OBJECT philosophers IS

PASSIVE

pragmas

```
PRAGMA line_feed
  (option => 1)
PRAGMA line_feed
  (option => 2)
PRAGMA main
  (operation_name => start,
   unit_name => run)
PRAGMA no_subunits
PRAGMA comment
PRAGMA compiler
  (name => gnat,
   options => --| |--)
```

ADA EXTRACTED CODE

extraction messages

```
=== begin code extraction : Mon Dec 23 10:42:38  2002
*** no textual description for : TYPE Philosopher of object : phil
*** no textual description for : TYPE Philosopher_Ptr of object : ph
phil
*** no textual description for : TYPE States of object : phil
*** no textual description for : TYPE Think_Times of object : phil
*** no textual description for : TYPE Meal_Times of object : phil
*** no textual description for : TYPE Life_Time of object : phil
*** no textual description for : DATA Think_Length of object : phil
*** no textual description for : DATA Meal_Length of object : phil
*** no textual description for : OPERATION start_eating of object :
phil
*** no textual description for : OBCS spec of object : phil
*** no textual description for : TYPE Unique_DNA_Codes of object : s
society
*** no textual description for : OPERATION get_name of object : soci
society
*** no textual description for : DATA Name_Register of object : soci
society
*** no textual description for : OPERATION get_name of object : soci
society
*** no textual description for : TYPE Stick of object : chop
*** no textual description for : OPERATION pick_up of object : chop
*** no textual description for : OPERATION put_down of object : chop
chop
*** no textual description for : TYPE Stick_Ptr of object : chop
*** no textual description for : OPERATION pick_up of object : chop
*** no textual description for : OPERATION put_down of object : chop
chop

Halt. Program terminated normally
=== end code extraction : Mon Dec 23 10:42:42  2002
=== begin compilation : Mon Dec 23 10:42:42  2002
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada run.a
run.adb
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada philo
philosophers.ads
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
```

```
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada room.
room.adb
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada phil.
phil.adb
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada windo
windows.adb
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada chop.
chop.adb
gcc -c -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada socie
society.adb
gcc -c -I./ -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I/
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada -I-
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_g
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_ge
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_gen
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_gene
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_gener
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generi
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generic
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generic.
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generic.a
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generic.ad
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generic.adb
//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada\random_generic.adb
gnatbind -aO./ -aO//Groslulu/home4/stood/stood4.3/libs/calendar/_ada
-aO//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -aO//Groslulu/
-aO//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -aO//Gro
-aO//Groslulu/home4/stood/stood4.3/examples/screen/_ada -aO//Groslul
-aO//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -aO//Gr
-aO//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada -I-
-x run.ali
gnatlink run.ali
=== end compilation : Mon Dec 23 10:43:19  2002
```

spec

```
-- Dining Philosophers - Ada 95 edition
--
-- Start is the main program.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  required interface :
--   Required OPERATION :
```

```
--     OPERATION : start_serving of object : room
--   Required EXCEPTION : NONE
--   Required TYPE : NONE
--   Required CONSTANT : NONE
--   Required DATA : NONE

--  visibility on required modules :

--  visibility on implementing modules :
with room;

package philosophers is

  -- Main procedure.
  procedure Start
     renames room.start_serving;

end philosophers;
```

makefile

```
SYSTEM_CONF_PATH=//Groslulu/home4/stood/stood4.3/libs/calendar/_ada:
//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada://Groslulu/
home4/stood/stood4.3/examples/philosophers/_ada://Groslulu/home4/sto
od/stood4.3/examples/screen/_ada://Groslulu/home4/stood/stood4.3/lib
s/standard/_ada://Groslulu/home4/stood/stood4.3/libs/text_io/_ada://
Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada://Groslulu/h
ome4/stood/stood4.3/examples/random_generic/_ada:

mv philosophers.adb run.adb
gnatmake -I//Groslulu/home4/stood/stood4.3/libs/calendar/_ada -I//Gr
-I//Groslulu/home4/stood/stood4.3/examples/nt_console/_ada -I//Grosl
-I//Groslulu/home4/stood/stood4.3/examples/screen/_ada -I//Groslulu/
-I//Groslulu/home4/stood/stood4.3/libs/discrete_random/_ada -I//Gros
-I//Groslulu/home4/stood/stood4.3/examples/random_generic/_ada   run
```

