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.collection;  
018  
019import java.io.Serializable;  
020import java.util.Collection;  
021import java.util.Iterator;  
022import java.util.function.Predicate;  
023  
024/\*\*  
025 \* Decorates another {@link Collection} to synchronize its behaviour  
026 \* for a multi-threaded environment.  
027 \* <p>  
028 \* Iterators must be manually synchronized:  
029 \* </p>  
030 \* <pre>  
031 \* synchronized (coll) {  
032 \* Iterator it = coll.iterator();  
033 \* // do stuff with iterator  
034 \* }  
035 \* </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 the collection  
041 \* @since 3.0  
042 \*/  
043public class SynchronizedCollection<E> implements Collection<E>, Serializable {  
044  
045 /\*\* Serialization version \*/  
046 private static final long serialVersionUID = 2412805092710877986L;  
047  
048 /\*\* The collection to decorate \*/  
049 private final Collection<E> collection;  
050 /\*\* The object to lock on, needed for List/SortedSet views \*/  
051 protected final Object lock;  
052  
053 /\*\*  
054 \* Factory method to create a synchronized collection.  
055 \*  
056 \* @param <T> the type of the elements in the collection  
057 \* @param coll the collection to decorate, must not be null  
058 \* @return a new synchronized collection  
059 \* @throws NullPointerException if collection is null  
060 \* @since 4.0  
061 \*/  
062 public static <T> SynchronizedCollection<T> synchronizedCollection(final Collection<T> coll) {  
063 return new SynchronizedCollection<>(coll);  
064 }  
065  
066 //-----------------------------------------------------------------------  
067 /\*\*  
068 \* Constructor that wraps (not copies).  
069 \*  
070 \* @param collection the collection to decorate, must not be null  
071 \* @throws NullPointerException if the collection is null  
072 \*/  
073 protected SynchronizedCollection(final Collection<E> collection) {  
074 if (collection == null) {  
075 throw new NullPointerException("Collection must not be null.");  
076 }  
077 this.collection = collection;  
078 this.lock = this;  
079 }  
080  
081 /\*\*  
082 \* Constructor that wraps (not copies).  
083 \*  
084 \* @param collection the collection to decorate, must not be null  
085 \* @param lock the lock object to use, must not be null  
086 \* @throws NullPointerException if the collection or lock is null  
087 \*/  
088 protected SynchronizedCollection(final Collection<E> collection, final Object lock) {  
089 if (collection == null) {  
090 throw new NullPointerException("Collection must not be null.");  
091 }  
092 if (lock == null) {  
093 throw new NullPointerException("Lock must not be null.");  
094 }  
095 this.collection = collection;  
096 this.lock = lock;  
097 }  
098  
099 /\*\*  
100 \* Gets the collection being decorated.  
101 \*  
102 \* @return the decorated collection  
103 \*/  
104 protected Collection<E> decorated() {  
105 return collection;  
106 }  
107  
108 //-----------------------------------------------------------------------  
109  
110 @Override  
111 public boolean add(final E object) {  
112 synchronized (lock) {  
113 return decorated().add(object);  
114 }  
115 }  
116  
117 @Override  
118 public boolean addAll(final Collection<? extends E> coll) {  
119 synchronized (lock) {  
120 return decorated().addAll(coll);  
121 }  
122 }  
123  
124 @Override  
125 public void clear() {  
126 synchronized (lock) {  
127 decorated().clear();  
128 }  
129 }  
130  
131 @Override  
132 public boolean contains(final Object object) {  
133 synchronized (lock) {  
134 return decorated().contains(object);  
135 }  
136 }  
137  
138 @Override  
139 public boolean containsAll(final Collection<?> coll) {  
140 synchronized (lock) {  
141 return decorated().containsAll(coll);  
142 }  
143 }  
144  
145 @Override  
146 public boolean isEmpty() {  
147 synchronized (lock) {  
148 return decorated().isEmpty();  
149 }  
150 }  
151  
152 /\*\*  
153 \* Iterators must be manually synchronized.  
154 \* <pre>  
155 \* synchronized (coll) {  
156 \* Iterator it = coll.iterator();  
157 \* // do stuff with iterator  
158 \* }  
159 \* </pre>  
160 \*  
161 \* @return an iterator that must be manually synchronized on the collection  
162 \*/  
163 @Override  
164 public Iterator<E> iterator() {  
165 return decorated().iterator();  
166 }  
167  
168 @Override  
169 public Object[] toArray() {  
170 synchronized (lock) {  
171 return decorated().toArray();  
172 }  
173 }  
174  
175 @Override  
176 public <T> T[] toArray(final T[] object) {  
177 synchronized (lock) {  
178 return decorated().toArray(object);  
179 }  
180 }  
181  
182 @Override  
183 public boolean remove(final Object object) {  
184 synchronized (lock) {  
185 return decorated().remove(object);  
186 }  
187 }  
188  
189 /\*\*  
190 \* @since 4.4  
191 \*/  
192 @Override  
193 public boolean removeIf(final Predicate<? super E> filter) {  
194 synchronized (lock) {  
195 return decorated().removeIf(filter);  
196 }  
197 }  
198  
199 @Override  
200 public boolean removeAll(final Collection<?> coll) {  
201 synchronized (lock) {  
202 return decorated().removeAll(coll);  
203 }  
204 }  
205  
206 @Override  
207 public boolean retainAll(final Collection<?> coll) {  
208 synchronized (lock) {  
209 return decorated().retainAll(coll);  
210 }  
211 }  
212  
213 @Override  
214 public int size() {  
215 synchronized (lock) {  
216 return decorated().size();  
217 }  
218 }  
219  
220 @Override  
221 public boolean equals(final Object object) {  
222 synchronized (lock) {  
223 if (object == this) {  
224 return true;  
225 }  
226 return object == this || decorated().equals(object);  
227 }  
228 }  
229  
230 @Override  
231 public int hashCode() {  
232 synchronized (lock) {  
233 return decorated().hashCode();  
234 }  
235 }  
236  
237 @Override  
238 public String toString() {  
239 synchronized (lock) {  
240 return decorated().toString();  
241 }  
242 }  
243  
244}