Seminář Java

VIII

Rekapitulace

- Grafické uživatelské rozhraní
- Swing vs AWT
- Aplety
  - Aplikační rámec, `JApplet`
  - spouštení v prohlížeči, `Appletviewer`
- Událostní model knihovny Swing
  - události
  - posluchači
- Jednotlivé komponenty knihovny Swing
  - přehled, příklady
- Rozmístění komponent
  - Správci rozmístění
- Vizuální programování a komponenty JavaBeans

Obsah

- Podprocesy, vlákna
  - Třida `Thread`, rozhraní `Runnable`
- Podprocesy typu Daemon
- Sdílení prostředků
- Blokování procesů
- Priority procesů
- Skupiny procesů

Procesy, podprocesy (vlákna)

- **Proces** - samostatný spuštěný program s vlastním adresovým prostorem.
- **Podproces (vlákno)** - samostatná, nezávisle na sobě spuštěná vedlejší úloha.
- **Multithreading** - technika rozdělení programu na více podprocesů
  - slouží k oddělení části programu, vázané na určité prostředky
  - vytváří podproces nezávislý na hlavním programu

Vytvoření vlákna

- vytvořením potomka třídy `Thread`
- implementováním rozhraní `Runnable`

Obojí obsaženo v balíku **java.lang**

Třída `Thread`

- nutné překrýt metodu `run()`
  - kód v této metodě bude spuštěn simultálně s kódem ostatních podprocesů daného programu
- metoda `start()` volá incializaci podprocesu a spouští metodu `run()`
- metoda `destroy()` ukončí celé vlákno (násilně), bez uvolnění ostatních zdrojů
- metoda `interrupt()` přeruší dané vlákno
- metoda `boolean interrupted()` vrácí je-li vlákno přerušeno
- metoda `sleep(long millis)` způsobí "uspání" vlákna na určitý počet milisekund.Vyhazuje `InterruptedException`
- metoda `yield()` dočasně zastaví aktuální vlákno a umožní tak spuštění vláken ostatních

Příklad 1 - jednoduché vlákno (1)

```
public class SimpleThread extends Thread {

  private int odpocitavani = 5;
  private static int pocetPodprocesu = 0;
  private int cisloPodprocesu = ++pocetPodprocesu;

  public SimpleThread() {
    System.out.println("Vytvarim " + cisloPodprocesu);
  }

  public void run() {
    while(true) {
      System.out.println("Podproces " + cisloPodprocesu
        + "(" + odpocitavani + ")");
      if(--odpocitavani == 0) return;
    }
  }
    .
    .
    .
```

Příklad 1 - jednoduché vlákno (2)

```
    .
    .
    .
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++)
      new SimpleThread().start();
    System.out.println("Vsechny podprocesy spusteny");
  }
}
```

- procesy nemusí být spuštěny v pořadí v jakém jsou vytvořeny
  - toto pořadí nemůžeme ani zjistit
- všechny vlákna zde mají stejnou prioritu

Příklad 2 - čítač (1)

```
public class Counter extends JFrame {

  private VedlejsiUloha vedlejsi = null;
  private JTextField txtPole = new JTextField(10);
  private JButton start = new JButton("Start"),
                  prepinac = new JButton("Prepinac");

  public Counter() {
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      initComponents();
  }

  private void initComponents() {
      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());
      cp.add(txtPole);
      .
      .
```

Příklad 2 - čítač (2)

```
        start.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            if (vedlejsi == null)
                vedlejsi = new VedlejsiUloha();
            }
        });
        cp.add(start);

        prepinac.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            if (vedlejsi != null)
                vedlejsi.zmenitPriznak();
            }
        });
        cp.add(prepinac);
        pack();
    }
      .
      .
```

Příklad 2 - čítač (3)

```
public static void main(String[] args) {
  Counter cnt = new Counter();
  cnt.setVisible(true);
}

private class VedlejsiUloha extends Thread {
  private int citac = 0;
  private boolean priznakSpusteni = true;

  VedlejsiUloha() { start();}
  void zmenitPriznak() {priznakSpusteni =! priznakSpusteni;}

  public void run() {
    while (true) {
      try {
        sleep(100);
      } catch(InterruptedException e) {
        System.err.println("Preruseno");
      }
```

Příklad 2 - čítač (4)

```
            if (priznakSpusteni)
              txtPole.setText(Integer.toString(citac++));
            }
          }
      }
 }
```

Rozhraní `Runnable`

- kombinace podprocesu s hlavní třídou
- získáme spustitelný objekt (ve smyslu vlákna)
  - není spuštěn -> spustit explicitně

Příklad 3 - čítač s použitím `Runnable`

- přesuneme metodu `run()` do hlavní třídy
- upravíme posluchače ošetřující akce start a přepínač
  - inicializaci provedeme `Thread podproces = new Thread(this);`
  - spuštění opět `podproces.start();`