END philosophers

OBJECT room IS

ACTIVE

spec

```
-- Dining Philosophers - Ada 95 edition
--
-- Room.OBCS is responsible for assigning seats at the table,
-- "left" and "right" chopsticks, and for reporting interesting
-- events to the outside world.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  required interface :
--   Required OPERATION :
--     OPERATION : open of object : windows
--     OPERATION : title of object : windows
--     OPERATION : borders of object : windows
--     OPERATION : put#1 of object : windows
--     OPERATION : new_line of object : windows
--     OPERATION : start_eating of object : phil
--     OPERATION : get_name of object : society
--     OPERATION : Clock of object : calendar
--   Required EXCEPTION : NONE
--   Required TYPE :
--     TYPE : Window of object : windows
--     TYPE : Philosopher of object : phil
--     TYPE : Philosopher_Ptr of object : phil
--     TYPE : States of object : phil
--     TYPE : Unique_DNA_Codes of object : society
--     TYPE : Stick of object : chop
--     TYPE : Stick_Ptr of object : chop
--     TYPE : Boolean of object : standard
--     TYPE : Integer of object : standard
--     TYPE : Natural of object : standard
--     TYPE : Positive of object : standard
--     TYPE : Time of object : calendar
--   Required CONSTANT : NONE
--   Required DATA : NONE

--  visibility on required modules :
with phil;
use type phil.Philosopher;
use type phil.Philosopher_Ptr;
use type phil.States;
with society;
use type society.Unique_DNA_Codes;
with chop;
use type chop.Stick;
use type chop.Stick_Ptr;

package room is

  -- Room.Get_Stick is an access function to internal Sticks variabl
variable.
  -- It requires a chopstick ID (Which_Stick) to return a pointer to
to relevant
  --  protected object.
  -- (cf.FR2/Provide_chopsticks:)
  function get_stick (
    which_Stick : IN Positive)
     return Chop.Stick_Ptr;

  -- A dedicated state variable manages current state of dining room
room.
```

```
  -- This variable "Started" has a default value of "FALSE" and beco
become "TRUE"
  --  after Start_Serving has been executed.
  -- Start_Serving and Report_State have both STATE and protocole co
constraints.
  task OBCS is
    entry start_serving;
    entry report_state (
      Which_Phil : IN Society.Unique_DNA_Codes;
      Which_State : IN Phil.States;
      How_Long : IN Natural := 0;
      Which_Meal : IN Natural := 0);
  end OBCS;
  -- Room.Start_Serving is called by main procedure and renames OBCS
OBCS.Start_Serving
  --  task entry.
  -- This procedure has no parameter.
  -- (cf.FR1/Prepare&begin_diner:)
  procedure start_serving
     renames OBCS.start_serving;

  -- Room.Report_State is called by Phil.Start_Eating and renames OB
OBCS.Report_State
  --  task entry.
  -- This procedure has four parameters:
  -- - Which_Phil: identifies actual Philosopher sending the message
message.
  -- - State: current state of sender.
  -- - How_Long: length of current state (or identifier of used chop
chopstick).
  -- - Which_Meal: current meal.
  -- (cf.FR3/Report_events:)
  procedure report_state (
    Which_Phil : IN Society.Unique_DNA_Codes;
    Which_State : IN Phil.States;
    How_Long : IN Natural := 0;
    Which_Meal : IN Natural := 0)
     renames OBCS.report_state;

end room;
```

