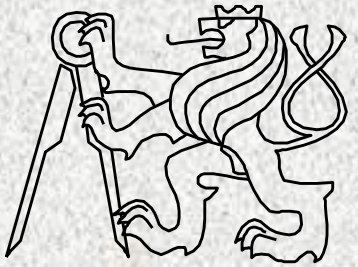
VLÁKNA



BD6B36PJV 8

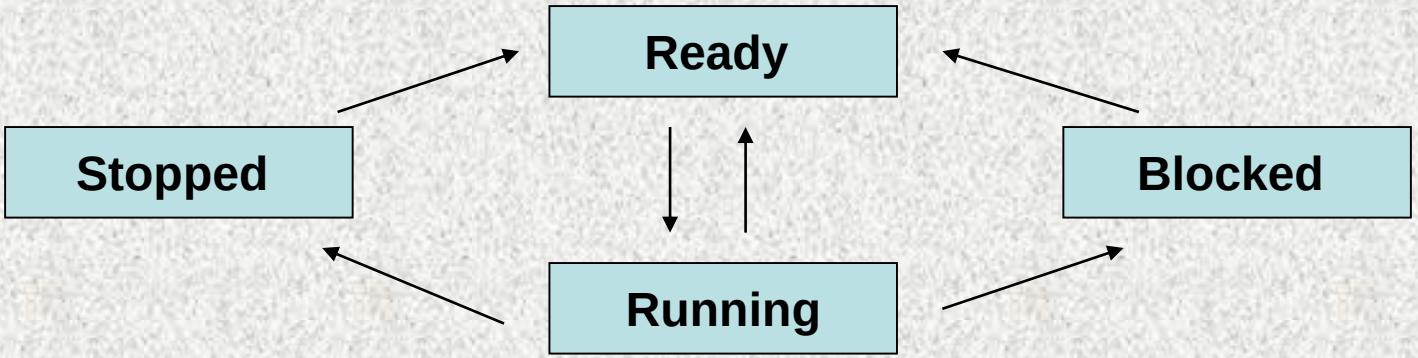
Fakulta elektrotechnická
České vysoké učení technické

Vlákna v Javě

- Oblasti použití
  - Nesoulad časových nároků různých nezávislých částí programu
  - Dlouhé periferní přenosy, čekání na odezvu na vstupu
  - Simulační výpočty, opakující se výpočty
  - Úlohy typu producent/konzument, server/klient

- Příklad, textový editor – psaní a pravopis
- Paralelní zpracování úloh
  1. Paralelní procesy, *řídí operační systém*
     - Různé oblasti paměti, přepínání konktextu
  2. Paralelní **vlákna** procesu, *řídí run-time (JRE) Javy*
     - Společná paměť procesu, sdílení paměti
       - asynchronní zpracování, možnost synchronizace synchronizační metodou
     - Priorita vláken

Stavy procesu



- Stavy procesu
  - **Running = proces se pravě provádí**
  - **Blocked = proces čeká na externí událost nebo na prostředek.**
  - **Ready = (pozastavené) proces je připraven a čeká na přidělení CPU.**
  - **Stopped = proces ukončen, není možné pokračovat**

Dvě možnosti implementace vláken

1. Třída, která je potomek třídy **`extend Thread`**
   - Tam kde potřebujeme více metod k práci s vlákny (isAlive(), join(), sleep(),..)
2. Třída implementující **`interface Runnable`**
   - Potřebujeme jen, aby objekty třídy pracovaly „ve vláknu“, měly schopnost „být vláknem“

- Každý program v Javě má implicitní vlákno
  - Z něj vznikají další paralelní (dceřiná) vlákna
  - Měly bychom zajistit, aby hlavní vlákno končilo poslední (uvolnění zdrojů)
  - Odkaz na hlavní vlákno **`Thread currentThread()`** v hlavním programu a tak můžeme vlákno řídit

Hlavní vlákno

```
public class DemoVlakno {// změní jméno vlákna
     public static void main(String[] args) {
        Thread t = Thread.currentThread();
        System.out.println("Akualni vlakno"+ t);
        t.setName("nove jmeno");
        System.out.println("Aktualni vlakno" + t);
    }
```

Output:

Akualni vlaknoThread[main,5,main]

Aktualni vlaknoThread[nove jmeno,5,main]

5 – priorita vlákna <1,2a>

Vlákna v Javě, implementace rozhraní `Runnable`

- Vlákno zkonstruujeme na jakémkoli objektu třídy, která implementuje rozhraní `Runnable`
- vytvoříme instanci třídy `Thread` a předáme objekt (referenci na instanci třídy) třídy, který bude probíhat jako samostatné vlákno
  - `new Thread(Runable <třída>, String <jméno vlákna>);`
- třída musí implementovat jednu metodu rozhraní `Runnable` : `run()`
  - metoda `run()` určuje činnost vlákna
  - nespouští se sama!
- Vhodné resp. nutné definovat metodu `start()` - není to „povinné“, ale odstraní to nejednoznačnost
- vlákno spustíme metodou `start()`, ta odstartuje metodu `run()`

Vytvoření vlákna **implements**

```
class MojeVlakno0 implements Runnable{
    Thread v;
    MojeVlakno0() {
        v = new Thread(this, "Moje vlakno");
        v.start();
    }
    public void run() {
        System.out.println("Vlákno spuštěno ");
        System.out.println("Vlákno ukončeno ");
    }
}
public class DemoVlakno0 {

    public static void main(String[] args) {
        new MojeVlakno0();
        System.out.println("Vlákno hlavní spuštěno ");
        System.out.println("Vlákno hlavní ukončeno ");
    }
}
```

Vytvoření vlákna **implements**

```
class MojeVlakno01 implements Runnable {
    Thread v;
    MojeVlakno01() {
        v = new Thread(this, "Moje vlakno");
    }
    public void run() {
        System.out.println("Vlákno spuštěno ");
        System.out.println("Vlákno ukončeno ");
    }
}
public class DemoVlakno01 {
    public static void main(String[] args) {
        MojeVlakno01 n= new MojeVlakno01();
        n.v.start();
        System.out.println("Vlákno hlavní spuštěno ");
        System.out.println("Vlákno hlavní ukončeno ");
    }
}
```

Vlákna v Javě, potomek třídy `Thread`

- vlákno je instancí třídy `Thread`
- v konstruktoru třídy (jejíž instance budou vlákny) voláme konstruktor nadřazené třídy `Thread` a předáváme jméno vlákna
- metodu třídy `Thread` `run()` překryjeme pomocí `super()`.
- metoda `run()` určuje činnost vlákna
  - nespouští se sama
- metoda `start()` spouští metodu `run()`

Vytvoření vlákna **extends**

```
class MojeVlakno0T extends Thread {
    MojeVlakno0T() {
        super("Moje vlakno");
        start();
    }

    public void run() {
        System.out.println("Vlákno spuštěno ");
        System.out.println("Vlákno ukončeno ");
    }
}
public class DemoVlakno0T {

    public static void main(String[] args) {
        new MojeVlakno0T();
        System.out.println("Vlákno hlavní spuštěno ");
        System.out.println("Vlákno hlavní ukončeno ");
    }
}
```

