الفصل العاشر

معالجة الأخطاء
Error Handling

Programming Concepts 10

فى هذا الفصل

في هذا الفصل نعرض موضوع الأخطاء الوارد حدوثها مع المبرمج ومع مستخدم البرنامج وذلك من خلال النقاط التالية:

- الأنواع المختلفة للأخطاء : Bugs
- الخطأ الهجائي Syntax error
- الخطأ أثناء التشغيل Runtime Error
- ON ERROR GOTO-
- ON ERROR RESUME NEXT-
- الخطأ المنطقي Logic Error
- طرق حل الأخطاء :
- طرق تلافى خطأ أثناء التشغيل
- طرق تتبع الخطأ المنطقي ومعرفة سببه
- استعمال الشاشات المختلفة
- استعمال التركيب الجديد Try.....Catch

عند القيام بإعداد برنامج يصادفك أخطاء ما أثناء الإعداد أو بعد توزيع البرنامج للاستخدام ، ومن عوامل نجاح البرنامج عدم ظهور أي أخطاء ، وفيما يلي نوضح أنواع الأخطاء وطرق تلافيها .

أنواع الأخطاء :

1 ـ الخطأ الهجائي Syntax Error

وهو أبسط أنواع الأخطاء حيث يظهر عند كتابة نصوص البرنامج وهو خطأ في الحروف الهجائية للأوامر المعرفة للغة أو خطأ في قواعد اللغة فمثلاً عند كتابة جملة For بدون Next أو العكس.

- وعند كتابة جملة If بدون End If.
- عند كتابة أمر MSGBOX ناقص حرف وهكذا.
- وهذا الخطأ يظهر سريعاً ويسهل علاجه.

ولتوضيح ذلك سوف نقوم بشرح مثال توضح فيه النوع الاول من الأخطاء SYNTAX ERROR كما يلي:

مثال :

1. ابدأ برنامج جديد بالخطوات المعتادة من النوع Windows Application .
2. وقع زر أمر Button على الـ FORM كما هو معتاد.
3. اضغط بالماوس على زر الأمر Button1 مرتين لتحصل على دالة Click الخاصة به واكتب بها السطور التالية :

```
Dim i As Integer
Dim a(10) As Integer
For i = 0 To 40
a(i) = I
```

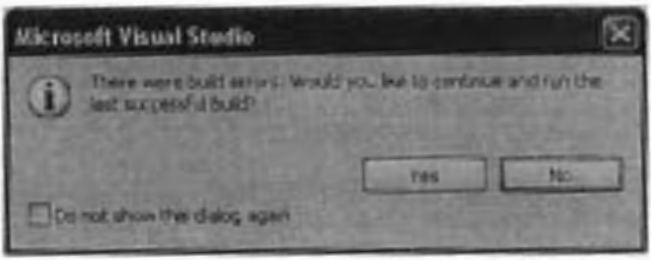
فى لغة C#

```
89          private void button1_Click(object sender, System.EventArgs e)
90          {
91              int i;
92              for (i=0;i<5)
93                  MessageBox.Show (i+"");
94          }
```

فى لغة Java

```
1  public class Test1 {
2
3      public static void main(String[] args) {
4      System.out.print("test the Syntax Error ");
5      }
6  }
```

نفذ البرنامج (F5) تلاحظ ظهور رسالة خطأ كما بالشكل (1-10).

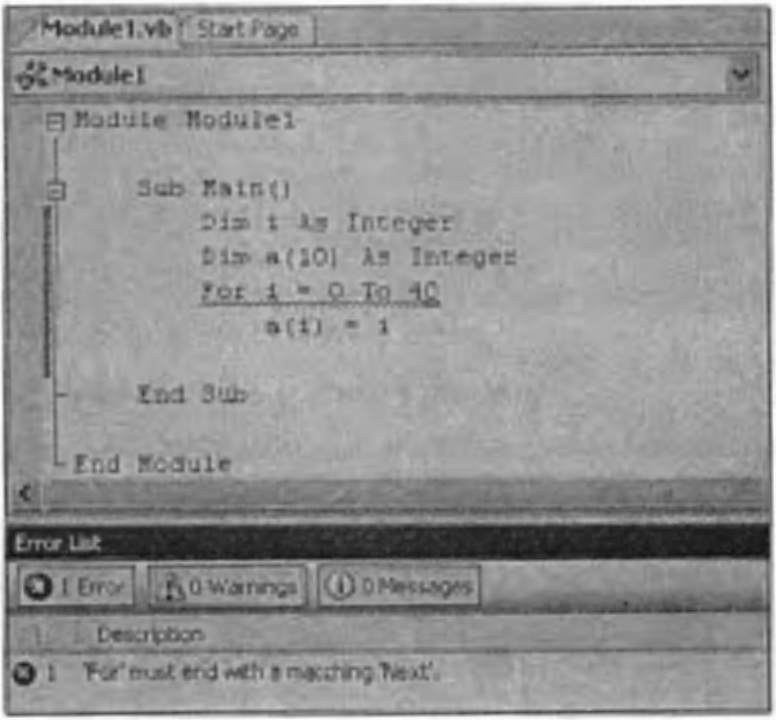


الشكل(1-10)

فى لغة Java

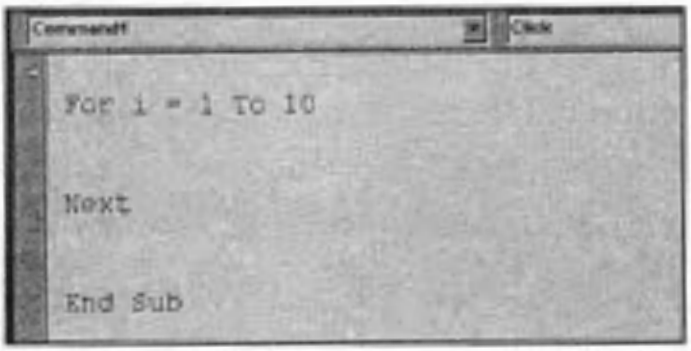
تظهر رسالة الخطأ كما فى الشكل:

1. اضغط زر No تلاحظ وجود رسالة الخطأ فى مربع Task Lists أسفل الشاشة اضغط الرسالة مرتين بالماوس تدخل الى محرر السطور مكان الخطأ اكتب كلمة NEXT I كما بالشكل (2-10).



الشكل(2-10)

نفذ البرنامج F5 يتم تنفيذ البرنامج بعد علاج الخطأ واختفاء رسالة الخطأ، وهذا هو النوع الأول من الأخطاء وهذه هى طريقة علاجه ، يظهر مع أى خطأ فى قواعد اللغة ويسهل تجنب أو تقليل هذه الأخطاء باستعمال القاعدة التى تظهر عند كتابة الأمر فمثلا عند كتابة الدالة MSGBOX يظهر تركيب الدالة بالصورة الصحيحة كما فى الشكل (3-10) .



الشكل(3-10)

فى لغة C#

قم بتصحيح الخطأ ليصبح كما فى السطور.

```
89          private void button1_Click(object sender, System.EventArgs e)
90          {
91              int i;
92              for (i=0;i<5;i++)
93                  MessageBox.Show (i+"");
94          }
```

و المقصود بهذا الشكل أننا كتبنا الأمر بهيكله الصحيح أولاً.

