
Algorithmique & programmation

Les Exceptions en Ada

Introduction

- Que se passe-t-il si, dans un programme,
 - Vous effectuez une division par zéro ?
 - Vous demandez à l'utilisateur de votre programme de saisir un entier positif et que celui ci tape une valeur négative ?
 - ...
- Le programme s'arrête car...
- ... pour garantir la cohérence de vos programmes, le système effectue des contrôles lors de l'exécution
 - il est capable par exemple, de détecter l'absence d'un fichier
 - de constater qu'une valeur n'a pas le type attendu
 - ...
- Lorsqu'il s'aperçoit d'une telle anomalie il crée (ou lève) ce que l'on appelle **une exception**

Introduction

- ❑ Le concept **d'exception** a été introduit pour la première fois en Ada et généralisé à tous les langages « modernes » : C++, Java, ...
- ❑ Le programmeur, **lors de la découverte d'une anomalie par le système** peut - **et devrait toujours** - traiter l'exception correspondante de façon à prendre en compte le problème associé.
- ❑ Le programmeur peut également déclencher volontairement ce mécanisme si besoin
- ❑ Les exceptions sont ainsi un moyen élégant de gérer les "conditions limites" de vos programmes

Introduction



Ce mécanisme est fondamental pour produire des logiciels robustes

- ❑ ...Une fusée Ariane (plusieurs millions d'€) a été perdue parce qu'une exception dans un programme Ada était mal traitée...

Définitions

- Une **exception** est un événement qui est déclenché pendant l'exécution d'un programme lorsqu'une situation anormale (exceptionnelle) est détectée
- Lorsque l'**exception est déclenchée** (on dit aussi levée), l'**exécution normale du programme est suspendue** pour que l'exception soit traitée.

- Traitement d'une exception
 - Le **traitement par défaut** des exceptions par le système consiste à **afficher un message d'erreur** et à **stopper l'exécution du programme**
 - Le **bon programmeur**, soucieux de la robustesse de son programme, **remplace ce traitement par défaut par un traitement adéquat**

Exceptions système courantes

- **Constraint_Error** : déclenchée si
 - une valeur trop grande ou trop petite est affectée à une variable d'un sous-type contraint
 - une valeur d'indice de tableau déborde de l'intervalle (forcément contraint) fixé pour les indices

- **Program_Error** : déclenchée si
 - on essaye d'exécuter une action erronée (si par exemple le return d'une fonction n'est pas exécuté)

Exceptions système courantes

□ **Storage_Error** :

- déclenchée lorsque le système manque de mémoire (pour les allocations dynamiques que l'on verra plus tard)

□ **Data_Error, Status_Error, Mode_Error, Name_Error, Device_Error, End_Error** :

- Exceptions spécifiques aux opérations d'E/S.
- Définies dans les spécifications des modules `Ada.Text_io`, `Ada.Sequential_io` et `Ada.Direct_io`
- A titre d'exemple regarder comment le package `P_Esiut` est réalisé à partir du module `Ada.Text_io` et comment il utilise les exceptions pour contrôler les opérations d'E/S

Types d'exception

□ On peut distinguer deux sortes d'exceptions :

- Celles déclenchées par le système sans que le programmeur le demande
- Celles déclarées et déclenchées explicitement par le programmeur

Traitement d'une exception

- Ada permet au programmeur de définir le traitement à accomplir lors de la détection d'une exception dans un programme
- Il est possible de définir, ...
 - ... à différents endroits du programme, ...
 - ... des traitements différents pour des exceptions semblables
- Le traitement que l'on peut définir pour une exception peut être arbitrairement complexe :
 - du plus simple (affichage d'un message)
 - au plus élaboré (contrôle de processus industriels...)

Traitement d'une exception

- Le traitement doit se trouver dans une partie particulière appelée **traite-exception** ayant la forme suivante :

exception

```
when Nom_D_Exception1 => Action_À Faire1
```

```
when Nom_D_Exception2 => Action_À Faire2
```

- Comme dans un **case** on peut utiliser la barre | et le **when others => ...**
- **Le traite-exception se trouve toujours à la fin d'un bloc d'instructions.**
 - Dans un sous-programme, le traite-exception se trouve donc juste avant le **end final...**

Exemple de traitement d'exception

```
subtype T_Note is Float range 0.0..20.0 ;
procedure Lecturenote1 is
  N : T_Note ;
begin
  Ecrire (" Donner la note : ") ;
  Lire (N) ;
  -- Aucune des 2 instructions suivantes
  -- n'est exécutée en cas de mauvaise saisie
  Ecrire ("La note saisie est : ") ;
  Ecrire (N) ;
exception
  when Constraint_Error =>
    Ecrire_Ligne("La note doit être entre 0 et 20.") ;
    Ecrire_Ligne("Procédure terminée : N non
  affectée.") ;
end Lecturenote1 ;
```

Trace de la procédure lecturenote1

La procédure est appelée ...

