CS 535 Object-Oriented Programming & Design
Spring Semester, 2003
Doc 9 Exceptions

References

Object-Oriented Design with Smalltalk — a Pure Object Language and its Environment, Ducasse, University of Bern, Lecture notes 2000/2001, http://www.iam.unibe.ch/~ducasse/WebPages/Smalltalk/ST00_01.pdf

VisualWorks Application Developer Guide, doc/vwadg.pdf in the VisualWorks installation. Chapter 10 Exception and Error Handling

Reading

VisualWorks Application Developer Guide, doc/vwadg.pdf in the VisualWorks installation. Chapter 10 Exception and Error Handling

Exceptions
Basic Handling of Exceptions

Format

```
[ProtectedBlock]
   on: ExceptionList
   do: [:exception | HandlerBlock]
```

Example

```
[numerator := 5.
denominator := 0.0.
numerator / denominator]
   on: ZeroDivide
   do:
      [:exception |
      Transcript
         show: exception description;
         cr]
```

Unlike Java, in Smalltalk zero divide by both integer and floats cause a zero divide exception to be raised

Exceptions are Classes

Exception class is the parent of all exceptions

Subclasses define specialized exceptions

Important Subclasses

- Error
  A condition that prevents the normal continuation of processing

- Notification
  Something interesting has occurred
  If it is not handled, it will pass by without effect

- Warning
  An unusual event the user needs to know about
  Asks the user if the program should continue

- MessageNotUnderstood
  A method was sent to an object that does not implement it

Important exception methods

- description
  Returns a string describing the actual exception

- defaultAction
  Executed when an exception occurs

Raising Exceptions
Implicitly Raised Exceptions

Exceptions can be raised by VM

12 / 0

Explicitly Raised Exceptions

Send one of following messages to an exception class

raiseSignal: aStringDescriptionOfProblem
raiseSignal

Examples

Warning raiseSignal: 'This string is the signal description' false

Error raiseSignal

self error: 'A message'

Object defines a method error: that raises an exception

```
Object>>error: aStringOrMessage
   | lastNonSpace aString|
   aString :=  aStringOrMessage asString.
   lastNonSpace := aString findLast: [:ch | ch ~= Character space].
   ^self errorSignal raiseErrorString:
         (aString copyFrom: 1 to: lastNonSpace)

Object>>errorSignal
   "Answer the Signal used for miscellaneous errors
   (self error:)."
   ^self class errorSignal

Object class>>errorSignal
   "Answer the Signal used for miscellaneous errors (self error:)."

   ^Error
```

Default exception raised is Error

To change the exception raised by error: override the class method errorSignal

```
Foo class>>errorSignal
   ^KeyNotFoundError
```

Template Method

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses

Subclasses redefine certain steps of an algorithm without changing the algorithm's structure

Important in class libraries

Inverted control structure

- Parent class calls subclass methods

Examples

printString and printOn:

Enumeration Template Method Example

Standard collection iterators
collect:, detect:, do:, inject:into:, reject:, select:

```
Collection>>collect: aBlock
   | newCollection |
   newCollection := self species new.
   self do: [:each | newCollection add: (aBlock value: each)].
   ^newCollection

Collection>>do: aBlock
   self subclassResponsibility

Collection>>inject: thisValue into: binaryBlock
   | nextValue |
   nextValue := thisValue.
   self do: [:each | nextValue := binaryBlock value: nextValue value: each].
   ^nextValue

Collection>>reject: aBlock
   ^self select: [:element | (aBlock value: element) == false]

Collection>>select: aBlock
   | newCollection |
   newCollection := self species new.
   self do: [:each | (aBlock value: each) ifTrue: [newCollection add: each]].
   ^newCollection
```

Subclasses only have to implement:
species, do:, add:

Species

```
Object>>species
   "Answer the preferred class for reconstructing the receiver.  For example,
   collections create new collections whenever enumeration messages such as
   collect: or select: are invoked.  The new kind of collection is determined
by
   the species of the original collection.  Species and class are not always the
   same.  For example, the species of Interval is Array."

   ^self class
```

Exceptions & Return Values

on:do: is a message, so returns a value

If an exception is raised the return value is:
The return value of the handler

If an exception is not raised the return value is:
The return value of the protected block

Example - No Exception Raised

This code assigns 10 to result

```
| result |
result := [10/1]
   on: ZeroDivide
   do: [:exception | Float zero ].
^result
```

Example - Exception Raised

This code assigns 0.0 to result

```
| result |
result := [10/0]
   on: ZeroDivide
   do: [:exception | Float zero ].
^result
```

Catching Multiple Exceptions

Use a comma or ExceptionSets

```
[1/0]
   on: Warning , ZeroDivide
   do: [:exception | code here]


| exceptions |
exceptions := ExceptionSet with: Warning with: ZeroDivide.
[1/0]
   on: exceptions
   do: [:exception | code here]
```

Inheritance and Exception

All subexceptions are caught by an exception in on:do:

ZeroDivide is a subclass of Error

The ZeroDivide exception will be caught in the following

```
[1/0]
  on: Error
  do:
    [:exception |
    Transcript
      show: exception description;
      cr]
```

Finding the Exception Handler

When an exception is raised

The enclosing handlers are searched

- Start with the code that raised the exception
- Search the "closest" enclosing handler first
- Continue searching the enclosing handers

The first handler that deals with the exception is used

If no handlers handle the exception the exception's default action is done

```
[[1/0]
  on: ZeroDivide
  do: [:exception | Transcript show: 'First']]
    on: ZeroDivide
    do: [:exception | Transcript show: 'Second']
```

Result in Transcript

First

Warning Default Action

Warning default action

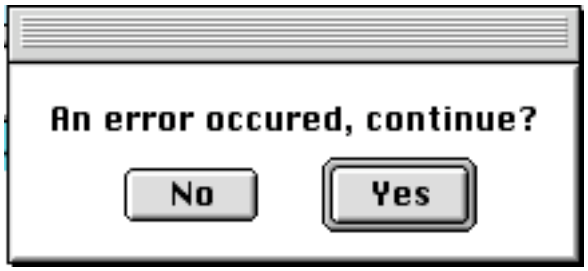
Show dialog asking user if program should continue

Example 1

The following code:
Displays a dialog asking if user wishes to continue
Returns boolean result depending on users answer

Warning raiseSignal: 'An error occured, continue?'.

