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.list;  
018  
019import java.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.util.Collection;  
023import java.util.List;  
024  
025/\*\*  
026 \* Serializable subclass of AbstractListDecorator.  
027 \*  
028 \* @since 3.1  
029 \*/  
030public abstract class AbstractSerializableListDecorator<E>  
031 extends AbstractListDecorator<E> {  
032  
033 /\*\* Serialization version \*/  
034 private static final long serialVersionUID = 2684959196747496299L;  
035  
036 /\*\*  
037 \* Constructor that wraps (not copies).  
038 \*  
039 \* @param list the list to decorate, must not be null  
040 \* @throws NullPointerException if list is null  
041 \*/  
042 protected AbstractSerializableListDecorator(final List<E> list) {  
043 super(list);  
044 }  
045  
046 //-----------------------------------------------------------------------  
047 /\*\*  
048 \* Write the list out using a custom routine.  
049 \*  
050 \* @param out the output stream  
051 \* @throws IOException if an error occurs while writing to the stream  
052 \*/  
053 private void writeObject(final ObjectOutputStream out) throws IOException {  
054 out.defaultWriteObject();  
055 out.writeObject(decorated());  
056 }  
057  
058 /\*\*  
059 \* Read the list in using a custom routine.  
060 \*  
061 \* @param in the input stream  
062 \* @throws IOException if an error occurs while reading from the stream  
063 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
064 \*/  
065 @SuppressWarnings("unchecked")  
066 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
067 in.defaultReadObject();  
068 setCollection((Collection<E>) in.readObject());  
069 }  
070  
071}