GUI v Javě

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 3

A0B36PR2 – Programování 2

Část 1 – Příklad - Generické typy, iterátor

Část 2 – GUI v Javě

Část I

Příklad - Generické typy, iterátor

Generické typy a nevýhody polymorfismu

- Flexibilita (znovupoužitelnost) tříd je tradičně v Javě řešena dědičností a polymorfismem
- Polymorfismus nám dovoluje vytvořit třídu (např. nějaký kontejner), která umožňuje uložit libovolný objekt (jako referenci na objekt typu **Object**)

Např. **ArrayList** z JFC

- Dynamická vazba polymorfismu však neposkytuje kontrolu správného (nebo očekávaného) typu během kompilace
- Případná chyba v důsledku „špatného" typu se tak projeví až za běhu programu
- Tato forma polymorfismu také vyžaduje explicitní přetypování objektu získaného z nějaké takovéto obecné kolekce

Příklad použití kolekce **ArrayList**

```
package cz.cvut.fel.pr2;

import java.util.ArrayList;

public class Simulator {

World world;
ArrayList participants;

Simulator(World world) {
 this.world = world;
 participants = new ArrayList();
}

public void nextRound() {
 for (int i = 0; i < participants.size(); ++i) {
   Participant player = (Participant) participants.get(i);
   Bet bet = world.doStep(player);
 }

}
```

- Explicitní přetypování (Participant) je nutné.

Generické typy

- Java 5 dovoluje použít generických tříd a metod
- Generický typ umožňuje určit typ instance tříd, které lze do kolekce ukládat
- Generický typ tak poskytuje statickou typovou kontrolu během překladu

- Generické typy představuji parametrizované definice třídy typu nějaké datové položky
- Parametr typu se zapisuje mezi <>, například

```
List<Participant> partList = new ArrayList<Participant>();
```

http://docs.oracle.com/javase/tutorial/java/generics/index.html

Příklad použití parametrizované kolekce **ArrayList**

```
package cz.cvut.fel.pr2;

import java.util.ArrayList;

public class Simulator {

World world;
ArrayList<Participant> participants;

Simulator(World world) {
 this.world = world;
 participants = new ArrayList();
}

public void nextRound() {
 for (int i = 0; i < participants.size(); ++i) {
   Participant player = participants.get(i);
   Bet bet = world.doStep(player);
 }

}
```

- Explicitní přetypování (Participant) není nutné

Příklad – generický a negenerický typ

```
ArrayList participants;
participants = new ArrayList();
participants.push(new PlayerRed());

// vlozit libovolny objekt je mozne
participants.push(new Bet());


ArrayList<Participant> participants2;
participants2 = new ArrayList<Participant>();
participants2.push(new PlayerRed());

// nelze prelozit
// typova kontrola na urovni prekladace
participants2.push(new Bet());
```


Příklad parametrizované třídy

```
import java.util.List;
import java.util.ArrayList;

class Library<E> {
   private List<E> resources = new ArrayList<E>();

   public void add(E x) {
      resources.add(x);
   }

   public E getLast() {
      int size = resources.size();
      return size > 0 ? resources.get(size-1) : null;
   }
}
```


Generické metody

- Generické metody mohou být členy generických tříd nebo normálních tříd

```
public class Methods {
   public <T> void print(T o) {
      System.out.println("Print Object: " + o);
   }
   public static void main(String[] args) {
      Integer i = 10;
      Double d = 5.5;

      Methods m1 = new Methods();

      m1.print(i);
      m1.print(d);

      m1.<Integer>print(i);

      /// nelze -- typova kontrola
      m1.<Integer>print(d);
   }
}
```

lec03/Methods

Příklad implementace spojového seznamu

- Třída **LinkedList** pro uchování objektů
- Implementujeme metody **push** a **print**

```
public class LinkedList {
   class ListNode {
      ListNode next;
      Object item;
      ListNode(Object item) { ... }
   }

   ListNode start;

   public LinkedList() { ... }
   public LinkedList push(Object obj) { ... }
   public void print() { ... }
}
```

lec03/LinkedList

Příklad použití

- Do seznamu můžeme přidávat libovolné objekty, např. **String**
- Tisk seznamu však realizuje vlastní metodou **print**

```
LinkedList lst = new LinkedList();
lst.push("Joe");
lst.push("Barbara");
lst.push("Charles");
lst.push("Jill");

lst.print();
```

