

Normas

Dispones de **dos horas** para realizar estas dos preguntas, que deberás entregar en hojas separadas.

Bolsa en red — Implementación

[2 horas, 7 puntos]

Una posible solución se muestra en el grafo de procesos y recursos:

Código de las tareas

```
Gestor : Bolsa;
...
task type Vendedor;
task body Vendedor is
  Id      : Natural;
  Titulo  : TTitulo;
begin
  <<obtener Titulo>>
  Gestor.Anunciar (Titulo,Id);
  Gestor.Continuar (Id);
end Vendedor;
```

```
task type Comprador;
task body Comprador is
  Id      : Natural;
  Hecho   : Boolean;
  Riesgo  : TRiesgo;
  Cliente : TCliente;
begin
  Id := 0;
  Hecho := False;
  <<obtener Riesgo y Cliente>>
  loop
    Gestor.Buscar(Riesgo,Titulo,Id);
    if Interesa(Cliente,Titulo) then
      Gestor.Adquirir(Id,Hecho)
    end if;
    exit when Hecho;
  end loop;
  <<comprar>>
end TProductor;
```

Recurso Bolsa

NOTA: En aquellos casos en que una variable v está declarada “de salida” hemos escrito v directamente en vez de v^{sal} .

C-TADSOL Bolsa**OPERACIONES**

ACCIÓN Anunciar: $TBolsa[es] \times TTitulo[e] \times \mathbb{N}[s]$

ACCIÓN Continuar: $TBolsa[es] \times \mathbb{N}[e]$

ACCIÓN Buscar: $TBolsa[e] \times TRiesgo[e] \times TTitulo[s] \times \mathbb{N}[es]$

ACCIÓN Adquirir: $TBolsa[es] \times \mathbb{N}[e] \times \mathbb{B}[s]$

PROTOCOLOS: Vender: Anunciar(b, t, id); Continuar(b, id)

Comprar: Buscar(b, r, t_1, id_1); Buscar(b, r, t_2, id_2); ... ; Buscar(b, r, t_n, id_n); Adquirir(b, id_n)

SEMÁNTICA**DOMINIO:**

TIPO: $TBolsa = (ultid : \mathbb{N} \times anuncios : \text{Conjunto}(TAnuncio))$

TIPO: $TAnuncio = (titulo : TTitulo \times id : \mathbb{N})$

INVARIANTE: $\forall b \in TBolsa \bullet \text{Card}(b.anuncios) \leq K \wedge$

$\forall a_1, a_2 \in b.anuncios \bullet a_1.id = a_2.id \Rightarrow a_1.titulo = a_2.titulo$

INICIAL(b): $b = (0, \emptyset)$

CPRE: Hay espacio en el tablón

CPRE: $\text{Card}(b.anuncios) < K$

Anunciar (b , $titulo$, id)

POST: Se añade el nuevo título al tablón y se asigna el identificador más reciente

POST: $b^{\text{ent}} = (ultid, anuncios) \wedge id = ultid + 1 \wedge b^{\text{sal}} = (id, anuncios \cup \{(titulo, id)\})$

CPRE: La oferta ha sido aceptada por un comprador

CPRE: $\neg \exists t \in TTitulo \bullet (t, id) \in b.anuncios$

Continuar (b , id)

POST: $b^{\text{sal}} = b^{\text{ent}}$

CPRE: Hay un título con un riesgo menor que el pedido y que aún no ha sido retornado al comprador

CPRE: $\exists i \in \mathbb{N} \exists t \in TTitulo \bullet (t, i) \in b.anuncios \wedge \text{Nivel_Riesgo}(t) \leq riesgo \wedge id < i$

Buscar (b , $riesgo$, $titulo$, id)

POST: $b^{\text{sal}} = b^{\text{ent}} \wedge$

$id^{\text{sal}} = \min i \in \mathbb{N} \bullet \exists t \in TTitulo \bullet (t, i) \in b.anuncios \wedge \text{Nivel_Riesgo}(t) \leq riesgo \wedge id^{\text{ent}} < i \wedge$

