Exceptions

Objectifs

— Comprendre et savoir utiliser les exceptions
— Comprendre la différence entre programmation offensive et programmation défensive

Exercice 1 : Propagation et traitement d'exceptions

Dans cet exercice, plusieurs programmes sont fournis, ils correspondent à différents cas de mise en œuvre des exceptions. Pour chaque programme, il est demandé de préciser les éléments affichés et de décrire le comportement du programme.

1. Indiquer ce qui est affiché lorsque le programme suivant est exécuté alors que l'utilisateur saisit le caractère `e`.

Listing 1 – Le programme `Exemple_1`

```
1  with text_io;             use text_io;
2  with ada.integer_text_io; use ada.integer_text_io;
3
4  procedure Exemple_1 is
5
6  -- spécification volontairement omise !
7  procedure Lire_Entier (FValeur : out Integer) is
8  begin
9      Put_Line ("Début lire_entier");
10     Get (FValeur);
11     Put_Line ("Fin lire_entier");
12 exception
13     when Data_Error =>
14         Put_Line ("Erreur de saisie dans lire_entier");
15 end Lire_Entier;
16
17 ------------------------------------------------ Programme principal ----
18     Nb : Integer; -- le nombre à lire
19 begin
20     Put_Line ("Début instructions du programme Exemple 1");
21     Lire_Entier (Nb);
22     Put_Line ("Fin instructions du programme Exemple 1");
23 end Exemple_1;
```

2. Indiquer ce qui est affiché lorsque le programme suivant est exécuté alors que l'utilisateur saisit le caractère `e`.

Listing 2 – Le programme `Exemple_2`

```
1  with text_io;              use text_io;
2  with ada.integer_text_io; use ada.integer_text_io;
3
4  procedure Exemple_2 is
5
6  -- spécification volontairement omise !
7  procedure Lire_Entier (FValeur : out Integer) is
8  begin
9      Put_Line ("Début de lire_entier");
10     Get (FValeur);
11     Put_Line ("Fin de lire_entier");
12 end lire_entier;
13
14 ------------------------------------------------ Programme principal ----
15     Nb : Integer; -- le nombre à lire
16 begin
17     Put_Line ("Début de Exemple_2");
18     Lire_Entier (Nb);
19     Put_Line ("Fin de Exemple_2");
20 exception
21     when Data_Error =>
22             Put_Line ("Erreur de saisie");
23 end Exemple_2;
```

3. Indiquer ce qui est affiché lorsque le programme suivant est exécuté alors que l'utilisateur saisit le caractère `e`.

Listing 3 – Le programme `Exemple_3`

```
1  with text_io;              use text_io;
2  with ada.integer_text_io; use ada.integer_text_io;
3
4  procedure Exemple_3 is
5
6  -- spécification volontairement omise !
7  procedure lire_entier (FValeur : out Integer) is
8
9      -- spécification volontairement omise !
10     procedure Lire_Interne (FValeur_Interne : out Integer) is
11     begin
12         Put_Line ("Début de Lire_Interne");
13         Get (FValeur_Interne);
14         Put_Line ("Fin de Lire_Interne");
15     end lire_interne;
16
17 begin
18     Put_Line ("Début de lire_entier");
19     lire_interne (FValeur);
20     Put_Line ("Fin de lire_entier");
21 exception
22     when Data_Error =>
23            Put_Line ("Erreur de saisie dans Lire_Entier");
24 end lire_entier;
25
26 ------------------------------------------------ Programme principal ----
27     Nb: Integer; -- le nombre à lire
28 begin
29     Put_Line ("Début de exemple_3");
30     Lire_Entier (Nb);
31     Put_Line ("Fin de exemple_3");
32 end Exemple_3;
```

4. Indiquer ce qui est affiché lorsque le programme suivant est exécuté alors que l'utilisateur saisit le caractère `e`.

Listing 4 – Le programme `Exemple_4`

```
1  with text_io;              use text_io;
2  with ada.integer_text_io; use ada.integer_text_io;
3
4  procedure Exemple_4 is
5
6  -- spécification volontairement omise !
7  procedure Lire_Entier (FValeur : out Integer) is
8
9      -- spécification volontairement omise !
10     procedure Lire_Interne (FValeur_Interne : out Integer) is
11     begin
12         Put_Line ("Début de Lire_Interne");
13         Get (FValeur_Interne);
14         Put_Line ("Fin de Lire_Interne");
15     end lire_interne;
16
17 begin
18     Put_Line ("Début de lire_entier");
19     Lire_Interne (FValeur);
20     Put_Line ("Fin de lire_entier");
21 end lire_entier;
22
23 ------------------------------------------------ Programme principal ----
24     Nb : Integer; -- le nombre à lire
25 begin
26     Put_Line ("Début de exemple_4");
27     Lire_Entier (Nb);
28     Put_Line ("Fin de exemple_4");
29 exception
30     when Data_Error =>
31            Put_Line ("Erreur de saisie");
32 end Exemple_4;
```

5. Indiquer ce qui est affiché lorsque le programme suivant est exécuté alors que l'utilisateur saisit le caractère `e` puis l'entier 2.

