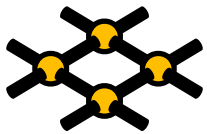
Vláknové programování
část II

Lukáš Hejmánek, Petr Holub

{xhejtman,hopet}@ics.muni.cz



Laboratoř pokročilých síťových technologií

PV192

2011–03–03

Přehled přednášky

Ada

"Regression testing?" What's that? If it compiles, it is good, if it boots up it is perfect.

– *Linus Torvalds*

Compilable but broken code is even worse than working code.

– *Alan Cox*

during a bright moment on the linux-kernel list

Ada:
Ideally, when compiles fine it also runs fine.

Ada

- Ada 83/95/2005
- jazyk orientovaný na spolehlivé aplikace: vojáci, letectví, …
- WORM: write once read many
- silně typovaný jazyk
- podpora paralelismu
- enkapsulace, výjimky, objekty, správa paměti, atd.
- GNAT
  - volně dostupný překladač, frontend k GCC
- materiály na webu
  - (Annotated) Reference Manual
    `http://www.adaic.org/standards/ada05.html`
  - `http://stwww.weizmann.ac.il/g-cs/benari/books/index.html#ase`
  - `http://en.wikibooks.org/wiki/Programming:Ada`
  - `http://www.adahome.com/Tutorials/Lovelace/lovelace.htm`
  - `http://www.pegasoft.ca/resources/boblap/book.html`

Ada: Hello World!

```
1 with Ada.Text_IO;

3 procedure Hello is
  begin
5         Ada.Text_IO.Put_Line ("Hello, world!");
  end Hello;
```

- normální kompilace

  `gnatmake hello.adb`

- přidané kontroly

  `gnatmake -gnatya -gnatyb -gnatyc -gnatye -gnatyf -gnatyi -gnatyk -gnatyl -gnatyn -gnatyp -gnatyr -gnatyt -g -gnato -gnatf -fstack-check hello.adb`

Ada: balíky

- Princip zapouzdření
  - rozdělení funkcionality do balíků (packages)
  - hierarchie balíků: child packages (Ada95)
  - privátní část (balíků, chráněných typů)
- Princip oddělení specifikace a implementace
  - `.ads` popisuje rozhraní
    - lze kompilovat i jen proti specifikaci (bez implementace), ale nelze v takovém případě linkovat
  - `.adb` je implementace
- Doporučené pojmenování souborů podle jmen balíků
  - `s/\./-/g`
- Nejsou požadavky na adresářovou strukturu
- Možnost externalizace implementací balíků

Ada: procedury, funkce

- procedury: `in`, `out` a `in out` parametry
- funkce: pouze `in` parametry, vrací hodnotu

```
procedure Procedura (A, B : in Integer := 0;
                     C : out Unbounded_String;
                     D : in out Natural)
is
begin
        C := To_Unbounded_String(A'Image & " " & B'Image);
        D := D + 1;
end Procedura;

function Funkce (A, B: Integer) return String
is
        Retezec : Unbounded_String;
        Prirozene_Cislo : Natural;
begin
        Procedura (A, B, Retezec, Prirozene_Cislo);
        Procedura (Retezec, Prirozene_Cislo);
        Procedura (B => B, A => A, Retezec, Prirozene_Cislo);
        return To_String (Retezec);
end Funkce;

type Callback_Procedury is access procedure (Id  : Integer; T : String);
type Callback_Function is access function (Id : Natural) return Natural;
```

Ada: balíky

balik.ads:

```
package Balik is

        type Muj_Typ is private;

        procedure Nastav_A (This : in out Muj_Typ;
                            An_A : in     Integer);

        function Dej_A (This : Muj_Typ) return Integer;

private

        type Muj_Typ is
        record
                A : Integer;
        end record ;

        pragma Inline (Dej_A);

end Balik;
```

Ada: balíky

balik.adb:

```
1  package body Balik is

3     procedure Nastav_A (This : in out Muj_Typ;
                          An_A : in     Integer)
5     is
      begin
7        This.A := An_A;
      end Nastav_A;
9
      function Dej_A (This : Muj_Typ) return Integer is
11    begin
         return This.A;
13    end Dej_A;

15 end Balik;
```

