```java
import java.util.Iterator;

/**
 * {@code Map} represented as a hash table using {@code Map}s for the buckets,
 * with implementations of primary methods.
 *
 * @param <K>
 *            type of {@code Map} domain (key) entries
 * @param <V>
 *            type of {@code Map} range (associated value) entries
 * @convention <pre>
 * |$this.hashTable| > 0  and
 * for all i: integer, pf: PARTIAL_FUNCTION, x: K
 *     where (0 <= i  and  i < |$this.hashTable|  and
 *            <pf> = $this.hashTable[i, i+1)  and
 *            x is in DOMAIN(pf))
 *   ([computed result of x.hashCode()] mod |$this.hashTable| = i))  and
 * for all i: integer
 *     where (0 <= i  and  i < |$this.hashTable|)
 *   ([entry at position i in $this.hashTable is not null])  and
 * $this.size = sum i: integer, pf: PARTIAL_FUNCTION
 *     where (0 <= i  and  i < |$this.hashTable|  and
 *            <pf> = $this.hashTable[i, i+1))
 *   (|pf|)
 * </pre>
 * @correspondence <pre>
 * this = union i: integer, pf: PARTIAL_FUNCTION
 *            where (0 <= i  and  i < |$this.hashTable|  and
 *                   <pf> = $this.hashTable[i, i+1))
 *          (pf)
 * </pre>
 *
 * @author Put your name here
 *
 */
public class Map4<K, V> extends MapSecondary<K, V> {

    /*
     * Private members --------------------------------------------------------
     */

    /**
     * Default size of hash table.
     */
    private static final int DEFAULT_HASH_TABLE_SIZE = 101;

    /**
     * Buckets for hashing.
     */
    private Map<K, V>[] hashTable;

    /**
     * Total size of abstract {@code this}.
     */
    private int size;

    /**
```

```
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         int mod = a % b;
79         if (mod < 0) {
80             mod = mod + b;
81         }
82         return mod;
83     }
84
85     /**
86      * Creator of initial representation.
87      *
88      * @param hashTableSize
89      *            the size of the hash table
90      * @requires hashTableSize > 0
91      * @ensures <pre>
92      * |$this.hashTable| = hashTableSize  and
93      * for all i: integer
94      *     where (0 <= i  and  i < |$this.hashTable|)
95      *   ($this.hashTable[i, i+1) = <{}>)  and
96      * $this.size = 0
97      * </pre>
98      */
99     @SuppressWarnings("unchecked")
100    private void createNewRep(int hashTableSize) {
101        /*
102         * With "new Map<K, V>[...]" in place of "new Map[...]" it does not
103         * compile; as shown, it results in a warning about an unchecked
104         * conversion, though it cannot fail.
105         */
106        this.hashTable = new Map[hashTableSize];
107        this.size = 0;
108
109        for (int i = 0; i < this.hashTable.length; i++) {
110            this.hashTable[i] = new Map2<K, V>();
111        }
112
113    }
114
115    /*
116     * Constructors ------------------------------------------------------------
117     */
118
119    /**
```

```java
     * No-argument constructor.
     */
    public Map4() {
        this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
    }

    /**
     * Constructor resulting in a hash table of size {@code hashTableSize}.
     *
     * @param hashTableSize
     *            size of hash table
     * @requires hashTableSize > 0
     * @ensures this = {}
     */
    public Map4(int hashTableSize) {
        assert hashTableSize > 0 : "Violation of: hashTableSize > 0";
        this.createNewRep(hashTableSize);
    }

    /*
     * Standard methods ------------------------------------------------------
     */

    @SuppressWarnings("unchecked")
    @Override
    public final Map<K, V> newInstance() {
        try {
            return this.getClass().getConstructor().newInstance();
        } catch (ReflectiveOperationException e) {
            throw new AssertionError(
                    "Cannot construct object of type " + this.getClass());
        }
    }

    @Override
    public final void clear() {
        this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
    }

    @Override
    public final void transferFrom(Map<K, V> source) {
        assert source != null : "Violation of: source is not null";
        assert source != this : "Violation of: source is not this";
        assert source instanceof Map4<?, ?> : ""
                + "Violation of: source is of dynamic type Map4<?,?>";
        /*
         * This cast cannot fail since the assert above would have stopped
         * execution in that case: source must be of dynamic type Map4<?,?>, and
         * the ?,? must be K,V or the call would not have compiled.
