Analyse et programmation langage ADA

Informatique 1ère année

Date : 16/02/2006

Les Fichiers

- **I. Introduction sur les différents types de fichiers**
  - I.1 Présentation des fichiers
- **II. Manupulation des fichiers texte**
  - II.1 Fichier texte
  - II.2 Manipulation des fichiers texte
  - II.3 Creation Ouverture Traitement et Fermeture de fichiers texte
  - II.4 Exemple d'utilisation d'un Fichier texte
  - II.5 Complément sur les fichiers texte
- **III. Manipulation des fichiers binaires**
  - III.1 Fichiers binaires
  - III.2 Creation Ouverture Traitement et Fermeture de fichiers binaires
  - III.3 Fichiers binaires séquentiels
  - III.4 Fichiers binaires à accès direct
  - III.5 Accès aux éléments dans un fichier binaire à accès direct
  - III.6 Remarques sur l'utilisation des fichiers binaires à accès direct
- **IV. Compléments sur l'utilisation des fichiers**
- **V. Exceptions lors de l'utilisation des fichiers**

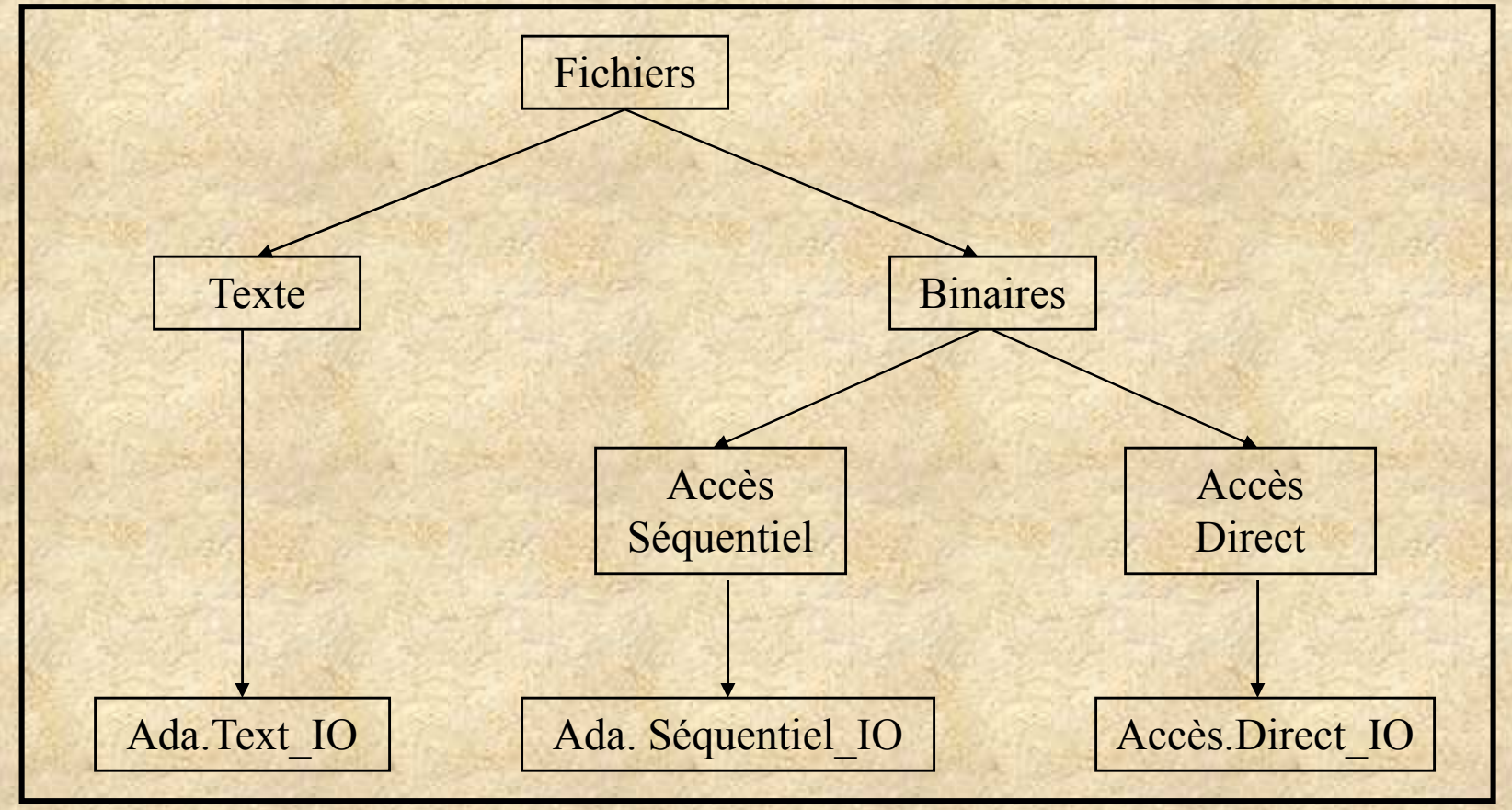
I.1 Présentation des Fichiers

- Un fichier doit exister pour être utilisé, il faudra donc qu'il soit créé au moins une fois.
- Un fichier doit être ouvert avant utilisation et on ne peut ouvrir qu'un fichier qui a été préalablement été créé.
- Un fichier a un début et une fin. On indique la fin de fichier par un symbole de fin de fichier.
- Un fichier peut s'ouvrir :

  (1) en mode : de *lecture*, (2) en mode d'*écriture* ou (3) *les deux*, selon le type de fichier.
- Il existe un curseur pour marquer la position courante dans un fichier.
- À l'ouverture, le curseur est au début du fichier ou à la fin, selon le mode d'ouverture.
- Toutes les opérations de lecture et d'écriture dans un fichier se font par rapport à ce curseur.
- Une lecture se fait à la position courante et le curseur est déplacé sur la prochaine donnée à lire.
- Une écriture se fait à la position courante et le curseur est déplacé sur la prochaine donnée à écrire.
- On ne doit jamais tenté de lire si le curseur est à vis à vis le symbole de fin de fichier.
- Il existe des fonctions pour savoir si le curseur est vis à vis le symbole de fin de fichier.
- On doit fermer un fichier lorsqu'on a terminé de s'en servir.

