

JAVA

Vlákna

Přehled

- (thread)
- podpora pro vícevláknové aplikace přímo v jazyce
- "hlavní" vlákno aplikace – metoda main()
- v JVM je vždy několik vláken
 - záleží na implementaci
- JVM skončí až skončí všechna vlákna (která nejsou nastavena jako daemon)
- vlákna a skupiny vláken (thread groups)
- podpora pro synchronizaci
 - **synchronized**

Vytvoření vlákna

- implementace vlákna
 1. dědění od třídy `java.lang.Thread`
 2. implementování interface `java.lang.Runnable`
- potomek třídy **Thread**
 - predefinování metody `void run()`
 - vlákno se spustí zavoláním metody `start()`
- interface **Runnable**
 - jediná metoda `void run()`
 - objekt ji implementuje
 - vytvoření vlákna – `new Thread(Runnable).start()`

Příklad

```
public class SimpleThread extends Thread {
    public SimpleThread() {
        start();
    }
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(getName() + " : " + i);
    }
    public static void main(String[] args) {
        for (int i=0; i<5; i++) {
            new SimpleThread();
        }
    }
}
```

yield

- metoda třídy Thread
 - dočasné pozastavení vlákna, aby mohlo běžet jiné vlákno
 - je to jen doporučení
- static metoda
- úprava předchozího příkladu

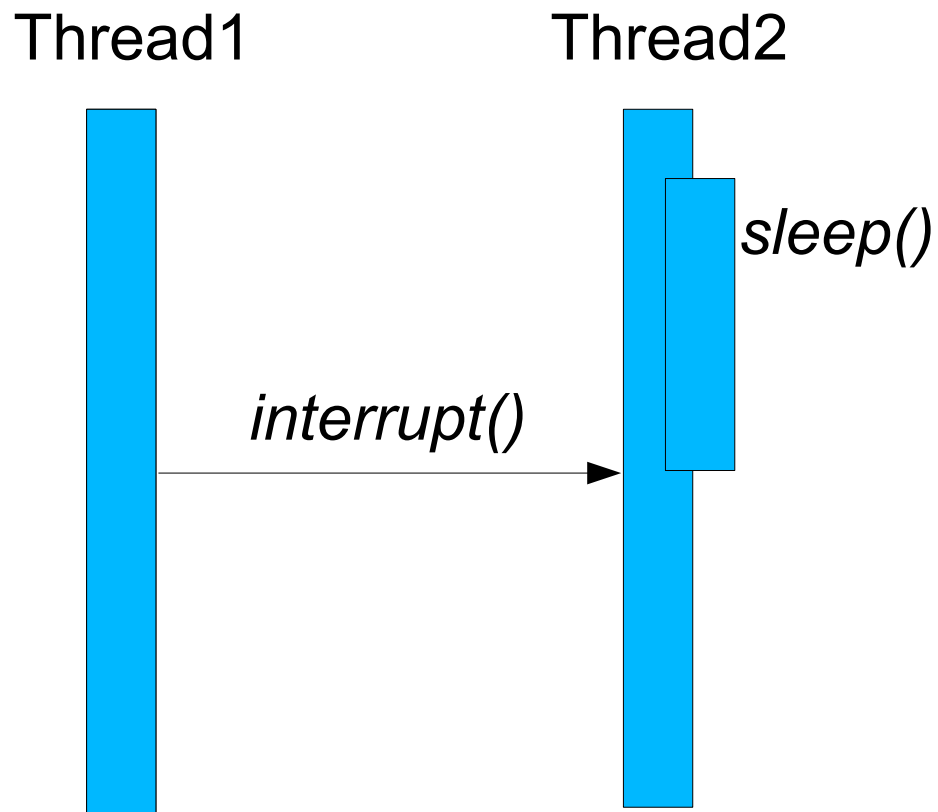
```
public void run() {  
    for (int i=0; i<5; i++) {  
        System.out.println(getName() + " : "+i);  
        yield();  
    }  
}
```

sleep

- dvě metody třídy Thread
 - `sleep(int millis)`
 - `sleep(int millis, int nanos)`
 - nanos v rozsahu 0-999999
- static metoda
- uspí vlákno na požadovanou dobu
- může být přerušeno (metodou `interrupt()`)
 - vyhodí výjimku `InterruptedException`

