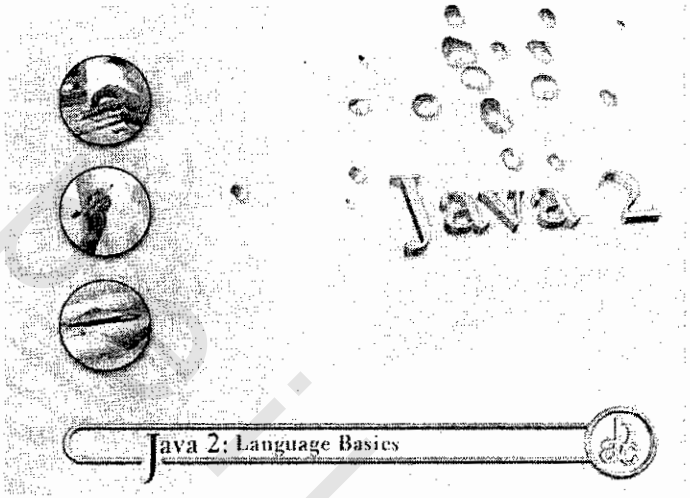
الفصل الثانى عشر



الاستثناءات ومعالجتها
Exception Handling

فى هذا الفصل سوف نتناول موضوع الاستثناءات Exceptions ومعالجتها وذلك من خلال النقاط التالية:

- أنواع الأخطاء.
- الخطأ الهجائى Syntax Error والخطأ أثناء التشغيل Runtime Error والخطأ المنطقى Logical Error.
- طرق اكتشاف الخطأ المنطقى Logical Error.
- ما هو الاستثناء Exception ولماذا الاهتمام به.
- نشر الأخطاء ونقلها إلى Call Stack.
- تجميع أنواع الأخطاء والتمييز بينها.
- أدوات معالجة الاستثناءات Exceptions Handling فى لغة Java.
- الصورة العامة لـ try ... catch وتعدد الجملة catch.
- معالجة الاستثناءات Exceptions Handling عن طريق الكلمة المحجوزة throws و throw و finally.
- إنشاء أنواع جديدة من الاستثناءات User Defined Exceptions.

مقدمة:

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

أنواع الأخطاء:

(1) الخطأ الهجائى Syntax Error :

- وهو أبسط أنواع الأخطاء حيث يظهر عند كتابة نصوص البرنامج ، وهو يمثل خطأ فى الحروف الهجائية للأوامر المعروفة في اللغة أو خطأ في قواعد اللغة.
- فمثلاً عند كتابة جملة for بدون أقواس بعدها أو عند كتابة جملة else بدون وجود جملة if قبلها أو عند كتابة أمر println ناقص حرف وهكذا.
- وهذا الخطأ يظهر سريعاً ويسهل علاجه.
- ولتوضيح ذلك سوف نقوم بشرح مثال نوضح فيه النوع الأول من الأخطاء الهجائية Syntax Errors كما يلى في المثال التالي.

مثال (1): الأخطاء الهجائية Syntax Errors :

- إبدأ برنامجاً جديداً بالخطوات المعتادة واكتب بها السطور التالية.

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

- نفذ البرنامج وستلاحظ ظهور رسالة خطأ كما بالشكل (12-1).

description	resource	folder	location
cannot find symbol method println(java.lang.String)	Test1.java	C:\Program Files\Xinox Software\JCreate	line 4

(الشكل 12-1)

- لاحظ وجود رسالة الخطأ فى مربع Task Lists أسفل الشاشة.
- اضغط الرسالة مرتين بالماوس Double-click لتنتقل إلى محرر السطور في مكان الخطأ.

- اكتب الكلمة الصحيحة println بدلاً من printl (أي قم بزيادة حرف n) ثم نفذ البرنامج وستجد أنه سيتم تنفيذ البرنامج بعد علاج الخطأ ويتم اختفاء رسالة الخطأ.
- وهذا هو النوع الأول من الأخطاء وهذه هى طريقة علاجه ، حيث يظهر هذا الخطأ مع أى خطأ فى قواعد اللغة.
- وبالطبع ستتجنب أو تقلل من هذه الأخطاء بالخبرة والممارسة.

النوع الثانى: الخطأ أثناء التشغيل Runtime Error:

- لا يظهر هذا الخطأ أثناء إعداد البرنامج إلا إذا قمت بتجربة تنفيذ البرنامج ، حيث يظهر هذا الخطأ للمستخدم نتيجة قيام المبرمج بكتابة أوامر تحقق عملية غير قابلة للتنفيذ.
- فمثلاً ، عند صدور أمر فتح ملف من مشغل أقراص (A, B, ...) وكان المشغل غير جاهز ، فتظهر رسالة خطأ Run Time Error وينقطع البرنامج ويعود إلى بيئة التشغيل.
- أيضاً عند صدور أمر فتح قاعدة بيانات غير موجودة ، تظهر رسالة خطأ وهكذا.
- ويعتبر هذا النوع من الأخطاء التى تعيب البرنامج ، فعندما يظهر للمستخدم الخطأ يتم إنهاء البرنامج ويتوقف عن العمل.
- وتعتبر هذه عيوب فى البرنامج ، وبالتالى يجب تلافى هذا الخطأ وإلا سينقطع العمل فى البرنامج.
- فمثلاً يمكن التأكد من وجود القرص قبل تنفيذ أمر فتح الملف أو وجود قاعدة البيانات أو غيره.