I.1 Présentation des Fichiers

- Les fichiers sont universellement utilisés pour communiquer des informations d'un programme à un autre et/ou pour communiquer des informations à un utilisateur, et/ou pour sauvegarder des données sur un support.
- Les 2 grandes catégories de fichiers sont les fichier *texte* et les fichiers *binaires*.



I.1 Présentation des Fichiers (Texte)

- Les fichiers ***texte*** sont les plus simples à utiliser.
- Ces fichiers contiennent du texte brut, reparti en ligne.
- Chaque ligne se termine par un marqueur de fin de ligne (FL) constitué de caractères spéciaux (retour chariot, saut de ligne).
- Un fichier de texte se termine par un symbole appelé "fin de fichier"(FF)
- Utilisation
  - Stocker du texte simple
  - Écrire des fichiers de configuration (fichier.ini)
  - Générer des fichiers lisibles par d'autres applications (HTML, éditeur ADA, RTF)

I.1 Présentation des Fichiers

- Les fichiers binaire contiennent des suites binaires donc non lisibles à l'oeil
- Les fichiers ***séquentiels*** ou fichier *typé*, servent à stocker un certain nombre d'*enregistrements* d'un type donné, et ce de façon automatisée.
- Tous les types de données ne peuvent pas être stockés dans ce type de fichier (classes, pointeurs ou tableaux dynamique).
- Ce genre de fichier est surtout utilisé pour stocker une liste des éléments de type record.
  - ✓ Point fort => facilité d'utilisation
  - ✓ Point faible => aucune liberté, pas d'utiliser de pointeurs et tableaux dynamiques
- Les fichiers à ***accès direct*** sont appelé fichiers universels. Ils peuvent contenir n'importe quel type de données (pointeurs, tableau dynamique…).

I.1 Présentation des Fichiers

- Traitement d'un fichier:

- Préparation, ouverture
  - Open / Create
    - ✓ Etablit le lien entre la variable fichier du programme et le fichier externe

  - Traitement proprement dit: Lecture / Ecriture
    - ✓ Get / Put Read / Write …
      - Réalise le traitement effectif désiré sur les enregistrements du fichier.

  - Terminaison, fermeture du fichier:
    - ✓ Close
      - Vide le tampon associé au fichier si nécessaire.
      - Coupe la liaison: variable de fichier du programme - Fichier externe.

