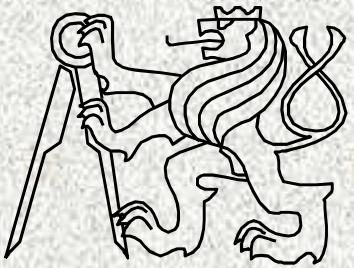
Události



A0B36PR2-Programování 2

Fakulta elektrotechnická
České vysoké učení technické

Události

- Událost
  - Prohlášení akademické obce ČVUT 2. 2. 2012
- Událost
  - **Rektorské volno**
    Rektorské volno od úterý 28.2.2012 16:00 do středy 29.2.2012 14:00.
- Událost
  - Týden neklidu
- Událost
  - demonstrace
- Událost
  - Za svobodné vysoké školy

Zpracování událostí - obsah

1. Připomenutí GUI
2. Co to je událost
3. Koncepce zpracování událostí
4. Zpracování vlastní události
5. *Zpracování události od více zdrojů*
    1. *Rozlišení typem*
    2. *Rozlišení popisem*
6. *Dva posluchači téhož zdroje*
7. Vnitřní třídy
8. Anonymní třídy
9. Příklad na myšku

Zpracování událostí

Mechanismus, který umožní zpracovávat vstupní informace programu, předávané nejčastěji přes GUI

Zpracování vstupní informace v GUI je realizováno:

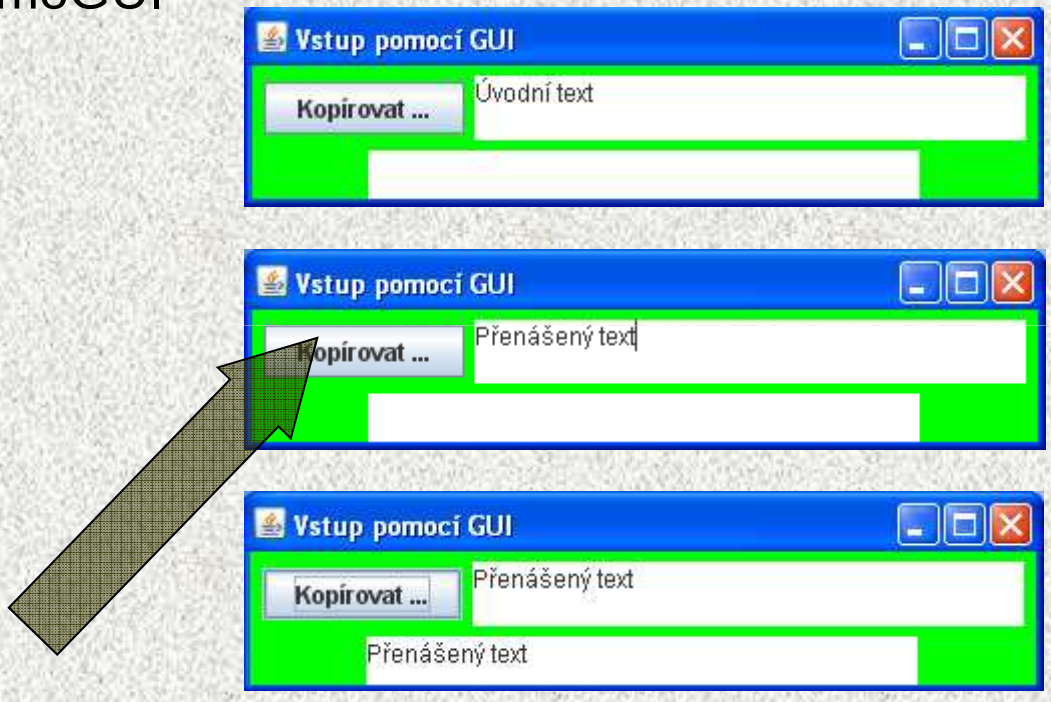
- vysláním „události“ na jedné straně „producentem“
- zachycením této „události“ „posluchačem“

Pojmy:

- Producentem události je třída (GUI), „systém“
- Událostí je objekt této třídy (prvek GUI), „systém“
- Posluchačem je „naše“ třída, kterou to „naučíme“

