Exceptions

When the Contract is Broken

Definitions

- A routine call **succeeds** if it terminates its execution in a state satisfying its contract

- A routine call **fails** if it terminates its execution in a state not satisfying its contract

- An **exception** is a run-time event that may cause a routine call to fail
  - » **Every failure is caused by an exception but not every exception causes a failure**

Exception Causes

- Try **a.f(...)** and **a** is void
- Calling a routine that fails
- Finding assertions fail
  - » **preconditions, postconditions, class invariants, check**
- Loops fail
  - » **loop invariant goes false, variant does not decrease**
- A hardware problem (divide by 0), or operating system error
- Trigger an exception explicitly

Failures and Exceptions

- A failure of a routine causes an exception in its caller
- Failure cases
  - » **A routine call fails if and only if**
    - > **an exception occurs during the execution of the routine**
    - > **the routine does not recover from the exception**

What Not to Do – C example

- C example
  - » **signal (exception_code , exception_handler)**
    - > **Notify OS that when exception_code occurs, pass control to exception_handler**
- Expected response is
  - » **exception_code occurs**
  - » **exception_handler invoked**
  - » **return to point of exception & continue**
- No guarantee
  - » **return to point of exception**
  - » **problem has been addressed**

What could be done – C example

- What should be done
  - » **correct the situation – perhaps modify initial state to improve it**
    - > **Allow network to choose a route**
  - » **rerun the routine**
- Can do it in C
  - » **Use setjmp to save a restart location**
  - » **Use longjmp to return – even over intervening subprogram calls**
    - > **Pops the runtime stack back to the setjmp location**

What Not to Do – Ada Example

```
sqrt ( n : REAL ) return REAL is
   begin
      if x < 0.0 then raise Negative
      else  normal_computation

   exception when Negative => put ("Negative") return
                  when others => ... return

end
```

- On Negative message printed and return to caller
- Caller not notified of the event
- Is this an appropriate response?

What could be done – Ada Example

- Need to use the raise exception mechanism in the exception handler
- Ada Exception Rule
  - » **The execution of any Ada exception handler should end by either executing a raise instruction or retrying the enclosing program unit**

Exception Handling Principle

- Ignore false alarms
  - » **Exception mechanism used in an event loop**
    - > **Resizing of a window -- better ways to handle it.**
- Only two responses
  - » **Retrying**
    - > **Attempt to change the conditions that led to the exception and execute the routine again from the beginning**
  - » **Failure – Organized panic**
    - > **Clean up the environment (reestablish invariants)**
    - > **Terminate the call**
    - > **Report failure to the caller**

On Retrying

- Best response is routine succeeds on retry
  - » **Caller is unaffected; is not disturbed**
- Sometimes nothing to do but retry as external conditions may have changed
  - » **Busy signal when attempting to phone someone**
- Could change initial conditions – within parameters of invariants
- Could try different algorithm

On Failure

- Make sure the caller is notified
  - » **Give up – panic mode**
- Restore consistent state
  - » **Be organized**
  - » **Change state so invariants are correct**

Rescue & Retry

- The rescue clause is invoked when an exception occurs

```
routine
  require  preconditions
  local  variables
  do body
  ensure postconditions
  rescue                              -- no rescue, routine fails
     if  ....  then ..... retry
     else ....                        -- no retry, routine fails
end
```

Exception History

- If no routine in the call chain is able to succeed when an exception is raised
  - » **System finally gets control**
  - » **Prints history of propagating the exception up to the root**
    - > **List**
      - – **Object, Class, Routine**
      - – **Nature of exception**
        - **– void reference**
        - **assertion failure – use assertion labels**
        - **routine failure**
      - – **Effect**
        - **fail or retry**

Example 1 – Keep Retrying

```
get_integer
  do
    print ("Enter an integer: ")
    read_one_integer
  rescue
     retry
end
```

Example 2 – Maximum retries

```
try_to_get_integer                    // note change from text
  local  attempts : INTEGER
  do
    if attempts < Max_attempts then
        print ("Enter an integer")
        read_one_integer  ;  integer_read := True
    else
        integer_read := False
    end
rescue
     attempts := attempts + 1 ; retry
end
```

Example 2 – Maximum retries – 2

```
get_integer
  do
    try_to_get_integer
    if integer_read then
        n := last_integer
    else
        ... Do next level of interaction ...
    end
end
```

Example 3 – Hardware or OS problem

```
-- Precondition fails but only know after computation

quasi_inverse ( x : REAL ) : REAL  -- 1 / x  if possible

 local  division_tried : BOOLEAN

 do

    if not division_tried then
       Result := 1 / x
    end

  rescue
     division_tried := True
     retry

 end
```

Result = 0 if x is too small and causes underflow

Example 4 – N version Programming

```
do_task                    -- try several algorithms
  local  attempts : INTEGER
  do
    if attempts = 0 then  do_version_1
    elseif attempts = 1 then do_version_2
    elseif attempts = 2 then do_version_3
    end
  rescue
    attempts := attempts + 1
    if  attempts < 3 then reset_state  ; retry
    else restore_invariant
    end
end
```

Correctness of the Rescue Clause

- Formal rule C2 for class correctness stated

  For every exported routine R and any set of valid arguments A_R

C2 **$\{ pre_R (A_R)$ and inv $\}$ $Body_R$ $\{ post_R (A_R)$ and inv$\}$**

- Correctness rule for failure inducing rescue clauses

C3 **$\{$ True $\}$ $Rescue_R$ $\{$ inv$\}$**

- Correctness rule for retry inducing rescue clauses

C4 **$\{$ True $\}$ $Retry_R$ $\{$ pre_R and inv$\}$**

- Precondition for C2 is stronger than C3 & C4, and its postcondition is also stronger.
  - » **C3 does not have to ensure the contract**

When there is no Rescue Clause

- Every routine has the following by default

  rescue default_rescue

  > **default_rescue does nothing but can be overridden**

  > **Creation routines establish the invariant. May be possible to use creation routines in writing a default_rescue**

EXCEPTIONS Class

- Can use the EXCEPTIONS class to give exception objects
  - » **Inherit from EXCEPTIONS and then customize**
  - » **Can know the nature of the last exception**
  - » **Can raise exceptions**

Exception Simplicity Principle

All processing done in a rescue clause should remain simple, and focused on the sole goal of bringing the recipient object back to a stable state, and, if possible, permitting a retry.