Module java.base
Package java.lang

Class Throwable

java.lang.Object
    java.lang.Throwable

All Implemented Interfaces:

`Serializable`

Direct Known Subclasses:

`Error`, `Exception`

```
public class Throwable
extends Object
implements Serializable
```

The `Throwable` class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java `throw` statement. Similarly, only this class or one of its subclasses can be the argument type in a `catch` clause. For the purposes of compile-time checking of exceptions, `Throwable` and any subclass of `Throwable` that is not also a subclass of either `RuntimeException` or `Error` are regarded as checked exceptions.

Instances of two subclasses, `Error` and `Exception`, are conventionally used to indicate that exceptional situations have occurred. Typically, these instances are freshly created in the context of the exceptional situation so as to include relevant information (such as stack trace data).

A throwable contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error. Over time, a throwable can suppress other throwables from being propagated. Finally, the throwable can also contain a *cause*: another throwable that caused this throwable to be constructed. The recording of this causal information is referred to as the *chained exception* facility, as the cause can, itself, have a cause, and so on, leading to a "chain" of exceptions, each caused by another.

One reason that a throwable may have a cause is that the class that throws it is built atop a lower layered abstraction, and an operation on the upper layer fails due to a failure in the lower layer. It would be bad design to let the throwable thrown by the lower layer propagate outward, as it is generally unrelated to the abstraction provided by the upper layer. Further, doing so would tie the API of the upper layer to the details of its implementation, assuming the lower layer's exception was a checked exception. Throwing a "wrapped exception" (i.e., an exception containing a cause) allows the upper layer to communicate the details of the failure to its caller without incurring either of these shortcomings. It preserves the flexibility to change the implementation of the upper layer without changing its API (in particular, the set of exceptions thrown by its methods).

A second reason that a throwable may have a cause is that the method that throws it must conform to a general-purpose interface that does not permit the method to throw the cause directly. For example, suppose a persistent collection conforms to the `Collection` interface, and that its persistence is implemented atop `java.io`. Suppose the internals of the `add` method can throw an `IOException`. The implementation can communicate the details of the `IOException` to its caller while conforming to the `Collection` interface by wrapping the `IOException` in an appropriate unchecked exception. (The specification for the persistent collection should indicate that it is capable of throwing such exceptions.)

A cause can be associated with a throwable in two ways: via a constructor that takes the cause as an argument, or via the `initCause(Throwable)` method. New throwable classes that wish to allow causes to be associated with them should provide constructors that take a cause and delegate (perhaps indirectly) to one of the `Throwable` constructors that takes a cause. Because the `initCause` method is public, it allows a cause to be associated with any throwable, even a "legacy throwable" whose implementation predates the addition of the exception chaining mechanism to `Throwable`.

By convention, class `Throwable` and its subclasses have two constructors, one that takes no arguments and one that takes a `String` argument that can be used to produce a detail message. Further, those subclasses that might likely have a cause associated with them should have two more constructors, one that takes a `Throwable` (the cause), and one that takes a `String` (the detail message) and a `Throwable` (the cause).

See *Java Language Specification*:

11.2 Compile-Time Checking of Exceptions

Since:

1.0

See Also:

Serialized Form

Constructor Summary

Constructors

Modifier	Constructor	Description
	`Throwable()`	Constructs a new throwable with `null` as its detail message.
	`Throwable(String message)`	Constructs a new throwable with the specified detail message.

	`Throwable(String message, Throwable cause)`	Constructs a new throwable with the specified detail message and cause.
`protected`	`Throwable(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace)`	Constructs a new throwable with the specified detail message, cause, suppression enabled or disabled, and writable stack trace enabled or disabled.
	`Throwable(Throwable cause)`	Constructs a new throwable with the specified cause and a detail message of `(cause==null ? null : cause.toString())` (which typically contains the class and detail message of `cause`).

