Java threads

Paralelně proveditelné jednotky jsou objekty s metodou run, jejíž kód může být prováděn souběžně s jinými takovými metodami a s metodou main. Metoda run se spustí nepřímo vyvoláním start( )
Jak definovat třídy, jejichž objekty mohou mít paralelně prováděné metody

1. **jako podtřídy třídy Thread (je součástí java.lang balíku, potomkem Object)**
2. **implementací rozhrani Runnable**

ad1.

```
class MyThread extends Thread   //1.Z třídy Thread odvodíme potomka (s run metodou)
{public void run( )  { . . .}
  . . .
}
. . .
MyThread   t = new MyThread( ); //2.Vytvoření instance této třídy potomka
. . .
```

----

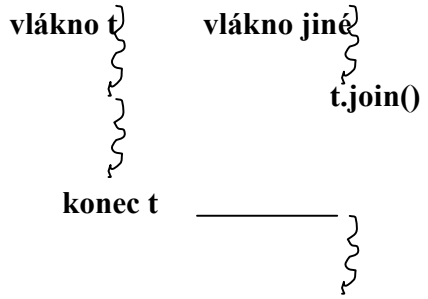
ad2.

```
class MyR implements Runnable     //1.konstruujeme třídu implementující Runnable
{public void run( )  { . . .}
  . . .
}
. . .
MyR  m = new MyR( );  // 2.konstrukce objektu teto třídy (s metodou run)
Thread t = new Thread(m); //3.vytvoření vlákna na tomto objektu
//je zde pouzit konstruktor Thread(Runnable threadOb)
. . .
```

Vlákno t se spustí až provedením příkazu **t.start( );**

Třída Thread má metody:

final String getName( ) //vrací jméno vlákna zadané Thread(Runnable jmR, String jméno)
final int getPriority( )
final void setPriority(int nováPriorita)
final boolean isAllive( )
final void join( )
void run( )
static void sleep(long milisekundy)
void start( )