body

```
-- Dining Philosophers - Ada 95 edition
--
-- Room.OBCS is responsible for assigning seats at the table,
-- "left" and "right" chopsticks, and for reporting interesting
-- events to the outside world.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.

with Phil;
pragma Elaborate(Phil);

--  visibility on required modules :
with calendar;
use type calendar.Time;
with windows;
use type windows.Window;

--  visibility on objects required by nested operation bodies :


package body room is

  -- Specifies the total number of seats around the table. It is lim
limited to
  --  five in this example.
```

```
Table_Size : constant := 5;

-- Identifies the possible locations around the table.
subtype Table_Type is Positive range 1..Table_Size;

-- First chopstick shared between seats 5 and 1.
S1 : aliased Chop.Stick;

-- second chopstick shared between seats 1 and 2.
S2 : aliased Chop.Stick;

-- Third chopstick shared between seats 2 and 3.
S3 : aliased Chop.Stick;

-- Fourth chopstick shared between seats 3 and 4.
S4 : aliased Chop.Stick;

-- Fifth chopstick shared between seats 4 and 5.
S5 : aliased Chop.Stick;

-- An array of pointers to the chopsticks.
Sticks : array (Table_Type) of Chop.Stick_Ptr;

-- First Philosopher.
P1 : aliased Phil.Philosopher(My_ID => 1);

-- Second Philosopher.
P2 : aliased Phil.Philosopher(My_ID => 2);

-- Third Philosopher.
P3 : aliased Phil.Philosopher(My_ID => 3);

-- Fourth Philosopher.
P4 : aliased Phil.Philosopher(My_ID => 4);

-- Fifth Philosopher.
P5 : aliased Phil.Philosopher(My_ID => 5);

-- An array of pointers to the Philosophers.
Phils : array (Table_Type) of Phil.Philosopher_Ptr;

-- An array of windows. One window for each seat.
Phil_Windows : array (Table_Type) of Windows.Window;

-- An array to indicate which seat each Philosopher occupies:
-- Philosopher 1 occupies seat 1;
-- Philosopher 2 occupies seat 3;
-- Philosopher 3 occupies seat 5;
-- Philosopher 4 occupies seat 4;
-- Philosopher 5 occupies seat 2;
Phil_Seats : array (Society.Unique_DNA_Codes) of Table_Type;

-- Current time obtained by Calendar.Clock.
T : Natural;

-- Time when application is launched.
Start_Time : Calendar.Time;

-- State variable to switch between "waiting" and "dining" states.
-- Initial state is "Waiting".
Started : boolean := false;

-- Performs following actions:
-- - Calculates Start_Time;
-- - Puts chopsticks on the table;
-- - Assigns Philosophers to seats at the table;
-- - Opens and draw a window to observe each seat;
-- - Assigns right and left chopsticks to each Philosopher;
procedure OPCS_start_serving is
begin
```

```
  -- starting date is stored:
  Start_Time := Calendar.Clock;

  -- chopsticks are put on the table:
  Sticks :=
    (S1'Access,
     S2'Access,
     S3'Access,
     S4'Access,
     S5'Access);

  -- philosophers are assigned to seats at the table
  Phils :=
    (P1'Access,
     P3'Access,
     P5'Access,
     P4'Access,
     P2'Access);

  -- which seat each phil occupies:
  Phil_Seats := (1, 3, 5, 4, 2);

  -- a window is open for each seat:
  Phil_Windows :=
    (Windows.Open(( 1, 24), 7, 30),
     Windows.Open(( 9,  2), 7, 30),
     Windows.Open(( 9, 46), 7, 30),
     Windows.Open((17,  7), 7, 30),
     Windows.Open((17, 41), 7, 30));

  -- windows borders are drawn:
  for Which_Win in Phil_Windows'range loop
    Windows.Borders(Phil_Windows(Which_Win),'+','|','-');
  end loop;

  -- philosophers are assigned their chopsticks:
  Phils (1).Start_Eating(1, 1, 2);
  Phils (3).Start_Eating(3, 3, 4);
  Phils (2).Start_Eating(2, 2, 3);
  Phils (5).Start_Eating(5, 1, 5);
  Phils (4).Start_Eating(4, 4, 5);

  -- dining room state changes:
  Started := true;
end OPCS_start_serving;

-- Performs following actions:
-- - Calculates current time;
-- - Displays a message on relevant window.
procedure OPCS_report_state (
  Which_Phil : IN Society.Unique_DNA_Codes;
  Which_State : IN Phil.States;
  How_Long : IN Natural := 0;
  Which_Meal : IN Natural := 0) is
begin

  T := Natural (Calendar.Clock - Start_Time);

  case Which_State is

    when Phil.Breathing =>
      Windows.Title(
          Phil_Windows(Phil_Seats(Which_Phil)),
          Society.Get_Name(Which_Phil), '-');
      Windows.Put(
          Phil_Windows(Phil_Seats(Which_Phil)),
          "T =" & Integer'Image (T) & " " & "Breathing...");
      Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));