II.1 Fichier Texte

L'ordinateur a besoin d'un espace mémoire pour chaque fichier qu'on appelle "tampon" ou en anglais "buffer". Ce tampon peut contenir plusieurs caractères et est rempli chaque fois qu'il est vide. Le mécanisme est que si une instruction de lecture est faite dans un fichier de texte et que le tampon est vide, il sera rempli par les caractères du fichier et ensuite l'affectation se fera à partir du tampon.

Pour associer un tampon avec un fichier, il faut déclarer une variable qui servira de tampon.

```
Fichier_Texte : Ada.Text_IO.File_Type;
```

C'est à l'ouverture du fichier que le lien se fera entre le fichier réel et le tampon en mémoire.

```
Ada.Text_IO.Create(Fichier_Texte, Ada.Text_IO.Out_File, "a :\ Data.txt");
```

Cette dernière instruction, créera un fichier appelé "Data.txt" sur l'unité de disquette "a:", et associera un tampon appelé "Fichier_Texte". Dorénavant, toutes les instructions dans ce fichier se feront via le tampon.

```
Ada.Text_IO.Put(Fichier_Texte,"salut vous autres");
```

À la fin du programme il faut couper le lien entre le fichier et le tampon. Pour ce faire il faut utiliser la procédure close:

```
Ada.Text_IO.Close(Fichier_Texte);
```

II.2 Manipulation des fichiers texte

```
with Ada.Text_IO ; use Ada.Text_IO
-- …
procedure Exemple is
    Original  : File_Type ;    -- Deux variables fichier texte
    Copie     : File_Type ;

    procedure Dupliquer (Source  : in File_Type ;  Destination : in File_Type) is
    begin
        ……..
    end Dupliquer;

begin -- Exemple
        ……………
end Exemple;
```

Remarque :

- La variable de fichier, objet (variable) interne au programme ADA
- Le fichier externe

II.3 Création, Ouverture Traitement et Fermeture de fichiers texte

- Création --nouveau fichier

```
procedure Create ( File  : in out File_Type;   --type de fichier, texte ou binaire
                  Mode : in File_Mode := out_File; --in_File ou Append_File
                  Name : in String := "";        --nom du fichier externe
                  Form  : in string := "");      --parametre fixant les proprietes…
                                                 --exemple : les droits d'acces
```

- Ouverture --fichier existant

```
procedure Open   ( File  : in out File_Type;
                   Mode : in File_Mode;
                   Name : in String;
                   Form  : in string := "");
```

- Traitement

  Lecture, Ecriture, ..(Put, Get , Put_Line, Get_Line, New_Line, Skip_Line)

- Ferméture --*suppression de l'association entre la variable fichier et le fichier externe*

```
procedure Close (File : in out File_Type);
```

II.4 Exemple 1 d'utilisation d'un Fichier texte

```
with Ada.Text_IO;              use Ada.Text_IO ;
with Ada.Integer_Text_IO;      use Ada.Integer_Text_IO ;
with Ada.Float_Text_IO;        use Ada.Float_Integer_Text_IO ;

procedure Exemple is
      Buffer  : File_Type ;                  -- Une variable fichier texte
      Titre   : string(1..10);               -- Une variable chaine de caracteres
      Lettre : Character ;                   -- Une variable caractere
      Nb_Entier : Integer;                   -- Une variable entiere
      Nb_Reel   : Float ;                    -- Une variable reelle
      L             : Natural;               -- Pour un appel correct a Get_Line
begin
      -- Creation et ouverture du fichier a ecrire appeler texte.txt
      Create(File => Fichier_Traite, Name => "texte.txt");
      Put_Line(Fichier_Traite, "Titre");     -- Ecriture sur une line du mot Titre
      Put(Fichier_Traite, 'O');              -- Ecriture d'un caractere
      Put(Fichier_Traite, 123);              -- Ecriture d'un entier
      Put(Fichier_Traite, 'N');              -- Ecriture d'un caractere
      New_Line (Fichier_Traite);             -- Passage a la ligne
      Put(Fichier_Traite, 3.14);             -- Ecriture d'un reel
      New_Line (Fichier_Traite);             -- Passage a la ligne
      Close(Fichier_Traite);                 -- Fermeture du fichier
  ……..
```

