EXCEPTION HANDLING

The definition of an exception is based on the observation that each [procedure](https://en.wikipedia.org/wiki/Subroutine) has a [precondition](https://en.wikipedia.org/wiki/Precondition), a set of circumstances for which it will terminate "normally".[[1]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Cristian-1) An exception handling mechanism allows the procedure to *raise an exception*[[2]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-FOOTNOTEGoodenough1975b683%E2%80%93684-2) if this precondition is violated,[[1]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Cristian-1) for example if the procedure has been called on an abnormal set of arguments. The exception handling mechanism then *handles* the exception.[[3]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEGoodenough1975b684-3) The precondition, and the definition of exception, is [subjective](https://en.wikipedia.org/wiki/Subjectivity). The set of "normal" circumstances is defined entirely by the programmer, e.g. the programmer may deem division by zero to be undefined, hence an exception, or devise some behavior such as returning zero or a special "ZERO DIVIDE" value (circumventing the need for exceptions).[[4]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEBlack198213%E2%80%9315-4) Common exceptions include an invalid argument (e.g. value is outside of the [domain of a function](https://en.wikipedia.org/wiki/Domain_of_a_function)), an unavailable resource (like a missing file, a hard disk error, or out-of-memory errors), or that the routine has detected a normal condition that requires special handling, e.g., attention, end of file.

Exception handling solves the [semipredicate problem](https://en.wikipedia.org/wiki/Semipredicate_problem" \o "Semipredicate problem), in that the mechanism distinguishes normal return values from erroneous ones. In languages without built-in exception handling such as C, routines would need to signal the error in some other way, such as the common [return code](https://en.wikipedia.org/wiki/Return_code) and [errno](https://en.wikipedia.org/wiki/Errno" \o "Errno) pattern.[[5]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Lang-5) Taking a broad view, errors can be considered to be a proper subset of exceptions,[[6]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-FOOTNOTELevin19775-6) and explicit error mechanisms such as errno can be considered (verbose) forms of exception handling.[[5]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Lang-5) The term "exception" is preferred to "error" because it does not imply that anything is wrong - a condition viewed as an error by one procedure or programmer may not be viewed that way by another. Even the term "exception" may be misleading because its [typical connotation](https://en.wiktionary.org/wiki/exception) of "outlier" indicates that something infrequent or unusual has occurred, when in fact raising the exception may be a normal and usual situation in the program.[[7]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-CLU-7) For example, suppose a lookup function for an [associative array](https://en.wikipedia.org/wiki/Associative_array) throws an exception if the key has no value associated. Depending on context, this "key absent" exception may occur much more often than a successful lookup.[[8]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTELevin19774-8)

A major influence on the scope and use of exceptions is social pressure, i.e. "examples of use, typically found in core libraries, and code examples in technical books, magazine articles, and online discussion forums, and in an organization’s code standards".[[9]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Kiniry-9)

History[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=2" \o "Edit section: History)]

The first hardware exception handling was found in the [UNIVAC I](https://en.wikipedia.org/wiki/UNIVAC_I) from 1951. Arithmetic overflow executed two instructions at address 0, which could transfer control or fix up the result.[[10]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Smotherman-10)

Software exception handling developed in the 1960s and 1970s. [LISP 1.5](https://en.wikipedia.org/wiki/LISP_1.5) (1958-1961)[[11]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-11) allowed exceptions to be raised by the ERROR pseudo-function, similarly to errors raised by the interpreter or compiler. Exceptions were caught by the ERRORSET keyword, which returned NIL in case of an error, instead of terminating the program or entering the debugger.[[12]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-12) [PL/I](https://en.wikipedia.org/wiki/PL/I#ON-units_and_exception_handling) introduced its own form of exception handling circa 1964, allowing interrupts to be handled with ON units.[[13]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-13) [MacLisp](https://en.wikipedia.org/wiki/MacLisp" \o "MacLisp) observed that ERRSET and ERR were used not only for error raising, but for non-local control flow, and thus added two new keywords, CATCH and THROW (June 1972).[[14]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEGabrielSteele20083-14) The cleanup behavior now generally called "finally" was introduced in [NIL](https://en.wikipedia.org/wiki/NIL_(programming_language)) (New Implementation of LISP) in the mid- to late-1970s as UNWIND-PROTECT.[[15]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEWhite1979194-15) This was then adopted by [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp). Contemporary with this was dynamic-wind in Scheme, which handled exceptions in closures. The first papers on structured exception handling were [Goodenough (1975a)](https://en.wikipedia.org/wiki/Exception_handling" \l "CITEREFGoodenough1975a) and [Goodenough (1975b)](https://en.wikipedia.org/wiki/Exception_handling" \l "CITEREFGoodenough1975b).[[16]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEStroustrup1994392-16) Exception handling was subsequently widely adopted by many programming languages from the 1980s onward.

Hardware exceptions[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=3" \o "Edit section: Hardware exceptions)]

*Main article:*[*Interrupt*](https://en.wikipedia.org/wiki/Interrupt)

There is no clear consensus as to the exact meaning of an exception with respect to hardware.[[17]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-17) From the implementation point of view, it is handled identically to an [interrupt](https://en.wikipedia.org/wiki/Interrupt): the processor halts execution of the current program, looks up the [interrupt handler](https://en.wikipedia.org/wiki/Interrupt_handler) in the [interrupt vector table](https://en.wikipedia.org/wiki/Interrupt_vector_table) for that exception or interrupt condition, saves state, and switches control.

IEEE 754 floating-point exceptions[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=4" \o "Edit section: IEEE 754 floating-point exceptions)]

Exception handling in the [IEEE 754](https://en.wikipedia.org/wiki/IEEE_754) [floating-point](https://en.wikipedia.org/wiki/Floating-point_arithmetic#Dealing_with_exceptional_cases) standard refers in general to exceptional conditions and defines an exception as "an event that occurs when an operation on some particular operands has no outcome suitable for every reasonable application. That operation might signal one or more exceptions by invoking the default or, if explicitly requested, a language-defined alternate handling."

By default, an IEEE 754 exception is resumable and is handled by substituting a predefined value for different exceptions, e.g. infinity for a divide by zero exception, and providing [status flags](https://en.wikipedia.org/w/index.php?title=Floating-point_arithmetict&action=edit&redlink=1) for later checking of whether the exception occurred (see [C99 programming language](https://en.wikipedia.org/wiki/C99#IEEE_754_floating-point_support) for a typical example of handling of IEEE 754 exceptions). An exception-handling style enabled by the use of status flags involves: first computing an expression using a fast, direct implementation; checking whether it failed by testing status flags; and then, if necessary, calling a slower, more numerically robust, implementation.[[18]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Xiaoye_Li_and_James_Demmel_1994_983%E2%80%93992-18)

The IEEE 754 standard uses the term "trapping" to refer to the calling of a user-supplied exception-handling routine on exceptional conditions, and is an optional feature of the standard. The standard recommends several usage scenarios for this, including the implementation of non-default pre-substitution of a value followed by resumption, to concisely handle [removable singularities](https://en.wikipedia.org/wiki/Removable_singularity).[[18]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Xiaoye_Li_and_James_Demmel_1994_983%E2%80%93992-18)[[19]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-grail-19)[[20]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-20)

The default IEEE 754 exception handling behaviour of resumption following pre-substitution of a default value avoids the risks inherent in changing flow of program control on numerical exceptions. For example, the 1996 [Cluster spacecraft](https://en.wikipedia.org/wiki/Cluster_(spacecraft)) launch ended in a catastrophic explosion due in part to the [Ada](https://en.wikipedia.org/wiki/Ada_(programming_language)" \o "Ada (programming language)) exception handling policy of aborting computation on arithmetic error. [William Kahan](https://en.wikipedia.org/wiki/William_Kahan) claims the default IEEE 754 exception handling behavior would have prevented this.[[19]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-grail-19)

Exception support in programming languages[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=5" \o "Edit section: Exception support in programming languages)]

Software exception handling and the support provided by software tools differs somewhat from what is understood by exception handling in hardware, but similar concepts are involved. In programming language mechanisms for exception handling, the term *exception* is typically used in a specific sense to denote a data structure storing information about an exceptional condition. One mechanism to transfer control, or *raise* an exception, is known as a *throw*. The exception is said to be *thrown*. Execution is transferred to a "catch".

Programming languages differ substantially in their notion of what an exception is. Contemporary languages can roughly be divided into two groups:[[9]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-Kiniry-9)

* Languages where exceptions are designed to be used as flow control structures: Ada, Modula-3, ML, OCaml, PL/I, Python, and Ruby fall in this category. For example, [Python's iterators](https://en.wikipedia.org/wiki/Iterator#Python) throw StopIteration exceptions to signal that there are no further items produced by the iterator.[[21]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-21)
* Languages where exceptions are only used to handle abnormal, unpredictable, erroneous situations: C++,[[22]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-22) Java,[[23]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-EffectiveJava-23) C#, Common Lisp, Eiffel, and Modula-2.

