Lec09 Tasks

Thursday, November 28, 2019 12:47 AM

Lec09

Tasks can be thought as an application that tuns concurrently with the main application.

在其他语言中也叫thread

Task 可以synchronize(同步的) with the main application 也可以process information independent from the main application

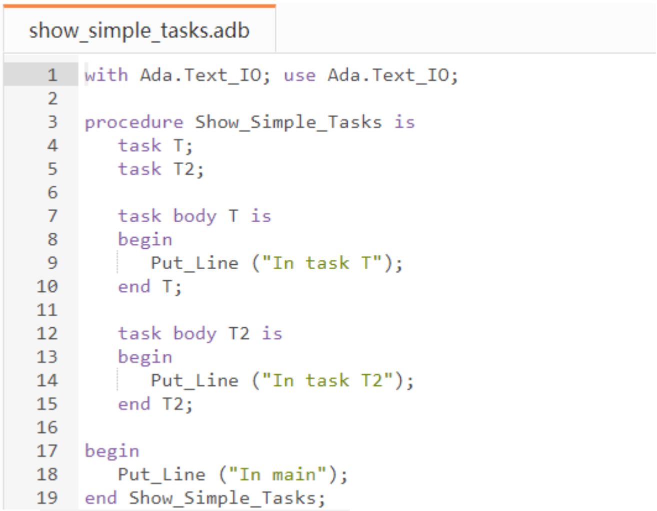
一旦主程序开始运行,

tasks start atuomatically ,不需要主动像python一样start()

he main application is itself a task (the main task).

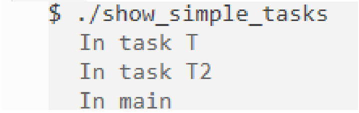
每个subtask 都有一个master task

simple task example



show_simple_tasks.adb

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Simple_Tasks is
   task T;
   task T2;

   task body T is
   begin
      Put_Line ("In task T");
   end T;

   task body T2 is
   begin
      Put_Line ("In task T2");
   end T2;

begin
   Put_Line ("In main");
end Show_Simple_Tasks;
```

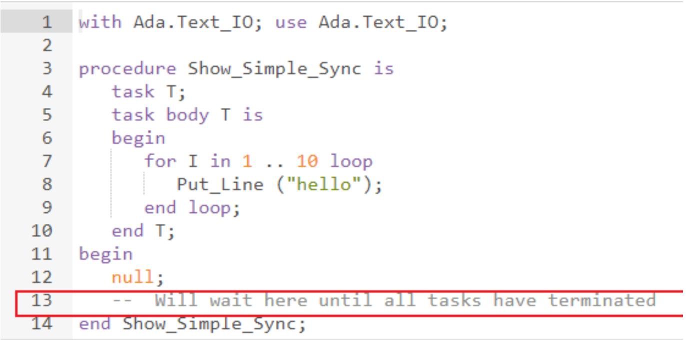


```
$ ./show_simple_tasks
   In task T
   In task T2
   In main
```

simple synchrnization

the task waits until its subtasks have finished before it allows itself to terminate.

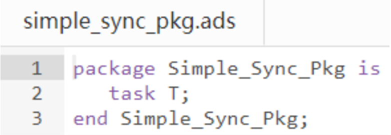
In other words, this waiting process provides synchronization between the main task and its subtasks. After this synchronization, the main task will terminate.



```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Simple_Sync is
   task T;
   task body T is
   begin
      for I in 1 .. 10 loop
         Put_Line ("hello");
      end loop;
   end T;
begin
   null;
   --  Will wait here until all tasks have terminated
end Show_Simple_Sync;
```

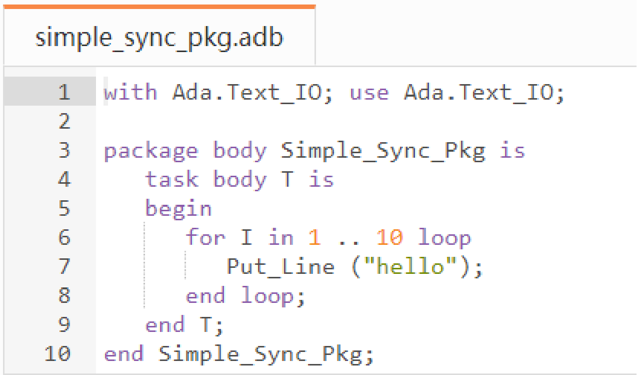
对其他subprams which conain subtasks 也同样适用

同样，对在package里面的subtask 也适用



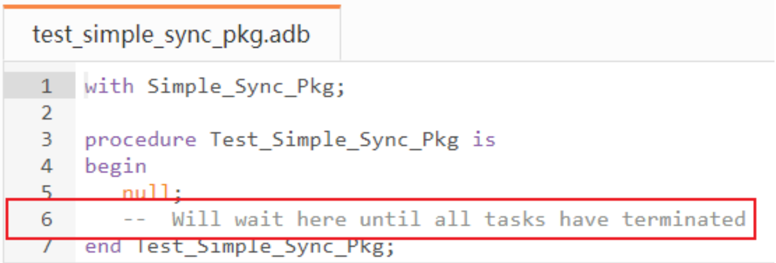
simple_sync_pkg.ads

```ada
package Simple_Sync_Pkg is
   task T;
end Simple_Sync_Pkg;
```



simple_sync_pkg.adb

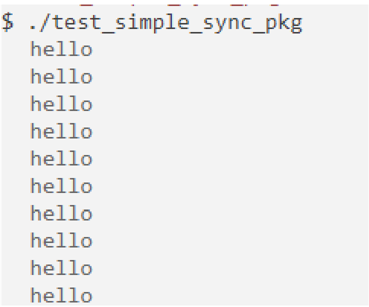
```ada
with Ada.Text_IO; use Ada.Text_IO;