         */
        Map4<K, V> localSource = (Map4<K, V>) source;
        this.hashTable = localSource.hashTable;
        this.size = localSource.size;
        localSource.createNewRep(DEFAULT_HASH_TABLE_SIZE);
    }

    /*
```

```
177      * Kernel methods ----------------------------------------------------------
178      */
179
180     @Override
181     public final void add(K key, V value) {
182         assert key != null : "Violation of: key is not null";
183         assert value != null : "Violation of: value is not null";
184         assert !this.hasKey(key) : "Violation of: key is not in DOMAIN(this)";
185
186         int bucket = mod(key.hashCode(), this.hashTable.length);
187         this.hashTable[bucket].add(key, value);
188         this.size += 1;
189
190     }
191
192     @Override
193     public final Pair<K, V> remove(K key) {
194         assert key != null : "Violation of: key is not null";
195         assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
196
197         int bucket = mod(key.hashCode(), this.hashTable.length);
198         this.size -= 1;
199         return this.hashTable[bucket].remove(key);
200     }
201
202     @Override
203     public final Pair<K, V> removeAny() {
204         assert this.size() > 0 : "Violation of: this /= empty_set";
205
206         int bucket = 0;
207
208         for (int i = 0; i < this.hashTable.length; i++) {
209             if (this.hashTable[i].size() > 0) {
210                 bucket = i;
211                 i = this.hashTable.length;
212             }
213         }
214
215         this.size -= 1;
216         return this.hashTable[bucket].removeAny();
217     }
218
219     @Override
220     public final V value(K key) {
221         assert key != null : "Violation of: key is not null";
222         assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
223
224         int bucket = mod(key.hashCode(), this.hashTable.length);
225         return this.hashTable[bucket].value(key);
226     }
227
228     @Override
229     public final boolean hasKey(K key) {
230         assert key != null : "Violation of: key is not null";
231
232         int bucket = mod(key.hashCode(), this.hashTable.length);
233         return this.hashTable[bucket].hasKey(key);
```

```
234     }
235
236     @Override
237     public final int size() {
238         return this.size;
239     }
240
241     @Override
242     public final Iterator<Pair<K, V>> iterator() {
243         return new Map4Iterator();
244     }
245
246     /**
247      * Implementation of {@code Iterator} interface for {@code Map4}.
248      */
249     private final class Map4Iterator implements Iterator<Pair<K, V>> {
250
251         /**
252          * Number of elements seen already (i.e., |~this.seen|).
253          */
254         private int numberSeen;
255
256         /**
257          * Bucket from which current bucket iterator comes.
258          */
259         private int currentBucket;
260
261         /**
262          * Bucket iterator from which next element will come.
263          */
264         private Iterator<Pair<K, V>> bucketIterator;
265
266         /**
267          * No-argument constructor.
268          */
269         Map4Iterator() {
270             this.numberSeen = 0;
271             this.currentBucket = 0;
272             this.bucketIterator = Map4.this.hashTable[0].iterator();
273         }
274
275         @Override
276         public boolean hasNext() {
277             return this.numberSeen < Map4.this.size;
278         }
279
280         @Override
281         public Pair<K, V> next() {
282             assert this.hasNext() : "Violation of: ~this.unseen /= <>";
283             if (!this.hasNext()) {
284                 /*
285                  * Exception is supposed to be thrown in this case, but with
286                  * assertion-checking enabled it cannot happen because of assert
287                  * above.
288                  */
289                 throw new NoSuchElementException();
290             }
```

```
291             this.numberSeen++;
292             while (!this.bucketIterator.hasNext()) {
293                 this.currentBucket++;
294                 this.bucketIterator = Map4.this.hashTable[this.currentBucket]
295                         .iterator();
296             }
297             return this.bucketIterator.next();
298         }
299 
300         @Override
301         public void remove() {
302             throw new UnsupportedOperationException(
303                     "remove operation not supported");
304         }
305 
306     }
307 
308 }
309 
```