- Využití konstrukce **for-each** vyžaduje, aby třída **LinkedList** implementovala rozhraní **Iterable**

```
for (Object o : lst) {
   System.out.println("Object:" + o);
}
```


Rozhraní **Iterable** a **Iterator**

- Rozhraní **Iterable** předepisuje metodu **iterator**, která vrací iterátor instanci třídy implementující rozhraní **Iterator**
- **Iterator** je objekt umožňující postupný přístup na položky seznamu
- Rozšíříme třídu **LinkedList** o implementaci rozhraní **Iterable** a vnitřní třídu `LLIterator` implementující rozhraní **Iterator**

http://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html

```
public class LinkedListIterable extends LinkedList
    implements Iterable {

   private class LLIterator implements Iterator { ... }

   @Override
   public Iterator iterator() {
      return new LLIterator(start); //kurzor <- start
} }
```

lec03/LinkedListIterable

Implementace rozhraní **Iterator**

- Rozhraní **Iterator** předepisuje metody **hasNext** a **next**

```
private class LLIterator implements Iterator {
   private ListNode cur;

   private LLIterator(ListNode cur) {
      this.cur = cur; // nastaveni kurzoru
   }

   @Override
   public boolean hasNext() {
      return cur != null;
   }

   @Override
   public Object next() {
      if (cur == null) {
         throw new NoSuchElementException();
      }
      Object ret = cur.item;
      cur = cur.next; //move forward
      return ret;
   }
}
```

lec03/LinkedListIterable

Příklad využití iterátoru v příkazu **for-each**

- Nahradíme implementace **LinkedList** za **LinkedListIterable**

```
// LinkedList lst = new LinkedList();
LinkedListIterable lst = new LinkedListIterable();
lst.push("Joe");
lst.push("Barbara");
lst.push("Charles");
lst.push("Jill");

lst.print();

for (Object o : lst) {
   System.out.println("Object:" + o);
}
```

lec03/LinkedListDemo

Spojový seznam specifických objektů

- Do spojového seznamu **LinkedList** můžeme ukládat libovolné objekty, což má i přes své výhody také nevýhody:
  - Nemáme statickou typovou kontrolu prvků seznamu
  - Musíme objekty explicitně přetypovat, například pro volání metody **toNiceString** objektu **Person**

```
public class Person {

   private final String name;
   private final int age;

   public Person(String name, int age) { ... }
   public String toNiceString() {
      return "Person name: " + name + " age: " + age;
   }
}
```


Příklad přetypování na **Person**

```
LinkedListIterable lst = new LinkedListIterable();
lst.push(new Person("Joe", 30));
lst.push(new Person("Barbara", 40));
lst.push(new Person("Charles", 50));
lst.push(new Person("Jill", 60));

for (Object o : lst) {
   System.out.println("Object: " + ((Person)o).
    toNiceString());
}
```


Generický typ

- Využitím generického typu můžeme předepsat konkrétní typ objektu
- Vytvoříme proto LinkedList přímo jako generický typ deklarací `class LinkedListGeneric<E>` a záměnou **Object** za **E**

%s/Object/E

```
public class LinkedListGeneric<E> {
   class ListNode {
      ListNode next;
      E item;
      ListNode(E item) { ... }
   }
   ListNode start
   public LinkedListGeneric() { ... }
   public LinkedListGeneric push(E obj) { ... }
   public void print() { ... }
}
```

lec03/LinkedListGeneric

Generický typ – **Iterable** a **Iterator**

- Podobně upravíme odvozený iterátor a doplníme typ také v rozhraní **Iterable** a **Iterator**

```
public class LinkedListGenericIterable<E> extends
    LinkedListGeneric<E> implements Iterable<E> {

  // vnitni trida pro iterator
  private class LLIterator implements Iterator<E> { ... }

  @Override
  public Iterator iterator() {
     return new LLIterator(start);
  }
}
```

lec03/LinkedListGenericIterable

Generický typ – **Iterator**

- Implementace iterátoru je identická jako v případě LinkedListIterable

```
private class LLIterator implements Iterator<E> {

   private ListNode cur;

   private LLIterator(ListNode cur) {
      this.cur = cur;
   }
   @Override
   public boolean hasNext() {
      return cur != null;
   }
   @Override
   public E next() {
      if (cur == null) {
         throw new NoSuchElementException();
      }
      E ret = cur.item;
      cur = cur.next; //move forward
      return ret;
   }
}
```

lec03/LinkedListGenericIterable

Příklad použití

```
LinkedListGenericIterable<Person> lst = new
    LinkedListGenericIterable();

lst.push(new Person("Joe", 30));
lst.push(new Person("Barbara", 40));
lst.push(new Person("Charles", 50));
lst.push(new Person("Jill", 60));

lst.print();

for (Person o : lst) {
   System.out.println("Object: " + o.toNiceString());
}
```

lec03/LinkedListGenericDemo

Část II

GUI v Javě

Grafické uživatelské rozhraní

- GUI – Graphical User Interface
- Zásadním způsobem ovlivňuje použitelnost, přívětivost aplikace a také produktivitu

