
# DISTRIBUOVANÉ PROGRAMOVÁNÍ

## Vlákna + JAVA

Ing. Igor Kopetschke – TUL, NTI

http://www.nti.tul.cz



TECHNICKÁ
UNIVERZITA
V LIBERCI

# Jak vytvořit vlákno

- Odvozením od třídy Thread
- Implementací rozhraní Runnable
- Runnable implementovat v případě, že není možno dědit od Thread
- V obou případech je nutno implementovat metodu **`public void run()`**

# Vlákno pomocí třídy Thread

```
class BasicThread extends Thread {
   // Tato metoda je volána po spuštění vlákna.
   public void run() {
      …… samotny kod vlakna ……
   }
}

// Vytvořte a spusťte vlákno.
Thread thread = new BasicThread();
thread.start();
```

# Vlákno pomocí třídy Thread jinak

```
Thread t = new Thread () {
   // Tato metoda je volána po spuštění vlákna.
   public void run() {
      …… samotny kod vlakna ……
   }
}

// Vytvořte a spusťte vlákno.
t.start();
```

# Vlákno pomocí rozhraní Runnable

```
class BasicThread implements Runnable {
  // Tato metoda je volána po spuštění vlákna.
  public void run() {
      …… samotny kod vlakna ……
  }
}

// Vytvořte objekt obsahující metodu run().
Runnable runnable = new BasicThread();

// Vytvořte vlákno, jemuž předáte objekt implementující
   rozhraní Runnable.
Thread thread = new Thread(runnable);
thread.start();
```

# Životní cyklus vlákna

- **Vytvořené, nespuštěné**
  - pouze instance, ještě nezavolán start()
- **Spuštěné běžící**
  - procesor vykonává kód
- **Spuštěné čekající**
  - ve frontě čeká na procesor
- **Spuštěné uspané nebo blokované**
  - Uspáno sleep() nebo blokováno wait(), join() aj.
- **Ukončené**
  - Metoda run() doběhla

# Jak správně zastavit vlákno

- Původně metodami **stop()** a **suspend()**
- Sloužily k asynchronnímu zastavení vlákna
- Tyto metody jsou zavrženy ! Docházelo k zablokování a nesprávnému uvolnění prostředků
- Princip správného zastavení vlákna – např. použití periodicky ověřované proměnné

# Vlákno a jeho zastavení

```
class NovyThread extends Thread {
  boolean allDone = false;
  public void run() {
    while (true) {
      // ... Sem vložte svůj kód...
      if (allDone) {
        return;
      }
      // ... Sem vložte svůj kód...
    }
  }
}

// Vytvořte a spusťte vlákno.
NovyThread thread = new NovyThread();
thread.start();

// ... Zde pokračuje program ...
// Zastavte vlákno.
thread.allDone = true;
```

# Zachycení okamžiku ukončení vlákna

```
Thread thread = new NovyThread();
thread.start();

if (thread.isAlive()) {
  // Vlákno je stále aktivní.
} else {
  // Vlákno bylo ukončeno.
}
```

# Zastavení všech vláken po Ctrl-C

```
// Vlákno lze registrovat do mechanismu předání
// události ukončení celé aplikace
Runtime().getRuntime.addShutdownHook( thread);
```

# Typy vláken

- Uživatelská vlákna – ty jsme si ukázali
- Systémová vlákna - démóni
  - Běží i po ukončení programu
  - Před spuštěním setDaemon(true)
  - Slouží jako služby, údržba apod.
  - Jak je ukončit?
    - Buď zajistíme, aby doběhly samy – jednorázový úklid
    - Běží periodicky v nekonečném cyklu – a to je už horší

# Jak zastavit démóna

- Restartovat PC

# Jak zastavit démóna

- Restartovat PC – to byl vtip ☺

# Jak zastavit démóna

- Restartovat PC – to byl vtip ☺
- "Zabít" JVM

# Jak zastavit démóna

- Restartovat PC – to byl vtip ☺
- "Zabít" JVM – nepraktické

# Jak zastavit démóna

- Restartovat PC – to byl vtip ☺
- "Zabít" JVM – nepraktické
- Použít shutdown hook (viz výše)
  - V démonovi implementovat logiku pro ukončení cyklu
  - Vytvořit Thread, které bude znát instanci démona
  - Předat instanci Threadu shutdown hooku

# Čekání na ukončení vlákna

Aktuální vlákno se zastaví a čeká, až doběhne vlákno, na kterém byla volána metoda **join()**

```
try {
  thread.join();
  // Vlákno bylo ukončeno.
} catch (InterruptedException e) {
  // Vlákno bylo přerušeno.
}
```

