(radioButton);
    radioButton.addActionListener(responder);
}

private class Responder implements ActionListener
{
    public void actionPerformed(ActionEvent e) {
        System.out.println(e paramString());
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            ActionEventTestRadioButton win =
                new
                ActionEventTestRadioButton();
            win.setVisible();
        }
    });
}
```

Trieda udalostí ItemEvent a rozhrania ItemListener

Sú určené pre objekty implementujúce rozhranie ItemSelectable.
Např. pre inštancie tried **JComboBox**, **JListBox**, **AbstractButton**.

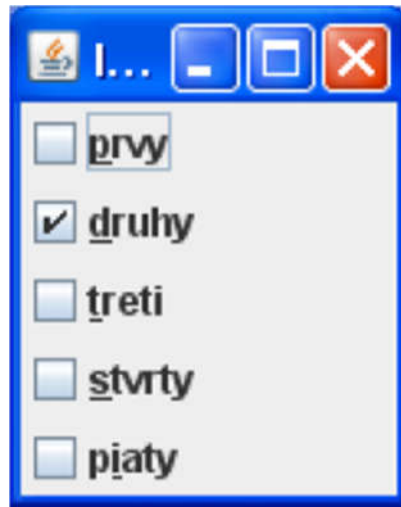
Metódy triedy ItemEvent:

- Object getItem()
- Returns the item affected by the event.
- ItemSelectable getItemSelectable()
- Returns the originator of the event.
- getStateChange()
- Returns the type of state change (selected or deselected).
- String paramString()
- Returns a parameter string identifying this item event.

Volaná metóda v rozhraní ItemListener	Hodnota atribútu id v triede ItemEvent	popis udalosti
itemStateChanged()	ITEM_STATE_CHANGED	položka bola vybraná, alebo bol je výber zrušený

Tab. 13 Tabuľka typov udalostí definovaných rozhraním ItemListener

Príklad – jcheckbox:



Obr.21 jcheckbox

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

public class ItemEventTestCheckBox extends JFrame {
    public ItemEventTestCheckBox() {
        setTitle("ItemEventTestCheckBox");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();

        pane.setLayout(new BoxLayout(pane,
        BoxLayout.Y_AXIS));
```

```
Responder responder = new Responder();

createCheckBox("prvy",
KeyEvent.VK_P,,pane,responder);
createCheckBox("druhy",      KeyEvent.VK_D,,
pane,responder);
createCheckBox("tretí",
KeyEvent.VK_T,,pane,responder);
createCheckBox("stvrty",KeyEvent.VK_S,,pane
,responder);      createCheckBox("piaty",
KeyEvent.VK_I,,pane,responder);

pack();

}

private void createCheckBox(String name,
mnemonic,
boolean selected, Container
container,
Responder responder) {
JCheckBox checkBox = new JCheckBox(name);
checkBox.setMnemonic(mnemonic);
checkBox.setSelected(selected);
container.add(checkBox);
checkBox.addItemListener(responder);
}
```

```
private class Responser implements ItemListener
{
    public void itemStateChanged(ItemEvent e) {
        String vyber =
            (e.getStateChange() ==
             ItemEvent.SELECTED)
            ? "vybrany" : "zruseny";
        JCheckBox checkBox =
            (JCheckBox)e.getItemSelectable();
        System.out.println(checkBox.getActionCommand()
            +
            " " + vyber);
    }
}

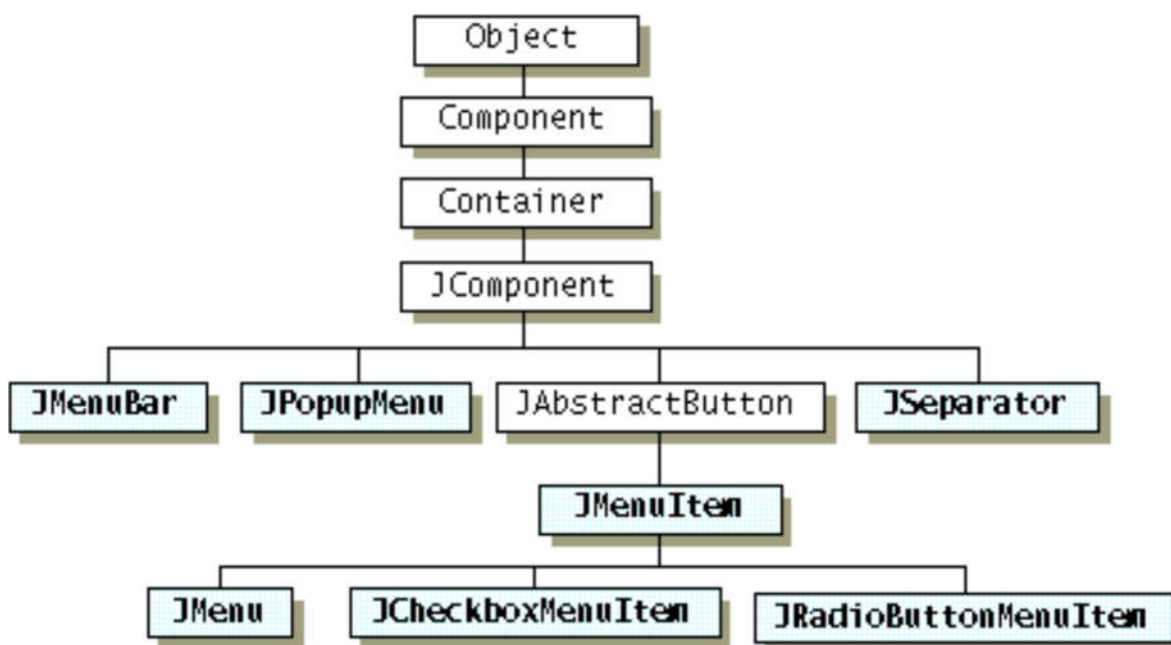
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            ItemEventTestCheckBox win =
                new
                    ItemEventTestCheckBox();
            win.setVisible();
        }
    });
}
```