Ada: typy

- Základní typy
  - celočíselné, plovoucí, decimální
  - výčty
  - pole
  - záznamy (struktury)
  - ukazatele (access)
  - úlohy (tasks)
  - rozhraní (interfaces)

```
Y : Boolean;
S : String;
F : Float;
Addr : System.Address;
type uhel is delta 1 / (60 * 60) range 0.0 .. 360.0;
type teplomer is delta 10.0**(-2) digits 10 range -273.15 .. 5000.0;
type pole is array (1 .. 10) of  Natural;
X : String (1 .. 10) := (others => ' ');
type Enum is (A, B, C);
E : Enum := A;
```

Ada: typy

- Neomezené řetězce
  - omezené velikostí haldy nebo `Integer'Last`

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;

procedure Strings is
        S : String := "Muj retezec bla";
begin
        declare
                US : Unbounded_String :=
                        To_Unbounded_String (S(S'First .. S'Last-4));
        begin
                Put_Line (To_String (US));
                Put_Line ("To " & "je " & "vse.");
        end;
end Strings;
```

```
$ gnatmake strings.adb
gcc -c strings.adb
gnatbind -x strings.ali
gnatlink strings.ali

$ ./strings
Muj retezec
To je vse.
```

Ada: typy

- Anonymní vs. pojmenované typy

```
X : String (1 .. 10) := (others => ' ');
subtype My_String is String (1 .. 10);
Y : My_String := (others => ' ');
```

- Privátní versus limitované privátní typy

```
type typ_X is private; -- nevidime dovnitr
type typ_X is limited private; -- nelze priradit
```

Ada: typy

- Záznamy / Struktury

```
type Car is record
   Identity        : Long_Long_Integer;
   Number_Wheels   : Positive range 1 .. 10;
   Paint           : Color;
   Horse_Power_kW  : Float range 0.0 .. 2_000.0;
   Consumption     : Float range 0.0 .. 100.0;
end record;

BMW : Car :=
  (Identity        => 2007_752_83992434,
   Number_Wheels   => 5,
   Horse_Power_kW  => 190.0,
   Consumption     => 10.1,
   Paint           => Blue);
```

- Variantní záznamy, uniony

Ada: typy

- Objekty – tagované záznamy

```
1  type Person is tagged
      record
3        Name   : String (1 .. 10);
         Gender : Gender_Type;
5     end record;

7  type Programmer is new Person with
      record
9        Skilled_In : Language_List;
      end record;
11
   function Get_Skills (P : Programmer) return Language_List;
```

Ada: typy

- Typy vs. podtypy

```
type Integer_1 is range 1 .. 10;
subtype Integer_2 is Integer_1 range 7 .. 10;
A : Integer_1 := 8;
B : Integer_2 := A; -- OK

type Integer_1 is range 1 .. 10;
type Integer_2 is new Integer_1 range 2 .. 8;
A : Integer_1 := 8;
B : Integer_2 := A; -- nelze!
C : Integer_2 := Integer_2 (A); -- OK
```

Ada: typy

- Ukazatele

```
type Day_Of_Month is range 1 .. 31;
type Day_Of_Month_Access is access Day_Of_Month;
type Day_Of_Month_Access_All is access all Day_Of_Month;
for Day_Of_Month_Access'Storage_Pool use Pool_Name;
procedure Test (Call_Back: access procedure (Id: Integer; Text: String));

DoM : Day_Of_Month;
DoMA : Day_Of_Month_Access := Dom'Access; -- neee
DoMAA : Day_Of_Month_Access_All := Dom'Access; -- jo

DoMA : Day_Of_Month_Access := new Day_Of_Month;
procedure Free is new Ada.Unchecked_Deallocation
        (Object => Day_Of_Month
        Name   => Day_Of_Month_Access);
Free (DoMA);
```

Ada: typy

- Atributy