package body Simple_Sync_Pkg is
   task body T is
   begin
      for I in 1 .. 10 loop
         Put_Line ("hello");
      end loop;
   end T;
end Simple_Sync_Pkg;
```



test_simple_sync_pkg.adb

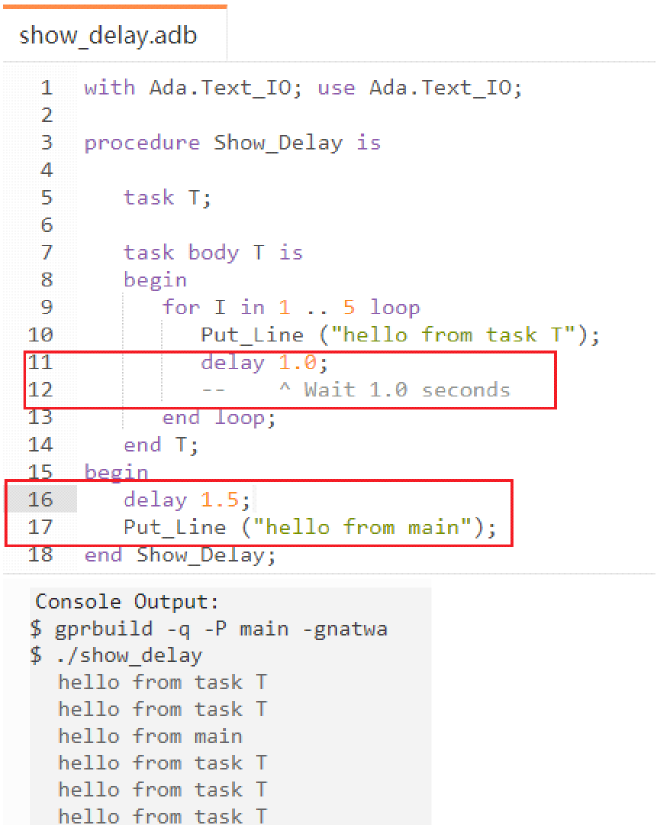
```ada
with Simple_Sync_Pkg;

procedure Test_Simple_Sync_Pkg is
begin
   null;
   --  Will wait here until all tasks have terminated
end Test_Simple_Sync_Pkg;
```

运行的结果是



```
$ ./test_simple_sync_pkg
  hello
  hello
  hello
  hello
  hello
  hello
  hello
  hello
  hello
  hello
```

Delay



show_delay.adb

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Delay is

   task T;

   task body T is
   begin
      for I in 1 .. 5 loop
         Put_Line ("hello from task T");
         delay 1.0;
         --     ^ Wait 1.0 seconds
      end loop;
   end T;
begin
   delay 1.5;
   Put_Line ("hello from main");
end Show_Delay;
```

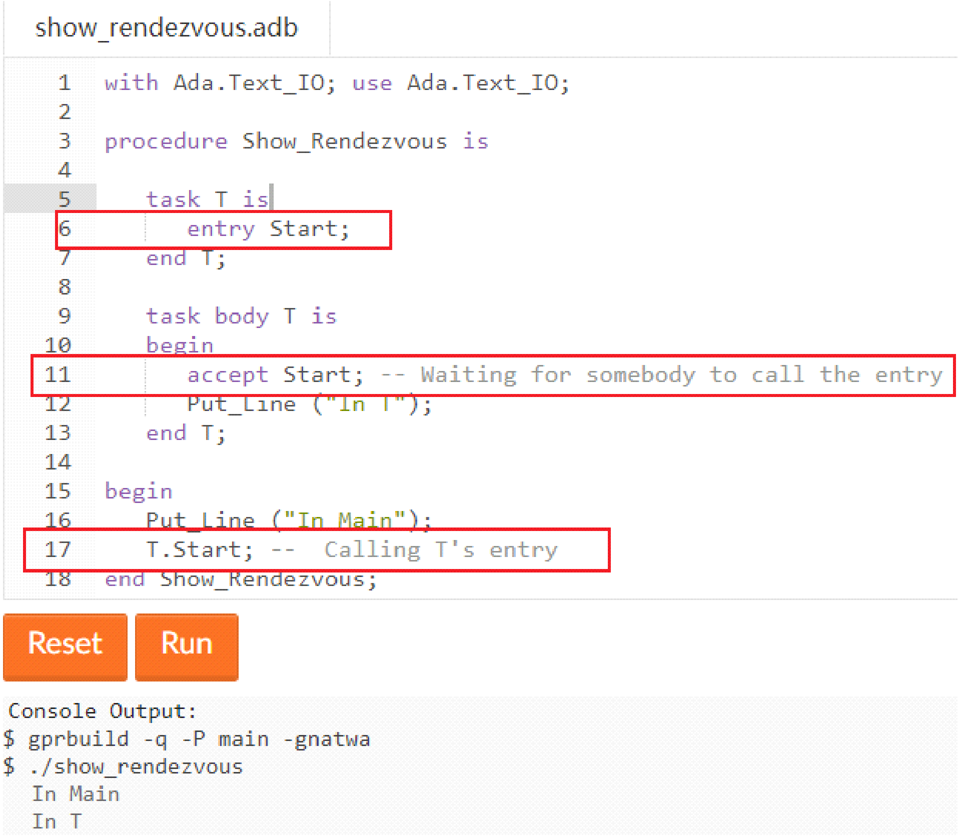
```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./show_delay
  hello from task T
  hello from task T
  hello from main
  hello from task T
  hello from task T
  hello from task T
```

两边同时运行，main要等1.5s，

会合

In the task definition, you define which part of the task will accept the entries by using the keyword accept. A task proceeds until it reaches an accept statement and then waits for the master task to synchronize with it.