النوع الثالث: الخطأ المنطقى Logical Error:

- لا يعطى هذا النوع أى رسائل خطأ ، بل إن 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 ، ووجدت أن البرنامج يطبع المرتب خطأ بالرغم من تحقق جميع العلميات ، فإن هذا الخطأ يرجع إلى منطق البرنامج وخط السير ، فقد يكون المبرمج وضع علامة الطرح مكان علامة الجمع أو ما شابه ذلك ، أو وضع قاعدة خطأ باستعمال جملة if وغير ذلك.
- وفى حالة بساطة البرنامج فإنه يسهل اكتشاف الخطأ بتتبع سطور البرنامج وتحديد

الأخطاء ، ولكن فى حالة البرامج الكبيرة فإنه يصعب تتبع سطور البرنامج ، لـذلك توجد بعض الطرق لاكتشاف الخطأ المنطقى Logical Error.

طرق اكتشاف الخطأ المنطقى Logical Error:

تنقسم هذه الطرق الى مجموعتين:

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

ب - إظهار معلومات عن قيم المتغيرات فى نوافذ خاصة.

- وفى هذا الفصل ، نتناول النوع الثانى من الأخطاء وهـو الخطـأ الـذى يحـدث أثنـاء التنفيذ ويسمى Run Time Error ، وهو ما يطلق عليه الاسـتثناءات Exceptions ، فلماذا يطلق عليه ذلك وكيف يمكن معالجته؟
- تابع معنا الفقرات التالية.

ما هو الاستثناء Exception ولماذا الاهتمام به:

- إن كلمة استثناء Exception هى إشارة للعبارة Exceptional Event ، أى وقوع حدث استثنائى.
- والاستثناء Exception فى البرمجة هو وقوع حدث أثناء تنفيذ البرنامج مما يؤدى إلى تعطيل التسلسل الطبيعى لتعليمات instructions البرنامج.
- إذاً نستطيع القول بأن الاستثناء هو حدوث خطأ ما ، وهـذا الخطـأ لـيس خطـأ فـى الكود Syntax Error ، ولكنه قد يكون له العديد من المصـادر والأسـباب ، مثـل وجود عيب فى المكونات المادية للجهاز كأن تحاول الكتابـة علـى قطـاع تـالف bad sector فى القرص الصلب ، أو أن يكون الخطأ فى البرنامج مثل أن تحاول التعديل فى سجل ما فى قاعدة بيانات Database وليس لك الحق فى ذلك وهكذا.
- وعند حدوث مثل هذه الأخطاء خلال تنفيذك لأحد الدوال Methods فـى لغـة Java ، تقوم الدالة Method بإنشاء هدف Object من نوع الاستثناء الذي حـدث Exception Object ويتم تمرير هذا الهدف object إلى نظام وقت التنفيذ Runtime System.
- وهـذا الهـدف Exception Object يحتـوى علـى المعلومـات الخاصـة بالاسـتثناء (الخطأ) Exception ، وهذه المعلومات تشتمل علـى نـوع الاسـتثناء Exception وحالة البرنامج عند حدوث هذا الخطأ.

- عندئذ يكون نظام وقت التنفيذ Runtime System مسئولاً عن إيجاد جزء الكود المسئول عن معالجة هذا الاستثناء (الخطأ) Exception.
- وفى لغة Java تسمى عملية إنشاء هدف الاستثناء Exception Object وتمريره إلى نظام وقت التنفيذ Runtime System بعملية إرسال الاستثناء Throwing an Exception.
- وبعد أن تقوم الدالة method بإرسال الاستثناء throw Exception ، ينطلق نظام وقت التنفيذ Runtime System للبحث عن من يقوم بمعالجة الاستثناء Exception Handler ، والذى يقوم بمعالجة الاستثناء Exception Handler هو دالة method يتم شحنها إلى منطقة معينة فى الذاكرة تسمى Call Stack عند تنفيذ البرنامج ، فيقوم نظام وقت التنفيذ Runtime System بالبحث تراجعياً Backwards فى Call Stack بادئاً بالدالة method التى حدث فيها الاستثناء Exception حتى يجد الدالة method التى بها معالج الاستثناء Exception Handler المناسب.
- إذاً كيف يعرف نظام وقت التنفيذ Runtime System أن هذا المعالج Exception Handler هو المناسب؟
- يعتبر معالج الاستثناء Exception Handler مناسباً لو كان نوع الاستثناء الملقى Exception thrown هو من نفس نوع معالج الاستثناء Exception handler.
- لذلك لو تخيلنا أن الاستثناء Exception هو عبارة عن كرة تلقى من أسفل إلى أعلى فى Call Stack -الذى هو عبارة عن مجموعة من الدوال methods تحتوى على معالجات الاستثناء Exception Handlers- ، فستجد تلك الكرة تصعد وتستمر فى الصعود حتى تجد الدالة method التى بها المعالج Handler المناسب فيقوم بإمساك الاستثناء Exception ومعالجته.
- ونسمى معالج الاستثناء Exception Handler المختار باسم "ملتقط الاستثناء" Catch the Exception.
- ولكن قد يبدو هنا سؤال وهو: ماذا يحدث لو أن نظام وقت التنفيذ Runtime System قام بالبحث فى جميع الدوال methods الموجودة فى Call Stack ولم يجد معالج الاستثناء Exception Handler المناسب؟