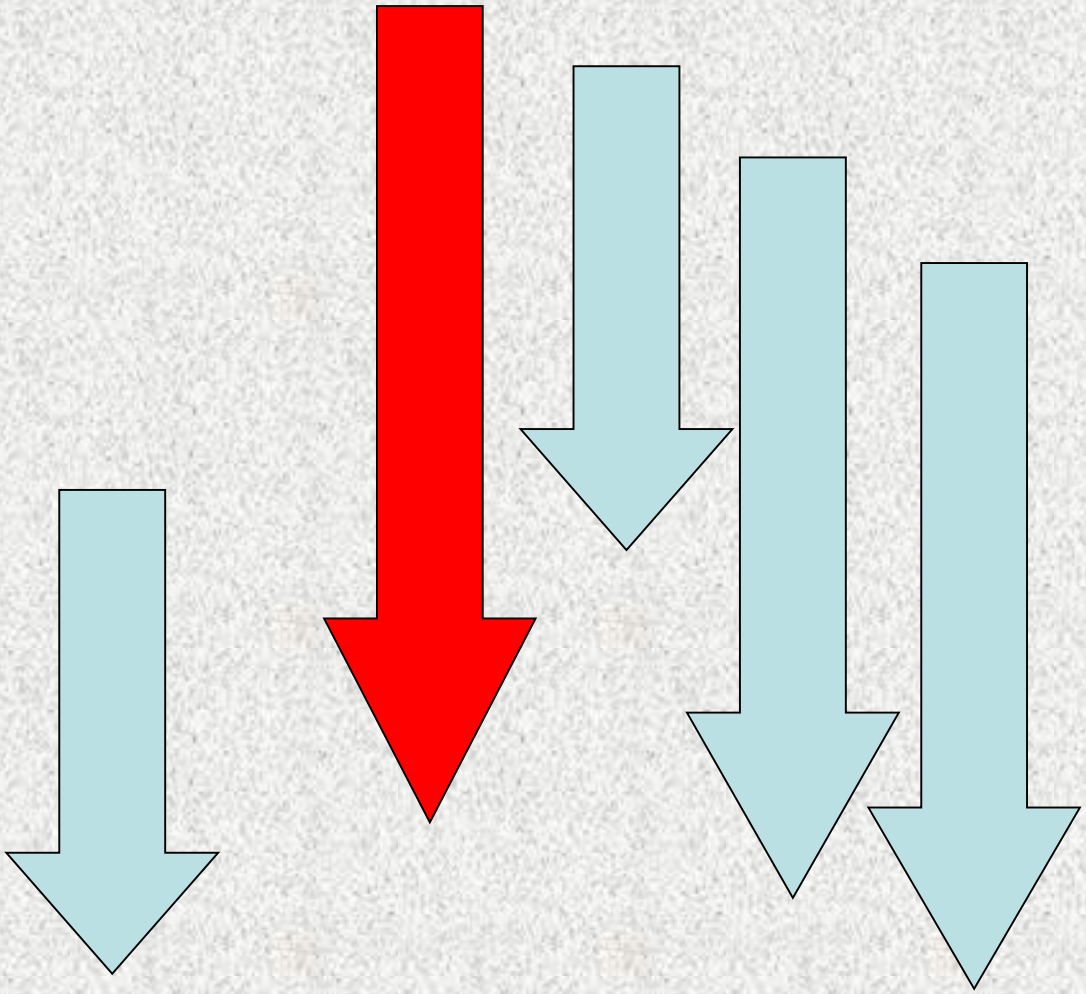
Vytvoření vlákna **extends**

```
class MojeVlakno0T1 extends Thread {
    @Override
    public void run() {
        System.out.println("Vlákno spuštěno ");
        System.out.println("Vlákno ukončeno ");
    }
}

public class DemoVlakno0T1 {

    public static void main(String[] args) {
        Thread t = new MojeVlakno0T1();
        t.start();
        System.out.println("Vlákno hlavní spuštěno ");
        System.out.println("Vlákno hlavní ukončeno ");
    }
}
```

Více vláken souběžně bez řízení

- Počet vláken není omezen, vlákna lze pojmenovat
- Dva způsoby:
  - Vytvoříme vlákno (konstruktor **`Thread`**) v konstruktoru a předáme mu odkaz na **`this`** objekt (třídy, která je **`Runnable`** – implementuje (překryjeme) **`run()`**), který bude „ve vláknu“ a odstartujeme metodou
    - Vytvořením objektu se vše provede
  - Vytvoříme vlákno (konstruktor **`Thread`**) v konstruktoru a předáme mu odkaz na **`this`** objekt (třídy, která je **`Runnable`** – implementuje **`run()`**), který bude „ve vláknu“
    - Vlastní start provedeme až po vytvoření objektu tak, ze zavoláme metodu atributu typu **`Thread – start()`**

Více vláken souběžně bez řízení



Více vláken souběžně bez řízení

- Pomocí metody **`Thread.sleep()`** můžeme vlákna uspávat
- Hlavní vlákno nemusí skončit jako poslední z vláken, zařídíme pomocí **`join()`**– je to dáno časem života hlavního vlákna

Více vláken souběžně bez řízení

```
class MojeVlakno2 implements Runnable {
String JmenoVlakna;
Thread vlakno;
public void run() {
try {
  System.out.println("Vlakno: " + JmenoVlakna);
  Thread.sleep(2000);
 } catch (InterruptedException v) {
  System.out.println("vyjimka" + JmenoVlakna + "preruseno");
 }
 System.out.println(" Ukonceni vlakna  " + JmenoVlakna);
  }

MojeVlakno2(String threadName) {
   JmenoVlakna = threadName;
   vlakno = new Thread(this, JmenoVlakna);
   vlakno.start();
    }
}
```

Více vláken souběžně bez řízení

```
public class DemoVlakno1_vice_vlaken0 {

public static void main(String[] args) {
 new MojeVlakno2("1");
 new MojeVlakno2("2");
 new MojeVlakno2("3");
 new MojeVlakno2("4");
 try {
    Thread.sleep(2000);
     } catch (InterruptedException v) {
    System.out.println(" preruseno hlavni vlakno");
     }
    System.out.println("Ukonceno hlavni vlakno");
    }
}
```

Více vláken souběžně bez řízení

```
public class DemoVlakno2a_vice_vlaken {

public static void main(String[] args) {
 MojeVlakno2a t1 = new MojeVlakno2a("1");
                                   t1.vlakno.start();
 MojeVlakno2a t2 = new MojeVlakno2a("2");
                                   t2.vlakno.start();
 MojeVlakno2a t3 = new MojeVlakno2a("3");
                                   t3.vlakno.start();
 MojeVlakno2a t4 = new MojeVlakno2a("4");
                                   t4.vlakno.start();
try {
    Thread.sleep(2000);
     } catch (InterruptedException v) {
    System.out.println(" preruseno hlavni vlakno");
     }
    System.out.println("Ukonceno hlavni vlakno");
    }
}
```

