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.bag;  
018  
019import java.util.Comparator;  
020  
021import org.apache.commons.collections4.Predicate;  
022import org.apache.commons.collections4.SortedBag;  
023  
024/\*\*  
025 \* Decorates another {@link SortedBag} to validate that additions  
026 \* match a specified predicate.  
027 \* <p>  
028 \* This bag exists to provide validation for the decorated bag.  
029 \* It is normally created to decorate an empty bag.  
030 \* If an object cannot be added to the bag, an {@link IllegalArgumentException} is thrown.  
031 \* </p>  
032 \* <p>  
033 \* One usage would be to ensure that no null entries are added to the bag.  
034 \* <pre>  
035 \* SortedBag bag = PredicatedSortedBag.predicatedSortedBag(new TreeBag(), NotNullPredicate.INSTANCE);  
036 \* </pre>  
037 \* <p>  
038 \* This class is Serializable from Commons Collections 3.1.  
039 \* </p>  
040 \*  
041 \* @param <E> the type of elements in this bag  
042 \* @since 3.0  
043 \*/  
044public class PredicatedSortedBag<E> extends PredicatedBag<E> implements SortedBag<E> {  
045  
046 /\*\* Serialization version \*/  
047 private static final long serialVersionUID = 3448581314086406616L;  
048  
049 /\*\*  
050 \* Factory method to create a predicated (validating) bag.  
051 \* <p>  
052 \* If there are any elements already in the bag being decorated, they  
053 \* are validated.  
054 \*  
055 \* @param <E> the type of the elements in the bag  
056 \* @param bag the bag to decorate, must not be null  
057 \* @param predicate the predicate to use for validation, must not be null  
058 \* @return a new predicated SortedBag  
059 \* @throws NullPointerException if bag or predicate is null  
060 \* @throws IllegalArgumentException if the bag contains invalid elements  
061 \* @since 4.0  
062 \*/  
063 public static <E> PredicatedSortedBag<E> predicatedSortedBag(final SortedBag<E> bag,  
064 final Predicate<? super E> predicate) {  
065 return new PredicatedSortedBag<>(bag, predicate);  
066 }  
067  
068 //-----------------------------------------------------------------------  
069 /\*\*  
070 \* Constructor that wraps (not copies).  
071 \* <p>If there are any elements already in the bag being decorated, they  
072 \* are validated.  
073 \*  
074 \* @param bag the bag to decorate, must not be null  
075 \* @param predicate the predicate to use for validation, must not be null  
076 \* @throws NullPointerException if bag or predicate is null  
077 \* @throws IllegalArgumentException if the bag contains invalid elements  
078 \*/  
079 protected PredicatedSortedBag(final SortedBag<E> bag, final Predicate<? super E> predicate) {  
080 super(bag, predicate);  
081 }  
082  
083 /\*\*  
084 \* Gets the decorated sorted bag.  
085 \*  
086 \* @return the decorated bag  
087 \*/  
088 @Override  
089 protected SortedBag<E> decorated() {  
090 return (SortedBag<E>) super.decorated();  
091 }  
092  
093 //-----------------------------------------------------------------------  
094  
095 @Override  
096 public E first() {  
097 return decorated().first();  
098 }  
099  
100 @Override  
101 public E last() {  
102 return decorated().last();  
103 }  
104  
105 @Override  
106 public Comparator<? super E> comparator() {  
107 return decorated().comparator();  
108 }  
109  
110}