. . .
Rozhraní Runnable má jen metodu run( )

```java
class MyThread implements Runnable { // pr. 3Vlakna
  int count;
  String thrdName;

  MyThread(String name) {
    count = 0;
    thrdName = name; //retezec slouzici jako jmeno vlakna
  }

    public void run() { // vstupni bod vlakna
    System.out.println(thrdName + " startuje.");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve vlaknu " + thrdName +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
      System.out.println(thrdName + " preruseny.");
    }
    System.out.println(thrdName + " ukonceny.");
  }
}

class Vlakno {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    // Nejdříve konstruujeme MyThread objekt. Má metodu run()
    MyThread mt = new MyThread("potomek");

    // Pak konstruuj vlakno z tohoto objektu, tím dodáme start(),…
    Thread newThrd = new Thread(mt);

    // Az pak startujeme vypocet vlakna
    newThrd.start();

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
        System.out.println("Hlavni vlakno prerusene.");
      }
    } while (mt.count != 5);

    System.out.println("Konci hlavni vlakno");
  }
}
```

```
class MyThread extends Thread { // pr. 3aVlakna          bez použití Runnable
  int count;

  MyThread(String name) {
    super(name); //předá nadtřídě Object jméno vlákna
    count = 0;
  }

    public void run() { // vstupni bod vlakna
    System.out.println(getName() + " startuje.");
    try {
      do {
        Thread.sleep(500); //Kvalifikace není nutna
        System.out.println("Ve vlaknu " + getName() +
                        ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
      System.out.println(getName() + " prerusene.");
    }
    System.out.println(getName() + " ukoncene.");
  }
}

class Vlakno {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    // Nejdříve konstruujeme MyThread objekt.
    MyThread mt = new MyThread("potomek");

    // Az pak startujeme vypocet vlakna
    mt.start();

    do {
      System.out.print(".");
      try {
        Thread.sleep(100); //Kvalifikace je nutna
      }
      catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
        System.out.println("Hlavni vlakno prerusene.");
      }
    } while (mt.count != 5);
    System.out.println("Konci hlavni vlakno");
  }
}
```

```
class MyThread implements Runnable { // př. 4Vlakna  Modifikace:
// vlákno se rozběhne v okamžiku jeho vytvoření
// jméno lze vláknu přiřadit v okamžiku spuštění
  int count;
  Thread thrd;

  // Konstruuje nove vlakno
  MyThread(String name) {
    thrd = new Thread(this, name);//vytvoří vlákno a přiřadí jméno
    count = 0;
    thrd.start(); // startuje vlakno rovnou v konstruktoru
  }

  // Začátek exekuce vlakna
  public void run() {
    System.out.println(thrd.getName() + " startuje ");
    try {
      do {
        Thread.sleep(500);
        System.out.println("V potomkovi " + thrd.getName() +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {
      System.out.println(thrd.getName() + " preruseny.");
    }
    System.out.println(thrd.getName() + " ukonceny.");
  }
}

class VlaknoLepsi {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt = new MyThread("potomek"); //v konstruktoru spustí

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vlakno prerusene.");
      }
    } while (mt.count != 5);

    System.out.println("Hlavni vlakno konci");
  }
}
```

```
class MyThread extends Thread {      // pr. 5Vlakna = totez jako 4Vlakna ale dedenim z Thread
  int count;

    MyThread(String name) {
    super(name); // jmeno vlakna
    count = 0;
    start(); // startuje v konstruktoru
  }

   public void run() {  //v potomkovi ze Thread musíme předefinovat run()
    System.out.println(getName() + " startuje");
    try {
      do {
        Thread.sleep(500);  //zde kvalifikace neni nutna, protože jsme v potomkovi ze Thread
        System.out.println("V " + getName() +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {
      System.out.println(getName() + " prerusene");
    }
    System.out.println(getName() + " ukoncene");
  }
}

class DediThread {
  public static void main(String args[]) {
    System.out.println("Hlavni vl.startuje");

    MyThread mt = new MyThread("potomek");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100); //zde je nutna kvalifikace
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vl. prerusene");
      }
    } while (mt.count != 5);

    System.out.println("Hlavni vl. konci");
  }
}
```

```
class MyThread implements Runnable {  // pr. 6Vlakna spusteni vice vlaken
  int count;
  Thread thrd;

  MyThread(String name) {
    thrd = new Thread(this, name);
    count = 0;
    thrd.start(); // start
  }

  public void run() {
    System.out.println(thrd.getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + thrd.getName() +
                           ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(thrd.getName() + " prerusene");
    }
    System.out.println(thrd.getName() + " ukoncene");
  }
}

class ViceVlaken {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vlakno prerusene");
      }
    } while (mt1.count < 3 ||
             mt2.count < 3 ||
             mt3.count < 3);

    System.out.println("Hlavni vl. konci");
  }
}
```

```
class MyThread extends Thread { // pr 6a Spusteni vice vlaken
  int count;

  MyThread(String name) {
    super(name);
    count = 0;
    start(); // start
  }

  public void run() {
    System.out.println(getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + getName() +
                           ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(getName() + " preruseny");
    }
    System.out.println(getName() + " ukonceny");
  }
}

class ViceVlaken {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vlakno prerusene");
      }
    } while (mt1.count < 3 ||
             mt2.count < 3 ||
             mt3.count < 3);

    System.out.println("Hlavni vl. konci");
  }
}
```

Identifikace ukončení činnosti vláken

-nejčastěji k zastavení dojde doběhnutím metody run( )

-stav lze testovat metodou isAllive( )
tvaru: final boolean isAllive( )
? jak využít v modifikaci předchozích programů? Viz poznámka

-čekáním na skončení jiného vlákna vyvoláním metody join()
tvaru: final void join( ) throws InterruptedException
Např.
Thread t = new Thread(m);
t.start( ); // zahaji cinnost
//rodic neco dela
t.join( ); //rodic ceka na skonceni t
// rodic pokracuje po skonceni t

-existuje alternativa čekání na skončení vlákna, informující, že se čeká na jeho konec
Thread t = new Thread(m);
t.start( ); // zahaji cinnost
//rodic neco dela
t.interrupt( ) ; //oznamuje t, ze na nej cekame, predcasne ho probudi
t.join( ); //rodic ceka na skonceni t
// rodic pokracuje po skonceni t