```

```
          when Phil.Thinking =>
            Windows.Put(
                Phil_Windows(Phil_Seats(Which_Phil)),
                "T =" & Integer'Image (T) & " " & "Thinking"
                & Integer'Image (How_Long) & " seconds.");
            Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));

          when Phil.Eating =>
            Windows.Put(
                Phil_Windows(Phil_Seats(Which_Phil)),
                "T =" & Integer'Image (T) & " " & "Meal"
                & Integer'Image (Which_Meal) & ","
                & Integer'Image (How_Long) & " seconds.");
            Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));

          when Phil.Done_Eating =>
            Windows.Put(
                Phil_Windows(Phil_Seats(Which_Phil)),
                "T =" & Integer'Image (T) & " " & "Yum-yum (burp)");
            Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));

          when Phil.Got_One_Stick =>
            Windows.Put(
                Phil_Windows(Phil_Seats(Which_Phil)),
                "T =" & Integer'Image (T) & " " & "First chopstick"
                & Integer'Image (How_Long));
            Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));

          when Phil.Got_Other_Stick =>
            Windows.Put(
                Phil_Windows(Phil_Seats(Which_Phil)),
                "T =" & Integer'Image (T) & " " & "Second chopstick"
                & Integer'Image (How_Long));
            Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));

          when Phil.Dying =>
            Windows.Put(
                Phil_Windows(Phil_Seats(Which_Phil)),
                "T =" & Integer'Image (T) & " " & "Croak");
            Windows.New_Line(Phil_Windows(Phil_Seats(Which_Phil)));

        end case; -- Which_State
      end OPCS_report_state;

      -- Just returns a pointer to specified chopstick.
      function get_stick (
        which_Stick : IN Positive)
         return Chop.Stick_Ptr is
      begin
        return Sticks(Which_Stick);
      end get_stick;

      -- A dedicated state variable manages current state of dining room
    room.
      -- This variable "Started" has a default value of "FALSE" and beco
    become "TRUE"
      --  after Start_Serving has been executed.
      -- Start_Serving and Report_State have both STATE and protocole co
    constraints.
      task body OBCS is

        start_serving_accepted : boolean := false;
        report_state_accepted : boolean := false;
      begin
        loop
          start_serving_accepted := not Started;
          report_state_accepted := Started;
          select
            -- ASER
            when start_serving_accepted =>
```

```
          accept start_serving;
          begin
            OPCS_start_serving;
            -- Initial state. Started is set to FALSE.
            -- This transition is triggered by Start_Serving execution
execution request.
            -- No additional condition, neither exception code is requ
required.
            Started := true;
          exception
            when others =>
              raise;
          end;
        or
          when not start_serving_accepted =>
          accept start_serving;
          null;
        or
          -- HSER
          when report_state_accepted =>
          accept report_state (
            Which_Phil : IN Society.Unique_DNA_Codes;
            Which_State : IN Phil.States;
            How_Long : IN Natural := 0;
            Which_Meal : IN Natural := 0) do
            begin
              OPCS_report_state (Which_Phil, Which_State, How_Long, Wh
Which_Meal );
              -- Running state. Started is set to TRUE.
              -- This transition is triggered by Report_Sta  te executi
execution request.
              -- No additional condition, neither exception code is re
required.
              -- Current state is not changed.
              Started := true;
            exception
              when others =>
                raise;
            end;
          end report_state;
        or
          when not report_state_accepted =>
          accept report_state (
            Which_Phil : IN Society.Unique_DNA_Codes;
            Which_State : IN Phil.States;
            How_Long : IN Natural := 0;
            Which_Meal : IN Natural := 0);
          null;
        or
          terminate;
        end select;
      end loop;

    end OBCS;


  end room;
```