$(titulo, id^{\text{sal}}) \in b.anuncios$

CPRE: true

Adquirir (b , id , $hecho$)

POST: $\exists t \in TTitulo \bullet (t, id) \in b.anuncios \Rightarrow hecho \wedge b^{\text{sal}} = (b^{\text{ent}}.ultid, b^{\text{ent}}.anuncios - \{(t, id)\}) \wedge$

$\neg \exists t \in TTitulo \bullet (t, id) \in b.anuncios \Rightarrow \neg hecho \wedge b^{\text{sal}} = b^{\text{ent}}$

Se pide:

2. Implementación en Ada95 mediante objetos protegidos del recurso *Bolsa*, incluyendo el código del tipo protegido. **[3.5 puntos]**

Solución propuesta:

```
-- Hay un máximo fijado de vendedores y compradores
Num_Vendedores: constant Positive := 9;
Num_Compradores: constant Positive := 9;

-- Hay también un máximo de elementos que caben en el tablón.
Max_Tablon: constant Positive := 100;

subtype Rango_Vendedores is Positive range 1..Num_Vendedores;
subtype Rango_Compradores is Positive range 1..Num_Compradores;
subtype Tipo_Id_Anuncio is Natural;

-- El tablón puede simularse perfectamente con una tabla donde la
-- clave es el identificador (único) que el recurso asigna a cada
-- anuncio (el invariante del recurso es exactamente igual que el
-- de una tabla). Puede utilizarse también una implementación de
-- conjuntos.

package Tipo_Tablon is
    new Tablas(Tipo_Clave      => Tipo_Id_Anuncio,
              Tipo_Informacion => Tipo_Titulo,
              Tamano          => Max_Tablon,
              "<"              => "<");
use Tipo_Tablon;

-- Los argumentos de las operaciones cuyas precondiciones dependen
-- de sus argumentos de entrada se guardan aquí. Es un esquema
-- general que tiene espacio tanto para las llamadas a Continuar
-- como para las llamadas a Buscar.

Type Tipo_Peticion is
    record
        Titulo: Tipo_Titulo;
        Id: Tipo_Id_Anuncio;
        Riesgo: Tipo_Riesgo;
        Ocupado: Boolean := False;
    end record;

type Tipo_Peticiones_Continuar is
    array(Rango_Vendedores) of Tipo_Peticion;

type Tipo_Peticiones_Buscar is
    array(Rango_Compradores) of Tipo_Peticion;

protected type Tipo_Bolsa is

    entry Anunciar (Titulo : in Tipo_Titulo;
                  Id : out Tipo_Id_Anuncio);

    entry Continuar (Id : in Tipo_Id_Anuncio);
```

```

    entry Buscar (Riesgo : in      Tipo_Riesgo;
                  Titulo  :      out Tipo_Titulo;
                  Id       : in out Tipo_Id_Anuncio);

    entry Adquirir (Id       : in      Tipo_Id_Anuncio;
                   Hecho    :      out Boolean);
private
    Anuncios: Tipo_Tablon.Tipo_Tabla; -- El tablón con todos los anuncios
    Id_Actual: Tipo_Id_Anuncio := 0; -- Contador para el siguiente
                                   -- identificador de anuncio.

    -- Vectores que guardan los datos de las llamadas que deben
    -- quedar suspendidas.
    Aplazados_Cont: Tipo_Peticiones_Continuar;
    Aplazados_Buscar: Tipo_Peticiones_Buscar;

    -- Definiciones de las familias de "entries" para suspender las
    -- llamadas que no pueden ser atendidas inmediatamente.
    entry Continuar_Apl (Rango_Vendedores)
        (Id : in Tipo_Id_Anuncio);

    entry Buscar_Apl (RangoCompradores)
        (Riesgo : in      Tipo_Riesgo;
         Titulo  :      out Tipo_Titulo;
         Id       : in out Tipo_Id_Anuncio);
end Tipo_Bolsa;

protected body Tipo_Bolsa is

    -- El anuncio se limita a añadir el título, indexado por su
    -- identificador (único) en el tablón y a aumentar el
    -- identificador para el nuevo título.
    entry Anunciar (Titulo : in      Tipo_Titulo;
                   Id       :      out Tipo_Id_Anuncio)
    when not Esta_Llena(Anuncios) is
    begin
        Id_Actual := Id_Actual + 1;
        Anadir(Anuncios, Id_Actual, Titulo);
        Id := Id_Actual;
    end Anunciar;

    -- 'Continuar' depende de los parámetros de entrada. Buscamos un
    -- lugar para esta llamada y guardamos los datos. Por
    -- definición debe haber un lugar libre para ella, porque el
    -- tamaño del vector es igual al número de procesos vendedores.
    -- Lo mismo vale para los procesos compradores.
    entry Continuar (Id : in Tipo_Id_Anuncio)
    when True is
        Id_Vendedor: Rango_Vendedores := 1;
    begin
        while Aplazados_Cont(Id_Vendedor).Ocupado loop
            Id_Vendedor := Id_Vendedor + 1;
        end loop;
        Aplazados_Cont(Id_Vendedor).Ocupado := True;
        Aplazados_Cont(Id_Vendedor).Id := Id;
        requeue Continuar_Apl(Id_Vendedor);
    end Continuar;