■ Exécution normale

- Donner la note : 12.0
- La note saisie est : 12.0

■ Exécution anormale

- Donner la note : 21.0
- La note doit être entre 0 et 20.
- Procédure terminée : N non affectée.

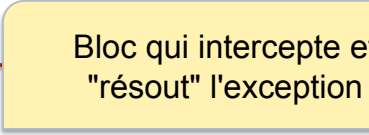
Traitement d'une exception

- ❑ Si l'on veut définir un traitement d'exception pour un ensemble d'instruction plus petit qu'un sous-programme complet, il faut définir explicitement un **bloc d'instructions**
- ❑ Il faut encadrer la suite d'instructions où l'on veut traiter l'exception par les mots **begin** et **end**
- ❑ le traite-exception du bloc sera alors défini juste avant le **end** de fin de bloc.

Exemple de traitement d'exception

```
subtype T_Note is Float range 0.0..20.0 ;

procedure Lecturenote2 is
  N : T_Note ;
begin
  Ecrire ("Donner la note : ") ;
  begin -- début du bloc d'instructions
    Lire (N) ;
    exception
      when Constraint_Error =>
        Ecrire_Ligne("La note doit être entre 0 et 20.") ;
        N := 10 ;
        Ecrire_Ligne("L'exécution continue avec N = 10.") ;
  end ; -- fin du bloc d'instructions
  -- Les instructions après le bloc d'instructions
  -- seront exécutées dans tous les cas
  Ecrire ("La note saisie est : ") ;
  Ecrire (N) ;
end Lecturenote2 ;
```



Bloc qui intercepte et "résout" l'exception

Trace de la procédure `lecturenote2`

□ La procédure est appelée ...

■ **Exécution normale**

- Donner la note : **12.0**
- La note saisie est : 12.0

■ **Exécution anormale**

- Donner la note : **21.0**
- La note doit être entre 0 et 20.
- L'exécution continue avec **N = 10.**
- La note saisie est : **10.0**

Exceptions programmées

□ Il est possible au programmeur de définir ses propres exceptions :

- en déclarant des variables de type exception
- en déclenchant ces exceptions explicitement au moyen de l'instruction **raise** :

raise *nom_d'exception* ;

□ Nous conviendrons de préfixer tous les noms d'exceptions par **E_**

Exemple

```
package body P_TraiteNombre is

  procedure Calcul (...) is
    I : Integer ;
  begin
    ...
    if I < 0 then
      raise E_NombreNegatif
    end if ;
    ...
  end Calcul ;

  procedure AppelCalcul (...) is
  begin
    Calcul(...) ;
  exception
    when E_NombreNegatif =>
      Ecrire ("Nombre négatif interdit") ;
  end AppelCalcul ;

end P_TraiteNombre ;
```

```
package P_TraiteNombre is
  ...
  E_NombreNegatif : exception ;
  ...
end P_TraiteNombre ;
```

Le programmeur définit une exception

La procédure **Calcul** peut lever une exception

La procédure **AppelCalcul** intercepte et traite l'exception **E_NombreNegatif**

Exemple

```
package body P_TraiteNombre is

  procedure Calcul (...) is
    I : Integer ;
  begin
    ...
    if I < 0 then
      raise E_NombreNegatif
    end if ;
    ...
  end Calcul ;

  procedure AppelCalcul (...) is
  begin
    Calcul(...) ;
  exception
    when E_NombreNegatif =>
      Ecrire ("Nombre négatif interdit") ;
  end AppelCalcul ;

end P_TraiteNombre ;
```

```
package P_TraiteNombre is
  ...
  E_NombreNegatif : exception ;
  ...
end P_TraiteNombre ;
```

Le programmeur définit une exception

La procédure **Calcul** peut lever une exception

La procédure **AppelCalcul** intercepte et traite l'exception **E_NombreNegatif**