- في هذه الحالة يقوم نظام وقت التنفيذ Runtime System بإنهاء البرنامج.
- وقد يتبادر سؤال هنا: ما المزايا التى يحصل عليها البرنامج باستخدام الاستثناءات Exceptions ، والإجابة أننا نحصل على ثلاثة مزايا ، وسنشرح كل منها بالترتيب:
  1- فصل كود معالجة الأخطاء عن باقى الكود.
  2- نشر الأخطاء ونقلها إلى Call Stack.
  3- تجميع وتصنيف كل خطأ بحيث يكون كل خطأ ينتمى إلى مجموعة عامة.

فصل كود معالجة الأخطاء عن باقى الكود:

- في البرمجة التقليدية كانت عملية اكتشاف الأخطاء وتصنيفها ومعالجتها تؤدى إلى تعقيد الكود Spaghetti code ، وكمثال على ذلك ، لنفترض أنه عندنا دالة method لقراءة ملف ، ولتبسيط المسألة سنكتبها فى صيغة جمل عادية فى خطوات كالآتى:

```
1. read_file
2. {
3. open The file
4. determine its size
5. allocate enough memory
6. read the file into memory
7. close the file
8. }
```

- وبنظرة سريعة لهذه الدالة method نجد أنها بسيطة ، ولكنها فى الواقع تجاهلت العديد من الأخطاء المتوقعة مثل:
  1- ماذا يحدث لو أننا لم نستطيع فتح الملف؟
  2- ما احتمال أن يكون الملف غير موجود فى المسار المحدد أو أن الملف به مشاكل ... تالف مثلاً.
  3- ماذا يحدث لو أننا لم نستطيع معرفة حجم الملف؟
  4- ماذا يحدث لو لم يكن هناك قدر كافى فى الذاكرة لفتح الملف؟
  5- ماذا يحدث لو أن قراءة الملف فشلت؟
  6- ماذا يحدث لو أننا لم نستطيع إغلاق الملف؟

والإجابة على هذه الأسئلة أن نقوم بإضافة الكثير من الكود لهذه الدالة method ، وقد تنتهى الدالة method إلى الشكل التالى:

```
    errorCodetype read_file {
  intialize errorcode =0;
  open the file ;
  if (the file is open)
  {
  determine the length of the file
  if (got the file length)
  {
  allocate that much memory
  if (got Enough memory)
  {
  red the faile into memory;
  if (read failed)
  {
  error code = -1;
  }
  } else
  {
     error code = -2; }
  } else
     error code = -3; }
  close the file
  if (the file didn't close && errorcode ==0)
  {
   error code = -4
  }
  } else
    error code =-5
  return error code;
}
```

وكما ترى من خلال الشكل الأول أنه لا يزيد عن سبعة سطور أما الشكل الثانى فيصل إلى 29 سطراً - وذلك بعد أن وضعنا آلية تتبع واكتشاف الأخطاء - ، ومن

خلال رؤية الكود فى الشكل الثانى للدالة method نجد أنه من الصعب جداً تتبع انسياب الدالة method flow.

- وهذا ما كان يتم فى اللغات القديمة وبالأسلوب التقليدى للبرمجة ، بالإضافة إلي استعمال التركيب المشهور on error goto الذى يستعمل بكثرة فى لغة Visual Basic.
- ولكن قامت لغة Java باستعمال أسلوب أخر استعملته كل اللغات الحديثة بعدها ، ويظهر هذا الأسلوب من خلال السطور التالية:

```
Read_file()
        {
          try
            {
          Open the file;
          Determine its size
          Allocate that much memory
          Read the file into memory
          Close the file
          } catch (file Open failed)
          {do some thing }
          catch (size Determination failed)
          { do some thing}
       catch (memory Allocation failed)
          { do something}
          catch (fileclosed failed)
          (do some thing}
}
```

- قارن بين هذه الصورة للدالة method والصورة السابقة ، تجد أن هذا الشكل أسهل فى الفهم وأيسر فى تصنيف الأخطاء ، وستجد أن الكود أقل ، مما يوفر الكثير من الجهد.
- وهذا الأسلوب أكثر تنظيماً حيث يضع سطور الأوامر الحقيقية للبرنامج - والتى من المتوقع حدوث بها خطأ أثناء التشغيل - في بلوك Block الأمر { } try ، ثم يتم وضع سطور التعامل مع هذه الأخطاء فى بلوك Block الأمر { }catch وكأنك تشير إلى هذا التركيب بالشكل التالى:

```
try
{
Program statements which may contains Run time Errors
}
catch
{
Handel try statements Run Time Errors
}
```

