Le langage Ada – 10 : Les exceptions

Yves Bailly

EN DEUX MOTS **Le mécanisme des exceptions, qui permet de réagir à une situation anormale et inattendue, a souvent été évoqué dans les articles précédents. Il est temps de voir les possibilités offertes par Ada pour intercepter et déclencher des exceptions, assurant ainsi un bon déroulement du programme même en cas d'erreur critique.**

Actualités

Un petit mot avant d'aborder notre sujet. Les lecteurs des articles précédents auront sans doute remarqué le changement survenu dans le titre : cette série d'articles n'est plus intitulée « Ada 95 », mais « Le langage Ada ». Nous sommes en effet maintenant en 2006, année durant laquelle devrait normalement être validé le nouveau standard du langage Ada [1]. Celui-ci ayant été conçu pour l'essentiel durant l'année 2005, on parle généralement de la version « Ada 2005 » (ou Ada05) du langage. Les dernières versions du compilateur de référence utilisé, le compilateur Gnat qui fait partie de la suite GCC, progresse régulièrement dans le support de ce nouveau standard. Aussi ai-je décidé de sauter le pas et de concentrer cette présentation du langage Ada sur cette dernière mouture du langage.

À noter également le remarquable développement du WikiBook [2] consacré au langage Ada, sous forme de didacticiels (ou tutoriels) progressifs sur plus de 200 pages. Il s'agit en réalité du premier ensemble de textes introduisant spécifiquement les nouveautés du standard Ada 2005. Enfin, une récente enquête [3] a montré que le marché autour du langage Ada peut être estimé à quelque chose comme 5.6 milliards de dollars, Europe et Amérique du Nord combinées, pour environ 322 millions de ligne de code. Les industries de la défense et de l'aérospatiale sont largement représentées, ce qui est traditionnel, mais le champ d'application du langage est beaucoup plus vaste, de la recherche génétique aux machines à voter en passant par le monde de la finance. Voilà sans doute de quoi modérer le dédain de ceux qui pensent que « personne n'utilise Ada... ». Mais revenons à nos exceptionnels moutons.

Interception

Commençons par générer un petit programme qui va lever une paire d'exceptions. Par exemple, sortir des limites d'un tableau et tenter de lire un fichier inexistant. Le voici :

```
 1 with Text_IO ;
 2 use  Text_IO ;
 3 procedure Interception_1 is
 4    type TTab is array(1..10) of Integer ;
 5    t: TTab ;
 6    f: File_Type ;
 7 begin
 8    t(0) := 1 ;
 9    Open(File => f,
10         Mode => In_File,
11         Name => "n_importe_quoi") ;
12    Put_Line("Fin du programme.") ;
13 end Interception_1 ;
```

Rien de bien méchant. À noter que la compilation du programme est assortie d'un avertissement :

```
$ gnatmake -gnat05 interception_1.adb
gcc -c interception_1.adb
interception_1.adb:8:06: warning: value not in range of subtype of
"Standard.Integer" defined at line 4
interception_1.adb:8:06: warning: "Constraint_Error" will be
raised at run time
gnatbind -x interception_1.ali
gnatlink interception_1.ali
```

Le paramètre `-gnat05` est destiné à activer le support du standard Ada 2005 par le compilateur Gnat, lequel a le bon goût de nous avertir que la ligne 8, bien que syntaxiquement correcte, va lever une exception (avec même l'indication de la position dans la ligne). Ce que l'on peut vérifier à l'exécution :

```
$ ./interception_1
raised CONSTRAINT_ERROR : interception_1.adb:8 range check failed
```

On obtient bien l'exception annoncée, `Contraint_Error`, lors de l'exécution de la ligne 8. Modifions dans un premier temps le programme de façon à intercepter toutes les exceptions. La syntaxe utilisée en Ada rappelle un peu celle du C++ ou de Python, les instructions à « protéger » étant inclues dans un bloc, les traitements d'exceptions étant donnés ensuite comme dans un choix multiple (`case...when`) :

```
 7 begin
 8    begin
 9       t(0) := 1 ;
10       Open(File => f,
11            Mode => In_File,
12            Name => "n_importe_quoi") ;
13    exception
14       when others =>
15          Put_Line("Oups, une erreur !") ;
16    end ;
17    Put_Line("Fin du programme.") ;
18 end Interception_2 ;
```

Les exceptions interceptées seront donc celles levées durant l'exécution des lignes 9 à 12. Les traitements sont introduits

par le mot-clef exception et terminés par la fin (end) du bloc, entre les lignes 13 et 16. La clause when others marque le début d'un traitement dédié à toutes les exceptions, le mot others étant à rapprocher de celui utilisé dans un bloc case. Avec cette modeste modification, notre programme ne plante plus :

```
$ ./interception_2
Oups, une erreur !
Fin du programme.
```