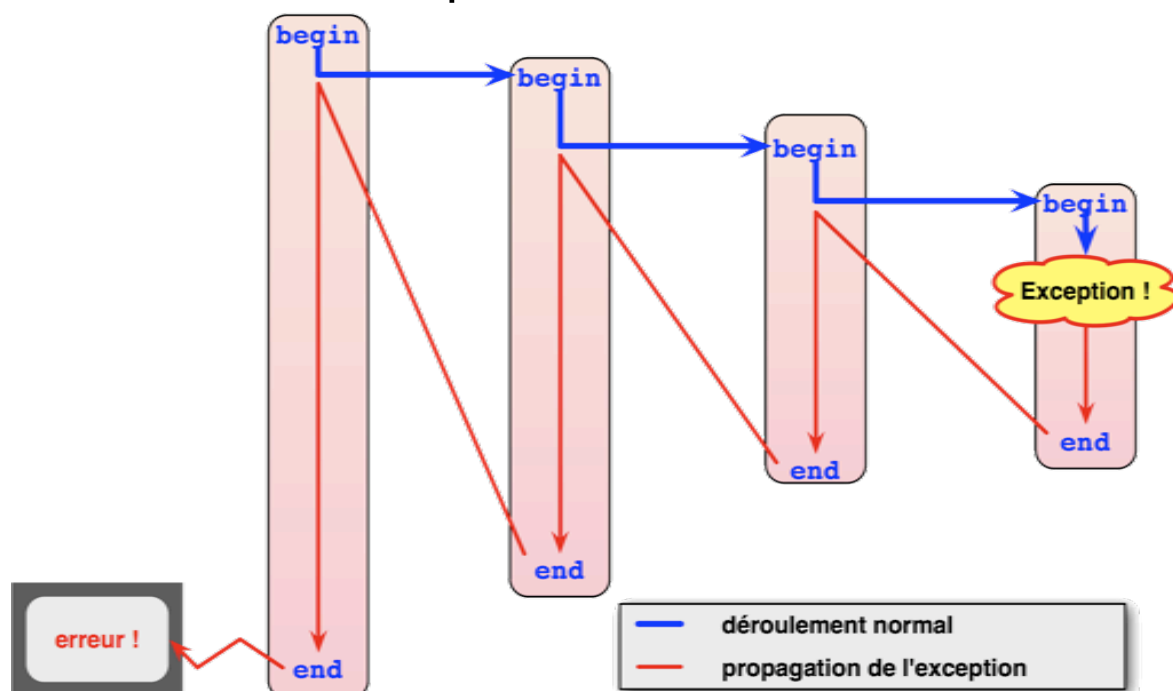
Propagation des exceptions

- Lorsque la partie **traite-exception** est présente dans un bloc d'instruction :
 - si l'exécution du bloc ne produit pas d'exception, cette exécution est faite normalement
 - sinon le traitement d'exception est effectué avant de quitter le bloc d'instruction
- Si le traitement d'une exception n'a pas été prévu dans le bloc où elle survient, l'exception se propage de blocs en blocs jusqu'à arriver dans un bloc où elle a été prévue et sera donc traitée.
- Si le traitement de cette exception n'a été prévu à aucun niveau, le programme s'arrête avec un message d'erreur :

Exception never handled : nom de l'exception

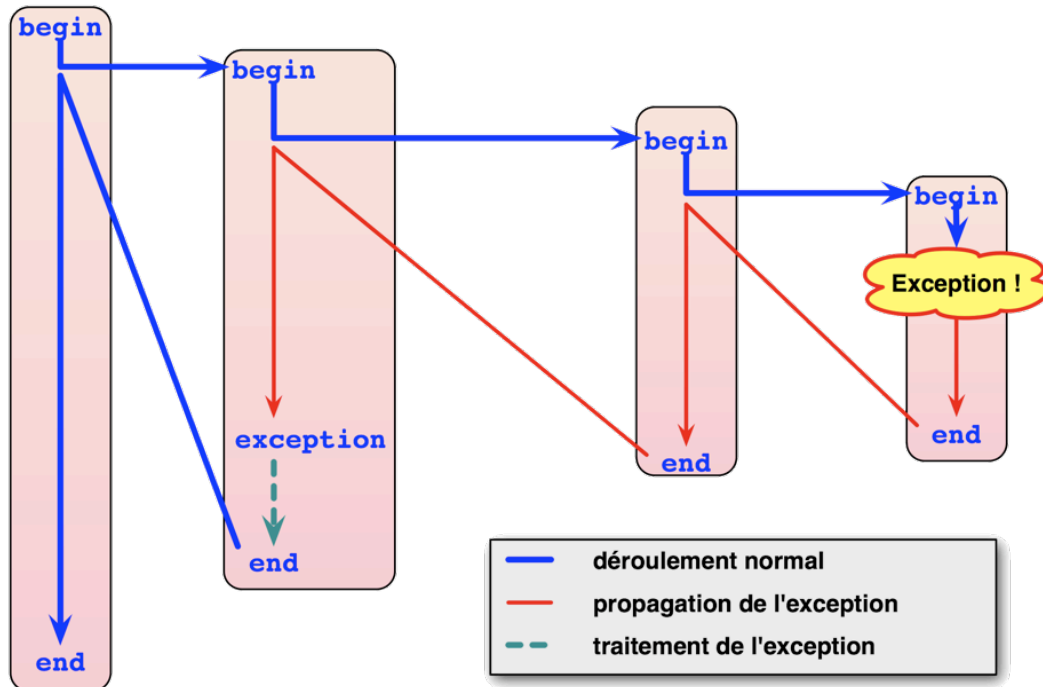
Propagation d'une exception ...