```
type barvy is (cervena, zelena, modra);
barvy'Pos(cervena) -- = 0
barvy'Val(0) -- = cervena
type pole is array (1 .. 10) of  Natural;
pole'First -- = 1
pole'Last -- = 10
pole'Range -- = 1 .. 10
type Byte is range -128 .. 127;
for Byte'Size use 8;
Byte'Min -- = -128
Byte'Max -- = 127
F : Float;
F'Ceiling;
F'Floor;
F'Rounding;
F'Image -- Float -> String
Float'Val -- String -> Float
```

Ada: řídící struktury

```
1  if condition then
           statement;
3  else
           other statement;
5  end if;

7  case X is
           when 1 =>
9                 Walk_The_Dog;
           when 5 =>
11                Launch_Nuke;
           when 8 | 10 =>
13                Sell_All_Stock;
           when others =>
15                Self_Destruct;
   end case;
17
   Until_Loop :
19 loop
           X := Calculate_Something;
21         exit Until_Loop when X > 5;
   end loop Until_Loop;
23
   for I in X'Range loop
25         X (I) := Get_Next_Element;
   end loop;
```

Ada: výjimky

```
package body Directory_Enquiries is

   procedure Insert (New_Name    : in Name;
                     New_Number  : in Number)
   is
   begin
      if New_Name = Old_Entry.A_Name then
         raise Name_Duplicated;
      end if;
      New_Entry :=  new Dir_Node'(New_Name, New_Number,);
   exception
      when Storage_Error => raise Directory_Full;
   end Insert;

   procedure Lookup (Given_Name  : in  Name;
                     Corr_Number : out Number)
   is
   begin
      if not Found then
         raise Name_Absent;
      end if;
   end Lookup;

end Directory_Enquiries;
```

Ada: generické typy

```
1  generic
     type Typ_Prvku is private;
3    with function "*" (X, Y: Typ_Prvku) return Typ_Prvku;
   function Umocni (X : Typ_Prvku) return Typ_Prvku;
5
   function Umocni (X: Typ_Prvku) return Typ_Prvku is
7  begin
     return X * X;
9  end Umocni;

11 function Umocni_Mnozinu is new Umocni
           (Typ_Prvku => Mnozina.Prvek, "*" => Mocneni);
```

Ada: konteinery

- Implementované pomocí generik
- Typy
  - vektory
  - dvojite spojene seznamy
  - mapy
  - mnoziny

Ada: konteinery

```
with Ada.Containers.Ordered_Sets;
with Ada.Containers.Hashed_Sets;
with Ada.Containers.Hashed_Maps;

package Priklad_Kolekci is

        function Unbounded_String_Hash (S : Unbounded_String)
        return Hash_Type;

        package UString_Int_Hash is new Ada.Containers.Hashed_Maps (
                        Key_Type => Unbounded_String,
                        Element_Type => Integer,
                        Hash => Unbounded_String_Hash,
                        Equivalent_Keys => "=");

        package UStringSet is new Ada.Containers.Hashed_Sets (
                        Element_Type => Unbounded_String,
                        Hash => Unbounded_String_Hash,
                        Equivalent_Elements => "=");
```

Ada: konteinery