-existuje alternative pro timeout t.join(milisekundy)

Poznámka:

V main

```
...
} while (mt1.thrd.isAlive() ||
         mt2.thrd.isAlive() ||
         mt3.thrd.isAlive());

    System.out.println("Main thread ending.");
  }
}
```

```
//Pr 7aVlakna   pouziti join
class MyThread extends Thread {
  int count;
  MyThread(String name) {
    super(name);
    count = 0;
    start(); // start
  }
   public void run() {
    System.out.println(getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + getName() +
                        ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(getName() + " preruseny");
    }
    System.out.println(getName() + " konci");
  }
}

class Join {
    public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    try {
      mt3.join();
      System.out.println("potomek3 joined.");
      mt2.join();
      System.out.println("potomek2 joined.");
      mt1.join();
      System.out.println("potomek1 joined.");
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno prerusene");
    }
     System.out.println("Hlavni vl. konci");
  }
}
```

Pr. 7Vlakna

```
class MyThread implements Runnable {  //MyThread má tvar jako v př. 6
  int count;
  Thread thrd;
  MyThread(String name) {
    thrd = new Thread(this, name);
    count = 0;
    thrd.start(); // start
  }
   public void run() {
    System.out.println(thrd.getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + thrd.getName() +  ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(thrd.getName() + " preruseny");
    }
    System.out.println(thrd.getName() + " konci");
  }
}
class Join {
    public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    try {
      mt1.thrd.join();
      System.out.println("potomek1 joined.");
      mt2.thrd.join();
      System.out.println("potomek2 joined.");
      mt3.thrd.join();
      System.out.println("potomek3 joined.");
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno preruseno");
    }
     System.out.println("Hlavni vl. konci");
  }
}
```

Priorita vláken

- **Vysoká priorita = hodně času procesoru**
- **Nízká priorita = méně času procesoru**
- **Implicitně je přidělena priorita potomkovi jako má nadřízený process**
- **Změnit lze prioritu metodou setPriority**

final void setPriority(int cislo)

Min_Priority ≤ cislo ≥ Max_Priority
1 . . 10

Norm_Priority = 5

- **Zjištění aktuální priority provedeme metodou**
  final int getPriority( )

```
class Priority extends Thread { //Pr.8aVlakna  - projevi se v OS s Time-Slicing
  int count;
  static boolean stop = false;
  static String currentName;
  Priority(String name) {
    super(name);
    count = 0;
    currentName = name;
  }
 public void run() {
    System.out.println(getName() + " start ");
    do {
      count++;
      if(currentName.compareTo(getName()) != 0) {
        currentName = getName();
        System.out.println("Ve " + currentName);
      }
    } while(stop == false && count < 50); //dvě možnosti ukončení běhu vlákna
    stop = true; //pokud skončilo jedno, tak pak skončí i druhé
    System.out.println("\n" + getName() + " konci");
  }
}

class Priorita {
  public static void main(String args[]) {
    Priority mt1 = new Priority("Vysoka Priorita"); //JVM to může, ale nemusí respektovat
    Priority mt2 = new Priority("Nizka Priorita");

    // nastaveni priorit
    mt1.setPriority(Thread.NORM_PRIORITY+2);
    mt2.setPriority(Thread.NORM_PRIORITY-2);
    // start vlaken
    mt1.start();
    mt2.start();
    try {
      mt1.join(); //main čeká na ukončení
      mt2.join();
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno konci");
    }
    System.out.println("Vlakno s velkou prioritou nacitalo " +
                    mt1.count);
    System.out.println("Vlakno s malou prioritou nacitalo " +
                    mt2.count);
  }
}
```

```
class Priority implements Runnable {  //Pr. 8Vlakna
  int count;  // Každý objekt ze třídy Priority má čítač a vlákno
  Thread thrd;
  static boolean stop = false;
  static String currentName;
  Priority(String name) {
    thrd = new Thread(this, name);
    count = 0;
    currentName = name;
  }
  public void run() {
    System.out.println(thrd.getName() + " start ");
    do {
      count++;

      if(currentName.compareTo(thrd.getName()) != 0) {
        currentName = thrd.getName();
        System.out.println("V " + currentName);
      }
    } while(stop == false && count < 500);
    stop = true;
    System.out.println("\n" + thrd.getName() + " terminating.");
  }
}

class Priorita {
  public static void main(String args[]) {
    Priority mt1 = new Priority("Vysoka Priorita");
    Priority mt2 = new Priority("Nizka Priorita");
    // nastaveni priorit
    mt1.thrd.setPriority(Thread.NORM_PRIORITY+2);
    mt2.thrd.setPriority(Thread.NORM_PRIORITY-2);
    // start vlaken
    mt1.thrd.start();
    mt2.thrd.start();

    try {
      mt1.thrd.join();
      mt2.thrd.join();
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno preruseno");
    }

    System.out.println("Vlakno s velkou prioritou nacitalo " +
                       mt1.count);
    System.out.println("Vlakno s malou prioritou nacitalo" +
                       mt2.count);
  }
}
```