也就是说，task 会进行直到它遇到accept，然后等待从master task里来的指令

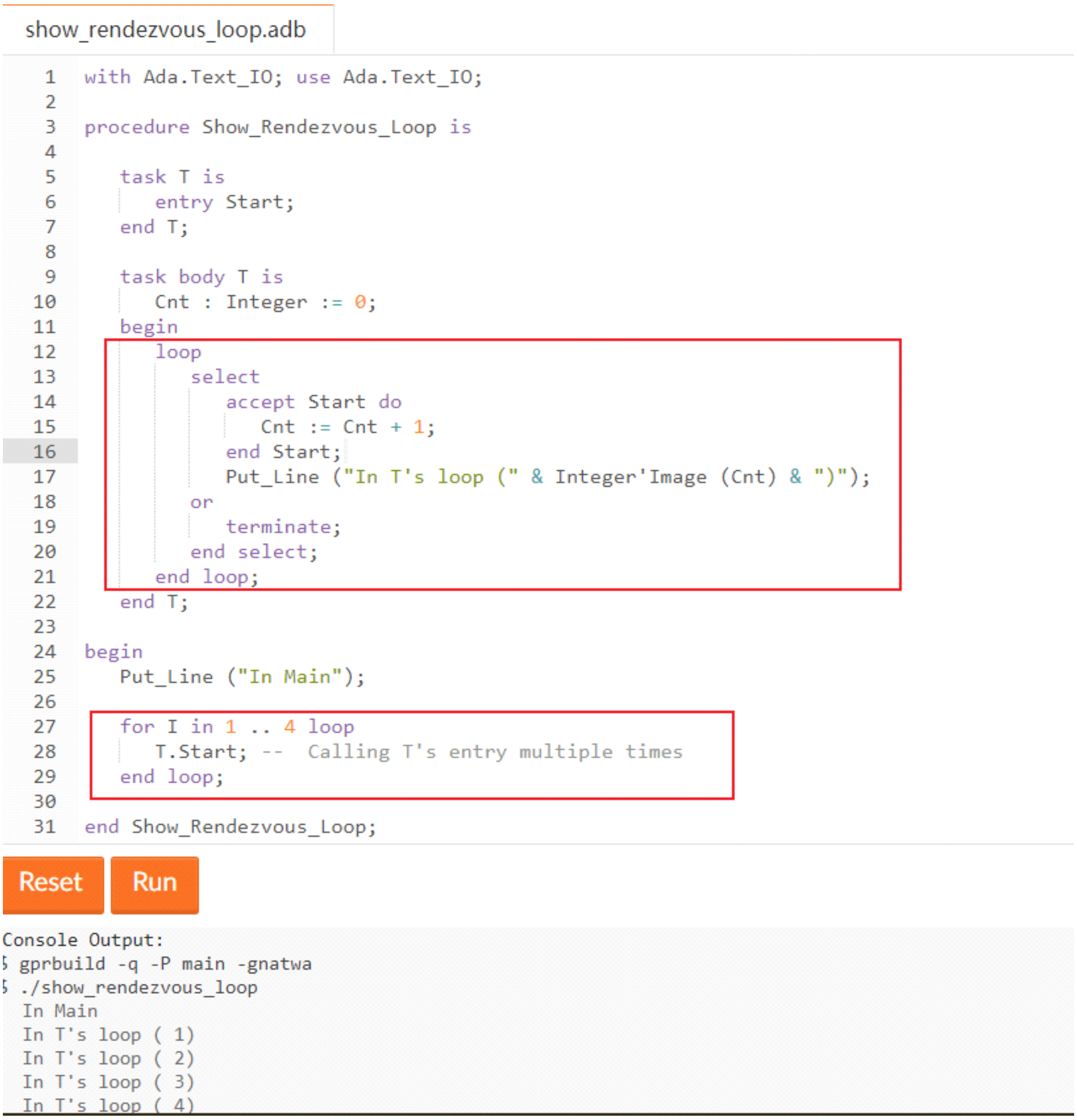


```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Rendezvous is

   task T is
      entry Start;
   end T;

   task body T is
   begin
      accept Start; -- Waiting for somebody to call the entry
      Put_Line ("In T");
   end T;

begin
   Put_Line ("In Main");
   T.Start; --  Calling T's entry
end Show_Rendezvous;
```

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./show_rendezvous
  In Main
  In T
```

Select loop

a loop containing accept statements in a task body is normally used in conjunction with a select ... or terminate statement. In simple terms, this statement allows the master task to automatically terminate the subtask when the master task finishes



```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Rendezvous_Loop is

   task T is
      entry Start;
   end T;

   task body T is
      Cnt : Integer := 0;
   begin
      loop
         select
            accept Start do
               Cnt := Cnt + 1;
            end Start;
            Put_Line ("In T's loop (" & Integer'Image (Cnt) & ")");
         or
            terminate;
         end select;
      end loop;
   end T;

begin
   Put_Line ("In Main");

   for I in 1 .. 4 loop
      T.Start; --  Calling T's entry multiple times
   end loop;

end Show_Rendezvous_Loop;
```

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./show_rendezvous_loop
  In Main
  In T's loop ( 1)
  In T's loop ( 2)
  In T's loop ( 3)
  In T's loop ( 4)
```

The accept E do ... end block is used to increment a counter.

* As long as task T is performing the do ... end block, the main task waits for the block to complete.

//也就是说，accept E do ..end 这个指令经常性用来increse a counter， 另外，只要task T 还在这个指令里面，master task就要等待它完成。

The main task is calling the Start entry multiple times in the loop from 1 .. 4.

* Because task T contains an infinite loop, it always accepts calls to the Start entry.

* When the main task finishes, it checks the status of the T task. Even though task Tcould accept new calls to the Start entry, the master task is allowed to terminate task T due to the or terminate part of the select statement.

当maintask快结束的时候，就算subtaks有infinite loop，它也可以让sub task 结束，因为有or terminate of select steatement

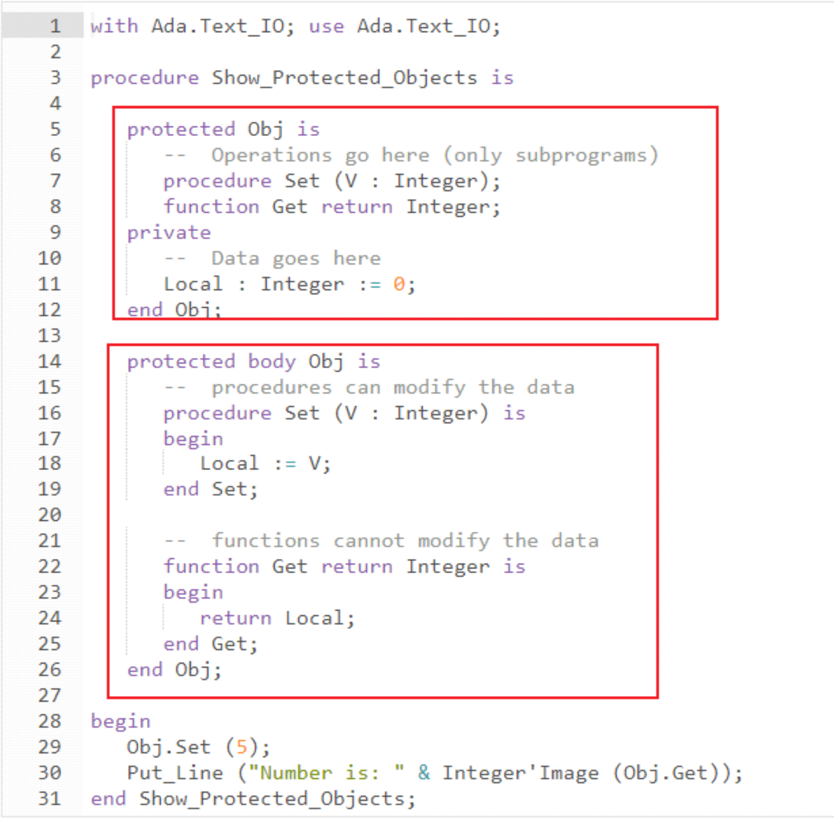
Protected objects

因为有时候，如果tasks accessing shared data，就有可能导致corruption。 比如一个task在改数据，而另一个task在读取数据。