II.4 Exemple 1 d'utilisation d'un Fichier texte (suite)

```
      -- suite de la procedure precedante Exemple

      -- Ouverture du meme fichier, cette fois-ci en lecture
      Open(File => Fichier_Traite, Mode => In_File, Name => "texte.txt");
      Get_Line( Fichier_Traite, Titre, L);     -- Lecture du titre (L vaut 5)
      Get(Fichier_Traite, Lettre);             -- Lecture d'un caractere (O)
      Get(Fichier_Traite, Nb_Entier);          -- Lecture d'un entier(123)
      Get(Fichier_Traite, Lettre);             -- Lecture d'un caractere (N)
      Skip_Line (Fichier_Traite);              -- Passage a la ligne
      Get(Fichier_Traite, Nb_Reel);            -- Lecture d'un reel
      Skip_Line (Fichier_Traite);              -- Passage a la ligne
      Close(Fichier_Traite);                   -- Fermeture du fichier
end Exemple ;
 ……..
```

Le fichier texte.txt créé par la procédure Exemple

```
Titre
O     123N
   3.14000E+00
```

II.4 Exemple 2 d'utilisation d'un Fichier texte

```
function End_Of_Line (File : in File_Type) return Boolean;
function End_Of_File (File : in File_Type)  return Boolean;
                                  -------------------------------
with Ada.Text_IO ; use Ada.Text_IO ;

procedure Exemple is
      Fichier_Traite  : File_Type ;          -- Une variable fichier texte
      Lettre : Character ;                   -- Une variable caractere

begin
      -- Ouverture du fichier a lire
      Open(File => Fichier_Traite, Mode => In_File, Name => "texte.txt");
      while not End_Of_File( Fichier_Traite) loop                  -- Fin de fichier ?
            while not End_Of_Line( Fichier_Traite) loop            -- Fin de ligne ?
                  Get(Fichier_Traite, Lettre);                     -- Lecture d'un caractere
                  …….
            end loop;
            Skip_Line (Fichier_Traite);                            -- Passage a la ligne
      end loop;
      Close(Fichier_Traite);                                       -- Fermeture du fichier
 end Exemple ;
```

II.5 Complément sur les fichiers texte

```
procedure Get_Immediate(Item      : out Character);
procedure Get_Immediate(Item      : out Character;
                        Available : out Boolean);
```

- Item le caractère lu immédiatement (clavier ou fichier);
- Available qui indique s'il exite un caractère à lire immédiatement
- S'il n'y a pas de cararctère à lire, la premieère forme fait attendre, la deuxième retourne False

```
procedure Look_Ahead ( Item        : out Character;
                       End_Of_Line : out Boolean);
```

- permet d'obtenir une copie du caractère à lire;
- Item le caractère obtenu si End_Of_Line est faux;
- End_Of_Line est vrai si la fin de ligne est atteinte. Dans ce cas Item n'a pas de valeur définie.

III. Fichiers binaires

Le contenu d'un fichier binaire peut être de tout type(élémentaire, tableau, article, tableau d'articles..). Il n'y a que 2 instructions qui permettent de manipuler le contenu d'un fichier binaire en Ada et c'est READ(lecture) et WRITE(écriture).

Pour pouvoir utiliser un fichier binaire, il faut déterminer son mode d'accès :

- Accès séquentielle => nous devons importer ADA.SEQUENTIAL_IO.
- Accès direct => nous devons importer ADA.DIRECT_IO.

Par la suite, nous devons spécifier de quel type est le fichier et cela se fait par l'instanciation de paquetage (*packetage générique*).

ex; PACKAGE ES_Fic_Etud IS NEW ADA.SEQUENTIAL_IO(T_Etud);

Cette dernière instruction nous donne tous les outils pour pouvoir utiliser un fichier binaire de type "T_Etud" en accès séquentiel. Naturellement, le type "T_Etud" doit avoir préalablement été défini. À partir du moment où le paquetage est défini, n'importe quel référence à un type, une procédure ou une fonction qui provient de ADA.SEQUENTIAL_IO doit être utiliser avec le nom du paquetage en préfixe.

```
ex: Un_Fichier_Binaire : ES_Fic_Etud.File_Type;
    ES_Fic_Etud.Open(Un_Fichier_Binaire, ES_Fic_Etud.IN_FILE, "Unfichier.bin");
```

