



```
long promenna = 10000000L;
```




Paměťový model Javy

```
2 void innerSet(V v) {  
3     for (;;) {  
4         int s = getState();  
5         if (ranOrCancelled(s))  
6             return;  
7         if (compareAndSetState(s, RAN))  
8             break;  
9     }  
10    result = v;  
11    releaseShared(0);  
12    done();  
13 }
```

```
2 V innerGet() throws InterruptedException JavaDoc, ExecutionException JavaDoc {  
3     acquireSharedInterruptibly(0);  
4     if (getState() == CANCELLED)  
5         throw new CancellationException JavaDoc();  
6     if (exception != null)  
7         throw new ExecutionException JavaDoc(exception);  
8     return result;  
9 }
```



Paměťový model Javy

```
1      protected int tryAcquireShared(int ignore) {
2          return innerIsDone()? 1 : -1;
3      }
4
5      /**
6       * Implements AQS base release to always signal after setting
7       * final done status by nulling runner thread.
8       */
9      protected boolean tryReleaseShared(int ignore) {
10         runner = null;
11         return true;
12     }
13
14     boolean innerIsCancelled() {
15         return getState() == CANCELLED;
16     }
17
18     boolean innerIsDone() {
19         return ranOrCancelled(getState()) && runner == null;
20     }
```


Rodiny entries

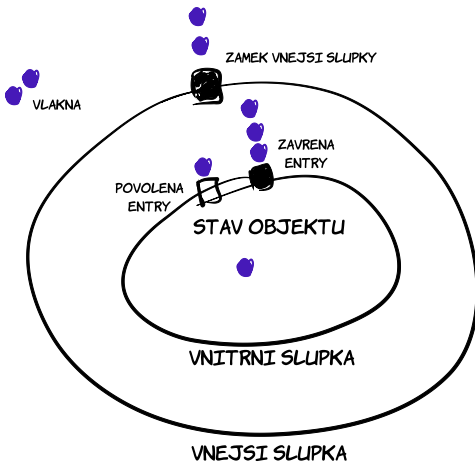
- Příklad řízení zdrojů

```
1 type Request_Range is range 1 .. Max;
3 protected Resource_Controller is
   entry Allocate(Request_Range) (R : out Resource);
5   procedure Release(R : Resource; Amount : Request_Range);
   private
7     Free : Request_Range := Request_Range'Last;
   end Resource_Controller;
9
11 protected body Resource_Controller is
   entry Allocate(for F in Request_Range) (R : out Resource)
13     when F <= Free is
   begin
15     Free := Free - F;
   end Allocate;
   procedure Release(R : Resource; Amount : Request_Range) is
17     begin
   Free := Free + Amount;
19     end Release;
   end Resource_Controller;
```

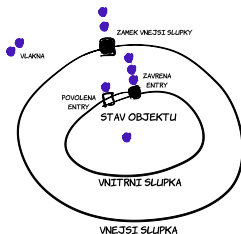
Rodiny entries

- Příklad řízení zdrojů
- Problémy
 - nevhodné pro větší množství alokovatelných zdrojů v jednom požadavku (m)
 - v případě soutěžení je výběr náhodný (prioritu lze nastavovat v rámci Real-Time Systems Annex)

„Eggshell“ model volání chráněného objektu

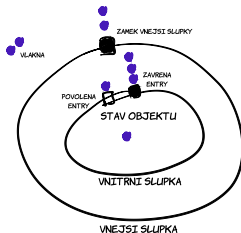


„Eggshell“ model volání chráněného objektu



- Přednost entry calls čekajících ve frontě
- Zámek vnější slupky
 - nedovolí vstup do slupky, pokud je jiné vlákno aktivní voláním procedury nebo entry
 - po vstupu se vyhodnocuje asociovaná podmínka (stráž)
 - tento mechanismus zajišťuje bezpečnost použití 'count'

„Eggshell“ model volání chráněného objektu



- Vnitřní slupka
 - po výstupu z každé procedury a entry se vyhodnocují podmínky a eventuálně se propouští volání čekající na vnitřní slupce

requeue

- Jak poslat za dveře někoho, koho už jsme si pustili do místnosti?
- `requeue` umožňuje vysunout aktivní vlákno v chráněném objektu do fronty před vnitřní slupku
 - nové volání musí mít stejnou signaturu
 - implicitně není přerušitelné pomocí `abort`, aby objekt nezůstal v rozpracovaném stavu
 - nové volání může být přerušitelné `requeue with abort`
- `requeue` umí fungovat i napříč více chráněnými objekty/úlohami
 - neobvyklé, používat opatrně