simple example

比较类似 package。

有declaration part ，有private part，有解释的part

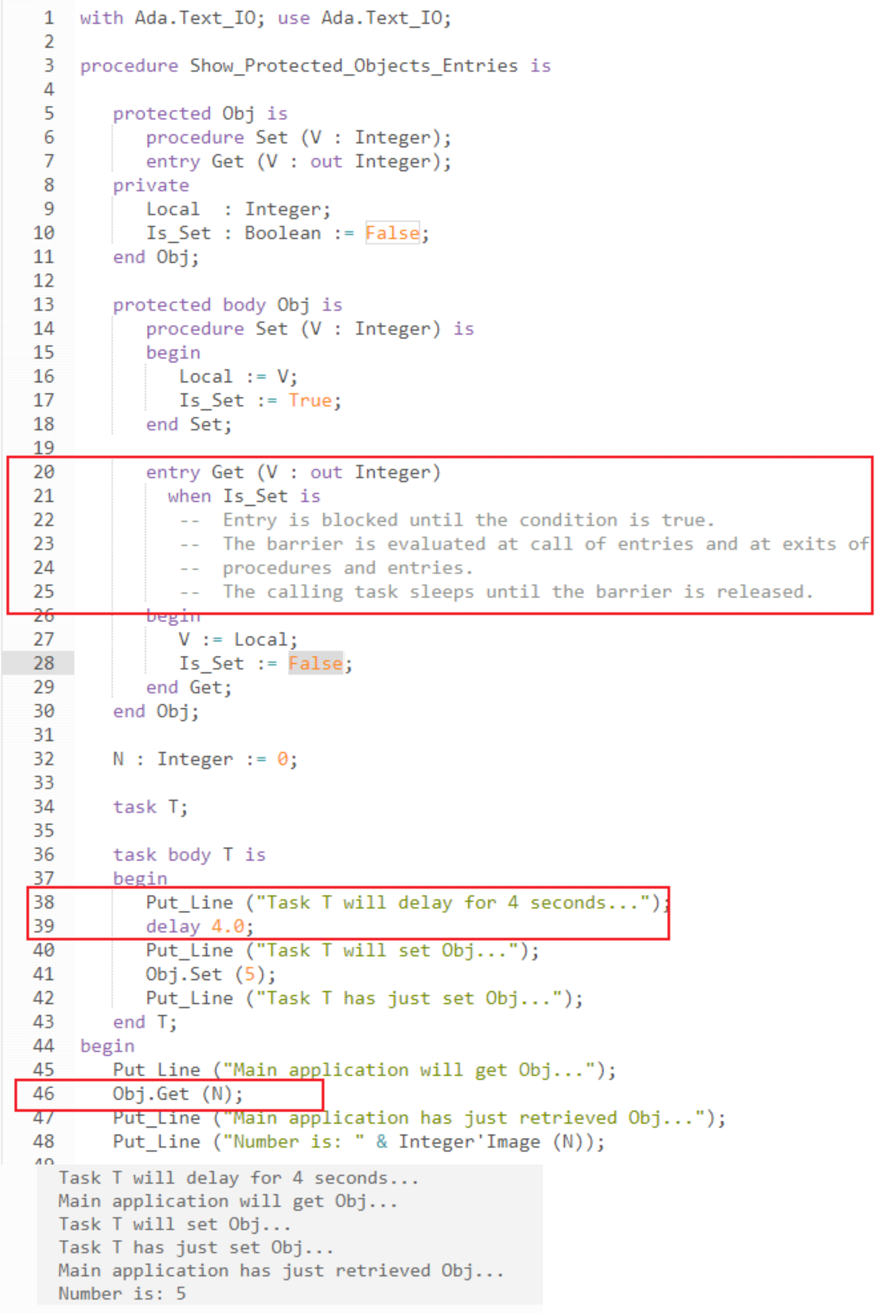


```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Protected_Objects is

   protected Obj is
      --  Operations go here (only subprograms)
      procedure Set (V : Integer);
      function Get return Integer;
   private
      --  Data goes here
      Local : Integer := 0;
   end Obj;

   protected body Obj is
      --  procedures can modify the data
      procedure Set (V : Integer) is
      begin
         Local := V;
      end Set;

      --  functions cannot modify the data
      function Get return Integer is
      begin
         return Local;
      end Get;
   end Obj;

begin
   Obj.Set (5);
   Put_Line ("Number is: " & Integer'Image (Obj.Get));
end Show_Protected_Objects;
```

Entries

为了让一个程序在读取数据之前一定要输入数据，也就是不能get before set。我们用entry和when ...is ，when就相当与一个barrier，当fulfil 的时候，我们称release the barrier

例子： 就算master task没有延迟，subtask延迟了4秒，但是因为entry 的缘故，master task还是要等待subtask完成set的步骤再进行读取。



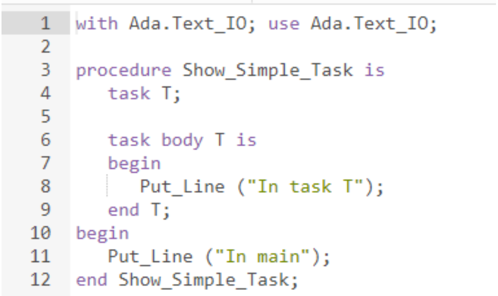
```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Protected_Objects_Entries is

   protected Obj is
      procedure Set (V : Integer);
      entry Get (V : out Integer);
   private
      Local  : Integer;
      Is_Set : Boolean := False;
   end Obj;

   protected body Obj is
      procedure Set (V : Integer) is
      begin
         Local := V;
         Is_Set := True;
      end Set;

      entry Get (V : out Integer)
        when Is_Set is
         --  Entry is blocked until the condition is true.
         --  The barrier is evaluated at call of entries and at exits of
         --  procedures and entries.
         --  The calling task sleeps until the barrier is released.
      begin
         V := Local;
         Is_Set := False;
      end Get;
   end Obj;

   N : Integer := 0;

   task T;

   task body T is
   begin
      Put_Line ("Task T will delay for 4 seconds...");
      delay 4.0;
      Put_Line ("Task T will set Obj...");
      Obj.Set (5);
      Put_Line ("Task T has just set Obj...");
   end T;
begin
   Put_Line ("Main application will get Obj...");
   Obj.Get (N);
   Put_Line ("Main application has just retrieved Obj...");
   Put_Line ("Number is: " & Integer'Image (N));
```

```
Task T will delay for 4 seconds...
Main application will get Obj...
Task T will set Obj...
Task T has just set Obj...
Main application has just retrieved Obj...
Number is: 5
```

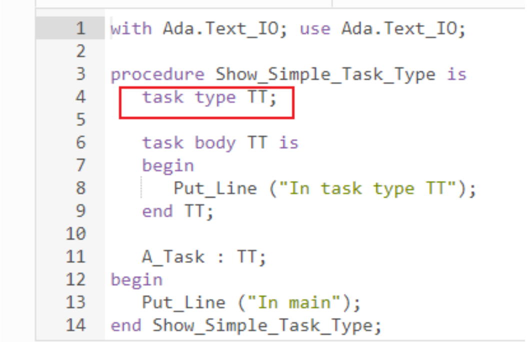
Task Type

就和variable的type一样。

对比 有type 和没type



```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Simple_Task is
   task T;

   task body T is
   begin
      Put_Line ("In task T");
   end T;
begin
   Put_Line ("In main");
end Show_Simple_Task;
```

