Multiagentové systémy

Dr. Andrej Lúčny

KAI FMFI UK

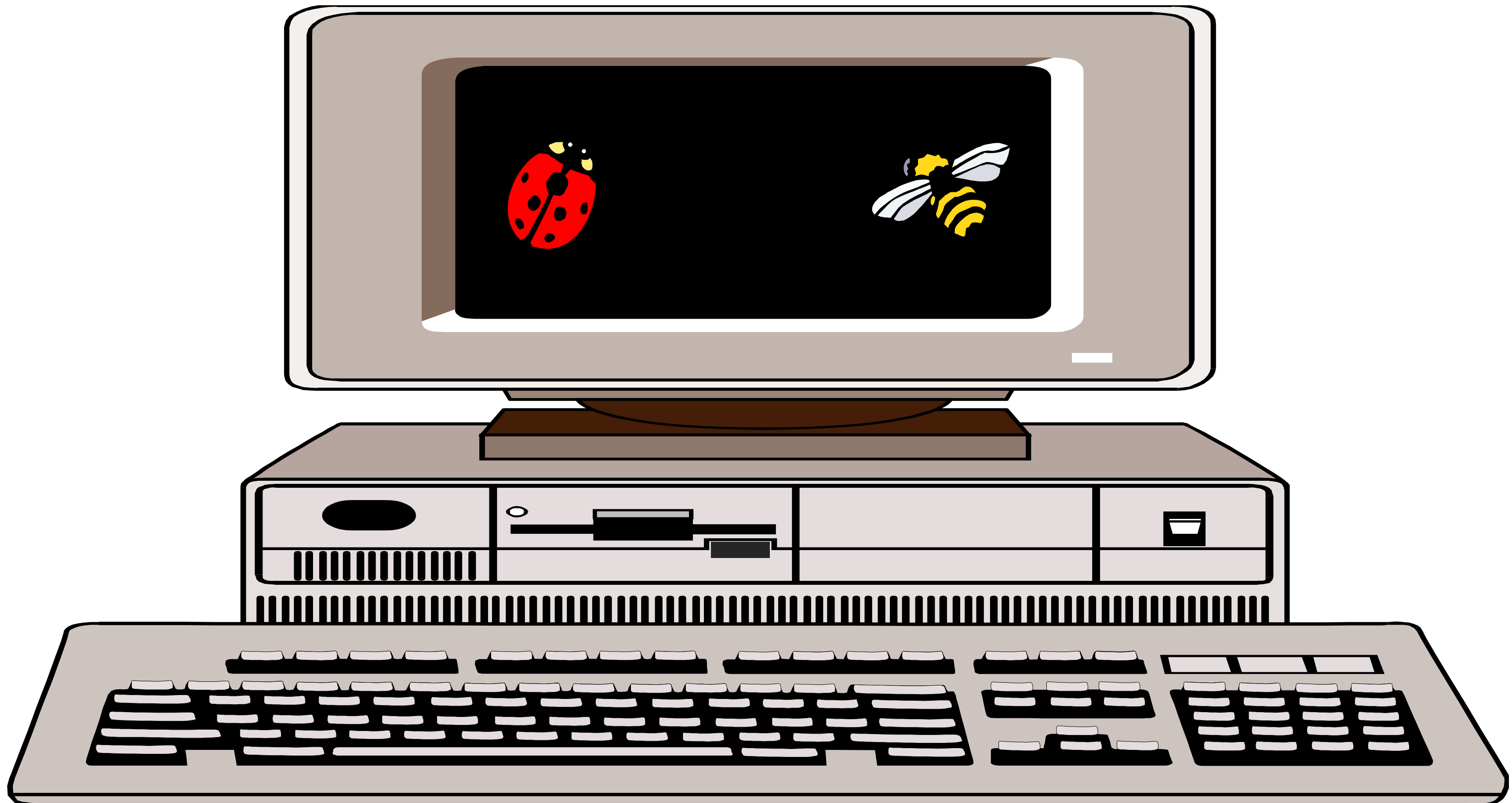
andy@microstep-mis.com

http://www.microstep-mis.sk/~andy

Opakovanie

- čo je klient a čo je server ?
- aké postavenie majú multiagentové systémy medzi distribuovanými systémami ?
- čo je space vzhľadom na klient-server (nepriama kom.) ?
- čo je agent vzhľadom na klient-server (nepriama kom.) ?
