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.trie;  
018  
019import java.io.Serializable;  
020import java.util.AbstractMap;  
021import java.util.Map;  
022import java.util.Map.Entry;  
023  
024import org.apache.commons.collections4.Trie;  
025  
026/\*\*  
027 \* This class provides some basic {@link Trie} functionality and  
028 \* utility methods for actual bitwise {@link Trie} implementations.  
029 \*  
030 \* @param <K> the type of the keys in this map  
031 \* @param <V> the type of the values in this map  
032 \* @since 4.0  
033 \*/  
034public abstract class AbstractBitwiseTrie<K, V> extends AbstractMap<K, V>  
035 implements Trie<K, V>, Serializable {  
036  
037 private static final long serialVersionUID = 5826987063535505652L;  
038  
039 /\*\*  
040 \* The {@link KeyAnalyzer} that's being used to build the PATRICIA {@link Trie}.  
041 \*/  
042 private final KeyAnalyzer<? super K> keyAnalyzer;  
043  
044 /\*\*  
045 \* Constructs a new {@link Trie} using the given {@link KeyAnalyzer}.  
046 \*  
047 \* @param keyAnalyzer the {@link KeyAnalyzer} to use  
048 \*/  
049 protected AbstractBitwiseTrie(final KeyAnalyzer<? super K> keyAnalyzer) {  
050 if (keyAnalyzer == null) {  
051 throw new NullPointerException("keyAnalyzer");  
052 }  
053  
054 this.keyAnalyzer = keyAnalyzer;  
055 }  
056  
057 /\*\*  
058 \* Returns the {@link KeyAnalyzer} that constructed the {@link Trie}.  
059 \* @return the {@link KeyAnalyzer} used by this {@link Trie}  
060 \*/  
061 protected KeyAnalyzer<? super K> getKeyAnalyzer() {  
062 return keyAnalyzer;  
063 }  
064  
065 @Override  
066 public String toString() {  
067 final StringBuilder buffer = new StringBuilder();  
068 buffer.append("Trie[").append(size()).append("]={\n");  
069 for (final Map.Entry<K, V> entry : entrySet()) {  
070 buffer.append(" ").append(entry).append("\n");  
071 }  
072 buffer.append("}\n");  
073 return buffer.toString();  
074 }  
075  
076 /\*\*  
077 \* A utility method to cast keys. It actually doesn't cast anything. It's just fooling the compiler!  
078 \*/  
079 @SuppressWarnings("unchecked")  
080 final K castKey(final Object key) {  
081 return (K) key;  
082 }  
083  
084 /\*\*  
085 \* Returns the length of the given key in bits  
086 \*  
087 \* @see KeyAnalyzer#lengthInBits(Object)  
088 \*/  
089 final int lengthInBits(final K key) {  
090 if (key == null) {  
091 return 0;  
092 }  
093  
094 return keyAnalyzer.lengthInBits(key);  
095 }  
096  
097 /\*\*  
098 \* Returns the number of bits per element in the key  
099 \*  
100 \* @see KeyAnalyzer#bitsPerElement()  
101 \*/  
102 final int bitsPerElement() {  
103 return keyAnalyzer.bitsPerElement();  
104 }  
105  
106 /\*\*  
107 \* Returns whether or not the given bit on the key is set or false if the key is null.  
108 \*  
109 \* @see KeyAnalyzer#isBitSet(Object, int, int)  
110 \*/  
111 final boolean isBitSet(final K key, final int bitIndex, final int lengthInBits) {  
112 if (key == null) { // root's might be null!  
113 return false;  
114 }  
115 return keyAnalyzer.isBitSet(key, bitIndex, lengthInBits);  
116 }  
117  
118 /\*\*  
119 \* Utility method for calling {@link KeyAnalyzer#bitIndex(Object, int, int, Object, int, int)}.  
120 \*/  
121 final int bitIndex(final K key, final K foundKey) {  
122 return keyAnalyzer.bitIndex(key, 0, lengthInBits(key), foundKey, 0, lengthInBits(foundKey));  
123 }  
124  
125 /\*\*  
126 \* An utility method for calling {@link KeyAnalyzer#compare(Object, Object)}  
127 \*/  
128 final boolean compareKeys(final K key, final K other) {  
129 if (key == null) {  
130 return other == null;  
131 } else if (other == null) {  
132 return false;  
133 }  
134  
135 return keyAnalyzer.compare(key, other) == 0;  
136 }  
137  
138 /\*\*  
139 \* Returns true if both values are either null or equal.  
140 \*/  
141 static boolean compare(final Object a, final Object b) {  
142 return a == null ? b == null : a.equals(b);  
143 }  
144  
145 /\*\*  
146 \* A basic implementation of {@link Entry}.  
147 \*/  
148 abstract static class BasicEntry<K, V> implements Map.Entry<K, V>, Serializable {  
149  
150 private static final long serialVersionUID = -944364551314110330L;  
151  
152 protected K key;  
153  
154 protected V value;  
155  
156 public BasicEntry(final K key) {  
157 this.key = key;  
158 }  
159  
160 public BasicEntry(final K key, final V value) {  
161 this.key = key;  
162 this.value = value;  
163 }  
164  
165 /\*\*  
166 \* Replaces the current key and value with the provided key & value.  
167 \*/  
168 public V setKeyValue(final K key, final V value) {  
169 this.key = key;  
170 return setValue(value);  
171 }  
172  
173 @Override  
174 public K getKey() {  
175 return key;  
176 }  
177  
178 @Override  
179 public V getValue() {  
180 return value;  
181 }  
182  
183 @Override  
184 public V setValue(final V value) {  
185 final V previous = this.value;  
186 this.value = value;  
187 return previous;  
188 }  
189  
190 @Override  
191 public int hashCode() {  
192 return (getKey() == null ? 0 : getKey().hashCode()) ^  
193 (getValue() == null ? 0 : getValue().hashCode());  
194 }  
195  
196 @Override  
197 public boolean equals(final Object o) {  
198 if (o == this) {  
199 return true;  
200 } else if (!(o instanceof Map.Entry)) {  
201 return false;  
202 }  
203  
204 final Map.Entry<?, ?> other = (Map.Entry<?, ?>)o;  
205 if (compare(key, other.getKey())  
206 && compare(value, other.getValue())) {  
207 return true;  
208 }  
209 return false;  
210 }  
211  
212 @Override  
213 public String toString() {  
214 return key + "=" + value;  
215 }  
216 }  
217}