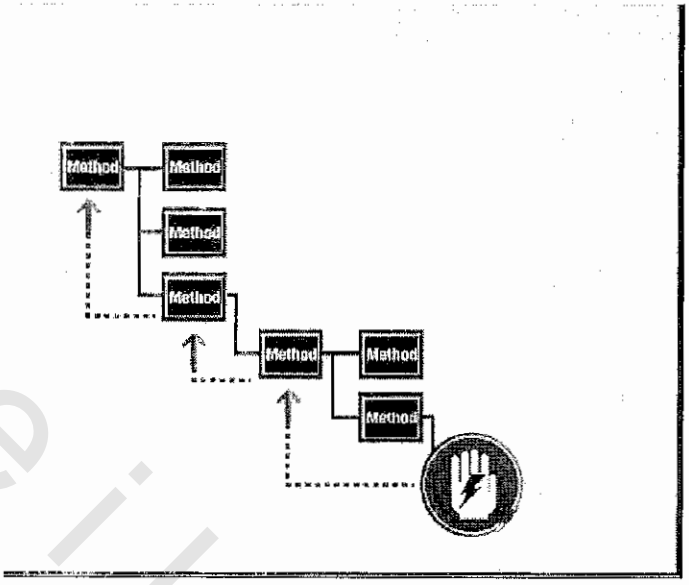
- وبهذا الشكل يكون الأمر أكثر نظاماً.

نشر الأخطاء ونقلها إلى Call Stack:

- وهذه هى الميزة الثانية للاستثناءات Exceptions وأسلوب لغة Java فى معالجة هذا النوع من الأخطاء ، ولكى يتضح ذلك تعال معى نعرض هذه المشكلة.
- لنفترض أن الدالة method السابقة read_file هى الدالة method الرابعة فى سلسلة متداخلة من الدوال methods التى تستدعى بعضها كالآتى:

```
Method1() {
        Call method 2;
              }
Method 2() {
        Call method 3;
              }
Method 3() {
        Call read_file
              }
```

- ولنفترض أيضاً أن الدالة Method الأولى هى الوحيدة المهتمة بالأخطاء التى تحدث فى الدالة read_file ، أى التى استعملت التركيب try{} catch {}.
- فى الطريقة التقليدية فإنه لكى يتم تبليغ الدالة Method1 بالخطأ الذى حدث بالدالة read_file ، فإنه يجب أولاً تبليغ الدالة Method3 ثم الدالة Method2 ثم الدالة Method1 كما فى الشكل التالى:



(الشكل 12-2)

```
Method1() {
    errorcode type error;
    error = call method2;
    if (error)
       do error processing;
    else
      continue;
    }
    errorcode type method 2{
    error codetype error;
    error = call method 3;
if  (error)
    return error;
    else
         continue;
}
    errorcodetype method 3{
    errorcodetype error;
    error = call read_file;
```

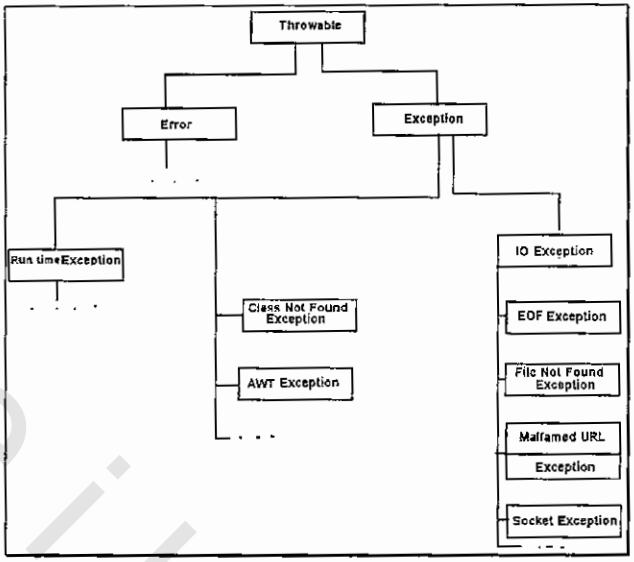
```
    if (error)
    return error;
     else
    continue;
}
```

وكما ذكرنا من قبل ، فإن نظام وقت التنفيذ للجافا Java Runtime System يقوم بالبحث تراجعياً فى Call Stack ليجد أى دالة method تقوم بمعالجة الاستثناء Exception Handling الحادث ، إذاً لماذا أجهد كل الدوال method بجعلها تقوم بالتقاط الاستثناء Catch an Exception وهى (أى الدوال methods) غير مهتمة بهذا الاستثناء Exception ، لماذا لا أقوم بكتابة معالج الاستثناء Exception Handler فقط فى الدالة method المهتمة بهذا الاستثناء Exception وهذا ما تقدمه لغة Java كالتالى:

```
method 1 {
    try {
        call method2; }
    catch (exception)
    {
method 2 throws exception {
        call method3; }
method 3 throws exception }
        call read-file;
    }
```

تجميع أنواع الأخطاء والتمييز بينها:

إن الإستثناءات Exceptions تنقسم إلى مجموعات ، بحيث تكون كل مجموعة تختص بمجموعة من الأخطاء ، ولأن كل شئ فى لغة Java هى عبارة عن فصائل Classes ، فإن الإستثناءات Exceptions فى لغة Java هى فصائل Classes ، وكل فصيلة Class تختص بنوع من الاستثناءات Exceptions ، وجميع هذه الفصائل classes تشتق من الفصيلة Throwable ، ونستطيع رسم مخطط لهذه الفصائل classes كالآتى:



- من هذا المخطط يتضح أن هناك فصيلة class يشتق منها جميع الفصائل (التى تمثل الاستثناءات Exceptions) وهى الفصيلة Throwable ، وترث منها فصيلتان subclasses هما Error و Exception.
- الفصيلة Error تمثل الاستثناءات Exceptions الناتجة عن أخطاء بسبب الآلة الافتراضية Virtual Machine للغة Java ، وهذه الأخطاء نادرة وقاتلة ولا يوجد ما تستطيع عمله إزاء هذه الأخطاء.
- أما الفصيلة Exception فهى الأهم حيث تهتم بالاستثناءات Exceptions التي تحدث من البرنامج الذي نكتبه.
- وكما تري من المخطط السابق أن الاستثناءات Exceptions عبارة عن فصائل classes ، وكل فصيلة class تهتم بنوع ما من الاستثناءات Exceptions ، وتستطيع الرجوع إلى توثيق Documentation لغة Java لمعرفة المزيد.
- بعد أن تعرفنا بشيء من التفصيل على الاستثناءات Exceptions ، تعال معى نستوضح المزيد من خلال الأمثلة العملية.

أدوات معالجة الاستثناءات Exceptions Handling فى لغة Java:

- إن معالجة الاستثناءات Exceptions Handling تتم من خلال خمسة كلمات محجوزة Reserved Keywords وهى الكلمات:

finally, try , catch , throw, throws

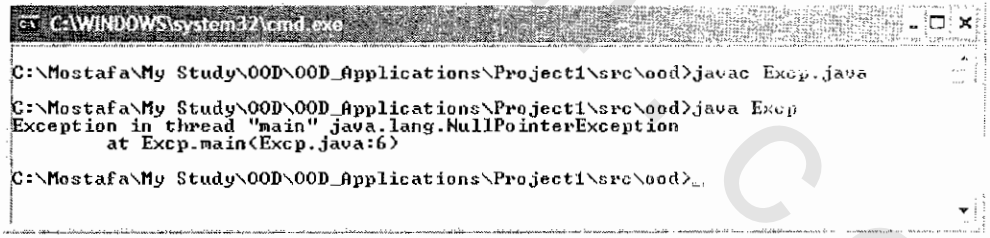
فتعال معى نرى كيفية استخدامها.

التركيب try ... catch:

أول أدوات معالجة الاستثناءات Exceptions Handling هو التركيب try … catch ولكى نرى كيفية عملهما ، فلنلق نظرة على نص البرنامج التالى.

```
1. class Excp
2. {
3. public static void main (String Arg [ ])
4. {
5. String s = null;
6. s.trim();
7. System.out.println("After Exception");
8. }
9. }
```

لو حاولت ترجمة compile ثم تنفيذ هذا المثال ، فستجد أن المترجم compiler قد أظهر الرسالة التالية:



(الشكل 12-3)

من الواضح أن البرنامج قد ترجم Compile بنجاح ولكن عند التنفيذ أظهر هذه الرسالة ، وهذه الرسالة تخبرنا بالمعلومات التالية.

هناك خطأ ظهر أثناء وقت التنفيذ ، ولاكتشاف الخطأ نقرأ الرسالة ، فنجد أنه حدث استثناء Exception ، وهذه أول كلمة فى الرسالة ، لكن أين حدث هذا الاستثناء Exception؟ نجد أن الرسالة تقول أنه فى "thread "main ، أى أنه حدث خطأ عند تنفيذ الدالة الرئيسية ()main.

- ولكن ما نوع هذا الاستثناء Exception ، نجد أنه من نوع NullPointerException ، وقد نتج عن محاولتنا استدعاء دالة method علي هدف object لا يزال بالقيمة null ، وهذا ناتج عن الجملة رقم 6 ، ونجد أن الرسالة تدلنا على اسم الفصيلة class التى حدث بها الاستثناء Exception ورقم السطر الذى حدث به الاستثناء Exception كما هو واضح بين الأقواس.
- من كل ما سبق نستخلص أن الدالة الرئيسية main() حدث بها استثناء Exception ، وقامت بإلقاء الاستثناء throw Exception إلى نظام وقت التنفيذ Runtime System ، وقام نظام وقت التنفيذ Runtime System بالبحث عن من يعالج هذا الاستثناء Exception فلم يجد فألقى الاستثناء Exception ثم خرج من النظام ، وستجد أيضاً أنه لم يصل إلى السطر رقم 7 لأنه توقف عند الاستثناء Exception الحادث.
- ولمعالجة هذا الاستثناء Exception تعال معي ننظر إلى نسخة معدلة من البرنامج فى الجزء التالى:

```
1.   public class Excp
2.   {
3.    public static void main (String [] arg)
4.    {
5.        try
6.        {
7.            String s = null;
8.            s.trim();
9.        }
10.       catch (NullPointerException  e)
11.       {
12.           System.out.println("null object");
13.           System.out.println(e.getMessage());
14.       }
15.       System.out.println("After Exception");
16.   }
17.  }
```

- حاول تنفيذ البرنامج الآن وستجد أن عملية التنفيذ تمت بدون مشاكل وستجد أن السطر رقم 15 قد تم تنفيذه فماذا حدث؟
- نجد أن الذى حدث هو أن الدالة method أخبرت المترجم compiler بأن يحاول تنفيذ الكود فى المنشأ try Block ، وإذا حدث استثناء Exception - وهو ما حدث - يتم التقاطه catch عن طريق المنشأ catch Block وتم إظهار الرسالة التى تفيد حدوث NullPointerException.
- وأخيراً تم ظهور نتيجة السطر رقم 15 لأن الاستثناء Exception الحادث قد تم التقاطه catch ومعالجته Handle.