PL/I used dynamically scoped exceptions. PL/I exception handling included events that are not errors, e.g., attention, end-of-file, modification of listed variables.[*[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)*]

**Syntax**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=6" \o "Edit section: Syntax)]

*Further information:*[*Exception handling syntax*](https://en.wikipedia.org/wiki/Exception_handling_syntax)

Many computer languages have built-in syntactic support for exceptions and exception handling. This includes [ActionScript](https://en.wikipedia.org/wiki/ActionScript), [Ada](https://en.wikipedia.org/wiki/Ada_programming_language), [BlitzMax](https://en.wikipedia.org/wiki/BlitzMax), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Clojure](https://en.wikipedia.org/wiki/Clojure), [COBOL](https://en.wikipedia.org/wiki/COBOL), [D](https://en.wikipedia.org/wiki/D_programming_language), [ECMAScript](https://en.wikipedia.org/wiki/ECMAScript), [Eiffel](https://en.wikipedia.org/wiki/Eiffel_(programming_language)), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [ML](https://en.wikipedia.org/wiki/ML_programming_language), [Object Pascal](https://en.wikipedia.org/wiki/Object_Pascal) (e.g. [Delphi](https://en.wikipedia.org/wiki/Delphi_(programming_language)), [Free Pascal](https://en.wikipedia.org/wiki/Free_Pascal), and the like), [PowerBuilder](https://en.wikipedia.org/wiki/PowerBuilder), [Objective-C](https://en.wikipedia.org/wiki/Objective-C), [OCaml](https://en.wikipedia.org/wiki/OCaml" \o "OCaml), [PHP](https://en.wikipedia.org/wiki/PHP) (as of version 5), [PL/I](https://en.wikipedia.org/wiki/PL/I), [PL/SQL](https://en.wikipedia.org/wiki/PL/SQL), [Prolog](https://en.wikipedia.org/wiki/Prolog), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [REALbasic](https://en.wikipedia.org/wiki/REALbasic), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)), [Seed7](https://en.wikipedia.org/wiki/Seed7), [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk), [Tcl](https://en.wikipedia.org/wiki/Tcl), [Visual Prolog](https://en.wikipedia.org/wiki/Visual_Prolog) and most [.NET](https://en.wikipedia.org/wiki/.NET_Framework) languages.

Excluding minor syntactic differences, there are only a couple of exception handling styles in use. In the most popular style, an exception is initiated by a special statement (throw or raise) with an exception object (e.g. with Java or Object Pascal) or a value of a special extendable enumerated type (e.g. with Ada or SML). The scope for exception handlers starts with a marker clause (try or the language's block starter such as begin) and ends in the start of the first handler clause (catch, except, rescue). Several handler clauses can follow, and each can specify which exception types it handles and what name it uses for the exception object. As a minor variation, some languages use a single handler clause, which deals with the class of the exception internally.

Also common is a related clause (finally or ensure) that is executed whether an exception occurred or not, typically to release resources acquired within the body of the exception-handling block. Notably, C++ does not provide this construct, recommending instead the [Resource Acquisition Is Initialization](https://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization) (RAII) technique which frees resources using [destructors](https://en.wikipedia.org/wiki/Destructor_(computer_programming)).[[24]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-24) According to a 2008 paper by Westley Weimer and [George Necula](https://en.wikipedia.org/wiki/George_Necula), the syntax of the try...finally blocks in Java is a contributing factor to software defects. When a method needs to handle the acquisition and release of 3–5 resources, programmers are apparently unwilling to nest enough blocks due to readability concerns, even when this would be a correct solution. It is possible to use a single try...finally block even when dealing with multiple resources, but that requires a correct use of [sentinel values](https://en.wikipedia.org/wiki/Sentinel_value), which is another common source of bugs for this type of problem.[[25]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-toplas2008-25): 8:6–8:7

Python and Ruby also permit a clause (else) that is used in case no exception occurred before the end of the handler's scope was reached.

In its whole, exception handling code might look like this (in [Java](https://en.wikipedia.org/wiki/Java_(programming_language))-like [pseudocode](https://en.wikipedia.org/wiki/Pseudocode" \o "Pseudocode)):

**try** {

line = console.readLine();

**if** (line.length() == 0) {

**throw** **new** EmptyLineException("The line read from console was empty!");

}

console.printLine("Hello %s!" % line);

}

**catch** (EmptyLineException e) {

console.printLine("Hello!");

}

**catch** (Exception e) {

console.printLine("Error: " + e.message());

}

**else** {

console.printLine("The program ran successfully.");

}

**finally** {

console.printLine("The program is now terminating.");

}

C does not have try-catch exception handling, but uses [return codes](https://en.wikipedia.org/wiki/Return_code) for error checking. The [setjmp and longjmp](https://en.wikipedia.org/wiki/Setjmp.h" \o "Setjmp.h) standard library functions can be used to implement try-catch handling via macros.[[26]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-26)

[Perl](https://en.wikipedia.org/wiki/Perl) 5 uses die for throw and eval {} **if** ($@) {} for try-catch. It has CPAN modules that offer try-catch semantics.[[27]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-27)

**Termination and resumption semantics**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=7" \o "Edit section: Termination and resumption semantics)]

When an exception is thrown, the program searches back through the [stack](https://en.wikipedia.org/wiki/Call_stack) of function calls until an exception handler is found. Some languages call for [unwinding](https://en.wikipedia.org/wiki/Stack_unwinding) the stack as this search progresses. That is, if function f, containing a handler H for exception E, calls function g, which in turn calls function h, and an exception E occurs in h, then functions h and g may be terminated, and H in f will handle E. This is said to be termination semantics. Alternately, the exception handling mechanisms may not unwind the stack on entry[[note 1]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-28) to an exception handler, giving the exception handler the option to restart the computation, resume or unwind. This allows the program to continue the computation at exactly the same place where the error occurred (for example when a previously missing file has become available) or to implement notifications, logging, queries and fluid variables on top of the exception handling mechanism (as done in Smalltalk). Allowing the computation to resume where it left off is termed resumption semantics.

There are theoretical and design arguments in favor of either decision. C++ standardization discussions in 1989–1991 resulted in a definitive decision to use termination semantics in C++.[[28]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEStroustrup199416.6_Exception_Handling:_Resumption_vs._Termination,_pp._390%E2%80%93393-29) [Bjarne Stroustrup](https://en.wikipedia.org/wiki/Bjarne_Stroustrup" \o "Bjarne Stroustrup) cites a presentation by [Jim Mitchell](https://en.wikipedia.org/wiki/James_G._Mitchell) as a key data point:

Jim had used exception handling in half a dozen languages over a period of 20 years and was an early proponent of resumption semantics as one of the main designers and implementers of Xerox's [Cedar/Mesa](https://en.wikipedia.org/wiki/Mesa/Cedar) system. His message was

“termination is preferred over resumption; this is not a matter of opinion but a matter of years of experience. Resumption is seductive, but not valid.”

He backed this statement with experience from several operating systems. The key example was Cedar/Mesa: It was written by people who liked and used resumption, but after ten years of use, there was only one use of resumption left in the half million line system – and that was a context inquiry. Because resumption wasn't actually necessary for such a context inquiry, they removed it and found a significant speed increase in that part of the system. In each and every case where resumption had been used it had – over the ten years – become a problem and a more appropriate design had replaced it. Basically, every use of resumption had represented a failure to keep separate levels of abstraction disjoint.[[16]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-FOOTNOTEStroustrup1994392-16)

Exception-handling languages with resumption include [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp) with its [Condition System](https://en.wikipedia.org/wiki/Exception_handling#Condition_systems), PL/I, Dylan, [R](https://en.wikipedia.org/wiki/R_(programming_language)),[[29]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-30) and [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk). However, the majority of newer programming languages follow C++ and use termination semantics.

**Exception handling implementation**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=8" \o "Edit section: Exception handling implementation)]

The implementation of exception handling in programming languages typically involves a fair amount of support from both a code generator and the [runtime system](https://en.wikipedia.org/wiki/Runtime_system) accompanying a compiler. (It was the addition of exception handling to C++ that ended the useful lifetime of the original C++ compiler, [Cfront](https://en.wikipedia.org/wiki/Cfront" \o "Cfront).[[30]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-31)) Two schemes are most common. The first, *dynamic registration*, generates code that continually updates structures about the program state in terms of exception handling.[[31]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-32) Typically, this adds a new element to the [stack frame layout](https://en.wikipedia.org/wiki/Call_stack) that knows what handlers are available for the function or method associated with that frame; if an exception is thrown, a pointer in the layout directs the runtime to the appropriate handler code. This approach is compact in terms of space, but adds execution overhead on frame entry and exit. It was commonly used in many Ada implementations, for example, where complex generation and runtime support was already needed for many other language features. Microsoft's 32-bit [Structured Exception Handling](https://en.wikipedia.org/wiki/Structured_Exception_Handling) (SEH) uses this approach with a separate exception stack.[[32]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-33) Dynamic registration, being fairly straightforward to define, is amenable to [proof of correctness](https://en.wikipedia.org/wiki/Proof_of_correctness).[[33]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-34)

