PV192: Otázky a odpovědi

Lukáš Hejtmánek, Petr Holub

Obsah

1 Obecné otázky

2 C/C++

3 Java

J-1. **Mohl bych se zeptat, kde je v následujícím kódu skryté to nebezpečné publikování?**

prednaska-02.pdf, s. 46:

```
public class NebezpecnePublikovani {
  public class Pytlicek {
    public int hodnota;
    public Pytlicek(int hodnota) {
      this.hodnota = hodnota;
    }
  }

  public Pytlicek pytlicek;

  public void incializujPytlicek(int i) {
    pytlicek = new Pytlicek(i);
  }
}
```

V metodě inicializujPytlicek() se přece zavolá konstruktor Pytlicku a až ten doběhne, tak se publikuje odkaz na pytlicek. Tzn. v poli pytlicek bude null a pak až hotový odkaz. Je to tak? Kde je potom ta publikace nedokončeného objektu? Ve zdroji (JCIP) taky nic nepíšou.

Na první pohled to vypadá nevinně, ale ten průšvih v tom kódu je v následujícím:

```
public Pytlicek pytlicek;
```

Díky tomu, že je ten odkaz public (a tudíž ani přístup k němu není nijak synchronizován), tak v době, kdy si jedno vlákno zavolá `incializujPytlicek(x)`, jiné vlákno může přistupovat k tomu veřejnému odkazu pytlíček a objekt může být v rozpracovaném stavu.

Díky za odpověď, ale pořád tomu asi nerozumím. Co to znamená rozpracovaný stav? Myslel jsem, že se to váže k úniku z konstruktoru. Ale tady jde o to, že to druhé vlákno metodou inicializujPytlcek() vytvoří nový objekt a vlákno první bude pořád pracovat se starým?

Může se stát, že do proměnné `pytlicek`, která je viditelná bez jakékoli ochrany zvenku objektů, už je přiřazeny odkaz na vznikající objekt typu Pytlíček, ale ještě není doběhnutý konstruktor toho objektů. Takže pokud si jiné vlákno sáhne po tom objektů v nevhodný okamžik, tak ho dostane v rozpracovaném stavu obdobně, jako by únik vznikl přímo z konstruktoru.

Problém je tady v tom, že se nevynutí pořadí - prvně dodělej konstruktor - poté vrať referenci což pro normální konzistentní chování programu potřebujete, protože konstruktor běží paralelně a nezávislé vzhledem k

libovolnému jinému vláknu, které by šlo po tom odkazu (public Pytlíček pytlíček).

Abyste to výše uvedené pořadí vynutil, musela by to být jedna z následujících možnosti:

- metoda, která objekt zkonstruuje a pak ho teprve vrátí (t.j. aby uvnitř té metody byla zajištěna ta sekvenční výše uvedena logika)
- schovat odkaz na pytlíček jako privátní a vracet ho přes getter, který ovšem bude synchronized vzhledem k incializujPytlicek

Speciální případy jsou pak ty zmíněné immutable objekty, u kterých se to takhle bezpečně dá dělat (akorát by ten odkaz měl být volatile nebo AtomicReference), protože těm se od JVMka dostává speciálního zacházení.

V této metodě by sekvenčnost musela být zajištěna explicitně?

Záleží na tom, jak přesně by to fungovalo:

- synchronizace není potřeba, pokud by ta metoda jen natvořila objekt a ten vrátila a nikam si ho neukládala uvnitř sebe (tj. vytvořila by ho jen pro volající vlákno, kterému by ho předala jako návratovou hodnotu)
- synchronizace by byla třeba, pokud by to fungovalo tak, že by metoda natvořila ten nový objekt do vnitrhniho stavu objektů (tak jak je to v tom příkladu), protože nechcete, aby se 2 vlákna voláním té metody potenciálně nekonzistentním způsobem pobily o to přiřazení

4 Jiné jazyky