- aké služby poskytuje space ?
- koľko rečových aktov upotrebia space-y ?
- koľko rečových aktov by upotrebili systémy obmedzujúce na priamu komunikáciu, ktoré by mali usporiadanie typu peer-to-peer ?
- ako vyzerá analógia na VM ?

MAS na VM



Virtuálna „mašina“

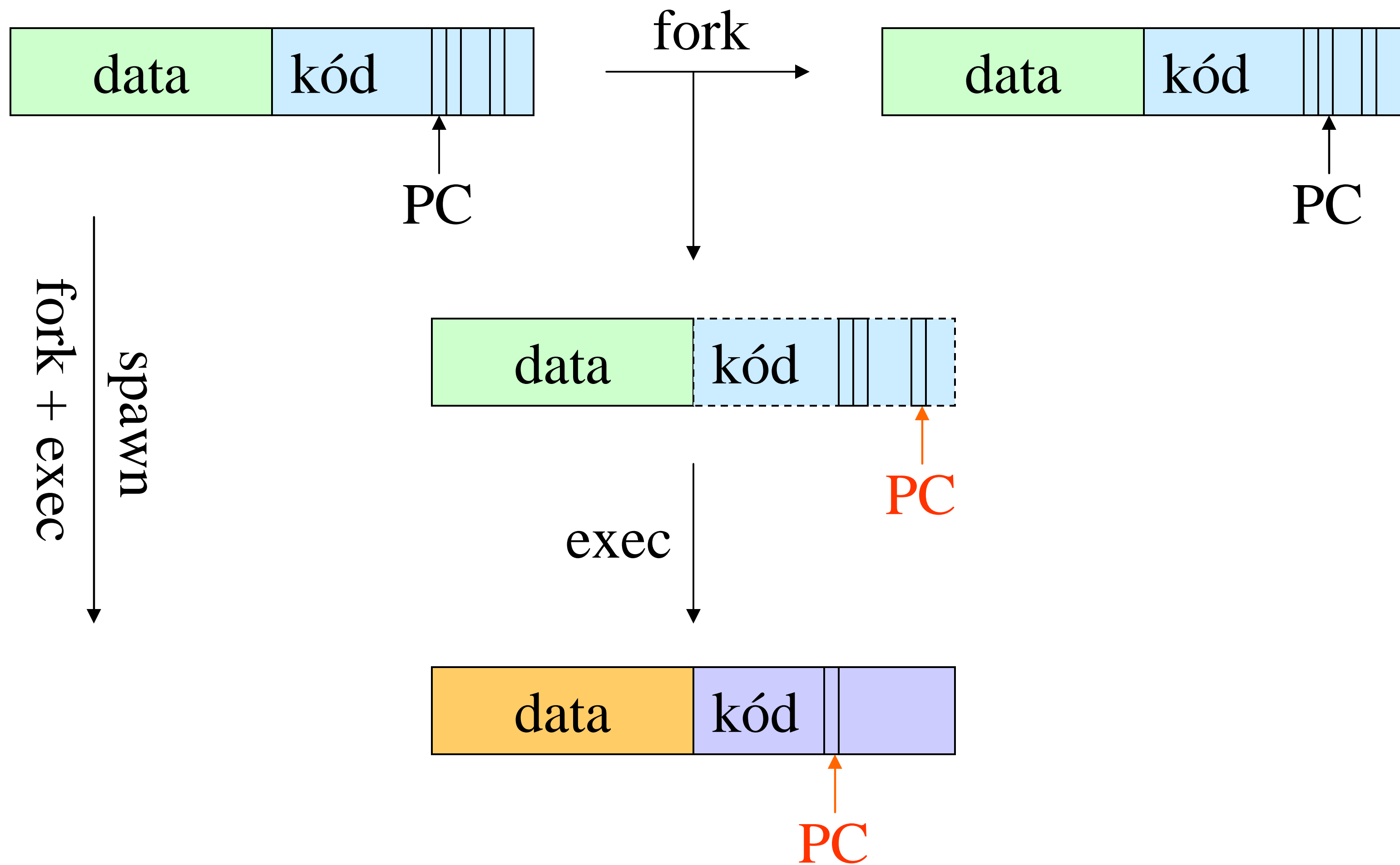
Softwarové prostredie, ktoré:

- zahladzuje rozdiely v operačnom systéme a hardware
- umožňuje spustiť programy skompilované do tzv. bytecode, t.j nevyužíva natívne operácie procesoru ale interpretuje niečo binárne
- týmto programom vytvára multi-(procesové, vláknové, objektové, ...) prostredie
- poskytuje im základnú sadu funkcií, tzv. primitívy

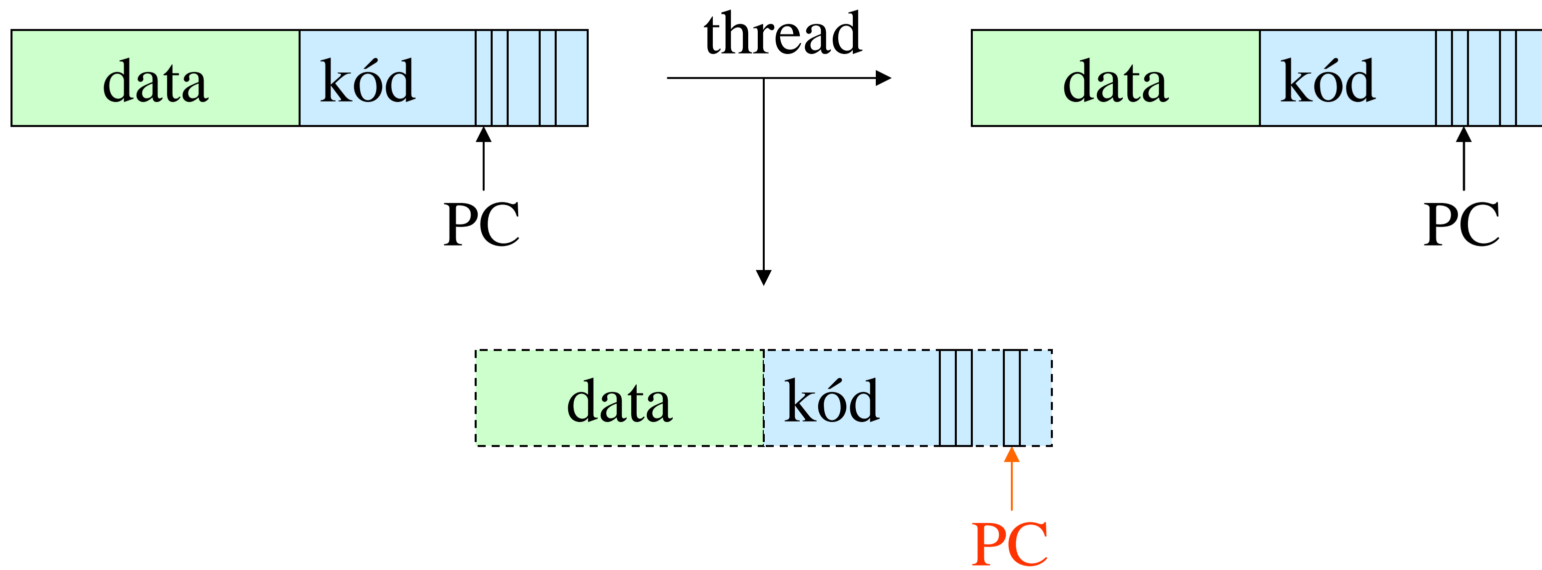
Procesy, vlákna a objekty

- proces je spustený program, ktorý má svoj kód, PC (program counter) a dáta
- vlákno je spustený program, ktorý má svoj kód, PC, ale dáta zdieľa s ostatnými vláknami
- objekt je „spustený“ program, ktorý má svoje dáta a kód, ale nemá PC