Seulement, c'est un peu grossier comme traitement d'exception. Il serait préférable d'avoir un traitement spécialisé par type d'exception, ainsi que de récupérer le message associé. Modifions à nouveau le programme, en utilisant le paquetage Ada.Exceptions, qui contient quelques outils pratiques liés aux exceptions :

```
 1 with Text_IO ;
 2 use  Text_IO ;
 3 with Ada.Exceptions ;
 4 procedure Interception_3 is
 5    type TTab is array(1..10) of Integer ;
 6    t: TTab ;
 7    f: File_Type ;
 8 begin
 9    begin
10       t(0) := 1 ;
11       Open(File => f,
12            Mode => In_File,
13            Name => "n_importe_quoi") ;
14    exception
15       when Ex: Constraint_Error =>
16          Put_Line("Erreur de contrainte : " &
17             Ada.Exceptions.Exception_Message(Ex)) ;
18       when Ex: others =>
19          Put_Line("Oups, une erreur : " &
20             Ada.Exceptions.Exception_Name(Ex) & ": " &
21             Ada.Exceptions.Exception_Message(Ex)) ;
22    end ;
23    Put_Line("Fin du programme.") ;
24 end Interception_3 ;
```

Le traitement spécialisé est introduit ligne 15. Le petit morceau « Ex: », facultatif, permet de donner un nom à l'exception reçue et donc d'en faire éventuellement quelque chose. Ce nom peut être ce que bon vous semble, en particulier quelque chose de plus explicite que Ex ; de plus vous pouvez choisir un nom différent pour chaque traitement, ici Ex apparaît deux fois par pur manque d'imagination. Derrière se trouve le nom de l'exception attendue, ou others pour un traitement général (comme ligne 18). Le message de l'exception est obtenu sous la forme d'une chaîne de caractères retournée par la fonction Exception_Message() du paquetage Ada.Exceptions : on peut ainsi l'afficher, ce qui aide toujours à la résolution du problème. L'exécution donne ceci :

```
$ ./interception_3
Erreur de contrainte : interception_3.adb:10 range check failed
Fin du programme.
```

C'est bien le traitement spécialisé qui a été exécuté. Le traitement général est toujours présent, lignes 18 à 21, affichant cette fois le nom de l'exception (obtenu par la fonction Exception_Name() de Ada.Exceptions) avec son message. Ce traitement général doit toujours apparaître après tous les traitements spécifiques.

Si on corrige l'erreur ligne 10, par exemple en changeant le 0 en 1, l'exécution donne :

```
$ ./interception_4
Oups, une erreur : ADA.IO_EXCEPTIONS.NAME_ERROR: s-fileio.adb:935
Fin du programme.
```

On tombe dans le traitement général, avec un message moins explicite. Cette exception est déclenchée par la tentative d'ouverture d'un fichier inexistant, lignes 11-13. Notez qu'elle n'apparaissait pas auparavant : le bloc d'instructions entre les lignes 9 et 13 est en effet interrompu dès qu'une exception survient. Donc quand l'instruction incorrecte ligne 10 est exécutée (avant qu'elle ne soit corrigée), les instructions suivantes sont tout simplement sautées. Ce qui est un comportement tout à fait normal. Notez le nom de l'exception : cela ressemble à ce que pourrait être le nom d'un objet contenu dans un paquetage IO_Exceptions, lui-même contenu dans le paquetage Ada... Voyons comment sont déclarées et déclenchées les exceptions.

Déclaration et déclenchement

La déclaration d'une exception se fait par une syntaxe horriblement complexe :

```
Nom_Exception: exception ;
```

C'est simple, mais cela ne permet pas de définir une hiérarchie d'exceptions, comme on peut le faire en C++. C'est là un désagrément avec lequel il va falloir vivre. Le déclenchement d'une exception se fait au moyen du mot-clef raise. Par exemple :

```
 1 with Text_IO ;
 2 use  Text_IO ;
 3 with Ada.Exceptions ;
 4 use  Ada.Exceptions ;
 5 procedure Decl_1 is
 6    Pas_Paire: exception ;
 7    procedure Test_Paire(i: in Integer) is
 8    begin
 9       if (i mod 2) /= 0
10       then
11          raise Pas_Paire
12             with Integer'Image(i) & " n'est pas paire" ;
13 --          Raise_Exception(Pas_Paire'Identity,
14 --             Integer'Image(i) & " n'est pas paire") ;
15       end if ;
16       Put_Line(Integer'Image(i) & " est paire") ;
17    end Test_Paire ;
18 begin
19    begin
20       Test_Paire(2) ;
21       Test_Paire(3) ;
22    exception
23       when Ex: Pas_Paire =>
24          Put_Line(Exception_Message(Ex)) ;
25    end ;
26    Put_Line("Fin du programme.") ;
27 end Decl_1 ;
```

Une exception nommée Pas_Paire est déclarée ligne 6. La procédure Test_Paire()