Zpracování událostí

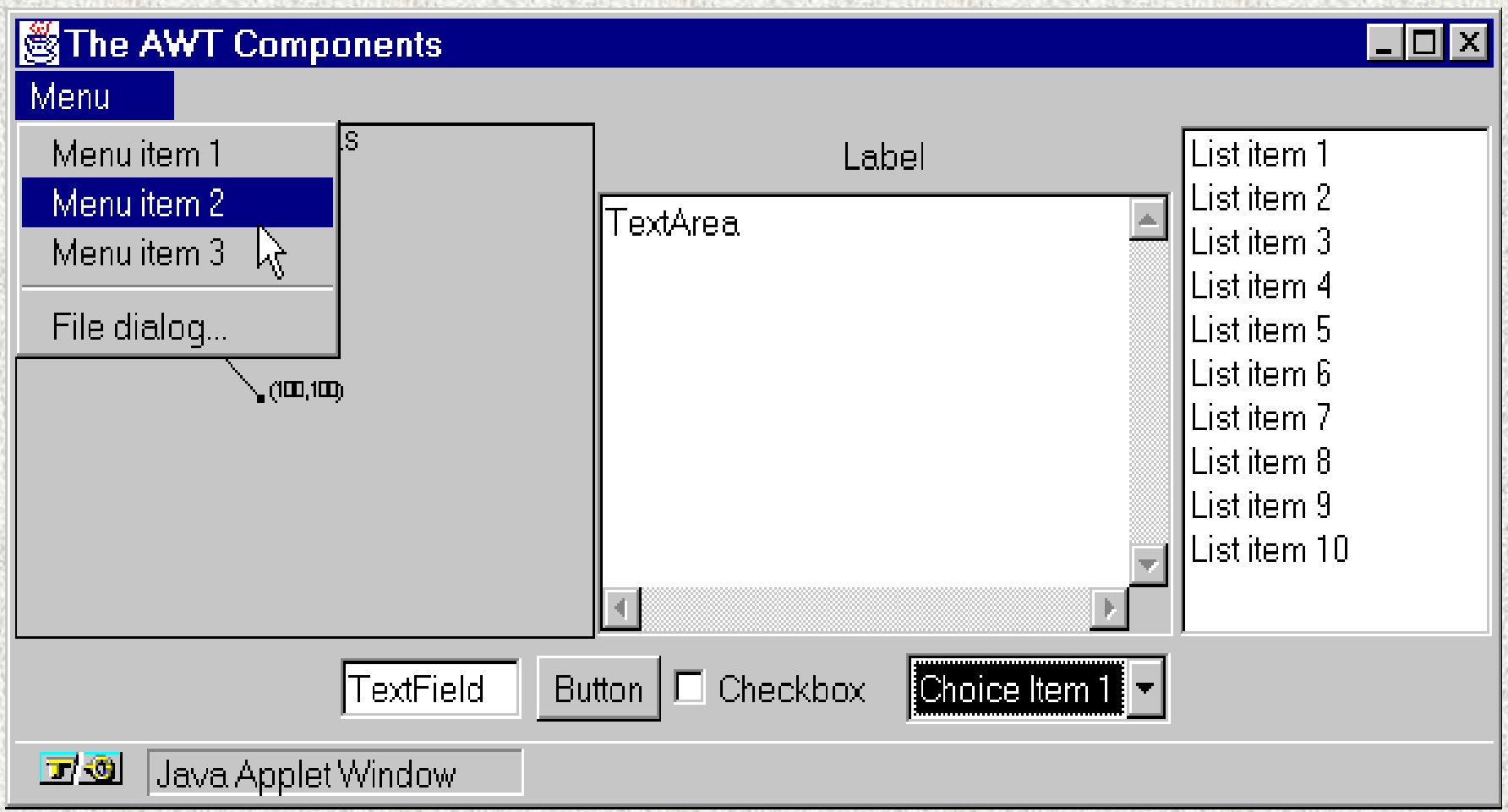
- Grafické uživatelské rozhraní
- Řešení reakce na událost – stisk tlačítka
  - DemoGUI



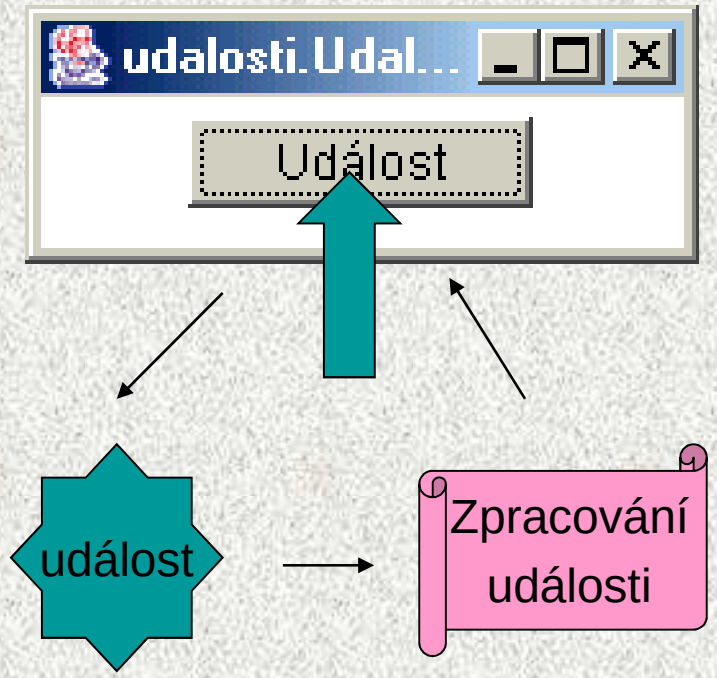
Zpracování událostí

- Událost je objekt, který vznikne změnou stavu zdroje
  - důsledek interakce uživatele s řídícími grafickými elementy GUI
- Událost vznikne:
  - kliknutím na tlačítko
  - stiskem klávesy
  - posunem kurzoru, atd.
- Události jsou produkovány tzv. producenty což jsou:
  - tlačítka
  - rámy
  - ostatními grafické prvky
- Java - promyšlený a konzistentní koncept vzniku a zpracování událostí