Listing 5 – Le programme `Exemple_5`

```
1  with text_io;              use text_io;
2  with ada.integer_text_io; use ada.integer_text_io;
3
4  procedure Exemple_5 is
5
6  -- spécification volontairement omise !
7  procedure Lire_Entier (FValeur : out Integer) is
8  begin
9      Put_Line ("Début lire_entier");
10     Get (FValeur);
11     Put_Line ("Fin lire_entier");
12 exception
13     when Data_Error =>
14         Put_Line ("Erreur de saisie dans lire_entier");
15         Skip_Line;
16         Lire_Entier (FValeur);
17 end Lire_Entier;
18
19 ------------------------------------------------ Programme principal ----
20     Nb : Integer; -- le nombre à lire
21 begin
22     Put_Line ("Début de Exemple_5");
23     Lire_Entier (Nb);
24     Put_Line ("Fin de Exemple_5");
25 end Exemple_5;
```

Exercice 2 : Utilisation des exceptions
Considérons le programme du listing 6.

Listing 6 – Le programme `Somme`

```
1  with ada.text_io;            use ada.text_io;
2  with ada.integer_text_io;    use ada.integer_text_io;
3
4  -- Calculer la somme d'une suite d'entiers lus clavier.  L'entier 0 marque la
5  -- fin de la série. Il n'en fait pas partie.
6  procedure Somme is
7      Somme  : Integer;    -- la somme de valeurs lues au clavier
8      Valeur : Integer;    -- valeur lue au clavier
9  begin
10     -- calculer la somme d'une suite de valeurs entières, se terminant par 0
11     Somme := 0;
12     loop
13         Put ("Entrez une valeur entière : ");
14         Get (Valeur);
15         Somme := Somme + Valeur;
16     exit when Valeur = 0;
17     end loop;
18
19     -- afficher la somme
20     Put ("la somme vaut : ");
21     Put (Somme, 1);
22     New_Line;
23 end Somme;
```

1. Expliquer pourquoi ce programme n'est pas robuste.
2. Modifier le programme afin que la lecture d'une donnée de type incorrect provoque l'affichage du message « Saisie invalide » (et pas la somme).
3. Modifier le programme pour qu'il s'arrête sur la première saisie invalide et affiche la somme en précisant avant « Attention, somme partielle ! ».
4. Modifier le programme pour qu'il ignore les saisies invalides (il affichera juste « saisie invalide... mais on continue ! ») et affiche la somme des entiers.

Exercice 3 : Module Piles et programmation défensive
Dans cet exercice, nous partons du module *Piles* (listing 7 pour son interface et 8 pour son implantation) qui a été écrit dans un style de programmation dite *offensive*. On souhaite le modifier pour adopter un style de programmation dite *défensive*.
1. Comparer programmation défensive et programmation offensive.
2. Modifier le module *Piles* pour qu'il mette en œuvre la programmation défensive.
3. Écrire un programme qui empile une suite d'entiers strictement positifs lus au clavier. Il s'arrête dès que l'utilisateur saisit un entier négatif ou nul. Ce programme devra afficher le message « Plus de place » lorsque la capacité de la pile est atteinte et demandera alors à l'utilisateur s'il veut continuer en lui proposant de dépiler un nombre (demandé à l'utilisateur) d'éléments pour continuer. Le programme devra être robuste.

Listing 7 – L'interface du module *Piles*

```
1  -- Spécification du module Piles.
2
3  generic
4      Capacite : Integer;   -- Nombre maximal d'éléments qu'une pile peut contenir
5      type T_Element is private;  -- Type des éléments de la pile
6
7  package Piles is
8
9      type T_Pile is private;
10
11     -- Initilaiser une pile.  La pile est vide.
12     procedure Initialiser (Pile : out T_Pile) with
13         Post => Est_Vide (Pile);
14
15     -- Est-ce que la pile est vide ?
16     function Est_Vide (Pile : in T_Pile) return Boolean;
17
18     -- Est-ce que la pile est pleine ?
19     function Est_Pleine (Pile : in T_Pile) return Boolean;
20
21     -- L'élément en sommet de la pile.
22     function Sommet (Pile : in T_Pile) return T_Element with
23         Pre => not Est_Vide (Pile);
24
25     -- Empiler l'élément en somment de la pile.
26     procedure Empiler (Pile : in out T_Pile; Element : in T_Element) with
27         Pre => not Est_Pleine (Pile),
28         Post => Sommet (Pile) = Element;
29
30     -- Supprimer l'élément en sommet de pile
31     procedure Depiler (Pile : in out T_Pile) with
32         Pre => not Est_Vide (Pile);
33
34 private
35
36     type T_Tab_Elements is array (1..Capacite) of T_Element;
37
38     type T_Pile is
39         record
40             Elements : T_Tab_Elements;    -- les éléments de la pile
41             Taille: Integer;        -- Nombre d'éléments dans la pile
42         end record;
43
44 end Piles;
```

Listing 8 – L'implantation du module *Piles*

```
1  -- Implantation du module Piles.
2
3  package body Piles is
4
5      procedure Initialiser (Pile : out T_Pile) is
6      begin
7          Pile.Taille := 0;
8      end Initialiser;
9
10     function Est_Vide (Pile : in T_Pile) return Boolean is
11     begin
12         return Pile.Taille = 0;
13     end Est_Vide;
14
15     function Est_Pleine (Pile : in T_Pile) return Boolean is
16     begin
17         return Pile.Taille >= Capacite;
18     end Est_Pleine;
19
20     function Sommet (Pile : in T_Pile) return T_Element is
21     begin
22         return Pile.Elements (Pile.Taille);
23     end Sommet;
24
25     procedure Empiler (Pile : in out T_Pile; Element : in T_Element) is
26     begin
27         Pile.Taille := Pile.Taille + 1;
28         Pile.Elements (Pile.Taille) := Element;
29     end Empiler;
30
31     procedure Depiler (Pile : in out T_Pile) is
32     begin
33         Pile.Taille := Pile.Taille - 1;
34     end Depiler;
35
36 end Piles;
```