```
 1 import java.util.Iterator;
 7
 8 /**
 9  * {@code Map} represented as a hash table using {@code Map}s for the
   buckets,
10  * with implementations of primary methods.
11  *
12  * @param <K>
13  *            type of {@code Map} domain (key) entries
14  * @param <V>
15  *            type of {@code Map} range (associated value) entries
16  * @convention <pre>
17  * |$this.hashTable| > 0  and
18  * for all i: integer, pf: PARTIAL_FUNCTION, x: K
19  *     where (0 <= i  and  i < |$this.hashTable|  and
20  *            <pf> = $this.hashTable[i, i+1)  and
21  *            x is in DOMAIN(pf))
22  *   ([computed result of x.hashCode()] mod |$this.hashTable| = i))
   and
23  * for all i: integer
24  *     where (0 <= i  and  i < |$this.hashTable|)
25  *   ([entry at position i in $this.hashTable is not null])  and
26  * $this.size = sum i: integer, pf: PARTIAL_FUNCTION
27  *     where (0 <= i  and  i < |$this.hashTable|  and
28  *            <pf> = $this.hashTable[i, i+1))
29  *   (|pf|)
30  * </pre>
31  * @correspondence <pre>
32  * this = union i: integer, pf: PARTIAL_FUNCTION
33  *            where (0 <= i  and  i < |$this.hashTable|  and
34  *                   <pf> = $this.hashTable[i, i+1))
35  *          (pf)
36  * </pre>
37  *
38  * @author Kwasi Fosu Bashir Ali
39  *
40  */
41 public class Map4<K, V> extends MapSecondary<K, V> {
42
43     /*
44      * Private members
   ------------------------------------------------------------
45      */
46
47     /**
48      * Default size of hash table.
49      */
```

```
50     private static final int DEFAULT_HASH_TABLE_SIZE = 101;
51
52     /**
53      * Buckets for hashing.
54      */
55     private Map<K, V>[] hashTable;
56
57     /**
58      * Total size of abstract {@code this}.
59      */
60     private int size;
61
62     /**
63      * Computes {@code a} mod {@code b} as % should have been defined
   to work.
64      *
65      * @param a
66      *            the number being reduced
67      * @param b
68      *            the modulus
69      * @return the result of a mod b, which satisfies 0 <= {@code mod}
   < b
70      * @requires b > 0
71      * @ensures <pre>
72      * 0 <= mod  and  mod < b  and
73      * there exists k: integer (a = k * b + mod)
74      * </pre>
75      */
76     private static int mod(int a, int b) {
77         assert b > 0 : "Violation of: b > 0";
78 //initialize c
79         int c = 0;
80 //if statement in case 0
81         if (a >= 0) {
82             c = a % b;
83         } else {
84             c = a % b;
85             c += b;
86         }
87         return c;
88     }
89
90     /**
91      * Creator of initial representation.
92      *
93      * @param hashTableSize
94      *            the size of the hash table
```

```
 95      * @requires hashTableSize > 0
 96      * @ensures <pre>
 97      * |$this.hashTable| = hashTableSize  and
 98      * for all i: integer
 99      *     where (0 <= i  and  i < |$this.hashTable|)
100      *   ($this.hashTable[i, i+1) = <{}>)  and
101      * $this.size = 0
102      * </pre>
103      */
104     @SuppressWarnings("unchecked")
105     private void createNewRep(int hashTableSize) {
106         this.hashTable = new Map[hashTableSize];
107
108         for (int i = 0; i < hashTableSize; i++) {
109             this.hashTable[i] = new Map2<K, V>();
110         }
111
112         this.size = 0;
113     }
114
115     /*
116      * Constructors
    ------------------------------------------------------------
117      */
118
119     /**
120      * No-argument constructor.
121      */
122     public Map4() {
123         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
124     }
125
126     /**
127      * Constructor resulting in a hash table of size {@code
    hashTableSize}.