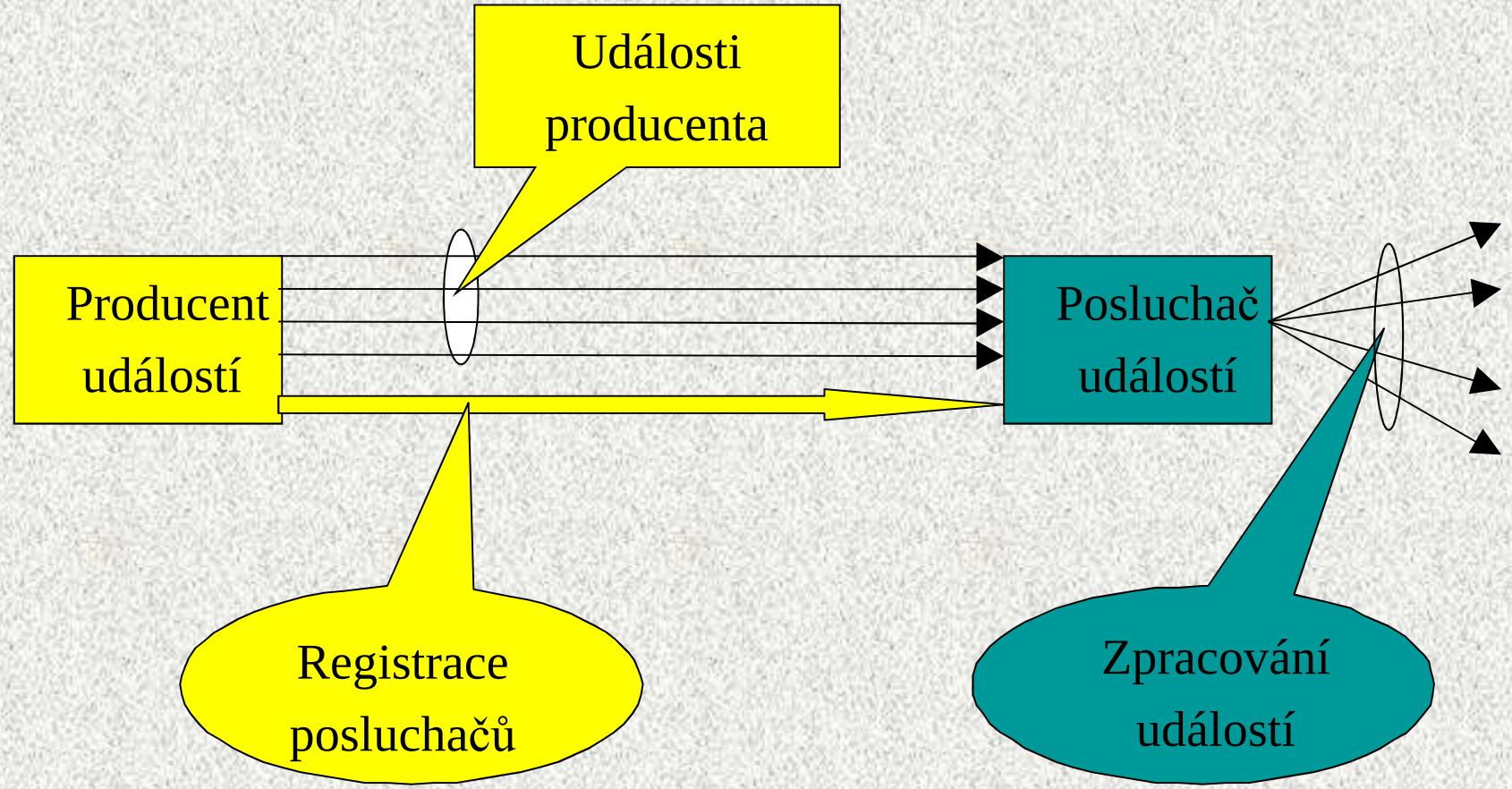
Příklad zdrojů událostí



Zpracování událostí



Model šíření událostí - schéma



Zpracování událostí - koncepce

- Informace o jedné události (zdroj události, poloha kurzoru, atd.) jsou shromážděny **v objektu**, jehož třída určuje charakter události, např.:

  `ActionEvent` ~ událost generovaná tlačítkem

  `WindowEvent` ~ událost generovaná oknem

  `MouseEvent` ~ událost generovaná myší

- Všechny třídy událostí jsou následníky třídy **`event`** a jsou umístěny v

`java.awt.event.*;`

- Základní princip zpracování událostí:
  - Události jsou **generovány zdroji událostí** (jsou to *objekty*, které nesou informaci o události)
  - Události jsou **přijímány** ke zpracování **posluchači událostí** (to jsou opět *objekty* tříd s metodami schopnými událost zpracovat)
  - **Zdroj události rozhoduje o tom, který posluchač má reagovat (registruje si svého posluchače)**