النوع الثانى : الخطأ أثناء التشغيل : Runtime Error

لا يظهر هذا الخطأ أثناء أعداد البرنامج الا إذا قمت بتجربة البرنامج بينما يظهر للمستخدم نتيجة قيام المبرمج بكتابة أوامر تحقق عملية غير محققة أثناء التنفيذ فمثلاً:

أ-عند صدور أمر فتح ملف من مشغل أقراص (A,B,...) والمشغل غير جاهز تظهر رسالة خطأ RunTime Error ويتقطع البرنامج ويعود الى بيئة التشغيل

ب - عند صدور أمر فتح قاعدة بيانات غير موجودة وهكذا ويعتبر هذا النوع من الأخطاء التى تعيب البرنامج فعندما يظهر للمستخدم الخطأ يقطع البرنامج ويخرج وتعتبر هذه عيوب فى البرنامج ، وبالتالى يجب تلافى هذا الخطأ والا ينقطع العمل فى البرنامج فمثلاً يمكن التأكد من وجود القرص قبل تنفيذ أمر فتح الملف أو وجود قاعدة البيانات أو غيره ومن أشهر هذه الأخطاء القسمة على القيمة صفر Divid By Zero حيث تعطى خطأ Over Flow ولتوضيح ذلك تابع المثال التالى:

1-وقع زر أمر Button واكتب به السطور التالية:

```
Dim d, i, r As Integer
    d = 100
    i = 0
    r = d / i
    MsgBox(r)
```

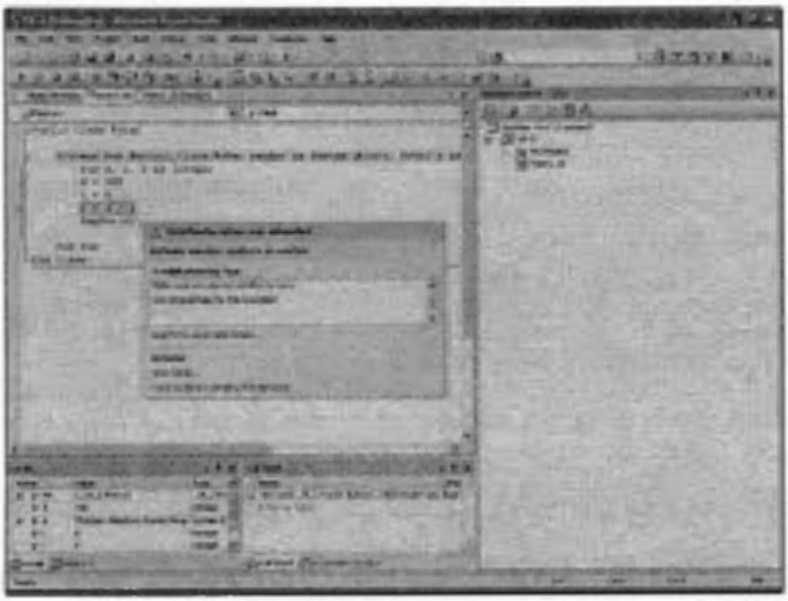
فى لغة C#

```
89          private void button1_Click(object sender, System.EventArgs e)
90          {
91              int d1=100;
92              int d2=0;
93                  int result = d1/d2;
94          }
```

فى لغة Java

```
1  public class Test1 {
2
3      public static void main(String[] args) {
4      int k;
5      k=100/0;
6      }
7  }
```

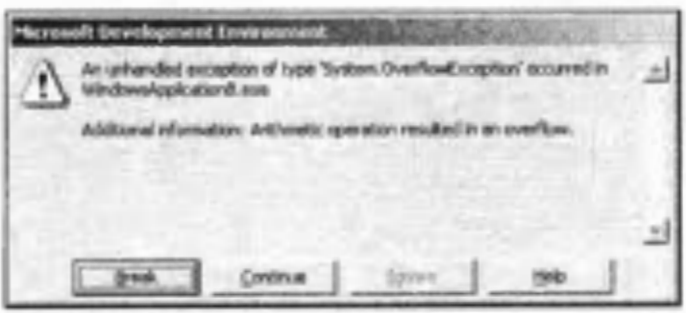
- فى هذه السطور يتم القسمة على المتغير I الذى يساوى 0 نفذ البرنامج واضغط زر الأمر Button1 تحصل على خطأ كما فى الشكل (4-10).



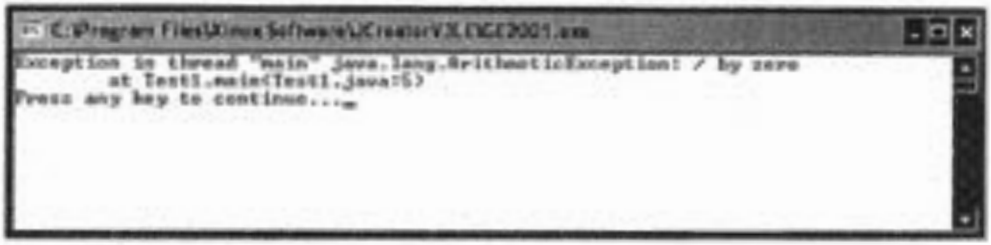
الشكل(4-10)

فى C#

تظهر رسالة الخطأ التالية



فى لغة Java



طرق علاج الخطأ

توجد أكثر من طريقة لعلاج هذا النوع من الأخطاء حسب أهمية العملية وهى :

1- استعمال الجملة ON Error Resume Next.

2- استعمال الجملة ON Error GOTO.

3- استعمال مصفوفة الأخطاء.

4- استعمال التركيب الجديد Try.....Catch.

فى Vb.Net

1. الجملة On Error Resume Next

تستخدم هذه الجملة لتلاشى الخطأ وتكتب لكى تجاهل تنفيذ الجملة التى تسبب الخطأ والانتقال الى الجملة التالية و لن يتم تنفيذ الجملة ولا الاشارة الى عدم تنفيذها ويتم استعمالها عندما تريد ذلك وذلك عندما لا توجد أهمية لإظهار رسالة الخطأ.

ولتوضيح ذلك نعود الى البرنامج السابق ونجرب استعمال هذه الجملة كما يلى :

1- افتح المثال السابق وأعد كتابة السطور لتصبح كما يلى :

```
On Error Resume Next
   Dim d, i, r As Integer
d = 100
i = 0
r = d / i
MsgBox(r)
```

2- نفذ البرنامج كما سبق.

3- اضغط الزر Button1 كما سبق تلاحظ عدم ظهور رسالة الخطأ ولكن عدم ربط ذلك بالخطأ وعدم ظهور ما يفيد ذلك.