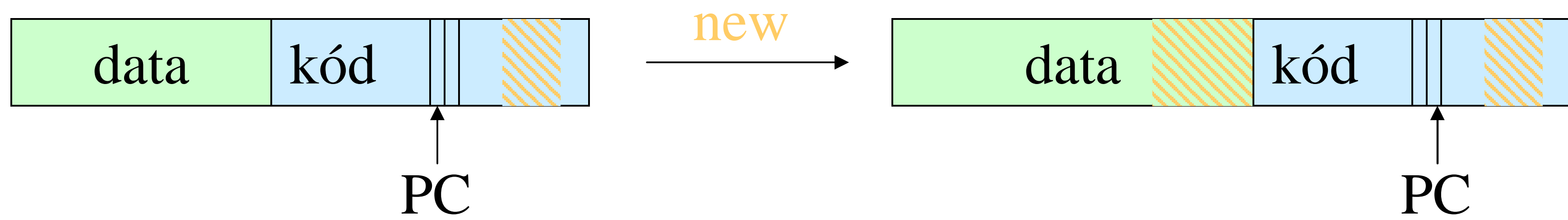
Vznik procesu



Vznik vlákna



Vznik objektu



OOP

- VM boli stvorené pre OOP (prvý raz pre Smalltalk) (hoci pojem je starší, napr. BIOS je „VM“ pre IBM PC)
- do počítača prenášame entity reálneho alebo virtuálneho sveta ako objekty a vzťahy medzi nimi definujeme posielaním správ