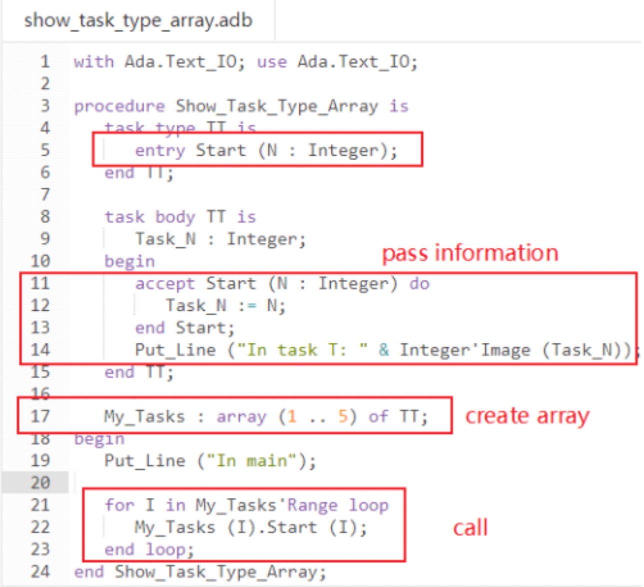


```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Simple_Task_Type is
   task type TT;

   task body TT is
   begin
      Put_Line ("In task type TT");
   end TT;

   A_Task : TT;
begin
   Put_Line ("In main");
end Show_Simple_Task_Type;
```

只是一次性和多次的区别。

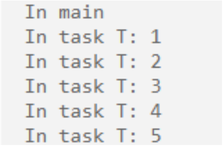
我们也可以把数据放在不同的task里，（entry）

然后我们要create array 也是跟其他type一样，



show_task_type_array.adb

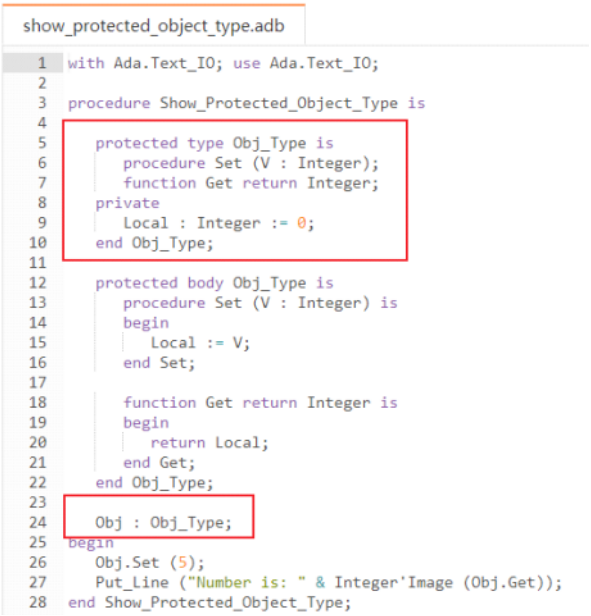
```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Task_Type_Array is
   task type TT is
      entry Start (N : Integer);
   end TT;

   task body TT is
      Task_N : Integer;
   begin
      accept Start (N : Integer) do
         Task_N := N;
      end Start;
      Put_Line ("In task T: " & Integer'Image (Task_N));
   end TT;

   My_Tasks : array (1 .. 5) of TT;
begin
   Put_Line ("In main");

   for I in My_Tasks'Range loop
      My_Tasks (I).Start (I);
   end loop;
end Show_Task_Type_Array;
```



```
In main
In task T: 1
In task T: 2
In task T: 3
In task T: 4
In task T: 5
```