XII. JMenuitem

Trieda JMenuitem je potomkom triedy **AbstractButton** a má troch potomkov:

- JMenu
- JCheckBoxMenuItem
- JRadioButtonMenuItem

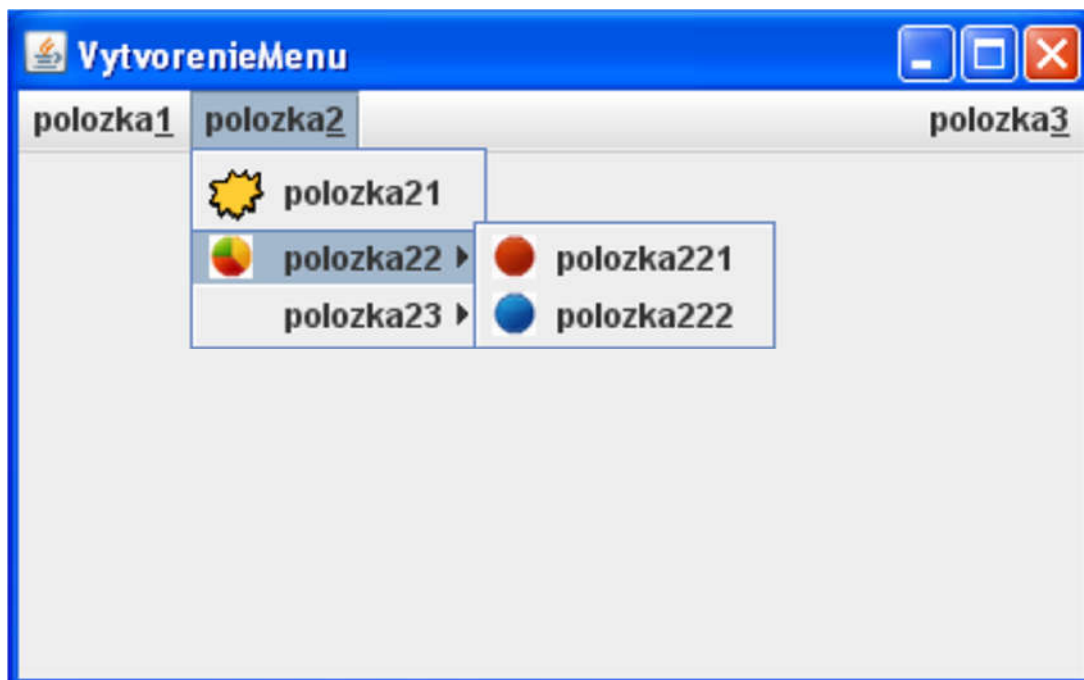
Ďalšou triedou používanou pre tvorbu menu je **JMenuBar**, ktorá reprezentuje **riadok** menu.



Obr. 22 Hierarchia komponentov pre vytváranie menu



Obr.23 Vytvorenie menu



Obr. 24 Vytvorenie menu

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

public class MenuExample extends JFrame {
    public MenuExample() {
        setTitle("VytvorenieMenu");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400,250);

        setJMenuBar(vytvorMenu());
    }
    private JMenuBar vytvorMenu() {
        JMenuBar menuBar = new JMenuBar();

        // 1. cast menu
        JMenu menu1 = new JMenu("polozka1");
        menu1.setMnemonic(KeyEvent.VK_1);

        JMenuItem item11 = new
        JMenuItem("polozka11",KeyEvent.VK_P);
        JMenuItem item12 = new
        JMenuItem("polozka12");
        item12.setMnemonic(KeyEvent.VK_O);
        item12.setAccelerator(KeyStroke.getKeyStroke(
            KeyEvent.VK_A,
            ActionEvent.ALT_MASK));
    }
}
```

```
JMenuItem          item13          =          new
JMenuItem("polozka13");
item13.setMnemonic(KeyEvent.VK_L);
item13.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_B, ActionEvent.CTRL_MASK));

JCheckBoxMenuItem   check12         =          new
JCheckBoxMenuItem("polozka12");

JCheckBoxMenuItem   check13         =          new
JCheckBoxMenuItem("polozka13");

JRadioButtonMenuItem radio14        =          new
JRadioButtonMenuItem("polozka14");
JRadioButtonMenuItem radio15        =          new
JRadioButtonMenuItem("polozka15");
ButtonGroup group1x = new ButtonGroup();
radio14.setSelected();
group1x.add(radio14);
group1x.add(radio15);

menuBar.add(menu1);
menu1.add(item11);
menu1.add(item12);
menu1.add(item13);
menu1.add(new JSeparator());
menu1.add(check12);
menu1.add(check13);
menu1.add(new JSeparator());
menu1.add(radio14);
menu1.add(radio15);
```

```
// 2. cast menu
JMenu menu2 = new JMenu("polozka2");
menu2.setMnemonic(KeyEvent.VK_2);

JMenuItem item21 = new
JMenuItem("polozka21", new ImageIcon(
getClass().getResource("images/middle.gif")
));

JMenu menu22 = new JMenu("polozka22");
menu22.setIcon(new
ImageIcon(getClass().getResource(
"images/pie_chart_16.png")));
JMenuItem item221 = new
JMenuItem("polozka221", new ImageIcon(
getClass().getResource("images/circle_red_1
6.png")));

