Java Mapping for Exceptions

On this page:

Java Mapping for User Exceptions

Here is a fragment of the Slice definition for our world time server once more:

Slice

```
exception GenericError {
    string reason;
};
exception BadTimeVal extends GenericError {};
exception BadZoneName extends GenericError {};
```

These exception definitions map as follows:

Java

```
public class GenericError extends Ice.UserException {
    public String reason;

    public GenericError() {}

    public GenericError(Throwable cause)
    {
        super(cause);
    }

    public GenericError(String reason)
    {
        this.reason = reason;
    }

    public GenericError(String reason, Throwable cause)
    {
        super(cause);
        this.reason = reason;
    }

    public String ice_name()
    {
        return "GenericError";
    }
}

public class BadTimeVal extends GenericError {
    public BadTimeVal() {}

    public BadTimeVal(Throwable cause)
    {
        super(cause);
    }

    public BadTimeVal(String reason)
    {
```

```
        super(reason);
    }

    public BadTimeVal(String reason, Throwable cause)
    {
        super(reason, cause);
    }

    public String ice_name()
    {
        return "BadTimeVal";
    }
}

public class BadZoneName extends GenericError {
    public BadZoneName() {}

    public BadZoneName(Throwable cause)
    {
        super(cause);
    }

    public BadZoneName(String reason)
    {
        super(reason);
    }

    public BadZoneName(String reason, Throwable cause)
    {
        super(reason, cause);
    }

    public String ice_name()
    {
        return "BadZoneName";
    }
}
```

Each Slice exception is mapped to a Java class with the same name. For each data member, the corresponding class contains a public data member. (Obviously, because `BadTimeVal` and `BadZoneName` do not have members, the generated classes for these exceptions also do not have members.) A JavaBean-style API is used for optional data members, and you can customize the mapping to force required members to use this same API.

The inheritance structure of the Slice exceptions is preserved for the generated classes, so `BadTimeVal` and `BadZoneName` inherit from `GenericError`.

Each exception also defines the `ice_name` member function, which returns the name of the exception.

All user exceptions are derived from the base class `Ice.UserException`. This allows you to catch all user exceptions generically by installing a handler for `Ice.UserException`. `Ice.UserException`, in turn, derives from `java.lang.Exception`.

`Ice.UserException` implements a `clone` method that is inherited by its derived exceptions, so you can make memberwise shallow copies of exceptions.

Note that the generated exception classes contain other member functions that are not shown. However, those member functions are internal to the Java mapping and are not meant to be called by application code.

Java Constructors for User Exceptions

An exception has a default constructor that default-constructs each data member. This means the constructor initializes members of primitive type to the equivalent of zero, and members of reference type to null. Note that applications must always explicitly initialize members of structure and enumerated types because the Ice run time does not accept null as a legal value for these types.

If you wish to ensure that data members of primitive and enumerated types are initialized to specific values, you can declare default values in your Slice definition. The default constructor initializes each of these data members to its declared value.

If an exception declares or inherits any data members, the generated class provides a second constructor that accepts one parameter for each data member so that you can construct and initialize an instance in a single statement (instead of first having to construct the instance and then assign to its members). For a derived exception, this constructor accepts one argument for each base exception member, plus one argument for each derived exception member, in base-to-derived order.

The generated class may include an additional constructor if the exception declares or inherits any optional data members.

The Slice compiler generates overloaded versions of all constructors that accept a trailing `Throwable` argument for preserving an exception chain.

Java Mapping for Run-Time Exceptions

The Ice run time throws run-time exceptions for a number of pre-defined error conditions. All run-time exceptions directly or indirectly derive from `Ice.LocalException` (which, in turn, derives from `java.lang.RuntimeException`).

`Ice.LocalException` implements a `clone` method that is inherited by its derived exceptions, so you can make memberwise shallow copies of exceptions.

Recall the inheritance diagram for user and run-time exceptions. By catching exceptions at the appropriate point in the hierarchy, you can handle exceptions according to the category of error they indicate:

- `Ice.LocalException`
  This is the root of the inheritance tree for run-time exceptions.
- `Ice.UserException`
  This is the root of the inheritance tree for user exceptions.
- `Ice.TimeoutException`
  This is the base exception for both operation-invocation and connection-establishment timeouts.
- `Ice.ConnectTimeoutException`
  This exception is raised when the initial attempt to establish a connection to a server times out.

For example, a `ConnectTimeoutException` can be handled as `ConnectTimeoutException`, `TimeoutException`, `LocalException`, or `java.lang.Exception`.

You will probably have little need to catch run-time exceptions as their most-derived type and instead catch them as `LocalException`; the fine-grained error handling offered by the remainder of the hierarchy is of interest mainly in the implementation of the Ice run time. Exceptions to this rule are the exceptions related to facet and object life cycles, which you may want to catch explicitly. These exceptions are `FacetNotExistException` and `ObjectNotExistException`, respectively.

See Also

- User Exceptions
- Run-Time Exceptions
- Java Mapping for Modules
- Java Mapping for Built-In Types
- Java Mapping for Enumerations
- Java Mapping for Structures
- Java Mapping for Sequences
- Java Mapping for Dictionaries
- Java Mapping for Constants
- Java Mapping for Optional Data Members
- JavaBean Mapping
- Facets and Versioning
- Object Life Cycle