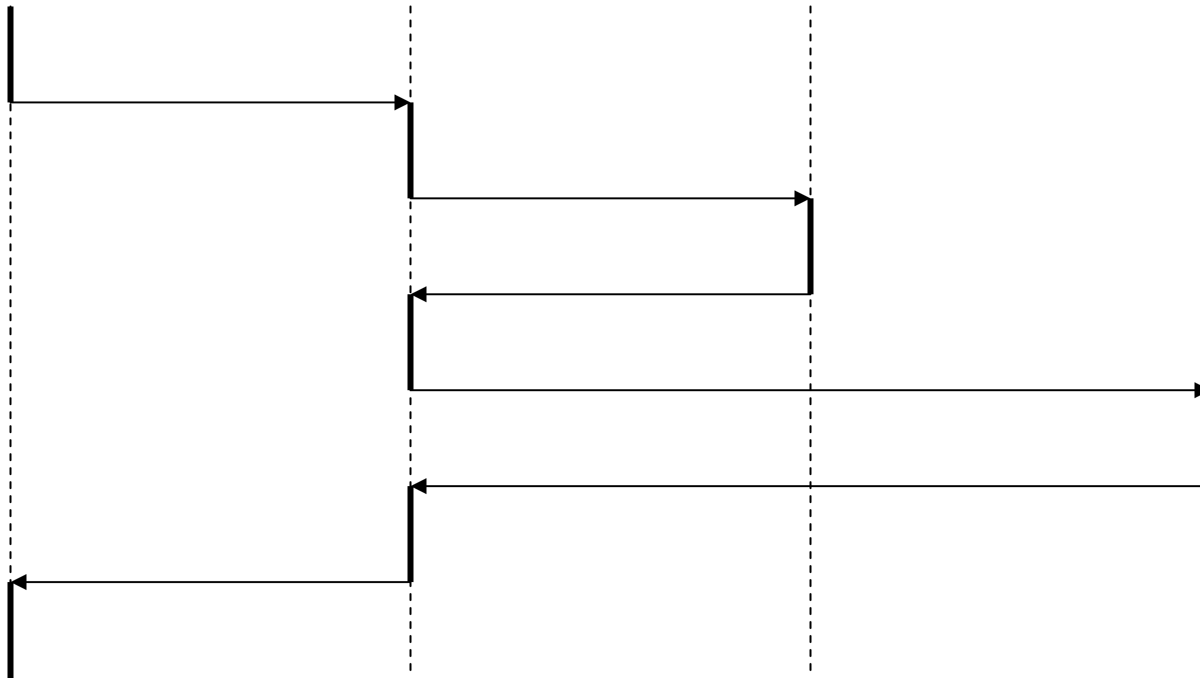
```
objekt.sprava(argumenty)
```

Modely výpočtu v OOP

- Štandardný – triedno-inštančný model
- objekt vzniká ako inštancia triedy
- medzi triedami existuje dedičnosť
- posielanie správy je synchrónne (zablokujeme sa a čakáme na odpoveď)
- poslanie správy zodpovedá zavolaniu metódy s určitými argumentmi

Riadenie v štandardnom modeli

- riadenie sa odovzdáva zavolaním metódy iného objektu a vracia odovzdaním návratovej hodnoty z nej

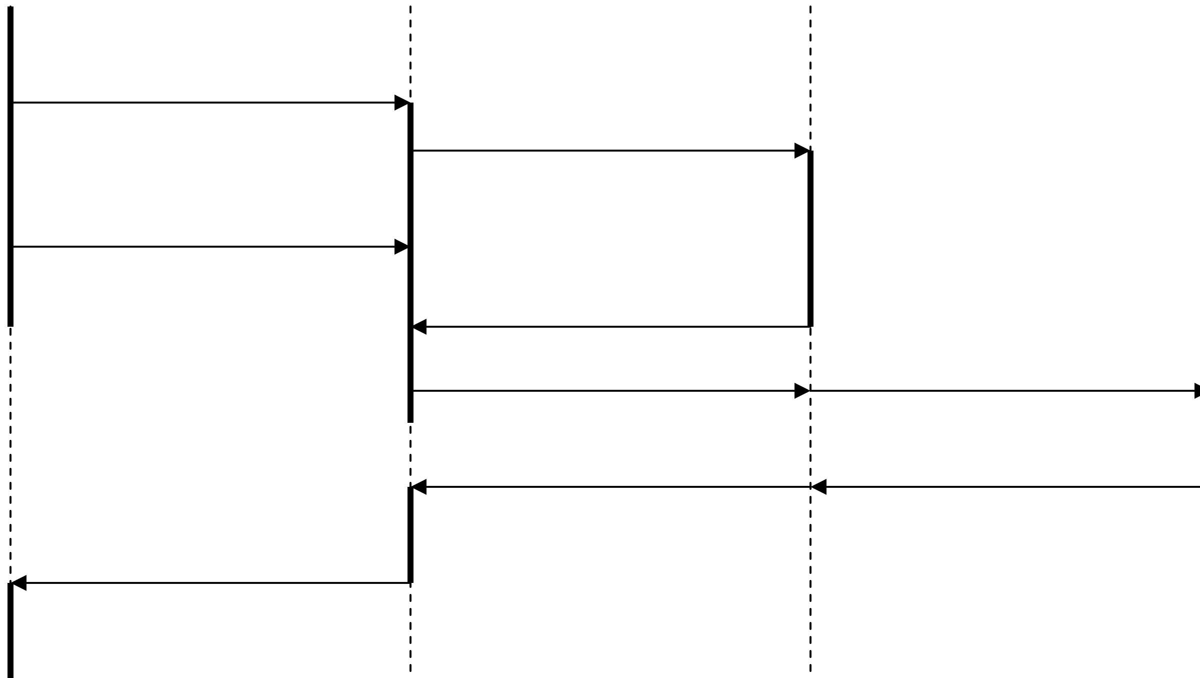


Modely výpočtu v OOP

- neštandardný – aktorový model
- objekt (aktor) vzniká ako upravená kópia iného objektu
- medzi objektami existuje delegovanie
- posielanie správy je asynchrónne (nezablokujeme sa a ideme ďalej)
- poslanie správy zodpovedá (paralelnému) zavolaniu metódy s určitými argumentami

