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
}

(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

r(...) is
require
    ...
    do
        op
    ...
    ensure
    ...
end

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)
Programming in the large - Lecture 24

Chair of Software Engineering

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");
  when others =>
    return;
end; -- sqrt
```

How not to do it

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)

```ada
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);
  if failures < Max_attempts then
    retry;
  else
    failed := True;
  end;
rescue
  failures := failures + 1
  if failures < Max_attempts then
    retry;
  else
    failed := True;
  end; -- attempt_transmission
end;
```

Transmitting over an unreliable line (2)

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

If no exception clause (1)

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

```ada
f (...) is
do
  ...
end; -- f (...) is
```

is an abbreviation for

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

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

(1-n) For every exported routine \( r \):
\[
\{ \text{INV and Pre} \} \text{ do } (\text{Post} \text{ and } \text{INV})
\]

(1-m) For every creation procedure \( cp \):
\[
\{ \text{Pre}, \text{ do } (\text{Post}, \text{ and } \text{INV})
\]

Exception correctness

For the normal body:
\[
(\text{INV and Pre}) \text{ do } (\text{Post} \text{ and } \text{INV})
\]

For the exception clause:
\[
\{ ??? } \text{ rescue } ( ??? )
\]

If no exception clause (2)

Absence of a rescue clause is equivalent to a default rescue clause:
\[
f(...) \text{ do } \\
df \text{ end} \\
\]

is an abbreviation for
\[
f(...) \text{ do } \\
\text{ rescue } \text{ default_rescue } \\
\text{ end}
\]

The task of default_rescue is to restore the invariant.

For finer-grain exception handling

Use class EXCEPTIONS from the Kernel Library.

Some features:
- `exception` (code of last exception that was triggered).
- `assertion_violation`, etc.
- `raise` ("exception_name")