Quelques exceptions prédéfinies — Tableau 1

Paquetage	Exception	Description
Standard	Constraint_Error	Sans doute la plus fréquente, cette exception est levée dès l'utilisation d'une valeur en dehors de l'intervalle de définition de son type. Elle est également levée si vous tentez d'accéder au contenu d'un pointeur nul.
	Storage_Error	Levée si vous tentez de réserver plus de mémoire qu'il n'en est disponible.
	Program_Error	Assez rare, cette exception survient lorsque le programme se trouve dans un état incohérent, par exemple si un sous-programme dans un paquetage est invoqué alors que ce paquetage n'a pas encore été élaboré (c'est-à-dire, initialisé).
Ada.Strings	Index_Error	Liée aux opérations sur les chaînes de caractères, cette exception survient si un indice donné à un sous-programme (comme la recherche d'une sous-chaîne) est invalide.
Ada.Direct_IO Ada.Directories Ada.IO_Exceptions Ada.Sequential_IO Ada.Streams.Stream_IO Ada.Text_IO	Name_Error	Exception déclenchée si vous tentez d'accéder à un fichier inexistant ou inaccessible.
	End_Error	Déclenchée si vous tentez de lire au-delà de la fin d'un fichier.
	Data_Error	Déclenchée si, lors de la lecture d'un fichier, les caractères reçus ne correspondent pas au type attendu.

vérifie si l'entier reçu en paramètre est pair ou non ; s'il ne l'est pas, l'exception Pas_Paire est levée (ligne 11). Celle-ci est interceptée ligne 23. Résultat :

```
$ ./decl_1
 2 est paire
 3 n'est pas paire
Fin du programme.
```

La syntaxe utilisée ici pour l'instruction raise est la nouvelle forme introduite par le standard Ada 2005. Auparavant, une exception était levée simplement par :

```
raise Nom_Exception ;
```

Cette forme est toujours disponible. Il est maintenant possible d'y adjoindre un membre « with "chaîne" » afin d'associer un message à l'exception, qui pourra être récupérée par Exception_Message() (ligne 24). En Ada95, pour obtenir le même effet, il fallait avoir recours à la procédure Raise_Exception() du paquetage Ada.Exceptions : un exemple est donné par les deux lignes commentées 13 et 14.

Types et attribut

Toute exception possède un identifiant, dont le type est Exception_Id déclaré dans Ada.Exceptions. Cet identifiant peut être obtenu à partir du nom de l'exception avec l'attribut 'Identity, comme cela est fait ligne 13 dans l'exemple précédent. Réciproquement, le nom de l'exception peut être retrouvé à partir de l'identifiant à l'aide de Exception_Name(), qui retourne une chaîne de caractères. Wide_Exception_Name() et Wide_Wide_Exception_Name() font de même, mais retournent respectivement une chaîne Unicode 16 et une chaîne Unicode 32. Nous avons vu qu'il était possible de donner un « nom » à une exception interceptée, par exemple le nom Ex ligne 23 du programme précédent. Ce nom est en fait celui d'une variable de type Exception_Occurrence (déclarée dans Ada.Exceptions) : ce n'est pas le nom de l'exception, mais un nom donné à une instance de celle-ci. Divers sous-programmes prennent un tel objet en paramètre, comme Exception_Message() (déjà rencontré), Exception_Identity() (pour obtenir l'identifiant) ou Exception_Name().

Exceptions prédéfinies

Le standard du langage impose l'existence de quelques exceptions, que vous êtes assuré de retrouver sur n'importe quelle implémentation. Celles-ci sont déclarées dans divers paquetages, selon l'emploi qui en est fait. À noter qu'un même nom d'exception peut apparaître dans différents paquetages : il s'agit alors naturellement de différentes exceptions, chacune possédant son propre identifiant. Voici quelques-unes des exceptions les plus courantes (consultez la section Q-4 du standard Ada pour une liste exhaustive Tabl. 1). Pour mémoire, le paquetage Standard n'a pas besoin d'être préfixé : il est toujours supposé visible et utilisé par défaut (comme si une clause use Standard était systématiquement insérée au début de tout programme ou paquetage).

Conclusion

Voilà pour cette petite présentation des exceptions en langage Ada. Pour plus de détails, voyez la section 11 du manuel de référence Ada [4], plus particulièrement la sous-section 11.4.1 consacrée au paquetage Ada.Exceptions. La prochaine fois, nous aborderons les aspects « orienté objet » du langage Ada, ce qui nous occupera quelques temps.

LIENS

- [1] Ada 2005 : http://adaic.org/standards/ada05.html
- [2] Ada WikiBook : http://en.wikibooks.org/wiki/Ada_Programming
- [3] Enquête sur le marché du langage Ada : http://adaic.org/news/survey-results.html
- [4] Le manuel de référence Ada 2005 : http://www.adaic.com/standards/05rm/html/RM-TTL.html

Yves Bailly,