requeue

```
1 type Request_Range is range 1 .. Max;
2
3 protected Resource_Controller is
4   entry Allocate(R : out Resource; Amount : Request_Range);
5   procedure Release(R : Resource; Amount : Request_Range);
6 private
7   entry Assign(R : out Resource; Amount : Request_Range);
8   Free : Request_Range := Request_Range'Last;
9   New_Resources_Released : Boolean := False;
10  To_Try : Natural := 0;
11  ...
12 end Resource_Controller;
13
14 protected body Resource_Controller is
15   entry Allocate(R : out Resource; Amount : Request_Range)
16     when Free > 0 is
17     begin
18       if Amount <= Free then
19         Free := Free - Amount;
20         -- allocate
21       else
22         requeue Assign;
23       end if;
24     end Allocate;
```

requeue

```
2  entry Assign(R : out Resource; Amount : Request_Range)
   when New_Resources_Released is
   begin
4     To_Try := To_Try - 1;
   if To_Try = 0 then
6     New_Resources_Released := False;
   end if;
8     if Amount <= Free then
       Free := Free - Amount;
10    -- allocate
   else
12     requeue Assign;
   end if;
14  end Assign;
```

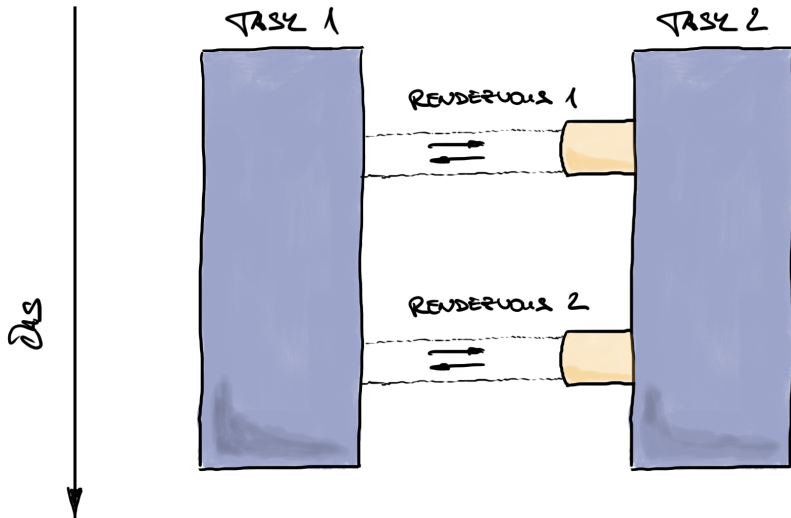
requeue

```
2  procedure Release(R : Resource; Amount : Request_Range) is
3  begin
4      Free := Free + Amount;
5      -- free resources
6      if Assign'Count > 0 then
7          To_Try := Assign'Count;
8          New_Resources_Released := True;
9      end if;
10     end Release;
11 end Resource_Controller;
```

Ada: Tasks, Rendezvous

- Koncept CSP: Communicating Sequential Processes
 - Hoare, 1978
 - paralelně běžící sekvenční procesy
 - komunikace: zasílání zpráv
 - synchronizace: synchronní zasílání zpráv
 - ◆ odesílatel se zablokuje, dokud příjemce není schopen přijmout zprávu
 - ◆ příjemce se zablokuje, dokud není schopen od odesílatele přijmout zprávu

Tasks, Rendezvous



Tasks

- **task**
 - lokálně definované
 - ◆ běží od začátku rozsahu, v němž jsou definované
 - dynamicky alokované
 - ◆ **access** typ
 - ◆ alokace pomocí **new**
 - ◆ běží až od alokace
 - pole tasků
- ukončování
 - spontánní
 - **abort**

Tasks

```
1 task T is
2 end T;

4 task body T is
5 begin
6     makam;
7 end T;

8
9 task type T_Type is
10 end T;

11
12 task body T_Type is
13 begin
14     loop
15         makam;
16     end loop;
17 end T_Type;

18
19 Pole_T : array (1..10) of T_Type;

20
21 type T_Type_Access is access T_Type;
22 Dynamicky_T : T_Type_Access;
23 Dynamicky_T := new T_Type;
```

Parametrizace vláken

- předávání parametrů při vzniku vlákna
- užitečné s typy vláken

```
1 type Monitor_Procedure_Type is access procedure;  
3 task type Monitor_Task_Type (Mon_Proc : Monitor_Procedure_Type) is  
   entry Run;  
   entry Stop;  
   entry Request_Terminate;  
7 end Monitor_Task_Type;
```

