exception.cm

```
/*
    ========================================================================

    Copyright (c) 2012-2016 Seppo Laakko
    http://sourceforge.net/projects/cmajor/

    Distributed under the GNU General Public License, version 3 (GPLv3).
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html
        )

    ========================================================================
    */

// Copyright (c) 1994
// Hewlett-Packard Company
// Copyright (c) 1996
// Silicon Graphics Computer Systems, Inc.
// Copyright (c) 2009 Alexander Stepanov and Paul McJones

namespace System
{
    public class Exception
    {
        public nothrow Exception(): message(), file(), line(0)
        {
        }
        public nothrow Exception(const string& message_): message(
            message_), file(), line(0)
        {
        }
        public nothrow default Exception(const Exception& that);
            // generate copy constructor although has user-defined
            destructor
        public nothrow default void operator=(const Exception& that);
            // generate copy assignment although has user-defined
            destructor
        public nothrow default Exception(Exception&&);
            // generate move constructor although has user-defined
            destructor
        public nothrow default void operator=(Exception&&);
            // generate move assignment although has user-defined
            destructor
        public nothrow virtual ~Exception()
        {
        }
        public virtual string ToString() const
        {
            string s = exceptionType;
            if (!file.IsEmpty())
            {
                s.Append(" at '");
```

```
            s.Append(file);
            if (line != 0)
            {
                s.Append("' line ");
                s.Append(ToString(line));
            }
            else
            {
                s.Append('\'');
            }
        }
        s.Append(":\n");
        s.Append(message);
        s.Append("\n");
        if (!callStack.IsEmpty())
        {
            s.Append(callStack);
        }
        return s;
    }
    public nothrow const string& ExceptionType() const
    {
        return exceptionType;
    }
    public nothrow void SetExceptionType(const string& exceptionType_
        )
    {
        exceptionType = exceptionType_;
    }
    public nothrow const string& Message() const
    {
        return message;
    }
    public nothrow const string& File() const
    {
        return file;
    }
    public nothrow int Line() const
    {
        return line;
    }
    public nothrow void SetFile(const string& file_)
    {
        file = file_;
    }
    public nothrow void SetLine(int line_)
    {
        line = line_;
    }
    public nothrow void SetCallStack(const string& callStack_)
    {
        callStack = callStack_;
    }
```

```
        private string exceptionType;
        private string message;
        private string file;
        private int line;
        private string callStack;
    }

    public class ExceptionPtr
    {
        public nothrow ExceptionPtr() : exceptionId(0), exception(null)
        {
        }
        suppress ExceptionPtr(const ExceptionPtr&);
        suppress void operator=(const ExceptionPtr&);
        public nothrow ExceptionPtr(ExceptionPtr&& that) : exceptionId(
            that.exceptionId), exception(that.exception)
        {
            that.exceptionId = 0;
            that.exception = null;
        }
        public nothrow void operator=(ExceptionPtr&& that)
        {
            Swap(exceptionId, that.exceptionId);
            Swap(exception, that.exception);
        }
        public nothrow ExceptionPtr(int exceptionId_, Exception*
            exception_) : exceptionId(exceptionId_), exception(exception_)
        {
        }
        public nothrow ~ExceptionPtr()
        {
            if (exception != null)
            {
                delete exception;
            }
        }
        public nothrow inline bool HasException() const
        {
            return exceptionId != 0 && exception != null;
        }
        public nothrow inline int ExceptionId() const
        {
            return exceptionId;
        }
        public nothrow Exception* Release() const
        {
            Exception* ex = exception;
            exception = null;
            exceptionId = 0;
            return ex;
        }
        private int exceptionId;
        private Exception* exception;
```

```
    }

    public nothrow ExceptionPtr CaptureCurrentException()
    {
        int current_exception_id = get_current_exception_id();
        if (current_exception_id != 0)
        {
            set_current_exception_captured();
            set_current_exception_id(0);
        }
        return ExceptionPtr(current_exception_id, cast<Exception*>(
            get_current_exception_addr()));
    }

    public void RethrowException(ExceptionPtr& capturedException)
    {
        if (capturedException.HasException())
        {
            set_current_exception_id(capturedException.ExceptionId());
            set_current_exception_addr(capturedException.Release());
        }
        else
        {
            throw Exception("exception ptr has no exception");
        }
    }
}
```