Vlákna v Javě, další metody `Thread`

`String getName()` – vrací jméno vlákna

`int getPriority()` – vrací prioritu vlákna

`void join()` – vlákno čeká na ukončení vlákna

`static void sleep()` – pozastaví vlákno na určenou dobu

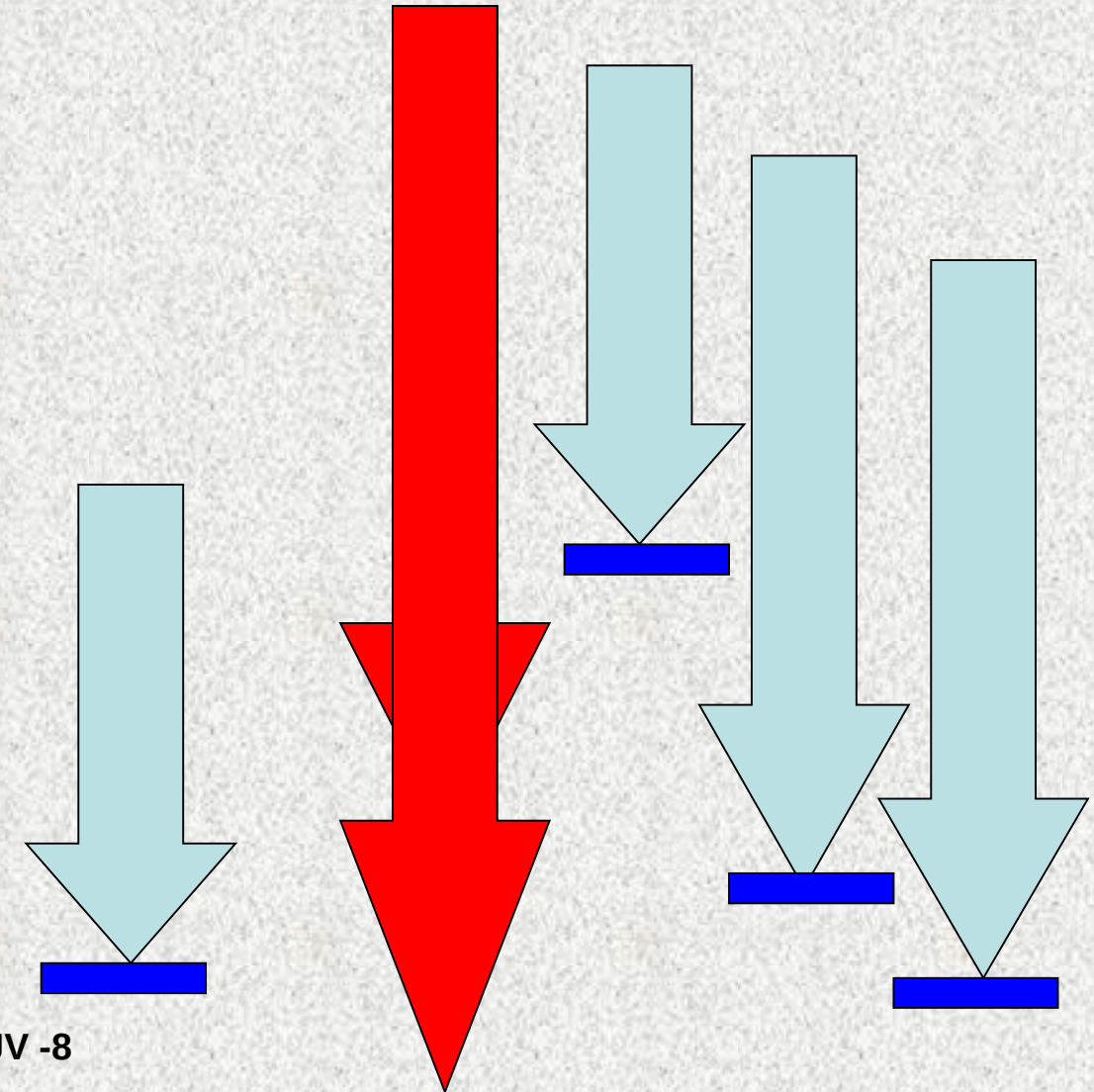
`boolean isAlive()` – zda vlákno běží

`static void yield()` – běžící vlákno předá řízení jinému

Čekání na ukončení běhu vlákna

- `static void sleep()` – pozastaví vlákno na určenou dobu
- `boolean isAlive()` – zjistí, zda vlákno běží
- `void join()` – vlákno čeká na ukončení dceřiného vlákna, aktuální vlákno se zastaví a bude čekat na skončení běhu onoho vlákna.
  - Lze využít i verze s časovým limitem - pak se bude čekat maximálně po zvolenou dobu, čeká, dokud se dceřiný proces neukončí
  - Vlákno je slušné a počká až mateřské dokončí nebo čeká jen učitou dobu, určitě hlavní vklákno nepředběhne

Čekání na ukončení běhu vlákna



Čekání na ukončení běhu vlákna

```
public class DemoVlakno3_join_isAlive {

public static void main(String[] args) {
  MojeVlakno vlakno1 = new MojeVlakno("1");
/////
////
System.out.println("Vlakno1 zije:" + vlakno1.vlakno.isAlive());
/////
////
try {
 vlakno1.vlakno.join();
/////
/////
} catch (InterruptedException e) {
  System.out.println("Hlav.vlákno přerušeno");
}
  System.out.println("Vlakno1: " +
        vlakno1.vlakno.isAlive());
//////
/////
  System.out.println("Hlavní vlákno ukončeno");
    }
}
```

Priority vláken

Jsou-li běhuschopná dvě vlákna, pak je řízení předáno tomu vláknu, které má vyšší prioritu a je ve stavu runnable

O předání řízení (přidělení CPU) se stará JVM Scheduler.

- Nastavení priority **`setPriority()`**
- Zjištění priority **`getPriority()`**
- Hodnoty priority
  - **`MAX_PRIORITY - 10`**
  - **`MIN_PRIORITY - 1`**
  - **`NORM_PRIORITY - 5`**

Priorita vláken

Pravidla plánovače:

- Běží vždy to, co má z běhuschopných vláken nejvyšší prioritu
- Je-li více vláken se stejnou prioritou, je řízení postupně předáváno všem v náhodném pořadí, možno vynutit předání `yield()`
- Vlákno s nižší prioritou mohou získat řízení, jen když vlákna s vyšší prioritou se dostanou do neběhuschopného stavu, vlákno s vyšší prioritou nelze přinutit k předání řízení standardním mechanismem plánování, jen zásahem zvenku
- Pokud se do běhuschopného stavu dostane vlákno s vyšší prioritou, je běžící vlákno okamžitě přinuceno předat řízení ve prospěch tohoto vlákna (preemptivní plánování)

Priorita vláken - pravidla

- Čekání na vstup - typický případ, běžící vlákno je přerušeno vláknem vyšší priority
- Vlákno je automaticky uvedeno do stavu neběhuschopné, když čeká na vstup/výstup
- Vstup/výstup má nejvyšší prioritu, kdykoli přeruší, když je možnost předávání dat
- Není třeba další synchronizace, spolupráce je asynchronní

Priority vláken

```
public class DemoVlakno4_priorita {
public static void main(String[] args) {
 Thread.currentThread().setPriority(2a);
 MojeVlakno5 lowPriority = new MojeVlakno5(3, "Vlakno s nízkou
   prioritou");
 MojeVlakno5 highPriority = new MojeVlakno5(7, "Vlakno s vysokou
   prioritou");
 lowPriority.start();
 highPriority.start();
 try {
  Thread.sleep(2a000);
     } catch (InterruptedException e) {
  System.out.println("Hlavní vlákno přerušeno");
      }
  highPriority.stop();
  lowPriority.stop();
 try {
  lowPriority.v.join();
  highPriority.v.join();
    } catch (InterruptedException e) {
   System.out.println("Výjimka (InterruptedException e)");
        }    }}
```