Parametrizace vláken

```
1 task body Monitor_Task_Type is
2     Finish_Flag : Boolean := False;
3     Terminate_Flag : Boolean := False;
4 begin
5     while not Terminate_Flag
6     loop
7         select
8             accept Run;
9             while not (Finish_Flag or Terminate_Flag)
10            loop
11                select
12                    accept Stop do
13                        Finish_Flag := True;
14                    end Stop;
15                else
16                    Mon_Proc.all;
17                end select;
18            end loop;
19        or
20            accept Request_Terminate do
21                Terminate_Flag := True;
22            end Request_Terminate;
23        end select;
24        Finish_Flag := False;
25    end loop;
26 end Monitor_Task_Type;
```

Rendezvous

- místa synchronizace – předávání dat
- **entry**
 - deklarace rendezvous bodu
 - `in, out, in out` parametry
- **accept**
 - implementace v těle tasku

Tasks, Rendezvous

```
1 procedure Task1 is
2
3     task Vlakno is
4         entry ZadejX (X : in Integer);
5         entry PrectiX (X : out Integer);
6     end Vlakno;
7
8     task body Vlakno is
9         Hodnota : Integer;
10    begin
11        accept ZadejX (X : in Integer) do
12            Hodnota := X;
13        end ZadejX;
14        Hodnota := Hodnota + 1;
15        accept PrectiX (X : out Integer) do
16            X := Hodnota;
17        end PrectiX;
18    end Vlakno;
19
20    Chci_Inkrementovat : Integer;
21
22    begin
23        Vlakno.ZadejX(Chci_Inkrementovat);
24        Vlakno.PrectiX(Chci_Inkrementovat);
25    end Task1;
```

Rendezvous

- **select**
 - výběr z více **accept** možností

```
1 task body T is
  begin
3     loop
        select
5         accept Rande1 do
            neco;
7         end Rande1;
        or
9         accept Rande2 do
            neco;
11        end Rande2;
            neco;
13        accept Rande3 do
            neco;
15        end Rande3;
        or
17        terminate;
    end loop;
19 end T;
```

Rendezvous

- **select**

- časovaný výběr

```
1 task body T is
  begin
3     loop
        select
5         accept Rande1 do
            neco;
7         end Rande1;
        or
9         delay 10.0;
            taky_neco;
11        end select;
    end loop;
13 end T;
```

Rendezvous

- **select**

- časovaný výběr

```
1 task body T is
  begin
3     loop
      select
5         accept Randel1 do
            neco;
7         end Randel1;
      else -- ekvivalent "or delay 0.0"
9         null; -- busy waiting
      end select;
11    end loop;
  end T;
```


Rendezvous

- výjimky během rendezvous
 - jsou doručeny jak volajícímu vláknu, tak i vlastnímu vláknu
 - pokud výjimka není ošetřena ve vláknu, je vlákno ukončeno a výjimka se nepropaguje do rodiče (považováno za příliš disruptivní)

```
begin
2   select
      accept Volani do
4       ...
      raise Chyba;
6       ...
      end Volani;
8   end select;
exception
10  when Chyba =>
      Naprav_Stav;
12  when other =>
      Oznam_Uzivateli;
14 end;
```

Rendezvous

- výjimky během rendezvous
 - jsou doručeny jak volajícímu vláknu, tak i vlastnímu vláknu
 - pokud výjimka není ošetřena ve vláknu, je vlákno ukončeno a výjimka se nepropaguje do rodiče (považováno za příliš disruptivní)

```
2  --  
4  --  
4  begin  
   T.Volani;  
6  exception  
   when Chyba =>  
8     Oznam_Uzivateli;  
   end;
```

Vnořené Rendezvous

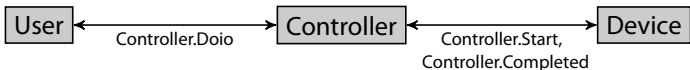
- možnost vícecestné synchronizace

```
1 procedure Three_Way is
2   task User;
3   task Device;
4
5   task Controller is
6     entry Doio (I : out Integer);
7     entry Start;
8     entry Completed (K : Integer);
9   end Controller;
10
11  task body User is ...;
12    -- includes calls to Controller.Doio(...)
13  task body Device is
14    J : Integer;
15    procedure Read (I : out Integer) is ...;
16  begin
17    loop
18      Controller.Start;
19      Read(J);
20      Controller.Completed(J);
21    end loop;
22  end Device;
```

Vnořené Rendezvous

- možnost vícecestné synchronizace

```
2  task body Controller is
    begin
        loop
4     accept Doio (I : out Integer) do
            accept Start;
6         accept Completed (K : Integer) do
                I := K;
8             end Completed;
            end Doio;
10        end loop;
        end Controller;
12 begin
        null;
14 end Three_Way;
```