Model šíření událostí

1. Události jsou předávány posluchačům, které si **nejprve musí producent zaregistrovat –** např. metodami:
   `addActionListener(), addWindowListener(), addMouseListener()`
- Producent vysílá **jen těm posluchačům, které si sám zaregistroval**

2. Posluchač musí implementovat některý z posluchačských rozhraní (!) (schopnost naslouchat):
   `ActionListener, WindowListener, MouseListener,` …

- Zatímco událost **producenta** je typicky objekt některé knihovní třídy („od tlačítka“), **posluchač** je objekt, jehož třída je deklarována v aplikaci.
- „Slovy Javy“: zdroj událostí – producent (potomek třídy **`java.awt.AWTEvent`**):
  - zaregistruje si jednu či více tříd svou metodou **`addXXXListener()`**
  - zaregistrovaná třída musí implementovat rozhraní **`XXXListener()`**

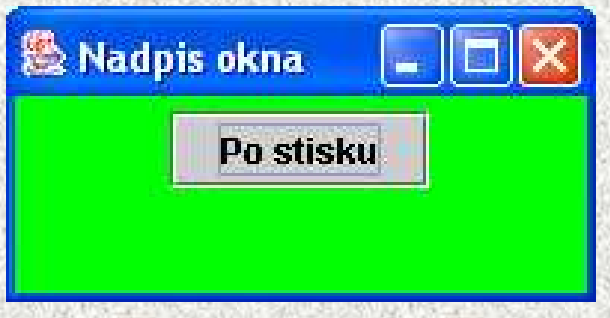
Analogie (Herout):
- *velitel (producent) si musí zaregistrovat své vojáky (posluchače), kteří mají schopnost či kvalifikaci poslouchat (implementují rozhraní)*
- *vojáci (posluchači) si nevybírají své velitele (producenty)*

Schematický příklad zpracování události

```
class Okno extends JFrame{
public Okno (){
…
    FlowLayout srb = new FlowLayout();
    kon.setLayout(srb);
    JButton aTlac = new JButton("Před stiskem");
    kon.add(aTlac);
    aTlac.addActionListener(new Posluchac0());
    setContentPane(kon);          // registrace posluchače
     }
}

 class Posluchac0 implements ActionListener {// posluchač
   public void actionPerformed(ActionEvent e) {
   JButton o=(JButton)e.getSource();//určen objekt události
   o.setLabel("Po stisku");          //reakce na udalost
 }}
```

Schematický příklad zpracování události, výsledek



Implementace modelu událostí

- Posluchač události musí implementovat příslušné rozhraní (interface), tj. implementovat příslušné **abstraktní metody** rozhraní
- Pro **každý druh události je definována abstraktní metoda** *(handler),* která tuto událost ošetřuje

  `actionPerformed, mouseClicked, windowClosing,` atd.
- *Handlery* jsou deklarovány v rozhraních - posluchači:

  `ActionListener, MouseListener, WindowListener,` atd.
- Předání události posluchači ve skutečnosti znamená vyvolání činnosti handleru. Objekt události je předán jako skutečný parametr handleru.

- Producent registruje posluchače zavoláním registrační metody:

`addActionListener, addMouseListener, addWindowListener,` atd.

- Vazba mezi producentem a posluchačem je vztah N:M
  - jeden posluchač může být registrován u více producentů
  - u jednoho producenta může být registrováno více posluchačů.
- Událost se předá všem posluchačům, avšak pořadí zpracování není zaručeno.

Funkcionalita zpracování události, příklad I

```
class Okno1 extends JFrame implements ActionListener {
     JLabel napisX; // třída sama vysílá i zpracovává události
     JButton aTlac;
     Container kon;
public Okno1 (){
[…].
    kon.setLayout(srb);
    aTlac = new JButton(„Změň");
    kon.add(aTlac);
    napisX = new JLabel("NÁPIS");
    kon.add(napisX);
    aTlac.addActionListener(this);
    setContentPane(kon);
     }
   public void actionPerformed(ActionEvent e) {
                                        //implementace
    napisX.setText("JINY");
 }
}
```

Poznámka: Třída Udalosti1 je producentem i posluchačem!!

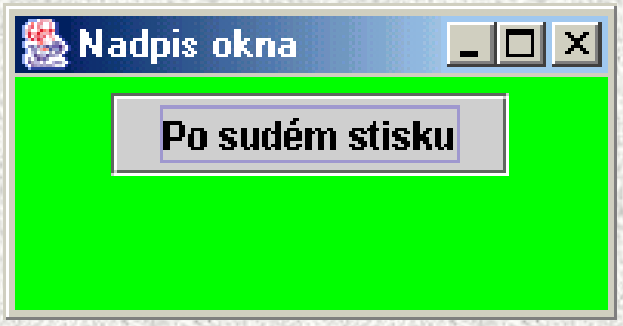
Funkcionalita zpracování události, výsledek