Podprocesy typu Daemon

- poskytuje služby na pozadí po celou dobu relace programu
- není podstatou programu
- ukončí se až po ukončení všech procesů
- voláním metody `boolean isDaemon()` zjistíme, zda je proces typu démon
- příznak nastavíme metodou `setDaemon()`

Příklad 4 - typu Daemon (1)

```
class Daemon extends Thread {
  private static final int VELIKOST = 10;
  private Thread[] podprocesy = new Thread[VELIKOST];
  public Daemon() {
    setDaemon(true);
    start();
  }
  public void run() {
    for(int i = 0; i < podprocesy.length; i++)
      podprocesy[i] = new VyplodDaemona(i);
    for(int i = 0; i < VELIKOST; i++)
      System.out.println("t[" + i + "].isDaemon() = "
        + podprocesy[i].isDaemon());
    while(true)
      yield();
  }
}
    .
    .
```

Příklad 4 - typu Daemon (2)

```
class VyplodDaemona extends Thread {
  public VyplodDaemona(int i) {
    System.out.println("VyplodDaemona " + i + " spusten");
    start();
  }
  public void run() {
    while(true)
      yield();
  }
}
```

Příklad 4 - typu Daemon (3)

```
public class Daemons {
 public static void main(String[] args) throws IOException {
    Thread podproces = new Daemon();
    System.out.println("podproces.isDaemon() = "
      + podproces.isDaemon());
    //umoznuje procesum typu daemon ukonceni jimi zapocatych
    //procesu
    System.out.println("Stisknete libovolnou klavesu");
    System.in.read();
  }
}
```

Příklad 5 - nesprávný přístup k prostředkům

```
public class Sharing1 extends JApplet {
  private static int accessCount = 0;
  private static JTextField aCount =
    new JTextField("0", 7);
  private JButton
    start = new JButton("Start"),
    watcher = new JButton("Watch");
  private boolean isApplet = true;
  private int numCounters = 12;
  private int numWatchers = 15;
  private TwoCounter[] s;

  public static void incrementAccess() {
    accessCount++;
    aCount.setText(Integer.toString(accessCount));
  }
    .
    .
```

Příklad 5 - nesprávný přístup k prostředkům

```
class TwoCounter extends Thread {
  private boolean started = false;
  private JTextField
    t1 = new JTextField(5),
    t2 = new JTextField(5);
  private JLabel l =
    new JLabel("count1 == count2");
  private int count1 = 0, count2 = 0;
  // přidá komponenty na panel a panel vloží
  public TwoCounter() {
    JPanel p = new JPanel();
    p.add(t1);
    p.add(t2);
    p.add(l);
    getContentPane().add(p);
  }
  .
  .
```

Příklad 5 - nesprávný přístup k prostředkům

```
public void start() {
  if(!started) {
    started = true;
    super.start();
  }
}
public void run() {
  while (true) {
    t1.setText(Integer.toString(count1++));
    t2.setText(Integer.toString(count2++));
    try {
      sleep(500);
    } catch(InterruptedException e) {
      System.err.println("Interrupted");
    }
  }
}
.
.
```

Příklad 5 - nesprávný přístup k prostředkům

```
    public void synchTest() {
      incrementAccess();
      if(count1 != count2)
        l.setText("Unsynched");
    }
  }

  class Watcher extends Thread {
    public Watcher() { start(); }
    public void run() {
      while(true) {
        for(int i = 0; i < s.length; i++)
          s[i].synchTest();
        try {
          sleep(500);
        } catch(InterruptedException e) {
          System.err.println("Interrupted");
        }
      }
    }
```

Příklad 5 - nesprávný přístup k prostředkům

```
class StartL implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    for(int i = 0; i < s.length; i++)
      s[i].start();
  }
}

class WatcherL implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    for(int i = 0; i < numWatchers; i++)
      new Watcher();
  }
}
  .
  .
```

Příklad 5 - nesprávný přístup k prostředkům

```
public void init() {
  if(isApplet) {
    String counters = getParameter("size");
    if(counters != null)
      numCounters = Integer.parseInt(counters);
    String watchers = getParameter("watchers");
    if(watchers != null)
      numWatchers = Integer.parseInt(watchers);
  }
  s = new TwoCounter[numCounters];
  Container cp = getContentPane();
  .
  .
```

Příklad 5 - nesprávný přístup k prostředkům

```
    cp.setLayout(new FlowLayout());
    for(int i = 0; i < s.length; i++)
      s[i] = new TwoCounter();
    JPanel p = new JPanel();
    start.addActionListener(new StartL());
    p.add(start);
    watcher.addActionListener(new WatcherL());
    p.add(watcher);
    p.add(new JLabel("Access Count"));
    p.add(aCount);
    cp.add(p);
  }
    .
    .
```