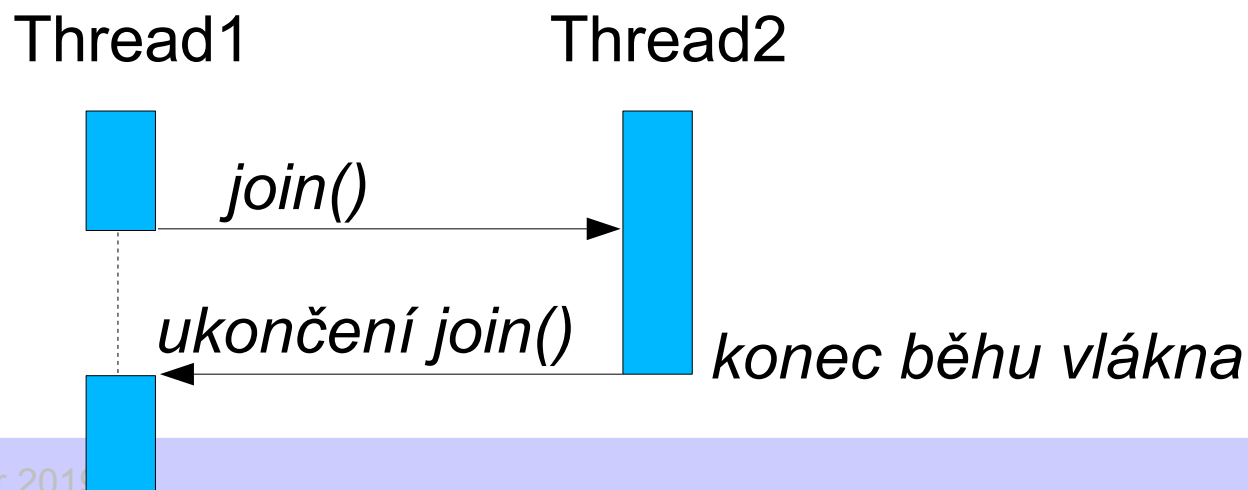
interrupt

- přeruší „čekání“ vlákna



join

- tři metody na třídě Thread
 - `join()`
 - čeká, dokud dané vlákno neskončí
 - `join(int milis)`
 - `join(int milis, int nanos)`
 - čekají dokud dané vlákno neskončí, maximálně však zadanou dobu (0...jako `join()` bez parametrů)
- opět mohou být přerušeny
 - výjimka `InterruptedException`



Priorita

- každé vlákno má prioritu
- `void setPriority(int newPriority)`
- `int getPriority()`
- 10 úrovní priority
- konstanty
 - `MAX_PRIORITY = 10`
 - `MIN_PRIORITY = 1`
 - `NORM_PRIORITY = 5`
- skupina vláken (`ThreadGroup`)
 - `getMaxPriority()`
 - `setPriority` vláknu nastaví maximálně prioritu povolenou pro skupinu vláken, do které vlákno náleží

Daemon vlákna

- "servisní" vlákna
- běží "na pozadí"
- nepatří přímo do aplikace
 - např. vlákno pro garbage collector
- JVM skončí pokud skončila všechna ne-daemon vlákna
- metody
 - void setDaemon(boolean daemon)
 - lze volat jen na ještě nenastartovaném vláknu
 - boolean isDaemon()

Synchronizace

- s každou instancí objektu je asociován jeden zámeček
- s každou třídou je asociován jeden zámeček
- příkaz/modifikátor **synchronized**
- příkaz
 - `synchronized (Výraz) Blok`
 - výraz se musí vyhodnotit na referenci na objekt
 - než se začne `Blok` vykonávat, vlákno musí získat pro sebe zámeček na objektu určeném pomocí `Výrazu`
 - po skončení `Bloku` se zámeček uvolní