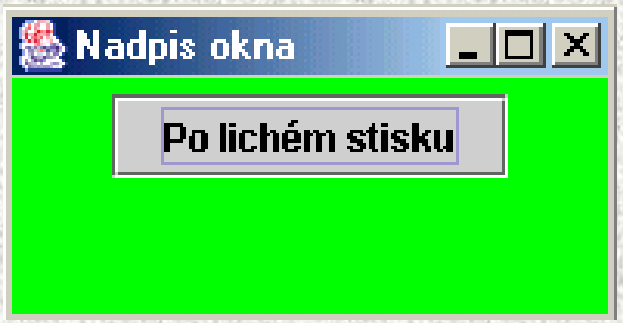
Zpracování vlastností události

```
class Okno2 extends JFrame{
public Okno2 (){
  …
    kon.setLayout(srb);
    JButton aTlac = new JButton("Před stiskem");
    kon.add(aTlac);
    aTlac.addActionListener(new Posluchac2());
    setContentPane(kon);
     }
}

 public void actionPerformed(ActionEvent e) {
        JButton o = (JButton) e.getSource();
        if (!i) {
            o.setText("Po lichém stisku");
            i = !i;
        } else {
            o.setText("Po sudém stisku");
            i = !i;
        }
    }
```

Zpracování vlastností události



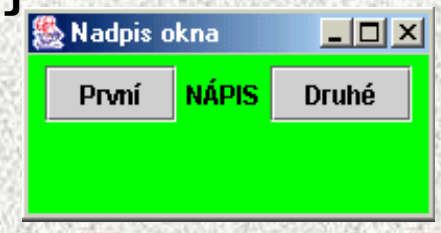


Nebyla tam chyba??

Více zdrojů téže události

```
class Okno3 extends Okno1 { // potomek!!!
  JButton bbTlac;
  Okno3() {
    bbTlac = new JButton("Druhé");
    kon.add(bbTlac);
    bbTlac.addActionListener(this); // zkusit jinak
    setContentPane(kon);
  }

  public void actionPerformed(ActionEvent){
  napisX.setText("JINY");
}}
```

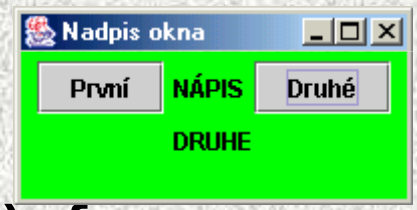


Rozlišení stejných událostí popisem zdroje


```
class Okno4 extends Okno3 {
    JLabel lab;
    Okno4() {
     lab = new JLabel("Nazdar");
     kon.add(lab);
//     bbTlac.addActionListener(this);
       setContentPane(kon);
  }
public void actionPerformed(ActionEvent e) {
String s = e.getActionCommand();
String napis =
        (s.equals(aTlac.getLabel()))?"PRVNI":"DRUHE";
lab.setText(napis);
  }}
```

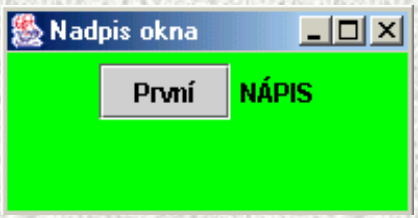
Rozlišení stejných událostí objektem zdroje

```
class Okno5 extends Okno3 {
   JLabel lab;
   Okno5() {
    lab = new JLabel("Nazdar");
    kon.add(lab);
//    bbTlac.addActionListener(this);
      setContentPane(kon);
  }
public void actionPerformed(ActionEvent e) {
Object o = e.getSource();
String napis = (o.equals(aTlac)) ? "PRVNI!":"DRUHE!";
lab.setText(napis);
 }}
```



Dva posluchači jednoho producenta

```
class Okno10 extends Okno1{
  Okno10() {
    aTlac.addActionListener(new Posluchacx2());
    setContentPane(kon);
  }
}
 class Posluchacx2 implements ActionListener {
   public void actionPerformed(ActionEvent e) {
     System.out.print("Pípnuto");
     Toolkit.getDefaultToolkit().beep();
   }
 }
```