END room

CLASS windows IS

PASSIVE

pragmas

```
PRAGMA init_bloc
  (init_op => initialize)
```

spec

```
-- Dining Philosophers - Ada 95 edition
--
-- Manager for simple, nonoverlapping screen windows.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  required interface :
--   Required OPERATION :
--     OPERATION : ClearScreen of object : screen
--     OPERATION : MoveCursor of object : screen
--     OPERATION : New_Line of object : text_io
--     OPERATION : Put of object : text_io
--   Required EXCEPTION : NONE
--   Required TYPE :
--     TYPE : Height of object : screen
--     TYPE : Width of object : screen
--     TYPE : Position of object : screen
--     TYPE : Character of object : standard
--     TYPE : String of object : standard
--   Required CONSTANT : NONE
--   Required DATA : NONE

--  visibility on required modules :
with screen;
use type screen.Height;
use type screen.Width;
use type screen.Position;

package windows is

  type Window is private;

  -- Pre: UpperLeft, Weight, and Width are defined
  -- Post:  returns a Window with the given upper-left corner, heigh
height, and width
  function open (
    UpperLeft : IN Screen.Position;
    Height : IN Screen.Height;
    Width : IN Screen.Width)
     return Window;

  -- Pre:   me, Name, and Under are defined
  -- Post:  Name is displayed at the top of the window me, underline
underlined with the
  --  character Under
  procedure title (
    me : IN OUT Window;
    Name : IN String;
    Under : IN Character);

  -- Pre: All parameters are defined
  -- Post:  Draw border around current writable area in window with
characters
  --  specified.
  --   Call this BEFORE Title.
```

```
  procedure borders (
    me : IN OUT Window;
    Corner : IN Character;
    Down : IN Character;
    Across : IN Character);

  -- Pre: me, and P are defined, and P lies within the area of me
  -- Post:  Cursor is moved to the specified position.
  --   Coordinates are relative to the upper left corner of me, whic
which is (1,1)
  procedure movecursor (
    me : IN OUT Window;
    P : IN Screen.Position);

  -- Pre: me, and Ch are defined.
  -- Post:  Ch is displayed in the window at the next available posi
position.
  --   If end of column, go to the next row.
  --   If end of window, go to the top of the window.
  procedure put (
    me : IN OUT Window;
    Ch : IN Character);

  -- Pre: me, and S are defined.
  -- Post:  Ch is displayed in the window, "line-wrapped" if necessa
necessary
  procedure put (
    me : IN OUT Window;
    S : IN String);

  -- Pre: me is defined.
  -- Post:  Cursor moves to beginning of next line of me;
  --   line is not blanked until next character is written
  procedure new_line (
    me : IN OUT Window);

private

  -- First : coordinates of upper left corner;
  -- Last : coordinates of lower right corner;
  -- Current : current cursor position.
  type Window is tagged
  record
    First : screen.Position;
    Last : screen.Position;
    Current : screen.Position;
  end record;

end windows;
```