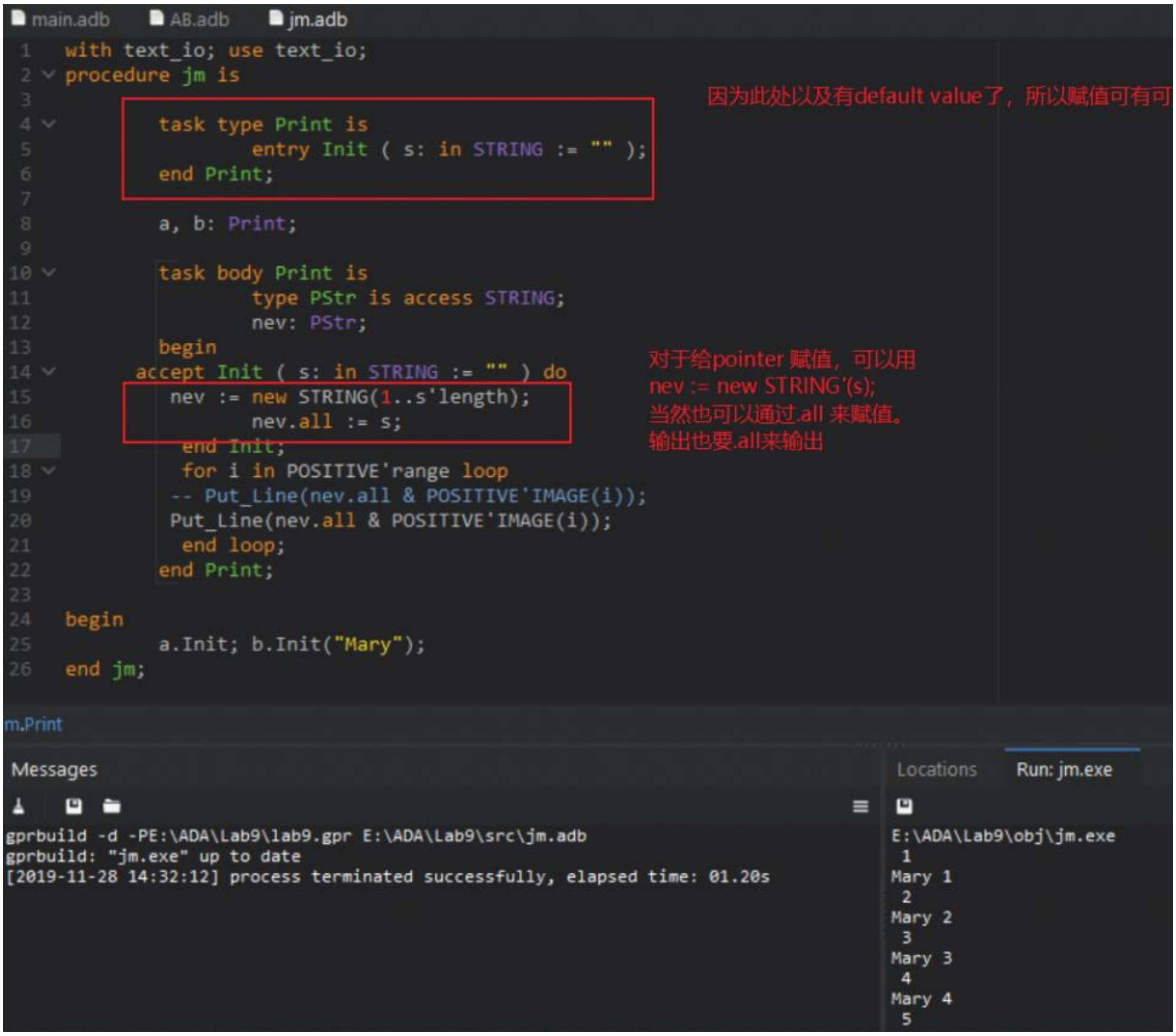
Protected types

只要把protected 替代成protected type ，其他不怎么变化。



```
with Ada.Text_IO; use Ada.Text_IO;

procedure Show_Protected_Object_Type is

   protected type Obj_Type is
      procedure Set (V : Integer);
      function Get return Integer;
   private
      Local : Integer := 0;
   end Obj_Type;

   protected body Obj_Type is
      procedure Set (V : Integer) is
      begin
         Local := V;
      end Set;

      function Get return Integer is
      begin
         return Local;
      end Get;
   end Obj_Type;

   Obj : Obj_Type;
begin
   Obj.Set (5);
   Put_Line ("Number is: " & Integer'Image (Obj.Get));
end Show_Protected_Object_Type;
```

example:



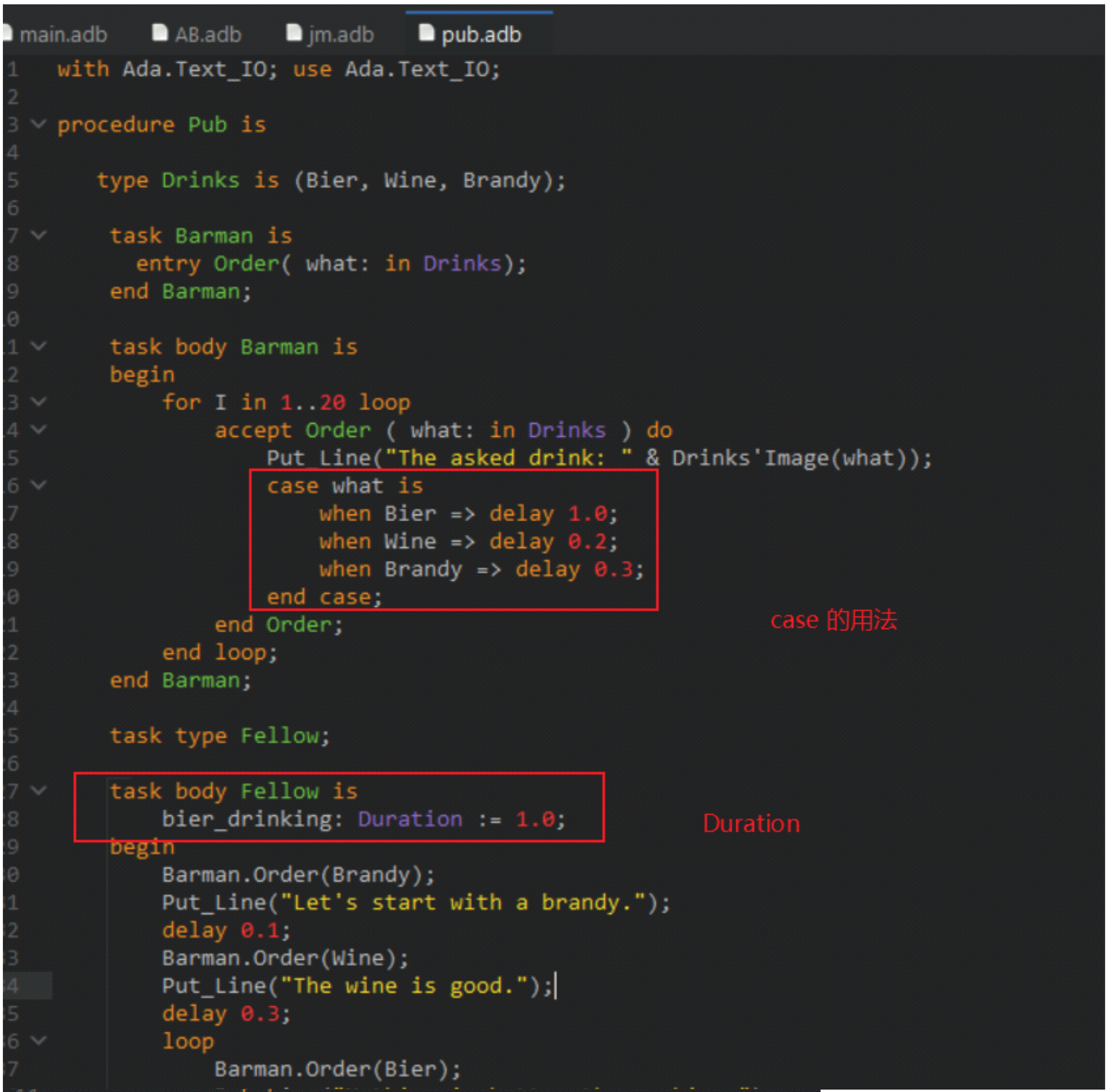
```
with text_io; use text_io;
procedure jm is

        task type Print is
                entry Init ( s: in STRING := "" );
        end Print;

        a, b: Print;

        task body Print is
                type PStr is access STRING;
                nev: PStr;
        begin
      accept Init ( s: in STRING := "" ) do
         nev := new STRING(1..s'length);
                nev.all := s;
         end Init;
         for i in POSITIVE'range loop
         -- Put_Line(nev.all & POSITIVE'IMAGE(i));
         Put_Line(nev.all & POSITIVE'IMAGE(i));
         end loop;
        end Print;

begin
        a.Init; b.Init("Mary");
end jm;
```

因为此处以及有default value了，所以赋值可有可

对于给pointer 赋值，可以用
nev := new STRING'(s);
当然也可以通过.all 来赋值。
输出也要.all来输出

```
gprbuild -d -PE:\ADA\Lab9\lab9.gpr E:\ADA\Lab9\src\jm.adb
gprbuild: "jm.exe" up to date
[2019-11-28 14:32:12] process terminated successfully, elapsed time: 01.20s
```