Př 81Vlakna Ilustruje rychlostní závislosti výpočtu Vypouštěním sleep simuluji různé rychlosti

```
class Konto { static int konto = 1000;}
class Koupe extends Thread {
   Koupe(String jmeno) {
    super(jmeno);
   }
   public void run()  { // vstupni bod vlakna
          System.out.println(getName() + " start.");
          int lokal;
          try {
          lokal = Konto.konto;
          System.out.println(getName() + " milenkam ");
          sleep(100);//////////////////////////////////////
          Konto.konto = lokal - 200;
          System.out.println(getName() + " ukoncene.");
          }
          catch (InterruptedException e) {}
} }
class Prodej extends Thread {
   Prodej(String jmeno) {
    super(jmeno);
   }
   public void run() { // vstupni bod vlakna
          System.out.println(getName() + " start.");
          int lokal;
          try {
          lokal = Konto.konto;
          System.out.println(getName() + " co se da ");
          sleep(200);/////////////////////////////////////
          Konto.konto = lokal + 500;
          System.out.println(getName() + " ukoncene.");
          }
          catch (InterruptedException e) {}
} }
class RZ {
      public static void main (String args[])
          throws InterruptedException {
          System.out.println("Hlavni vlakno startuje");
          Koupe nakup = new Koupe("kupuji");
          Prodej prodej = new Prodej ("prodavam");
          nakup.start();
          prodej.start();
          Thread.sleep(500); // zajistí, že nákup i prodej skončil
          System.out.println(Konto.konto);
          System.out.println("Konci hlavni vlakno");
}    }
```

Kritické sekce

(řešení problému sdílení zdrojů formou vzájemného vyloučení současného přístupu)

- **metoda s označením synchronized uzamkne objekt pro který je volána**
- **jiná vlákna pokoušející se použít synchr. metodu uzamčeného objektu musí čekat ve frontě**
- **když proces opustí synchr. metodu, objekt se odemkne**
- **objekt může mít současně synch.i nesynch. metody, a ty nevyžadují zámek = vada**

(Každý objekt Javy je vybaven zámkem, který musí vlákno vlastnit, chce-li provést synchronized metodu na objektu.)

```
např.  class Queue {
            . . .
            public synchronized int vyber( ){ …}
            . . .
            public synchronized  void uloz(int co){ …}
            . . .
       }
```

- **synchronizovaný příkaz tvaru**
  synchronized (výraz s hodnotou objekt) příkaz

  zamkne přístup k objektu nad kterým pracuje. Objekt musí systém vybavit frontou pro metody, které chtějí s objektem pracovat.

Komunikace mezi vlákny

(řeší situaci, kdy metoda vlákna potřebuje přístup k dočasně nepřístupnému zdroji

- **může čekat v nějakém cyklu (neefektivní využití objektu nad nímž pracuje)**
- **může se zřeknout kontroly nad objektem a jiným vláknům umožnit ho používat, musí jim to ale dát na vědomí**

Kooperace procesů zajišťují metody:

- **wait() vlákno přejde do stavu blokovaný a uvolní zámek objektu**
  verze:
  final void wait( ) throws InterruptedException
  final void wait(long milisec) throws InterruptedException
  final void wait(long milisec, int nonosec) throws InterruptedException
- **final void notify( ) oživí vlákno z čela fronty na objekt**
- **final void notifyAll( ) oživí všechna vlákna nárokující si přístup k objektu**

mohou být volány jen ze synchronized metod, jsou děděny z pratřídy Object

Př. Semafor jako ADT v Javě

```
class Semafor {
        private int count;

        public Semafor(int initialCount) {
                count = initialCount;
        }

        public synchronized void cekej( ) {
                try {
                while (count <= 0 ) wait( );  //musí  být nedělitelné nad instancí semaforu
                count--;
                }
                catch (InterruptedException e) { }
        }

        public synchronized void uvolni( ) {
                count++;
                notify( );
        }
}
```

Př. 82Vlakna Odstranění rychlostních závislostí z př.81. Výsledek na času sleep nezávisí

```
class Konto { // instance z třídy Konto udelame monitorem
   private int konto;
   public Konto(int i){ konto =i;}
   public int stav() {return konto;}  //není sychronized
   public synchronized void vyber(int kolik) {
         int lokal; //pro zachovani podminek jako u RZ
         try { lokal = konto;
         Thread.sleep(100);//////////////////////////////////////
         konto = lokal - kolik; notify();
         }        catch (InterruptedException e) {}
   }
   public synchronized void vloz(int kolik) {
         int lokal;
         try { lokal = konto;
         Thread.sleep(300);//////////////////////////////////////////
         konto = lokal + kolik; notify();
         }        catch (InterruptedException e) {}
} }
class Koupe extends Thread {
   private Konto k;
   Koupe(Konto x, String jmeno) { super(jmeno); k = x; }
   public void run() { // vstupni bod vlakna
         System.out.println(getName() + " start.");
         k.vyber(200);
         System.out.println(getName() + " ukoncene.");
} }
class Prodej extends Thread {
  private Konto k;
  Prodej(Konto x, String jmeno) { super(jmeno); k = x;  }
  public void run() { // vstupni bod vlakna
         System.out.println(getName() + " start.");
         k.vloz(500);
         System.out.println(getName() + " ukoncene.");
} }
class Konta {
     public static void main (String args[]) throws InterruptedException {
         System.out.println("Hlavni vlakno startuje");
         Konto meKonto = new Konto(1000);//vytvářím konto, mohu jich udělat i víc
         Koupe nakup = new Koupe(meKonto," nakupuji ");
         Prodej prodej = new Prodej (meKonto, " prodavam ");
         nakup.start();
         prodej.start();
         Thread.sleep(500);
         System.out.println(meKonto.stav());
         System.out.println("Konci hlavni vlakno");
} }
```