The second scheme, and the one implemented in many production-quality C++ compilers and 64-bit Microsoft [SEH](https://en.wikipedia.org/wiki/Structured_Exception_Handling), is a *table-driven* approach. This creates static tables at [compile time](https://en.wikipedia.org/wiki/Compile_time) and [link time](https://en.wikipedia.org/wiki/Link_time) that relate ranges of the [program counter](https://en.wikipedia.org/wiki/Program_counter) to the program state with respect to exception handling.[[34]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-35) Then, if an exception is thrown, the runtime system looks up the current instruction location in the tables and determines what handlers are in play and what needs to be done. This approach minimizes executive overhead for the case where an exception is not thrown. This happens at the cost of some space, but this space can be allocated into read-only, special-purpose data sections that are not loaded or relocated until an exception is actually thrown.[[35]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-cppeh-36) The location (in memory) of the code for handling an exception need not be located within (or even near) the region of memory where the rest of the function's code is stored. So if an exception is thrown then a performance hit – roughly comparable to a function call[[36]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-MiscrosoftDocsExceptions-37) – may occur if the necessary exception handling code needs to be loaded/cached. However, this scheme has minimal performance cost if no exception is thrown. Since exceptions in C++ are supposed to be *exceptional* (i.e. uncommon/rare) events, the phrase "zero-cost exceptions"[[note 2]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-38) is sometimes used to describe exception handling in C++. Like [runtime type identification](https://en.wikipedia.org/wiki/Runtime_type_identification) (RTTI), exceptions might not adhere to C++'s [zero-overhead principle](https://en.cppreference.com/w/cpp/language/Zero-overhead_principle) as implementing exception handling at run-time requires a non-zero amount of memory for the lookup table.[[37]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-StroustrupExceptions2019-39) For this reason, exception handling (and RTTI) can be disabled in many C++ compilers, which may be useful for systems with very limited memory[[37]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-StroustrupExceptions2019-39) (such as [embedded systems](https://en.wikipedia.org/wiki/Embedded_system)). This second approach is also superior in terms of achieving [thread safety](https://en.wikipedia.org/wiki/Thread_safety)[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)].

Other definitional and implementation schemes have been proposed as well. For languages that support [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming" \o "Metaprogramming), approaches that involve no overhead at all (beyond the already present support for [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_science))) have been advanced.[[38]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-40)

**Exception handling based on design by contract**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=9" \o "Edit section: Exception handling based on design by contract)]

A different view of exceptions is based on the principles of [design by contract](https://en.wikipedia.org/wiki/Design_by_contract) and is supported in particular by the [Eiffel language](https://en.wikipedia.org/wiki/Eiffel_(programming_language)). The idea is to provide a more rigorous basis for exception handling by defining precisely what is "normal" and "abnormal" behavior. Specifically, the approach is based on two concepts:

* **Failure**: the inability of an operation to fulfill its contract. For example, an addition may produce an arithmetic overflow (it does not fulfill its contract of computing a good approximation to the mathematical sum); or a routine may fail to meet its postcondition.
* **Exception**: an abnormal event occurring during the execution of a routine (that routine is the "*recipient*" of the exception) during its execution. Such an abnormal event results from the *failure* of an operation called by the routine.

The "Safe Exception Handling principle" as introduced by Bertrand Meyer in [Object-Oriented Software Construction](https://en.wikipedia.org/wiki/Object-Oriented_Software_Construction) then holds that there are only two meaningful ways a routine can react when an exception occurs:

* Failure, or "organized panic": The routine fixes the object's state by re-establishing the invariant (this is the "organized" part), and then fails (panics), triggering an exception in its caller (so that the abnormal event is not ignored).
* Retry: The routine tries the algorithm again, usually after changing some values so that the next attempt will have a better chance to succeed.

In particular, simply ignoring an exception is not permitted; a block must either be retried and successfully complete, or propagate the exception to its caller.

Here is an example expressed in Eiffel syntax. It assumes that a routine send\_fast is normally the better way to send a message, but it may fail, triggering an exception; if so, the algorithm next uses send\_slow, which will fail less often. If send\_slow fails, the routine send as a whole should fail, causing the caller to get an exception.

send (m: **MESSAGE**) is

*-- Send m through fast link, if possible, otherwise through slow link.*

**local**

tried\_fast, tried\_slow: **BOOLEAN**

**do**

**if** tried\_fast **then**

tried\_slow := **True**

send\_slow (m)

**else**

tried\_fast := **True**

send\_fast (m)

**end**

**rescue**

**if** **not** tried\_slow **then**

**retry**

**end**

**end**

The boolean local variables are initialized to False at the start. If send\_fast fails, the body (**do** clause) will be executed again, causing execution of send\_slow. If this execution of send\_slow fails, the **rescue** clause will execute to the end with no **retry** (no **else** clause in the final **if**), causing the routine execution as a whole to fail.

This approach has the merit of defining clearly what "normal" and "abnormal" cases are: an abnormal case, causing an exception, is one in which the routine is unable to fulfill its contract. It defines a clear distribution of roles: the **do** clause (normal body) is in charge of achieving, or attempting to achieve, the routine's contract; the **rescue** clause is in charge of reestablishing the context and restarting the process, if this has a chance of succeeding, but not of performing any actual computation.

Although exceptions in Eiffel have a fairly clear philosophy, Kiniry (2006) criticizes their implementation because "Exceptions that are part of the language definition are represented by INTEGER values, developer-defined exceptions by STRING values. [...] Additionally, because they are basic values and not objects, they have no inherent semantics beyond that which is expressed in a helper routine which necessarily cannot be foolproof because of the representation overloading in effect (e.g., one cannot differentiate two integers of the same value)."[[9]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Kiniry-9)

**Uncaught exceptions**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=10" \o "Edit section: Uncaught exceptions)]

Contemporary applications face many design challenges when considering exception handling strategies. Particularly in modern enterprise level applications, exceptions must often cross process boundaries and machine boundaries. Part of designing a solid exception handling strategy is recognizing when a process has failed to the point where it cannot be economically handled by the software portion of the process.[[39]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-41)

If an exception is thrown and not caught (operationally, an exception is thrown when there is no applicable handler specified), the uncaught exception is handled by the runtime; the routine that does this is called the **uncaught exception handler**.[[40]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-cocoa-42)[[41]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-43) The most common default behavior is to terminate the program and print an error message to the console, usually including debug information such as a string representation of the exception and the [stack trace](https://en.wikipedia.org/wiki/Stack_trace).[[40]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-cocoa-42)[[42]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-44)[[43]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-45) This is often avoided by having a top-level (application-level) handler (for example in an [event loop](https://en.wikipedia.org/wiki/Event_loop)) that catches exceptions before they reach the runtime.[[40]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-cocoa-42)[[44]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-pmotw-46)

Note that even though an uncaught exception may result in the *program* terminating abnormally (the program may not be correct if an exception is not caught, notably by not rolling back partially completed transactions, or not releasing resources), the *process* terminates normally (assuming the runtime works correctly), as the runtime (which is controlling execution of the program) can ensure orderly shutdown of the process.

In a multithreaded program, an uncaught exception in a thread may instead result in termination of just that thread, not the entire process (uncaught exceptions in the thread-level handler are caught by the top-level handler). This is particularly important for servers, where for example a [servlet](https://en.wikipedia.org/wiki/Servlet" \o "Servlet) (running in its own thread) can be terminated without the server overall being affected.

This default uncaught exception handler may be overridden, either globally or per-thread, for example to provide alternative logging or end-user reporting of uncaught exceptions, or to restart threads that terminate due to an uncaught exception. For example, in Java this is done for a single thread via [Thread.setUncaughtExceptionHandler](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html" \l "setUncaughtExceptionHandler-java.lang.Thread.UncaughtExceptionHandler-) and globally via [Thread.setDefaultUncaughtExceptionHandler](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html" \l "setDefaultUncaughtExceptionHandler-); in Python this is done by modifying [sys.excepthook](https://docs.python.org/3/library/sys.html" \l "sys.excepthook).

**Checked exceptions**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=11" \o "Edit section: Checked exceptions)]

[Java](https://en.wikipedia.org/wiki/Java_(programming_language)) introduced the notion of checked exceptions,[[45]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-47)[[46]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-48) which are special classes of exceptions. The checked exceptions that a method may raise must be part of the method's [signature](https://en.wikipedia.org/wiki/Type_signature). For instance, if a method might throw an IOException, it must declare this fact explicitly in its method signature. Failure to do so raises a compile-time error. According to Hanspeter Mössenböck, checked exceptions are less convenient but more robust.[[47]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-49) Checked exceptions can, at [compile time](https://en.wikipedia.org/wiki/Compile_time), reduce the incidence of unhandled exceptions surfacing at [runtime](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)) in a given application.

Kiniry writes that "As any Java programmer knows, the volume of [try catch](https://en.wikipedia.org/wiki/Try_catch) code in a typical Java application is sometimes larger than the comparable code necessary for explicit formal parameter and return value checking in other languages that do not have checked exceptions. In fact, the general consensus among in-the-trenches Java programmers is that dealing with checked exceptions is nearly as unpleasant a task as writing documentation. Thus, many programmers report that they “resent” checked exceptions.".[[9]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-Kiniry-9) [Martin Fowler](https://en.wikipedia.org/wiki/Martin_Fowler_(software_engineer)) has written "...on the whole I think that exceptions are good, but Java checked exceptions are more trouble than they are worth."[[48]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Eckel-50) As of 2006 no major programming language has followed Java in adding checked exceptions.[[48]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Eckel-50) For example, [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) does not require or allow declaration of any exception specifications, with the following posted by Eric Gunnerson:[[49]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-51)[[9]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Kiniry-9)[[48]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Eckel-50)

"Examination of small programs leads to the conclusion that requiring exception specifications could both enhance developer productivity and enhance code quality, but experience with large software projects suggests a different result – decreased productivity and little or no increase in code quality."