III. Fichiers binaires

```
with Ada.Sequential_IO;                -- Pas de use sur un paquetage qui
with Ada.Direct_IO;                    -- est generique (cf. ...)
procedure Exemple is
   type T_Mois_De_L_Annee is (Janvier, Fevrier, Mars, Avril, Mai, Juin,  Juillet, Aout,
                              Septembre, Octobre, Novembre, Decembre );
   type T_Date is record
                    Jour    : Integer;
                    Mois    : T_Mois_De_L_Annee;
                    Annee : Integer;
                  end record;
   -- Pour utiliser des fichiers binaires sequentiels d'entiers
   package Entiers_Seq_IO is new Ada.Sequential_IO ( Integer );     use Entiers_Seq_IO;
   -- Pour utiliser des fichiers binaires directs de dates
   package Dates_Dir_IO is new Ada.Direct_IO ( T_Date );            use Dates_Dir_IO;
   Fichier_Entiers : Entiers_Seq_IO.File_Type;     -- Deux variables fichiers
   Fichier_Dates   : Dates_Dir_IO.File_Type;       -- Prefixe necessaire
   -- Lit le fichier d'entiers Mesures
   procedure Lire (Mesures : in Entiers_Seq_IO.File_Type) is ... end Lire;
   -- Ecrit le fichier de dates Dates
   procedure Ecrire (Dates : in Dates_Dir_IO.File_Type) is ... end Ecrire;
begin -- Exemple
```

III.2 Création, Ouverture Traitement et Fermeture de fichiers binaires

- **Création**

```
procedure Create (File    : in out File_Type;
                  Mode : in File_Mode := …; -- (Seq : Out_File, In_File, Append_File…
                  Name : in String := "";       -- Bin : InOut_File ,In_File, Out_File)
                  Form : in String := "" );     -- (Valeur par defaut du parametre Name…
```

- **Ouverture** -- *permet de créer un fichier temporaire)*

```
procedure Open (  File    : in out File_Type;
                  Mode : in File_Mode;
                  Name : in String;
                  Form : in String := "" );
```

- **Traitement Accès aux éléments : fichier binaire séquentiel**

```
procedure Read ( File : in File_Type; Item : out Element_Type );
procedure Write ( File : in File_Type; Item : in Element_Type );
```

- **Fermeture**

```
procedure Close ( File : in out File_Type );
```

III.3 Fichiers binaires séquentiels

```
with Ada.Sequential_IO;
...
procedure Exemple is
        -- Pour utiliser des fichiers binaires sequentiels formes de dates,
        -- les types sont declares comme dans l'exemple precedant
        package Dates_Seq_IO is new Ada.Sequential_IO ( T_Date );   use Dates_Seq_IO;
        Fichier_Dates : File_Type;              -- Une variable fichier binaire
        Date : T_Date;                          -- Une date reelle
begin -- Exemple
        -- Creation et ouverture du fichier a ecrire
        Create ( File => Fichier_Dates, Name => "dates.dat" );
        Date := (1, Avril, 1958);
        Write ( Fichier_Dates, Date );                  -- Ecriture de quelques dates
        Write ( Fichier_Dates, ( 26, Decembre, 1964 ) );
        Write ( Fichier_Dates, ( 7, Septembre, 1991 ) );
        Write ( Fichier_Dates, ( 28, Juillet, 1998 ) );
        Close ( Fichier_Dates );                        -- Fermeture du fichier
        -- Ouverture du même fichier pour le relire
        Open ( File => Fichier_Dates, Mode => In_File, Name => "dates.dat" );
        Read ( Fichier_Dates, Date );                   -- Lecture des quatre dates dans
        Read ( Fichier_Dates, Date );                   -- l'ordre selon lequel elles
        Read ( Fichier_Dates, Date );                   -- viennent d'etre ecrites
        Read ( Fichier_Dates, Date );
        Close ( Fichier_Dates );                        -- Fermeture du fichier
```

III.3 Fichiers binaires séquentiels

```
with Ada.Sequential_IO;
----.
procedure Exemple is
        -- Pour utiliser des fichiers binaires sequentiels formes de dates,
        -- les types sont declares comme dans l'exemple précédant
        package Dates_Seq_IO is new Ada.Sequential_IO ( T_Date );
        use Dates_Seq_IO;
        Original : File_Type;          -- Deux variables fichiers binaires sequentiels
        Copie : File_Type;             -- On ne peut pas faire Original := Copie
        Date : T_Date;                 -- Une variable pour la copie d'un element
begin -- Exemple
        -- Ouverture du fichier a lire et creation de la copie
        Open ( File => Original, Mode => In_File, Name => "dates.dat" );
        Create ( File => Copie, Name => "copie de dates.dat" );
        while not End_Of_File ( Original ) loop      -- Fin de fichier?
                Read ( Original, Date );             -- Lecture d'une date
                Write ( Copie, Date );               -- Ecriture d'une date
        end loop;
        Close ( Original );                          -- Fermeture des fichiers
        Close ( Copie );
end exemple;
```