Priority vláken

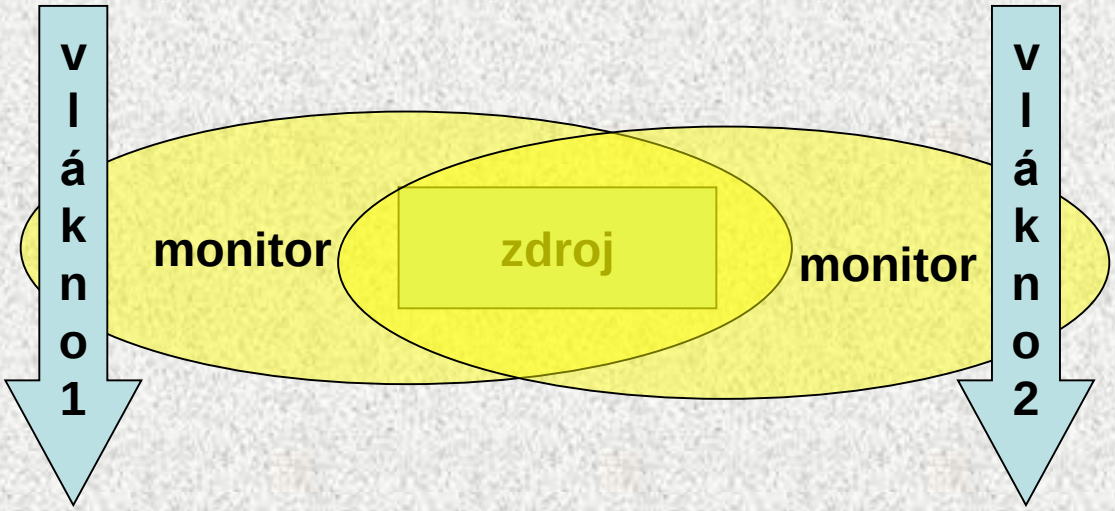
```
class MojeVlakno5 implements Runnable {
    Thread v;
    private boolean spusteno = true;

    public MojeVlakno5(int p, String tName) {
        v = new Thread(this, tName);
        v.setPriority(p);
    }

    public void run() {
        System.out.println(v.getName() + " běží");
    }

    public void stop() {
        spusteno = false;
        System.out.println(v.getName() + " zastaveno");
    }

    public void start() {
        System.out.println(v.getName() + " spuštěno");
        v.start();
    }
}
```

Synchronizace v Javě

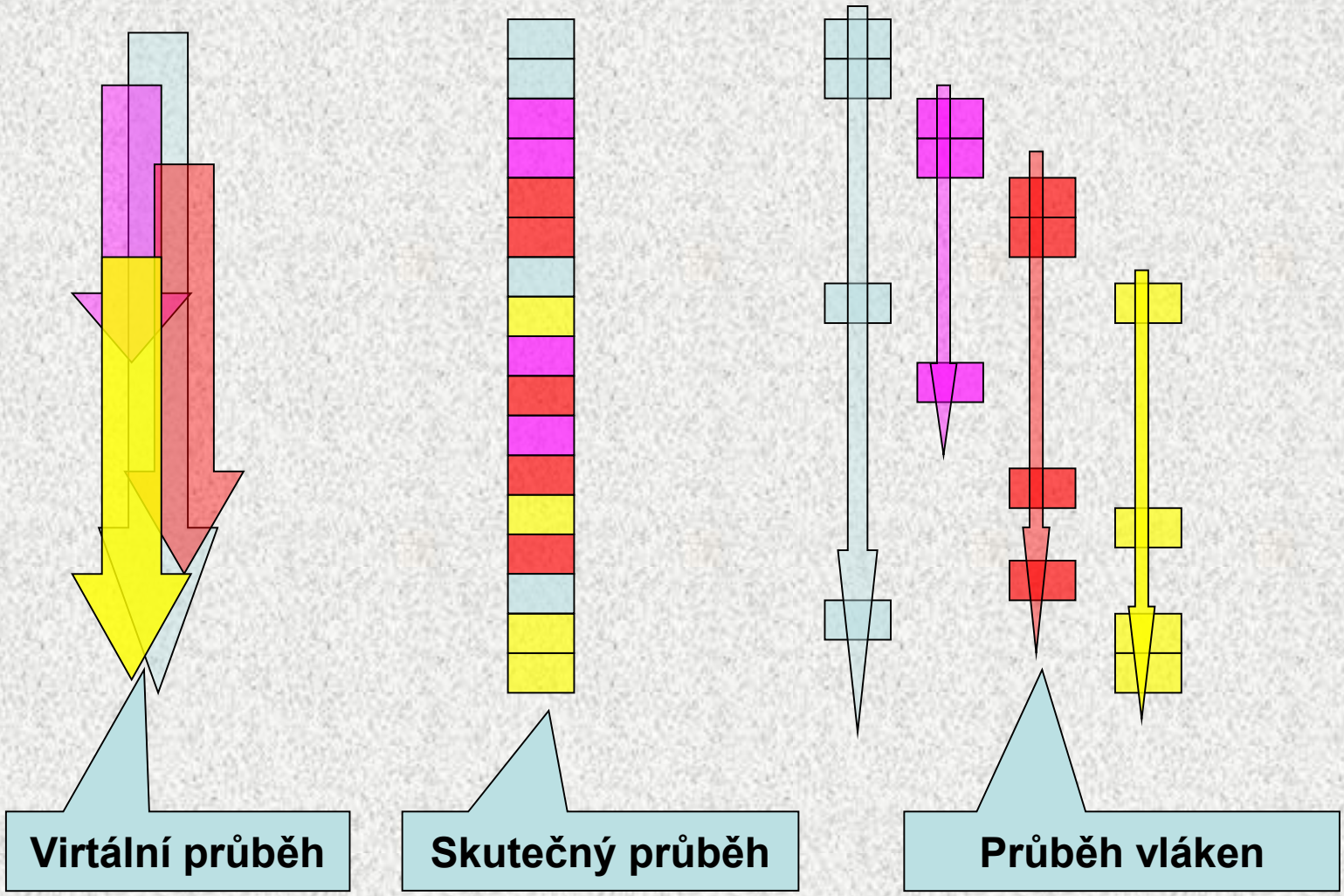
Hlavní synchronizačním primitivem jsou monitory

Každý objekt má automaticky přiřazen svůj monitor.