  User experience
- Elegantní návrh s intuitivní a **konzistentní** funkcionalitou
- Respektujte styl a **zvyklosti** uživatele

  Cílová skupina laik vs expert
- Jednoduchost bývá zpravidla lepší než složité komponenty

  Vytvořit jednoduché a dobře použitelní rozhraní je zpravidla výrazně časově náročnější než se na první pohled zdá.
- Klíčová je zpětná vazba od uživatelů a testování

  „Testováno na lidech!"
- Návrh **dobrého** rozhraní je o rozložení grafických prvků, volbě barev a tvarů, vizualizačních efektech, písmu, ...

Programování a tvorba grafického rozhraní

- Z programátorského hlediska se však vždy v podstatě jedná o zadání vstupu a prezentaci výstupu
- Pro interakci s uživatelem lze využít sadů základních grafických komponent tzv. **Widgets**
- Softwarová knihovna pro tvorbu rozhraní se nazývá **Widget toolkit** nebo grafický **toolkit**
- Klíčem k jednoduchosti, použitelnosti a také přenositelnosti mezi platformami je unifikace grafických prvků

- Velkou výhodou Javy je, že knihovny pro grafické prvky jsou součástí standardního JDK

Grafické knihovny v Javě

- AWT – Abstract Window ToolKit (První gui v Javě – `heavyweight`)
  - Vykreslování zajišťuje hostitelská platform, na které běží JVM

    Vykreslování je tak rychlejší, ale vše nemusí fungovat identicky na jiných platformách
- Swing – Výrazné rozšíření (a zlepšení) GUI (oproti AWT)
  - Doporučené standardní GUI v Javě
  - *Look&Feel* je platformově nezávislý a respektuje **i18n**

    i18n – i-internationalizatio-n
  - Důsledné oddělení modelu od pohledu

    http://docs.oracle.com/javase/tutorial/uiswing
- JavaFX – nový GUI ToolKit (následovník Swing)
  - Styl vzhledu přes CSS *HMTL rendering engine (WebKit)*

    http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

- SWT – Standard Widget Tookit (Eclipse)
  - Platformově závislý, ale unifikuje vzhled

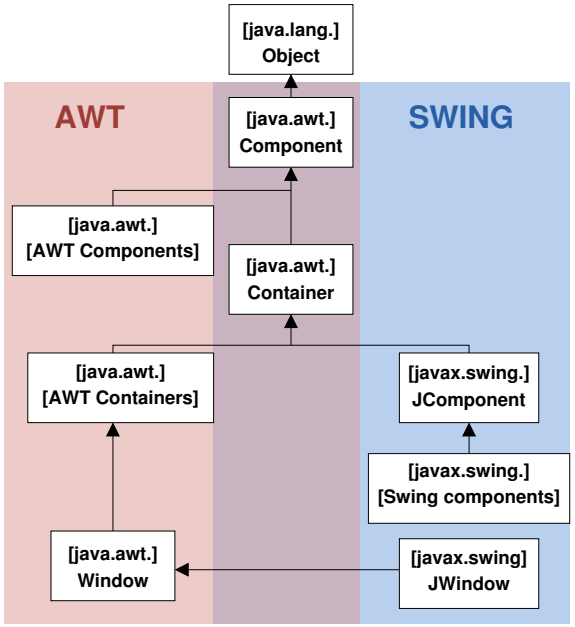
    https://www.eclipse.org/swt

Základní prvky grafického rozhraní

- **Komponenty** – tlačítka, textová pole, menu, posuvníky, ...
- **Kontejnery** – komponenty, do kterých lze vkládat komponenty

  Například pro rozdělení plochy a volbu rozmístění
- **Správce rozvržení** (*Layout manager*) – rozmísťuje komponenty v ploše kontejneru
- Interakce s uživatelem dále zpravidla vyžaduje mechanismus událostí a jejich zachytávání