```
1       procedure Output_Host_Set (HostSet : UStringSet.Set) is
            use UStringSet;
3           Cur : UStringSet.Cursor;
            Is_Tag_Open : Boolean := False;
5       begin
            Cur := First (HostSet);
7           if Cur /= UStringSet.No_Element then
                Open_Tag (Is_Tag_Open, "      <hosts>");
9               while Cur /= UStringSet.No_Element loop
                    Put_Line ("          <host>" & To
11  _String (Element (Cur)) & "</host>");
                    Cur := Next (Cur);
13              end loop;
                Close_Tag (Is_Tag_Open, "      </hosts>"
15  );
            end if;
17
        exception
19          when An_Exception : others =>
                Syslog (LOG_ERR, "State dump failed! " &
21              & Exception_Information (An_Exception));
                Close_Tag (Is_Tag_Open, "      </hosts>"
23  );
        end Output_Host_Set;
25
    end Priklad_Kolekci
```

Instalace GNATu

- Instalační soubory jsou ve studijních materiálech
  - **`/el/1433/jaro2011/PV192/um/23046574/`**
  - Linux x86 32 b
  - Linux x86 64 b
  - Windows
- Součásti:

GNAT GPL Ada	Ada compiler, debugger, tools, libraries
AUnit gpl-2010	Ada unit testing framework
SPARK GPL	SPARK Examiner and Simplifier (static prover)
GNATbench	Ada plugins for the Eclipse/Workbench IDEs
PolyORB gpl-2010	CORBA and Ada distributed annex
AWS gpl-2.8.0	Ada web server library
ASIS gpl-2010	Library to create tools for Ada software
AJIS gpl-2010	Ada Java interfacing suite
GNATcoll gpl-2010	Reusable Ada components library
GtkAda gpl-2.14.1	Create modern native GUIs in Ada
XMLAda gpl-3.2.1	Library to process XML streams
Florist gpl-2010	Interface to POSIX libraries (pouze pro Linux)
Win32 Ada gpl-2010	Ada API to the Windows library (pouze pro Windows)

Instalace GNATu

- Instalace překladače
- Linux:
  - adacore-X86LNX-201102271805.zip:
    `x86-linux/gnatgpl-gpl-2010/`
    `gnat-gpl-2010-i686-gnu-linux-libc2.3-bin.tar.gz`
    `gnat-2010-i686-gnu-linux-libc2.3-bin`
    `./doinstall`
- Windows:
  - adacore-X86WIN-201102271212.zip:
    `x86-windows/gnatgpl-gpl-2010/`
    `gnat-gpl-2010-1-i686-pc-mingw32-bin.exe`

Přehled přednášky

Vlákna v jazyce Java

- Mechanismy
  - potomek třídy Thread
  - Executory
  - objekt implementující úlohu pro Executor
    - objekt implementující interface Runnable (metoda `run()`)
    - objekt implementující interface Callable (metoda `call()`)
- Synchronizace a viditelnost operací
- Implementace monitorů pomocí `synchronized`
- Signalizace mezi objekty: `wait`, `notify`, `notifyAll`
- Knihovny `java.util.concurrent`

Třída Thread

- Základní třída pro vlákna
- Metoda `run()`
  - „vnitřnosti“ vlákna
  - přepisuje se vlastním kódem
- Metoda `start()`
  - startování vlákna
  - za normálních okolností nepřepisuje!
  - pokud už se přepisuje, je třeba volat `super.start()`

Třída Thread

```
1  public class PrikladVlakna {
2
3      static class MojeVlakno extends Thread {
4          MojeVlakno(String jmenoVlakna) {
5              super(jmenoVlakna);
6          }
7
8          public void run() {
9              for (int i = 0; i < 10; i++) {
10                 System.out.println(this.getName() +
11                         ": pocitam vzbuzeni - " + (i + 1));
12                 try {
13                     sleep(Math.round(Math.random()));
14                 } catch (InterruptedException e) {
15                     System.out.println(this.getName() +
16                             ": probudil jsem se nenadale! :-|");
17                 }
18             }
19         }
20     }
21
22     public static void main(String[] args) {
23         new MojeVlakno("vlakno1").start();
24         new MojeVlakno("vlakno2").start();
25     }
}
```

Viditelnost a synchronizace operací

(nic) > `volatile` > `AtomicXXX` > `synchronized`, explicitní zámky

- problém viditelnosti změn
- problém atomicity operací
  - např. přiřazení do 64-bitového typu (`long`, `double`) není nutně atomický!
- problém synchronizace při změnách hodnot

Viditelnost a synchronizace operací

(nic) > `volatile` > `AtomicXXX` > `synchronized`, explicitní zámky

```
public class Nic {
    private static long cislo = 10000000L;
    private static boolean pripraven = false;

    public static class Vlakno extends Thread {
        public void run() {
            while (!pripraven) {
                Thread.yield();
            }
            System.out.println(cislo);
        }
    }