Každé vlákno je instancí třídy java.lang.Thread nebo jejího potomka

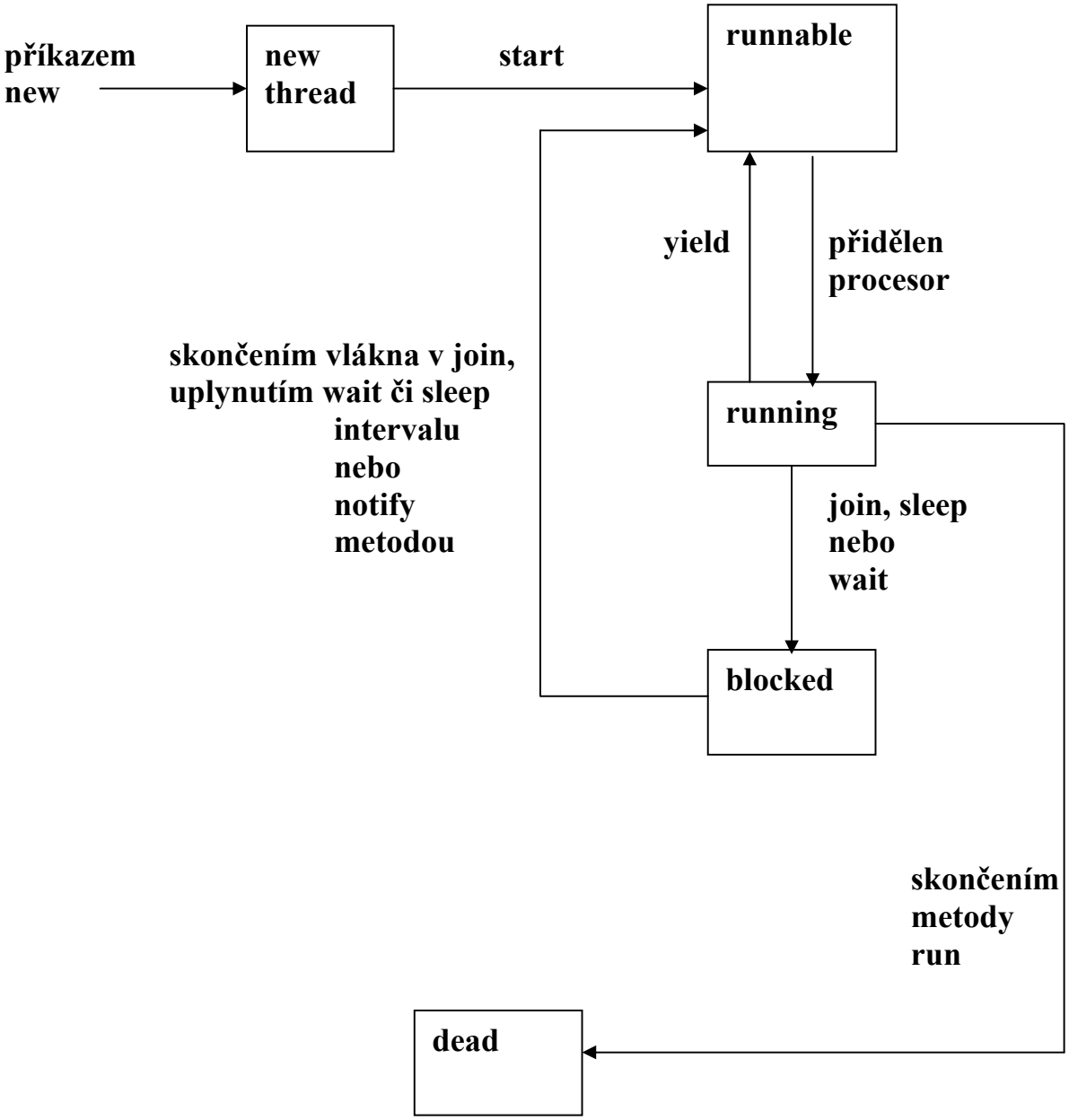
Thread má metody:

- **run() je vždy přepsána v potomku Thread, udává činnost vlákna**
- **start() spustí vlákno (tj. metodu run) a volající start pak pokračuje ve výpočtu. Metoda run není přímo spustitelná**
- **yield() odevzdání zbytku přiděleného času a zařazení do fronty na procesor**
- **sleep(milisec) zablokování vlákna na daný čas. Interval**
- **isAllive( ) běží-li, vrací true, jinak false**
- **join( )**
- **getPriority**
- **setPriority**
- **final void notify() oživí vlákno z čela fronty na objekt**
- **final void notifyAll() oživí všechna vlákna nárokující si přístup k objektu**
-
- **. . . a další cca 20**

stavy vláken:

- **nové**
- **připravené**
- **běžící**
- **blokované**
- **mrtvé**

Plánovač vybere z fronty připravených vlákno s nejvyšší prioritou



Obr. Přechody mezi stavy vlákna Javy

```
import java.io.*;     // pr. 9Vlakna. Producent ukládá do bafru, konzument z něj vybírá
class Queue { private int [] que;     // Bafr má podobu kruhové fronty realizované polem
        private int nextIn, nextOut, filled, queSize;
        public Queue(int size) {
                que = new int [size];
                filled = 0;       //zaplněnost bafru
                nextIn = 1;     //kam vkládat
                nextOut = 1;  //odkud vybírat
                queSize = size;
        } // konec konstruktoru

        public synchronized void deposit (int item) {
                try {
                        while (filled == queSize)
                                wait();
                        que [nextIn] = item;
                        nextIn = (nextIn % queSize) + 1;
                        filled++;
                        notify();  //budí vlákno konzumenta a uvolňuje monitor
                } //konec try
                catch (InterruptedException e) {
                        System.out.println("int.depos");
                }
        } //konec deposit
        public synchronized int fetch() {
                int item = 0;
                try {
                        while (filled == 0)
                                wait();
                        item = que [nextOut];
                        nextOut = (nextOut % queSize) + 1;
                        filled--;
                        notify(); //budí vlákno producenta a uvolňuje monitor
                } //konec try
                catch(InterruptedException e) {
                        System.out.println("int.fetch");
                }
                return item;
        } //konec fetch
} //konec tridy Queue
```

```
class Producer extends Thread {    //producent čte z klávesnice a ukládá do bufru
        private Queue buffer;
        public Producer(Queue que) {
                buffer = que;
        }
        public void run() {
           int new_item = 0; // neprelozi bez inicializace
          opakuj: while (new_item > -1) { /*ukoncime-1 nebo
                                          zapornym cislem*/
                 try {//produkce
                byte[] vstupniBuffer = new byte[20];
                System.in.read(vstupniBuffer);
                String s = new String(vstupniBuffer).trim();
                new_item = Integer.valueOf(s).intValue();
                 }
                 catch (NumberFormatException e) {  // když číslo není správně zapsané
                        System.out.println("nebylo to dobre");
                        continue opakuj;
                 }
                catch (IOException e) {//zachytava nepripr.klavesnici
                        System.out.println("chyba cteni");
                 }
                 buffer.deposit(new_item);
                }
        }
}
class Consumer extends Thread { //konzument vybírá údaj z bufru a tiskne ho
        private Queue buffer;
        public Consumer(Queue que) {
                buffer = que;
        }
        public void run() {
                int stored_item = 0;  //chce inicializaci
                while (stored_item > -1) {/*ukoncime -1 nebo
                                          minus cislem*/
                 stored_item = buffer.fetch();
                 System.out.println(stored_item);       //konzumace
                }
        }
}
```

```
public class P_C {
	public static void main(String [] args) {
		Queue buff1 = new Queue(100);
		Producer producer1 = new Producer(buff1);
		Consumer consumer1 = new Consumer(buff1);
		producer1.start();
		consumer1.start();
	}
}
```

Paralelismus na úrovni příkazů jazyka

Occam

parbegin S1; S2; . . . Sn; parend;

High performance Fortran

Založen na modelu SIMD:

- **výpočet je popsán jednovláknovým programem**
- **proměnné (obvykle pole) lze distribuovat mezi více procesorů**
- **distribuce, přístup k proměnným a synchronizace procesorů je zabezpečena kompilátorem**

```
REAL DIMENSION (1000, 1000) :: A, B
INTEGER  I, J
…
DO I = 2, N
   DO J = 1, I – 1
         A(I, J) = A(I, J) / A(I, I)
   END DO
END DO
```

```
FORALL (I = 2 : N, J = 1 : N, J .LT. I )  A(I, J) = A(I, J) / A(I, I)
```

FORALL představuje zobecněný přiřazovací příkaz (a ne smyčku)
FORALL lze použít, pokud je zaručeno, že výsledek seriového i paralelního zpracování budou identické.

Paralelismus na úrovni programů

Pouze celý program může být paralelní aktivitou.
Fork příkazem Unixu vznikne potomek – přesná kopie volajícího procesu

```
#define SIZE 100
#define NUMPROCS 10
int a[SIZE] [SIZE], b[SIZE] [SIZE], c[SIZE] [SIZE];

void multiply(int myid)
{ int i, j, k;
  for (i = myid; i < SIZE; i+= NUMPROCS)
    for (j = 0; j < SIZE; ++j)
    { c[i][j] = 0;
      for (k = 0; k < SIZE; ++k)
        c[i][j] += a[i][k] * b[k][j];
    }
}

main()
{ int myid;
  /* prikazy vstupu a, b */
  for (myid = 0; myid < NUMPROCS; ++myid)
    if (fork() == 0)
    { multiply(myid);
      exit(0);}
  for (myid = 0; myid < NUMPROCS; ++myid)
    wait(0);
  /* prikazy vystupu c */
  return 0;
}
```