JMenuItem item222 = new
JMenuItem("polozka222", new ImageIcon(
getClass().getResource("images/circle_blue_
16.png")));

JMenu menu23 = new JMenu("polozka23");
JMenuItem item231 = new
JMenuItem("polozka231");
JMenuItem item232 = new
```

```
JMenuItem("polozka232");

menuBar.add(menu2);
menu2.add(item21);
menu2.add(menu22);
menu2.add(menu23);
menu22.add(item221);
menu22.add(item222);
menu23.add(item231);

menu23.add(item232);

// medzera
menuBar.add(Box.createHorizontalGlue());

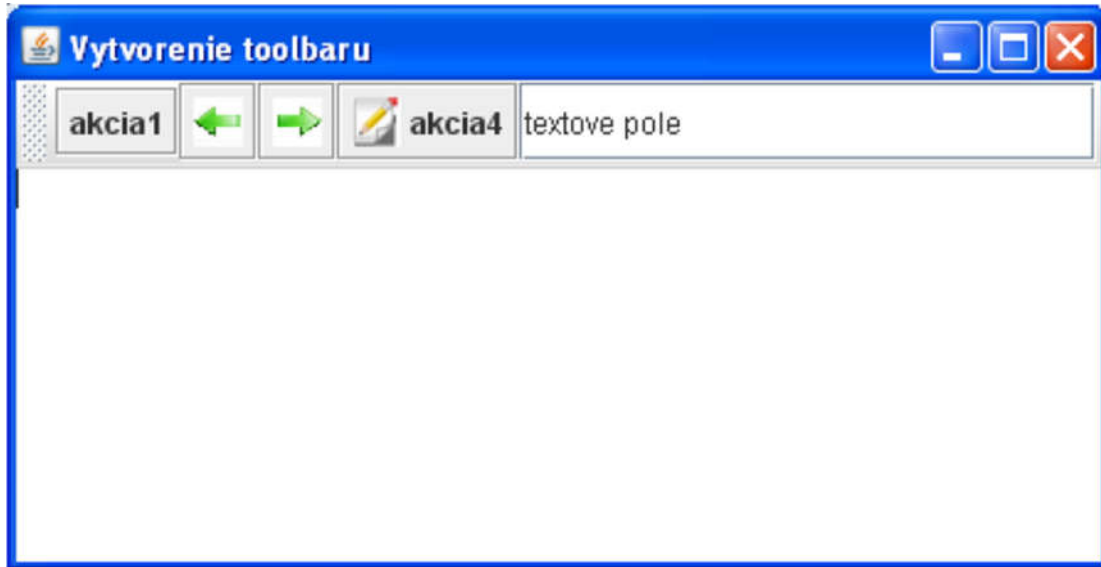
// 3. cast menu
JMenu menu3 = new JMenu("polozka3");
menu3.setMnemonic(KeyEvent.VK_3);
menuBar.add(menu3);

return menuBar;
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            MenuExample win = new MenuExample();
            win.setVisible();
        }
    });
}
}
```

XIII. Vytvorenie toolbaru

Príklad:



Obr. 25 Vytvorenie toolbaru

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

public class ToolBarExample extends JFrame {
    public ToolBarExample() { setTitle("Vytvorenie
toolbaru");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());

        pane.add(createToolBar(),BorderLayout.NORTH);
        pane.add(new
                                JTextArea(10,40),
BorderLayout.CENTER);
        pack();
    }
}
```

```
private JToolBar createToolBar() {
    JToolBar toolBar = new JToolBar("meno
toolbaru");
    JButton button1 = new JButton("akcia1");
    toolBar.add(button1);

    JButton button2 = new JButton(new ImageIcon(
getClass().getResource("images/arrow_left_g
reen_20.png"))));
    toolBar.add(button2);

    JButton button3 = new JButton(new ImageIcon(
getClass().getResource("images/arrow_right_g
reen_20.png"))));
    toolBar.add(button3);

    JButton button4 = new JButton("akcia4", new
ImageIcon(
getClass().getResource("images/paper&pencil_
20.png"))));
    toolBar.add(button4);

    JTextField textField = new
JTextField("textove pole");
    toolBar.add(textField);

    return toolBar;
}
```

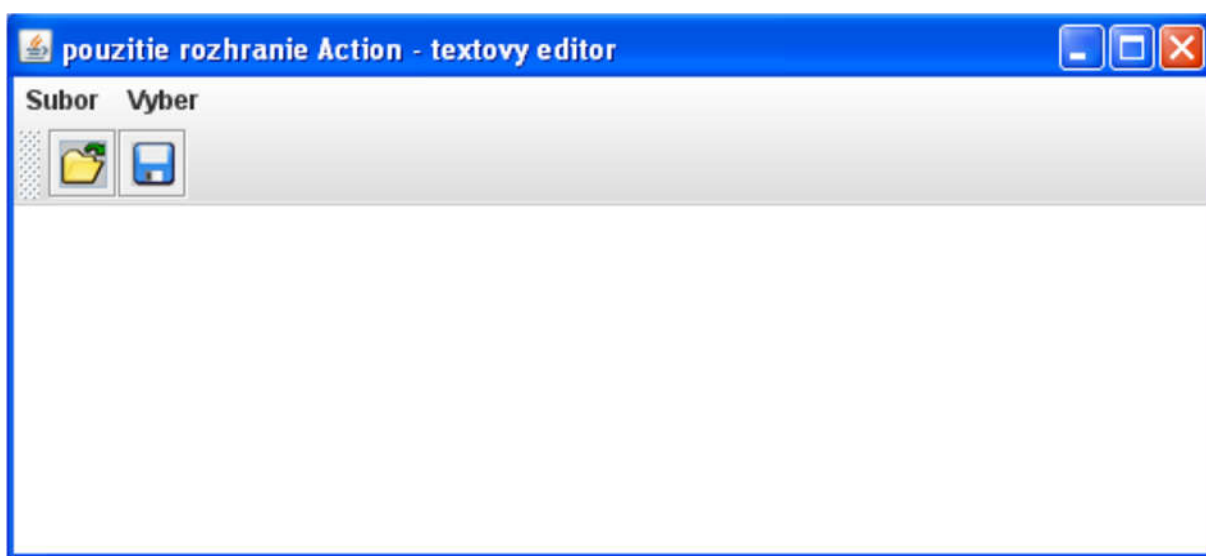
```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            ToolbarExample win = new  
            ToolbarExample();  
            win.setVisible();  
        }  
    });  
}
```