2. الجملة ON ERROR GO TO

تستخدم هذه الجملة لتوجيه تنفيذ البرنامج الى مجموعة سطور تظهر رسالة الخطأ وتتولى علاجه وهى أفضل من الطريقة السابقة حيث تظهر رسالة مفيدة توضح موقف البرنامج ولتوضيح ذلك أعد فتح البرنامج السابق وتابع ما يلى :

اضغط أداة Button1 مرتين وأعد كتابة السطور لتصبح كما بالشكل (5-10) :

```
Dim lab As Label
On Error GoTo lab
Dim d, i, r As Integer
d = 100
i = 0
r = d / i
MsgBox(r)

lab:
MsgBox(Err.Description)
```

الشكل(5-10)

فى هذه السطور :

- فى السطر رقم 2 تم استعمال العبارة ON ERROR GOTO lab بمعنى إذا حدث خطأ فى السطور التالية ووجه التنفيذ إلى الخطوة التى يوجد أمامها العنوان lab .
- فى السطر 3 الأمر exit sub للخروج من الامر ما لم يصل إلى الخطأ.
- فى السطر رقم 4 العنوان lab لبداية جمل معالجة الأخطاء.
- فى السطر 5 يتم عرض رسالة الخطأ الهدف ثقق الذى يشير الى الخطأ والرسالة يتم عرضها بالخاصية DESCRIPTION.
- فى السطر رقم 6 يتم عرض رقم الخطأ.
- ولكن الرسالة تظهر باللغة الإنجليزية ولاظهارها باللغة العربية أعد كتابة السطور لتصبح كما بالشكل (6-10) :

```
59          Dim lab As Label
60          On Error GoTo lab
61          Dim d, i, r As Integer
62          d = 100
63          i = 0
64          r = d / i
65          MsgBox(r)
66      lab:
67          '   MsgBox(Err.Number)
68          If Err.Number = 6 Then
69              MsgBox("تم القسمة على صفر")
70          End If
71      End Sub
```

الشكل(6-10)

في هذه السطور نختبر رقم الخطأ اذا كان 6 وهو رقم الخطأ الذي يعني القسمة على صفر يتم عرض رسالة "تم القسمة على صفر".

ولكن بهذا الأسلوب لابد أن نختبر أرقام الأخطاء لتلافي الأخطاء المتوقعة لذلك يفضل استخدام الحل الثالث وهو مصفوفة الأخطاء.

استعمال التركيب الجديد Try.....Catch

هذا الأسلوب يعتبر جديد في الاصدار Vb.Net ولكنه كان موجود في اللغة VC++ وكذلك لغة Java ويأخذ الشكل.

```
Try
Statment1
Statment2

Catch Exception1
Action1

Catch Exception2
Action2

Catch Exception3
Action3
End Try
```

- فى هذه السطور نبدأ بالكلمة Try بمعنى حاول تنفيذ الجمل التالية وهى Statment1,Statment2 فاذا تم تنفيذهم بنجاح اذهب الى End Try وتابع البرنامج
- واذا ظهرت مشاكل أذهب الى مجموعة جمل Catch (أى امسك) التى تتعامل مع الخطأ الناتج.
- فاذا كان الخطأ الناتج من النوع Exception1 نفذ Action1 و كان الخطأ الناتج من النوع Exception2 نفذ Action2.

وهكذا ولتوضيح ذلك تابع الخطوات التالية:

فى Vb.NET

1- فى أول البرنامج قم بكتابة السطر التالى

```
Imports System.IO
```

فى هذا السطر تم استعمال الأمر imports للإشارة إلى الـ IO NameSpace

2-وقع زر أمر Button واكتب بداله السطور التالية كما بالشكل (8-10).

```
62          Dim filest As FileStream
63
64          Try
65              filest = New FileStream("a:\data.txt", FileMode.Open)
66          Catch ex As FileNotFoundException
67              MsgBox("The file could not be found.")
68              Exit Sub
69          Catch ex As IOException
70              MsgBox("An error occurred: " & ex.Message)
71              Exit Sub
72          End Try
73
```

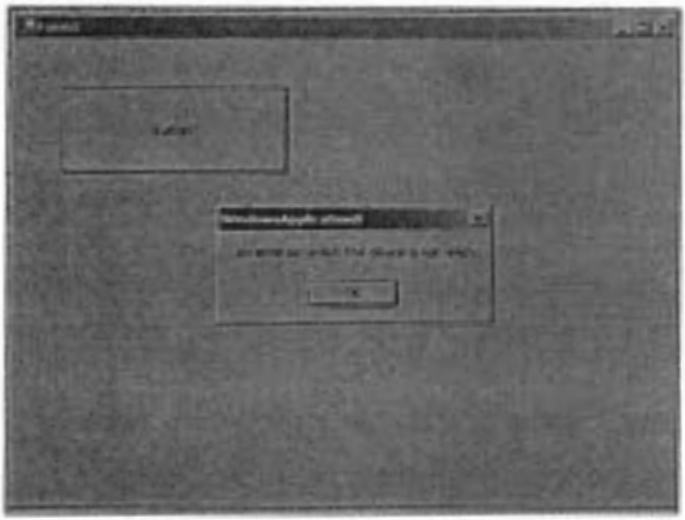
الشكل(8-10)

فى هذه السطور:

يتم تعرف متغير كم النوع filestream لتتمكن من فتح ملف والتعامل معه ثم محاولة فتح هذا الملف من مشغل الديسكات A فإذا كان الديسك غير موجود يتحول الخطأ إلى IOException وتظهر الرسالة الخاصة بهذ الخطأ.

وإذا كان الديسك موجود ولكن الملف غير موجود بالديسك يتحول الخطأ إلى FileNotFoundException وتظهر الرسالة الخاصة بهذا الخطأ.

3-نفذ البرامج واضغط زر الأمر Button1 يحاول الكمبيوتر فتح الملف data.txt من مشغل الديسكات A ثم تظهر رسالة خطأ كما فى الشكل (9-10).



الشكل(9-10)

النوع الثالث : الخطأ المنطقى Logical Error

لا يعطى هذا النوع أى رسائل خطأ بل أن البرنامج قد يعمل جيداً بدون مشاكل مع ظهور الرسائل المعبرة وكل شئ سليم ولكن الخطأ فى نتيجة البرنامج :

فمثلاً : عند قيام برنامج المرتبات بطباعة صافى مرتب كل موظف يطبع المرتب خطأ بالرغم من تحقق جميع العلميات ويرجع هذا النوع إلى منطق البرنامج وخط السير، فقد يكون معد البرنامج وضع علامة الطرح مكان علامة الجمع أو ما شابه ذلك أو وضع قاعدة خطأ باستعمال جملة If وغير ذلك .

وفى حالة بساطة البرنامج يسهل اكتشاف الخطأ بتتبع سطور البرنامج وتحديد الأخطاء ولكن فى حالة البرامج الكبيرة يصعب تتبع سطور البرنامج لذلك توجد طرق لإكتشاف الخطأ المنطقى.

طرق اكتشاف الخطأ المنطقى

وتنقسم هذه الطرق الى مجموعتين:

أ - التحكم فى سير البرنامج لمعرفة السطر الذى به الخطأ.

ب - إظهار معلومات عن متغيرات أو تعبيرات فى نوافذ خاصة.

(أ) التحكم فى سير البرنامج

ولتحقيق ذلك استخدم شريط أدوات Debug كما يلى :

1- قم باظهار شريط أدوات Debug وذلك بعمل نقرة يالطرف الأيمن للـ mouse أثناء وضعه فوق شريط القائمة تظهر قائمة اختار منها debug كما بالشكل (10-10).

الشكل(10-10)

2- هذه القائمة توفر أدوات تسهل عملية متابعة سير البرنامج

جرب استعمال هذه المفاتيح من خلال برنامج كما يلى :

استعمال نقاط التوقف Break Points

تعتبر فكرة اضافة نقاط توقف Break Points فى سطور البرنامج بحيث يتوقف تنفيذ البرنامج عنده لمشاهدة قيم المتغيرات ومتابعة البرنامج عندها من أشهر وأقدم طرق اكتشاف الخطأ المنطقى.