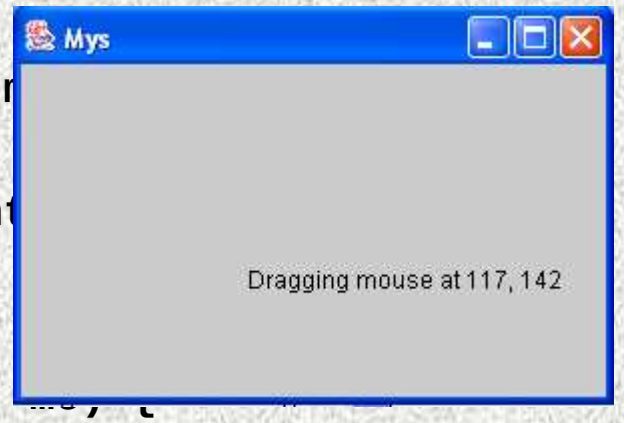


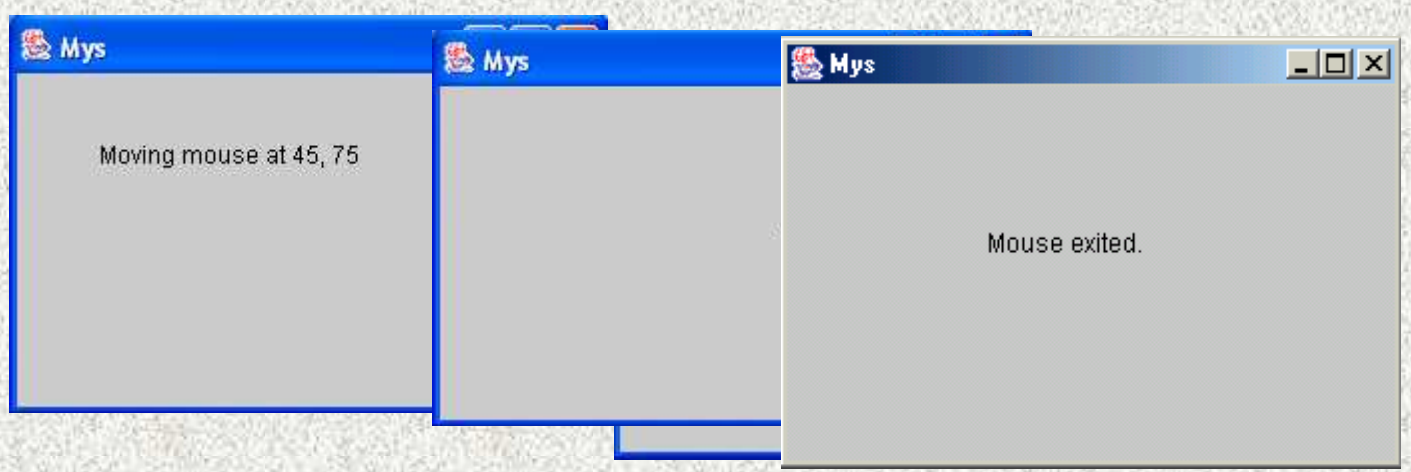
Pípnuto

Události myši

```
implements MouseListener {
        public void mouseClicked(MouseEvent me) {
        public void mouseEntered(MouseEvent
        public void mouseExited(MouseEvent m
        public void mousePressed(MouseEvent
        public void mouseReleased(MouseEvent

implements MouseMotionListener {
        public void mouseDragged(MouseEvent
        public void mouseMoved(MouseEvent me) {
```





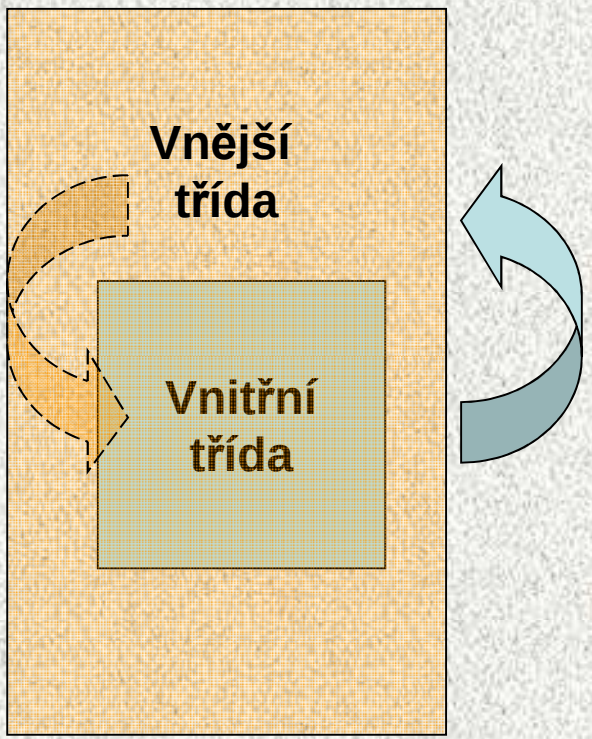
Vnitřní třídy

- Logické seskupení tříd, které se používají jen v jednom místě
  - Efektivita kódu
  - Princip „pomocné“ třídy
- Princip zapouzdření
  - Situace:
    - Třída B potřebuje přístup k členům třídy A, ale které mají být nepřístupny jiným třídám (**`private`**)
    - Je-li B vnitřní třídou třídy A, pak členy **`private`** třídy A jsou přístupné i třídě B
    - Třída B je skryta mimo třídu A
- Zvýšení čitelnosti kódu a jeho lepší údržbu
  - Kód použité třídy je blíže svému použití

Vnitřní třídy

- Prvkem třídy může být jiná třída – vnořená třída
  - Třída, která obsahuje vnořenou třídu – vnější třída
- Vnořená třída
  - Statická vnořená třída – **`static`**
    - Nemůže přímo přistupovat k instančním členům vnější třídy, musí vytvořit její instanci, přes ni má pak přístup
    - Nepoužívají se
  - Vnitřní třída – bez **`static`**
    - **Má přístup ke všem členům vnější třídy, bez vytvoření instance, i k prvkům `private`**
    - Má své vlastní proměnné a metody
    - Nemá statické členy
    - **Vnější třída může do vnitřní jen přes její instanci**
    - **Vnitřní třída není přístupná vně definice vnější třídy, jen v rámci vnější třídy!**
- **Když nás nezajímá jméno vnitřní třídy, můžeme použít anonymní vnitřní třídu**