XIV. Spracovanie udalostí menu a toolBaru

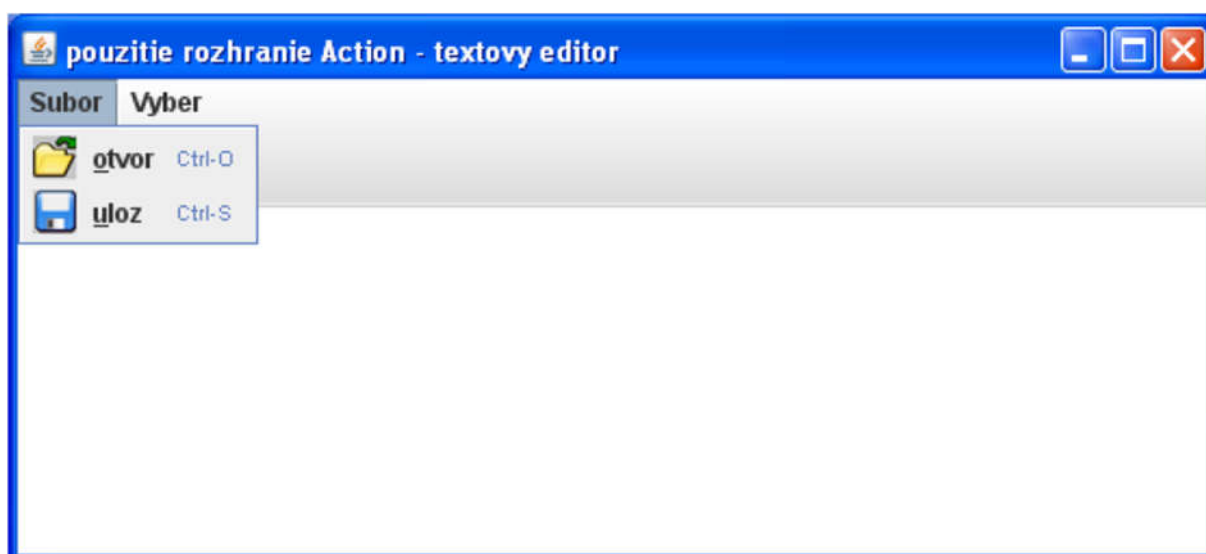
Pre spracovanie udalostí menu a toolbaru možno použiť **ActionListener** alebo **ItemListener** výhodnejšie je však použiť rozhranie **Action**.

Použitie rozhrania Action

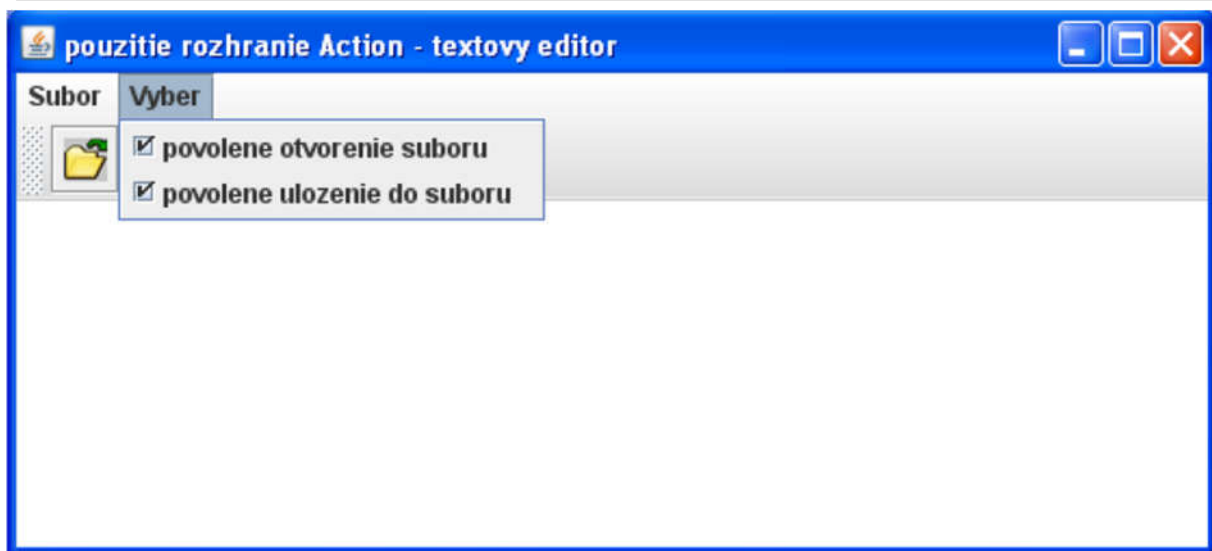
Príklad: (Action, AbstrackAction, JFileChooser)