ويتم ذلك باختيار السطر الذى ترغب فى وضع نقطة توقف Break Point عنده ثم الضغط على المفاتيح Ctrl+B.

ثم تنفيذ البرنامج تلاحظ أنه يتوقف عند نقطة التوقف كلما ضغط على F5.

الاستفادة من الـ Snippet

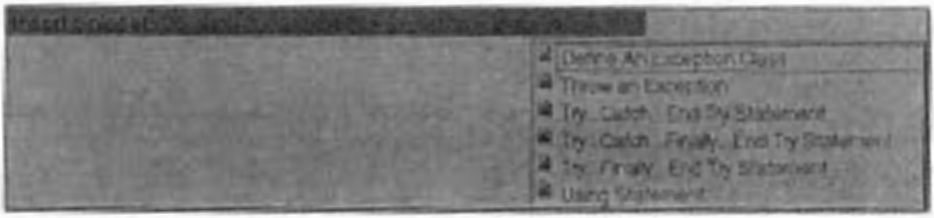
1- توفر قطع الكود الجاهزة Snippet بعض العمليات المفيدة مع أخطاء البرنامج كما فى الشكل (11-10).

الشكل(11-10)

2- وعند اختيار Common Code Patterns تحصل على قائمة أخرى كما فى الشكل (12-10).

الشكل(12-10)

3- اختر Exception Handling تحصل على مجموعة من الاكواد كما فى الشكل (13-10).



الشكل(13-10)

وفيما يلى عرض بعض هذه الأكواد.

```
Try
    Catch ex As ApplicationException
    End Try
```

فى هذه السطور:

- يتم عرض الشكل التقليدى للتركيب Try...Catch.. Enf Try كما عرضناه.

اضافة الامر Finally

```
    Try
    Catch ex As ApplicationException
    Finally
    End Try
```

وهكذا باقى قطع الكود.

أدوات معالجة الإستثناءات فى لغة Java

معذرة ربما أطلنا عليك بتناول هذا الموضوع بدون أمثلة كثيرة .. ولكن هذا الموضوع مهم ، حيث لا يمكن أن يتجاهله مبرمج، وإدارة معالجة الإستثناءات تتم من خلال خمسة كلمات محجوزة Keywords وهى الكلمات.

```
finally, try , catch , throw, throws
```

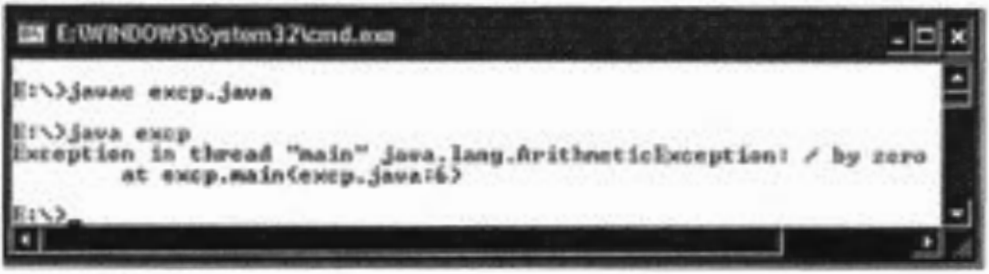
فتعال معى نرى كيفية استخدامها.

التركيب Try ------ catch

أول أدوات معالجة الإستثناءات هم التركيب try ------ catch وهما من أسهل الأدوات ولكى نرى كيفية عملها فلنلقى نظرة على نص البرنامج التالى:

```
1.  class Excp
2.  {
3.  public static void main (String Arg[ ])
4.  {
5.  int d= 0;
6.  int a=60 /d;
7.  System.out.println("After Exception");
8.      {
9.  {
10.
```

لو حاولت ترجمة ثم تنفيذ هذا المثال ستجد أن المترجم قد اظهر الرسالة التالية:



من الواضح أن البرنامج قد ترجم Compile بنجاح ولكن عند التنفيذ اظهر هذه الرسالة وهذه الرسالة تخبرنا بالمعلومات التالية:-

- هناك خطأ ظهر أثناء وقت التنفيذ ولاكتشاف الخطأ نقرأ الرسالة فنجد انه حدث استثناء Exception وهذه أول كلمة فى الرسالة لكن أين حدث هذا الاستثناء؟ نجد أن الرسالة تقول أنة فى Thread main اى انه حدث خطأ عند تنفيذ الدالة الرئيسية main ولكن ما نوع هذا الاستثناء نجد انه من نوع Arithmetic وقد نتج عن محاولتنا القسمة على صفر وهذا ناتج عن الجملة رقم 6 ونجد أن الرسالة تدلنا على اسم الفصيلة التى حدث بها الاستثناء ورقم السطر الذى حدث به الاستثناء كما هو واضح بين الأقواس.

من كل ما سبق نستخلص أن الدالة main حدث بها استثناء Exception وقد ألقت الاستثناء إلى نظام وقت التنفيذ Runtime system وقام نظام وقت التنفيذ بالبحث عن من يعالج هذا الاستثناء فلم يجد فألقى الاستثناء ثم خرج من النظام . وستجد أيضا أنه لم يصل إلى السطر 7 لأنه توقف عند الاستثناء الحادث.

ولمعالجة هذا الاستثناء تعالى معى ننظر إلى نسخه معدله من البرنامج فى الجزء التالى:

```
2.  class Excp
3.  {
4.  public static void main (String [] arg)
5.  {
6.  try{
7.  int d=0;
```

```
8.  int a;
9.  a =60/d
10.  {catch (ArithmeticException  e)
11. }
12. System.out.println("division by zero");
13. System.out.println(e.getMessage());

14. }
15. System.out.println("After Exception");
16. }
```

- حاول تنفيذ البرنامج الآن ستجد أن تم التنفيذ بدون مشاكل وستجد ان السطر 14 قد تم تنفيذه فماذا حدث.
- نجد أن الذى حدث أن الدالة أخبرت المترجم بأن يحاول تنفيذ الكود فى المنشأ try Block وإذا حدث استثناء وهو ما حدث يتم التقاطة عند طريق المنشأ catch Block وتم إظهار الرسالة التى تفيد حدوث القسمة على صفر.
- واخيراً تم ظهور نتيجة السطر 14 لأن الاستثناء الحادث قد تم إلتقاطة ومعالجته.
- الصورة العامة try –catch كالأتى:

```
try {
 // الكود الذى يحدث استثناء
}catch(Exception Object Declaration)
{
}
```

تعدد الكلمة catch

فى بعض الأحيان قد ينتج عن كود ما أكثر من استثناء فما الحل فى هذه الحالة؟

الحل أن يوجد العديد من معالج الاستثناء Exception Handler وفى حالة استخدام try catch ---- تكون catch هى معالج الاستثناء وبالتالى لابد من وجود أكثر من بلوك catch ويصبح تركيب البرنامج كما فى السطور التالية:

```
try {
// Code that might generate exceptions
} catch(Type1 id1) {
// Handle exceptions of Type1
} catch(Type2 id2) {
// Handle exceptions of Type2
} catch(Type3 id3) {
// Handle exceptions of Type3
```