```

```

-- Bloqueo hasta que se haya vendido el título que hemos
-- dejado. Marcamos como liberado el lugar que estaba siendo
-- utilizado para almacenar los datos de esta llamada.
entry Continuar_Apl (for I in Rango_Vendedores)
  (Id : in Tipo_Id_Anuncio)
when not Esta(Anuncios, Aplazados_Cont(I).Id) is
begin
  Aplazados_Cont(I).Ocupado := False;
end Continuar_Apl;

-- Al igual que 'Continuar', Buscar depende de sus datos de
-- entrada. Usamos la misma técnica; el mismo supuesto sobre
-- el número de procesos y el tamaño del vector son aplicables.
entry Buscar (Riesgo : in Tipo_Riesgo;
               Titulo : out Tipo_Titulo;
               Id : in out Tipo_Id_Anuncio)
when True is
  Id_Comprador := Rango_Compradores := 1;
begin
  while Aplazados_Buscar(Id_Comprador).Ocupado loop
    Id_Comprador := Id_Comprador + 1;
  end loop;
  Aplazados_Buscar(Id_Comprador).Id := Id;
  Aplazados_Buscar(Id_Comprador).Riesgo := Riesgo;
  Aplazados_Buscar(Id_Comprador).Ocupado := True;
  requeue Buscar_Apl(Id_Comprador);
end Buscar;

-- Esta función nos ayudará a ahorrar código. Suponemos que la
-- tabla que estamos utilizando tiene primitivas para recorrer
-- las claves una a una. De no ser así se pueden guardar los
-- identificadores asignados a cada título (que son claves de la
-- tabla) en una estructura separadas (por ejemplo, un vector) o
-- incluso guardarlos directamente en la estructura de datos
-- utilizada para almacenar los parámetros de entrada de
-- Continuar (porque hay una correspondencia uno a uno entre los
-- procesos que quieren vender y los títulos en la bolsa).
-- Opcionalmente siempre es posible realizar una implementación
-- de una tabla in situ, que no es complicada, pero que necesita
-- algo más de tiempo de implementación.

-- Esta función devuelve el menor identificador de título cuyo
-- riesgo es menor o igual que el indicado, y que es mayor que
-- el identificador pasado como argumento. Si no existe un
-- título con estas características, devuelve un identificador
-- mayor que el mayor existente (que está siempre guardado en
-- Id_Actual).
function Menor_Id (Id : in Tipo_Id_Anuncio;
                   Riesgo : in Tipo_Riesgo)
  return Tipo_Id_Anuncio is
  Menor_Id : Tipo_Id_Anuncio := Id_Actual + 1;
  Cursor : Tipo_Cursor := Principio(Anuncios);
  Id_Titulo : Tipo_Id_Anuncio;
  Titulo : Tipo_Titulo;
begin
  while (not Es_Fin(Anuncios, Cursor)) loop