- ... non interceptée



Propagation d'une exception ...

- ... interceptée et traitée



Exemple

```
procedure A is
  E_Erreur : exception ;
  procedure B is
    begin
      ...
      raise E_Erreur ;
    end B ;
  procedure C is
    begin
      ...
      raise E_Erreur ;
      exception
        when E_Erreur =>
          Ecrire_Ligne ("erreur dans C") ;
    end C ;
begin -- de A
  B ;
  C ;
  exception
    when E_Erreur =>
      Ecrire_Ligne ("erreur dans A") ;
end A ;
```

- Il n'y a pas de traitement de **E_Erreur** dans **B**
- L'exception **E_Erreur** déclenchée dans **C** est traitée dans la procédure **C**
- L'exception **E_Erreur** déclenchée dans **B** est traitée dans **A**

Exemple de propagation programmée

```
procedure Bidon(D: in out Integer) is
begin
  D:=D*Integer'Last ;    -- Levée d'une d'exception
exception
  when Constraint_Error =>
    Ecrire_Ligne("valeur non acceptée") ;
    raise ;    -- Exception propagée aux niveaux
               -- supérieurs
end Bidon ;
```

Ce qu'il faut retenir

- Une exception est un événement, portant un nom
 - exemple `Constraint_Error`
- Une exception est déclarée, comme toutes les autres entités d'Ada
- Le déclenchement d'une exception signale l'apparition d'une situation exceptionnelle
 - Ceci peut arriver implicitement (si une règle du langage n'est pas satisfaite) ou explicitement si une instruction `raise` est exécutée.
- Tout traitement qui déclenche une exception est interrompu
 - Le contrôle est alors passé à la séquence de traitement d'exception si elle a été prévue selon les règles de portée correspondant à l'imbrication de blocs.

Exercice 1

- ❑ On considère le sous-type suivant :
`subtype T_Note is float range 0.0..20.0;`
- ❑ Ecrire une procédure pour saisir de façon robuste au clavier une valeur de type `T_Note`
- ❑ Cette procédure "bouclera" - avec un message d'erreur - tant que la valeur saisie par l'utilisateur ne sera pas correcte (c'est-à-dire tant que la lecture au clavier déclenchera une exception).

Exercice 2

- ❑ Ecrire le corps du paquetage suivant :

```
package P_Produit is
    type TR_Produit is record
        Code : Integer ; -- code unique pour chaque produit
        Prix : Float ;
    end record;

    type TV_Produits is array (Positive range <>) of TR_Produit ;
    E_ProduitPasTrouve : exception ;

    function RecherchePrix (V      : in TV_Produits ;
                           Code   : in Integer ) return Float ;
    -- V trié sur les codes
    -- résultat = prix du produit désigné par code
    -- sinon, la fonction lève une exception E_ProduitPasTrouve

    function MoyennePrix (V      : in TV_Produits) return Float ;
    -- résultat = moyenne des prix du vecteur V
    -- Si V est vide, la fonction lèvera une exception NUMERIC_ERROR

end P_Produit ;
```

Exercice 2 - Suite

- Ecrire une procédure principale qui
 - déclare un vecteur de produits
 - l'initialise avec un agrégat
 - Lit un code produit au clavier et affiche son prix (si le produit n'existe pas, on recommence la saisie...)
 - Affiche la moyenne des prix en traitant l'exception **NUMERIC_ERROR**