```
-------------------------------------------------------------------------------
-- NOM DU CSU (pricipal)            : Exemples.adb
-- AUTEUR DU CSU                    : Pascal Pignard
-- VERSION DU CSU                   : 1.3a
-- DATE DE LA DERNIERE MISE A JOUR  : 15 mars 2013
-- ROLE DU CSU                      : Exemples illustrant la traduction
--                                    du langage Java en Ada avec JVM-GNAT
-- FONCTIONS EXPORTEES DU CSU       :
--
--
-- FONCTIONS LOCALES DU CSU         :
--
--
-- NOTES                            : Ada 2005, UTF8
--  Basé sur les exemples du livre "Introduction à Java"
--  Patrick Niemeyer, Jonathan Knudsen
--  O'Reilly 2000
--
-- COPYRIGHT                        : (c) Pascal Pignard 2010-2013
-- LICENCE                          : CeCILL V2 (http://www.cecill.info)
-- CONTACT                          : http://blady.pagesperso-orange.fr
-------------------------------------------------------------------------------

with Ada.Text_IO;      use Ada.Text_IO;
with java;             use java;
with java.Lang.Object; use java.Lang.Object;
with java.Lang.String; use java.Lang.String;
with java.Lang.Math;   use java.Lang.Math;

procedure Exemples is

   --  2) Les équivalences Ada aux structures Java

   --   a) Le code source
   --  JVM-GNAT utilise par défaut le codage ISO 8859-1 (Latin-1).
   --  L'option "-gnatW8" permet l'utilisation du codage UTF-8.

   --   b) Les commentaires
   --  // commentaire d'une ligne
   --  est traduit par :
   -- commentaire d'une ligne
   --
   --  /* commentaire
   --       sur
   --           plusieurs lignes */
   --  est traduit par :
   -- commentaire
   --       sur
   --           plusieurs lignes

   --   c) Les types de bases
   I  : java.int  := 1230; -- décimal
   J  : java.int  := 8#1230#; -- octal équivalent à 01230
   L  : java.int  := 16#1230#; -- hexadécimal équivalent à 0x1230
   A  : java.char := 'a';
   NL : java.char := java.char'Val (10); -- équivalent à '\n'
   U  : java.char := java.char'Val (16#263a#); -- équivalent à '\u263a'

   --   d) Les chaînes de caractères
   --  String S = "une chaîne";
   --  est traduit par :
```

```
   S : java.Lang.String.Ref := +"une chaîne";
   -- '+' effectue la conversion et la création de l'objet

   --   h) Les tableaux
   --  int [ ] tableauEntiers;
   --  est traduit par :
   package TabEntiers is
      type TÉlémentsEntiers is array (Natural range <>) of int;
      type Typ (Max : Natural) is new java.Lang.Object.Typ with record
      -- Max est la longeur - 1 !!!
      -- Ada n'autorise pas de calcul avec un discriminant :-(
         Éléments : TÉlémentsEntiers (0 .. Max) := (others => 0);
      end record;
      type Ref is access all Typ'Class;
   end TabEntiers;
   tableauEntiers1 : TabEntiers.Ref;

   --  int [ ] tableauEntiers;
   --  est alors traduit par :
   type TÉlémentsEntiers is array (Natural range <>) of int;
   type TTabEntiers is access all TÉlémentsEntiers;
   tableauEntiers2 : TTabEntiers;

   --  int [ ] altitude = new int [30];
   --  altitude[0] = 99;
   --  altitude[1] = 72;
   --  // altitude[2] == 0
   --  est traduit par :
   procedure test1 is
      type TÉlémentsEntiers is array (Natural range <>) of int;
      type TTabEntiers is access all TÉlémentsEntiers;
      subtype TAltitude is TÉlémentsEntiers (0 .. 29); -- indice fin = longueur - 1
      altitude : TTabEntiers := new TAltitude'(others => 0);
   begin
      altitude (0) := 99;
      altitude (1) := 72;
      -- altitude[2] = 0
   end test1;

   --  String noms [ ] = new String [4];
   --  noms[0] = new String();
   --  noms[1] = "Hello Java with Ada";
   --  noms[2] = unObjet.toString();
   --  // noms[3] == null
   --  est traduit par :
   procedure test2 is
      type TÉlémentsChaines is array (Natural range <>) of java.Lang.String.Ref;
      type TTabChaines is access all TÉlémentsChaines;