body

```
-- Dining Philosophers - Ada 95 edition
--
-- Manager for simple, nonoverlapping screen windows.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  visibility on required modules :
with text_io;

--  visibility on objects required by nested operation bodies :

package body windows is

  -- Used to erase partially the screen.
  procedure erasetoendofline (
```

```
  me : IN OUT Window);

-- Instanciates a new Window named "Result"
-- Sets Result attributes (Current, First and Last)
-- Returns Result.
function open (
  UpperLeft : IN Screen.Position;
  Height : IN Screen.Height;
  Width : IN Screen.Width)
   return Window is
  Result : Window;
begin
  Result.Current := UpperLeft;
  Result.First := UpperLeft;
  Result.Last := (Row => UpperLeft.Row + Height - 1,
     Column => UpperLeft.Column + Width - 1);
  return Result;
end open;

-- Sets cursor at the beginning of first line.
-- Writes title string
-- If "Under" is blank then continue
-- else draw a separation line
-- Reduces writable area as required.
procedure title (
  me : IN OUT Window;
  Name : IN String;
  Under : IN Character) is
begin

  -- Put name on top line
  me.Current := me.First;
  Put(me, Name);
  New_Line(me);

  -- Underline name if desired, and reduce the writable area
  -- of the window by one line
  if Under = ' ' then
    -- no underlining
    me.First.Row := me.First.Row + 1;
  else
    -- go across the row, underlining
    for Count in me.First.Column..me.Last.Column loop
      Put(me, Under);
    end loop;
    New_Line(me);
    -- reduce writable area
    me.First.Row := me.First.Row + 2;
  end if;
end title;

-- Draws top line border.
-- Draws the two side lines.
-- Draws the bottom line of the border.
-- Make the Window smaller by one character on each side.
procedure borders (
  me : IN OUT Window;
  Corner : IN Character;
  Down : IN Character;
  Across : IN Character) is
begin

  -- Put top line of border
  Screen.MoveCursor(me.First);
  Text_IO.Put(Corner);
  for Count in me.First.Column+1 .. me.Last.Column-1 loop
    Text_IO.Put(Across);
  end loop;
  Text_IO.Put(Corner);
```

```
      -- Put the two side lines
      for Count in me.First.Row+1 .. me.Last.Row-1 loop
        Screen.MoveCursor((Row => Count,Column => me.First.Column));
        Text_IO.Put(Down);
        Screen.MoveCursor((Row => Count,Column => me.Last.Column));
        Text_IO.Put(Down);
      end loop;

      -- Put the bottom line of the border
      Screen.MoveCursor((Row => me.Last.Row,Column => me.First.Column)
me.First.Column));
      Text_IO.Put(corner);
      for Count in me.First.Column+1 .. me.Last.Column-1 loop
        Text_IO.Put (Across);
      end loop;
      Text_IO.Put(Corner);

      -- Make the Window smaller by one character on each side
      me.First := (Row => me.First.Row+1,Column => me.First.Column+1);
      me.Last := (Row => me.Last.Row-1,Column => me.Last.Column-1);
      me.Current := me.First;
    end borders;

    -- Cursor position passed as parameter is relative to window bound
boundaries.
    procedure movecursor (
      me : IN OUT Window;
      P : IN Screen.Position) is
    -- Relative to writable Window boundaries, of course
    begin
      me.Current.Row := me.First.Row + P.Row;
      me.Current.Column := me.First.Column + P.Column;
    end movecursor;

    -- If at end of current line then move to next line.
    -- If at beginning of current line then erase the entire line.
    -- Writes given character.
    procedure put (
      me : IN OUT Window;
      Ch : IN Character) is
    begin

      -- If at end of current line, move to next line
      if me.Current.Column > me.Last.Column then
        if me.Current.Row = me.Last.Row then
          me.Current.Row := me.First.Row;
        else
          me.Current.Row := me.Current.Row + 1;
        end if;
        me.Current.Column := me.First.Column;
      end if;

      -- If at First char, erase line
      if me.Current.Column = me.First.Column then
        EraseToEndOfLine(me);
      end if;

      Screen.MoveCursor(To => me.Current);

      -- here is where we actually write the character!
      Text_IO.Put(Ch);
      me.Current.Column := me.Current.Column + 1;
    end put;

    -- Uses put#1 to write each character of the string.
    procedure put (
      me : IN OUT Window;
      S : IN String) is
    begin
      for Count in S'Range loop
```

```
        Put(me, S (Count));
      end loop;
    end put;

    -- If cursor is at beginning of a line then first erase this line.
    -- If cursor is on last line then put it on first line.
    -- Else put it on next line.
    procedure new_line (
      me : IN OUT Window) is
    begin
      if me.Current.Column = 1 then
        EraseToEndOfLine(me);
      end if;
      if me.Current.Row = me.Last.Row then
        me.Current.Row := me.First.Row;
      else
        me.Current.Row := me.Current.Row + 1;
      end if;
      me.Current.Column := me.First.Column;
    end new_line;

    -- Puts blank characters from current cursor position to the end
  of current
    --  line.
    -- Current cursor position remains unchanged.
    procedure erasetoendofline (
      me : IN OUT Window) is
    begin
      Screen.MoveCursor (me.Current);
        for Count in me.Current.Column .. me.Last.Column loop
          Text_IO.Put (' ');
        end loop;
      Screen.MoveCursor (me.Current);
    end erasetoendofline;

  begin
    Text_IO.New_Line;
    Screen.ClearScreen;
    Text_IO.New_Line;

  end windows;
```