Riadenie v aktorovom modeli

- riadenie sa zavolaním metódy iného objektu neodovzdáva, máme viac PC, ktoré sa spravodlivo striedajú



Implementácia aktorov v štandardnom modeli

- Aktorový model výpočtu môžeme zrealizovať v štandardnom modeli tým, že zriadime tzv. MessageQueue a to jedno PC čo máme použijeme na to, aby sme vždy vybrali z neho správu a vykonali metódu, ktorá jej zodpovedá čím sa ďalšie správy zaradia do MessageQueue

Aktory a agenty

- Aktor je veľmi podobný až zhodný agentovi v koncepte priamej komunikácie
- Aktor nedisponuje prostriedkami nepriamej komunikácie
- (Aktor je o čosi viac vo vleku správ, ktoré mu prichádzajú, než agent. Je to dané tým, že v aktorovi je správa pevne previazaná s metódou, ktorá ju ošetruje. V agentovi sú správu preberané spravidla všeobecným handlerom.)

Aktory a agenty

- Aktory sú dotiahnuté ad absurdum – niet v systéme iných štruktúr
- Agenty naopak pracujú so štruktúrami jednoduchšej povahy, spravidla s objektami, alebo záznamami (record, struct)
- Hybridná povaha MAS umožňuje programátorovi lepšie rozdeliť, ktoré entity v systéme budú „pokročilejšie“ štruktúry a ktoré „zastaralejšie“. (Nepoužije sa delo na vrabce, ani vzduchovka na tanky)

Implementácia MAS na VM

- spravidla v štandardnom modeli
- dosť zle funguje v jednovláknovej VM. Je to dané práve hybridnosťou MAS a následnou potrebou (preemptívne) prerušovať výpočet v agentovi
- lepšie je, keď má každý agent samostatné vlákno, aby mohol byť (preemptívne) prerušený.
- preto je VM tým lepšia, čím má nižšiu latenciu

Implementácia MAS na VM

- priamu komunikáciu zabezpečíme cez MessageQueue (jeden centrálny alebo každý agent bude mať svoj)
- nepriamu komunikáciu cez Space

JVM

- príkladom VM je Java Virtual Machine
- poskytuje multi-objektové prostredie
- poskytuje multi-vláknové prostredie
- neposkytuje multi-procesové prostredie

Vlákna

trieda Thread

```
final
class A extends Thread {
  public A () {
    start();
  }
  public void run () {
    for (;;) { … };
  }
}
```

rozhranie Runnable

```
class A implements Runnable {
  public A () {
    new Thread(this).start();
  }
  public void run () {
    for (;;) { … };
  }
}
```

Synchronizácia vlákien

- každý objekt má v JVM tzv. monitor, ktorý umožňuje synchronizovať vlákna vykonávajúce operácie s týmto objektom
- monitor môže „vlastniť" najviac jedno vlákno
- na vyznačenie kritických oblastí vzhľadom na určitý objekt slúži prvok jazyka

```
synchronized (obj) { …
}
```

Synchronizácia vlákien

- keď chce vlákno posunúť svoje PC v kritickej oblasti, musí vlastniť monitor objektu na ktorý je kritická
- ak monitor už vlastní iné vlákno, vlákno počká.
- ak monitor nik nevlastní, vlákno ktoré chce ísť dnu sa stáva jeho vlastníkom a vniká dnu

Synchronizácia vlákien

- Synchronizované metódy

```
… method (…) {
  synchronized (this) {
    …
  }
}
```

```
synchronized … method (…) {
  …
}
```

