```
1 import java.util.Iterator;
2 import java.util.NoSuchElementException;
3
4 import components.map.Map;
5 import components.map.Map2;
6 import components.map.MapSecondary;
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 Chloe Feller and Krish Patel
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
 79        int modded = a % b;
 80
 81        /**
 82         * Add b if a is negative.
 83         */
 84        if (modded < 0) {
 85            modded += b;
 86        }
 87
 88        return modded;
 89    }
 90
 91    /**
 92     * Creator of initial representation.
 93     *
 94     * @param hashTableSize
 95     *            the size of the hash table
 96     * @requires hashTableSize > 0
 97     * @ensures <pre>
 98     * |$this.hashTable| = hashTableSize  and
 99     * for all i: integer
100     *     where (0 <= i  and  i < |$this.hashTable|)
101     *   ($this.hashTable[i, i+1) = <{}>)  and
102     * $this.size = 0
103     * </pre>
104     */
105    @SuppressWarnings("unchecked")
106    private void createNewRep(int hashTableSize) {
107        /*
108         * With "new Map<K, V>[...]" in place of "new Map[...]" it does not
109         * compile; as shown, it results in a warning about an unchecked
110         * conversion, though it cannot fail.
111         */
112        this.hashTable = new Map[hashTableSize];
113
114        int i = 0;
115        while (hashTableSize > i) {
116            this.hashTable[i] = new Map2<K, V>();
117            i++;
118        }
```

```
119         this.size = 0;
120     }
121
122     /*
123      * Constructors ------------------------------------------------------------
124      */
125
126     /**
127      * No-argument constructor.
128      */
129     public Map4() {
130
131         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
132
133     }
134
135     /**
136      * Constructor resulting in a hash table of size {@code hashTableSize}.
137      *
138      * @param hashTableSize
139      *            size of hash table
140      * @requires hashTableSize > 0
141      * @ensures this = {}
142      */
143     public Map4(int hashTableSize) {
144         assert hashTableSize > 0 : "Violation of : hashTableSize is not > 0";
145
146         this.createNewRep(hashTableSize);
147
148     }
149
150     /*
151      * Standard methods --------------------------------------------------------
152      */
153
154     @SuppressWarnings("unchecked")
155     @Override
156     public final Map<K, V> newInstance() {
157         try {
158             return this.getClass().getConstructor().newInstance();
159         } catch (ReflectiveOperationException e) {
160             throw new AssertionError(
161                     "Cannot construct object of type " + this.getClass());
162         }
163     }
164
165     @Override
166     public final void clear() {
167         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
168     }
169
170     @Override
171     public final void transferFrom(Map<K, V> source) {
172         assert source != null : "Violation of: source is not null";
173         assert source != this : "Violation of: source is not this";
174         assert source instanceof Map4<?, ?> : ""
175                 + "Violation of: source is of dynamic type Map4<?,?>";
176         /*
177          * This cast cannot fail since the assert above would have stopped
```

```
178          * execution in that case: source must be of dynamic type Map4<?,?>, and
179          * the ?,? must be K,V or the call would not have compiled.
180          */
181         Map4<K, V> localSource = (Map4<K, V>) source;
182         this.hashTable = localSource.hashTable;
183         this.size = localSource.size;
184         localSource.createNewRep(DEFAULT_HASH_TABLE_SIZE);
185     }
186
187     /*
188      * Kernel methods ----------------------------------------------------------
189      */
190
191     @Override
192     public final void add(K key, V value) {
193         assert key != null : "Violation of: key is not null";
194         assert value != null : "Violation of: value is not null";
195         assert !this.hasKey(key) : "Violation of: key is not in DOMAIN(this)";
196
197         int bucket = mod(key.hashCode(), this.hashTable.length);
198         this.hashTable[bucket].add(key, value);
199
200         this.size++;
201     }
202
203     @Override
204     public final Pair<K, V> remove(K key) {
205         assert key != null : "Violation of: key is not null";
206         assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
207
208         int bucket = mod(key.hashCode(), this.hashTable.length);
209
210         this.size--;
211
212         return this.hashTable[bucket].remove(key);
213     }
214
215     @Override
216     public final Pair<K, V> removeAny() {
217         assert this.size() > 0 : "Violation of: this /= empty_set";
218
219         int i = 0;
220         Pair<K, V> removed = null;
221         boolean empty = true;
222
223         while (empty) {
224             if (this.hashTable[i].size() > 0) {
225                 removed = this.hashTable[i].removeAny();
226                 empty = false;
227             }
228             i++;
229         }
230
231         this.size--;
232
233         // This line added just to make the component compilable.
234         return removed;
235     }
236
```

```
237     @Override
238     public final V value(K key) {
239         assert key != null : "Violation of: key is not null";
240         assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
241
242         int findValue = mod(key.hashCode(), this.hashTable.length);
243
244         // This line added just to make the component compilable.
245         return this.hashTable[findValue].value(key);
246     }
247
248     @Override
249     public final boolean hasKey(K key) {
250         assert key != null : "Violation of: key is not null";
251
252         int findValue = mod(key.hashCode(), this.hashTable.length);
253
254         // This line added just to make the component compilable.
255         return this.hashTable[findValue].hasKey(key);
256     }
257
258     @Override
259     public final int size() {
260
261         // This line added just to make the component compilable.
262         return this.size;
263     }
264
265     @Override
266     public final Iterator<Pair<K, V>> iterator() {
267         return new Map4Iterator();
268     }
269
270     /**
271      * Implementation of {@code Iterator} interface for {@code Map4}.
272      */
273     private final class Map4Iterator implements Iterator<Pair<K, V>> {
274
275         /**
276          * Number of elements seen already (i.e., |~this.seen|).
277          */
278         private int numberSeen;
279
280         /**
281          * Bucket from which current bucket iterator comes.
282          */
283         private int currentBucket;
284
285         /**
286          * Bucket iterator from which next element will come.
287          */
288         private Iterator<Pair<K, V>> bucketIterator;
289
290         /**
291          * No-argument constructor.
292          */
293         Map4Iterator() {
294             this.numberSeen = 0;
295             this.currentBucket = 0;
```

```
296             this.bucketIterator = Map4.this.hashTable[0].iterator();
297         }
298 
299         @Override
300         public boolean hasNext() {
301             return this.numberSeen < Map4.this.size;
302         }
303 
304         @Override
305         public Pair<K, V> next() {
306             assert this.hasNext() : "Violation of: ~this.unseen /= <>";
307             if (!this.hasNext()) {
308                 /*
309                  * Exception is supposed to be thrown in this case, but with
310                  * assertion-checking enabled it cannot happen because of assert
311                  * above.
312                  */
313                 throw new NoSuchElementException();
314             }
315             this.numberSeen++;
316             while (!this.bucketIterator.hasNext()) {
317                 this.currentBucket++;
318                 this.bucketIterator = Map4.this.hashTable[this.currentBucket]
319                         .iterator();
320             }
321             return this.bucketIterator.next();
322         }
323 
324         @Override
325         public void remove() {
326             throw new UnsupportedOperationException(
327                     "remove operation not supported");
328         }
329 
330     }
331 
332 }
333 
```