AWT a Swing



Základní součásti GUI

- **Komponenty** a dialogové prvky
  javax.swing
  - Tlačítka, text, textová pole, seznamy, přepínače
  - Společné metody pro velikost, barvu, umístění textu, ...
- **Kontejnery** (v oknech, která zpravidla řeší prostředí OS)
  javax.swing
  - Komponenty obsahují komponenty
    Komponenty musí být umístěny v kontejneru
  - Kontejnery se vkládají do oken
  - **JFrame** – obecný kontejner
  - **JPanel** – kontejner po jednoduché komponenty
- **Layout Manager** – Správce rozmístění
  javax.swing a **java.awt**
  - Definuje pozici komponent v kontejneru
  - Relativní k okrajům, pevná pozice, v mřížce, ...
  - Určuje vzhled a chování aplikace
- **Events** – Obsluha událost (**java.awt.event**)

Přehled základních grafických komponent

Komponenty

- `JLabel` – Zobrazení popisku, bez generování události
- `JButton` – Tlačítko s událostí kliknutí na tlačítko
- `JTextField` – Zadání textu
- `JPasswordField` – Zadání textu (hesla), vložené znaky se zobrazují jako hvězdičky
- `JList` – Seznam položek, možnost vybrat jednu nebo více položek
- `JComboBox` – Rozevírací seznam položek, klepnutím na položku se generuje událost
- `JCheckBox` – Zaškrtávací políčko, prvek je/není vybrán
- `JRadioButton` – Přepínač, výběr z možností

Kontejnery a správce rozvržení

- **JFrame** – Kontejner s ohraničením a záhlavím

  `http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html`

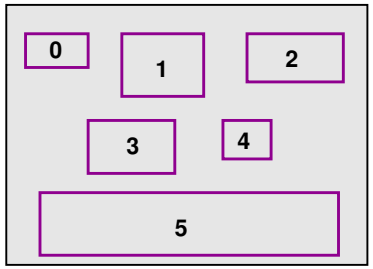
- **JPanel** – Kontejner bez ohraničení, implicitně rozmístění `FlowLayout`
  Může být jednodušší na použití

  `http://docs.oracle.com/javase/tutorial/uiswing/components/panel.html`

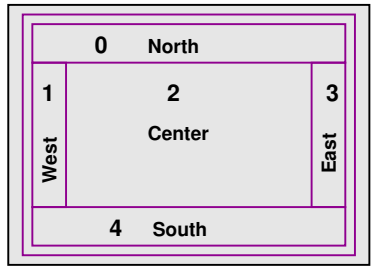
- **Layout Manager** (správce rozvržení)
  - `BorderLayout` – Rozmístění podle okrajů okna (panelu/kontejneru)
  - `BoxLayout` – Rozmístění do podkontejnerů, sdružování komponent
  - `FlowLayout` – Rozmístění zleva doprava a shora dolů
  - `GridLayout` – Rozmístění do pevné mřížky

  `http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html`

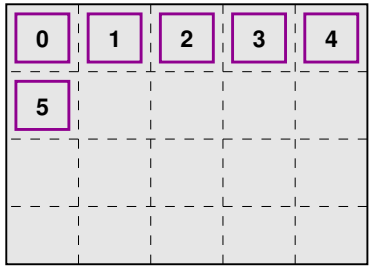
Správci rozmístění komponent – Layout Manager



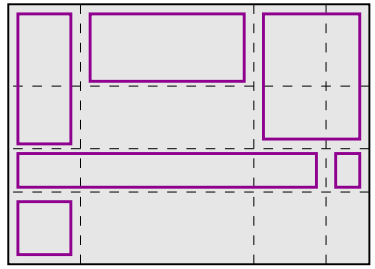
`FlowLayout`



`BorderLayout`



`GridLayout`



`GridBagLayout`

Příklad okna a vložení komponenty (JLabel)

- JFrame Swing – „Hello World"


```
// okno a jeho titulek
JFrame frame = new JFrame("HelloWorldSwing");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JLabel label = new JLabel("Hello World");
frame.getContentPane().add(label, BorderLayout.NORTH);

frame.pack();  //nastav velikost okna
frame.setVisible(true); //zobrazi okno
```

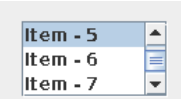
Metoda `demo` v `lec03/DemoGuiComponents`