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`final void`	`addSuppressed(Throwable exception)`	Appends the specified exception to the exceptions that were suppressed in order to deliver this exception.
`Throwable`	`fillInStackTrace()`	Fills in the execution stack trace.
`Throwable`	`getCause()`	Returns the cause of this throwable or `null` if the cause is nonexistent or unknown.
`String`	`getLocalizedMessage()`	Creates a localized description of this throwable.
`String`	`getMessage()`	Returns the detail message string of this throwable.
`StackTraceElement[]`	`getStackTrace()`	Provides programmatic access to the stack trace information printed by `printStackTrace()`.
`final Throwable[]`	`getSuppressed()`	Returns an array containing all of the exceptions that were suppressed, typically by the `try`-with-resources statement, in order to deliver this exception.
`Throwable`	`initCause(Throwable cause)`	Initializes the *cause* of this throwable to the specified value.
`void`	`printStackTrace()`	Prints this throwable and its backtrace to the standard error stream.
`void`	`printStackTrace(PrintStream s)`	Prints this throwable and its backtrace to the specified print stream.
`void`	`printStackTrace(PrintWriter s)`	Prints this throwable and its backtrace to the specified print writer.
`void`	`setStackTrace(StackTraceElement[] stackTrace)`	Sets the stack trace elements that will be returned by `getStackTrace()` and printed by `printStackTrace()` and related methods.
`String`	`toString()`	Returns a short description of this throwable.

Methods declared in class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Constructor Details

Throwable

`public Throwable()`

Constructs a new throwable with `null` as its detail message. The cause is not initialized, and may subsequently be initialized by a call to `initCause(java.lang.Throwable)`.

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Throwable

`public Throwable(String message)`

Constructs a new throwable with the specified detail message. The cause is not initialized, and may subsequently be initialized by a call to `initCause(java.lang.Throwable)`.

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Parameters:

`message` - the detail message. The detail message is saved for later retrieval by the `getMessage()` method.

Throwable

```
public Throwable(String message,
                 Throwable cause)
```

Constructs a new throwable with the specified detail message and cause.

Note that the detail message associated with `cause` is *not* automatically incorporated in this throwable's detail message.

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Parameters:

`message` - the detail message (which is saved for later retrieval by the `getMessage()` method).

`cause` - the cause (which is saved for later retrieval by the `getCause()` method). (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

Since:

1.4

Throwable

```
public Throwable(Throwable cause)
```

Constructs a new throwable with the specified cause and a detail message of `(cause==null ? null : cause.toString())` (which typically contains the class and detail message of `cause`). This constructor is useful for throwables that are little more than wrappers for other throwables (for example, `PrivilegedActionException`).

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Parameters:

`cause` - the cause (which is saved for later retrieval by the `getCause()` method). (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

Since:

1.4

Throwable

```
protected Throwable(String message,
                    Throwable cause,
                    boolean enableSuppression,
                    boolean writableStackTrace)
```

Constructs a new throwable with the specified detail message, cause, suppression enabled or disabled, and writable stack trace enabled or disabled. If suppression is disabled, `getSuppressed()` for this object will return a zero-length array and calls to `addSuppressed(java.lang.Throwable)` that would otherwise append an exception to the suppressed list will have no effect. If the writable stack trace is false, this constructor will not call `fillInStackTrace()`, a `null` will be written to the `stackTrace` field, and subsequent calls to `fillInStackTrace` and `setStackTrace(StackTraceElement[])` will not set the stack trace. If the writable stack trace is false, `getStackTrace()` will return a zero length array.

Note that the other constructors of `Throwable` treat suppression as being enabled and the stack trace as being writable. Subclasses of `Throwable` should document any conditions under which suppression is disabled and document conditions under which the stack trace is not writable. Disabling of suppression should only occur in exceptional circumstances where special requirements exist, such as a virtual machine reusing exception objects under low-memory situations. Circumstances where a given exception object is repeatedly caught and rethrown, such as to implement control flow between two sub-systems, is another situation where immutable throwable objects would be appropriate.

Parameters:

`message` - the detail message.