      subtype TNoms is TÉlémentsChaines (0 .. 3); -- indice fin = longueur - 1
      noms    : TTabChaines          := new TNoms'(others => null);
      unObjet : java.Lang.Object.Ref := New_Object;
   begin
      noms (0) := java.Lang.String.New_String;
      noms (1) := +"Hello Java with Ada";
      noms (2) := java.Lang.String.Ref (unObjet.ToString);
      -- noms(3) = null
   end test2;

   --  int [ ] premier = {1, 2, 3, 5, 7, 7+4}; // premiers[2] == 3
   --  est traduit par :
   --type TÉléments is array (Natural range <>) of int;
```

```
   --type TTabEntiers is access all TÉléments;
   premier : TTabEntiers := new TÉlémentsEntiers'(1, 2, 3, 5, 7, 7 + 4); -- premier(2) = 3

   --  String [ ] mousquetaires = {"Aramis", "Atos", "Portos"};
   --  int nombre = mousquetaires.length; // nombre == 3
   --  est traduit par :
   type TÉlémentsChaines is array (Natural range <>) of java.Lang.String.Ref;
   type TTabChaines is access all TÉlémentsChaines;
   mousquetaires : TTabChaines := new TÉlémentsChaines'(+"Aramis", +"Atos", +"Portos");
   nombre        : int         := mousquetaires'Length; -- nombre = 3

   --   i) Les tableaux anonymes
   --  Chien pokey = new Chien("gris");
   --  Chat squiggles = new Chat("noir");
   --  Chat jasmine = new Chat("orange");
   --  setAnimaux(new Animaux [ ] {pokey, squiggles, jasmine});
   --  est traduit par :
   procedure test3 is
      package Animaux is
         type Typ is new java.Lang.Object.Typ with null record;
         type Ref is access all Typ'Class;
      end Animaux;
      package Chien is
         type Typ is new Animaux.Typ with null record;
         type Ref is access all Typ'Class;
         function New_Chien (S : java.Lang.String.Ref) return Ref;
      end Chien;
      package body Chien is
         function New_Chien (S : java.Lang.String.Ref) return Ref is
         begin
            return null;
         end New_Chien;
      end Chien;
      package Chat is
         type Typ is new Animaux.Typ with null record;
         type Ref is access all Typ'Class;
         function New_Chat (S : java.Lang.String.Ref) return Ref;
      end Chat;
      package body Chat is
         function New_Chat (S : java.Lang.String.Ref) return Ref is
         begin
            return null;
         end New_Chat;
      end Chat;
      use Chien, Chat;
      pokey     : Chien.Ref := New_Chien (+"gris");
      squiggles : Chat.Ref  := New_Chat (+"noir");
      jasmine   : Chat.Ref  := New_Chat (+"orange");
      type TÉlémentsAnimaux is array (Natural range <>) of Animaux.Ref;
      type TTabAnimaux is access all TÉlémentsAnimaux;
      function New_TTabAnimaux (Éléments : TÉlémentsAnimaux) return TTabAnimaux is
      begin
         return new TÉlémentsAnimaux'(Éléments);
      end New_TTabAnimaux;
      procedure setAnimaux (T : TTabAnimaux) is null;
   begin
      setAnimaux
        (New_TTabAnimaux
            ((Animaux.Ref (pokey), Animaux.Ref (squiggles), Animaux.Ref (jasmine))));
   end test3;
```

```
--   j) Les tableaux multi-dimensionnels
--  pièceÉchecs [ ][ ] jeuÉchecs;
--  jeuÉchecs = new pièceÉchecs [8][8];
--  jeuÉchecs[0][0] = new pièceÉchecs("Tour");
--  jeuÉchecs[1][0] = new pièceÉchecs("Pion");
--  est traduit par :
procedure test4 is
   package pièceÉchecs is
      type Typ is new java.Lang.Object.Typ with null record;
      type Ref is access all Typ'Class;
      function New_PièceÉchecs (S : java.Lang.String.Ref) return Ref;
   end pièceÉchecs;
   package body pièceÉchecs is
      function New_PièceÉchecs (S : java.Lang.String.Ref) return Ref is
      begin
         return null;
      end New_PièceÉchecs;
   end pièceÉchecs;
   use pièceÉchecs;
   type TÉlémentsPièceÉchecs is array (Natural range <>) of pièceÉchecs.Ref;
   type TTabPièceÉchecs is access all TÉlémentsPièceÉchecs;
   type TÉlémentsPièceÉchecs_2 is array (Natural range <>) of TTabPièceÉchecs;