Synchronizace

- modifikátor metody
 - synchronized v hlavičce metody
 - chová se stejně jako příkaz synchronized
 - vlákno než začne vykonávat metodu musí získat zámek na objektu
 - po skončení metodu se zámek uvolní
 - static synchronized metody – získává se zámek asociovaný se třídou
- vzájemně vyloučeny jsou jen synchronized metody a bloky
- nějaké vlákno má zámek na objektu – ostatní vlákna mohou používat atributy objektu i volat ne-synchronized metody

wait & notify

- s každým objektem je asociována fronta čekajících vláken
 - při vytvoření objektu je prázdná
- používají ji metody `wait`, `notify` a `notifyAll`
 - definovány na `java.lang.Object`
- `void wait()`
 - lze volat jen pokud volající vlákno drží zámek na daném objektu (tj. v `synchronized` sekci)
 - jinak výjimka `IllegalMonitorStateException`
 - vloží vlákno do fronty čekajících vláken
 - uvolní zámek na objektu
 - jiná vlákna mohou získat zámek, tj. začít provádět `synchronized` blok

wait & notify

- po `wait` je vlákno ve frontě čekajících vláken dokud někdo nezavolá `notify` nebo `notifyAll`
- `void notify()`
 - "probudí" jedno (nějaké) vlákno z fronty čekajících (pokud není prázdná)
 - lze volat jen ze `synchronized` sekcí
 - jinak výjimka `IllegalMonitorStateException`
 - probuzené vlákno pokračuje ve výpočtu až získá zámeček (tj. vlákno, co drželo zámeček (a zavolalo `notify`) opustí `synchronized` sekci)
- `void notifyAll()`
 - probudí všechna vlákna ve frontě
 - vlákna budou pokračovat, až získají zámeček

wait & notify

- tři varianty wait
 - void wait()
 - void wait(int millis)
 - void wait(int millis, int nanos)
 - čekají ve frontě, dokud někdo neprobudí, maximálně však zadanou dobu
- čekání ve wait lze přerušit (metoda interrupt())
 - vyhodí InterruptedException
- wait, notify i notifyAll jsou final
- metoda sleep() neuvolňuje zámek

Příklad – Jednoduchý zámek

```
public class SimpleLock {  
    private boolean locked;  
  
    public SimpleLock() {  
        locked = false;  
    }  
  
    synchronized public boolean lock() {  
        try {  
            while (locked)  
                wait();  
            locked = true;  
        } catch (InterruptedException e) {  
            return false;  
        }  
        return true;  
    }  
  
    synchronized public void unlock() {  
        locked = false;  
        notify();  
    }  
}
```


Zastavení vlákna

- `destroy()`
- `stop()`
- `stop(Throwable t)`
- `suspend()`
- `resume()`
 - všechny **deprecated** (většina od Java 1.2)
 - nebezpečné
 - mohly způsobovat nekonzistentní stav aplikace nebo způsobovat deadlock
- `destroy()` a `stop(Throwable)`
 - od Java 11 odstraněny

Skupiny vláken

- vlákna patří do skupiny vláken
- třída ThreadGroup
- skupina vláken může obsahovat jak vlákna tak další skupiny vláken
 - stromová hierarchie
- lze zjistit
 - všechny vlákna ve skupině
 - nadřazenou skupinu v hierarchii
 - aktivní vlákna ve skupině
- lze je ignorovat

Jméno vlákna

- vlákno má jméno
 - lze ho určit při vytváření vlákna
 - konstruktory
 - Thread(String name)
 - Thread(Runnable obj, String name)
 - za běhu
 - setName(String name)
 - zjištění jména
 - String getName()
- pokud se jméno nezadá, vytvoří se automaticky
 - "Thread-"+n
 - n je pořadové číslo