`cause` - the cause. (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

`enableSuppression` - whether or not suppression is enabled

`writableStackTrace` - whether or not the stack trace is writable

Since:

1.7

See Also:

`OutOfMemoryError`, `NullPointerException`, `ArithmeticException`

Method Details

getMessage

`public String getMessage()`

Returns the detail message string of this throwable.

Returns:

the detail message string of this `Throwable` instance (which may be `null`).

getLocalizedMessage

`public String getLocalizedMessage()`

Creates a localized description of this throwable. Subclasses may override this method in order to produce a locale-specific message. For subclasses that do not override this method, the default implementation returns the same result as `getMessage()`.

Returns:

The localized description of this throwable.

Since:

1.1

getCause

`public Throwable getCause()`

Returns the cause of this throwable or `null` if the cause is nonexistent or unknown. (The cause is the throwable that caused this throwable to get thrown.)

This implementation returns the cause that was supplied via one of the constructors requiring a `Throwable`, or that was set after creation with the `initCause(Throwable)` method. While it is typically unnecessary to override this method, a subclass can override it to return a cause set by some other means. This is appropriate for a "legacy chained throwable" that predates the addition of chained exceptions to `Throwable`. Note that it is *not* necessary to override any of the `PrintStackTrace` methods, all of which invoke the `getCause` method to determine the cause of a throwable.

Returns:

the cause of this throwable or `null` if the cause is nonexistent or unknown.

Since:

1.4

initCause

`public Throwable initCause(Throwable cause)`

Initializes the *cause* of this throwable to the specified value. (The cause is the throwable that caused this throwable to get thrown.)

This method can be called at most once. It is generally called from within the constructor, or immediately after creating the throwable. If this throwable was created with `Throwable(Throwable)` or `Throwable(String,Throwable)`, this method cannot be called even once.

An example of using this method on a legacy throwable type without other support for setting the cause is:

```
 try {
     lowLevelOp();
 } catch (LowLevelException le) {
     throw (HighLevelException)
           new HighLevelException().initCause(le); // Legacy constructor
 }
```

Parameters:

`cause` - the cause (which is saved for later retrieval by the `getCause()` method). (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

Returns:

a reference to this `Throwable` instance.

Throws:

`IllegalArgumentException` - if `cause` is this throwable. (A throwable cannot be its own cause.)

`IllegalStateException` - if this throwable was created with `Throwable(Throwable)` or `Throwable(String,Throwable)`, or this method has already been called on this throwable.

Since:

1.4

toString

```
public String toString()
```

Returns a short description of this throwable. The result is the concatenation of:

- the name of the class of this object
- ": " (a colon and a space)
- the result of invoking this object's `getLocalizedMessage()` method

If `getLocalizedMessage` returns `null`, then just the class name is returned.

Overrides:

`toString` in class `Object`

Returns:

a string representation of this throwable.

printStackTrace

```
public void printStackTrace()
```

Prints this throwable and its backtrace to the standard error stream. This method prints a stack trace for this `Throwable` object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object. Remaining lines represent data previously recorded by the method `fillInStackTrace()`. The format of this information depends on the implementation, but the following example may be regarded as typical:

```
 java.lang.NullPointerException
         at MyClass.mash(MyClass.java:9)
         at MyClass.crunch(MyClass.java:6)
         at MyClass.main(MyClass.java:3)
```

This example was produced by running the program:

```
class MyClass {
    public static void main(String[] args) {
        crunch(null);
    }
    static void crunch(int[] a) {
        mash(a);
    }
    static void mash(int[] b) {
        System.out.println(b[0]);
    }
}
```