III.4 Fichiers binaires à accès direct

-- *Quelques déclarations appartenant au paquetage Ada.Direct_IO*

type Count **is range** 0 .. *implementation_defined*;
subtype Positive_Count **is** Count **range** 1 .. Count'Last;

procedure Read (File : **in** File_Type; Item : **out** Element_Type);

procedure Read (File : **in** File_Type; Item : **out** Element_Type; From : **in** Positive_Count);

procedure Write (File : **in** File_Type; Item : **in** Element_Type);

procedure Write (File : **in** File_Type; Item : **in** Element_Type; To : **in** Positive_Count);

Remarque : A chaque lecture ou écriture, l 'indexe est augmenté de 1

procedure Set_Index (File : **in** File_Type; To: **in** Positive_Count);

function Index (File : **in** File_Type) **return** Positive_Count;

III.4 Traitement d ’un fichier binaire à accès direct

```
with Ada.Direct_IO;
-----
procedure Exemple is
        -- Pour utiliser des fichiers binaires a acces direct formes de dates
        -- les types sont declares comme dans l’exemple précédant
        package Dates_Dir_IO is new Ada.Direct_IO ( T_Date );
        use Dates_Dir_IO;
        Original : File_Type;              -- Deux variables fichiers a acces direct
        Copie : File_Type;                 -- Une variable pour la copie d’un element
begin -- Exemple
         -- Ouverture du fichier a lire et creation de la copie
         Open ( File => Original, Mode => In_File, Name => "dates.dat" );
         Create ( File => Copie, Name => "copie de dates.dat" );
         -- Copier tous les elements
        for No_Element_Courant in reverse 1 .. Size ( Original ) loop
                -- Placer l’index sur l’element No_Element_Courant du fichier lu
                 Set_Index ( Original, No_Element_Courant );
                 Read ( Original, Date );   -- Lecture d’une date
                 Write ( Copie, Date );     -- Ecriture d’une date
        end loop;
       Close ( Original );                 -- Fermeture des fichiers
       Close ( Copie );
end Exemple;
```

IV. Compléments sur l'utilisation des fichiers (texte et binaires)

procedure Delete (File : **in out** File_Type);
« *Ferme le fichier et l'efface de la mémoire secondaire* »

procedure Reset (File : **in out** File_Type);
« *Rembobine dans le cas du mode lecture ou écriture* »

procedure Reset (File : **in out** File_Type; Mode : **in** File_Mode);
« *Repositionne au dernier élément dans le cas du mode adjonction* »

function Is_Open (File : **in** File_Type) **return** Boolean;
« *Vérifier si le fichier est ouvert ou non* »

function Mode (File : **in** File_Type) **return** File_Mode;
« *Informe sur le mode d 'ouverture du fichier* »

function Name (File : **in** File_Type) **return** String;
« *Informe sur le nom du fichier externe connecté* »

V. Exceptions lors de l'utilisation des fichiers (texte et binaires)

Toutes les opérations de traitement des fichiers ou de leurs éléments peuvent générer des exceptions. Ces exceptions sont toutes déclarées dans le paquetage Ada.IO_Exception

- **Status_Error** le fichier est ouvert alors qu'il devrait être fermé ou vice-versa;
- **Mode_Error** le fichier n'a pas le bon mode, par exemple `In_File` au lieu de `Out_File`;
- **Name_Error** une erreur est apparue dans le nom externe de `Create` ou `Open`;
- **Use_Error** diverses raisons (!); paramètre `Form` inacceptable, ordre d'impression sur un périphérique d'entrée etc;
- **Device_Error** le dispositif physique est en panne ou n'est pas connecté;
- **End_Error** tentative de lecture au-delà de la fin de fichier;
- **Data_Error** `Read` ou `Get` ne peut pas interpréter les données comme valeurs du type désiré;
- **Layout_Error** un problème de formatage dans `Ada.Text_IO`, ou bien `Put` écrit trop de caractères dans une chaîne.