[Anders Hejlsberg](https://en.wikipedia.org/wiki/Anders_Hejlsberg) describes two concerns with checked exceptions:[[50]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-Trouble-52)

* Versioning: A method may be declared to throw exceptions X and Y. In a later version of the code, one cannot throw exception Z from the method, because it would make the new code incompatible with the earlier uses. Checked exceptions require the method's callers to either add Z to their throws clause or handle the exception. Alternately, Z may be misrepresented as an X or a Y.
* Scalability: In a hierarchical design, each systems may have several subsystems. Each subsystem may throw several exceptions. Each parent system must deal with the exceptions of all subsystems below it, resulting in an exponential number of exceptions to be dealt with. Checked exceptions require all of these exceptions to be dealt with explicitly.

To work around these, Hejlsberg says programmers resort to circumventing the feature by using a **throws** Exception declaration. Another circumvention is to use a **try** { ... } **catch** (Exception e) {} handler.[[50]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Trouble-52) This is referred to as catch-all exception handling or **Pokémon exception handling** after [the show](https://en.wikipedia.org/wiki/Pok%C3%A9mon)'s catchphrase "Gotta Catch ‘Em All!".[[51]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-53) The Java Tutorials discourage catch-all exception handling as it may catch exceptions "for which the handler was not intended".[[52]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-54) Still another discouraged circumvention is to make all exceptions subclass RuntimeException.[[53]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-55) An encouraged solution is to use a catch-all handler or throws clause but with a specific [superclass](https://en.wikipedia.org/wiki/Superclass_(computer_science)" \o "Superclass (computer science)) of all potentially thrown exceptions rather than the general superclass Exception. Another encouraged solution is to define and declare exception types that are suitable for the level of abstraction of the called method[[54]](https://en.wikipedia.org/wiki/Exception_handling" \l "cite_note-56) and map lower level exceptions to these types by using [exception chaining](https://en.wikipedia.org/wiki/Exception_chaining).

**Similar mechanisms**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=12" \o "Edit section: Similar mechanisms)]

The roots of checked exceptions go back to the [CLU programming language](https://en.wikipedia.org/wiki/CLU_programming_language)'s notion of exception specification.[[55]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Mindview-57) A function could raise only exceptions listed in its type, but any leaking exceptions from called functions would automatically be turned into the sole runtime exception, failure, instead of resulting in compile-time error.[[7]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-CLU-7) Later, [Modula-3](https://en.wikipedia.org/wiki/Modula-3) had a similar feature.[[56]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-58) These features don't include the compile time checking that is central in the concept of checked exceptions.[[55]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-Mindview-57)

Early versions of the C++ programming language included an optional mechanism similar to checked exceptions, called **exception specifications**. By default any function could throw any exception, but this could be limited by a **throw** clause added to the function signature, that specified which exceptions the function may throw. Exception specifications were not enforced at compile-time. Violations resulted in the global function std::unexpected being called.[[57]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-bjarne-exc-59) An empty exception specification could be given, which indicated that the function will throw no exception. This was not made the default when exception handling was added to the language because it would have required too much modification of existing code, would have impeded interaction with code written in other languages, and would have tempted programmers into writing too many handlers at the local level.[[57]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-bjarne-exc-59) Explicit use of empty exception specifications could, however, allow C++ compilers to perform significant code and stack layout optimizations that are precluded when exception handling may take place in a function.[[35]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-cppeh-36) Some analysts viewed the proper use of exception specifications in C++ as difficult to achieve.[[58]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-60) This use of exception specifications was included in [C++98](https://en.wikipedia.org/wiki/C%2B%2B98) and [C++03](https://en.wikipedia.org/wiki/C%2B%2B03), [deprecated](https://en.wikipedia.org/wiki/Deprecated) in the 2012 C++ language standard ([C++11](https://en.wikipedia.org/wiki/C%2B%2B11)),[[59]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-61) and was removed from the language in [C++17](https://en.wikipedia.org/wiki/C%2B%2B17). A function that will not throw any exceptions can now be denoted by the **noexcept** keyword.

An uncaught exceptions analyzer exists for the [OCaml](https://en.wikipedia.org/wiki/OCaml" \o "OCaml) programming language.[[60]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-62) The tool reports the set of raised exceptions as an extended type signature. But, unlike checked exceptions, the tool does not require any syntactic annotations and is external (i.e. it is possible to compile and run a program without having checked the exceptions).

**Dynamic checking of exceptions**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=13" \o "Edit section: Dynamic checking of exceptions)]

The point of exception handling routines is to ensure that the code can handle error conditions. In order to establish that exception handling routines are sufficiently robust, it is necessary to present the code with a wide spectrum of invalid or unexpected inputs, such as can be created via software [fault injection](https://en.wikipedia.org/wiki/Fault_injection) and [mutation testing](https://en.wikipedia.org/wiki/Mutation_testing) (that is also sometimes referred to as [fuzz testing](https://en.wikipedia.org/wiki/Fuzz_testing)). One of the most difficult types of software for which to write exception handling routines is protocol software, since a robust protocol implementation must be prepared to receive input that does not comply with the relevant specification(s).

In order to ensure that meaningful regression analysis can be conducted throughout a [software development lifecycle process](https://en.wikipedia.org/wiki/Software_development_process), any exception handling testing should be highly automated, and the test cases must be generated in a scientific, repeatable fashion. Several commercially available systems exist that perform such testing.

In runtime engine environments such as [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) or [.NET](https://en.wikipedia.org/wiki/.NET_Framework), there exist tools that attach to the runtime engine and every time that an exception of interest occurs, they record debugging information that existed in memory at the time the exception was thrown ([call stack](https://en.wikipedia.org/wiki/Call_stack) and [heap](https://en.wikipedia.org/wiki/Heap_(data_structure)) values). These tools are called [automated exception handling](https://en.wikipedia.org/wiki/Automated_exception_handling) or error interception tools and provide 'root-cause' information for exceptions.

**Asynchronous exceptions**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=14" \o "Edit section: Asynchronous exceptions)]

**Asynchronous exceptions** are events raised by a separate thread or external process, such as pressing [Ctrl-C](https://en.wikipedia.org/wiki/Control-C) to interrupt a program, receiving a [signal](https://en.wikipedia.org/wiki/Signal_(computing)), or sending a disruptive message such as "stop" or "suspend" from another [thread of execution](https://en.wikipedia.org/wiki/Thread_(computer_science)).[[61]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-63)[[62]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-64) Whereas synchronous exceptions happen at a specific throw statement, asynchronous exceptions can be raised at any time. It follows that asynchronous exception handling can't be optimized out by the compiler, as it cannot prove the absence of asynchronous exceptions. They are also difficult to program with correctly, as asynchronous exceptions must be blocked during cleanup operations to avoid resource leaks.

Programming languages typically avoid or restrict asynchronous exception handling, for example C++ forbids raising exceptions from signal handlers, and Java has deprecated the use of its ThreadDeath exception that was used to allow one thread to stop another one.[[63]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-65) Another feature is a semi-asynchronous mechanism that raises an asynchronous exception only during certain operations of the program. For example Java's Thread.interrupt() only affects the thread when the thread calls an operation that throws InterruptedException.[[64]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-66) The similar POSIX pthread\_cancel API has race conditions which make it impossible to use safely.[[65]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-67)

**Condition systems**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=15" \o "Edit section: Condition systems)]

[Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Dylan](https://en.wikipedia.org/wiki/Dylan_(programming_language)) and [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk) have a [condition system](https://en.wikipedia.org/w/index.php?title=Condition_system&action=edit&redlink=1)[[66]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-68) (see [Common Lisp Condition System](https://en.wikipedia.org/wiki/Common_Lisp#Condition_system)) that encompasses the aforementioned exception handling systems. In those languages or environments the advent of a condition (a "generalisation of an error" according to [Kent Pitman](https://en.wikipedia.org/wiki/Kent_Pitman)) implies a function call, and only late in the exception handler the decision to unwind the stack may be taken.

Conditions are a generalization of exceptions. When a condition arises, an appropriate condition handler is searched for and selected, in stack order, to handle the condition. Conditions that do not represent errors may safely go unhandled entirely; their only purpose may be to propagate hints or warnings toward the user.[[67]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-69)

**Continuable exceptions**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=16" \o "Edit section: Continuable exceptions)]

This is related to the so-called *resumption model* of exception handling, in which some exceptions are said to be *continuable*: it is permitted to return to the expression that signaled an exception, after having taken corrective action in the handler. The condition system is generalized thus: within the handler of a non-serious condition (a.k.a. *continuable exception*), it is possible to jump to predefined restart points (a.k.a. *restarts*) that lie between the signaling expression and the condition handler. Restarts are functions closed over some lexical environment, allowing the programmer to repair this environment before exiting the condition handler completely or unwinding the stack even partially.