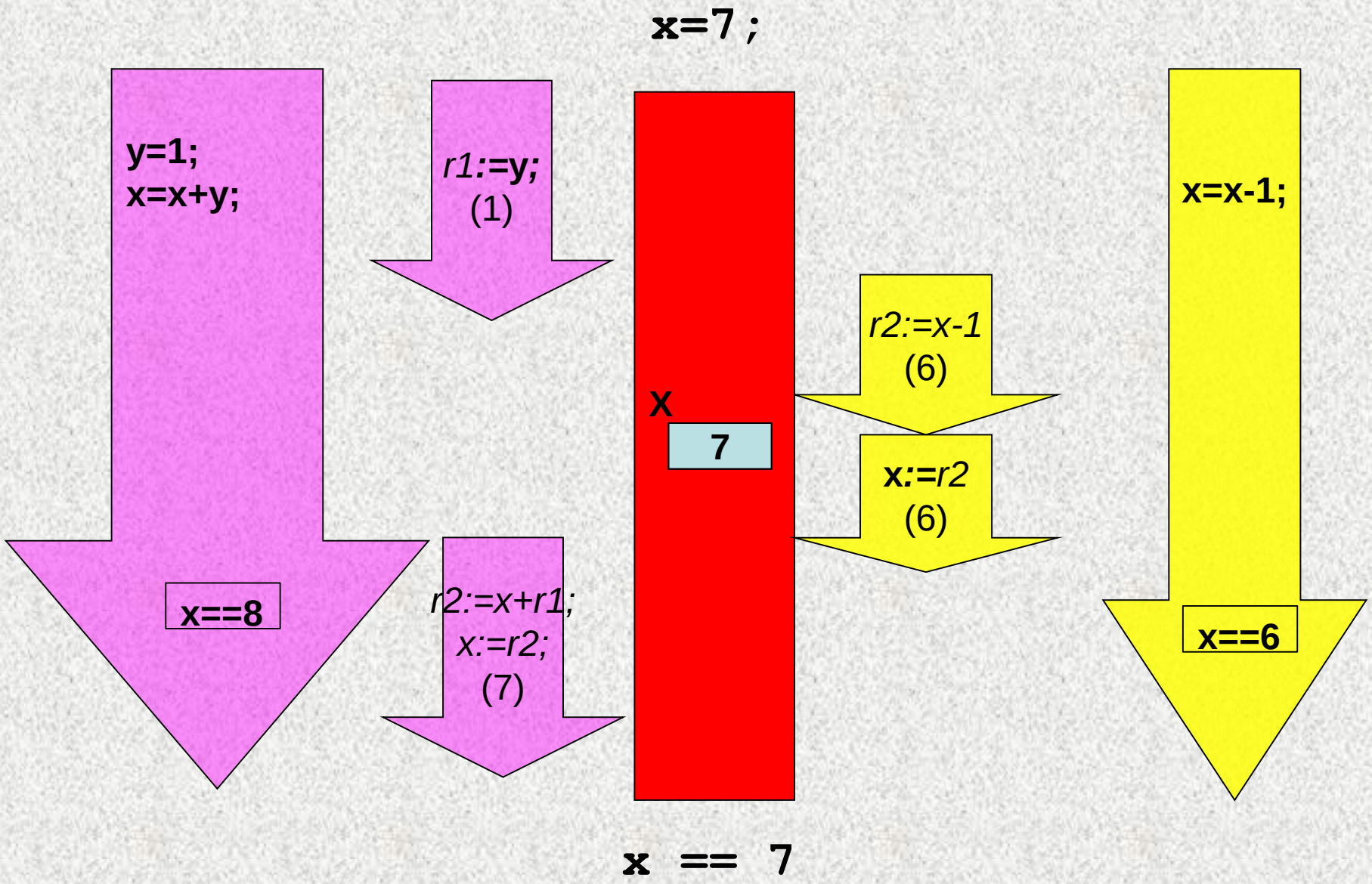
1. Metody, které patří do monitoru, jsou označeny pomocí klíčového slova **`synchronized`**.
2. Do monitoru libovolného objektu však lze obalit libovolný příkaz (blok) kódu pomocí konstrukce pro objekt, který tuto metodu volá: **`synchronized(objekt) { ... }`**