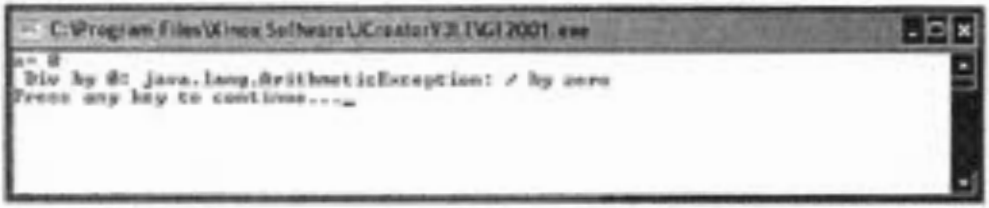
ولتوضيح ذلك تابع معنا سطور البرنامج التالية :

```
1  public class Test1 {
2
3     public static void main(String[] args) {
4
5  try {
6  int a= args.length;
7  System.out.println("a= " + a);
8  int b = 42/a;
9  int c [] = {1};
10 c [42 ] = 99;}
11 catch (ArithmeticException e)
12 {
13 System.out.println(" Div by 0: "+e);
14 }
15 catch (ArrayIndexOutOfBoundsException e)
16 {
17 System.out.println(" array index Out of Bounds "+e);
18 }
19
20    }
```

- فى هذا المثال ينتج استثناءان.
- الأول ناتج عن القسم على صفر وذلك كما فى السطر 7 إذا بدأنا البرنامج بدون إعطاؤه أى معاملات.
- والثانى ناتج عن محاولة تخطى سعه المصفوفة c فقد أعلنا أن سعتها عنصر واحد وذلك كما فى السطر 8 ثم فى السطر 9 نحاول تخصيص قيمة للعنصر 43 بينما المصفوفة سعتها عنصر واحد فقط.
- لذلك قمنا بإعداد اثنان من معالجى الاستثناء Exception Handler وذلك باستخدام اثنان من المنشئات Block من catch.
- المنشأ Block الأول يعالج الاستثناء الناتج عن محاولة القسمة على صفر وذلك كما يتضح فى السطور من 11 إلى 14 .

- المنشأ Block الثانى يعالج الاستثناء الناتج عن محاولة تخطى سعه المصفوفة e وذلك كما يتضح فى السطور من 15-18.

ويظهر ذلك من نتيجة التنفيذ كما فى الشكل.



معالجة الاستثناءات عن طريق الكلمة المحجوزة Throws

فى بعض الأحيان عندما نقوم بتصميم دالة Method فى فصيلة ما ونعرف أن هذه الدالة ستلقى باستثناء فمن المستحب ان نجعل مستخدم الدالة هو الذى يقوم بمعالجة الاستثناء وللتوضيح أكثر تعال معى نرى نص البرنامج التالى.

```
1.  class ThrowsDemo {
2.  static void delta() throws IllegalAccessException
3.  {
4.  System.out.println("inside delta");
5.  throw new IllegalAccessException ("demo");
6.  }
7.   public static void main (String args[ ])
8.   {
9.        try {
10.             delta();
11. }
12. catch (IllegalAccessException e)
13.{
14.       System.out.println(" caught" + e);
15.   }
16.  }
17.}
```

- فى هذا المثال قمنا بتصميم الدالة delta() وهذه الدالة نعرف إنها ستلقى باستثناء وفى هذه الحالة لابد من تحديد الاستثناء الذى ستلقيه الدالة حتى تستطيع الدوال الأخرى

عند استدعاء الدالة delta() حماية نفسها بتوفير معالج استثناء من نوع الاستثناء والذى ستلقيه delta() .

- والدالة delta ستلقى الاستثناء بالكلمة المحجوزة throws ثم نوع الاستثناء وذلك كما فى السطر 2 .
- والشكل العام لمثل هذه الدوال كالآتى:

Accesstype methodtype methodName (arg- list) throws Exception 1, Exception 2,....

وكمثال كما رأينا فى المثال السابق:

static void delta ( ) throws IllegalAccessException.

- ومن هذه الصورة العامه يتضح إننا نستطيع إرسال أكثر من استثناء.
- بعد السطر 2 جملة بناء الدالة Delta ولكن ما يهمنا هنا هو السطر رقم 5 وهذا السطر هو المسئول عن عملية إرسال الاستثناء ولاحظ أنه لو عندنا أكثر من استثناء لابد من إلقائهم بالأمر throw وليس throws.
- لاحظ أيضا أن الدالة انتهت عند السطر 6 ولم تهتم بمعالجة الاستثناء لأن هذا أصبح ليس مهمتها ولكن مهمة من يستدعى هذه الدالة وذلك كما فى السطور من 9-15.
- ففيالسطور 9-15 قامت الدالة main باستدعاء الدالة delta ولأن الدالة main تعرف ان الدالة delta ستلقى بإستثناء فقامت بحماية نفسها عند استدعاء الدالة delta بـ try-catch والتقطت الاستثناء وعالجته وذلك كما فى السطور من 11 -14.

معالجة الاستثناءات عن طريق الكلمة المحجوزة Throw

يستخدم الكلمة المحجوزة throw فى إلقاء استثناء كما أوضحنا فى المثال السابق ولكنه ليس بالضرورة أن يستخدم مع throws فيمكن أن يستخدم وحدة كما سنوضحه فى نص البرنامج التالى:

```
1.  class ThrowDemo {
2.        static void demo ()
3.        {
4.              try {
```

```
5.                  throw new NullPointerException ("demo");
6.                 } catch (NullPointerException e){
7.                 System.out.println("cought inside demo");
8.                 throw e ;
9.                 }
10.       }
11.    public static void main (String arg[])
12.    {
13.          try {
14.             demo ();
15.             } catch (NullPointerException e)
16.             {
17.                System.out.println("Recaught : " + e);
18.             }
19.     }
20. }
```

- إن الكلمة المحجوزة throw يستخدم لإرسال الاستثناء بطريقة إجبارية.
- ولكى نستخدم throw لابد من إنشاء هدف من نوع الاستثناء الذى أود أن أرسله وذلك كما فى السطر 5 ثم لابد من وجود معالج للاستثناء Exception Handler وذلك كما يوضحه السطر 6 وذلك إذا أردت ان أعالج الاستثناء داخل الدالة.
- أما إذا أردت إرسال استثناء لتعالجه دالة أخرى عندما نستدعى الدالة المعلن فيها الاستثناء فلابد أن إرسال يهدف كما يوضحه السطر 8 والذى يستقبله معالج الاستثناء الموجود فى دالة الاستدعاء كما فى سطر 15.

معالجة الاستثناءات عن طريق الكلمة المحجوزة Finally

عندما يتم ارسال استثناء Exception فإن تسلسل flow التنفيذ فى الدالة Method يتم تغييره ويتم تحويل التسلسل للبحث عن أقرب معالج للاستثناء وفى حالة عدم وجود معالج للاستثناء مثل catch لا يتم تنفيذ الجمل الموجودة بعد الموقع الحادث به الاستثناء.