Řídicí komponenty 1/2

- Tlačítka
  - JButton – „zvonková"
  - JToggleButton – přepínací
  - JCheckBox – zaškrtávací
  - JRadioButton a ButtonGroup

http://docs.oracle.com/javase/tutorial/uiswing/components/button.html

- JList – seznam
  SINGLE_SELECTION,
  SINGLE_INTERVAL_SELECTION,
  MULTIPLE_INTERVAL_SELECTION



http://docs.oracle.com/javase/tutorial/uiswing/components/list.html

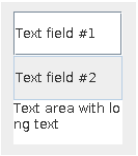
- JComboBox – seznam rozbalovací

http://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html

lec03/DemoGuiComponents

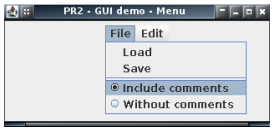
Řídicí komponenty 2/2

- JTextField – vstupní pole pro data (editovatelné nebo needitovatelné)
  - TextArea



http://docs.oracle.com/javase/tutorial/uiswing/components/textfield.html

- JMenuBar, JMenu, JMenuItem
  - JRadioButtonMenuItem, ButtonGroup



http://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

- JSlider

http://docs.oracle.com/javase/tutorial/uiswing/components/slider.html

lec03/DemoGuiComponents

Vlastní grafika v Javě – Plátno (Canvas)

- Základní třídy **java.awt.Graphics**, **java.awt.Graphics2D**
- Základní možnosti třídy **Graphics**: *Již od JDK ver. 1.2*
  - Kreslení základních 2D objektů (grafických primitiv)
  - Vykreslování textu a obrázků
  - Nastavování a testování barev, fontů, ořezání, ...
- Okamžik zobrazení „není" časově určen
- Kreslit lze v komponentách JPanel a JFrame
- Vykreslování probíhá v grafickém kontextu tvořeného třídou **Graphics**
  - Grafický kontext je parametrem (zděděné) metody **Container.paint(Graphics g)**, ve které probíhá vlastní kreslení do kontextu („plátna")
  - Definuje počáteční vykreslení, nevolá se přímo
    Třída Graphics je abstraktní, předávaný objekt g je „automatický" objekt, o který se nestaráme.
- Překreslování je realizováno metodami **repaint** a **update**

http://docs.oracle.com/javase/tutorial/uiswing/painting

Příklad vykreslení grafických primitiv 1/3

```
public class Canvas extends JFrame {
   public Canvas() {
      setTitle("PR2 Demo Canvas");
      setSize(640, 480);
      setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }

   @Override
   public void paint(Graphics g) { ... }
}
```

lec03/Canvas

Příklad vykreslení grafických primitiv 2/3

```
@Override
public void paint(Graphics g) {
   Graphics2D g2d = (Graphics2D) g;
   g2d.setColor(Color.RED);
   g2d.fillOval(110, 210, 30, 30);
   g2d.drawOval(360, 320, 30, 30);

   g2d.setColor(Color.BLUE);
   g2d.fillRect(150, 50, 30, 30);

   g2d.drawPolygon(
         new int[]{200, 250, 300, 290, 180},
         new int[]{150, 200, 180, 210, 240},
         5);
   g2d.draw(new Ellipse2D.Double(320, 240, 30, 30));
}
```

lec03/Canvas

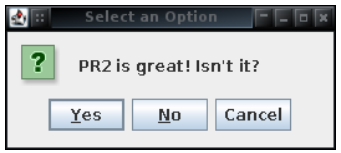
Příklad vykreslení grafických primitiv 3/3

```
public class DemoGuiCanvas {

   public void demo() {
      Canvas canvas = new Canvas();
      canvas.show();
   }

   public static void main(String[] args) {
      DemoGuiCanvas gui = new DemoGuiCanvas();
      gui.demo();
   }

}
```

lec03/DemoGuiCanvas

Dialogové okno

- Dialogové okno je dočasné „nezávislé" okno zpravidla vyžadující interakci uživatele
- Slouží pro informování uživatele nebo pro získání uživatelské vstupu



http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

- Je možné je vyvolat metodami třídy **JOptionPane**, např.
  - `showMessageDialog`
  - `showConfirmDialog`
  - `showInputDialog`