```

```

    Id_Titulo := Recupera_Clave(Anuncios, Cursor);
    Titulo    := Recuperar(Anuncios, Id_Titulo);
    if (Nivel_Riesgo(Titulo) <= Riesgo and
        Id_Titulo > Id and
        Id_Titulo < Menor_Id) then
        Menor_Id := Id_Titulo;
    end if;
    Cursor := Siguiente(Anuncios, Cursor);
end loop;
return Menor_Id;
end Menor_Id;

-- Las llamadas suspendidas se bloquean aquí hasta que haya un
-- título que cumpla los requisitos exigidos.
entry Buscar_Apl (for I in RangoCompradores)
(Riesgo : in Tipo_Riesgo;
 Titulo : out Tipo_Titulo;
 Id      : in out Tipo_Id_Anuncio)
when Menor_Id(Aplazados_Buscar(I).Id,
    Aplazados_Buscar(I).Riesgo) <= Id_Actual
is
begin
    Id := Menor_Id(Aplazados_Buscar(I).Id,
        Aplazados_Buscar(I).Riesgo);
    Titulo := Recuperar(Anuncios, Id);
    Aplazados_Buscar(I).Ocupado := False;
end Buscar_Apl;

-- 'Adquirir' recibe el identificador requerido, pero puede que el
-- título pedido ya no esté disponible. 'Hecho' refleja si se
-- ha podido adquirir o no el título.
entry Adquirir (Id : in Tipo_Id_Anuncio;
    Hecho : out boolean)
when True is
begin
    Hecho := Esta(Anuncios, Id);
    if Hecho then
        Borrar(Anuncios, Id);
    end if;
end Adquirir;

end Tipo_Bolsa;
```

3. Implementación en Ada95 mediante *rendez-vous*.**[3.5 puntos]****Solución propuesta:**

```

[...]  

subtype Rango_Vendedores is Positive range 1..Num_Vendedores;  

subtype Rango_Compradores is Positive range 1..Num_Compradores;  

subtype Tipo_Id_Anuncio is Natural;  

type Tipo_Anuncio is  

  record  

    Id : Tipo_Id_Anuncio;  

    Tit : Tipo_Titulo;  

  end record;  

  

-- Instanciamos Channel. Vamos a usarlos para enviar un  

-- título (Buscar) y una confirmación (Continuar).  

package ChannelT is new Channel (Tipo_Anuncio);  

type CanalT is new ChannelT.Channel;  

type CanalT_P is access all CanalT;  

package ChannelC is new Channel (Boolean);  

type CanalC is new ChannelC.Channel;  

type CanalC_P is access all CanalC;  

  

-- El tablón puede simularse perfectamente con una tabla donde la  

-- clave es el identificador (nico) que el recurso asigna a cada  

-- anuncio (el invariante del recurso es exactamente igual que el  

-- de una tabla). Puede utilizarse también  $\frac{1}{2}$  una implementación de  

-- conjuntos.  

  

package Tipo_Tablon is  

  new Tablas(Tipo_Clave => Tipo_Id_Anuncio,  

    Tipo_Informacion => Tipo_Titulo,  

    Tamano => Max_Tablon,  

    "<" => "<");  

use Tipo_Tablon;  

  

-- Los argumentos de las operaciones cuyas precondiciones dependen  

-- de sus argumentos de entrada se guardan aquí.  

  

type Tipo_Peticion_Continuar is  

  record  

    Id : Tipo_Id_Anuncio;  

    Ocupado : Boolean := False;  

    Ack : CanalC_P;  

  end record;  

  

type Tipo_Peticion_Buscar is  

  record  

    Id : Tipo_Id_Anuncio;  

    Riesgo : Tipo_Riesgo;  

    Ocupado : Boolean := False;  

    Resp : CanalT_P;  

  end record;  

  

type Tipo_Peticiones_Continuar is  

  array(Rango_Vendedores) of Tipo_Peticion_Continuar;