Příklad 5 - nesprávný přístup k prostředkům

```
  public static void main(String[] args) {
    Sharing1 applet = new Sharing1();
    // nespouštím jako aplet, argumenty z příkazového řádku
    applet.isApplet = false;
    applet.numCounters =
      (args.length == 0 ? 12 :
        Integer.parseInt(args[0]));
    applet.numWatchers =
      (args.length < 2 ? 15 :
        Integer.parseInt(args[1]));
    Console.run(applet, 350,
      applet.numCounters * 50);
  }
}
```

Sdílení prostředků

- souběžný přístup k prostředkům (paměť, soubory, ...)
- použití modifikátoru `synchronized`

Příklad

```
synchronized void f() {};
synchronized void g() {};
```

- každý objekt obsahuje jeden zámek (monitor)
  - automaticky se stává součástí **objektu**

Synchronizovat lze

- objekty
- metody
- bloky

Příklad 6 - oprava souběžného přístupu

Příklad 5 lze upravit např.

- přídáním modifikátoru `synchronized` před metody třídy `TwoCounter`
  - `synchronized run()` a `synchronized synchTest()`
- přídáním zámku na objekt `TwoCounter` v metodě `run()`

```
public void run() {
  while (true) {
    synchronized (this) {
      t1.setText(Integer.toString(count1++));
      t2.setText(Integer.toString(count2++));
    }
    ...
```

Efektivita synchronizovaného volání

- při uzamčení určitého objektu se zvýší náklady na režii
  - změnit návrh
  - použít zámek na konkrétní objekt (**NIKDY** neodbýt souběžný přístup synchronizací všech přístupových metod)

Blokování procesů

Proces může být v jednom ze čtyř stavů

- nový
  - vytvořen
  - neinicializován -> nemůže být spuštěn
- spuštěný
  - lze spustit (může, ale nemusí být spuštěný)
- neužívaný
  - možnost zastavit zavrhovanou (deprecated) metodou `stop()`
  - krajní případ `destroy()`
- blokovaný
  - může být spuštěn, ale je mu bráněno v činnosti
  - nevykonává žádný kód až do svého návratu do stavu spustitelný

Zablokování procesu

Podproces lze zablokovat pěti způsoby

1. vstoupil do režimu spánku (voláním metody `sleep()`)
2. pozastavení metodou `suspend()`, opětovné spuštění metodou `resume()` (obojí zavrhováno)
3. pozastavení metodou `wait()`, opětovné spuštění metodou `notify()` nebo `notifyAll()`
4. čekání na dokončení I/O operace
   - datový proud je automaticky zablokován, čeká-li na dokončení I/O operace
5. pokus o volání metody synchronizovaného objektu

Metody `wait()` *a* `notify()`

- odemyká uzamčené objekty
- metoda `wait(long millis)` - stejný význam jako `sleep(long millis)`
  - zámek objektu je **uvolněn**
  - vyhazuje vyjímku `IllegalMonitorStateException`
- `notify()` návrat ze stavu čekání

Priority

Priorita podprocesu - vlastnost sdělující míru důležitosti daného podprocesu.

- plánovací modul postupně spouští podprocesy s nejvýšší prioritou
- nížší priorita způsobí méně časté spouštění daného podprocesu

Čtení a nastavení priorit

- metoda `getPriority()` a `setPriority()`
- priorita podprocesu se může pohybovat v rozmezí 1 - 10
- implicitní priorita vytvořeného podprocesu 5

Skupiny podprocesů

- podproces vždy patří k nějaké skupině
  - implicitně k systémových podprocesům
  - systémový podproces bude vždy rodičovský
    - `Thread.currentThread().getThreadGroup()`
- podproces je svázán s určenou skupinou po celou dobu své existence
  - nelze změnit
- procesy sdružujeme ve skupině podprocesů třídy `ThreadGroup`
  - umožňuje nám to řídit naráz celé skupiny podprocesů

Třída `ThreadGroup`

- dva konstruktory `ThreadGroup(String name)` a `ThreadGroup(ThreadGroup parent, String name)`
- metoda `activeCount()` vrací počet podskupin ve skupině
- metoda `activeGroupCount()` vrací počet podprocesů ve skupině
- metoda `getMaxPriority()` maximální priorita v dané skupině
- metoda `getParent()` vrátí rodičovskou skupinu procesů
- metoda `boolean parentOf(ThreadGroup)` vrací zda je rodičem skupiny předané v argumentu
- shodné metody pro řízení vláken jako třída `Thread`, viz API

Vložení podprocesu do skupiny

- předáním skupiny podprocesů do jednoho z konstruktorů třídy tt Thread
  - např. Thread(ThreadGroup group, String name)