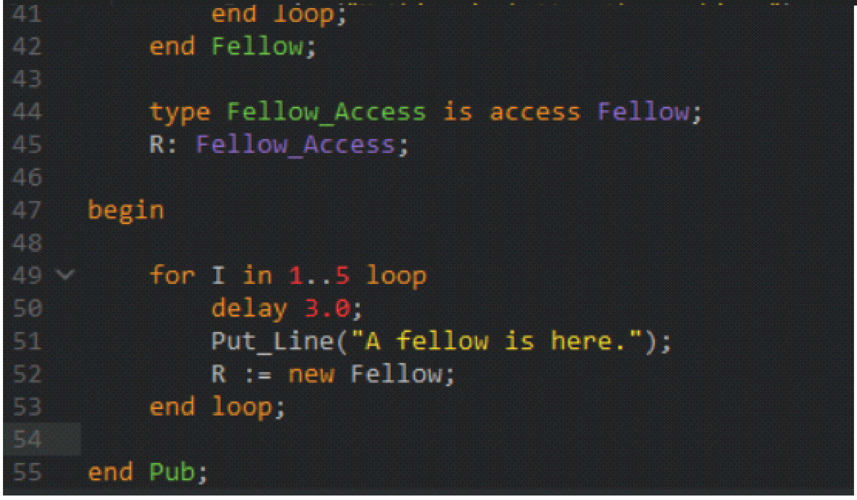
```
E:\ADA\Lab9\obj\jm.exe
 1
Mary 1
 2
Mary 2
 3
Mary 3
 4
Mary 4
 5
```



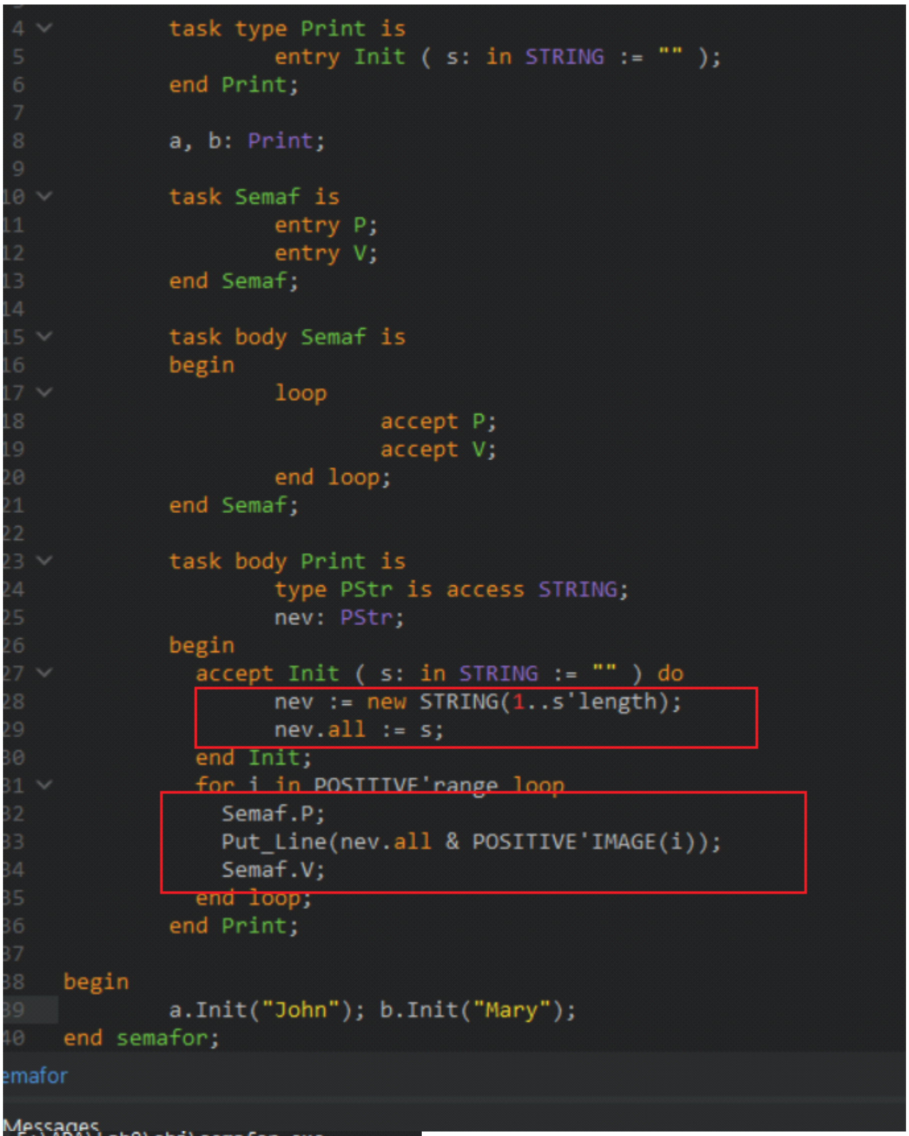
```
with Ada.Text_IO; use Ada.Text_IO;

procedure Pub is

   type Drinks is (Bier, Wine, Brandy);

    task Barman is
      entry Order( what: in Drinks);
    end Barman;

    task body Barman is
    begin
        for I in 1..20 loop
            accept Order ( what: in Drinks ) do
                Put_Line("The asked drink: " & Drinks'Image(what));
                case what is
                    when Bier => delay 1.0;
                    when Wine => delay 0.2;
                    when Brandy => delay 0.3;
                end case;
            end Order;
        end loop;
    end Barman;

    task type Fellow;

    task body Fellow is
        bier_drinking: Duration := 1.0;
    begin
        Barman.Order(Brandy);
        Put_Line("Let's start with a brandy.");
        delay 0.1;
        Barman.Order(Wine);
        Put_Line("The wine is good.");
        delay 0.3;
        loop
            Barman.Order(Bier);
```