The backtrace for a throwable with an initialized, non-null cause should generally include the backtrace for the cause. The format of this information depends on the implementation, but the following example may be regarded as typical:

```
HighLevelException: MidLevelException: LowLevelException
        at Junk.a(Junk.java:13)
        at Junk.main(Junk.java:4)
Caused by: MidLevelException: LowLevelException
        at Junk.c(Junk.java:23)
        at Junk.b(Junk.java:17)
        at Junk.a(Junk.java:11)
        ... 1 more
Caused by: LowLevelException
        at Junk.e(Junk.java:30)
        at Junk.d(Junk.java:27)
        at Junk.c(Junk.java:21)
        ... 3 more
```

Note the presence of lines containing the characters `"..."`. These lines indicate that the remainder of the stack trace for this exception matches the indicated number of frames from the bottom of the stack trace of the exception that was caused by this exception (the "enclosing" exception). This shorthand can greatly reduce the length of the output in the common case where a wrapped exception is thrown from the same method as the "causative exception" is caught. The above example was produced by running the program:

```
public class Junk {
    public static void main(String args[]) {
        try {
            a();
        } catch(HighLevelException e) {
            e.printStackTrace();
        }
    }
    static void a() throws HighLevelException {
        try {
            b();
        } catch(MidLevelException e) {
            throw new HighLevelException(e);
        }
    }
    static void b() throws MidLevelException {
        c();
    }
    static void c() throws MidLevelException {
        try {
            d();
        } catch(LowLevelException e) {
            throw new MidLevelException(e);
        }
    }
    static void d() throws LowLevelException {
       e();
    }
    static void e() throws LowLevelException {
        throw new LowLevelException();
    }
}

class HighLevelException extends Exception {
    HighLevelException(Throwable cause) { super(cause); }
}

class MidLevelException extends Exception {
    MidLevelException(Throwable cause)  { super(cause); }
}

class LowLevelException extends Exception {
}
```

As of release 7, the platform supports the notion of *suppressed exceptions* (in conjunction with the `try`-with-resources statement). Any exceptions that were suppressed in order to deliver an exception are printed out beneath the stack trace. The format of this information depends on the implementation, but the following example may be regarded as typical:

```
Exception in thread "main" java.lang.Exception: Something happened
        at Foo.bar(Foo.java:10)
        at Foo.main(Foo.java:5)
        Suppressed: Resource$CloseFailException: Resource ID = 0
                at Resource.close(Resource.java:26)
                at Foo.bar(Foo.java:9)
                ... 1 more
```

Note that the "... n more" notation is used on suppressed exceptions just as it is used on causes. Unlike causes, suppressed exceptions are indented beyond their "containing exceptions."

An exception can have both a cause and one or more suppressed exceptions:

```
Exception in thread "main" java.lang.Exception: Main block
        at Foo3.main(Foo3.java:7)
        Suppressed: Resource$CloseFailException: Resource ID = 2
                at Resource.close(Resource.java:26)
                at Foo3.main(Foo3.java:5)
        Suppressed: Resource$CloseFailException: Resource ID = 1
                at Resource.close(Resource.java:26)
                at Foo3.main(Foo3.java:5)
Caused by: java.lang.Exception: I did it
        at Foo3.main(Foo3.java:8)
```

Likewise, a suppressed exception can have a cause:

```
Exception in thread "main" java.lang.Exception: Main block
        at Foo4.main(Foo4.java:6)
        Suppressed: Resource2$CloseFailException: Resource ID = 1
                at Resource2.close(Resource2.java:20)
                at Foo4.main(Foo4.java:5)
        Caused by: java.lang.Exception: Rats, you caught me
                at Resource2$CloseFailException.<init>(Resource2.java:45)
                ... 2 more
```

printStackTrace

```
public void printStackTrace(PrintStream s)
```

Prints this throwable and its backtrace to the specified print stream.

Parameters:

`s` - `PrintStream` to use for output

printStackTrace

```
public void printStackTrace(PrintWriter s)
```

Prints this throwable and its backtrace to the specified print writer.

Parameters:

`s` - `PrintWriter` to use for output

Since:

1.1

fillInStackTrace

```
public Throwable fillInStackTrace()
```

Fills in the execution stack trace. This method records within this `Throwable` object information about the current state of the stack frames for the current thread.