Obr.26 Použitie rozhrania Action



Obr.27 Ponuka Súbor



Obr.28 Výber v rozhraní Action

TexEditorFrame.java:

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

public class TexEditorFrame extends JFrame {
    JTextArea textArea;
    OpenFileAction openFileAction;
    SaveFileAction saveFileAction;

    public TexEditorFrame() {
        setTitle("Action - textovy editor");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

```
        textArea = new JTextArea(10,50);
        openFileAction = new
OpenFileAction(textArea);
        saveFileAction = new
SaveFileAction(textArea);

        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());

        setJMenuBar(createMenu()); // J

        pane.add(createToolBar(),
BorderLayout.NORTH);

        pane.add(textArea, BorderLayout.CENTER);

        pack();
    }
    private JMenuBar createMenu() {
        JMenuBar menuBar = new JMenuBar();

        // Subor
        JMenu fileMenu = new JMenu("Subor");
        fileMenu.setMnemonic(KeyEvent.VK_S);
        menuBar.add(fileMenu);

        // Subor podmenu
        fileMenu.add(openFileAction);
        fileMenu.add(saveFileAction);

        // Vyber
        JMenu checkMenu = new JMenu("Vyber");
        checkMenu.setMnemonic(KeyEvent.VK_V);
        menuBar.add(checkMenu);
    }
}
```

```
// Vyber podmenu
JCheckBoxMenuItem openCheck =
    new JCheckBoxMenuItem("povolene   otvorenie
suboru", );
JCheckBoxMenuItem saveCheck =
    new JCheckBoxMenuItem("povolene   ulozenie
suboru", );

checkMenu.add(openCheck);
checkMenu.add(saveCheck);

openCheck.addItemListener(new
ActionEnabler(openFileAction));
saveCheck.addItemListener(new
ActionEnabler(saveFileAction));

return menuBar;
}

private JToolBar createToolBar() {
    JToolBar toolBar = new JToolBar();

    toolBar.add(openFileAction);
    toolBar.add(saveFileAction);

    return toolBar;
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
```

```
        public void run() {
            new TextEditorFrame().setVisible();
        }

    });
}

}
```

FileAction.java:

```
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import
javax.swing.filechooser.FileNameExtensionFilter;

public abstract class FileAction extends
AbstractAction {
    final protected JTextArea textArea;
    final private String dialogName;
    final private String errorMessage;

    final protected static JFileChooser
fileChooser;

    static {
        fileChooser = new JFileChooser();
        fileChooser.setCurrentDirectory(
new File(System.getProperty("user.dir")));
        FileNameExtensionFilter filter =new
        FileNameExtensionFilter(
            "textove subory", "txt" /*, "text"*/);
        fileChooser.setFileFilter(filter);
    }
}
```

```
public FileAction(JTextArea textArea,
                  String dialogName,
                  String errorMessage) {
    this.textArea = textArea;
    this.dialogName = dialogName;
    this.errorMessage = errorMessage;
}

public void actionPerformed(ActionEvent e) {
    // returnValue = fileChooser.showOpenDialog(
        SwingUtilities.getWindowAncestor(textArea));
    returnValue =
        fileChooser.showDialog(
            SwingUtilities.getWindowAncestor(textArea), dialogName);
    if (returnValue ==
        JFileChooser.APPROVE_OPTION) {
        String fileName =
            fileChooser.getSelectedFile().getAbsolutePath();
        try {
            fileAction(fileName);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(
                JOptionPane.getFrameForComponent(
textArea),
```

```
        "nazov suboru: \"" + fileName +
"\",
        errorMessage,
        JOptionPane.ERROR_MESSAGE);
    }
}
protected abstract void fileAction(String fileName)
    throws OException;
}
```

OpenFileAction.java:

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

public class OpenFileAction extends FileAction {

    public OpenFileAction(JTextArea textArea) {
        super(textArea, "Otvor", "Chyba pri citani zo
suboru");
        putValue(NAME, "otvor");
        putValue(SHORT_DESCRIPTION, "nacitanie textu
zo suboru");
        putValue(MNEMONIC_KEY, KeyEvent.VK_O);
        putValue(ACCELERATOR_KEY,
```

```
        KeyStroke.getKeyStroke(  
            KeyEvent.VK_O,  
            ActionEvent.CTRL_MASK));  
        putValue(SMALL_ICON, new ImageIcon(  
            getClass().getResource("images/open  
                .png"))));  
        //putValue(LARGE_ICON_KEY, .....  
    }  
  
    protected void fileAction(String fileName)  
throws IOException {  
        FileReader reader = new  
FileReader(fileName);  
        char[] buffer = new char[1000];  
        textArea.setText("");  
        numChar;  
        while ((numChar = reader.read(buffer)) > 0)  
        {  
            textArea.append(String.valueOf(buffer,  
                0, numChar));  
        }  
        reader.close();  
    }  
}
```

SaveFileAction.java:

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

public class SaveFileAction extends FileAction {

    public SaveFileAction(JTextArea textArea) {
        super(textArea, "Ulož", "Chyba pri zapise do
suboru");
        putValue(NAME, "ulož");
        putValue(SHORT_DESCRIPTION, "ulozenie textu
do suboru");
        putValue(MNEMONIC_KEY, KeyEvent.VK_U);
        putValue(ACCELERATOR_KEY,
        KeyStroke.getKeyStroke(
            KeyEvent.VK_S,
            ActionEvent.CTRL_MASK));
        putValue(SMALL_ICON, ImageIcon(
            getClass().getResource("images/save.p
ng"))));
    }

    protected void fileAction(String fileName) throws
IOException {
        final String extension = ".txt";
        if (! fileName.endsWith(extension)) {
            fileName = fileName.concat(extension);
        }
        FileWriter writer = new FileWriter(fileName);
        writer.write(textArea.getText());
        writer.close();
    }
}
```