- Synchronizované objekty – objekty vzoru adaptor, ktoré zaobaľujú určité objekty tým, že z každej ich metódy urobia synchronizovanú

Synchronizácia vlákien

- Vlákno, ktoré vojde do kritickej oblasti sa môže vzdať vlastníctva monitoru objektu (na ktorý je oblasť kritická) zavolaním `wait()` na tento objekt. Tým pádom sa jeho vykonávanie zablokuje.
- Vrátiť vlákno do stavu, v ktorom sa snaží získať monitor naspäť a obnoviť vykonávanie, môže iné vlákno zavolaním `notify()`, alebo `notifyAll()` na ten istý objekt
- Rozdiel medzi `notify()` a `notifyAll()` spočíva v tom, že `notify()` zobudí len jedno z vlákien čo sa vzdali monitoru, zatiaľ čo `notifyAll()` zobudí všetky

Synchronizácia vlákien

```
synchronized (obj) {
  try {
    obj.wait();
  } catch (InterruptedException e) {
  }
}
```

```
synchronized (obj) {
  try {
    obj.notify();
  } catch (InterruptedException e) {
  }
}
```

Space - singleton

- Space sa implementuje ako objekt vzoru singleton
- jeho konštruktor je `private`
- miesto neho poskytuje statickú metódu `getInstance()`, ktorá vracia jedinú inštanciu objektu, uloženú v statickom `private` atribúte

```
public class Singleton {
  static Singleton instance =
    new Singleton();
  private Singleton () {
    ...
  }
  public static
    Singleton getInstance() {
      return instance;
  }
}
```

Agent

- objekt, ktorý má vlastné vlákno
- objekt, ktorý disponuje určitým komunikačným mechanizmom na výmenu dát s ostatnými agentami
- objekt, ktorého metódy nie sú určené na to aby volal niekto iný než agent

Časovače (timery)

- Časovanie sa v jave opiera o preťaženú verziu `wait()`, ktorej sa ako argument dá definovať timeout v milisekundách
- s jej pomocou java implementuje objekt Timer, ktorý má vlastné vlákno, v ktorom sa blokuje a vhodne púšťa určité kódy
- na ich zavolanie má triedu TimeredTask od ktorej sa odvodí konkrétna úloha prekrytím metódy `public void run()`

Spúšte (Triggery)

- Agent si požiada o notifikáciu udalosti
- Agent zavolá wait()
- Nastane udalosť
- Trigger zavolá notify()

Propagácia udalostí

- na agentovú modularitu na OOP VM sa dá pozerať ako na snahu odstrániť u prijímateľa udalosti potrebu implementovať pre každý typ správy špecifické rozhranie

Propagácia udalostí – tradičná

```
class Listener implements Event1Listener,
  Event2Listener, ..., EventNListener {
  ...
  obj1.addListener(this); ...
  obj2.addListener(this); ...
  objN.addListener(this);
  ...
 ... event1Performed(... arg1 ...) { ... };
 ... event2Performed(... arg2 ...) { ... };
 ...
 ... eventNPerformed(... argN ...) { ... };
}
```

Propagácia udalostí – “agentová”

```
class Listener extends Agent {
  ...
  for (;;) {
    ... event = getEvent(...); // Receive
    switch (event.getType) {
      case ... : event1Performed(...); break;
      case ... : event2Performed(...); break;
      ...
      case ... : eventNPerformed(...); break;
    }
  };
}
```

Propagácia udalostí

- prípade tradičnej propagácie sa spracovanie udalosti udeje vo vlákne, v ktorom udalosť vznikla
- v prípade agentovej propagácie sa vo vlákne v ktorom udalosť vznikla udeje len informovanie príjemcu udalosti a k samotnému spracovaniu dôjde v inom vlákne – vo vlákne príjemcu
- tj. každý príjemca udalosti má vlastné vlákno, v ktorom udalosti spracúva