An example is the **ENDPAGE** condition in PL/I; the ON unit might write page trailer lines and header lines for the next page, then fall through to resume execution of the interrupted code.

**Restarts separate mechanism from policy**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=17" \o "Edit section: Restarts separate mechanism from policy)]

Condition handling moreover provides a [separation of mechanism from policy](https://en.wikipedia.org/wiki/Separation_of_mechanism_from_policy). Restarts provide various possible mechanisms for recovering from error, but do not select which mechanism is appropriate in a given situation. That is the province of the condition handler, which (since it is located in higher-level code) has access to a broader view.

An example: Suppose there is a library function whose purpose is to parse a single [syslog](https://en.wikipedia.org/wiki/Syslog" \o "Syslog) file entry. What should this function do if the entry is malformed? There is no one right answer, because the same library could be deployed in programs for many different purposes. In an interactive log-file browser, the right thing to do might be to return the entry unparsed, so the user can see it—but in an automated log-summarizing program, the right thing to do might be to supply null values for the unreadable fields, but abort with an error, if too many entries have been malformed.

That is to say, the question can only be answered in terms of the broader goals of the program, which are not known to the general-purpose library function. Nonetheless, exiting with an error message is only rarely the right answer. So instead of simply exiting with an error, the function may *establish restarts* offering various ways to continue—for instance, to skip the log entry, to supply default or null values for the unreadable fields, to ask the user for the missing values, *or* to unwind the stack and abort processing with an error message. The restarts offered constitute the *mechanisms* available for recovering from error; the selection of restart by the condition handler supplies the [*policy*](https://en.wikipedia.org/wiki/Policy).

**Criticism**[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=18" \o "Edit section: Criticism)]

Exception handling is often not handled correctly in software, especially when there are multiple sources of exceptions; [data flow analysis](https://en.wikipedia.org/wiki/Data_flow_analysis) of 5 million lines of Java code found over 1300 exception handling defects.[[25]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-toplas2008-25) Citing multiple prior studies by others (1999–2004) and their own results, Weimer and Necula wrote that a significant problem with exceptions is that they "create hidden control-flow paths that are difficult for programmers to reason about".[[25]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-toplas2008-25): 8:27 "While try-catch-finally is conceptually simple, it has the most complicated execution description in the language specification [Gosling et al. 1996] and requires four levels of nested “if”s in its official English description. In short, it contains a large number of [corner cases](https://en.wikipedia.org/wiki/Corner_cases) that programmers often overlook."[[25]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-toplas2008-25): 8:13–8:14

Exceptions, as unstructured flow, increase the risk of [resource leaks](https://en.wikipedia.org/wiki/Resource_leak) (such as escaping a section locked by a [mutex](https://en.wikipedia.org/wiki/Mutex" \o "Mutex), or one temporarily holding a file open) or inconsistent state. There are various techniques for [resource management](https://en.wikipedia.org/wiki/Resource_management_(computing)) in the presence of exceptions, most commonly combining the [dispose pattern](https://en.wikipedia.org/wiki/Dispose_pattern) with some form of unwind protection (like a finally clause), which automatically releases the resource when control exits a section of code.

[Tony Hoare](https://en.wikipedia.org/wiki/Tony_Hoare) in 1980 described the [Ada programming language](https://en.wikipedia.org/wiki/Ada_(programming_language)" \o "Ada (programming language)) as having "...a plethora of features and notational conventions, many of them unnecessary and some of them, like exception handling, even dangerous. [...] Do not allow this language in its present state to be used in applications where reliability is critical [...]. The next rocket to go astray as a result of a programming language error may not be an exploratory space rocket on a harmless trip to Venus: It may be a nuclear warhead exploding over one of our own cities."[[68]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-70)

The [Go](https://en.wikipedia.org/wiki/Go_(programming_language)#Omissions) developers believe that the try-catch-finally idiom obfuscates [control flow](https://en.wikipedia.org/wiki/Control_flow),[[69]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-71) and introduced the exception-like panic/recover mechanism.[[70]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-72) recover() differs from catch in that it can only be called from within a **defer** code block in a function, so the handler can only do clean-up and change the function's return values, and cannot return control to an arbitrary point within the function.[[71]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-73) The **defer** block itself functions similarly to a finally clause.

Exception handling in UI hierarchies[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=19" \o "Edit section: Exception handling in UI hierarchies)]

Front-end web frameworks, such as [React](https://en.wikipedia.org/wiki/React_(JavaScript_library)) and [Vue](https://en.wikipedia.org/wiki/Vue.js" \o "Vue.js), have introduced error handling mechanisms where errors propagate up the UI component hierarchy, in a way that is analogous to how errors propagate up the call stack in executing code.[[72]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-74)[[73]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-75) Here the error boundary mechanism serves as an analogue to the typical try-catch mechanism. Thus a component can ensure that errors from its child components are caught and handled, and not propagated up to parent components.

For example, in Vue, a component would catch errors by implementing errorCaptured

Vue.component('parent', {

template: '<div><slot></slot></div>',

errorCaptured: (err, vm, info) => alert('An error occurred');

})

Vue.component('child', {

template: '<div>{{ cause\_error() }}</div>'

})

When used like this in markup:

<**parent**>

<**child**></**child**>

</**parent**>

The error produced by the child component is caught and handled by the parent component.[[74]](https://en.wikipedia.org/wiki/Exception_handling#cite_note-76)

See also[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=20" \o "Edit section: See also)]

* [Automated exception handling](https://en.wikipedia.org/wiki/Automated_exception_handling)
* [Exception safety](https://en.wikipedia.org/wiki/Exception_safety)
* [Continuation](https://en.wikipedia.org/wiki/Continuation)
* [Defensive programming](https://en.wikipedia.org/wiki/Defensive_programming)
* [Triple fault](https://en.wikipedia.org/wiki/Triple_fault)
* [Option types](https://en.wikipedia.org/wiki/Option_type) and [Result types](https://en.wikipedia.org/wiki/Result_type), alternative ways of handling errors in [functional programming](https://en.wikipedia.org/wiki/Functional_programming) without exceptions
* [Data validation](https://en.wikipedia.org/wiki/Data_validation)

Notes[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=21" \o "Edit section: Notes)]

* 1. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-28) In, e.g., PL/I, a normal exit from an exception handler unwinds the stack.
  2. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-38) There is "zero [processing] cost" only if no exception is throw (although there will be a memory cost since memory is needed for the lookup table). There is a (potentially significant) cost if an exception is thrown (that is, if throw is executed). Implementing exception handling may also limit the possible [compiler optimizations](https://en.wikipedia.org/wiki/Optimizing_compiler) that may be performed.

References[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=22" \o "Edit section: References)]

