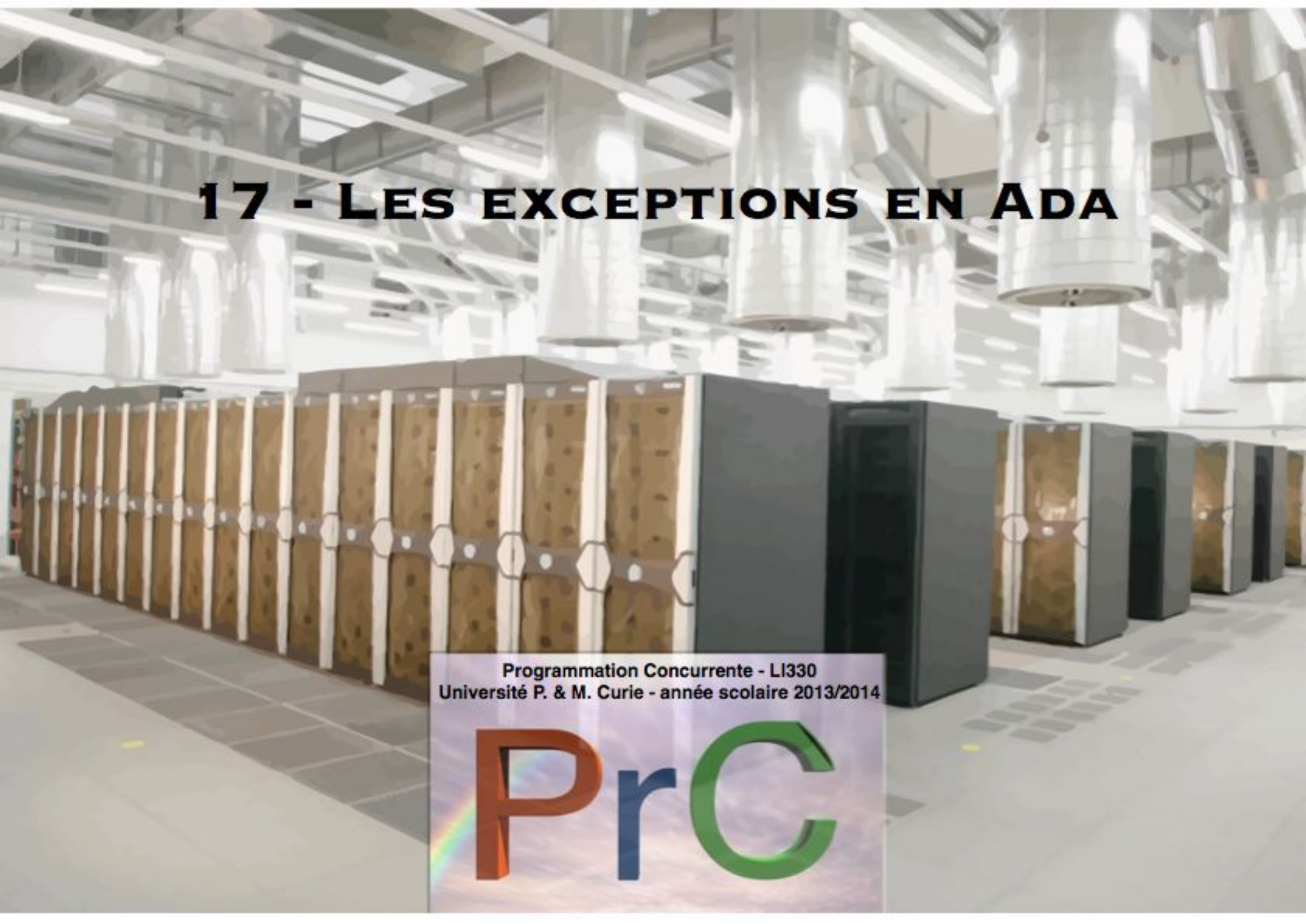

17 - Les exceptions en Ada

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

Exceptions en Ada

Exceptions prédéfinies

- **Constraint_Error:** toute tentative d'associer une valeur illicite par rapport au type d'une expression ou d'une variable
- **Storage_Error:** dépassement de la capacité mémoire
- **Program_Error:** fin d'une fonction atteinte sans instruction return + ...
- **Tasking_Error:** pour la concurrence