# Pozastavení aktuálního vlákna

```
long numMillisecondsToSleep = 5000;
// pozastaveni 5 sekund.
Thread.sleep(numMillisecondsToSleep);
```

# Určení okamžiku ukončení vlákna

```
long delayMillis = 5000; // 5 sekund.

try {
  thread.join(delayMillis);

  if (thread.isAlive()) {
    // Stanovený limit uplynul. Vlákno nebylo
  ukončeno.
  } else {
    // Vlákno bylo ukončeno.
  }
} catch (InterruptedException e) {
    // Vlákno bylo přerušeno.
}
```

# Jak správně pozastavit vlákno

- Původně metodami **suspend()** a **resume()**
- Tyto metody jsou zavrženy ! Docházelo k zablokování a nesprávnému uvolnění prostředků
- Správné je užití metod **wait()**,**notify()** a **notifyAll()**

# Korektní pozastavení vlákna - 1

```
// Třída, která umožňuje korektní pozastavení běhu:
class NovyThread extends Thread {
  boolean pleaseWait = false;

  public void run() {
    while (true) {
      // ... libovolny kod
      synchronized (this) {
        while (pleaseWait) {    // Má vlákno čekat?
          wait();
        }
      }
      // ... libovolny kod
    }
  }
}
```

# Korektní pozastavení vlákna - 2

```
NovyThread thread = new NovyThread();
thread.start();

// ... libovolny kod
synchronized (thread) {
   thread.pleaseWait = true; // Pozastavení vlákna.
}
// ... libovolny kod

synchronized (thread) {
   thread.pleaseWait = false;
   thread.notify();           // Obnovení běhu vlákna.
}
```

# „Nenažraná“ vlákna

- Vlákna mohou okupovat procesor, i když jsou v blokující operaci (čtení, zápis, čekání aj. )
- Proto je nutno vláknu vynutit vzdát se procesoru
  - **wait()** – uvolní zámek, uspí se
  - **notify(), notifyAll()** – probudí jiné vlákno/vlákna
  - **yield()** – dobrovolně se vzdá procesoru, je nově naplánováno (spolehlivé)
  - **Thread.sleep(ms)** – uspí aktuální vlákno
- Modifikátor **volatile**
  - Při práci se stejnou proměnnou se tato **necachuje** – zaručení aktuální hodnoty

# Synchronizace vláken

- V případě užití sdílených prostředků nebo kooperace vláken je nutno použít synchronizaci
- V Javě řešeno na úrovni objektu – **monitor**
  - **Zámek** – exkluzivita přístupu
  - **Wait condition** – wait(), notify(), notifyAll()
- Na úrovni proměnných **volatile**
  - Aktuální hodnota místo cachované
  - Atomicita čtení a zápisu
- metoda **join()** – vlákno čekání na ukončení vlákna, na kterém byla metoda volána

# Synchronizace vláken

- Klíčové slovo **synchronized**
- Série příkazů jako atomická operace s exkluzivním přístupem
- **V deklaraci metody**
  - monitorem je objekt, kterému metoda patří
  - na jednom objektu smí být aktuálně prováděna pouze jedna synchronizovaná metoda
- **Blok synchronized**
  - `synchronized (o) { … }`
  - monitorem je objekt **o**
  - větší variabilita, možnost použít externí objekt

# Pracovní fronta vláken

- Přizpůsobena pro souběžné užití ve více vláknech
- Možnost simultánně přidávat nebo odebírat objekty
- Zajištěna proti konfliktu s jinými podprocesy, které využívají stejné prostředky

# Pracovní fronta vláken - implementace

```
class WorkQueue {
  LinkedList queue = new LinkedList();

  public synchronized void addWork(Object o) {
    queue.addLast(o);
    notify();
  }

  public synchronized Object getWork() throws
   InterruptedException {
    while (queue.isEmpty()) {
      wait();
    }
    return queue.removeFirst();
  }
}
```

# Seskupování vláken

```
ThreadGroup group = new ThreadGroup("Skupina A");

for (int i = 0; i < 10; i++) {
  new NovyThread(group, "nazev").start();
}
```

# Vyhledání kořenové skupiny vláken

```
ThreadGroup root =
   Thread.currentThread().getThreadGroup().getParent();

while (root.getParent() != null) {
  root = root.getParent();
}
```