Chráněné entries u vláken

- podobně jako u chráněných objektů s mírně odlišnou syntaxí

```
task body Ukazka is
2   Zinicializovano : Boolean := False;
   Hodnota : Data;
4  begin
   loop
6     select
       when Zinicializovano =>
8       accept Cti (H : out Data) do
           H := Hodnota;
10      end;
       or
12      accept Zapis (H : in Data) do
           Hodnota := H;
14      end;
           Zinicializovano := True;
16     end select;
   end loop;
18 end Ukazka;
```

- podmínka se vyhodnocuje při každém průchodu přes `select`
- pokud není žádná podmínka splněna, je vyhozena výjimka `Program_Error` (možno použít strukturu `select ... else ... end select;`)
- změna hodnot mezi testem a rendezvous (viz komentář u `'Count`)

Atributy vláken

- Ada 95 Quality and Style Guide, Section 6.2.3
- 'Terminated
 - bezpečné pouze testování na `True` (po ukončení vlákno nemůže obživnout)
- 'Callable
 - bezpečné pouze testování na `False` (po ukončení vlákno nemůže obživnout)

Atributy vláken

- Ada 95 Quality and Style Guide, Section 6.2.3
- 'Count
 - chování vlákna by nemělo záviset na tomto atributu (používat raději jen s chráněnými objekty)

```
select
2 when Transmit'Count > 0 and Receive'Count = 0 =>
  accept Transmit;
4 ...
or
6 accept Receive;
  ...
8 end select;
```



stav 'count se může změnit mezi vyhodnocením a následnou akcí (např. volající použil časově omezené volání a mezi testem a `accept` se ukončil)

- u chráněných objektů: každá práce s frontou je chráněná (viz eggshell model)

Chráněné entries s timeoutem

- Nelze implementovat pomocí chráněných typů, jsou třeba vlákna

```
task body Ukazka is
2   Zinicializovano : Boolean := False;
   Hodnota : Data;
4   begin
   loop
6     select
       when Zinicializovano =>
8       accept Cti (H : out Data) do
           H := Hodnota;
10      end;
       or
12      delay 1.0;
           -- neco
14     end select;
   end loop;
16 end Ukazka;
```


Asynchronous Transfer of Control

- Potřeba *rychle* reagovat na asynchronní události
 - reakce na chyby (např. výpadek HW, kvůli němuž se akce nikdy nedokončí)
 - změny režimů v důsledku (neočekávaných) událostí
 - dosažení co nejlepšího výsledku v případě iterativního přerušitelného výpočtu
 - přerušení uživatelem

Asynchronous Transfer of Control

- Struktura

```
select
2   -- triggering_statement
   delay 5.0;
4   -- post_trigger_part
   Put_Line ("Tudy cesta nevede!");
6 then abort
   -- abortable_part
8   Prevelevelmidlouhe_Volani;
end select;
```

- pokud `abortable_part` doběhne dříve než `triggering_statement`, pokusí se ukončit `triggering_statement`
- pokud `triggering_statement` doběhne dřív než `abortable_part`, je `abortable_part` ukončena a provede se část `post_trigger_part`
- `triggering_statement` – v Ada 95 `delay/entry`, v Ada 2005 i procedury
- `abortable_part` nemusí být implementována jako samostatný task

Asynchronous Transfer of Control

- Příklad: iterativní dlouhý výpočet, chceme nejlepší odhad v době, kdy jej potřebujeme (J. Barnes, Ada 95)
 - chráněný objekt na předávání posledního výsledku

```
1 begin Result is
    procedure Set_Estimate(X : in Data);
3    function Get_Estimate return Data;
    private
5    Est : Data;
end;
```

- signalizační objekt (např. uživatel chce výsledek)

```
begin Trigger is
2    entry Wait;
    -- when Flag
4    procedure Signal;
    private
6    Flag : Boolean := False;
end;
```

Asynchronous Transfer of Control

- Příklad: iterativní dlouhý výpočet, chceme nejlepší odhad v době, kdy jej potřebujeme (J. Barnes, Ada 95)

- použití

```
1 select
    Trigger.Wait;
3 then abort
    Computation;
5 end select;
```

```
1 Trigger.Signal;
E := Result.Get_Estimate;
```

- oddělení logiky výpočtu od jeho ukončování – výpočet nemusí zjišťovat, kdy má končit

Asynchronous Transfer of Control

- Výjimky při ATC
 - pokud se odehraje jen jedna výjimka (v jedné z částí), je možno ji zachytit
 - pokud se odehrají dvě výjimky současně v řídicí i přerušitelné části, je výjimka z přerušitelné části ztracena