   type TÉchiquier is access all TÉlémentsPièceÉchecs_2;
   subtype TLignes is TÉlémentsPièceÉchecs (0 .. 7); -- indice fin = longueur - 1
   subtype TJeuÉchecs is TÉlémentsPièceÉchecs_2 (0 .. 7); -- indice fin = longueur - 1
   jeuÉchecs : TÉchiquier;
begin
   jeuÉchecs          := new TJeuÉchecs'(others => new TLignes'(others => null));
   jeuÉchecs (0) (0)  := New_PièceÉchecs (+"Tour");
   jeuÉchecs (1) (0)  := New_PièceÉchecs (+"Pion");
end test4;

--  int [ ][ ] triangle = new int [5][ ];
--  for (int i=0; i < triangle.length; i++) {
--    triangle[i] = new int [i+1];
--    for (int j=0; j<i+1; j++)
--      triangle[i][j] = i + j;
--  }
--  est traduit par :
procedure test5 is
   type TÉlémentsEntiers is array (Natural range <>) of int;
   type TTabEntiers is access all TÉlémentsEntiers;
   type TÉlémentsEntiers_2 is array (Natural range <>) of TTabEntiers;
   type TTriangle is access all TÉlémentsEntiers_2;
   subtype TLignes is TÉlémentsEntiers_2 (0 .. 4); -- indice fin = longueur - 1
   triangle : TTriangle := new TLignes'(others => null);
begin
   for i in 0 .. triangle'Length - 1 loop -- indice fin = longueur - 1
      triangle (i) := new TÉlémentsEntiers (0 .. i); -- indice fin = longueur - 1
      for j in 0 .. i loop -- indice fin = longueur - 1
         triangle (i) (j)  := i + j;
      end loop;
   end loop;
end test5;

--  3) Les objets Java
--   a) Les classes
--  class Pendule {
--   static float gravitation = 9.80;
--   float masse;
--   float longueur = 1.0;
```

```
--   int cycles;
--   float position (float date) {
--       ...
--   }
--  }
--  est traduit par :
--  (Un champ statique ou variable de classe est déclaré comme variable du paquetage hors
--  type objet)
procedure Test7 is
   package Pendule is
      gravitation : Float := 9.80;
      type Typ is new java.Lang.Object.Typ with record
         masse    : Float;
         longueur : Float := 1.0;
         cycles   : int;
      end record;
      function position (this : access Typ; date : Float) return Float;
   end Pendule;
   package body Pendule is
      function position (this : access Typ; date : Float) return Float is
      begin
         return date;
      end position;
   end Pendule;
begin
   null;
end Test7;

--  Si un champs est privé alors tous le sont en Ada:
--  (avec une variable de classe constante)
procedure Test8 is
   package Pendule is
      GRAVITATION : constant Float := 9.80;
      type Typ is new java.Lang.Object.Typ with private;
      function position (this : access Typ; date : Float) return Float;
   private
      type Typ is new java.Lang.Object.Typ with record
         masse    : Float;
         longueur : Float := 1.0;
         cycles   : int;
      end record;
   end Pendule;
   package body Pendule is
      function position (this : access Typ; date : Float) return Float is
      begin
         return date;
      end position;
   end Pendule;
begin
   null;
end Test8;

--   b) Les méthodes
--     class Oiseaux {
--   int xPos, yPos;
--   double voler (int x, int y) {
--       double distance = Math.sqrt(x*x + y*y);
--       battreDesAiles (distance);
--       xPos = x;
--       yPos = y;
--       return distance;
```

```
--   }
--  }
--  est traduit par :
procedure Test9 is
   package Oiseaux is
      type Typ is new java.Lang.Object.Typ with record
         xPos, yPos : int;
      end record;
      function voler (this : access Typ; x, y : int) return double;
      procedure battreDesAiles (D : double) is null;
   end Oiseaux;
   package body Oiseaux is
      function voler (this : access Typ; x, y : int) return double is
         distance : double := java.Lang.Math.Sqrt (double (x * x + y * y));
      begin
         battreDesAiles (distance);
         this.xPos := x;
         this.yPos := y;
         return distance;
      end voler;
   end Oiseaux;
begin
   null;
end Test9;

--     class Oiseaux {
--   ...