إذن ماذا نفعل فى حالة إذا أردنا أن يتم تنفيذ جمل أو قطعة من الكود بغض النظر عن ما إذا كان الاستثناء قد تم معالجته أم لا؟

فى هذه الحالة يبرز لنا الكلمة المحجوزة finally حيث أنه يتم تنفيذ الجمل التى بداخله بغض النظر عن ما حدث للاستثناء فحتى ولو لم يكن هناك معالج مثل catch لالتقاط

الاستثناء ومعالجته فسيتم تنفيذ الكود الموجود داخل finally ويتم تنفيذ الموجود داخل finally مباشرة بعد try Block ولنوضح الكلام أكثر فى نص البرنامج التالى:

```
1. class FinallyDemo {
2. static void procA() {
3. try {
4. System.out.println("inside proc A");
5. throw new RuntimeException ("demo");
6. }
7. finally {
8. System.out.println("proc A finally");
9. }
10.}
11. static void procB() {
12. try {
13. System.out.println(" inside proc B");
14. return;
15.}
16. finally {
17. System.out.println("proc B funally");
18.}
19.}
20. public static void main (String arg [ ])
21.{
22. try {
23. procA();
24.}
25. catch (Exception e){
26. procB();)
27.}
28.}
```

- فى هذا المثال على الرغم من إلقاء الدالة procA() باستثناء كما يوضحه السطر 5 إلا أنه لا يوجد معالج للاستثناء فلا يوجد لدينا catch فكان من الطبيعى أن يتوقف عمل البرنامج ولكنه المترجم يجد أمامه finally فيضطر إلى إكمال التسلسل وتنفيذ البرنامج بدون مشاكل وذلك كما فى السطور 6-8.
- ثم نجدد شيئاً آخر غريب وهو الدالة procB وخصوصاً السطر 12 والموجود به جملة return فوجود تلك الجملة كان يعنى أن يتم كسر التسلسل والخروج ولكن لأن الدالة

بها finally فقد تم إجبار المترجم على تكملة التسلسل أما بقية البرنامج فسير الدالة طبيعى جدا حيث يتم استدعاء الدالة procA وذلك كما فى السطر 20 ولأننا نعرف أن الدالة الإرسال باستثناء فقد حمينا نفسنا منه بـ try --- catch.

- وفى السطر 22 استدعاء الدالة procB وحيث أن الدالة لا ترسل باستثناء فقد تم استدعاؤها بالطريقة المعتادة.

وعند التنفيذ نحصل على نتيجة التنفيذ كما فى الشكل.

ويصبح تركيب البرنامج كما فى الشكل.

```
try {
// The guarded region: Dangerous activities
// that might throw A, B, or C
} catch(A a1) {
// Handler for situation A
} catch(B b1) {
// Handler for situation B
} catch(C c1) {
// Handler for situation C
} finally {
// Activities that happen every time
```

إنشاء أنواع جديدة من الاستثناءات User Defined Exceptions

حتى الآن فإن كل أنواع الاستثناءات التى استخدمناها هى من الفصائل الموجودة فى اللغة وهذه الفصائل تغطى تقريباً كافة الاستثناءات الممكنة الحدوث ولكن ماذا نفعل إذا احتجنا استثناء غير موجود فى اللغة أى إذا كان الخطأ الذى يظهر غير معرف باللغة ؟

الحل هو إنشاء (تعريف) هذا الاستثناء فكيف يتم ذلك!؟

بما أن الاستثناءات فى الجافا هى فصائل Classes إذن فلنا الحق فى أن ننشىء فصائل ترث الفصائل الموجودة فى الجافا ونطيف إليها الجديد بما يناسب منطق البرنامج.

والفصيلة التى نقوم ببنائها لا تفعل أكثر من إصدار رسائل توضح أن هناك استثناء ما قد حدث وتبين نوعه وهذا للتوضيح فقط ولكن فى الواقع يتم إنشاء فصائل لعلاج أخطاء حقيقية بالبرنامج ويتضح ذها المثال من خلال السطور التالية :

```
1. class MyException extends Exception {
2. private int details;
3. MyException (int a)
4. {
5. detail=a;
6. }
7. public String toString()
8. {
9. return "My Exception [" + detail + " ]";
10. }
11. }
12. class UsingExceptionDemo
13. {
14. static void compute (int a) throws MyException
15. {
16. System.out.println("called compute (" + a + "). ");
17. if (a>15)
18. throw new MyException (a);
19. System.out.println("normal exit ");
20. }
21. public static void main (String args [])
22. {
23. try{
24. compute(1);
25. compute(20);
26. } catch (MyException e)
27. {System.out.println("caught" + e);}
28. }
29. }
```

فى هذه السطور:

- فى هذا المثال افترضنا أننا نريد أن ننشئ استثناء غير موجود.
- قمنا بإنشاء فصيلة وهذه الفصيلة ترث الفصيلة Exception وذلك كما فى السطر 1.
- فى هذه الفصيلة أردنا لو تم استدعاء دالة ما بقيمة أكبر من 15 فإنه يحدث استثناء وتظهر رسالة توضح قيمة الرقم فإنشاءنا Constructor يأخذ قيمة كما يوضحة السطر 3.

- ثم قمنا بعملية overload للدالة toString والتى مهمتها إظهار قيمة الإستثناء الذى حدث.
- ثم قمنا بإنشاء الفصيلة UsingExceptionDemo وقمنا بإنشاء دالة compute والتى تلقى باستثناء من النوع الذى أنشأته فى الفصيلة My Exception وذلك كما فى السطر 15.
- ثم حددنا شرط لحدوث الاستثناء وهو إذا كانت القيمة أكبر من 15 وذلك كما فى السطر 18 فيتم إلقاء الاستثناء أما إذا كانت القيمة أقل فلا يتم إلقاء الاستثناء.
- ثم قمنا باستدعاء الدالة compute مرتين كما فى السطرين 25،26 مرة بقيمة أقل من 15 فلا يحدث استثناء ومرة أكبر فيحدث استثناء وهكذا نجد أن النتيجة كالآتى:

```
C:\Java Exception Demo
    Called Compute(1)
    Normal exit (20)
    Caught My Exception (20)
```

وهكذا نكون قد انتهينا من الإستثناءات و طرق معالجتها.

يرجع مرة أخرى للكتاب.

فى Oracle PL/SQL

معنى Exception

الكلمة معناها استثناء وهو جزء من الأوامر يتم توجيه سير البرنامج إليه عند حدوث خطأ ليقوم هو بتناول الخطأ والقيام بعمل اللازم Error Handling بالإضافة إلى إظهار الرسالة وذلك بعد قيام المبرمج بتحديد العمل اللازم عند حدوث الخطأ وكذلك الرسائل المطلوب إظهارها عند حدوث الخطأ.

أنواع Exceptions

توجد ثلاث أنواع للاستثناء Exceptions وهى :

1- استثناء معرف Predefined Exception.

2- استثناء من تعريف المبرمج User defined Exception.

3- استثناء داخلي Internal Exception.

1ـ الاستثناء المعرف من قبل Predefined Exception

وهو استثناء معرف من قبل Oracle لمجموعة من الحالات والتعامل معها وتسمى Predefined أو built-in ، فمثلاً هناك خطأ معرف يسمى NO DATA عند قيام جملة Select بإعادة بيانات ولا تعيد يظهر No Data أو خطأ أخر وهو قيام جملة select بإعادة اكثر من سجل في حين أنك تستعمل select into حيث تخزن هذه البيانات فى متغير سجل واحد ، وهذه الأخطاء يتم التعامل معها من قبل Oracle ولكن بإظهار رسالة من داخل Oracle ولكن لتغير الرسالة والتعامل مع هذا الخطأ من قبل المبرمج تقوم بكتابة الأوامر فى جزء الـ Exception فى نهاية بلوك الأوامر كما فى شرح بلوك الدالة كما سبق.