    public static void main(String[] args) {
        new Vlakno().start();
        cislo = 42L;
        pripraven = true;
    }
}
```

Viditelnost a synchronizace operací

(nic) > `volatile` > `AtomicXXX` > `synchronized`, explicitní zámky

- Jak to dopadne?
  - neatomičnost 64-bitového přiřazení
  - přeuspořádání přiřazení
  - kterákoli ze změn hodnot nemusí být viditelná
  - *jakkoli…*
    - 10.000.000 nebo 42 nebo něco jiného (neatomičnost – vlákno se může trefit mezi přiřazení horní a dolní poloviny 64 b operace)
    - ale také může navždy cyklit (Vlakno neuvidí nastavení pripraven)

pročež platí

1. Pokud více vláken čte jednu proměnnou, musí se řešit viditelnost.
2. Pokud více vláken zapisuje do jedné proměnné, musí se synchronizovat.

Viditelnost a synchronizace operací

(nic) > `volatile` > `AtomicXXX` > `synchronized`, explicitní zámky

- `volatile`
  - zajišťuje viditelnost změn mezi vlákny
  - překladač nesmí dělat presumpce/optimalizace, které by mohly ovlivnit viditelnost
  - u 64 b přiřazení zajišťuje atomičnost
  - nezajišťuje atomičnost operace načti–změň–zapiš!
    - nelze použít pro thread-safe `i++`
  - lze použít pokud jsou splněny obě následující podmínky
    1. nová hodnota proměnné nezávisí na její předchozí hodnotě
    2. proměnná se nevyskytuje v invariantech spolu s jinými proměnnými (např. `a<=b`)

    - např. příznak ukončení nebo jiné události, který nastavuje pouze jedno vlákno – pomohlo by v příkladě třídy Nic (slajd 31)
    - příklady použití: `http://www.ibm.com/developerworks/java/library/j-jtp06197.html`
  - pokud si nejsme jisti, použijeme raději silnější synchronizaci

Viditelnost a synchronizace operací

(nic) > `volatile` > `AtomicXXX` > `synchronized`, explicitní zámky

```
public class VolatileInvariant {
    private volatile int horniMez, dolniMez;

    public int getHorniMez() { return horniMez; }

    public void setHorniMez(int horniMez) {
        if (horniMez < this.dolniMez)
            throw new IllegalArgumentException("Horni < dolni!");
        else
            this.horniMez = horniMez;
    }

    public int getDolniMez() { return dolniMez; }

    public void setDolniMez(int dolniMez) {
        if (dolniMez > this.horniMez)
            throw new IllegalArgumentException("Dolni > horni!");
        else
            this.dolniMez = dolniMez;
    }
}
```

Viditelnost a synchronizace operací

(nic) > `volatile` > `AtomicXXX` > `synchronized`, explicitní zámky

- `AtomicXXX`
  - zajišťuje viditelnost
  - zajišťuje atomičnost operace načti–změň–zapiš nad objektem
    - potřebujeme-li udělat více takových operací synchronně, nelze použít
- `synchronized`, explicitní zámky
  - zajištění viditelnosti
  - zajištění vyloučení (a tedy i atomičnosti) v kritické sekci

Publikování a únik

- Publikování objektu
  - zveřejnění odkazu na objekt
  - thread-safe třídy nesmí publikovat objekty bez zajištěné synchronizace
  - nepřímé publikování
    - publikování jiného objektu, přes něhož je daný objekt *dosažitelný*
    - např. publikování kolekce obsahující objekt
    - např. instance vnitřní třídy obsahuje odkaz na vnější třídu
  - „vetřelecké“ (*alien*) metody
    - metody, jejichž chování není plně definováno samotnou třídou
    - všechny metody, které nejsou **`private`** nebo **`final`** – mohou být přepsány v potomkovi
    - předání objektu vetřelecké metodě = publikování objektu

Publikování a únik

- Únik stavu objektu
  - publikování reference na interní měnitelné (*mutable*) objekty
  - v níže uvedeném příkladě může klient měnit pole **`states`**
  - potřeba hlubokého kopírování (*deep copy*)

```
1  import java.util.Arrays;

3  public class UnikStavu {
       private String[] stavy = new String[]{"Stav1", "Stav2", "Stav3"};
5
       public String[] getStavySpatne() {
7          return stavy;
       }
9
       public String[] getStavySpravne() {
11         return Arrays.copyOf(stavy, stavy.length);
       }
13 }
```

Publikování a únik

- Únik z konstruktoru
  - až do návratu z konstruktoru je objekt v „rozpracovaném“ stavu
  - publikování objektu v tomto stavu je obzvláště nebezpečné
  - pozor na skryté publikování přes `this` v rámci instance vnitřní třídy
    - registrace listenerů na události