```

```

type Tipo_Peticiones_Buscar is
  array(Rango_Compradores) of Tipo_Peticion_Buscar;

-- Las operaciones del recurso cuyas CPREs dependen
-- de parámetros de entrada reciben, además, un canal de respuesta
-- para bloqueo en 2 fases. Este canal identifica a cada cliente
-- (comprador o vendedor) y puede servir también para devolver
-- un valor de salida (op. Buscar)

task type BolsaServer is
  entry Anunciar (Titulo : in      Tipo_Titulo;
                 Id      :         out Tipo_Id_Anuncio);
  entry Continuar (Id      : in      Tipo_Id_Anuncio;
                  Ack      : in      CanalC_P);
  entry Buscar (Riesgo : in      Tipo_Riesgo;
                Id      : in      Tipo_Id_Anuncio;
                Resp     : in      CanalT_P);
  entry Adquirir (Id      : in      Tipo_Id_Anuncio;
                 Hecho    :         out Boolean);
end BolsaServer;

-----
task body BolsaServer is

  Anuncios: Tipo_Tablon.Tipo_Tabla; -- El tablón con todos los anuncios
  Id_Actual: Tipo_Id_Anuncio := 0; -- Contador para el siguiente
                                   -- identificador de anuncio.

  Id_Vendedor : Rango_Vendedores;
  Id_Comprador : Rango_Compradores; -- Usados para recorrer los vectores

  Id_Menor : Tipo_Id_Anuncio;
  Respuesta : Tipo_Anuncio; -- Para contestar a los que Buscan

  -- Vectores que guardan los datos de las llamadas que deben
  -- quedar suspendidas.
  Aplazados_Cont: Tipo_Peticiones_Continuar;
  Aplazados_Buscar: Tipo_Peticiones_Buscar;

  -- Esta función devuelve el menor identificador de título cuyo
  -- riesgo es menor o igual que el indicado, y que es mayor que
  -- el identificador pasado como argumento. Si no existe un
  -- título con estas características, devuelve un identificador
  -- mayor que el mayor existente (que está siempre guardado en
  -- Id_Actual).
  function Menor_Id (Id      : in Tipo_Id_Anuncio;
                    Riesgo : in Tipo_Riesgo)
    return Tipo_Id_Anuncio is
    Menor_Id : Tipo_Id_Anuncio := Id_Actual + 1;
    Cursor: Tipo_Cursor := Principio(Anuncios);
    Id_Titulo: Tipo_Id_Anuncio;
    Titulo: Tipo_Titulo;
  begin
    while (not Es_Fin(Anuncios, Cursor)) loop
      Id_Titulo := Recupera_Clave(Anuncios, Cursor);
      Titulo := Recuperar(Anuncios, Id_Titulo);
      if (Nivel_Riesgo(Titulo) <= Riesgo and
        Id_Titulo > Id and
        Id_Titulo < Menor_Id) then
        Menor_Id := Id_Titulo;
    end loop