Ostatní metody

- `static Thread currentThread()`
 - vrátí referenci na právě prováděné vlákno
- `boolean isAlive()`
 - test zda je vlákno prováděno
 - `false` pokud vlákno není nastartováno, nebo pokud už skončilo
- `boolean isInterrupted()`
 - test, zda má vlákno nastavený příznak `interrupted`
- `boolean interrupted()`
 - jako `isInterrupted()`, ale zároveň zruší příznak `interrupted`
- `String toString()`
 - řetězcová reprezentace je složena z
 - jméno
 - priorita
 - skupina

java.util.concurrent

- java.util.concurrent
- java.util.concurrent.atomic
- java.util.concurrent.locks
 - balíky od JDK 5.0
 - obsahují třídy pro paralelní přístup k datům, zámky, semaforey,...

java.util.concurrent

- Executor
 - interface
 - různé implementace
 - ThreadPoolExecutor, ForkJoinPool,...
 - void execute(Runnable command)
 - spustí „příkaz“ v budoucnu
- ExecutorService
 - interface, potomek od Executor
 - přidává další metody
 - Future<T> submit(Callable<T> task)
 - List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
 - ...

java.util.concurrent

- **Callable<T>**
 - interface
 - T call() throws Exception
 - obdoba Runnable, ale vrací hodnotu a může vyhodit výjimku
- **Future<T>**
 - interface
 - výsledek asynchronní operace
 - T get()
 - vrátí výsledek
 - čeká pokud výsledek ještě není k dispozici

ForkJoinPool

- od Java 7
- implementuje ExecutorService
- pro „rozděl a panuj“
- podporuje „work-stealing“
- ForkJoinTask<V>
 - úloha pro ForkJoinPool, abstraktní třída
 - potomci
 - RecursiveAction
 - abstract void compute()
 - RecursiveTask<V>
 - abstract V compute()

ForkJoinPool

- metody pro spouštění úloh
 - execute()
 - asynchronní spuštění
 - submit(), submitAll()
 - asynchronní spuštění + vrací Future
 - invoke(), invokeAll()
 - spuštění a čekání až skončí
- obdobné metody jsou i na ForkJoinTask
 - spouštění „podúloh“
- získání poolu
 - konstruktory, nebo
 - `ForkJoinPool.commonPool()`

ForkJoinPool

```
class CustomRecursiveAction extends RecursiveAction {  
  
    @Override  
    protected void compute() {  
        if (...) {  
            ForkJoinTask.invokeAll(createSubtasks());  
        }  
    }  
  
    public static void main() {  
        CustomRecursiveAction cra =  
            new CustomRecursiveAction()  
        ForkJoinPool.commonPool().invoke(cra);  
    }  
}
```

Executors

- třída
- pouze statické „pomocné“ metody
 - konverze Runnable na Callable
 - získání různých thread-poolů
 - newFixedThreadPool()
 - newSingleThreadPool()
 - ...
 - ...

JAVA

`java.lang.System`

java.lang.System

- obsahuje jen statické elementy
- nelze od ní vytvářet instance
- atributy
 - `java.io.InputStream in`
 - standardní vstup
 - `java.io.PrintStream out`
 - standardní výstup
 - `java.io.PrintStream err`
 - standardní chybový výstup

Metody

- `void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
 - kopíruje pole
 - funguje i pokud `src==dest`
- `long currentTimeMillis()`
 - aktuální čas v milisekundách od 1.1.1970
 - přesnost záleží na OS
- `long nanoTime()`
 - hodnota systémového timeru v nanosekundách
 - počet nanosekund od *nějakého* pevného času
 - i v budoucnosti, tj. hodnota může být záporná
 - pro počítání nějaké doby
 - od Java 5

Metody

- `void exit(int status)`
 - ukončí JVM
- `void gc()`
 - „doporučení“ pro JVM, ať pustí garbage collector
- `void setIn(InputStream s)`
`void setOut(PrintStream s)`
`void setErr(PrintStream s)`
 - znovu nastaví daný vstup nebo výstup
- `int identityHashCode(Object x)`
 - vrátí implicitní hash kód objektu