* 1. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Cristian_1-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Cristian_1-1) *Cristian, Flaviu (1980). "Exception Handling and Software Fault Tolerance". Proc. 10th Int. Symp. On Fault Tolerant Computing (FTCS-25 reprint ed.) (6): 531–540.*[*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.116.8736*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.116.8736)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/TC.1982.1676035*](https://doi.org/10.1109%2FTC.1982.1676035)*.*[*OCLC*](https://en.wikipedia.org/wiki/OCLC_(identifier))[*1029229019*](https://www.worldcat.org/oclc/1029229019)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*18345469*](https://api.semanticscholar.org/CorpusID:18345469)*.*
  2. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEGoodenough1975b683%E2%80%93684_2-0) [Goodenough 1975b](https://en.wikipedia.org/wiki/Exception_handling" \l "CITEREFGoodenough1975b), pp. 683–684.
  3. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEGoodenough1975b684_3-0) [Goodenough 1975b](https://en.wikipedia.org/wiki/Exception_handling" \l "CITEREFGoodenough1975b), p. 684.
  4. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEBlack198213%E2%80%9315_4-0) [Black 1982](https://en.wikipedia.org/wiki/Exception_handling#CITEREFBlack1982), pp. 13–15.
  5. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Lang_5-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Lang_5-1) *Lang, Jun; Stewart, David B. (March 1998). "A study of the applicability of existing exception-handling techniques to component-based real-time software technology". ACM Transactions on Programming Languages and Systems.****20****(2): 276.*[*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.33.3400*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3400)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/276393.276395*](https://doi.org/10.1145%2F276393.276395)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*18875882*](https://api.semanticscholar.org/CorpusID:18875882)*. Perhaps the most common form of exception-handling method used by software programmers is the "return-code" technique that was popularized as part of C and UNIX.*
  6. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTELevin19775_6-0) [Levin 1977](https://en.wikipedia.org/wiki/Exception_handling#CITEREFLevin1977), p. 5.
  7. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-CLU_7-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-CLU_7-1) *Liskov, B.H.; Snyder, A. (November 1979).*[*"Exception Handling in CLU"*](http://csg.csail.mit.edu/CSGArchives/memos/Memo-155-3.pdf)*(PDF). IEEE Transactions on Software Engineering.****SE-5****(6): 546–558.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/TSE.1979.230191*](https://doi.org/10.1109%2FTSE.1979.230191)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*15506879*](https://api.semanticscholar.org/CorpusID:15506879)*. Retrieved 19 December 2021.*
  8. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTELevin19774_8-0) [Levin 1977](https://en.wikipedia.org/wiki/Exception_handling#CITEREFLevin1977), p. 4.
  9. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Kiniry_9-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Kiniry_9-1) [***c***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Kiniry_9-2) [***d***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Kiniry_9-3) [***e***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Kiniry_9-4) *Kiniry, J. R. (2006). "Exceptions in Java and Eiffel: Two Extremes in Exception Design and Application".*[*Advanced Topics in Exception Handling Techniques*](http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/papers/ExceptionsInEiffelAndJava.pdf)*(PDF). Lecture Notes in Computer Science. Vol. 4119. pp. 288–300.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/11818502\_16*](https://doi.org/10.1007%2F11818502_16)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-3-540-37443-5*](https://en.wikipedia.org/wiki/Special:BookSources/978-3-540-37443-5)*.*
  10. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Smotherman_10-0) *Smotherman, Mark.*[*"Interrupts"*](https://people.cs.clemson.edu/~mark/interrupts.html)*. Retrieved 4 January 2022.*
  11. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-11) *McCarthy, John (12 February 1979).*[*"History of Lisp"*](http://www-formal.stanford.edu/jmc/history/lisp/node1.html)*. www-formal.stanford.edu. Retrieved 13 January 2022.*
  12. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-12) *McCarthy, John; Levin, Michael I.; Abrahams, Paul W.; Edwards, Daniel J.; Hart, Timothy P. (14 July 1961).*[*LISP 1.5 programmer's manual*](http://www.softwarepreservation.org/projects/LISP/book/LISP%201.5%20Programmers%20Manual-1961.07.14.pdf#page=58)*(PDF). Retrieved 13 January 2022.*
  13. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-13) [*"The ON Statement"*](http://www.bitsavers.org/pdf/ibm/360/pli/C28-6571-3_PL_I_Language_Specifications_Jul66.pdf#page=120)*(PDF).*[*IBM System/360 Operating System, PL/I Language Specifications*](http://www.bitsavers.org/pdf/ibm/360/pli/C28-6571-3_PL_I_Language_Specifications_Jul66.pdf)*(PDF). IBM. July 1966. p. 120. C28-6571-3.*
  14. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEGabrielSteele20083_14-0) [Gabriel & Steele 2008](https://en.wikipedia.org/wiki/Exception_handling#CITEREFGabrielSteele2008), p. 3.
  15. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEWhite1979194_15-0) [White 1979](https://en.wikipedia.org/wiki/Exception_handling#CITEREFWhite1979), p. 194.
  16. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEStroustrup1994392_16-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEStroustrup1994392_16-1) [Stroustrup 1994](https://en.wikipedia.org/wiki/Exception_handling" \l "CITEREFStroustrup1994), p. 392.
  17. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-17) *Hyde, Randall.*[*"Art of Assembly: Chapter Seventeen"*](https://www.plantation-productions.com/Webster/www.artofasm.com/DOS/ch17/CH17-1.html#HEADING1-0)*. www.plantation-productions.com. Retrieved 22 December 2021.*
  18. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Xiaoye_Li_and_James_Demmel_1994_983%E2%80%93992_18-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Xiaoye_Li_and_James_Demmel_1994_983%E2%80%93992_18-1) *[Xiaoye Li](https://en.wikipedia.org/wiki/Sherry_Li" \o "Sherry Li); James Demmel (1994). "Faster Numerical Algorithms via Exception Handling, IEEE Transactions on Computers, 43(8)": 983–992.*
  19. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-grail_19-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-grail_19-1) *W.Kahan (July 5, 2005).*[*"A Demonstration of Presubstitution for ∞/∞"*](http://www.cs.berkeley.edu/~wkahan/Grail.pdf)*(PDF).*[*Archived*](https://web.archive.org/web/20120310130507/http:/www.cs.berkeley.edu/~wkahan/Grail.pdf)*(PDF) from the original on March 10, 2012.*
  20. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-20) *Hauser, John R. (March 1996). "Handling floating-point exceptions in numeric programs". ACM Transactions on Programming Languages and Systems.****18****(2): 139–174.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/227699.227701*](https://doi.org/10.1145%2F227699.227701)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*9820157*](https://api.semanticscholar.org/CorpusID:9820157)*.*
  21. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-21) [*"Built-in Exceptions — Python 3.10.4 documentation"*](https://docs.python.org/3/library/exceptions.html#StopIteration)*. docs.python.org. Retrieved 17 May 2022.*
  22. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-22) [*"Stroustrup: C++ Style and Technique FAQ"*](http://www.stroustrup.com/bs_faq2.html#exceptions-what-not)*. www.stroustrup.com.*[*Archived*](https://web.archive.org/web/20180202012417/http:/www.stroustrup.com/bs_faq2.html#exceptions-what-not)*from the original on 2 February 2018. Retrieved 5 May 2018.*
  23. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-EffectiveJava_23-0) *Bloch, Joshua (2008).*[*"Item 57: Use exceptions only for exceptional situations"*](https://archive.org/details/effectivejava00bloc_0/page/241)*. Effective Java (Second ed.). Addison-Wesley. p.*[*241*](https://archive.org/details/effectivejava00bloc_0/page/241)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-321-35668-0*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-321-35668-0)*.*
  24. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-24) *Stroustrup, Bjarne.*[*"C++ Style and Technique FAQ"*](https://www.stroustrup.com/bs_faq2.html#finally)*. www.stroustrup.com. Retrieved 12 January 2022.*
  25. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-toplas2008_25-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-toplas2008_25-1) [***c***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-toplas2008_25-2) [***d***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-toplas2008_25-3) *Weimer, W; Necula, G.C. (2008).*[*"Exceptional Situations and Program Reliability"*](http://www.cs.virginia.edu/~weimer/p/weimer-toplas2008.pdf)*(PDF). ACM Transactions on Programming Languages and Systems. Vol. 30, no. 2.*[*Archived*](https://web.archive.org/web/20150923211739/http:/www.cs.virginia.edu/~weimer/p/weimer-toplas2008.pdf)*(PDF) from the original on 2015-09-23.*
  26. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-26) *Roberts, Eric S. (21 March 1989).*[*"Implementing Exceptions in C"*](http://bitsavers.informatik.uni-stuttgart.de/pdf/dec/tech_reports/SRC-RR-40.pdf)*(PDF).*[*DEC Systems Research Center*](https://en.wikipedia.org/wiki/DEC_Systems_Research_Center)*. SRC-RR-40. Retrieved 4 January 2022.*
  27. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-27) *Christiansen, Tom; Torkington, Nathan (2003). "10.12. Handling Exceptions".*[*Perl cookbook*](https://docstore.mik.ua/orelly/perl4/cook/ch10_13.htm)*(2nd ed.). Beijing: O'Reilly.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-596-00313-7*](https://en.wikipedia.org/wiki/Special:BookSources/0-596-00313-7)*.*
  28. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-FOOTNOTEStroustrup199416.6_Exception_Handling:_Resumption_vs._Termination,_pp._390%E2%80%93393_29-0) [Stroustrup 1994](https://en.wikipedia.org/wiki/Exception_handling" \l "CITEREFStroustrup1994), 16.6 Exception Handling: Resumption vs. Termination, pp. 390–393.
  29. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-30) [*"R: Condition Handling and Recovery"*](https://search.r-project.org/R/refmans/base/html/conditions.html)*. search.r-project.org. Retrieved 2022-12-05.*
  30. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-31) [Scott Meyers](https://en.wikipedia.org/wiki/Scott_Meyers), [The Most Important C++ Software...Ever](http://www.artima.com/cppsource/top_cpp_software.html) [Archived](https://web.archive.org/web/20110428221802/http:/www.artima.com/cppsource/top_cpp_software.html) 2011-04-28 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine), 2006
  31. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-32) D. Cameron, P. Faust, D. Lenkov, M. Mehta, "A portable implementation of C++ exception handling", *Proceedings of the C++ Conference* (August 1992) [USENIX](https://en.wikipedia.org/wiki/USENIX).
  32. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-33) *Peter Kleissner (February 14, 2009).*[*"Windows Exception Handling - Peter Kleissner"*](https://web.archive.org/web/20131014204335/http:/stoned-vienna.com/html/index.php?page=windows-exception-handling)*. Archived from*[*the original*](http://stoned-vienna.com/html/index.php?page=windows-exception-handling)*on October 14, 2013. Retrieved 2009-11-21.*, *Compiler based Structured Exception Handling* section
  33. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-34) Graham Hutton, Joel Wright, "[Compiling Exceptions Correctly](http://www.cs.nott.ac.uk/~gmh/exceptions.pdf) [Archived](https://web.archive.org/web/20140911173720/http:/www.cs.nott.ac.uk/~gmh/exceptions.pdf) 2014-09-11 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)". *Proceedings of the 7th International Conference on Mathematics of Program Construction*, 2004.
  34. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-35) *Lajoie, Josée (March–April 1994). "Exception handling – Supporting the runtime mechanism". C++ Report.****6****(3).*
  35. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-cppeh_36-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-cppeh_36-1) *Schilling, Jonathan L. (August 1998). "Optimizing away C++ exception handling".*[*SIGPLAN Notices*](https://en.wikipedia.org/wiki/SIGPLAN_Notices)*.****33****(8): 40–47.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/286385.286390*](https://doi.org/10.1145%2F286385.286390)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*1522664*](https://api.semanticscholar.org/CorpusID:1522664)*.*
  36. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-MiscrosoftDocsExceptions_37-0) [*"Modern C++ best practices for exceptions and error handling"*](https://docs.microsoft.com/en-us/cpp/cpp/errors-and-exception-handling-modern-cpp)*. Microsoft. 8 March 2021. Retrieved 21 March 2022.*
  37. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-StroustrupExceptions2019_39-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-StroustrupExceptions2019_39-1) *[Stroustrup, Bjarne](https://en.wikipedia.org/wiki/Bjarne_Stroustrup" \o "Bjarne Stroustrup) (18 November 2019).*[*"C++ exceptions and alternatives"*](http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1947r0.pdf)*(PDF). Retrieved 23 March 2022.*
  38. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-40) M. Hof, H. Mössenböck, P. Pirkelbauer, "[Zero-Overhead Exception Handling Using Metaprogramming](http://www.ssw.uni-linz.ac.at/Research/Papers/Hof97b.html) [Archived](https://web.archive.org/web/20160303180327/http:/www.ssw.uni-linz.ac.at/Research/Papers/Hof97b.html) 2016-03-03 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)", *Proceedings SOFSEM'97*, November 1997, *Lecture Notes in Computer Science 1338*, pp. 423-431.
  39. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-41) All Exceptions Are Handled, Jim Wilcox, [*"All Exceptions Are Handled"*](https://www.granitestateusersgroups.net/2008/02/21/all-exceptions-are-handled/)*.*
  40. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-cocoa_42-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-cocoa_42-1) [***c***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-cocoa_42-2) *Mac Developer Library*, "[Uncaught Exceptions](https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Exceptions/Concepts/UncaughtExceptions.html) [Archived](https://web.archive.org/web/20160304111516/https:/developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Exceptions/Concepts/UncaughtExceptions.html) 2016-03-04 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)"