```

```

        end if ;
        Cursor := Siguiente(Anuncios, Cursor);
    end loop;
    return Menor_Id;
end Menor_Id;

begin
loop
select
-- El anuncio se limita a añadir el título, indexado por su
-- identificador (único) en el tablón y a aumentar el
-- identificador para el nuevo título.
when not Esta_Llena(Anuncios) =>
    accept Anunciar (Titulo : in      Tipo_Titulo;
                    Id      :      out Tipo_Id_Anuncio) do
        Id_Actual := Id_Actual + 1;
        Anadir(Anuncios, Id_Actual, Titulo);
        Id := Id_Actual;
    end Anunciar;
or
-- 'Continuar' depende de los parámetros de entrada.
-- Buscamos un lugar para esta llamada y guardamos los
-- datos. Por definición debe haber un lugar libre para
-- ella, porque el tamaño del vector es igual al número de
-- procesos vendedores. Lo mismo vale para los procesos
-- compradores.
when True =>
    accept Continuar (Id : in Tipo_Id_Anuncio;
                     Ack : in CanalC_P) do
        Id_Vendedor := 1;
        while Aplazados_Cont(Id_Vendedor).Ocupado loop
            Id_Vendedor := Id_Vendedor + 1;
        end loop;
        Aplazados_Cont(Id_Vendedor).Ocupado := True;
        Aplazados_Cont(Id_Vendedor).Id := Id;
        Aplazados_Cont(Id_Vendedor).Ack := Ack;
    end Continuar;
or
-- Al igual que 'Continuar', Buscar depende de sus datos de
-- entrada. Usamos la misma técnica; el mismo supuesto sobre
-- el número de procesos y el tamaño del vector son
-- aplicables.
when True =>
    accept Buscar (Riesgo : in Tipo_Riesgo;
                  Id      : in Tipo_Id_Anuncio;
                  Resp    : in CanalT_P) do
        Id_Comprador := 1;
        while Aplazados_Buscar(Id_Comprador).Ocupado loop
            Id_Comprador := Id_Comprador + 1;
        end loop;
        Aplazados_Buscar(Id_Comprador).Id := Id;
        Aplazados_Buscar(Id_Comprador).Riesgo := Riesgo;
        Aplazados_Buscar(Id_Comprador).Ocupado := True;
        Aplazados_Buscar(Id_Comprador).Resp := Resp;
    end Buscar;
or
-- 'Adquirir' recibe el identificador requerido, pero puede que el
-- título pedido ya no esté disponible. 'Hecho' refleja si se

```

```

-- ha podido adquirir o no el título.
when True =>
    accept Adquirir (Id      : in      Tipo_Id_Anuncio;
                    Hecho   :      out boolean) do
        Hecho := Esta(Anuncios, Id);
        if Hecho then
            Borrar(Anuncios, Id);
        end if;
    end Adquirir;
end select;

-- =====
-- A partir de aquí se ejecuta el código de desbloqueo de
-- operaciones aplazadas.
-- =====

-- Buscamos un vendedor pendiente de Continuar
for Id_Vendedor in Rango_Vendedores loop
    if Aplazados_Cont(Id_Vendedor).Ocupado
        and then not Esta(Anuncios, Aplazados_Cont(Id_Vendedor).Id)
    then
        Aplazados_Cont(Id_Vendedor).Ack.Send(True);
        Aplazados_Cont(Id_Vendedor).Ocupado := False;
    end if;
end loop;

-- Buscamos un comprador pendiente de encontrar un título
-- adecuado a sus necesidades

-- El desbloqueo propiamente dicho
for Id_Comprador in Rango_Compradores loop
    if Aplazados_Buscar(Id_Comprador).Ocupado
    then
        Id_Menor :=
            Menor_Id(Aplazados_Buscar(Id_Comprador).Id,
                    Aplazados_Buscar(Id_Comprador).Riesgo);
        if Id_Menor <= Id_Actual
        then
            Respuesta.Id := Id_Menor;
            Respuesta.Tit := Recuperar(Anuncios, Id_Menor);
            Aplazados_Buscar(Id_Vendedor).Resp.Send(Respuesta);
            Aplazados_Buscar(Id_Vendedor).Ocupado := False;
        end if;
    end if;
end loop;
end BolsaServer;

-- Creamos un gestor
Gestor : BolsaServer;

-- El código de las tareas compradoras y vendedoras se adapta al
-- uso de canales para recibir respuestas/confirmaciones del
-- gestor.
task type T_Vendedor is
    entry GetChannel(Ack : in CanalC_P);
end T_Vendedor;
task body T_Vendedor is
    Id      : Natural;

