001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017package org.apache.commons.collections4.functors;  
018  
019import org.apache.commons.collections4.Closure;  
020import org.apache.commons.collections4.FunctorException;  
021  
022/\*\*  
023 \* {@link Closure} that catches any checked exception and re-throws it as a  
024 \* {@link FunctorException} runtime exception. Example usage:  
025 \*  
026 \* <pre>  
027 \* // Create a catch and re-throw closure via anonymous subclass  
028 \* CatchAndRethrowClosure<String> writer = new ThrowingClosure() {  
029 \* private java.io.Writer out = // some writer  
030 \*  
031 \* protected void executeAndThrow(String input) throws IOException {  
032 \* out.write(input); // throwing of IOException allowed  
033 \* }  
034 \* };  
035 \*  
036 \* // use catch and re-throw closure  
037 \* java.util.List<String> strList = // some list  
038 \* try {  
039 \* CollectionUtils.forAllDo(strList, writer);  
040 \* } catch (FunctorException ex) {  
041 \* Throwable originalError = ex.getCause();  
042 \* // handle error  
043 \* }  
044 \* </pre>  
045 \*  
046 \* @since 4.0  
047 \*/  
048public abstract class CatchAndRethrowClosure<E> implements Closure<E> {  
049  
050 /\*\*  
051 \* Execute this closure on the specified input object.  
052 \*  
053 \* @param input the input to execute on  
054 \* @throws FunctorException (runtime) if the closure execution resulted in a  
055 \* checked exception.  
056 \*/  
057 @Override  
058 public void execute(final E input) {  
059 try {  
060 executeAndThrow(input);  
061 } catch (final RuntimeException ex) {  
062 throw ex;  
063 } catch (final Throwable t) {  
064 throw new FunctorException(t);  
065 }  
066 }  
067  
068 /\*\*  
069 \* Execute this closure on the specified input object.  
070 \*  
071 \* @param input the input to execute on  
072 \* @throws Throwable if the closure execution resulted in a checked  
073 \* exception.  
074 \*/  
075 protected abstract void executeAndThrow(E input) throws Throwable;  
076}