```
1  public class UnikZKonstruktoru {
       public UnikZKonstruktoru(EventSource zdroj) {
3          zdroj.registerListener(
                   new EventListener() {
5                      public void onEvent(Event e) {
                           zpracujUdalost(e);
7                      }
                   }
9          );
       }
11 }
```

Publikování a únik

- Únik z konstruktoru
  - když musíš, tak musíš… ale aspoň takto:
    1. vytvořit soukromý konstruktor
    2. vytvořit veřejnou factory

```
1  public class BezpecnyListener {
       private final EventListener listener;
3
       private BezpecnyListener() {
5          listener = new EventListener() {
               public void onEvent(Event e) {
7                  zpracujUdalost(e);
               }
9          };
       }
11
       public static BezpecnyListener novaInstance(EventSource zdroj) {
13         BezpecnyListener bl = new BezpecnyListener();
           zdroj.registerListener(bl.listener);
15         return bl;
       }
17 }
```

Thead-safe data

- *ad hoc*
  - zodpovědnost čistě ponechaná na implementaci
  - nepoužívá podporu jazyka
  - pokud možno nepoužívat

Thead-safe data

- data omezená na zásobník
  - data na zásobníku patří pouze danému vláknu
  - týká se lokálních proměnných
    - u primitivních lokálních proměnných nelze získat ukazatel a tudíž je nelze publikovat mimo vlákno
    - ukazatele na objekty je třeba hlídat (programátor), že se objekt nepublikuje a zůstává lokální
    - lze používat ne-thread-safe objekty, ale je rozumné to dokumentovat (pro následné udržovatele kódu)

```
1  import java.util.Collection;

3  public class PocitejKulicky {
       public class Kulicka {
5      }

7      public int pocetKulicek(Collection<Kulicka> kulicky) {
           int pocet = 0;
9          for (Kulicka kulicka : kulicky) {
               pocet++;
11         }
           return pocet;
13     }
   }
```

Thead-safe data

- **`ThreadLocal`**
  - data asociovaná s každým vláknem zvlášť, ukládá se do Thread
  - používá se často v kombinaci se Singletony a globálními proměnnými
  - JDBC spojení na databázi nemusí být thread-safe

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class PrikladTL {
    private static ThreadLocal<Connection> connectionHolder =
            new ThreadLocal<Connection>() {
                protected Connection initialValue() {
                    try {
                        return DriverManager.getConnection("DB_URL");
                    } catch (SQLException e) {
                        return null;
                    }
                }
            };

    public static Connection getConnection() {
        return connectionHolder.get();
    }
}
```

Thead-safe data

- Neměnné (*immutable*) objekty
  - neměnný objekt je takový
    - jehož stav se nemůže změnit, jakmile je zkonstruován
    - všechny jeho pole jsou `final`
    - je řádně zkonstruován (nedojde k úniku z konstruktoru)
  - neměnné objekty jsou automaticky thread-safe
  - pokud potřebujeme provést složenou akci atomicky, můžeme ji zabalit do vytvoření neměnného objektu na zásobníku a jeho publikaci pomocí `volatile` odkazu
    - nemůžeme předpokládat atomičnost načti–změň–zapiš (`i++` chování)
  - díky levné alokaci[1] nových objektů (JDK verze 5 a výš) se dají efektivně používat

[1] Do JDK 5.0 se používalo `ThreadLocal` pro recyklaci bufferů metody `Integer.toString`. Od verze 5.0 se vždy alokuje nový buffer, je to rychlejší.