ActionEnabler.java:

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

public class ActionEnabler implements ItemListener
{
    private Action action;

    public ActionEnabler(Action action) {
        this.action = action;
    }

    public void itemStateChanged(ItemEvent e) {
        action.setEnabled(e.getStateChange() ==
        ItemEvent.SELECTED);
    }
}
```

XV. Popup menu

Príklad:

PopupAction.java:

```
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;

public class PopupAction extends AbstractAction {

    public PopupAction(String name) {
        super(name);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("spustena akcia " +
            e.getActionCommand());
    }

}
```

PopupMenuListener.java:

```
import javax.swing.event.PopupMenuEvent;
import javax.swing.event.PopupMenuListener;

public class PopupListener implements
PopupMenuListener {

    public void
popupMenuWillBecomeVisible(PopupMenuEvent e) {
        System.out.println("popupMenuWillBecomeVisible
            ");
    }

}
```

```
public void  
popupMenuWillBecomeInvisible(PopupMenuEvent e) {  
    System.out.println("popupMenuWillBecomeInvisib  
le");  
}  
public void popupMenuCanceled(PopupMenuEvent e)  
{  
    System.out.println("popupMenuCanceled");  
}  
}
```

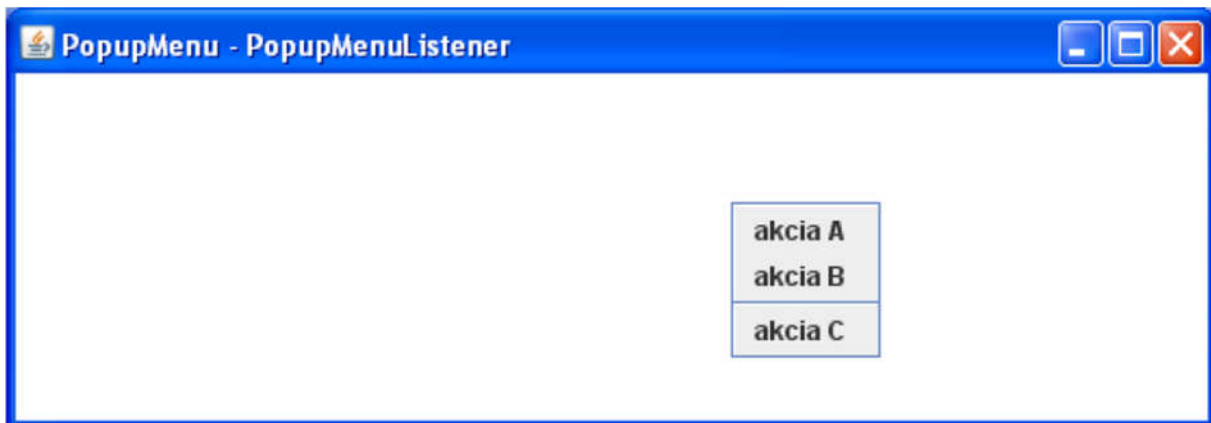
PopupMenuListener.java:

```
import java.awt.event.*;  
import javax.swing.JPopupMenu;  
  
public class PopupMouseListener extends  
MouseAdapter {  
    private JPopupMenu popupMenu;  
  
    public PopupMouseListener(JPopupMenu popupMenu)  
{  
        this.popupMenu = popupMenu;  
    }  
  
    @Override  
    public void mousePressed(MouseEvent e) {  
        maybeShowPopup(e);  
    }  
  
    @Override  
    public void mouseReleased(MouseEvent e) {  
        maybeShowPopup(e);  
    }  
}
```

```

private void maybeShowPopup(MouseEvent e) {
    if (e.isPopupTrigger()) {
        System.out.println("spustene popup menu");
        popupMenu.show(e.getComponent(),
            e.getX(), e.getY());
    }
}
}

```



Obr. 29 PopupMenuListener

PopupMenuPopupMenuListener.java:

```

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

public class PopupMenuPopupMenuListener extends
JFrame {
    public PopupMenuPopupMenuListener() {
        setTitle("PopupMenu - PopupMenuListener");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
    }
}

```

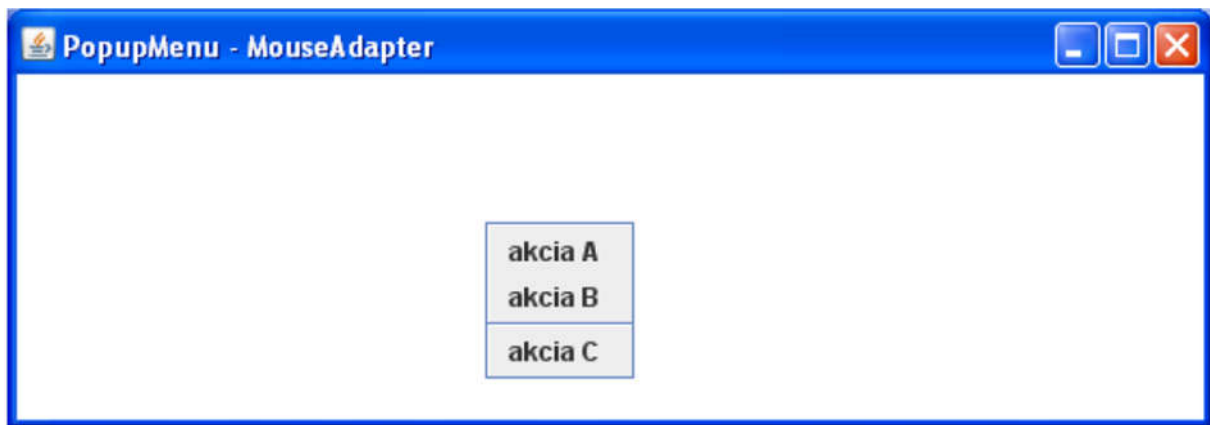
```
Container pane = getContentPane();
pane.setLayout(new BorderLayout());
JTextArea text = new JTextArea(10,50);
pane.add(text, BorderLayout.CENTER);