case 的用法

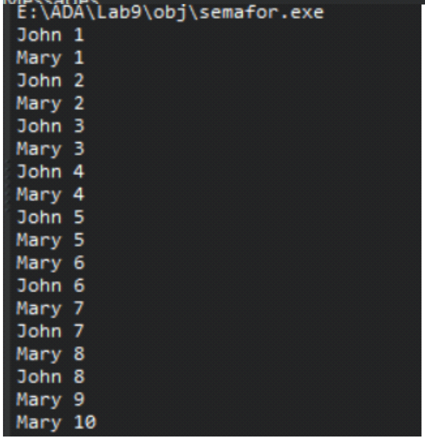
Duration



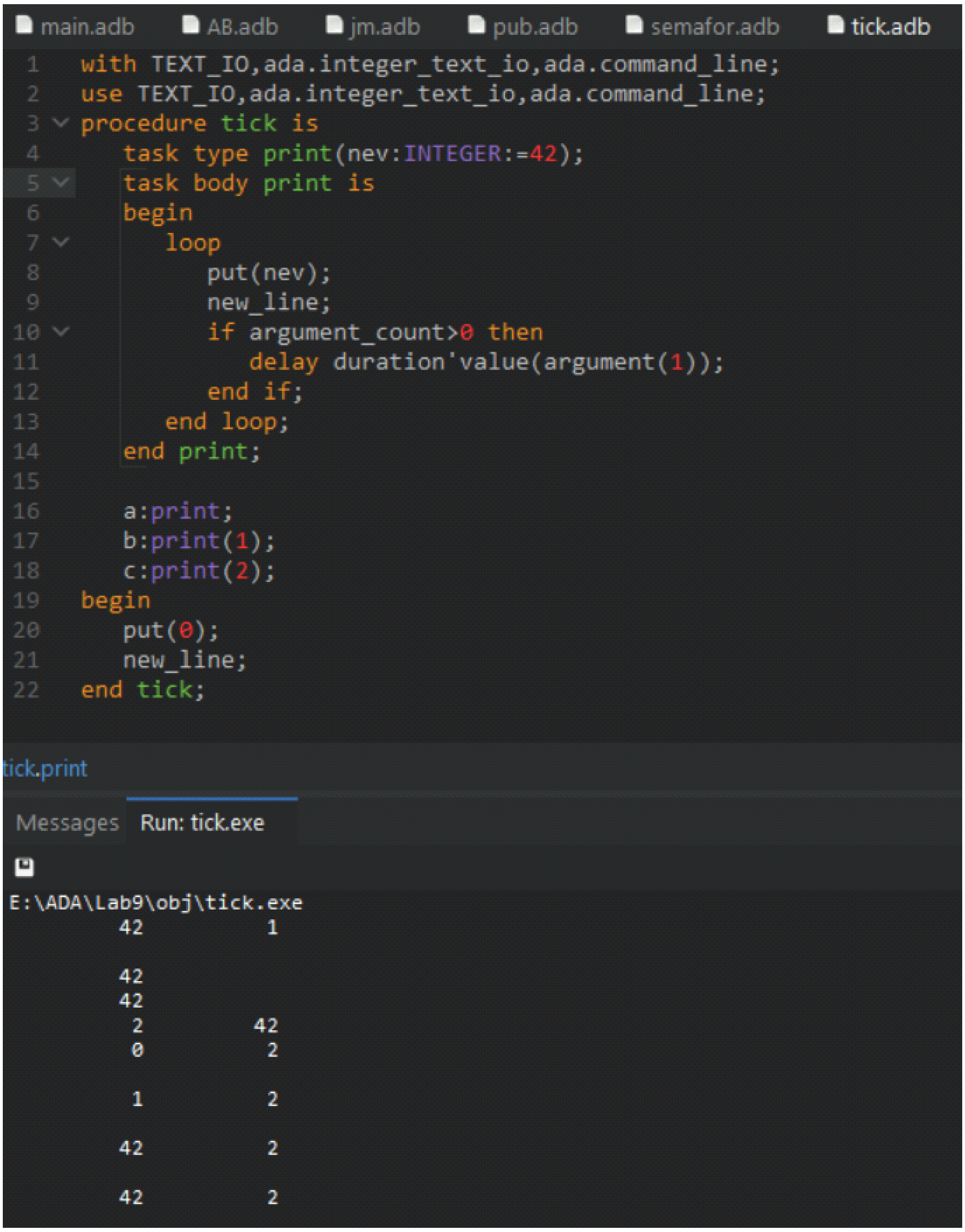
```
        end loop;
    end Fellow;

    type Fellow_Access is access Fellow;
    R: Fellow_Access;

begin

    for I in 1..5 loop
        delay 3.0;
        Put_Line("A fellow is here.");
        R := new Fellow;
    end loop;

end Pub;
```



```
        task type Print is
                entry Init ( s: in STRING := "" );
        end Print;

        a, b: Print;

        task Semaf is
                entry P;
                entry V;
        end Semaf;

        task body Semaf is
        begin
                loop
                        accept P;
                        accept V;
                end loop;
        end Semaf;

        task body Print is
                type PStr is access STRING;
                nev: PStr;
        begin
          accept Init ( s: in STRING := "" ) do
                nev := new STRING(1..s'length);
                nev.all := s;
          end Init;
          for i in POSITIVE'range loop
            Semaf.P;
            Put_Line(nev.all & POSITIVE'IMAGE(i));
            Semaf.V;
          end loop;
        end Print;

begin
        a.Init("John"); b.Init("Mary");
end semafor;
```



```
E:\ADA\Lab9\obj\semafor.exe
John 1
Mary 1
John 2
Mary 2
John 3
Mary 3
John 4
Mary 4
John 5
Mary 5
Mary 6
John 6
Mary 7
John 7
Mary 8
John 8
Mary 9
Mary 10
```



```ada
with TEXT_IO,ada.integer_text_io,ada.command_line;
use TEXT_IO,ada.integer_text_io,ada.command_line;
procedure tick is
   task type print(nev:INTEGER:=42);
   task body print is
   begin
      loop
         put(nev);
         new_line;
         if argument_count>0 then
            delay duration'value(argument(1));
         end if;
      end loop;
   end print;

   a:print;
   b:print(1);
   c:print(2);
begin
   put(0);
   new_line;
end tick;
```

```
E:\ADA\Lab9\obj\tick.exe
         42          1

         42
         42
          2         42
          0          2

          1          2

         42          2

         42          2
```

？？

Ada.command_line

function Argument_Count return Natural;//计算指令数量

function Argument (Number : in Positive) return String; //返回指令，number从1. Argument_Count

Given a **petrol station** with N filling stations and more then N cars, write an agenda of the activities of the station.

1.The station should be protected. **(protected object')**

2.The cars after arrival will fill their tank (max N cars at the same time) and leave the station. If more cars are arriving, they should queue at the stations and wait for an empty one. As soon as one is empty, the car will go there.

Each car has: **（Task type）**

a licence number, and when is filling up has to give it to the station to register in the agenda, and a filling up time (both are discriminants of the dynamically created cars).

The cars are arriving in random time intervals (between 0.1 and 0.5 seconds) at the station.

There are 3 types of **drivers**:

impatient, if no station is free then leaves immediately,

patient that waits 0.5 seconds for an empty place, and the third who waits anyhow for a free place, since he has no fuel left.

The type of the drivers should be determined randomly.

Write every activity on the screen using a protected orinter.