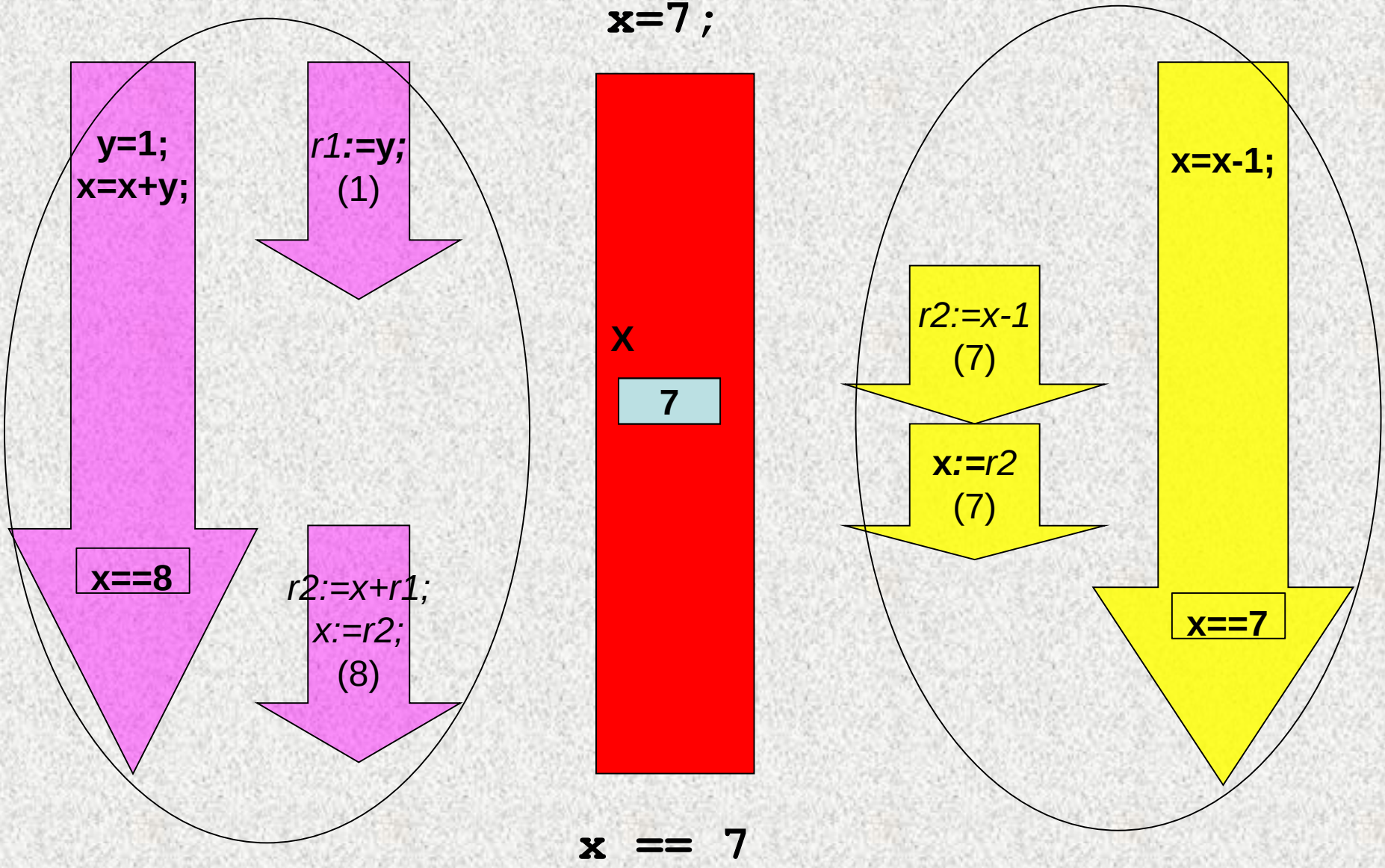
Vlákna – průběh skutečný a virtuální



Problém sdílení prostředků, kolize



Problém sdílení prostředků, synchronizace



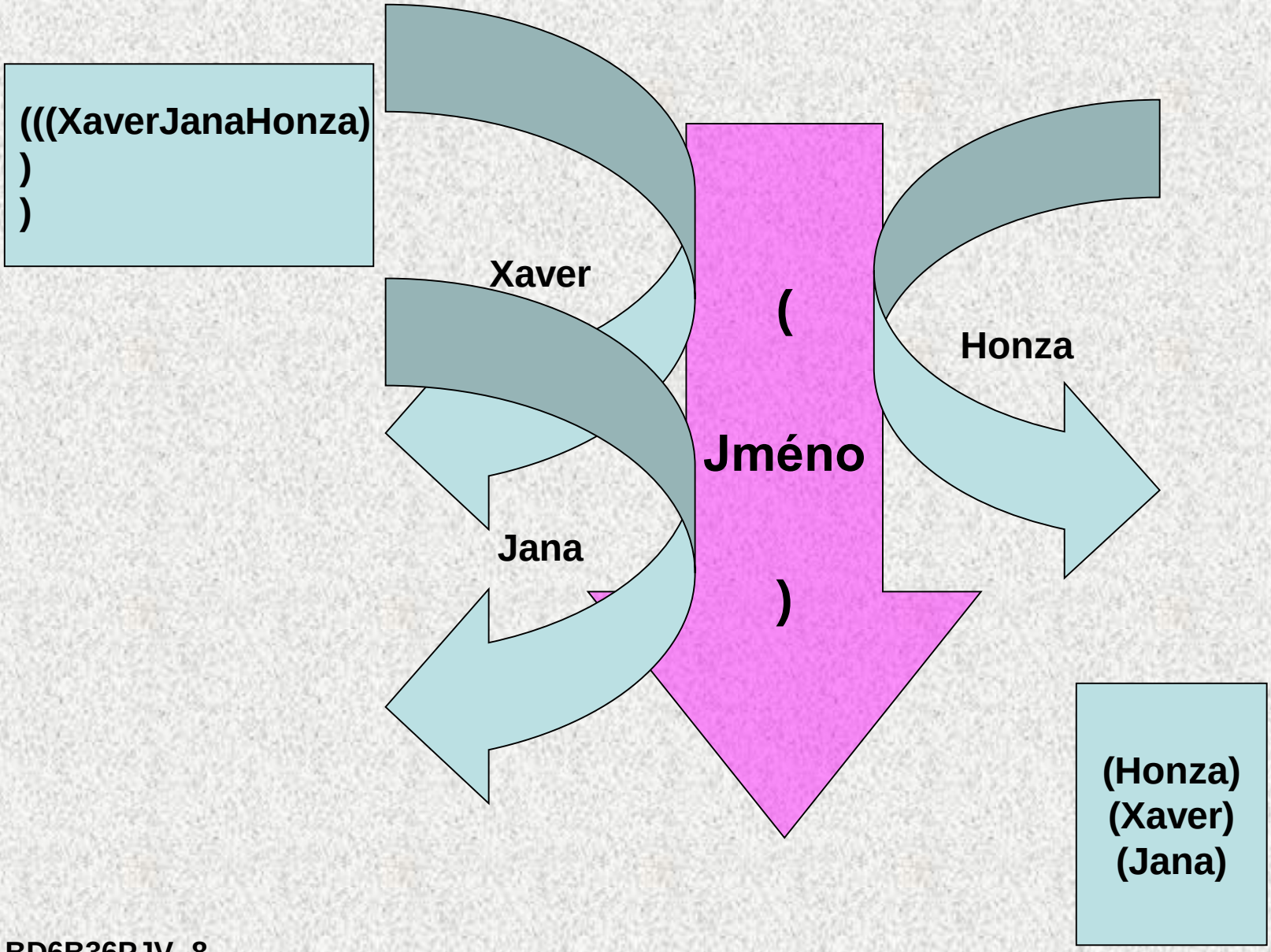
Synchronizace činnosti vláken

- Vlákna spolupracují, problém sdílení datového prostoru
  - Problém nedeterminovanosti přístupu ke společným prostorům
- Možné řešení je tzv. monitor (kritická sekce)
  - objekt, který vláknu zpřístupní zdroj (paměť, linku,..),
  - v daném okamžiku aktivně umožní monitor používat jen jedno vlákno
  - „pro daný časový interval vlákno vlastní monitor“, monitor smí vlastnit jen jedno vlákno
  - vlákno běží, jen když vlastní monitor, jinak čeká
- Všechny objekty v Javě „mají“ monitor
  - Vlákno vstoupí do monitoru **voláním metody** s přívlastkem **`synchronized`** (**lze i příkaz**) - tuto metodu resp. příkaz nazýváme synchronizovanou
  - O vláknu, které úspěšně jako první zavolá synchronizovanou metodu říkáme, že je „uvnitř“ metody a má k dispozici všechny zdroje používanými touto metodou
  - Jiné vlákno, které volá synchronizovanou metodu čeká, dokud aktivní vlákno synchronizovanou metodu neopustí (nebo uvolní zámek pomocí **`wait`**)
- Synchronizace - řízená „linearizace vláken“

Synchronizace činnosti vláken

- Synchronizace
  - metodou – v třídě, která bude vstupovat do vlákna
  - příkazem – mimo třídy, ve vláknu se určí, který příkaz kterého objektu je „kritický“

Příklad na závorky, synchronizovaná metoda



Synchronizovaná metoda

```
class Zavorky {

   synchronized void zobrazit(String s) {
        System.out.print("(");
        try {
            Thread.sleep(2a00);
        } catch (InterruptedException e) {
            System.out.println(" Preruseno");
        }
        System.out.print(s);
        try {
            Thread.sleep(2a00);
        } catch (InterruptedException e) {
            System.out.println(" Preruseno");
        }
        System.out.println(")");
    }
}
```

Synchronizovaná metoda

```
class MyThread implements Runnable {

    String s1;
    Zavorky z1;
    Thread v;

    public MyThread(Zavorky z2, String s2) {
        z1 = z2;
        s1 = s2;
        v = new Thread(this);
        v.start();
    }

    public void run(){
    z1.zobrazit(s1);
    }
    }
```

Synchronizovaná metoda

```
public class DemoVlakno5_zavorky {

    public static void main(String[] args) {
        Zavorky z3 = new Zavorky();
        MyThread name1 = new MyThread(z3, "Honza");
        MyThread name2 = new MyThread(z3, "Jana");
        MyThread name3 = new MyThread(z3, "Xaver");

}
    }
```

Synchronizovaný příkaz

```
class Zavorky2 {
    void zobrazit(String s) {
        System.out.print("(");
        try {
            Thread.sleep(2a00);
        } catch (InterruptedException e) {
            System.out.println(" Preruseno");
        }
        System.out.print(s);
        try {
            Thread.sleep(2a00);
        } catch (InterruptedException e) {
            System.out.println(" Preruseno");
        }
        System.out.println(")");
    }
}
```