--   static String [ ] recupérerTypesOiseaux () {
--       String [ ] types;
--       // Créer une liste des types
--   return types;
--   }
--   ...
--  }
--  est traduit par :
procedure Test10 is
   package Oiseaux is
      -- ...
      function recupérerTypesOiseaux return  java.Lang.String.Arr;
      -- ...
   end Oiseaux;
   package body Oiseaux is
      function recupérerTypesOiseaux return  java.Lang.String.Arr is
         types : java.Lang.String.Arr;
      begin
         -- Créer une liste des types
         return types;
      end recupérerTypesOiseaux;
   end Oiseaux;
begin
   null;
end Test10;

--   c) Les constructeurs
--  class Date {
--    long heure;
--    Date() {
--      heure = heureCourante();
--    }
--    Date(String date) {
--      heure = convertitDate(date);
```

```
--    }
--  }
--  Date maintenant = new Date();
--  Date noël = new Date("25 décembre 2010");
--  est traduit par :
procedure Test11 is
   package Date is
      type Typ is new java.Lang.Object.Typ with record
         heure : long;
      end record;
      type Ref is access all Typ'Class;
      function New_Date (This : Ref := null) return Ref;
      function New_Date (date : java.Lang.String.Ref; This : Ref := null) return Ref;
      pragma Java_Constructor (New_Date);
   end Date;
   package body Date is
      function heureCourante return long is
      begin
         return 0;
      end heureCourante;
      function convertitDate (date : java.Lang.String.Ref) return long is
      begin
         return 0;
      end convertitDate;
      function New_Date (This : Ref := null) return Ref is
      -- ligne suivante valide que si Date est un paquetage externe
      --Super : java.Lang.Object.Ref := java.Lang.Object.New_Object (java.Lang.Object.Ref
      --  (This)); -- appel au constructeur parent
      begin
         This.heure := heureCourante;
         return This;
      end New_Date;
      function New_Date (date : java.Lang.String.Ref; This : Ref := null) return Ref is
      -- ligne suivante valide que si Date est un paquetage externe
      --Super : java.Lang.Object.Ref := java.Lang.Object.New_Object (java.Lang.Object.Ref
      --  (This)); -- appel au constructeur parent
      begin
         This.heure := convertitDate (date);
         return This;
      end New_Date;
   end Date;
   maintenant : Date.Ref := Date.New_Date;
   noël       : Date.Ref := Date.New_Date (+"25 décembre 2010");
begin
   null;
end Test11;

--    class Voiture {
--    String modele;
--    int portes;
--    Voiture(String m, int d) {
--      modele = m;
--      portes = d;
--    }
--    Voiture(String m) {
--      this(m, 4);
--    }
--  }
--  est traduit par :
procedure Test12 is
   package Voiture is
```

```
         type Typ is new java.Lang.Object.Typ with record
            modele : java.Lang.String.Ref;
            portes : int;
         end record;
         type Ref is access all Typ'Class;
         function New_Voiture
           (m    : java.Lang.String.Ref;
            d    : int;
            This : Ref := null)
            return Ref;
         function New_Voiture (m : java.Lang.String.Ref; This : Ref := null) return Ref;
         pragma Java_Constructor (New_Voiture);
      end Voiture;
      package body Voiture is
         function New_Voiture
           (m    : java.Lang.String.Ref;
            d    : int;
            This : Ref := null)
            return Ref
         is
         -- ligne suivante valide que si Voiture est un paquetage externe
         --Super : java.Lang.Object.Ref :=
         --  java.Lang.Object.New_Object(Java.Lang.Object.Ref(This));
         -- appel au constructeur parent
         begin
            This.modele := m;
            This.portes := d;
            return This;
         end New_Voiture;
         function New_Voiture (m : java.Lang.String.Ref; This : Ref := null) return Ref is
            Temp : Ref := New_Voiture (m, 4, This);
         begin
            return This;
         end New_Voiture;
      end Voiture;
   begin
      null;
   end Test12;

begin
   Put_Line ("Hello Java World.");
end Exemples;
```