pack();

text.setComponentPopupMenu(createPopupMenu(
));
}
private JPopupMenu createPopupMenu() {
    JPopupMenu popup = new JPopupMenu();
    popup.addPopupMenuListener(new
PopupMenuListener());

    popup.add(new JMenuItem(new
PopupMenuAction("akcia A")));
    popup.add(new JMenuItem(new
PopupMenuAction("akcia B"))));
    popup.addSeparator();
    popup.add(new JMenuItem(new
PopupMenuAction("akcia C"))));
    return popup;
}
```

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            new  
            PopupMenuPopupMenuListener().setVisible  
            ();  
        }  
    });  
}
```



Obr. 30 MouseAdapter

PopupMenuMouseAdapter.java:

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

public class PopupMenuMouseAdapter extends JFrame {

    public PopupMenuMouseAdapter() {
        setTitle("PopupMenu - MouseAdapter");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    Container pane = getContentPane();
    pane.setLayout(new BorderLayout());
    JTextArea textArea = new JTextArea(10, 50);
    pane.add(textArea);
    pack();

    textArea.addMouseListener(
        new
        PopupMouseListener(createPopupMenu()));
    }

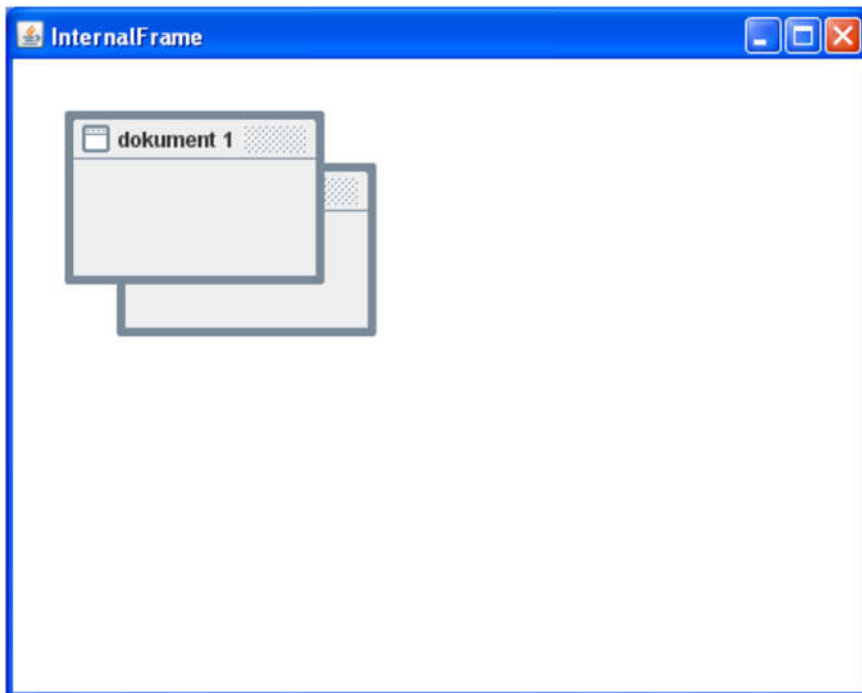
    private JPopupMenu createPopupMenu() {
        JPopupMenu popup = new JPopupMenu();
        popup.addPopupMenuListener(new
        PopupListener());
        popup.add(new JMenuItem(new PopupAction("akcia
A"))));
        popup.add(new JMenuItem(new
        PopupAction(("akcia B"))));
        popup.addSeparator();
        popup.add(new JMenuItem(new
        PopupAction(("akcia C"))));

        return popup;
    }
}
```



```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            PopupMenuMouseAdapter      win=new  
            PopupMenuMouseAdapter();  
            win.setVisible();  
        }  
    });  
}
```

XVI. Trieda JInfernalFrame



Obr. 31 Trieda Infernal Frame

MyinfernalFrame.java

```
import javax.swing.*;

public class MyinfernalFrame extends JInfernalFrame
{
    static openFrameCount = 0;
    static final xOffset = 30, yOffset = 30;
    public MyinfernalFrame() {
        super("dokument " + (++openFrameCount));
        setSize(150,100);
        setLocation(xOffset*openFrameCount,
yOffset*openFrameCount);
    }
}
```

InfernalFrameTest.java

```
import javax.swing.*;

public class infernalFrameTest extends JFrame {
    public infernalFrameTest() {
        setTitle("infernalFrame");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
        ;

        JDesktopPane desktopPane = new JDesktopPane();
        setContentPane(desktopPane);