Synchronizovaný příkaz

```
class MyThread2 implements Runnable {
    String s1;
    Zavorky2 z1;
    Thread v;

    public MyThread2(Zavorky2 z2, String s2) {
        z1 = z2;
        s1 = s2;
        v = new Thread(this);
        v.start();
    }

    public void run() {
        synchronized (z1){
            z1.zobrazit(s1);
        }
    }
}
```

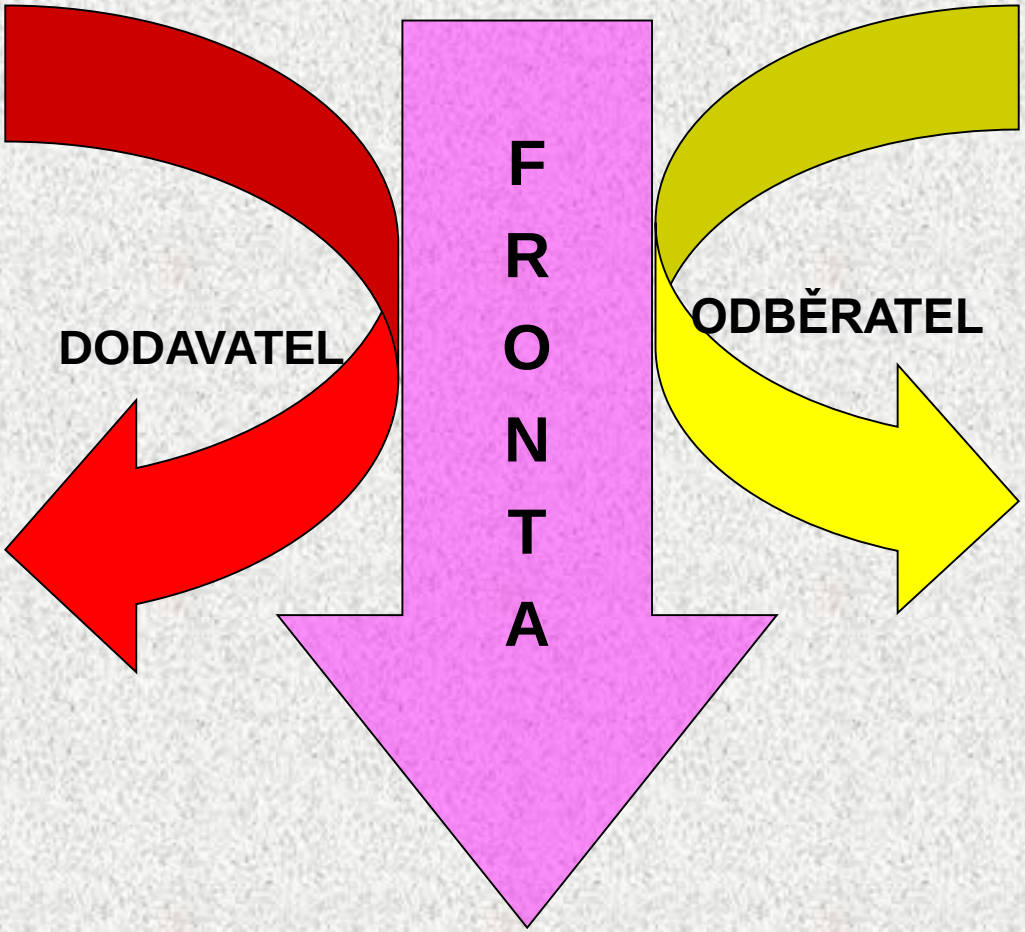
Synchronizovaný příkaz

```
public class DemoVlakno5_zavorky_2 {

    public static void main(String[] args) {
        Zavorky2 z3 = new Zavorky2();
        MyThread2 name1 = new MyThread2(z3, "Honza");
        MyThread2 name2 = new MyThread2(z3, "Jana");
        MyThread2 name3 = new MyThread2(z3, "Xaver");
}
    }
```

Komunikace mezi vlákny

- Nejedná se o společnou metodu různých vláken kterou vlákna musí používat synchronně
- Jedná se o různé metody různých vláken nad týmž prostorem - uvnitř synchronizované metody lze volat metody, které umožní ovládat komunikaci mezi vlákny:
- Řeší se to synchronizovanou metodou, která obsahuje:
  - **`wait()`** – uvolni monitor a pozastaví svou činnost
    - do doby signálu od **`notify()`** jiného vlákna
    - do daného času
  - **`notify()`** – signál pro vlákno pozastavené **`wait()`** aby obnovilo činnost a převzalo monitor
    - **`notifyAll()`** – probudí všechna vlákna pozastavená metodou **`wait()`**, monitoru se zmocní vlákno s nejvyšší prioritou

Komunikace mezi vlákny



Komunikace mezi vlákny, příklad

vydavatel

Vlozit do Fronty

Není obsazeno

1) **vložit**
2) **obsazeno = !obsazeno**
3) **notify() zakazníka**

Je obsazeno

1) **wait()**

zákazník

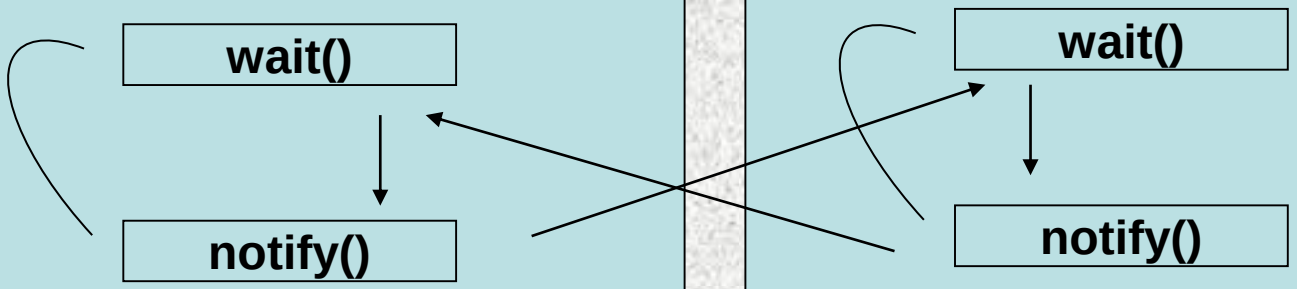
Vyjmout z Fronty

Je obsazeno

1) **vyjmout**
2) **obsazeno = !obsazeno**
3) **notify() vydavatele**

Není obsazeno

1) **wait()**



Komunikace mezi vlákny

```
class Fronta {
 int vyjmenovanaHodnota;
 boolean obsazeno = false;

 synchronized int vyjmout() {
   if (!obsazeno) {
     try {
      wait(); // zastavi  toto vlakno az do jeho probuzeni
                          metodou notify() jinym vlaknem
        } catch (InterruptedException v) {
       System.out.println("Vyjmout: (InterruptedException ");
          }
       }
       obsazeno = false;
       System.out.println("Vyjmout:" + vyjmenovanaHodnota);
       notify();
      // spust vlakno, ktere pozastavilo toto vlakno
       return vyjmenovanaHodnota;
   }
```