  41. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-43) *MSDN*, [AppDomain.UnhandledException Event](https://msdn.microsoft.com/en-us/library/system.appdomain.unhandledexception(v=vs.110).aspx) [Archived](https://web.archive.org/web/20160304131615/https:/msdn.microsoft.com/en-us/library/system.appdomain.unhandledexception(v=vs.110).aspx) 2016-03-04 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)
  42. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-44) *The Python Tutorial*, "[8. Errors and Exceptions](https://docs.python.org/2/tutorial/errors.html) [Archived](https://web.archive.org/web/20150901043830/https:/docs.python.org/2/tutorial/errors.html) 2015-09-01 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)"
  43. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-45) [*"Java Practices -> Provide an uncaught exception handler"*](http://www.javapractices.com/topic/TopicAction.do?Id=229)*. www.javapractices.com.*[*Archived*](https://web.archive.org/web/20160909002524/http:/www.javapractices.com/topic/TopicAction.do?Id=229)*from the original on 9 September 2016. Retrieved 5 May 2018.*
  44. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-pmotw_46-0) PyMOTW (Python Module Of The Week), "[Exception Handling](https://pymotw.com/2/sys/exceptions.html) [Archived](https://web.archive.org/web/20150915032624/https:/pymotw.com/2/sys/exceptions.html) 2015-09-15 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)"
  45. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-47) [*"Google Answers: The origin of checked exceptions"*](http://answers.google.com/answers/threadview?id=26101)*.*[*Archived*](http://archive.wikiwix.com/cache/20110806090553/http:/answers.google.com/answers/threadview?id=26101)*from the original on 2011-08-06. Retrieved 2011-12-15.*
  46. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-48) Java Language Specification, chapter 11.2. <http://java.sun.com/docs/books/jls/third_edition/html/exceptions.html#11.2> [Archived](https://web.archive.org/web/20061208042454/http:/java.sun.com/docs/books/jls/third_edition/html/exceptions.html) 2006-12-08 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine)
  47. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-49) *Mössenböck, Hanspeter (2002-03-25).*[*"Advanced C#: Variable Number of Parameters"*](http://ssw.jku.at/Teaching/Lectures/CSharp/Tutorial/Part2.pdf)*(PDF). Institut für Systemsoftware, Johannes Kepler Universität Linz, Fachbereich Informatik. p. 32.*[*Archived*](https://web.archive.org/web/20110920080842/http:/ssw.jku.at/Teaching/Lectures/CSharp/Tutorial/Part2.pdf)*(PDF) from the original on 2011-09-20. Retrieved 2011-08-05.*
  48. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Eckel_50-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Eckel_50-1) [***c***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Eckel_50-2) *Eckel, Bruce (2006). Thinking in Java (4th ed.). Upper Saddle River, NJ: Prentice Hall. pp. 347–348.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-13-187248-6*](https://en.wikipedia.org/wiki/Special:BookSources/0-13-187248-6)*.*
  49. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-51) *Gunnerson, Eric (9 November 2000).*[*"C# and exception specifications"*](https://web.archive.org/web/20060101083304/http:/discuss.develop.com/archives/wa.exe?A2=ind0011A&L=DOTNET&P=R32820)*. Archived from*[*the original*](http://discuss.develop.com/archives/wa.exe?A2=ind0011A&L=DOTNET&P=R32820)*on 1 January 2006.*
  50. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Trouble_52-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Trouble_52-1) *Bill Venners;*[*Bruce Eckel*](https://en.wikipedia.org/wiki/Bruce_Eckel)*(August 18, 2003).*[*"The Trouble with Checked Exceptions: A Conversation with Anders Hejlsberg, Part II"*](https://www.artima.com/articles/the-trouble-with-checked-exceptions)*. Retrieved 4 January 2022.*
  51. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-53) *Juneau, Josh (31 May 2017).*[*Java 9 Recipes: A Problem-Solution Approach*](https://books.google.com/books?id=TSYmDwAAQBAJ&pg=PA226)*. Apress. p. 226.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-4842-1976-8*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-4842-1976-8)*.*
  52. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-54) [*"Advantages of Exceptions (The Java™ Tutorials : Essential Classes : Exceptions)"*](http://download.oracle.com/javase/tutorial/essential/exceptions/advantages.html)*. Download.oracle.com.*[*Archived*](https://web.archive.org/web/20111026121217/http:/download.oracle.com/javase/tutorial/essential/exceptions/advantages.html)*from the original on 2011-10-26. Retrieved 2011-12-15.*
  53. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-55) [*"Unchecked Exceptions – The Controversy (The Java™ Tutorials : Essential Classes : Exceptions)"*](http://download.oracle.com/javase/tutorial/essential/exceptions/runtime.html)*. Download.oracle.com.*[*Archived*](https://web.archive.org/web/20111117042228/http:/download.oracle.com/javase/tutorial/essential/exceptions/runtime.html)*from the original on 2011-11-17. Retrieved 2011-12-15.*
  54. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-56) Bloch 2001:178 *Bloch, Joshua (2001).*[*Effective Java Programming Language Guide*](https://archive.org/details/effectivejavapro00bloc)*. Addison-Wesley Professional.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-201-31005-4*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-201-31005-4)*.*
  55. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Mindview_57-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-Mindview_57-1) [*"Bruce Eckel's MindView, Inc: Does Java need Checked Exceptions?"*](https://web.archive.org/web/20020405175011/http:/www.mindview.net/Etc/Discussions/CheckedExceptions)*. Mindview.net. Archived from*[*the original*](http://www.mindview.net/Etc/Discussions/CheckedExceptions)*on 2002-04-05. Retrieved 2011-12-15.*
  56. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-58) [*"Modula-3 - Procedure Types"*](http://www1.cs.columbia.edu/graphics/modula3/tutorial/www/m3_23.html#SEC23)*. .cs.columbia.edu. 1995-03-08.*[*Archived*](https://web.archive.org/web/20080509143753/http:/www1.cs.columbia.edu/graphics/modula3/tutorial/www/m3_23.html#SEC23)*from the original on 2008-05-09. Retrieved 2011-12-15.*
  57. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-bjarne-exc_59-0) [***b***](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-bjarne-exc_59-1) [Bjarne Stroustrup](https://en.wikipedia.org/wiki/Bjarne_Stroustrup" \o "Bjarne Stroustrup), [*The C++ Programming Language*](https://en.wikipedia.org/wiki/The_C%2B%2B_Programming_Language) Third Edition, [Addison Wesley](https://en.wikipedia.org/wiki/Addison_Wesley), 1997. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-201-88954-4](https://en.wikipedia.org/wiki/Special:BookSources/0-201-88954-4). pp. 375-380.
  58. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-60) *Reeves, J.W. (July 1996). "Ten Guidelines for Exception Specifications". C++ Report.****8****(7).*
  59. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-61) [*Sutter, Herb*](https://en.wikipedia.org/wiki/Herb_Sutter)*(3 March 2010).*[*"Trip Report: March 2010 ISO C++ Standards Meeting"*](http://herbsutter.com/2010/03/13/trip-report-march-2010-iso-c-standards-meeting/)*.*[*Archived*](https://web.archive.org/web/20100323082634/http:/herbsutter.com/2010/03/13/trip-report-march-2010-iso-c-standards-meeting/)*from the original on 23 March 2010. Retrieved 24 March 2010.*
  60. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-62) [*"OcamlExc - An uncaught exceptions analyzer for Objective Caml"*](http://caml.inria.fr/pub/old_caml_site/ocamlexc/ocamlexc.htm)*. Caml.inria.fr.*[*Archived*](http://archive.wikiwix.com/cache/20110806090555/http:/caml.inria.fr/pub/old_caml_site/ocamlexc/ocamlexc.htm)*from the original on 2011-08-06. Retrieved 2011-12-15.*
  61. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-63) [*"Asynchronous Exceptions in Haskell - Marlow, Jones, Moran (ResearchIndex)"*](http://citeseer.ist.psu.edu/415348.html)*. Citeseer.ist.psu.edu.*[*Archived*](http://archive.wikiwix.com/cache/20110223164151/http:/citeseer.ist.psu.edu/415348.html)*from the original on 2011-02-23. Retrieved 2011-12-15.*
  62. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-64) *Freund, Stephen N.; Mitchell, Mark P.*[*"Safe Asynchronous Exceptions For Python"*](http://www.cs.williams.edu/~freund/papers/python.pdf)*(PDF). Retrieved 4 January 2022.*
  63. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-65) [*"Java Thread Primitive Deprecation"*](http://java.sun.com/j2se/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html)*. Java.sun.com.*[*Archived*](https://web.archive.org/web/20090426200153/http:/java.sun.com/j2se/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html)*from the original on 2009-04-26. Retrieved 2011-12-15.*
  64. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-66) [*"Interrupts (The Java™ Tutorials > Essential Java Classes > Concurrency)"*](https://docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html)*. docs.oracle.com. Retrieved 5 January 2022.*
  65. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-67) *Felker, Rich.*[*"Thread cancellation and resource leaks"*](http://ewontfix.com/2/)*. ewontfix.com. Retrieved 5 January 2022.*
  66. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-68) *What Conditions (Exceptions) are Really About (2008-03-24).*[*"What Conditions (Exceptions) are Really About"*](https://web.archive.org/web/20130201124021/http:/danweinreb.org/blog/what-conditions-exceptions-are-really-about)*. Danweinreb.org. Archived from*[*the original*](http://danweinreb.org/blog/what-conditions-exceptions-are-really-about)*on February 1, 2013. Retrieved 2014-09-18.*
  67. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-69) [*"Condition System Concepts"*](https://web.archive.org/web/20070628213221/http:/www.franz.com/support/documentation/6.2/ansicl/section/conditio.htm)*. Franz.com. 2009-07-21. Archived from*[*the original*](http://www.franz.com/support/documentation/6.2/ansicl/section/conditio.htm)*on 2007-06-28. Retrieved 2011-12-15.*
  68. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-70) C.A.R. Hoare. "The Emperor's Old Clothes". 1980 Turing Award Lecture
  69. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-71) [*"Frequently Asked Questions"*](https://golang.org/doc/faq#exceptions)*.*[*Archived*](https://web.archive.org/web/20170503205801/https:/golang.org/doc/faq#exceptions)*from the original on 2017-05-03. Retrieved 2017-04-27. We believe that coupling exceptions to a control structure, as in the try-catch-finally idiom, results in convoluted code. It also tends to encourage programmers to label too many ordinary errors, such as failing to open a file, as exceptional.*
  70. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-72) [Panic And Recover](https://code.google.com/p/go-wiki/wiki/PanicAndRecover) [Archived](https://web.archive.org/web/20131024144034/https:/code.google.com/p/go-wiki/wiki/PanicAndRecover) 2013-10-24 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine), Go wiki
  71. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-73) *Bendersky, Eli (8 August 2018).*[*"On the uses and misuses of panics in Go"*](https://eli.thegreenplace.net/2018/on-the-uses-and-misuses-of-panics-in-go/)*. Eli Bendersky's website. Retrieved 5 January 2022. The specific limitation is that recover can only be called in a defer code block, which cannot return control to an arbitrary point, but can only do clean-ups and tweak the function's return values.*
  72. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-74) [*"Error Boundaries"*](https://reactjs.org/docs/error-boundaries.html)*. React. Retrieved 2018-12-10.*
  73. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-75) [*"Vue.js API"*](https://vuejs.org/v2/api/#errorCaptured)*. Vue.js. Retrieved 2018-12-10.*
  74. [**^**](https://en.wikipedia.org/wiki/Exception_handling#cite_ref-76) [*"Error handling with Vue.js"*](https://catchjs.com/Docs/Vue)*. CatchJS. Retrieved 2018-12-10.*
* *Black, Andrew P. (January 1982).*[*Exception handling: The case against*](http://web.cecs.pdx.edu/~black/publications/Black%20D.%20Phil%20Thesis.pdf)*(PDF) (PhD). University of Oxford. [CiteSeerX](https://en.wikipedia.org/wiki/CiteSeerX_(identifier)" \o "CiteSeerX (identifier))*[*10.1.1.94.5554*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.5554)*.*[*OCLC*](https://en.wikipedia.org/wiki/OCLC_(identifier))[*123311492*](https://www.worldcat.org/oclc/123311492)*.*
* [*Gabriel, Richard P.*](https://en.wikipedia.org/wiki/Richard_P._Gabriel)*;*[*Steele, Guy L.*](https://en.wikipedia.org/wiki/Guy_L._Steele,_Jr.)*(2008).*[*A Pattern of Language Evolution*](http://www.dreamsongs.com/Files/PatternOfLanguageEvolution.pdf)*(PDF). LISP50: Celebrating the 50th Anniversary of Lisp. pp. 1–10.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1529966.1529967*](https://doi.org/10.1145%2F1529966.1529967)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-60558-383-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-60558-383-9)*.*
* *Goodenough, John B. (1975a). Structured exception handling. Proceedings of the 2nd ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL '75. pp. 204–224.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/512976.512997*](https://doi.org/10.1145%2F512976.512997)*.*
* *Goodenough, John B. (1975).*[*"Exception handling: Issues and a proposed notation"*](http://www.cs.colorado.edu/~bec/courses/csci5535-s09/reading/goodenough-exceptions.pdf)*(PDF). Communications of the ACM.****18****(12): 683–696.*[*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.122.7791*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.122.7791)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/361227.361230*](https://doi.org/10.1145%2F361227.361230)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*12935051*](https://api.semanticscholar.org/CorpusID:12935051)*.*
* *Levin, Roy (June 1977).*[*Program Structures for Exceptional Condition Handling*](https://apps.dtic.mil/sti/pdfs/ADA043449.pdf)*(PDF) (PhD). Carnegie-Mellon University.*[*DTIC*](https://en.wikipedia.org/wiki/DTIC_(identifier))[*ADA043449*](https://apps.dtic.mil/sti/citations/ADA043449)*.*[*Archived*](https://web.archive.org/web/20211222205616/https:/apps.dtic.mil/sti/pdfs/ADA043449.pdf)*(PDF) from the original on December 22, 2021.*
* *Stroustrup, Bjarne (1994). The design and evolution of C++ (1st ed.). Reading, Mass.: Addison-Wesley.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-201-54330-3*](https://en.wikipedia.org/wiki/Special:BookSources/0-201-54330-3)*.*
* *White, Jon L (May 1979).*[*NIL - A Perspective*](http://www.softwarepreservation.org/projects/LISP/MIT/White-NIL_A_Perspective-1979.pdf)*(PDF). Proceedings of the 1979 Macsyma User's Conference.*