```

```

    Titulo  : Tipo_Titulo;
    Gen_Tit : Titulo_Aleatorio.Generator;
    Ack_P   : CanalC_P;
    Token   : Boolean;
begin
    Reset(Gen_Tit);
    select
        when True =>
            accept GetChannel(Ack : in CanalC_P) do
                Ack_P := Ack;
            end GetChannel;
    end select;
    loop
        Titulo := Random(Gen_Tit);
        Put_Line("Anunciando titulo " & Tipo_Titulo'Image(Titulo));
        Put_Line("Nivel de riesgo: " &
            Tipo_Riesgo'Image(Nivel_Riesgo(Titulo)));
        Gestor.Anunciar (Titulo, Id);
        Gestor.Continuar (Id, Ack_P);
        Ack_P.Receive(Token);
        Put_Line("Vendido titulo " & Tipo_Titulo'Image(Titulo));
        delay Retraso_Vendedor;
    end loop;
exception
    when E: others =>
        Put_Line("Excepcion en Vendedor: " & Exception_Name(E));
end T_Vendedor;

task type T_Comprador(B: Boolean) is
    entry GetChannel(Resp : in CanalT_P);
end T_Comprador;
task body T_Comprador is

    function Interesa (Cliente : in Tipo_Cliente;
        Titulo : in Tipo_Titulo)
        return Boolean is
    begin
        return True; -- Integer(Cliente) mod 2 = Integer(Titulo) mod 2;
    end Interesa;

    Id      : Natural;
    Hecho   : Boolean;
    Resp_P  : CanalT_P;
    Respuesta : Tipo_Anuncio;
    Titulo  : Tipo_Titulo;
    Cliente : Tipo_Cliente;
    Gen_Cli : Cliente_Aleatorio.Generator;
    Riesgo  : Tipo_Riesgo;
    Gen_Riesgo : Riesgo_Aleatorio.Generator;
begin
    Reset(Gen_Cli);
    Reset(Gen_Riesgo);
    select
        when True =>
            accept GetChannel(Resp : in CanalT_P) do
                Resp_P := Resp;
            end GetChannel;

```

```

    end select;
    loop
        Id := 0;
        Hecho := False;
        Cliente := Random(Gen_Cli);
        if B then
            Riesgo := Random(Gen_Riesgo);
        else
            Riesgo := Alto;
        end if;
        loop
            Put_Line("Buscando_riesgo_" & Tipo_Riesgo'Image(Riesgo));
            Gestor.Buscar(Riesgo, Id, Resp_P);
            Resp_P.Receive(Respuesta);
            Id := Respuesta.Id;
            Titulo := Respuesta.Tit;
            if Interesa(Cliente, Titulo) then
                Gestor.Adquirir(Id, Hecho);
            end if;
            exit when Hecho;
        end loop;
        Put_Line("Comprado_titulo_" & Tipo_Titulo'Image(Titulo));
        delay Retraso_Comprador;
    end loop;
exception
    when E: others =>
        Put_Line("Excepcion_en_Comprador:_" & Exception_Name(E));
end T_Comprador;

Vendedores      : array (Rango_Vendedores) of T_Vendedor;
Confirmaciones  : array (Rango_Vendedores) of aliased CanalC;
subtype Rango_Compradores_1 is Positive range 1..Num_Compradores - 1;
Compradores     : array (Rango_Compradores_1) of T_Comprador(True);
Respuestas      : array (Rango_Compradores_1) of aliased CanalT;
C: T_Comprador(False); -- This ensures liveness
Respuesta : aliased CanalT;

begin
    -- Repartimos los canales globales a cada tarea
    for I in Rango_Vendedores loop
        Vendedores(I).GetChannel(Confirmaciones(I)'Access);
    end loop;
    for I in Rango_Compradores_1 loop
        Compradores(I).GetChannel(Respuestas(I)'Access);
    end loop;
    C.GetChannel(Respuesta'Access);
end Bolsa_RV;

```