Exceptions définies par l'utilisateur

- **Il faut donner une sémantique précise à toutes les exceptions définies**

Déclaration

décl_excep identificateur {, identificateur} : **exception;**

```
Mon_Error : exception;
Pile_Vide_Error, Pile_Pleine_Error : exception;
```

Traitement d'une exception

- À la fin de tout «bloc»
  - Fonction
  - Procédure
  - Initialisation d'un paquetage
  - Instruction-bloc
- Syntaxe

```
except_sec exception
              trait_excep
              {trait_excep}
trait_excep when lst_excep =>
               liste_d_instructions
lst_excep   identificateur                                  |
            identificateur | {identificateur }              |
            [identificateur :] others
```

Levée d'une exception

- Levée implicite
  - Dans une instruction du langage
  - Pour les exceptions prédéfinies du langage

- Levée explicite
  - Dans une suite d'instructions
  - Pour les exception définies

```
raise identificateur;
```

- Propagation implicite
  - Dans un traitement d'exception

```
raise [identificateur];
```

Exemple 1: Saisie d'un entier


```
-- Saisie d'un entier non sécurisée. Si Constraint_Error est  levée, elle sera propagée à
-- l'appelant.
with Ada.Text_Io;
use Ada.Text_Io;

procedure Get_1 (V : out Integer) is

   S : string (1..30);
   L : Natural;

begin
   Get_Line (S, L);
   V := Integer'Value (S (1 .. L));
end Get_1;
```

Exemple 1: traitement de l'exception au niveau de l'appelant


```
-- Nouveau programme de cumul
with Ada.Text_Io, Get_1;
use Ada.Text_Io;

procedure Cumul_1 is

   Cumul, V : Integer := 0;

begin
   Put_Line ("Saisissez une serie de chiffres a cumuler, tapez
  n'importe quoi pour sortir");
   loop
      begin
         Get_1 (V);
         Cumul := Cumul + V;
      exception
         when Constraint_Error =>
            exit;
      end;
   end loop;
   Put_Line ("La somme totale =" & Integer'Image (Cumul));
end Cumul_1;
```

Exemple 1: exécution


```
$ cumul_1
Saisissez une serie de chiffres a cumuler, tapez n'importe quoi pour
  sortir
10
20
2
-10
.
La somme totale = 22
```

Exemple 2: Get sécurisé

```
-- Saisie "securisee" d'un entier. On redemande jusqu'à ce que l'entier soit valide
with Ada.Text_Io;
use Ada.Text_Io;

procedure Get_2 (V : out Integer) is

   S : string (1..30);
   L : Natural;

begin
   loop
      begin
         Get_Line (S, L);
         V := Integer'Value (S (1 .. L));
         exit;
      exception
         when Constraint_Error =>
            Put_Line ("Vous n'avez pas saisi un entier");
            Skip_Line;
      end;
   end loop;
end Get_2;
```

Exemple 2: test du Get sécurisé

```
with Ada.Text_Io, Get_2;
use Ada.Text_Io;

procedure Cumul_2 is

   Cumul, V : Integer := 0;
   S : String (1..10);
   L : Natural;

begin
   Put_Line ("Saisissez une serie de chiffres a cumuler");
   loop
      Get_2 (V); -- Saisie sécurisée, on a forcément un entier
      Cumul := Cumul + V;
      Put_Line ("Voulez-Vous Continuer ?");
      Get_Line (S, L);
      if L = 1 and (S (1) = 'N' or S (1) = 'n') then
         exit;
      end if;
   end loop;
   Put_Line ("La somme totale =" & Integer'Image (Cumul));
end Cumul_2;
```