Properties

- dvojice
 - klíč – hodnota
 - String (klíč i hodnota)
- systémové i vlastní
- `Properties` `getProperties()`
 - vrátí všechny nastavené properties
 - `java.util.Properties` – potomek `java.util.Hashtable`
- `String` `getProperty(String key)`
 - vrátí hodnotu
 - pokud klíč není nastaven, vrací `null`
- `String` `getProperty(String key, String def)`
 - vrátí hodnotu
 - pokud klíč není nastaven, vrací `def`

Properties

- `void setProperties(Properties props)`
 - nastaví properties v props
- `String setProperty(String key, String val)`
 - nastaví danou property
 - vrací původní hodnotu nebo null
- `String clearProperty(String key)`
 - zruší danou property
- nastavení properties při startu JVM
 - parametr `-Dkey=value`
 - př. `java -DdefaultDir=/usr Program`
- typicky se jako klíče používají hierarchická jména oddělená tečkami

Vždy nastavené properties

- `java.version`
- `java.home`
 - adresář, ve kterém je instalace JAVy
- `java.class.path`
- `java.io.tmpdir`
 - kde se budou vytvářet dočasné soubory
- `os.name`, `os.architecture`, `os.version`
 - identifikace operačního systému
- `file.separator`
 - oddělovač souborů v cestě (unix `"/"`, win `"\"`)
- `path.separator`
 - oddělovač cest (unix `":"`, win `";"`)
- `line.separator`
 - oddělovač řádků (unix `"LF"`, win `"CR LF"`)

Vždy nastavené properties

- `user.name`
 - název účtu aktuálního uživatele
- `user.home`
 - domovský adresář
- `user.dir`
 - aktuální adresář
- dále několik properties identifikujících VM

Proměnné prostředí

- **Map<String, String> getenv()**
 - všechny proměnné prostředí
 - nemodifikovatelná kolekce
- **String getenv(String name)**
 - proměnná s daným jménem nebo null

JAVA

`java.lang.Runtime`

Runtime

- vždy existuje jedna instance
 - nelze vytvářet další instance
- `Runtime.getRuntime()`
 - statická metoda
 - vrátí instanci `Runtime`
- `int availableProcessors()`
 - závisí na implementaci
 - vrácená hodnota se může za běhu programu měnit
- `long freeMemory()`
 - volná paměť dostupná pro JVM
- `long maxMemory()`
 - maximální dostupná paměť pro JVM
- `void halt(int status)`
 - ukončení JVM, na nic nečeká

Runtime

- `void addShutdownHook(Thread hook)`
 - nastaví vlákno, které se má provést při ukončování JVM
 - hook – vytvořené, ale nenastartované vlákno
 - může být nastaveno více vláken
 - začnou se provádět v "nějakém" pořadí
 - daemon vlákna běží i během ukončování JVM
 - nastavená vlákna se nevykonají, pokud JVM byla ukončena pomocí `halt()`
- `boolean removeShutdownHook(Thread hook)`
 - odstraní dříve nastavené vlákno
 - vrací `false`, pokud dané vlákno nebylo nastaveno

Runtime

- `Process exec(String command)`
 - spustí externí proces
 - několik variant funkce (různé parametry)
 - nemusí vždy spolehlivě fungovat
- třída `Process`
 - reprezentuje externí proces
 - metody
 - `void destroy()`
 - zabije proces
 - `int exitValue()`
 - návratová hodnota
 - `int waitFor()`
 - čeká, dokud proces neskončí
 - vrací návratovou hodnotu procesu
 - může být přerušeno

JAVA

`java.lang.Math`

java.lang.Math

- statické atributy a metody pro základní matematické konstanty a operace
- atributy
 - PI, E
- metody
 - abs, ceil, floor, round, min, max,...
 - pow, sqrt,...
 - sin, cos, tan, asin, acos, atan,...
 - toDegrees, toRadians,...
 - ...



Verze prezentace J08.cz.2019.01

Tato prezentace podléhá licenci [Creative Commons Uved'te autora-Neužívejte komerčně 4.0 Mezinárodní License](https://creativecommons.org/licenses/by-nc/4.0/).