END windows

CLASS phil IS

ACTIVE

pragmas

```
PRAGMA discriminant
  (type_name => Philosopher,
   attribute_name => --|My_ID|--)
```

spec

```
-- Dining Philosophers - Ada 95 edition
--
-- Philosopher is an Ada 95 task type with discriminant.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  required interface :
--   Required OPERATION :
--     OPERATION : report_state of object : room
--     OPERATION : get_stick of object : room
--     OPERATION : pick_up of object : chop
--     OPERATION : put_down of object : chop
--   Required EXCEPTION : NONE
--   Required TYPE :
--     TYPE : Unique_DNA_Codes of object : society
--     TYPE : Positive of object : standard
--     TYPE : Duration of object : standard
--   Required CONSTANT : NONE
--   Required DATA : NONE

--  visibility on required modules :
with society;
use type society.Unique_DNA_Codes;

package phil is

  task type Philosopher (My_ID : society.Unique_DNA_Codes)  is

    entry start_eating (
      Who_Am_I : IN Society.Unique_DNA_Codes;
      Chopstick1 : IN Positive;
      Chopstick2 : IN Positive);
  end Philosopher;

  type Philosopher_Ptr is access all Philosopher;

  type States is (
    Breathing, Thinking, Eating, Done_Eating,
    Got_One_Stick, Got_Other_Stick, Dying);

  procedure start_eating (
    me : IN OUT Philosopher;
    Who_Am_I : IN Society.Unique_DNA_Codes;
    Chopstick1 : IN Positive;
    Chopstick2 : IN Positive);

end phil;
```

body

```
-- Dining Philosophers - Ada 95 edition
--
-- Philosopher is an Ada 95 task type with discriminant.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  visibility on required modules :
with room;
with chop;
with Random_Generic;

--  visibility on objects required by nested operation bodies :

package body phil is

  subtype Think_Times is Positive range 1..8;

  subtype Meal_Times is Positive range 1..10;

  subtype Life_Time is Positive range 1 .. 5;

  package Think_Length is new Random_Generic(
    Result_Subtype => Think_Times);

  package Meal_Length is new Random_Generic(
    Result_Subtype => Meal_Times);

  procedure OPCS_start_eating (
    Who_Am_I : IN Society.Unique_DNA_Codes;
    Chopstick1 : IN Positive;
    Chopstick2 : IN Positive) is
    Meal_Time : Meal_Times;
    Think_Time : Think_Times;

  begin

    Room.Report_State(Who_Am_I, Breathing);

    for Meal in Life_Time loop

      Room.Get_Stick(Chopstick1).all.Pick_Up;
      Room.Report_State(Who_Am_I,Got_One_Stick,Chopstick1);

      Room.Get_Stick(Chopstick2).all.Pick_Up;
      Room.Report_State(Who_Am_I,Got_Other_Stick,Chopstick2);

      Meal_Time := Meal_Length.Random_Value;
      Room.Report_State(Who_Am_I,Eating,Meal_Time,Meal);

      delay Duration(Meal_Time);

      Room.Report_State(Who_Am_I,Done_Eating);

      Room.Get_Stick(Chopstick1).all.Put_Down;
      Room.Get_Stick(Chopstick2).all.Put_Down;

      Think_Time := Think_Length.Random_Value;
      Room.Report_State(Who_Am_I,Thinking,Think_Time);

      delay Duration(Think_Time);

    end loop;

    Room.Report_State(Who_Am_I,Dying);
  end OPCS_start_eating;
```

```
  task body Philosopher is


    start_eating_Who_Am_I : Society.Unique_DNA_Codes;
    start_eating_Chopstick1 : Positive;
    start_eating_Chopstick2 : Positive;
  begin
    loop
      select
        -- LSER
        accept start_eating (
          Who_Am_I : IN Society.Unique_DNA_Codes;
          Chopstick1 : IN Positive;
          Chopstick2 : IN Positive) do
          start_eating_Who_Am_I := Who_Am_I;
          start_eating_Chopstick1 := Chopstick1;
          start_eating_Chopstick2 := Chopstick2;
        end start_eating;
        begin
          OPCS_start_eating (start_eating_Who_Am_I, start_eating_Cho
start_eating_Chopstick1, start_eating_Chopstick2 );
        exception
          when others =>
            raise;
        end;
      or
        terminate;
      end select;
    end loop;

  end Philosopher;

  procedure start_eating (
    me : IN OUT Philosopher;
    Who_Am_I : IN Society.Unique_DNA_Codes;
    Chopstick1 : IN Positive;
    Chopstick2 : IN Positive) is
  begin
    me.start_eating (Who_Am_I, Chopstick1, Chopstick2 );
  end start_eating;


end phil;
```