Exemple 2: exécution

```
$ cumul_2
Saisissez une serie de chiffres a cumuler
4
Voulez-Vous Continuer ?
O
xsxd
Vous n'avez pas saisi un entier
34
Voulez-Vous Continuer ?
o
-10
Voulez-Vous Continuer ?
n
La somme totale = 28
```

Leçons

- Faire extrêmement attention dans le rattrapage des exceptions
  - Il est clair que le programmeur de Get n'en connaît pas les usages futurs
- Le choix de rattraper les exception et de les traiter dépend du cahier des charges

Exemple 3 : Get_Mixte


```
-- Procédure permettant de saisir un entier ou un flottant selon les deux syntaxes
--(avec ou sans ".").
-- raise Constraint_Error si on n'a ni un entier ni un flottant
with Ada.Text_Io;
use Ada.Text_Io;
procedure Get_Mixte (V : out Float) is
   S : string (1..30);
   L : Natural;
begin
   Get_Line (S, L);
   begin
      V := Float'Value (S (1 .. L));
   exception
      when Constraint_Error =>
         begin
            V := Float (Integer'Value (S (1 .. L)));
         exception
            when Constraint_Error => raise;
         end;
   end;
end Get_Mixte;
```

Exemple 3 : Test_Get_Mixte


```
-- Test de Get_Mixte
with Ada.Text_Io, Get_Mixte;
use Ada.Text_Io;

procedure Test_Get_Mixte is

   V : Float;

begin
   Get_Mixte (V);
   Put_Line ("Voici V: " & Float'Image (V));
end Test_Get_Mixte;
```

Exemple 3 : Exécution

```
$ test_get_mixte
12
Voici V:  1.20000E+01
$ test_get_mixte
12.0
Voici V:  1.20000E+01
$ test_get_mixte
12.2
Voici V:  1.22000E+01
$ test_get_mixte
sdfddsf

raised CONSTRAINT_ERROR : s-valuns.adb:85
```

Constraint_Error levée pour la première conversion de la chaîne de caractères

Constraint_Error n'est pas levée dans cette exécution

Constraint_Error n'est pas levée dans cette exécution

Exemple 4 : Tangente


```
-- Fonction pour calculer l'arc-tangente avec cosinus et sinus
with Ada.Numerics.Elementary_Functions;
use Ada.Numerics.Elementary_Functions;

function Tangente (V : Float) return float is


begin
   return Sin (V) / Cos (V);
exception
   when Constraint_Error =>
      if Sin (V) >= 0.0 then
         return Float'Last;
      else
         return Float'First;
      end if;
end Tangente;
```

Exemple 4 : Test_Tangente


```
-- Test de Tangente

with Ada.Text_Io, Arc_Tangente, Get_Mixte;
use Ada.Text_Io;

procedure Test_Tangente is

   V : Float;

begin
   Get_Mixte (V);
   Put_Line ("Voici Tangente (" & Float'Image (V) &
             "): " & Float'Image (Tangente (V)));
end Test_Tangente;
```

Exemple 4 : Exécution


```
$ test_tangente
1.0E-99
Voici Tangente ( 0.00000E+00):  +Inf*******
$ test_tangente
1.0E-10
Voici Tangente ( 1.00000E-10):  1.00000E+10
$ test_tangente
-1.0E-10
Voici Tangente (-1.00000E-10): -1.00000E+10
[$ test_tangente
-1.0E-100
Voici Tangente (-0.00000E+00): -Inf*******
$ test_tangente
1
Voici Tangente ( 1.00000E+00):  6.42093E-01
```

Constraint_Error est levée

Exemple 5 : "+" sur des matrices (1)

```
-- Paquetage minimal pour illustration des exceptions
package Matrice_Min is

   -- Exception en cas d'erreur
   Matrices_Incompatibles_Error : exception;

   -- le type Matrice
   type Matrice is array (Positive range <>, Positive range <>)
       of Integer;

   -- Addition entre deux matrices
   -- raise Matrice_Incompatibles_Error
   function "+" (M1, M2 : Matrice) return Matrice;

end Matrice_Min;
```

