
Exception Handling

- Motivation: what is the behavior of a procedure/function when certain error conditions occur?
- Two components:
 1. What is the nature of error condition?
 2. What to do when such conditions arise?
- Examples of error conditions: divide-by-zero, end-of-file, wrong type of arguments, etc.
- Exception: unanticipated event (sort of..) that disrupts the normal flow of execution. Encompass hardware and software interrupts. Just like hardware, unanticipated events must be planned for. (Identify the nature of error condition.)
- Exception handler: mechanism to deal with exception. (What to do)
- Exceptions can be raised by language implementation or programs may raise it explicitly.
- Why language support for exception handling? In the absence of support, programmer must explicitly test for exception conditions, devise mechanisms for taking actions when exceptions occur.
 - Return a value that corresponds to some error condition
 - Return an explicit "status" value, which caller must check (aka UNIX system calls)
 - Return a closure for an error-handling routine that caller can invokeThe above make programs more difficult to write, clutter the code with tedious and repetitive checks
- Exception handling move error-checking code out of line ⇒ arrange for control to branch to a handler when appropriate.

Design issues

- Are hardware-detectable errors treated as exceptions that may be handled?
- Are there any built-in exceptions?
- Should it be possible to disable exceptions?
- How are handlers represented in language? Separate program units vs. code fragments?
 - code fragment ⇒ can be embedded in units that raise exceptions.
 - Unit ⇒ use the scope mechanism to determine how code unit and exception handling unit interact.
- How is an exception occurrence bound to a handler?
 - On the unit level, how the same exception begin raised at different points in a unit is bound to different exceptions. For instance, different "divide by zero" exceptions at different points in the program may be bound to different instructions.
 - On a higher level, what if no exception handler is bound? How does propagation take place and how far does propagation go?
 - Is the handler bound statically or dynamically?
- What happens after a handler has terminated? Termination, Return to the statement that caused the problem?, Return to statement after the statement? Return to a different block?
- How do users specify exception handlers?
- Should there be default exception handlers for programs that do not provide their own?
- Can built-in exceptions be explicitly raised?

Exception handling models

- Depends on interaction between a signaler and its handler:
 1. *Termination model*: a signaler is terminated after raising an exception.
 2. *Strict-resumption model*: a signaler must be resumed after the raised exception is handled.
 3. *Resumption model*: a signaler may be resumed after the raised exception is serviced. (depends on defined action for each possible exception.)
 4. *Combination model*: a signaler can decide which of the other signaling conventions it wants to follow when it raises an exception. (Up to the programmer what to do.)
- Thus,
 1. a handler may react only passively in the termination model and the strict resumption model.
 2. but a handler has complete control over further execution of the signaler in the resumption model.
 3. A handlers response in the combination model may or may not have any effect on the signaler.
- Summary: In some way can simplify programmer's life. Instead of passing extra parameters and testing return values. Can do this once. Useful feature of real programs. Add considerable complexity to the language.

Exceptions in Ada

- Predefined Exceptions:
 - NUMERIC_ERROR, e.g., divide by zero, overflow, etc.
 - CONSTRAINT_ERROR, e.g., subscript out of range.
 - STORAGE_ERROR, e.g., out of heap or stack.
- User-defined exceptions through `exception` keyword:
`SINGULAR: exception`
Declaration of an exception follows the same static scope rules as other declarations.
- User defined exceptions must be signaled *explicitly* through `raise` statement.

How does an exception handler look?