If the stack trace of this `Throwable` is not writable, calling this method has no effect.

Returns:

a reference to this `Throwable` instance.

See Also:

`printStackTrace()`

getStackTrace

```
public StackTraceElement[] getStackTrace()
```

Provides programmatic access to the stack trace information printed by `printStackTrace()`. Returns an array of stack trace elements, each representing one stack frame. The zeroth element of the array (assuming the array's length is non-zero) represents the top of the stack, which is the last method invocation in the sequence. Typically, this is the point at which this throwable was created and thrown. The last element of the array (assuming the array's length is non-zero) represents the bottom of the stack, which is the first method invocation in the sequence.

Some virtual machines may, under some circumstances, omit one or more stack frames from the stack trace. In the extreme case, a virtual machine that has no stack trace information concerning this throwable is permitted to return a zero-length array from this method. Generally speaking, the array returned by this method will contain one element for every frame that would be printed by `printStackTrace`. Writes to the returned array do not affect future calls to this method.

Returns:

an array of stack trace elements representing the stack trace pertaining to this throwable.

Since:

1.4

setStackTrace

```
public void setStackTrace(StackTraceElement[] stackTrace)
```

Sets the stack trace elements that will be returned by `getStackTrace()` and printed by `printStackTrace()` and related methods. This method, which is designed for use by RPC frameworks and other advanced systems, allows the client to override the default stack trace that is either generated by `fillInStackTrace()` when a throwable is constructed or deserialized when a throwable is read from a serialization stream.

If the stack trace of this `Throwable` is not writable, calling this method has no effect other than validating its argument.

Parameters:

`stackTrace` - the stack trace elements to be associated with this `Throwable`. The specified array is copied by this call; changes in the specified array after the method invocation returns will have no effect on this `Throwable`'s stack trace.

Throws:

`NullPointerException` - if `stackTrace` is `null` or if any of the elements of `stackTrace` are `null`

Since:

1.4

addSuppressed

```
public final void addSuppressed(Throwable exception)
```

Appends the specified exception to the exceptions that were suppressed in order to deliver this exception. This method is thread-safe and typically called (automatically and implicitly) by the `try`-with-resources statement.

The suppression behavior is enabled *unless* disabled via a constructor. When suppression is disabled, this method does nothing other than to validate its argument.

Note that when one exception causes another exception, the first exception is usually caught and then the second exception is thrown in response. In other words, there is a causal connection between the two exceptions. In contrast, there are situations where two independent exceptions can be thrown in sibling code blocks, in particular in the `try` block of a `try`-with-resources statement and the compiler-generated `finally` block which closes the resource. In these situations, only one of the thrown exceptions can be propagated. In the `try`-with-resources statement, when there are two such exceptions, the exception originating from the `try` block is propagated and the exception from the `finally` block is added to the list of exceptions suppressed by the exception from the `try` block. As an exception unwinds the stack, it can accumulate multiple suppressed exceptions.

An exception may have suppressed exceptions while also being caused by another exception. Whether or not an exception has a cause is semantically known at the time of its creation, unlike whether or not an exception will suppress other exceptions which is typically only determined after an exception is thrown.

Note that programmer written code is also able to take advantage of calling this method in situations where there are multiple sibling exceptions and only one can be propagated.

Parameters:

`exception` - the exception to be added to the list of suppressed exceptions

Throws:

`IllegalArgumentException` - if `exception` is this throwable; a throwable cannot suppress itself.

`NullPointerException` - if `exception` is `null`

Since:

1.7

getSuppressed

```
public final Throwable[] getSuppressed()
```

Returns an array containing all of the exceptions that were suppressed, typically by the `try`-with-resources statement, in order to deliver this exception. If no exceptions were suppressed or suppression is disabled, an empty array is returned. This method is thread-safe. Writes to the returned array do not affect future calls to this method.

Returns:

an array containing all of the exceptions that were suppressed to deliver this exception.

Since:

1.7