        MyinfernalFrame      frame1      =      new
MyinfernalFrame();
        MyinfernalFrame      frame2      =      new
MyinfernalFrame();
        frame1.setVisible();
        frame2.setVisible();
        desktopPane.add(frame1);
        desktopPane.add(frame2);

        setSize(500,400);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                infernalFrameTest      win      =      new
infernalFrameTest();
                win.setVisible();
            }
        });
    }
}
```

XVII. Trieda JTabbedPane



Obr.32 Trieda JTabbedPane

Príklad:

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

public class TabbedPaneTest extends JFrame {
    public TabbedPaneTest() {
        setTitle("TabbedPaneTest");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container pane = getContentPane();

        JTabbedPane tabbedPane = new JTabbedPane();

        tabbedPane.addTab("panel 1", createFirstPanel());
```

```
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

tabbedPane.addTab("panel 2",
createSecondPanel());
tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);

pane.add(tabbedPane);

pack();
}
private JPanel createFirstPanel() {
    JPanel panel = new JPanel();
    numRows = 4;
    numCols = 4;
    panel.setLayout(new GridLayout(numRows,
numCols));
    for (i=0; i<(numRows*numCols); i++) {
        panel.add(new JTextField(5));
    }
    return panel;
}
private JPanel createSecondPanel() {
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.add(new JLabel("druhy panel"));
    return panel;
}
```

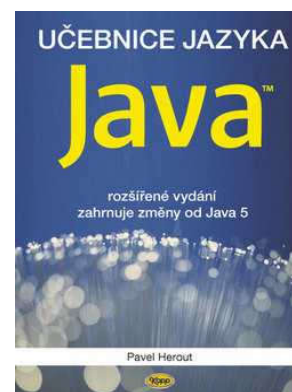
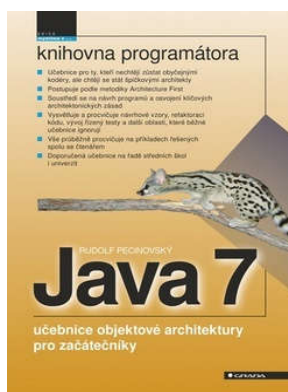
```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            TabbedPaneTest win = new  
            TabbedPaneTest();  
            win.setVisible();  
        }  
    });  
}
```

XVIII. Odporúčaná literatúra a zdroje

1. Java 8 - Herbert Schildt
2. Java bez předchozích znalostí - James Keogh
3. Java 8 - Rudolf Pecinovský
4. Myslíme objektově v jazyku Java - Rudolf Pecinovský

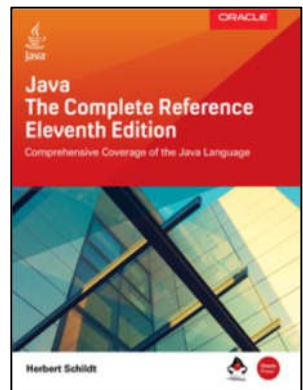
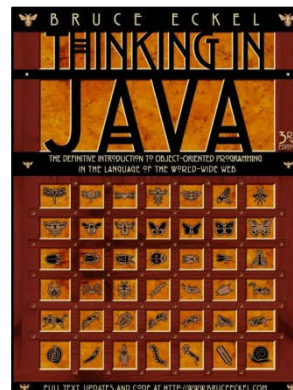
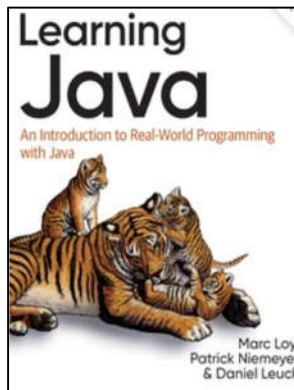


5. Mistrovství Java - Herbert Schildt
6. Java 7 - Rudolf Pecinovský
7. 1001 tipů a triků pro jazyk Java - Bogdan Kiszka
8. Učebnice jazyka Java 5 - Pavel Herout

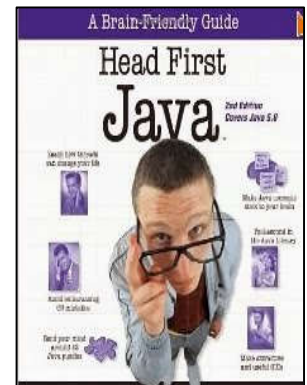
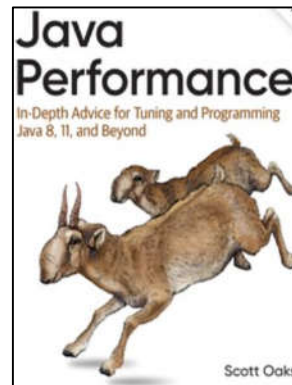
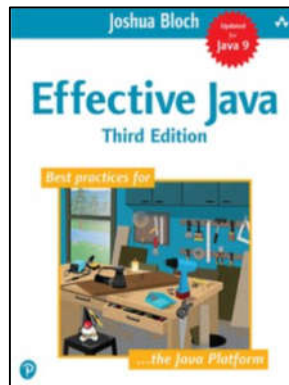
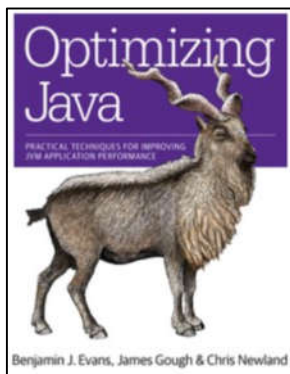


Zahraničná literatúra

1. Learning Java - Marc Loy
2. Java for beginners - Manuj Aggarwal
3. Thinking in Java - Bruce Eckel
4. Java: The Complete Reference - Herbert Schildt



5. Optimizing Java - Benjamin J Evans
6. Effective Java - Joshua Bloch
7. Java Performance - Scott Oaks
8. Head First Java - Kathy Sierra, Bert Bates





IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

info@it-academy.sk



IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

info@it-academy.sk



IT Academy s.r.o., Tomášikova 50/A, 831 04 Bratislava

tel.: 0917/095 406, 0907/375 543

IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

[illegible]