Komunikace mezi vlákny

```
synchronized void vlozit(int vyjmenovanaHodnota) {
 if (obsazeno) {
  try {
   wait(); // zastavi  toto vlakno az do jeho probuzeni
  metodou notify() jinym vlaknem
  } catch (InterruptedException v) {
  System.out.println("Vlozit: (InterruptedException ");
            }
        }
  this.vyjmenovanaHodnota = vyjmenovanaHodnota;
  obsazeno = true;
  System.out.println("Vlozit:" + vyjmenovanaHodnota);
  notify(); // spust vlakno, ktere pozastavilo toto
  vlakno
    }
}
```

Komunikace mezi vlákny

```
class Vydavatel implements Runnable {

    Fronta f;
    Thread t;

    Vydavatel(Fronta f) {
        this.f = f;
        t = new Thread(this, "Vydavatel");
        t.start();
    }

    public void run() {
        for (int i = 0; i < 5; i++) {
            f.vlozit(i);
        }
    }
}
```

Komunikace mezi vlákny

```
class Zakaznik implements Runnable {

    Fronta f;
    Thread t;

    Zakaznik(Fronta f) {
        this.f = f;
        t = new Thread(this, "Zakaznik");
        t.start();
    }

    public void run() {
        for (int i = 0; i < 5; i++) {
         f.vyjmout();
        }

    }
}
```

Komunikace mezi vlákny

```
public class DemoVlakno7 {

    public static void main(String[] args) {
        Fronta f = new Fronta();
        Vydavatel t1 = new Vydavatel(f);
        Zakaznik t2 = new Zakaznik(f);
    }
}
```

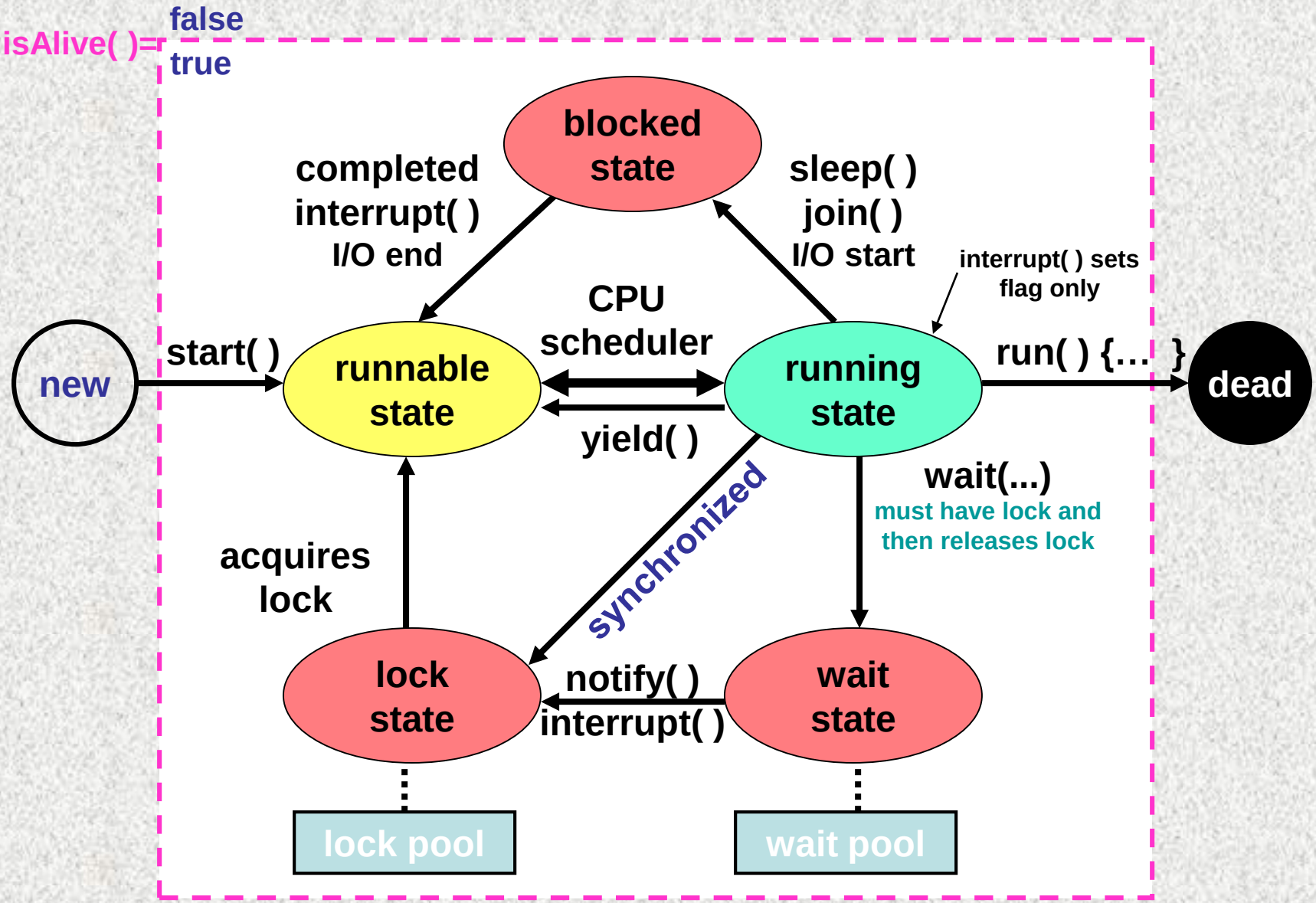
Stavy vláken

- Nové vlákno, spuštěno **`start()`**
- Běžící **`(running)`**, jedno z vláken je běžící, ostatní čekají
- Běhuschopné **`(runnable)`**, je spuštěno, ale neběží, čeká na předání řízení
- Neběhuschopné
  - Blokováno (blocked)
    - uspáno **`sleep()`**, **`join()`**, čeká na O/I
  - Čekající (wait)
    - čeká **`wait()`**
- Mrtvé vlákno, skončila metoda **`run()`**

Doba a pořadí předávání řízení závisí na:

- na stavu okolních vláken,
- na prioritě vlákna,
- na schopnostech OS.

Stavy vlákna



Čekání na ukončení běhu vlákna, `join()` - 1

```
class Vlakno2 extends Thread {

    Vlakno2() {
        super("Vlakno2");
    }

    public void run() {
        System.out.println("Vlakno2 ");
        try {
            Thread.sleep(500);
        } catch (InterruptedException v) {
            System.out.println("vyjimka");
        }
        System.out.println("Vlakno2");
    }
}
```

Čekání na ukončení běhu vlákna, `join()` - 2

```
class Vlakno1 extends Thread {
    Vlakno1() {
        super("Vlakno");
    }
    public void run() {
        System.out.println("Vlákno");
        Thread t = new Vlakno2();
        t.start();


        try {
            t.join();//Thread.sleep(2a00);
        } catch (InterruptedException v) {
            System.out.println("vyjimka");
        }


        System.out.println("Vlákno1");
    }
}
```

Čekání na ukončení běhu vlákna, `join()` - 3

```
public class DemoVlakno3aVlakna {

    public static void main(String[] args) {
        System.out.println("Vlákno hlavní spuštěno ");
        Thread t = new Vlakno1();
        t.start();

        try {
            t.join();//Thread.sleep(2000);
        } catch (InterruptedException v) {
            System.out.println("vyjimka");
        }

        System.out.println("Vlákno hlavní ukončeno ");
    }
```