Exemple 5 : "+" sur des matrices (2)


```
package body Matrice_Min is

   function "+" (M1, M2 : Matrice) return Matrice is

      R : Matrice (M1'Range (1), M1'Range (2));

   begin
      for I in M1'Range (1) loop
         for J in M1'Range (2) loop
            R (I, J) := M1 (I, J) + M2 (I, J);
         end loop;
      end loop;
      return R;
   exception
      when Constraint_Error =>
         raise Matrices_Incompatibles_Error;
   end "+";

end Matrice_Min;
```

Exemple 5 : test du "+" sur des matrices


```
with Ada.Text_Io, Matrice_Min; use Ada.Text_Io, Matrice_Min;
procedure Test_Matrice_Min is
   M1 : Matrice (1..2, 1..2) := ((1, 2), (3, 4)); -- Matrice 2x2
   M2 : Matrice (3..4, 1..2) := ((4, 5), (6, 7)); -- Matrice 2x2
   M3 : Matrice (1..2, 1..3) := ((1, 2, 3), (4, 5, 6)); -- Matrice 2x3
begin
   begin
      M1 := M1 + M1; Put_Line ("On est passe sur M1 + M1");
   exception
      when Matrices_Incompatibles => Put_Line ("Pb sur M1 + M1");
   end;
   begin
      M1 := M1 + M2; Put_Line ("On est passe sur M1 + M2");
   exception
      when Matrices_Incompatibles => Put_Line ("Pb sur M1 + M2");
   end;
   begin
      M1 := M1 + M3; Put_Line ("On est passe sur M1 + M3");
   exception
      when Matrices_Incompatibles_Error => Put_Line ("Pb sur M1 + M3");
   end;
end Test_Matrice_Min;
```

Exemple 5 : Exécution

```
$ test_matrice_min
On est passe sur M1 + M1
Pb sur M1 + M2
On est passe sur M1 + M3
```

Normal, mêmes dimensions et de mêmes intervalles d'indices

Normal, mêmes dimensions mais intervalles d'indices différents

Meilleure programmation de "+"


```
function "+" (M1, M2 : Matrice) return Matrice is
   R : Matrice (M1'Range (1), M1'Range (2));
begin
   if M1'First(1) /= M2'First(1) or else
      M1'Last(1) /= M2'Last(1) or else
      M1'First(2) /= M2'First(2) or else
      M1'Last(2) /= M2'Last(2) then
      raise Matrices_Incompatibles_Error;
   end if;
   for I in M1'Range (1) loop
      for J in M1'Range (2) loop
         R (I, J) := M1 (I, J) + M2 (I, J);
      end loop;
   end loop;
   return R;
end "+";
```

Comparaison explicite des dimensions

```
$ test_matrice_min
On est passe sur M1 + M1
Pb sur M1 + M2
Pb sur M1 + M3
```

La bonne programmation de "+"


```
function "+" (M1, M2 : Matrice) return Matrice is
   R : Matrice (1 .. M1'Length (1), 1 .. M1'Length (2));
begin
   if M1'Length(1) /= M2'Length(1) or else
      M1'Length(2) /= M2'Length(2) then
      raise Matrices_Incompatibles_Error;
   end if;
   for I in 1 .. M1'Length (1) loop
      for J in 1 .. M1'Length (2) loop
         R (I, J) := M1 (M1'First (1) + I - 1, M1'First (2) + J - 1) +
                     M2 (M2'First (1) + I - 1, M2'First (2) + J - 1);
      end loop;
   end loop;
   return R;
end "+";
```

```
$ test_matrice_min
On est passe sur M1 + M1
On est passe sur M1 + M2
Pb sur M1 + M3
```

Autres exceptions utiles (1)

- Data_Error
  - Définie dans Ada.Text_Io & consort…
  - S'applique à tout problème de saisie
- Name_Error
  - Définie dans Ada.Text_Io & consort…
  - S'applique à tout tentative d'ouverture d'un fichier inexistant
- End_Error
  - Définie dans Ada.Text_Io & consort…
  - S'applique dès que l'on dépasse la fin d'un fichier
- Status_Error
  - Définie dans Ada.Text_Io & consort…
  - Tentative d'ouverture, lecture ou d'écriture dans un fichier non ouvert
  - Tentative de fermeture d'un fichier fermé