الصورة العامة لـ try ... catch :

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

تعدد الجملة catch:

- فى بعض الأحيان قد ينتج عن كود ما أكثر من استثناء Exception ، فما الحل فى هذه الحالة؟
- الحل أن يوجد العديد من معالجات الاستثناء Exception Handlers ، وبالتالى لا بد من وجود أكثر من بلوك catch ، ويصبح تركيب البرنامج كما فى السطور التالية:

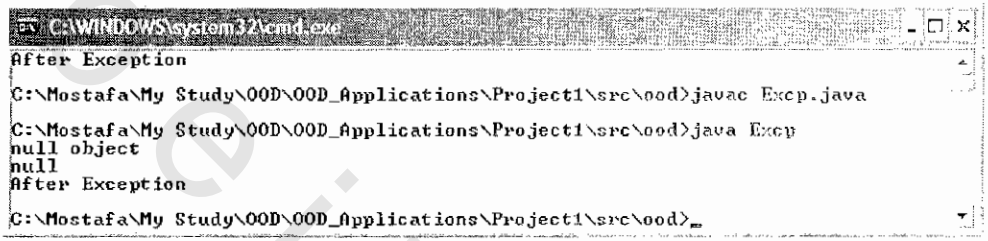
```
try {
// Code that might generate exceptions
} catch(Type1 id1) {
// Handle exceptions of Type1
} catch(Type2 id2) {
// Handle exceptions of Type2
} catch(Type3 id3) {
// Handle exceptions of Type3
```

ولتوضيح ذلك ، تابع معنا سطور البرنامج التالية :

```
1.  public class Excp
2.  {
3.   public static void main (String [] arg)
4.   {
5.       try
6.       {
7.           String s = null;
8.           s.trim();
9.           int c[] = {1};
10.          c[40] = 2;
11.      }
12.      catch (NullPointerException e)
13.      {
14.          System.out.println("null object");
15.          System.out.println(e.getMessage());
16.      }
17.      catch (ArrayIndexOutOfBoundsException  e)
18.      {
19.          System.out.println("array error");
20.          System.out.println(e.getMessage());
21.      }
22.      System.out.println("After Exception");
23.  }
24. }
```

- فى هذا المثال ينتج استثناءان Exceptions.
- الأول ناتج عن استدعاء دالة method علي هدف object بقيمة null وذلك كما فى السطر رقم 8.
- والثانى ناتج عن محاولة تخطى سعة المصفوفة c ، فقد قمنا بالإعلان أن سعتها عنصر واحد وذلك كما فى السطر رقم 9 ثم فى السطر رقم 10 نحاول تخصيص قيمة للعنصر 41 (تذكر أن أول عنصر له الترتيب صفر) بينما المصفوفة array سعتها عنصر واحد فقط.
- لذلك قمنا بإعداد اثنين من معالجى الاستثناء Exception Handler وذلك باستخدام اثنين من جمل catch.

- المنشأ Block الأول يعالج الاستثناء Exception من نوع NullPointerException وذلك كما يتضح في السطور من 12 إلى 16.
- المنشأ Block الثاني يعالج الاستثناء Exception الناتج عن محاولة تخطى سعة المصفوفة c وذلك كما يتضح في السطور من 17 إلي 21.
- ويظهر ذلك من نتيجة التنفيذ كما في الشكل التالي.



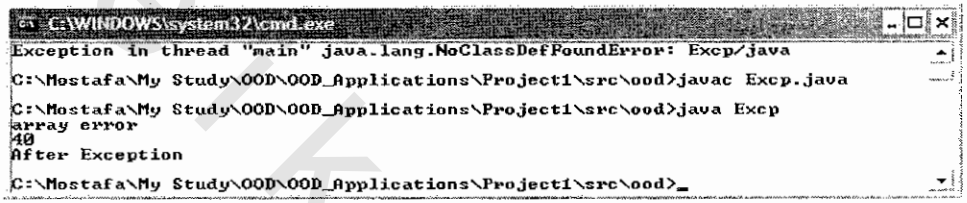
(الشكل 12-4)

- لاحظ أن أول استثناء Exception قابل نظام وقت التنفيذ Runtime System هو الذي تمت معالجته ولم يتم تنفيذ باقي الكود داخل جملة try.
- يمكنك تغيير ترتيب الكود كما في السطور التالية.

```
1.    public class Excp
2.    {
3.     public static void main (String [] arg)
4.     {
5.          try
6.          {
7.               int c[] = {1};
8.               c[40] = 2;
9.               String s = null;
10.              s.trim();
11.         }
12.         catch (NullPointerException e)
13.         {
14.              System.out.println("null object");
15.              System.out.println(e.getMessage());
16.         }
17.         catch (ArrayIndexOutOfBoundsException  e)
```

```
18.         {
19.             System.out.println("array error");
20.             System.out.println(e.getMessage());
21.         }
22.         System.out.println("After Exception");
23.     }
24. }
```

- قمنا فقط بتبديل مكان السطرين 7 و 8 مكان السطرين 9 و 10.
- تأكد أن نتيجة التنفيذ ظهرت كما في الشكل التالي.



