Exceptions...a simple introduction

Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

Exceptions

Standardized by ANSI in 1996
Exception is the root of the exception hierarchy:
84 predefined exceptions. The two most important
classes are:
   - Error
   - Notification
Specialised into predefined exceptions -> subclass
them to create your own exceptions
Some methods of Exception:
   - defaultAction is executed when an exception occurs

Catching an Exception

\[ x \ y \]
x := 7.
y := 0.
\[ x/\ y \]
on: ZeroDivide
do: [:exception| Transcript show:
   exception description, cr.
   0....]

Signaling an Exception

Error signal
Warning signal: 'description of the exception'
Exception

An Exception Handler is defined using `on:do:` and is composed of an exception class (ZeroDivide) and a handler block `[theException] Transcript show: ' division by zero']

An Exception Handler completes by returning the value of the handler block in place of the value of the protected block (here [x/y]). We can exit the current method by putting an explicit return inside the handler block.

The Main Exceptions of VW

<table>
<thead>
<tr>
<th>Exception class</th>
<th>Exceptional Event</th>
<th>Default Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Any program error</td>
<td>Open a Notifier</td>
</tr>
<tr>
<td>ArithmeticError</td>
<td>Any error evaluating an arithmetic</td>
<td>Inherited from Error</td>
</tr>
<tr>
<td>MessageNotUnderstood</td>
<td>Any unusual event that does not impair continued execution of the program</td>
<td>Inherited from Error</td>
</tr>
<tr>
<td>Notification</td>
<td>Notification</td>
<td>Do nothing continuing executing</td>
</tr>
<tr>
<td>Warning</td>
<td>An unusual event that the user should be informed about</td>
<td>Display Yes/No dialog and return a boolean value to the signaler</td>
</tr>
</tbody>
</table>

Exception Sets

Exception Sets
- `[do some work]
  on: ZeroDivide, Warning
do: [:ex| what you want]

Or
  |exceptionSets|
  exceptionSets := ExceptionSet with: ZeroDivide
  with: Warning.
- `[do some work]
  on: exceptionSets
do: [:ex| what you want]

Exception Environment

Each process has its own exception environment: an ordered list of active handlers.

Process starts -> list empty

[aaaa] on: Error do: [bbb] -> Error; bbb added to the beginning of the list

When an exception is signaled, the system sends a message to the first handler of the exception handler.

If the handler cannot handle the exception, the next one is asked

If no handler can handle the exception then the default action is performed
A handler block completes by executing the last statement of the block. The value of the last statement is then the value returned by the handler block. Where this value should be returned depends:

Nonresumable (Error)

```Scheme
Sq: ([Error signal. 'Value from protected block']
  on: Error
  do: [ex|ex return: 'Value from handler'])
```

> 'Value from handler'

Resumable (Warning, Notification)

In this case Notification signal raises an exception, then the context is restored and the value returned normally

```Scheme
(Notification signal. 'Value from protected block'
  on: Notification
  do: [ex|ex resume: 'Value from handler'])
```

Resume:/Return:

Transcript show:

```Scheme
[Notification raiseSignal. 'Value from protected block'
  on: Notification
  do: [ex| Transcript show: 'Entering handler '.
       'Value from handler': '5']

-> Entering handler 5
```

Resume:/Return:

Transcript show:

```Scheme
[Notification raiseSignal. 'Value from protected block'
  on: Notification
  do: [ex| Transcript show: 'Entering handler '.
       ex resume: 'Value from handler'. '5']

> Entering handler Value from protected block
```

Transcript show:

```Scheme
[Notification raiseSignal. 'Value from protected block'
  on: Notification
  do: [ex| Transcript show: 'Entering handler '.
       Transcript show: 'Value from handler'. '5']
```

> Entering handler
**Exiting Handlers Explicitly**

- **exit or exit:** (VW specific) Resumes on a resumable and
  returns on a nonresumable exception
- **resume or resume:** Attempts to continue processing the
  protected block, immediately following the message that
  triggered the exception.
- **return or return:** ends processing the protected block
  that triggered the exception
- **retry** re-evaluates the protected block
- **retryUsing:** evaluates a new block in place of the
  protected block

**Examples**

Look in Exception class examples categories

-2.0 to: 2.0 do: [:i | [ 10.0 / i. Transcript cr; show: i printString ]
on: Number divisionByZeroSignal do:
  [:ex | Transcript cr; show: 'divideByZero abort'. ex return ]
]

-2.0
-1.0
divideByZero abort
1.0

**Exiting Handlers Explicitly (ii)**

- **resignalAs:** resignal the exception as another on
  pass exit the current handler and pass to the next outer
  handler, control does not return to the passer
  outer as with pass, except will regain control if the outer
  handler resumes
  exit, resume and return: return their argument as the
  return value, instead of the value of the final statement of
  the handler block

**Examples**

retry recreates the exception environment of active handlers

[ x /y]
on: ZeroDivide
do: [:ex | y := 0.00001. exception retry]