وهذا النوع يتميز بأن الـ Oracle يعرف حالة حدوثه لذلك ما عليك سوى كتابة الأوامر التى تنفذ عند حدوث هذا الخطأ.

2ـ خطأ معرف من قبل المبرمج User_ defined exception

هذا النوع لا يعرفه الـ Oracle ولكن يتم تعريفه من قبل المبرمج لذلك يسمى User defined وله ثلاثة أجزاء هى :

1ـ جزء التعريف Exception declaration

وهو جزء الإعلان عن اسم هذا النوع من الخطأ لأنه غير معرف لدى Oracle فيتم تعريف متغير جديد من النوع exception

2ـ اختبار الخطأ Exception teasing

وفيه يتم وضع شرط حدوث هذا الخطأ لأن الـ Oracle لا يعرف معناه ولا يعرف وقت حدوثه ليوجه سير البرنامج إلى قسم الـ Exception.

3ـ التعامل مع الخطأ Exception handling

وهو قسم الـ Exception التقليدى الذى ينتظر أن يوجه إليه الخطأ بعد تعريف وتحديد حالة حدوثه للقيام باللازم وتنفيذ الأوامر والرسائل التى يكتبها المبرمج.

ولتوضيح هذا الإجراء تابع السطور التالية:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

في هذه السطور:

يبدأ التعامل مع الاخطاء بقسم الاخطاء بالكلمة EXCEPTION.

يلي ذلك الامر WHEN الذي يعني عندما ويليه نوع الخطأ المحدد بكلمات معرفة بديل للعبارة exception1 ومن هذه القيم المعرفة ما يلي:

- NO_DATA_FOUND
- TOO_MANY_ROWS
- INVALID_CURSOR
- ZERO_DIVIDE
- DUP_VAL_ON_INDEX

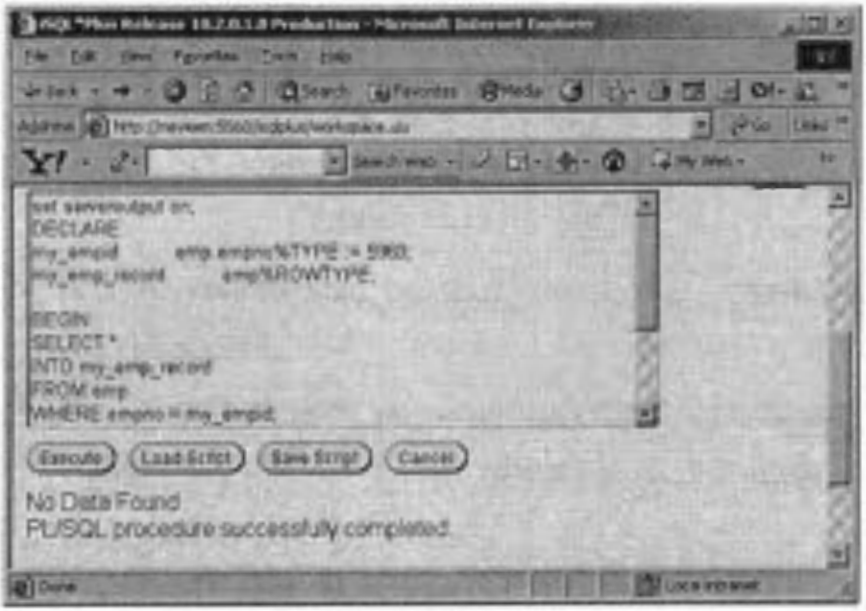
- وفيها العبارة NO_DATA_FOUND تستعمل عند تنفيذ أمر SELECT أو CURSOR ولكن لم تعيد بيانات.
- العبارة TOO_MANY_ROWS وهي عكس العبارة السابقة حيث تستعمل عندما تكون عدد القيم المسترجعة من الامر SELECT كبير بحيث لا يصلح وضعه في متغير واحد.
- المتغير INVALID_CURSOR وتنتج عندما يتم التعامل مع متغير على أنه CURSOR ولكنه ليس كذلك.
- المتغير ZERO_DIVIDE ويعني حدوث عملية قسمة على ZERO.

- المتغير DUP_VAL_ON_INDEX معناها محاولة الإضافة فى جدول مع محاولة تكرار القيمة فى الحقل PRIMARY KEY.

والسطور التالية توضح كيفية التعامل مع الأخطاء المعرفة :

```
DECLARE
my_empid     emp.empno%TYPE := 5969;
my_emp_record     emp%ROWTYPE;
BEGIN
SELECT *
INTO my_emp_record
FROM emp
WHERE empno = my_empid;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No Data Found');
 END;
```

كما فى الشكل (14-10).



الشكل (14-10)

فى هذه السطور:

- بعد الأمر BEGIN الذى يعنى بداية الأوامر يتم استدعاء الأمر SELECT لعرض بيانات الموظف رقم 5969 ولكن فعليا لا يوجد موظف بهذا الرقم.
- لذلك يحدث خطأ يعرفه الـ ORACLE وهو عدم وجود بيانات NO_DATA_FOUND فيؤدى ذلك الى انتقال التنفيذ الى قسم التعامل مع الأخطاء EXECPTION.
- فى القسم الخاص بالأخطاء EXCEPTION تم كتابة الجملة WHEN التى تقول عندما ويليها حالة الخطأ وهى NO_DATA_FOUND أى عندما يحدث الخطأ NO_DATA_FOUND أى لا توجد بيانات وبالتالى يتم طباعة الرسالة المحددة.

ومما سبق نلاحظ أن الصورة العامة للتعامل مع الأخطاء هى:

```
0. DECLARE
1. BEGIN
2. EXCEPTION
3. WHEN Exception1 THEN
4. Sequence of statements
5. WHEN Exception2 THEN
6. Sequence of statements
7. WHEN Exception3 THEN
8. Sequence of statements
9. END;
```

الدوال التى تتعامل مع الأخطاء

يوجد فى ORACLE بعض الدوال للتعامل مع الأخطاء منها الدالة SQLCODE وتعيد رقم يعبر عن رقم الخطأ الناتج والدالة SQLERRM تعيد الرسالة المحفوظة لهذا الخطأ والجدول التالى يوضح أمثلة لبعض القيم المرجعة من SQLCODE.

Example SQLCODE Values

SQLCODE Value	Description
0	No exception encountered
1	User-defined exception
+100	NO_DATA_FOUND exception
negative number	Another Oracle server error number

والسطور التالية توضح استعمال هذه الدوال والمتغيرات

```
DECLARE
  v_error_code      NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code := SQLCODE;
    v_error_message := SQLERRM;
    INSERT INTO errors
    VALUES(v_error_code, v_error_message);
END;
```

في هذه السطور:

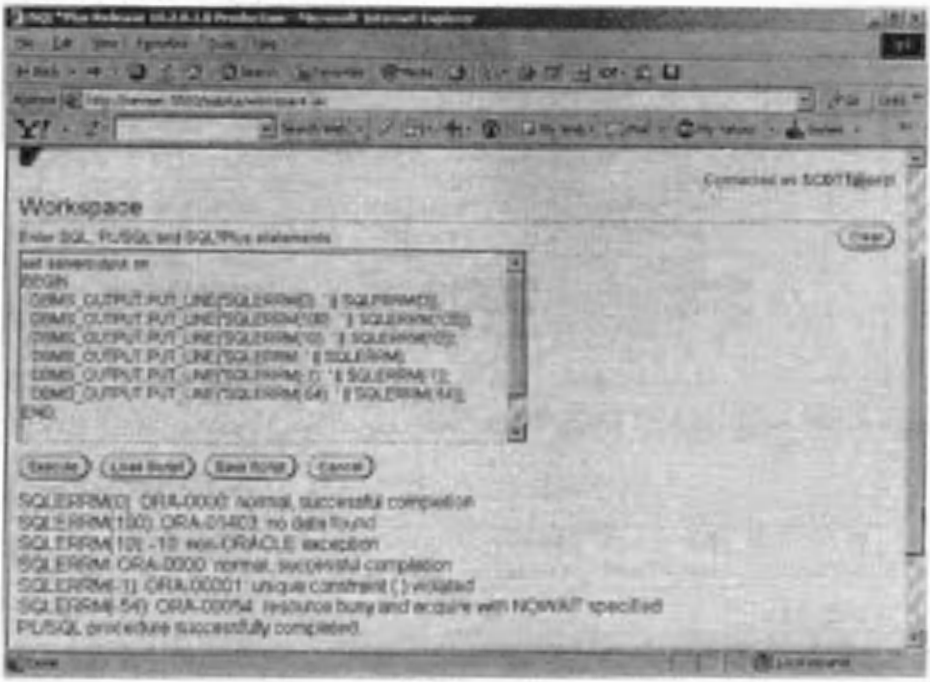
- تم تخزين القيمة الناتجة من SQLCODE في المتغير v_error_code وهي رقم الخطأ error الناتج.
- و تم تخزين القيمة الناتجة من SQLERRM في المتغير v_error_message وهي نص الخطأ error الناتج.

الدالة SQLERRM

الدالة SQLERRM تعرض الرسالة المقابلة للخطأ المحدد رقمه والسطور التالية توضح كيفية استعمالها لطباعة الرسائل المقابلة لبعض الأرقام.

```
/*
 * SQLERRM.sql
 */
set serveroutput on
BEGIN
  DBMS_OUTPUT.PUT_LINE('SQLERRM(0): ' || SQLERRM(0));
  DBMS_OUTPUT.PUT_LINE('SQLERRM(100): ' || SQLERRM(100));
  DBMS_OUTPUT.PUT_LINE('SQLERRM(10): ' || SQLERRM(10));
  DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
  DBMS_OUTPUT.PUT_LINE('SQLERRM(-1): ' || SQLERRM(-1));
  DBMS_OUTPUT.PUT_LINE('SQLERRM(-54): ' || SQLERRM(-54));
END;
/
```

وعند تنفيذ هذه السطور نحصل على نتيجة التنفيذ التالية: كما فى الشكل(15-10).

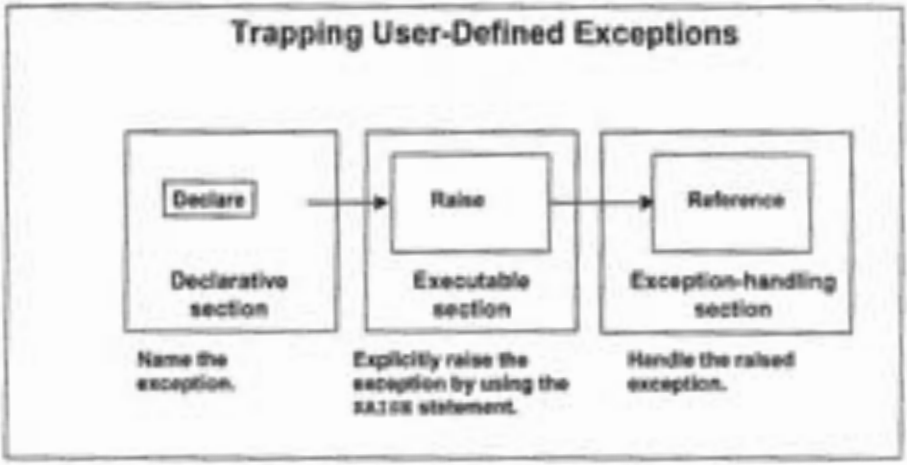


الشكل (15-10)

التعامل مع الاخطاء المعرفة من المبرمج USER DEFINED ERRORS

الأخطاء المعرفة من قبل المبرمج USER DEFINED ERRORS هى أخطاء لا يعرفها الـ ORACLE وربما لا تكون أخطاء مباشرة ولكنها من وجهة منطق البرنامج أخطاء ، فمثلا عندما تزيد نسبة نجاح الطالب عن 100% يعتبر البرنامج ذلك خطأ فى حين لا يعرف الـ ORACLE ذلك وبالتالى يقوم المبرمج بتعريف معنى وحالة حدوث هذا الخطأ وكيفية التعامل.

ويأخذ ذلك التركيب التالى:



والمثال التالى يوضح كيفية تعريف خطأ EXCEPTION من قبل المبرمج ثم التعامل معه فى قسم الـ EXCEPTION.

```
DEFINE p_department_desc = 'Information Technology '
DEFINE P_department_number = 300
```

```
DECLARE
   e_invalid_department EXCEPTION;
BEGIN
  UPDATE       departments
  SET          department_name = '&p_department_desc'
  WHERE        department_id = &p_department_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department  THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

فى هذه السطور:

- تم تعريف متغيرين الأول أخذ عبارة حرفية والثانى أخذ قيمة رقمية وذلك فى بيئة SQL PLUS.
- تم الإعلان عن متغير بالاسم v_invalid_department من النوع EXCEPTION.
- بعد الأمر BEGIN تم استعمال الأمر UPDATE للتعديل فى بيانات الإدارة رقم 300 حسب المتغير المعرف من قبل.

- ثم تم استعمال الأمر IF لاختبار هل تم التعديل أم لا وبالطبع لا لعدم وجود الموظف فتستدعى الجملة RAISE التى تنقل التنفيذ إلى قسم الأخطاء EXCEPTION لتجد الأمر WHEN الذى يتعامل مع الخطأ.

مثال:

نوضح كيفية تعريف خطأ من قبل المستخدم من النوع USER DEFINED ERROR وكيفية التعامل معه وذلك كما فى السطور التالية:

```
DECLARE
my_empid employee.empid%TYPE := 59694;
my_emp_record employee%ROWTYPE;
my_salary_null EXCEPTION;
BEGIN
SELECT * FROM employee
INTO my_emp_record
WHERE empid = my_empid;

IF my_emp_record.salary IS NULL THEN
RAISE my_salary_null;
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No Data Found');
WHEN my_salary_null THEN
DBMS_OUTPUT.PUT_LINE('Salary column was null for employee');
END;
```

فى هذه السطور:

- فى السطر رقم 4 تم الإعلان عن exception بالاسم salary_ NULL.EXR مع ملاحظة أن نوعه EXCEPTION.
- في السطر رقم 9 تم استعمال جملة IF للاختبار قيمة حقل المرتب SALARY فإذا كانت NULL تم استعمال الأمر RAISE للإرسال التنفيذ إلى القسم EXCEPTION إلى السطور الخاصة بهذا الخطأ .
- في السطر رقم 14 تم استعمال WHENمع اسم الخطأ Salary_ NULL.EXR أي في حالة حدوث هذا الخطأ THEN أي إذن قم بتنفيذ ما يلي.
- وهو طباعة الرسالة.