(الشكل 12-5)

معالجة الاستثناءات Exceptions Handling عن طريق الكلمة المحجوزة throws:

- في بعض الأحيان عندما نقوم بتصميم دالة Method في فصيلة class ما ، ونعرف أن هذه الدالة method ستلقى باستثناء Exception ، فمن المستحب أن تجعل مستخدم الدالة method هو الذى يقوم بمعالجة الاستثناء Exception Handling ، وللتوضيح أكثر تعال معى نرى نص البرنامج التالى.

```
1.class ThrowsDemo {
2.static void delta() throws IlegallAccessException
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.    }
```

- في هذا المثال قمنا بتصميم الدالة delta() ، وهذه الدالة method نعرف أنها ستلقى باستثناء Exception ، وفي هذه الحالة لا بد من تحديد الاستثناء Exception الذى ستلقيه الدالة method حتى تستطيع الدوال methods الأخرى حماية نفسها بتوفير معالج استثناء Exception Handler من نوع الاستثناء Exception الذى ستلقيه الدالة delta() عند استدعائها.
- والدالة delta() تلقى الاستثناء Exception بالكلمة المحجوزة throws ويليها نوع الاستثناء Exception وذلك كما فى السطر رقم 2.
- والشكل العام لمثل هذه الدوال methods يكون كالآتى:

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

- ومن هذه الصورة العامة يتضح أننا نستطيع إرسال أكثر من استثناء Exception.
- بعد السطر رقم 2 نجد جملة بناء الدالة delta() ، وما يهمنا هنا هو السطر رقم 5 ، وهذا السطر هو المسئول عن عملية إرسال الاستثناء Exception التي تتم من خلال الأمر throw وليس throws وسنقوم لاحقاً بشرح المزيد عن الأمر throw.
- لاحظ أيضاً أن الدالة method انتهت عند السطر رقم 6 ولم تهتم بمعالجة الاستثناء Exception Handling لأن هذا أصبح مهمة من يستدعى هذه الدالة method وذلك كما فى السطور من 9 إلي 15.
- ففى السطور من 9 إلي 15 قامت الدالة main() باستدعاء الدالة delta() ، ولأن الدالة main() تعرف أن الدالة delta() ستلقى باستثناء throw Exception ، فقد قامت بحماية نفسها عند استدعاء الدالة delta() بجملة try ... catch ، والتقطت الاستثناء Exception وعالجته وذلك كما فى السطور من 12 –15.

معالجة الاستثناءات عن طريق الكلمة المحجوزة throw:

- يتم استخدام الكلمة المحجوزة throw فى إلقاء استثناء Exception كما أوضحنا فى المثال السابق ، ولكنه ليس بالضرورة أن يستخدم مع throws ، فيمكن أن يستخدم بصفة مستقلة 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 تستخدم لإرسال الاستثناء Exception بطريقة إجبارية.
- ولكى نستخدم throw لا بد من إنشاء هدف object من نوع الاستثناء Exception الذى نريد أن نرسله ، وذلك كما فى السطر رقم 5 ، ثم لا بد من وجود معالج للاستثناء Exception Handler وذلك كما يوضحه السطر رقم 6 وذلك إذا أردت أن تعالج الاستثناء Exception داخل الدالة method.
- أما إذا أردت إرسال استثناء Exception لتعالجه دالة method أخرى ، فعندما نستدعى الدالة method المعلن فيها الاستثناء Exception ، فلا بد من إرسال

هدف object من نوع الاستثناء Exception ، وذلك كما يوضحه السطر رقم 8 ، ويتم استقبال هذا الهدف object في معالج الاستثناء Exception Handler الموجود فى دالة method الاستدعاء كما فى السطر رقم 15.

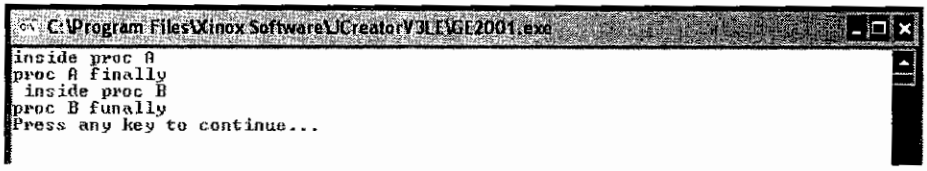
معالجة الاستثناءات عن طريق الكلمة المحجوزة finally:

- عندما يتم إرسال استثناء Exception ، فإن تسلسل flow التنفيذ فى الدالة Method يتم تغييره ويتم تحويل التسلسل للبحث عن أقرب معالج للاستثناء Exception Handler ، وفى حالة عدم وجود معالج للاستثناء Exception Handler مثل catch ، فلا يتم تنفيذ الجمل الموجودة بعد السطر الحادث به الاستثناء Exception.
- إذن ماذا نفعل فى حالة إذا أردنا أن يتم تنفيذ جمل أو قطعة من الكود بغض النظر عن ما إذا كان الاستثناء Exception قد تم معالجته أم لا؟
- فى هذه الحالة يبرز لنا استخدام الكلمة المحجوزة finally ، حيث أنه يتم تنفيذ الجمل التى بداخل جملة finally بغض النظر عن حدوث أو عدم حدوث استثناء Exception ، فحتى لو لم يكن هناك معالج استثناء Exception Handler مثل catch لالتقاط الاستثناء catch Exception ومعالجته ، فسيتم تنفيذ الكود الموجود داخل جملة finally ، ويتم تنفيذ الكود الموجود داخل جملة finally مباشرة بعد جملة try ، 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.  }
```

- في هذا المثال ، نجد أنه بالرغم من إلقاء الدالة procA() باستثناء Exception كما يوضحه السطر رقم 5 ، إلا أنه لا يوجد معالج للاستثناء Exception Handler ، فلا يوجد لدينا جملة catch ، فكان من الطبيعي أن يتوقف عمل البرنامج ، ولكن بسبب وجود جملة finally ، فإن البرنامج يكمل التسلسل ويتم تنفيذ البرنامج بدون مشاكل وذلك كما في السطور من 7 إلي 10.
- ثم نجد شيئاً أخر غريباً وهو الدالة procB() وخصوصاً السطر رقم 14 والموجود به جملة return ، فوجود تلك الجملة كان يعني أن يتم كسر التسلسل والخروج ، ولكن لأن الدالة method بها جملة finally ، فقد تم إجبار البرنامج على تكملة التسلسل ، أما بقية البرنامج فنجد أن سير الدالة method طبيعي جداً ، حيث يتم استدعاء الدالة procA() وذلك كما في السطر 23 ، ولأننا نعرف أن الدالة method تقوم بإرسال استثناء Exception فقد حمينا أنفسنا منه بجملة try ... catch.
- وفي السطر رقم 26 تم استدعاء الدالة procB() ، وحيث أن الدالة method لا تقوم بإرسال استثناء Exception ، فقد تم استدعاؤها بالطريقة المعتادة.
- وعند التنفيذ تحصل على نتيجة التنفيذ كما في الشكل (12-6).



(الشكل 12-6)

ويصبح تركيب البرنامج كما فى السطور التالية.

```
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 Exceptions التى استخدمناها هى من الفصائل classes الموجودة فى اللغة ، وهذه الفصائل classes تغطى تقريباً كافة الاستثناءات Exceptions الممكنة الحدوث ، ولكن ماذا نفعل إذا احتجنا استثناء Exception غير موجود فى اللغة ، أى إذا كان الخطأ الذى يظهر غير معروف باللغة؟
- الحل هو إنشاء (تعريف) هذا الاستثناء Exception ، فكيف يتم ذلك؟
- بما أن الاستثناءات Exceptions فى لغة Java هى فصائل Classes ، إذن فلنا الحق فى أن ننشىء فصائل classes ترث الفصائل classes الموجودة فى لغة Java ونضيف إليها الجديد بما يناسب منطق البرنامج.
- والفصيلة class التى نقوم ببنائها لا تفعل أكثر من إصدار رسائل توضح أن هناك استثناء Exception ما قد حدث وتبين نوعه وهذا للتوضيح فقط ، وفى الواقع يتم إنشاء فصائل classes لعلاج أخطاء حقيقية بالبرنامج ، ويتضح ذلك من المثال التالي.

```
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.}
```

شرح السطور:

- في هذا المثال افترضنا أننا نريد أن ننشئ استثناء Exception غير موجود.
- قمنا بإنشاء فصيلة class ترث من الفصيلة Exception وذلك كما في السطر رقم 1.

- نريد تصميم الاستثناء Exception بحيث يحدث لو تم استدعاء دالة method ما وتم تمرير معامل Parameter لها بقيمة أكبر من 15 ، ونريد أن تظهر رسالة توضح قيمة الرقم ، ولذلك أنشأنا دالة البناء Constructor تأخذ معاملاً Parameter كما هو واضح في السطر رقم 3.
- ثم قمنا بعملية نسخ override للدالة ()toString والتى مهمتها إظهار وصف الاستثناء Exception الذى حدث.
- ثم قمنا بإنشاء الفصيلة UsingExceptionDemo وقمنا بإنشاء الدالة ()compute والتى تلقى باستثناء Exception من النوع My Exception وذلك كما فى السطر رقم 14.
- ثم حددنا شرط لحدوث الاستثناء Exception وهو أنه إذا كانت قيمة المعامل Parameter أكبر من 15 -وذلك كما فى السطر رقم 17- فيتم إلقاء الاستثناءthrow Exception ، أما إذا كانت القيمة أقل فلا يتم إلقاء الاستثناء throw Exception.
- ثم قمنا باستدعاء الدالة ()compute مرتين كما فى السطرين 24 و 25 مرة بقيمة أقل من 15 فلا يحدث استثناء Exception ، ومرة أخري بقيمة أكبر من 15 فيحدث استثناء Exception وبالتالي تجد أن نتيجة تنفيذ البرنامج كالآتى.

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

ملخص الفصل:

- تعرضنا في هذا الفصل لشرح الاستثناءات Exceptions وطرق معالجتها.
- في الفصل القادم سوف نتعرف - بإذن الله - علي كيفية تنفيذ عملية إدخال وإخراج البيانات من خلال عملية التدفق Streams ، فتابع معنا الفصل القادم.