Autres exceptions utiles (2)

- Mode_Error
  - Définie dans Ada.Text_Io & consort…
  - Violation du mode d'utilisation d'un fichier
    - Tentative de lecture sur un fichier ouvert en écriture
    - Tentative d'écriture sur un fichier ouvert en lecture
- Device_Error
  - Définie dans Ada.Text_Io & consort…
  - Problème lié au support (le disque en l'occurrence), en général lorsque le disque est plein
- etc.

Le paquetage Ada.Exceptions (1)


```
-- Paquetage issu de la norme du langage
package Ada.Exceptions is
   type Exception_Id is private;
   Null_Id : constant Exception_Id; -- Lié à l'attribut Except'Identity
   function Exception_Name(Id : Exception_Id) return String;

   type Exception_Occurrence is limited private;
   type Exception_Occurrence_Access is
        access all Exception_Occurrence;
   Null_Occurrence : constant Exception_Occurrence;

   procedure Raise_Exception (E : in Exception_Id;
                              Message : in String := "");

   function Exception_Message(X : Exception_Occurrence)
       return String;

   procedure Reraise_Occurrence(X : in Exception_Occurrence);
```

Le paquetage Ada.Exceptions (2)


```
   function Exception_Identity(X : Exception_Occurrence)
      return Exception_Id;

   function Exception_Name(X : Exception_Occurrence) return String;

   -- Same as Exception_Name(Exception_Identity(X)).
   function Exception_Information(X : Exception_Occurrence)
      return String;

   procedure Save_Occurrence (Target : out Exception_Occurrence;
                              Source : in Exception_Occurrence);

   function Save_Occurrence(Source : Exception_Occurrence)
     return Exception_Occurrence_Access;

private
... -- not specified by the language
end Ada.Exceptions;
```

Exemple d'utilisation de Ada.Exceptions: nouvelle version de "+"


```
with Ada.Exceptions;
use Ada;

...

function "+" (M1, M2 : Matrice) return Matrice is
      R : Matrice (1 .. M1'Length (1), 1 .. M1'Length (2));
   begin
      if M1'Length(1) /= M2'Length(1) or else
         M1'Length(2) /= M2'Length(2) then
         Exceptions.Raise_Exception
            (Matrices_Incompatibles_Error'identity,
             "Matrices de tailles differentes!");
      end if;
      for I in 1 .. M1'Length (1) loop
         for J in 1 .. M1'Length (2) loop
            R (I, J) := M1 (M1'First (1)+I-1, M1'First (2)+J-1) +
                        M2 (M2'First (1)+I-1, M2'First (2)+J-1);
         end loop;
      end loop;
      return R;
   end "+";
```

Test de la nouvelle fonction "+"

```
with Ada.Text_Io, Matrice_Min, Ada.Exceptions;
use Ada.Text_Io, Matrice_Min, Ada;

procedure Test_Matrice_Min is

   M1 : Matrice (1..2, 1..2) := ((1, 2), (3, 4)); -- Matrice 2x2
   M3 : Matrice (1..2, 1..3) := ((1, 2, 3), (4, 5, 6)); -- Matrice 2x3

begin
   begin
      M1 := M1 + M3;
      Put_Line ("On est passe sur M1 + M3");
   exception
      when E : others =>
         Put_Line ("Pb sur M1 + M3");
         Put_Line ("l'exception levee est " &
                   Exceptions.Exception_Name (E));
         Put_Line ("Message associe : "  &
                   Exceptions.Exception_Message (E));
   end;
end Test_Matrice_Min;
```

Exécution de la fonction "+"

```
$ test_matrice_min
Pb sur M1 + M3
l'exception levee est MATRICE_MIN.MATRICES_INCOMPATIBLES_ERROR
Message associe : Matrices de tailles differentes!
```