Dialog Displayed



Example 2

The following code:
Prints 'Handler' in the Transcript

```
[Warning raiseSignal: 'Hi Mom'.
Transcript show: 'End']
   on: Warning
   do: [:exception | Transcript show: 'Handler']
```

Result in Transcript

Handler

Notification Default Action

Notification default action

Do nothing & continue as normal

Example 1

The following code:

Prints 'End' in the Transcript

```
Notification raiseSignal: 'Hi Mom'.
Transcript show: 'End'
```

Example 2

The following code:

Prints 'Handler' in the Transcript

```
[Notification raiseSignal: 'Hi Mom'.
Transcript show: 'End']
    on: Exception
    do: [:exception | Transcript show: 'Handler']
```

This example shows why you may not want to wrap your code in a general catch all handler

What is the Default Action for Exception X?

Look at the defaultAction method in the exception's class

Resumable Exceptions

Some exceptions are resumable

```
| result |
[result := 10/0.
Transcript show: result printString]
	on: ZeroDivide
	do:
		[:exception |
		exception resume: Float zero ]
```

Output in Transcript

0.0

Exception Messages that exit the Handler

resume or resume:
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
Reevaluates the protected block

retryUsing:
Evaluates a new block in place of the protected block

resignalAs:
Resignal the exception as another exception

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

resume: and return: return their argument as the return value, instead of the value of the final statement of the handler block

Example - resume:

10/0 raises an exception
The handler requests resumption with value 1
The expression 10/0 returns 1
The sum becomes 1 + 5

```
| result |
[result := 10/0 + 5.
Transcript show: result printString]
   on: ZeroDivide
   do:
      [:exception |
      exception resume: 1 ]
```

Output in Transcript

6

Example - resume

10/0 raises an exception
The handler requests resumption, no value
The expression 10/0 returns nil

```
| result |
[result := 10/0.
Transcript show: result printString]
   on: ZeroDivide
   do:
      [:exception |
      exception resume ]
```

Output in Transcript

nil

Example - retry

x/y raises an exception
The handler sets y := 1.
The block is reexecuted

```
| x y result |
x := 10.
y := 0.
[result := x / y.
Transcript show: result printString]
   on: ZeroDivide
   do:
      [:exception |
      y := 1.
      exception retry ]
```

Output in Transcript

10

Example - return

The following are equivalent

```
| result |
[result := 10/0.
Transcript show: result printString]
   on: ZeroDivide
   do: [:exception | exception return]

| result |
[result := 10/0.
Transcript show: result printString]
   on: ZeroDivide
   do: [:exception | nil]
```

Result

No output in Transcript

Example - return:

The following are equivalent

```
| result |
result := [10/0]
   on: Error
   do: [:exception | Float zero ].
^result
```

```
| result |
result := [10/0]
   on: Error
   do: [:exception | exception return: Float zero ].
^result
```

Some Error Handling Messages in Object

self error: 'Error message'
  Raises Error exception with given message

self halt
self halt: 'Message"
  Raises Halt exception.
  Allows user to invoke debugger or resume

self notify: 'Notify message"
  Like halt except user does not see stack history

self shouldNotImplement
  Used in subclasses in inherited methods that do not belong in the subclass

self subclassResponsibility
  Used in methods to declare them abstract
  Indicated subclasses must implement this method

Clean Up or Unwind Protection

```
mySafeMethod
   | grades |
   grades := 'cs535Grades' asFilename.
   gradeIO := grades readWriteStream.
   Bar yourUnsafeMethod.
   "Change some grades here - code not shown"
   gradeIO close.
```

If yourUnsafeMethod raises an exception gradeIO is not closed

If you do not know what exception might be raised it is hard to handle it

Use ensure: or ifCurtailed:

ensure:

Format

```
[block] ensure: [clean up block]
```

Ensure that the clean up block will be done

If block ends due to an exception

Execute handler for exception

Execute clean up block

You code should not depend on the order of execution of the handler and clean up block

Example

```
[[10/0] ensure: [Transcript show: 'In ensure'; cr]]
   on: ZeroDivide
   do: [:exception | Transcript show: 'In handler';cr ]
```

Output in Transcript

In handler
In ensure

ifCurtailed:

Format

[block] ifCurtailed: [clean up block]

Clean up block is done only if [block] ends abnormally

Translating Exceptions

At times you may need to rethrow an exception, but as a different exception

```
[low-level I/O]
   on: OperatingSystemException
   do:
      [ex|
      ex errorCode = -213
         ifTrue: [ex resignalAs: EndOfFile new]
         ifFalse: [ex resignalAs:
            (Error new messageText: 'OS Error']]
```

Creating Your Own Exceptions

Subclass the correct existing Exception
  Usually Error or Notification

If you want the exception to be resumable

  Make method isResumable return true

If you want non-standard default behavior

  Override the method defaultAction