External links[[edit](https://en.wikipedia.org/w/index.php?title=Exception_handling&action=edit&section=23)]

* [A Crash Course on the Depths of Win32 Structured Exception Handling](https://web.archive.org/web/20150929000043/http:/www.microsoft.com/msj/0197/exception/exception.aspx) by [Matt Pietrek](https://en.wikipedia.org/wiki/Matt_Pietrek) - Microsoft Systems Journal (1997)
* Article "[C++ Exception Handling](https://db.usenix.org/events/wiess2000/full_papers/dinechin/dinechin.pdf)" by Christophe de Dinechin
* Article "[Exceptional practices](http://java.sun.com/developer/technicalArticles/Programming/exceptions2/index.html)" by Brian Goetz
* Article "[Object Oriented Exception Handling in Perl](http://perl.com/pub/a/2002/11/14/exception.html)" by Arun Udaya Shankar
* Article "[Programming with Exceptions in C++](http://oreillynet.com/pub/a/network/2003/05/05/cpluspocketref.html)" by Kyle Loudon
* Article "[Unchecked Exceptions - The Controversy](http://java.sun.com/docs/books/tutorial/essential/exceptions/runtime.html)"
* Conference slides [Floating-Point Exception-Handling policies (pdf p. 46)](http://www.eecs.berkeley.edu/~wkahan/Boulder.pdf)by William Kahan
* [Descriptions from Portland Pattern Repository](http://c2.com/cgi/wiki?CategoryException)
* [Does Java Need Checked Exceptions?](https://web.archive.org/web/20020405175011/http:/www.mindview.net/Etc/Discussions/CheckedExceptions)