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.set;  
018  
019import java.util.Set;  
020  
021import org.apache.commons.collections4.Predicate;  
022import org.apache.commons.collections4.collection.PredicatedCollection;  
023  
024/\*\*  
025 \* Decorates another <code>Set</code> to validate that all additions  
026 \* match a specified predicate.  
027 \* <p>  
028 \* This set exists to provide validation for the decorated set.  
029 \* It is normally created to decorate an empty set.  
030 \* If an object cannot be added to the set, an IllegalArgumentException is thrown.  
031 \* </p>  
032 \* <p>  
033 \* One usage would be to ensure that no null entries are added to the set.  
034 \* </p>  
035 \* <pre>Set set = PredicatedSet.decorate(new HashSet(), NotNullPredicate.INSTANCE);</pre>  
036 \* <p>  
037 \* This class is Serializable from Commons Collections 3.1.  
038 \* </p>  
039 \*  
040 \* @param <E> the type of the elements in this set  
041 \* @since 3.0  
042 \*/  
043public class PredicatedSet<E> extends PredicatedCollection<E> implements Set<E> {  
044  
045 /\*\* Serialization version \*/  
046 private static final long serialVersionUID = -684521469108685117L;  
047  
048 /\*\*  
049 \* Factory method to create a predicated (validating) set.  
050 \* <p>  
051 \* If there are any elements already in the set being decorated, they  
052 \* are validated.  
053 \*  
054 \* @param <E> the element type  
055 \* @param set the set to decorate, must not be null  
056 \* @param predicate the predicate to use for validation, must not be null  
057 \* @return a decorated set  
058 \* @throws NullPointerException if set or predicate is null  
059 \* @throws IllegalArgumentException if the set contains invalid elements  
060 \* @since 4.0  
061 \*/  
062 public static <E> PredicatedSet<E> predicatedSet(final Set<E> set, final Predicate<? super E> predicate) {  
063 return new PredicatedSet<>(set, predicate);  
064 }  
065  
066 //-----------------------------------------------------------------------  
067 /\*\*  
068 \* Constructor that wraps (not copies).  
069 \* <p>  
070 \* If there are any elements already in the set being decorated, they  
071 \* are validated.  
072 \*  
073 \* @param set the set to decorate, must not be null  
074 \* @param predicate the predicate to use for validation, must not be null  
075 \* @throws NullPointerException if set or predicate is null  
076 \* @throws IllegalArgumentException if the set contains invalid elements  
077 \*/  
078 protected PredicatedSet(final Set<E> set, final Predicate<? super E> predicate) {  
079 super(set, predicate);  
080 }  
081  
082 /\*\*  
083 \* Gets the set being decorated.  
084 \*  
085 \* @return the decorated set  
086 \*/  
087 @Override  
088 protected Set<E> decorated() {  
089 return (Set<E>) super.decorated();  
090 }  
091  
092 @Override  
093 public boolean equals(final Object object) {  
094 return object == this || decorated().equals(object);  
095 }  
096  
097 @Override  
098 public int hashCode() {  
099 return decorated().hashCode();  
100 }  
101  
102}