128      *
129      * @param hashTableSize
130      *            size of hash table
131      * @requires hashTableSize > 0
132      * @ensures this = {}
133      */
134     public Map4(int hashTableSize) {
135         this.createNewRep(hashTableSize);
136     }
137
138     /*
139      * Standard methods
```

```
    ------------------------------------------------------------
140      */
141
142     @SuppressWarnings("unchecked")
143     @Override
144     public final Map<K, V> newInstance() {
145         try {
146             return this.getClass().getConstructor().newInstance();
147         } catch (ReflectiveOperationException e) {
148             throw new AssertionError(
149                     "Cannot construct object of type " +
    this.getClass());
150         }
151     }
152
153     @Override
154     public final void clear() {
155         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
156     }
157
158     @Override
159     public final void transferFrom(Map<K, V> source) {
160         assert source != null : "Violation of: source is not null";
161         assert source != this : "Violation of: source is not this";
162         assert source instanceof Map4<?, ?> : ""
163                 + "Violation of: source is of dynamic type Map4<?,?>";
164         Map4<K, V> localSource = (Map4<K, V>) source;
165         this.hashTable = localSource.hashTable;
166         this.size = localSource.size;
167         localSource.createNewRep(DEFAULT_HASH_TABLE_SIZE);
168     }
169
170     /*
171      * Kernel methods
    ------------------------------------------------------------
172      */
173
174     @Override
175     public final void add(K key, V value) {
176         assert key != null : "Violation of: key is not null";
177         assert value != null : "Violation of: value is not null";
178         assert !this.hasKey(key) : "Violation of: key is not in
    DOMAIN(this)";
179
180         int mod = mod(key.hashCode(), this.hashTable.length);
181         Map<K, V> m = this.hashTable[mod];
182         m.add(key, value);
```

```
183         //increment size
184         this.size++;
185     }
186 
187     @Override
188     public final Pair<K, V> remove(K key) {
189         assert key != null : "Violation of: key is not null";
190         assert this.hasKey(key) : "Violation of: key is in
    DOMAIN(this)";
191 
192         int mod = mod(key.hashCode(), this.hashTable.length);
193         Map<K, V> m = this.hashTable[mod];
194         Pair<K, V> pair = m.remove(key);
195         this.size--;
196         //decrement size
197         return pair;
198     }
199 
200     @Override
201     public final Pair<K, V> removeAny() {
202         assert this.size() > 0 : "Violation of: this /= empty_set";
203 
204         int mod = 0;
205         while (this.hashTable[mod].size() == 0) {
206             mod++;
207         }
208         Map<K, V> m = this.hashTable[mod];
209         Pair<K, V> pair = m.removeAny();
210         //decrement size
211         this.size--;
212         return pair;
213     }
214 
215     @Override
216     public final V value(K key) {
217         assert key != null : "Violation of: key is not null";
218         assert this.hasKey(key) : "Violation of: key is in
    DOMAIN(this)";
219 
220         int mod = mod(key.hashCode(), this.hashTable.length);
221         Map<K, V> m = this.hashTable[mod];
222         return m.value(key);
223     }
224 
225     @Override
226     public final boolean hasKey(K key) {
227         assert key != null : "Violation of: key is not null";
```

```
228         int mod = mod(key.hashCode(), this.hashTable.length);
229         Map<K, V> m = this.hashTable[mod];
230         return m.hasKey(key);
231     }
232 
233     @Override
234     public final int size() {
235         return this.size;
236     }
237 
238     @Override
239     public final Iterator<Pair<K, V>> iterator() {
240         return new Map4Iterator();
241     }
242 
243     /**
244      * Implementation of {@code Iterator} interface for {@code Map4}.
245      */
246     private final class Map4Iterator implements Iterator<Pair<K, V>> {
247 
248         /**
249          * Number of elements seen already (i.e., |~this.seen|).
250          */
251         private int numberSeen;
252 
253         /**
254          * Bucket from which current bucket iterator comes.
255          */
256         private int currentBucket;
257 
258         /**
259          * Bucket iterator from which next element will come.
260          */
261         private Iterator<Pair<K, V>> bucketIterator;
262 
263         /**
264          * No-argument constructor.
265          */
266         Map4Iterator() {
267             this.numberSeen = 0;
268             this.currentBucket = 0;
269             this.bucketIterator = Map4.this.hashTable[0].iterator();
270         }
271 
272         @Override
273         public boolean hasNext() {
274             return this.numberSeen < Map4.this.size;
```

```
275         }
276
277         @Override
278         public Pair<K, V> next() {
279             assert this.hasNext() : "Violation of: ~this.unseen /= <>";
280             if (!this.hasNext()) {
281                 throw new NoSuchElementException();
282             }
283             this.numberSeen++;
284             while (!this.bucketIterator.hasNext()) {
285                 this.currentBucket++;
286                 this.bucketIterator =
    Map4.this.hashTable[this.currentBucket]
287                                 .iterator();
288             }
289             return this.bucketIterator.next();
290         }
291
292         @Override
293         public void remove() {
294             throw new UnsupportedOperationException(
295                     "remove operation not supported");
296         }
297     }
298 }
299
```