- Exception handlers in Ada are associated with blocks and are declared at the end of block:

```
begin
  statement list
exception
  exception_handler
end
```
- Any statement that is legitimate in a block is legal (including another raise)
- Definition

```
begin
  ...
exception
  when SINGLULAR | NUMERIC_ERROR =>
    put('`matrix is singular`');
  when others =>
    put('`error`');
end
```

Exceptions in Ada - Binding

- Associate a specific exception with a handler through “raise” statement
- Example:

```
begin
  x := 4/pop() + a[i];
  y := 30;
exception
when NUMERIC_ERROR => x := 10
when CONSTRAINT_ERROR =>
  raise CONSTRAINT_ERROR
end
```
- What happens when an exception is raised in a procedure or a block?
 - **abandon** execution: do not resume evaluation of expression or statement list.
 - If unit has no handler for it, exception is implicitly propagated to the calling unit at invocation (note: trace through execution path.)
 - If no handler in main program, terminate main program.
 - If there exists a handler (or default),
 - * do exception
 - * program resumes after the block that raised exception
 - What if handler raises exception? **abandon**, propagate up.
 - Note that begin blocks may be nested, so that entire calling procedure needn't be terminated if the called procedure raises exception; e.g., programmer can put a procedure invocation within its own begin block.

Exception in Ada – cont'd.

- Another example (pseudo-Ada):

```
proc P is
  OOPS: exception;
  proc Q is
  begin
    ... raise OOPS --- E2
  R; ...
  exception
    ... when OOPS => ... --- H-E2
  end Q
  proc R is
  begin
    ... raise OOPS --- E3
  end R
begin
  ... raise OOPS ---E1;
  ... Q; R; ...
exception
  ..when OOPS => ... --- H-E1
end P
```
- When can OOPS occur:
 - E1, use H-E1
 - E2 (inside Q), use H-E2
 - E3 (inside R) ? Call from P, use H-E1. Call from Q, use H-E2
- Can say dynamic scoping? Inconsistent with rest of Ada. However useful, so caller can define what is intended. But dangerous as may end up handling error with unintended handler.
- Implementation: follow dynamic call back on stack. Pop activation records as go, since don't need to resume.

Exception Handling in C++

- How are exceptions defined ?

C++ exception handlers appear in a compound statement following a compound statement that contains code that can raise exceptions.

```
try {
  // code expected to raise objection
}
catch (formal parameter) {
  handler body
}
...
catch (formal parameter) {
  handler body
}
```
- How are exceptions raised?

throw expression;

type of the expression used to determine which handler will be used.

Exception handling in C++

- What happens when an exception is raised in a try block?
 - Terminate execution of code in the try block.
 - Search for a matching handler that immediately follow the try construct.
 - If a match is found,
 - * Handler is executed.
 - * Control flow to *first* statement following the last handler in the sequence of handlers of which it is an element.
 - If a match is not found:
 - * Search for a matching handler in a dynamically surrounding try-block.
 - * If no matching handler found, call `terminate()`.
- C++ exception handling mechanism vs. Ada:
 - Similar in i) binding exceptions to handlers is static, and ii) unhandled exceptions are propagated to the function's caller (dynamic).
 - Different in i) no built-in hardware-detectable exceptions, ii) exceptions are not named.

Exceptions in Java

- How are exceptions defined?

- Exception in Java: objects
- User can define different exception types, each derivable from superclass `Exception`

```
public class AnException extends Exception {
    public String attrName;
    AnException(String name) {
        ...
        // hold information that was passed
        attrName = name;
    }
}
```

- How are exceptions raised?

- Raised using `throw` statement

```
public void
    replaceVal(String name,
               Object newVal)
    throws AnException {
    Attr attr = find(name);
    if (attr == null)
        throw new AnException(name);
    attr.setValue(newVal);
}
```

Exceptions in Java - cont'd.

- How are exceptions bound?

- Note that every method describes all exception it throws.
- All "throw" statements must throw objects of type shown in declaration

- try and catch:

```
try {
    statements
} catch (exception_type1 id1) {
    ...
} catch (exception_type2 id2) {
    ...
} finally {
    ...
}
```

- Example:

```
Object value = new Integer(8);
try {
    attributedObj.replaceVal('`Age`', value);
} catch (AnException e) {
    Attr attr = new Attr(e.attrName, value);
    attributeObj.add(attr);
}
```

- How are exceptions caught at runtime?
- What happens once an exception handler executed?