Vnitřní třídy



Vnitřní třídy v událostech

```
class Okno6 extends Okno3 {
JLabel lab1;
  class Udalost implements ActionListener{
   String vypis;
   public Udalost(String vypis) {
     this.vypis = vypis;
   }
   public void actionPerformed(ActionEvent e) {
     lab1.setText(vypis);
   }}
 Udalost ahojUD, nazdarUD;
 Okno6() {
  lab1 = new JLabel("Nazdar6");
  kon.add(lab1);
   aUD = new Udalost("AHOJ");
   aTlac.addActionListener(aUD);
   nUD = new Udalost("NAZDAR");
   bbTlac.addActionListener(nUD);
 }}
```

vnitřní třída

Vnitřní třídy v událostech

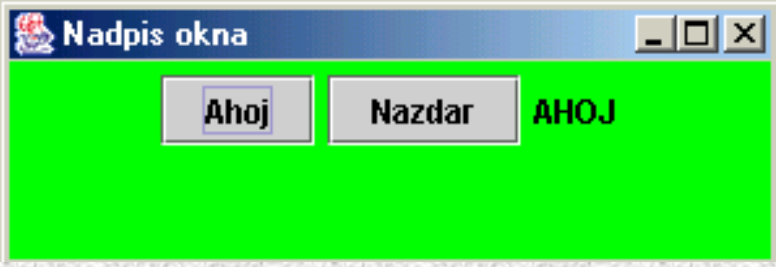
Použití více vnitřních tříd - doporučený způsob

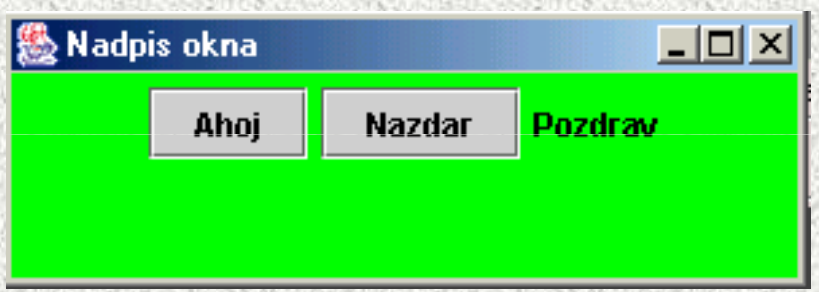
```
public Okno7() {
  …
    JButton aTlac = new JButton("Ahoj");
    kon.add(aTlac);
    aTlac.addActionListener(new AhojBTAL());
    JButton bbTlac = new JButton("Nazdar");
    kon.add(bbTlac);
    bbTlac.addActionListener(new NazdarBTAL());
    lab = new JLabel("Nazdar");
    kon.add(lab);
    setContentPane(kon);
  }
  class AhojBTAL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      lab.setText("AHOJ");
    }}
  class NazdarBTAL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      lab.setText("NAZDAR");
    }}}
```

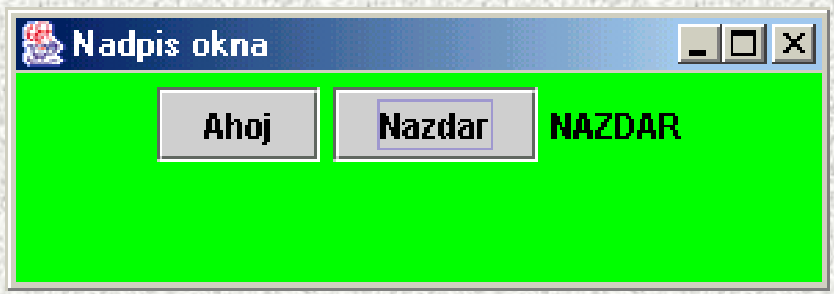
Vnitřní třída

Vnitřní třída

Použití více vnitřních tříd







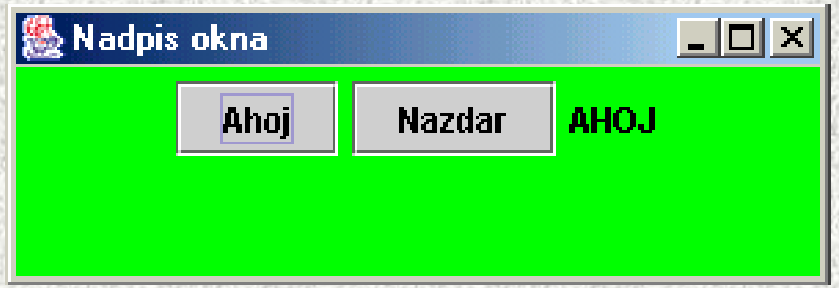
Anonymní vnitřní třídy

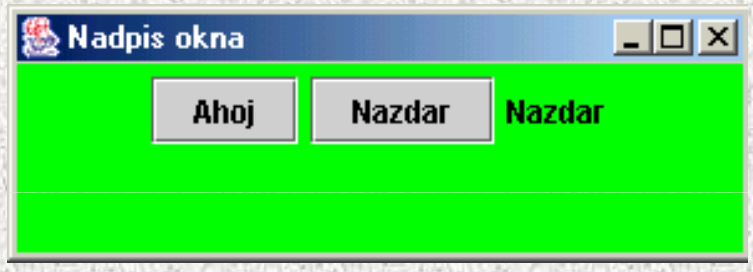
```
public Okno8()
…
    JButton aTlac = new JButton("Ahoj");
    kon.add(aTlac);
    aTlac.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        lab.setText("AHOJ");
      }
    });
    JButton bbTlac = new JButton("Nazdar");
    kon.add(bbTlac);
    bbTlac.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        lab.setText("NAZDAR");
      }
    });
    lab = new JLabel("Nazdar");
    kon.add(lab);
    setContentPane(kon);
  }}
```