Příklad dialogového okna

```
import javax.swing.JOptionPane;


JOptionPane.showMessageDialog(null, "Message");

int answr =  //0 - Yes, 1 - No, 2 - Cancel
   JOptionPane.showConfirmDialog(null, "Config?");

String str =
   JOptionPane.showInputDialog(null, "Entry");
```

lec03/DemoDialog

Dialog zobrazení informace

- Zobrazení informace můžeme anotovat podle významu
  - ERROR_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE

```
int response = JOptionPane.showConfirmDialog(null, "PR2 is great
    ! Isn't it?");
switch (response) {
   case 0:
      JOptionPane.showMessageDialog(null, "You are right!",
            "Confirm", JOptionPane.PLAIN_MESSAGE);
      break;
   case 1:
      JOptionPane.showMessageDialog(null, "You are wrong!",
            "Error", JOptionPane.ERROR_MESSAGE);
      break;
   case 2:
      JOptionPane.showMessageDialog(null, "You should know!",
            "Warn", JOptionPane.WARNING_MESSAGE);
      break;
}
```

lec03/DemoDialog

Modalita dialogových oken

- Modalita dialogu určuje, zda-li dialogové okno blokuje ostatní okna
- **Dialog.ModalityType**
  - APPLICATION_MODAL
  - DOCUMENT_MODAL
  - MODELESS
  - TOOLKIT_MODAL
- Volíme dle typu aplikace a dialogu např.:
  - Jedno hlavní okno, ostatní dialogová okna slouží pro zadání vstupu nebo informování uživatele (např. výběr souboru), po uzavření přecházíme do hlavního okna
  - Více „hlavních" oken, kterými procházíme a vždy pracujeme pouze s jedním oknem
  - Více „plovoucích" nezávislých oken

Modeless

http://docs.oracle.com/javase/tutorial/uiswing/misc/modality.html

Příklad modálního a nemodálního okna

```
final JFrame parent = new JFrame("Parent Frame");
parent.setLayout(new FlowLayout());
parent.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
parent.setBounds(100, 100, 300, 200);
parent.add(new JButton("Button"));

parent.setVisible(true);

JDialog dialog1 = new JDialog(parent, "Modeless Dialog");
dialog1.setBounds(200, 200, 300, 200);
dialog1.setVisible(true);

JDialog dialog2 = new JDialog(parent,
      "Document-Modal Dialog",
      Dialog.ModalityType.DOCUMENT_MODAL);
dialog2.setBounds(300, 300, 300, 200);
dialog2.setVisible(true);
```

lec03/DemoModality

Zpracování událostí

- Interakce uživatele s rozhraním vyvolává události, na které je potřeba reagovat
- Dialogová okna (modální) představují synchronní mechanismus, kdy je běh aplikace „pozastaven" a aplikace čeká na uživatelský vstup
- Zpravidla, chceme uživatelům umožnit vyšší interaktivitu a s tím související „nezávislé" generování událostí

- Generované události je však nutné zpracovávat

Obsluha událostí

- Mechanismus reakce na akci uživatele, např.
  - Stisk tlačítka, zadání textu, pohyb kurzoru

    Množinu možných typů událostí definuje Toolkit a souvisí s rozhraním (uživatelským) počítače

- Pro každou komponentu je nutné
  1. Deklarovat typ zachytávané událost, kterou chceme zpracovávat
  2. Určit „posluchače", který má událost obsloužit
- Akcí uživatele vznikne událost, která je objektem Javy
- Zachycené události
  - jsou zpracovány (obslouženy) „posluchači" (**listener**)
    - Třídami s **uživatelskými metodami pro reakci na událost**
  - „**posluchači**", které implementují rozhraní „naslouchání"

    Tj. musejí mít schopnost naslouchat dané události

Obsluha souvisí s tzv. Event-driven programováním, které je náplní 4. přednášky

Příklad zpracování stisku tlačítka

```
JFrame frame = new JFrame("PR2 - GUI button click demo");
Container pane = frame.getContentPane();

JButton printButton = new JButton("Print");
printButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      System.out.println("User click to print");
   }
});

JButton exitButton = new JButton("Quit");
exitButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      System.out.println("User click to exit");
      System.exit(0);
   }
});

pane.add(printButton);
pane.add(exitButton);
```

lec03/DemoButtonEvent

Shrnutí přednášky

Diskutovaná témata

- Kolekce – **Java Collection Framework** (JFC)
  - generické typy
- GUI v Javě
  - Komponenty a kontejnery
  - Dialogová okna (modalita)
  - Události a obsluha událostí (nástin)

- Příště: GUI v Javě a událostmi řízené programování (Even-Driven Programming)