Thead-safe data

- Neměnné (*immutable*) objekty

```
public class NemennaCache {
    private final String lastURL;
    private final String lastContent;

    public NemennaCache(String lastURL, String lastContent) {
        this.lastURL = lastURL;
        this.lastContent = lastContent;
    }

    public String vemZCache(String url) {
        if (lastURL == null || !lastURL.equals(url))
            return null;
        else
            return lastContent;
    }
}
```

Thead-safe data

- Neměnné (*immutable*) objekty

```
public class PouzitiCache {
    private volatile NemennaCache cache = new NemennaCache(null, null);

    public String nactiURL(String URL) {
        String obsah = cache.vemZCache(URL);
        if (obsah == null) {
            obsah = fetch(URL);
            cache = new NemennaCache(URL, obsah);
            return obsah;
        } else
            return obsah;
    }
}
```

Bezpečné publikování

- Nebezpečné publikování
  - publikování potenciálně nedokončeného měnitelného objektu
  - takto by šlo publikovat pouze neměnné objekty (lépe s použitím `volatile`)

```
1  public class NebezpecnePublikovani {
       public class Pytlicek {
3          public int hodnota;

5          public Pytlicek(int hodnota) {
               this.hodnota = hodnota;
7          }
       }
9
       public Pytlicek pytlicek;
11
       public void incializujPytlicek(int i) {
13         pytlicek = new Pytlicek(i);
       }
15 }
```

Bezpečné publikování

- Způsoby bezpečného publikování měnitelných objektů
  1. inicializace odkazu ze statického inicializátoru
  2. uložení odkazu do `volatile` nebo `AtomicReference` pole
  3. uložení odkazu do `final` pole – bezpečné až po návratu z konstruktoru
  4. uložení odkazu do pole, které je chráněno zámky/monitorem
  5. uložení do thread-safe kolekce (`Hashtable`, `synchronizedMap`, `ConcurrentMap`, `Vector`, `CopyOnWriteArray{List,Set}`, `synchronized{List,Set}`, `BlockingQueue`, `ConcurrentLinkedQueue`)
     - objekt i odkaz musí být publikovány současně

Bezpečné publikování

- Efektivně neměnné objekty
  - pokud se k objektu chováme jako neměnnému
  - bezpečné publikování je dostatečné
- Měnitelné objetky
  - bezpečné publikování zajistí pouze viditelnost ve výchozím stavu
  - změny je třeba synchronizovat (zámky/monitory)

Bezpečné sdílení objektů

- Uzavřené ve vlákně
- Sdílené jen pro čtení
  - neměnné a efektivně neměnné objekty
- Thread-safe objekty
  - zajišťují si synchronizaci uvnitř samy
- Chráněné objekty
  - zabalené do thread-safe konstrukcí (thread-safe objektů, chráněny zámkem/monitorem)

Ukončování vláken

- Vlákna by se měla zásadně ukončovat dobrovolně a samostatně
  - metoda `Thread.stop()` je deprecated
  - násilné ukončení vlákna může nechat systém v nekonzistentním stavu
    - výjimka `ThreadDeath` tiše odemkne všechny monitory, které vlákno drželo
    - objekty, které byly monitory chráněny, mohou být v nekonzistentním stavu
  - `http://java.sun.com/j2se/1.4.2/docs/guide/misc/threadPrimitiveDeprecation.html`
- Jak na to?
  - zavést si proměnnou, která bude signalizovat požadavek na ukončení nebo
  - využít příznak `isInterrupted()`
  - použití metody `interrupt()`
  - použití I/O blokujících omezenou dobu
- Vlákna lze násilně ukončovat ve speciálních případech
  - ExecutorService
  - Futures

Ukončování vláken

```
1  import static java.lang.Thread.sleep;

3  public class PrikladUkonceni {
       static class MojeVlakno extends Thread {
5          private volatile boolean ukonciSe = false;

7          public void run() {
               while (!ukonciSe) {
9                  try {
                       System.out.println("...chrnim...");
11                     sleep(1000);
                   } catch (InterruptedException e) {
13                     System.out.println("Vzbudil jsem se necekane!");
                   }
15             }
           }
17
           public void skonci() {
19             ukonciSe = true;
           }
21     }
```

Ukončování vláken

```
    public static void main(String[] args) {
24      try {
            MojeVlakno vlakno = new MojeVlakno();
26          vlakno.start();
            sleep(2000);
28          vlakno.skonci();
            vlakno.interrupt();
30          vlakno.join();
        } catch (InterruptedException e) {
32          e.printStackTrace();
        }
34  }
}
```