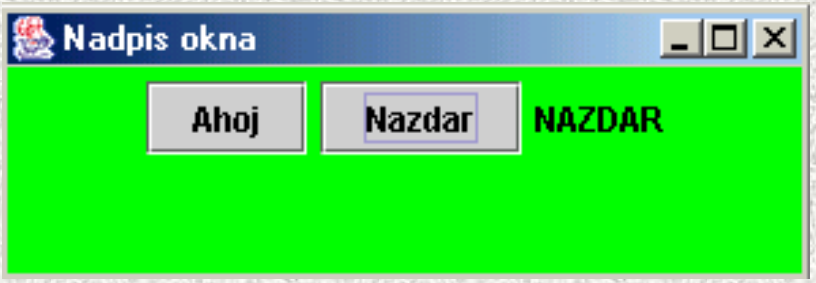
Vnitřní anonymní třída

Vnitřní anonymní třída

Anonymní vnitřní třídy







Další možnosti

- Zrušení posluchače – už nechceme reakce

`removeActionListener(ActionListener posluchac)`

- Programové vyvolání události – vyvolání události jinak než „tlačítkem“
- Více metod rozhraní posluchače – např. u událostí myši – nutnost implementovat všechny metody rozhraní!!
  - **`mousePressed(), mouseRealased(), mouseClicked(),`** …
  - možnost použití tzv. adaptérů
    - Implementace rozhraní s prázdnými metodami, uživatel není nucen všechny metody rozhraní implementovat

- Podrobný přehled událostí – viz dokumentace

Vnořené třídy

- Prvkem třídy může být jiná třída – vnořená třída
  - Třída, která obsahuje vnořenou třídu – vnější třída
- Vnořená třída
  - Statická vnořená třída – **`static`**
  - Vnitřní třída – bez **`static`**
- Vlastnosti vnitřní třídy:
  - použitelná pouze v rámci vnější třídy
  - vnořená třída může neomezeně přistupovat ke strukturám vnější třídy, i když jsou **`private`**
  - vnější třída nemá přístup do vnitřní třídy (ani jiná zvenku)
    - Není ani možný přístup přes vlastní referenční proměnnou
- Použití vnitřní třídy
  - Metoda rozhraní bude přístupná jen přes referenční proměnnou typu rozhraní (přes ref. proměnnou na vnější třídu nebude přístup k implementované metodě rozhraní)
  - Když nás nezajímá jméno vnitřní třídy, můžeme použít anonymní vnitřní třídu

Jiné použití vnitřní třídy

Situace:

- Metody rozhraní můžeme
  - volat přes referenční proměnnou třídy, která rozhraní implementuje
  - volat přes referenční proměnnou typu rozhraní
- Chceme, aby metody rozhraní šlo volat jen prostřednictvím referenční proměnné typu rozhraní
  - Chceme zabránit tomu, abychom mohli volat metodu rozhraní přes proměnnou třídy
  - Chceme, aby metody rozhraní šly volat jen přes referenční proměnnou rozhraní (a do této proměnné můžeme dynamicky uložit odkaz na objekt libovolné třídy, která toto rozhraní implemetuje)

Příklad implementace rozhraní vnitřní třídou

```
class Usecka {
    int delka;
    Usecka(int delka) {
        this.delka = delka;
    }
    public Info informace() {
        return new UseckaInfo();
    }
    class UseckaInfo implements Info {
        public void kdoJsem() {
            System.out.println("Usecka " + delka);
        }
    }}

public class TestVnitrniTrida {
    public static void main(String[] args) {
        Usecka u = new Usecka(5);
//    u.kdoJsem();   // nelze
//    Info i = u;    // nelze
        Info i = u.informace();
        i.kdoJsem();
    }}
-----------------
 public interface Info {
//  public void kdoJsem();
  void kdoJsem();
}
```

Vnitřní třída

Příklad implementace rozhraní anonymní třídou

```
class Usecka {
    int delka;
    Usecka(int delka) {
        this.delka = delka;
    }
    public Info informace() {
        return new Info() {
            public void kdoJsem() {
                System.out.println("Usecka " + delka);
            }
        };
}}

public class AnonymniTrida {
    public static void main(String[] args) {
        Usecka u = new Usecka(5);
//      u.kdoJsem();                    // chyba
        Info i = u.informace();
        i.kdoJsem();
    }
}
```

Anonymní třída