# Rekurzivní procházení vláken ve skupině

```
public static void visit(ThreadGroup group, int level) {
  // Dotaz na skupiny vláken ve skupině "group".
  int numThreads = group.activeCount();
  Thread[] threads = new Thread[numThreads*2];
  numThreads = group.enumerate(threads, false);

  // Výčet všech vláken ve skupině "group".
  for (int i=0; i<numThreads; i++) {
    // Get thread
    Thread thread = threads[i];
  }

  // Dotaz na všechny podskupiny vláken skupiny "group".
  int numGroups = group.activeGroupCount();
  ThreadGroup[] groups = new ThreadGroup[numGroups*2];
  numGroups = group.enumerate(groups, false);

  // Rekurzivní procházení všech podskupin.
  for (int i=0; i<numGroups; i++) {
    visit(groups[i], level+1);
  }
}
```

# Výpis všech spuštěných vláken

```
ThreadGroup root =
   Thread.currentThread().getThreadGroup().getParent();

while (root.getParent() != null) {
   root = root.getParent();
}

// prochazeni jednotlivych podskupin
// metoda visit() viz predchozi slide
visit(root, 0);
```

# Drží vlákno synchronizační zámek ?

```
public synchronized void checkLock( Object o ) {
  boolean hasLock = false;

  // 1. Stanovení, zda je aktuální vlákno
  // držitelem zámku pro objekt o.
  hasLock = Thread.holdsLock(o);      // spatne

  synchronized (o) {
    hasLock = Thread.holdsLock(o);    // spravne
  }

  // 2. Stanovení, zda aktuální vlákno drží zámek
  // pro aktuální objekt.
  hasLock = Thread.holdsLock(this);   // true
}
```

# Thread-safe třída pro užití ve vláknech

```
public class ThreadSafeClass {
  private String a;
  private Object syncObject = new Object();

  public ThreadSafeClass (String a) {
    this.a = a;
  }

  public void setA(String a) {
    synchronized (syncObject) {
      this.a = a;
    }
  }

  public String getA() {
    synchronized (syncObject) {
      return a;
    }
  }
}
```

# Jaký bude výsledek ?

```
public class IncrementalListClass {
    public volatile static int i = 0; // volatile - necachuj na procesoru

    public static void main(String[] args) throws InterruptedException {
        Thread t1 = new MyThread();
        Thread t2 = new MyThread();

        t1.start();
        t2.start();
        // na vterinu uspime hlavni vlakno a nechame dve pridana pracovat
        Thread.sleep(1000);
        System.out.println(i);
    }
}
```

# Jaký bude výsledek ?

```
public class IncrementalListClass {
    public volatile static int i = 0; // volatile == necachuj na procesoru

    public static void main(String[] args) throws InterruptedException {
        Thread t1 = new MyThread();
        Thread t2 = new MyThread();

        t1.start();
        t2.start();
        //na vterinu uspime hlavni vlakno a nechame dve pridana pracovat
        Thread.sleep(1000);
        System.out.println(i); // ocekavame vysledek 20000
    }
}

class MyThread extends Thread {

    @Override
    public void run() {
        for(int i = 0; i < 10000; i++) {
                IncrementalListClass.i++;
         }
    }
}
```

# Jaký bude výsledek ?

```
public class IncrementalListClass {
    public volatile static int i = 0; // volatile == necachuj na procesoru

    public static void main(String[] args) throws InterruptedException {
        Thread t1 = new MyThread();
        Thread t2 = new MyThread();

        t1.start();
        t2.start();
        //na vterinu uspime hlavni vlakno a nechame dve pridana pracovat
        Thread.sleep(1000);
        System.out.println(i); // ocekavame vysledek 20000
    }
}

class MyThread extends Thread {

    @Override
    public void run() {
        for(int i = 0; i < 10000; i++) {
                // NonAtomicExample.i = NonAtomicExample.i + 1
                // jsou to 2 dve operace: scitani a prirazeni
                IncrementalListClass.i++;
         }
    }
}
```

# Jaký bude výsledek ?

```
public class IncrementalListClass {
    public volatile static int i = 0; // volatile == necachuj na procesoru

    public static void main(String[] args) throws InterruptedException {
        Thread t1 = new MyThread();
        Thread t2 = new MyThread();

        t1.start();
        t2.start();
        //na vterinu uspime hlavni vlakno a nechame dve pridana pracovat
        Thread.sleep(1000);
        System.out.println(i); // ocekavame vysledek 20000
    }
}

class MyThread extends Thread {

    @Override
    public void run() {
        for(int i = 0; i < 10000; i++) {
                synchronized(IncrementalListClass.class){
                        // NonAtomicExample.i = NonAtomicExample.i + 1
                        // jsou to 2 dve operace: scitani a prirazeni
                        IncrementalListClass.i++;
                }
         }
    }
}
```

# .. A to je pro dnešek vše

DĚKUJI ZA POZORNOST