A Book Chapter

By Narendra Kumar Chahar

on **Exception Handling**

Index:

Chapter 6

Exception Handling

Exception handling:

• In a language without exception handling

–When an exception occurs, control goes to the operating system, where a message is displayed and the program is terminated.

• In a language with exception handling.

–Programs are allowed to trap some exceptions, thereby providing the possibility of fixing the problem and continuing.

Basic Concepts

•Many languages allow programs to trap input/output errors (including EOF).

•An exception is any unusual event, either erroneous or not, detectable by either hardware or software, that may require special processing.

•The special processing that may be required after detection of an exceptions called exception handling.

•The exception handling code unit is called an exception handler.

Exception Handling Alternatives

•An exception is raised when its associated event occurs.

•A language that does not have exception handling capabilities can still define, detect, raise, and handle exceptions (user defined, software detected).

•Alternatives:

–Send an auxiliary parameter or use the return value to indicate the return status of a subprogram.

–Pass an exception handling subprogram to all subprograms.

Advantages of Built-in Exception Handling

•Error detection code is tedious to write and it clutters the program.

•Exception handling encourages programmers to consider many different possible errors.

•Exception propagation allows a high level of reuse of exception handling code.

Design Issues

•How are user-defined exceptions specified?

•Should there be default exception handlers for programs that do not provide their own?

•Can built-in exceptions be explicitly raised?

•Are hardware-detectable errors treated as exceptions that can be handled?

•Are there any built-in exceptions?

•How can exceptions be disabled, if at all?

•How and where are exception handlers specified and what is their scope?

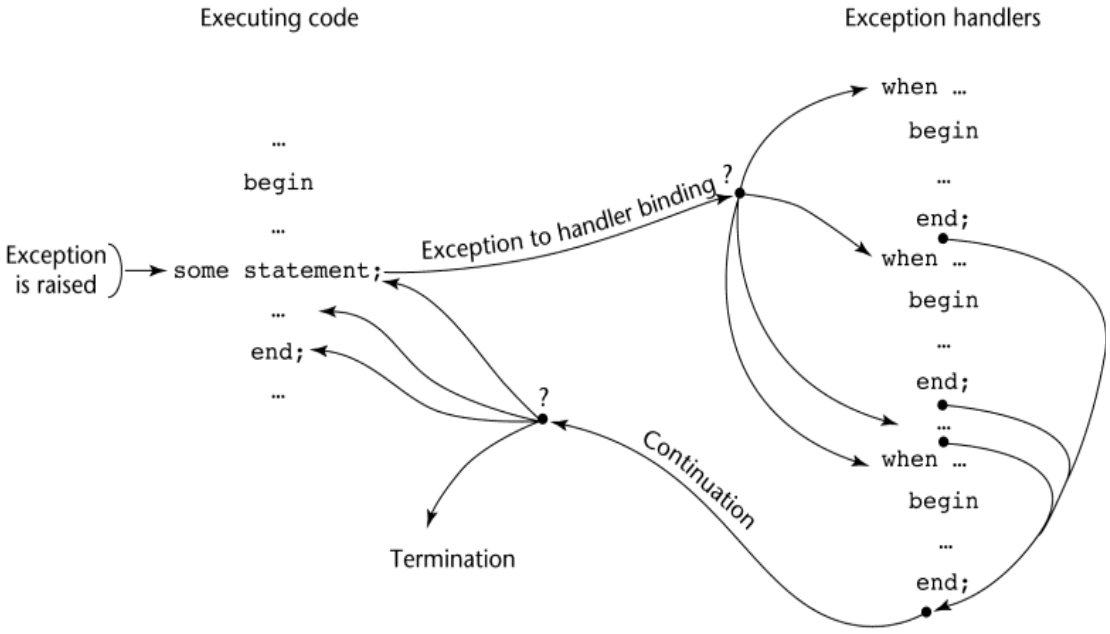
•How is an exception occurrence bound to an exception handler?

•Can information about the exception be passed to the handler?

•Where does execution continue, if at all, after an exception handler completes its execution? (Continuation vs. resumption)

•Is some form of finalization provided?

Exception Handling Control Flow



Exception Handling in Ada

•The frame of an exception handler in Ada is either a subprogram body, a package body, a task, or a block.

•Because exception handlers are usually local to the code in which the exception can be raised, they do not have parameters.

Ada Exception Handlers

•Handler form:

when exception_choice{|exception_choice} => statement_sequence

...

[when others =>

statement_sequence]

exception_choice form:

exception_name | others

•Handlers are placed at the end of the block or unit in which they occur.

