RMod

Exceptions...a simple introduction

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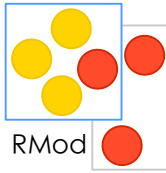
Exceptions

Standardized by ANSI in 1996
Exception is the root of the exception hierarchy:
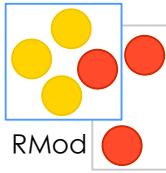84 predefined exceptions. The two most important classes are:
- Error
- Notification

Specialised into predefined exceptions -> subclass them to create your own exceptions
Some methods of Exception:
- defaultAction is executed when an exception occurs

S.Ducasse

Catching an Exception

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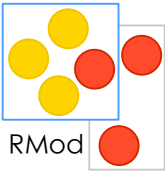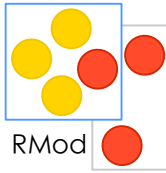
S.Ducasse

Signaling an Exception

Error **signal**

Warning **signal:** 'description of the exception'

S.Ducasse

Exception

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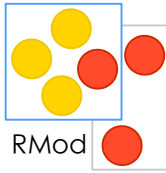a *handler block* [:theException| Transcript show:' division by zero']

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

S.Ducasse

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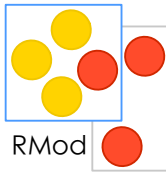
Exception class	Exceptional Event	Default Action
Error	Any program error	Open a Notifier
ArithmeticError	Any error evaluating an arithmetic	Inherited from Error
MessageNotUnderstood	Any unusual event that does not impair continued execution of the program	Inherited from Error
Notification	Notification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

S.Ducasse

Tuesday 24 March 15

Exception Sets

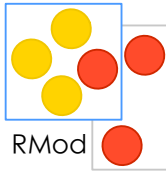Exception Sets

```
[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]
```

S.Ducasse

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

S.Ducasse

Tuesday 24 March 15

Resumable and Non-Resumable (i)
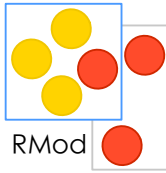
A handler block completes by executing the last statement of the block. The value of the last statement is then the value returned by the handler block.
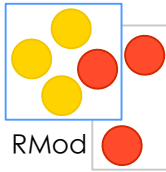Where this value should be returned depends:
Nonresumable (Error)

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

 > 'Value from handler'

S.Ducasse

Resumable and Non-Resumable (ii)

Resumable (Warning, Notification)

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

> [Notification raiseSignal. 'Value from protected block']
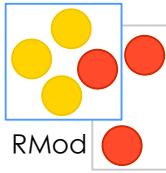>   on: Notification
>   do: [:ex|ex *resume:* 'Value from handler']

> [Notification signal. 'Value from protected block']
>   on: Notification
>   do: [:ex|ex *resume:* 'Value from handler']

S.Ducasse

Tuesday 24 March 15

Resume:/Return:
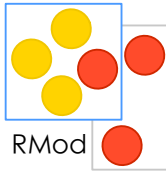
Transcript show:

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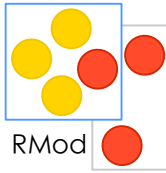> Entering handler Value from protected block

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

S.Ducasse

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, immeditely following the message that triggered the exception.

return or **return:** ends processing the protected block that triggered the exception

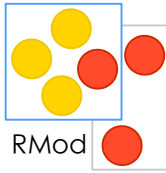**retry** re-evaluates the protected block
retryUsing: evaluates a new block in place of the protected block

Exiting Handlers Explicitly (ii)
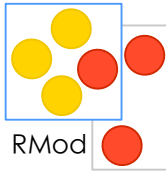
resignalAs: resignal the exception as another on

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

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 ]
                ]
```
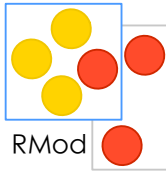
-2.0
   -1.0
   divideByZero abort
1.0

S.Ducasse

Examples

retry recreates the exception environment of active handlers

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