Programming in the large

Bertrand Meyer

Lecture 24: Exception handling in object-oriented programming

Causes of exceptions

- Void call (x.f with no object attached to x)
- Operating system signal: arithmetic overflow, no more memory, interrupt...
- Assertion violation (if contracts are being monitored)
Exceptions

An exception is an "abnormal case" occurring program execution, causing a disruption of the default flow of control.

How to use exceptions?

Two opposite styles:

- Exceptions as a control structure: Use an exception to handle all cases other than the most favorable ones (e.g., a key not found in a hash table triggers an exception)
- Exceptions as a technique of last resort

Exception vocabulary

- "Raise", "trigger" or "throw" an exception
- "Handle" or "catch" an exception
Java exceptions

Exceptions are objects, descendants of Throwable:

Java: raising an exception

Instruction:

    throw my_exception

The enclosing routine should be of the form

    my_routine (...) throws my_exception {
        ...
        if abnormal_condition
            throw my_exception;
    }

Java: handling an exception

try {
    instruction_1;
    instruction_2;
    ...
    instruction_n;
} catch (Expected_exception_type e) {
    handling_code
} (Possible "finally" clause to complete both cases)
Exception handling

The need for exceptions arises when a contract is broken.

Two concepts:
- Failure: a routine, or other operation, is unable to fulfill its contract.
- Exception: an undesirable event occurs during the execution of a routine — as a result of the failure of some operation called by the routine.

The original strategy

\[
\begin{align*}
    r(\ldots) & \text{ is } \\
    & \text{ require } \\
    & \text{ ... } \\
    & \text{ do } \\
    & \text{ op } \\
    & \text{ ensure } \\
    & \text{ ... } \\
    & \text{ end }
\end{align*}
\]

Fails, triggering an exception in \( r \) (\( r \) is recipient of exception).

Handling exceptions properly

Safe exception handling principle:

There are only two acceptable ways to react for the recipient of an exception:
- Concede failure, and trigger an exception in the caller (Organized Panic)
- Try again, using a different strategy (or repeating the same strategy) (Retrying)

(Rare third case: false alarm)
How not to do it

(From an Ada textbook)

```ada
sqrt (x: REAL) return REAL is
begin
  if x < 0.0 then
    raise Negative;
  else
    normal_square_root_computation;
  end
exception
  when Negative =>
    put ("Negative argument");
    return;
  when others => ...
end sqrt;
```

The call chain

Exception mechanism

Two constructs:
- A routine may contain a rescue clause.
- A rescue clause may contain a retry instruction.

A rescue clause that does not execute a retry leads to failure of the routine (this is the organized panic case).
### Transmitting over an unreliable line (1)

```plaintext
Max_attempts: INTEGER is 100
attempt_transmission(message: STRING) is
  -- Transmit message in at most Max_attempts attempts.
local failures: INTEGER do
  unsafe_transmit(message)
rescue
  failures := failures + 1
  if failures < Max_attempts then retry
end
```

### Transmitting over an unreliable line (2)

```plaintext
Max_attempts: INTEGER is 100
failed BOOLEAN
attempt_transmission(message: STRING) is
  -- Try to transmit message.
  -- If impossible in at most Max_attempts attempts, set failed to true.
local failures: INTEGER do
  if failures < Max_attempts then unsafe_transmit(message)
    else failed := True
end
rescue
  failures := failures + 1
  retry
```

### If no exception clause (1)

Absence of a rescue clause is equivalent, in first approximation, to an empty rescue clause:

```plaintext
f(...) is do
  ...
end
```

is an abbreviation for

```plaintext
f(...) is do
  ...
rescue -- Nothing here
end
```

(This is a provisional rule; see next.)
The correctness of a class

(1-n) For every exported routine \( r \):

\( \{ \text{INV and Pre} \} \) do: \( \{ \text{Post and INV} \} \)

(1-m) For every creation procedure \( cp \):

\( \{ \text{Pre} \} \) do: \( \{ \text{Post} \} \) and INV

Exception correctness

For the normal body:

\( \{ \text{INV and Pre} \} \) do: \( \{ \text{Post} \} \) and INV

For the exception clause:

\( \{ \text{True} \} \) rescue: \( \{ \text{INV} \} \)
If no exception clause (2)

Absence of a rescue clause is equivalent to a default rescue clause:

\[
\begin{align*}
\text{f(...)} & \text{ do} \\
\text{...} & \text{ end}
\end{align*}
\]

is an abbreviation for

\[
\begin{align*}
\text{f(...)} & \text{ do} \\
\text{...} & \text{ rescue default_rescue} \\
\text{end}
\end{align*}
\]

The task of default_rescue is to restore the invariant.

For finer-grain exception handling

Use class EXCEPTIONS from the Kernel Library.

Some features:

- \text{exception} (code of last exception that was triggered).
- \text{assertion_violation}, etc.
- \text{raise} ("exception_name")