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.sequence;  
018  
019/\*\*  
020 \* Abstract base class for all commands used to transform an objects sequence  
021 \* into another one.  
022 \* <p>  
023 \* When two objects sequences are compared through the  
024 \* {@link SequencesComparator#getScript SequencesComparator.getScript} method,  
025 \* the result is provided has a {@link EditScript script} containing the commands  
026 \* that progressively transform the first sequence into the second one.  
027 \* </p>  
028 \* <p>  
029 \* There are only three types of commands, all of which are subclasses of this  
030 \* abstract class. Each command is associated with one object belonging to at  
031 \* least one of the sequences. These commands are {@link InsertCommand  
032 \* InsertCommand} which correspond to an object of the second sequence being  
033 \* inserted into the first sequence, {@link DeleteCommand DeleteCommand} which  
034 \* correspond to an object of the first sequence being removed and  
035 \* {@link KeepCommand KeepCommand} which correspond to an object of the first  
036 \* sequence which <code>equals</code> an object in the second sequence. It is  
037 \* guaranteed that comparison is always performed this way (i.e. the  
038 \* <code>equals</code> method of the object from the first sequence is used and  
039 \* the object passed as an argument comes from the second sequence) ; this can  
040 \* be important if subclassing is used for some elements in the first sequence  
041 \* and the <code>equals</code> method is specialized.  
042 \* </p>  
043 \*  
044 \* @see SequencesComparator  
045 \* @see EditScript  
046 \*  
047 \* @since 4.0  
048 \*/  
049public abstract class EditCommand<T> {  
050  
051 /\*\* Object on which the command should be applied. \*/  
052 private final T object;  
053  
054 /\*\*  
055 \* Simple constructor. Creates a new instance of EditCommand  
056 \*  
057 \* @param object reference to the object associated with this command, this  
058 \* refers to an element of one of the sequences being compared  
059 \*/  
060 protected EditCommand(final T object) {  
061 this.object = object;  
062 }  
063  
064 /\*\*  
065 \* Returns the object associated with this command.  
066 \*  
067 \* @return the object on which the command is applied  
068 \*/  
069 protected T getObject() {  
070 return object;  
071 }  
072  
073 /\*\*  
074 \* Accept a visitor.  
075 \* <p>  
076 \* This method is invoked for each commands belonging to  
077 \* an {@link EditScript EditScript}, in order to implement the visitor design pattern  
078 \*  
079 \* @param visitor the visitor to be accepted  
080 \*/  
081 public abstract void accept(CommandVisitor<T> visitor);  
082  
083}