Binding Exceptions to Handlers

•If the block or unit in which an exception is raised does not have a handler for that exception, the exception is propagated elsewhere to be handled.

–Procedures - propagate it to the caller.

–Blocks - propagate it to the scope in which it appears.

–Package body - propagate it to the declaration part of the unit that declared the package (if it is a library unit, the program is terminated).

–Task - no propagation; if it has a handler, execute it; in either case, mark it "completed"

Continuation

•The block or unit that raises an exception but does not handle it is always terminated (also any block or unit to which it is propagated that does not handle it)

Other Design Choices

•User-defined Exceptions form:

exception_name_list **:** exception**;**

•Raising Exceptions form:

raise [exception_name]

–(the exception name is not required if it is in a handler--in this case, it propagates the same exception).

•Exception conditions can be disabled with:

pragma SUPPRESS(exception_list)

Predefined Exceptions

•CONSTRAINT_ERROR - index constraints, range constraints, etc.

•NUMERIC_ERROR - numeric operation cannot return a correct value (overflow, division by zero, etc.)

•PROGRAM_ERROR - call to a subprogram whose body has not been elaborated.

•STORAGE_ERROR - system runs out of heap.

•TASKING_ERROR - an error associated with tasks.

Evaluation

•The Ada design for exception handling embodies the state-of-the-art in language design in 1980.

•A significant advance over PL/I

•Ada was the only widely used language with exception handling until it was added to C++.

Exception Handling in C++

•Added to C++ in 1990

•Design is based on that of CLU, Ada, and ML

C++ Exception Handlers

•Exception Handlers Form:

```
try {
-- code that is expected to raise an exception
}
catch (formal parameter) {
-- handler code
}
...
catch (formal parameter) {
-- handler code
}
```

The catch Function

•catch is the name of all handlers--it is an overloaded name, so the formal parameter of each must be unique.

•The formal parameter need not have a variable.

–It can be simply a type name to distinguish the handler it is in from others.

•The formal parameter can be used to transfer information to the handler.

•The formal parameter can be an ellipsis, in which case it handles all exceptions not yet handled.

Throwing Exceptions

•Exceptions are all raised explicitly by the statement:

throw [expression];

•The brackets are meta symbols.

•A throw without an operand can only appear in a handler; when it appears, it simply re-raises the exception, which is then handled elsewhere.

•The type of the expression disambiguates the intended handler.

Unhandled Exceptions

•An unhandled exception is propagated to the caller of the function in which it is raised.

•This propagation continues to the main function.

Continuation

•After a handler completes its execution, control flows to the first statement after the last handler in the sequence of handlers of which it is an element.

•Other design choices

–All exceptions are user-defined.

–Exceptions are neither specified nor declared.

–Functions can list the exceptions they may raise.

–Without a specification, a function can raise any exception (the throw clause).

Evaluation

•It is odd that exceptions are not named and that hardware- and system software-

detectable exceptions cannot be handled.

•Binding exceptions to handlers through the type of the parameter certainly does not promote readability.

Exception Handling in Java

•Based on that of C++, but more in line with OOP philosophy.

•All exceptions are objects of classes that are descendants of the Throw able class.

Classes of Exceptions

•The Java library includes two subclasses of Throw able :

–Error

•Thrown by the Java interpreter for events such as heap overflow.

•Never handled by user programs.

–Exception

•User-defined exceptions are usually subclasses of this.

•Has two predefined subclasses, IO Exception and Runtime Exception (e.g., Array Index Out Of Bounds Exception and Null Pointer Exception.

References:

[1]. Goodenough, John B. "Exception handling: Issues and a proposed notation." *Communications of the ACM* 18.12 (1975): 683-696.

[2]. Cristian, Flaviu. *Exception handling*. IBM Thomas J. Watson Research Division, 1987.

[3]. Buhr, Peter A., and WY Russell Mok. "Advanced exception handling mechanisms." *IEEE Transactions on Software Engineering* 26.9 (2000): 820-836.

[4]. Lerner, Barbara Staudt, et al. "Exception handling patterns for process modeling." *IEEE Transactions on Software Engineering* 36.2 (2010): 162-183.

[5]. Sinha, Saurabh, and Mary Jean Harrold. "Analysis and testing of programs with exception handling constructs." *IEEE Transactions on Software Engineering* 26.9 (2000): 849-871.

[6]. Cabral, Bruno, and Paulo Marques. "Exception handling: A field study in java and. net." *European Conference on Object-Oriented Programming*. Springer, Berlin, Heidelberg, 2007.