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.Set;  
020  
021import org.apache.commons.collections4.Bag;  
022import org.apache.commons.collections4.Predicate;  
023import org.apache.commons.collections4.collection.PredicatedCollection;  
024  
025/\*\*  
026 \* Decorates another {@link Bag} to validate that additions  
027 \* match a specified predicate.  
028 \* <p>  
029 \* This bag exists to provide validation for the decorated bag.  
030 \* It is normally created to decorate an empty bag.  
031 \* If an object cannot be added to the bag, an {@link IllegalArgumentException} is thrown.  
032 \* </p>  
033 \* <p>  
034 \* One usage would be to ensure that no null entries are added to the bag.  
035 \* </p>  
036 \* <pre>  
037 \* Bag bag = PredicatedBag.predicatedBag(new HashBag(), NotNullPredicate.INSTANCE);  
038 \* </pre>  
039 \* <p>  
040 \* This class is Serializable from Commons Collections 3.1.  
041 \* </p>  
042 \*  
043 \* @param <E> the type of elements in this bag  
044 \* @since 3.0  
045 \*/  
046public class PredicatedBag<E> extends PredicatedCollection<E> implements Bag<E> {  
047  
048 /\*\* Serialization version \*/  
049 private static final long serialVersionUID = -2575833140344736876L;  
050  
051 /\*\*  
052 \* Factory method to create a predicated (validating) bag.  
053 \* <p>  
054 \* If there are any elements already in the bag being decorated, they  
055 \* are validated.  
056 \*  
057 \* @param <E> the type of the elements in the bag  
058 \* @param bag the bag to decorate, must not be null  
059 \* @param predicate the predicate to use for validation, must not be null  
060 \* @return a new predicated Bag  
061 \* @throws NullPointerException if bag or predicate is null  
062 \* @throws IllegalArgumentException if the bag contains invalid elements  
063 \* @since 4.0  
064 \*/  
065 public static <E> PredicatedBag<E> predicatedBag(final Bag<E> bag, final Predicate<? super E> predicate) {  
066 return new PredicatedBag<>(bag, predicate);  
067 }  
068  
069 //-----------------------------------------------------------------------  
070 /\*\*  
071 \* Constructor that wraps (not copies).  
072 \* <p>  
073 \* If there are any elements already in the bag being decorated, they  
074 \* are validated.  
075 \*  
076 \* @param bag the bag to decorate, must not be null  
077 \* @param predicate the predicate to use for validation, must not be null  
078 \* @throws NullPointerException if bag or predicate is null  
079 \* @throws IllegalArgumentException if the bag contains invalid elements  
080 \*/  
081 protected PredicatedBag(final Bag<E> bag, final Predicate<? super E> predicate) {  
082 super(bag, predicate);  
083 }  
084  
085 /\*\*  
086 \* Gets the decorated bag.  
087 \*  
088 \* @return the decorated bag  
089 \*/  
090 @Override  
091 protected Bag<E> decorated() {  
092 return (Bag<E>) super.decorated();  
093 }  
094  
095 @Override  
096 public boolean equals(final Object object) {  
097 return object == this || decorated().equals(object);  
098 }  
099  
100 @Override  
101 public int hashCode() {  
102 return decorated().hashCode();  
103 }  
104  
105 //-----------------------------------------------------------------------  
106  
107 @Override  
108 public boolean add(final E object, final int count) {  
109 validate(object);  
110 return decorated().add(object, count);  
111 }  
112  
113 @Override  
114 public boolean remove(final Object object, final int count) {  
115 return decorated().remove(object, count);  
116 }  
117  
118 @Override  
119 public Set<E> uniqueSet() {  
120 return decorated().uniqueSet();  
121 }  
122  
123 @Override  
124 public int getCount(final Object object) {  
125 return decorated().getCount(object);  
126 }  
127  
128}