END phil

OBJECT society IS

PASSIVE

spec

```
-- Dining Philosophers - Ada 95 edition
--
-- Society gives unique ID's to people, and register their names.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  required interface :
--   Required OPERATION : NONE
--   Required EXCEPTION : NONE
--   Required TYPE :
--     TYPE : Positive of object : standard
--     TYPE : String of object : standard
--   Required CONSTANT : NONE
--   Required DATA : NONE

--  visibility on required modules :

package society is

  subtype Unique_DNA_Codes is Positive range 1..5;

  function get_name (
    Code : IN Unique_DNA_Codes)
     return String;

end society;
```

body

```
-- Dining Philosophers - Ada 95 edition
--
-- Society gives unique ID's to people, and register their names.
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  visibility on required modules :

--  visibility on objects required by nested operation bodies :

package body society is

  Name_Register : array(Unique_DNA_Codes) of String(1..18) :=
    ("Philosopher #1    ",
     "Philosopher #2    ",
     "Philosopher #3    ",
     "Philosopher #4    ",
     "Philosopher #5    ");

  function get_name (
    Code : IN Unique_DNA_Codes)
     return String is
  begin
    return Name_Register(Code);
  end get_name;


end society;
```

END society

CLASS chop IS

PASSIVE

spec

```
-- Dining Philosophers - Ada 95 edition
--
-- Chopstick is an Ada95 protected type
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  required interface :
--   Required OPERATION : NONE
--   Required EXCEPTION : NONE
--   Required TYPE : NONE
--   Required CONSTANT : NONE
--   Required DATA : NONE

--  visibility on required modules :

package chop is

  protected type Stick is

    entry pick_up;

    procedure put_down;

  private
    In_Use : Boolean := false;
  end Stick;

  type Stick_Ptr is access all Stick;

  procedure pick_up (
    me : IN OUT stick);

  procedure put_down (
    me : IN OUT stick);

end chop;
```

body

```
-- Dining Philosophers - Ada 95 edition
--
-- Chopstick is an Ada95 protected type
--
-- Michael B. Feldman, The George Washington University, July 1995.
-- HOOD version by Pierre Dissaux, TNI, June 1998.


--  visibility on required modules :

--  visibility on objects required by nested operation bodies :

package body chop is

  protected body Stick is

    entry pick_up when not In_Use is
    begin
      In_Use := True;
    end pick_up;
```

```
      procedure put_down is
      begin
        In_Use := False;
      end put_down;

    end Stick;

    procedure pick_up (
      me : IN OUT stick) is
    begin
      me.pick_up;
    end pick_up;

    procedure put_down (
      me : IN OUT stick) is
    begin
      me.put_down;
    end put_down;


  end chop;
```

END chop