CSE 2231 – Software 2: Software Development and Design

Professor: Rob LaTour

Project #3: Hashing Implementation of Map

The Ohio State University

College of Engineering

Columbus, Ohio

```
import java.util.Iterator;
import java.util.NoSuchElementException;

import components.map.Map;
import components.map.Map2;
import components.map.MapSecondary;

/**
 * {@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 Danny Kan (kan.74@osu.edu)
* @author Jatin Mamtani (mamtani.6@osu.edu)
*
*/
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;

  /**
   * Computes {@code a} mod {@code b} as % should have been defined to work.
```

```
 *
 * @param a
 *            the number being reduced
 * @param b
 *            the modulus
 * @return the result of a mod b, which satisfies 0 <= {@code mod} < b
 * @requires b > 0
 * @ensures <pre>
 * 0 <= mod  and  mod < b  and
 * there exists k: integer (a = k * b + mod)
 * </pre>
 */
private static int mod(int a, int b) {
    assert b > 0 : "Violation of: b > 0";
    int r = a % b; // returns the remainder.
    // ex.1) r = ((a = -30) mod (b = 10)) = 0.
    // ex.2) r = ((a = -32) mod (b = 10)) = -2.
    if ((a < 0) && (r != 0)) {
        // we should take the absolute value of r and subtract it from b.
        r = b - (-1 * r); // note: r = b - (-1 * r) = b - Math.abs(r).
    }
    return r;
}

/**
 * Creator of initial representation.
 *
 * @param hashTableSize
 *            the size of the hash table
 * @requires hashTableSize > 0
 * @ensures <pre>
 * |$this.hashTable| = hashTableSize  and
```

```
 * for all i: integer
 *     where (0 <= i  and  i < |$this.hashTable|)
 *   ($this.hashTable[i, i+1) = <{}>)  and
 * $this.size = 0
 * </pre>
 */
@SuppressWarnings("unchecked")
private void createNewRep(int hashTableSize) {
    /*
     * With "new Map<K, V>[...]" in place of "new Map[...]" it does not
     * compile; as shown, it results in a warning about an unchecked
     * conversion, though it cannot fail.
     */
    this.hashTable = new Map[hashTableSize];
    this.size = 0;
    int i = 0;
    while (i < hashTableSize) {
        this.hashTable[i] = new Map2<K, V>();
        i++;
    }
}

/*
 * Constructors ------------------------------------------------------------
 */

/**
 * 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) {
    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);
}

/*
 * Kernel methods ----------------------------------------------------------
 */

@Override
public final void add(K key, V value) {
    assert key != null : "Violation of: key is not null";
    assert value != null : "Violation of: value is not null";
    assert !this.hasKey(key) : "Violation of: key is not in DOMAIN(this)";

    this.size++;
    this.hashTable[mod(key.hashCode(), this.hashTable.length)].add(key,
            value);
}
```

```
@Override
public final Pair<K, V> remove(K key) {
    assert key != null : "Violation of: key is not null";
    assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";

    this.size--;
    return this.hashTable[mod(key.hashCode(), this.hashTable.length)]
            .remove(key);
}

@Override
public final Pair<K, V> removeAny() {
    assert this.size() > 0 : "Violation of: this /= empty_set";

    this.size--;
    int i = 0;
    // to ensure we do not stop at an empty bucket.
    while (i < this.hashTable.length && this.hashTable[i].size() == 0) {
        i++;
    }
    return this.hashTable[i].removeAny();
}

@Override
public final V value(K key) {
    assert key != null : "Violation of: key is not null";
    assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";

    return this.hashTable[mod(key.hashCode(), this.hashTable.length)]
            .value(key);
}
```

```
@Override
public final boolean hasKey(K key) {
    assert key != null : "Violation of: key is not null";

    return this.hashTable[mod(key.hashCode(), this.hashTable.length)]
            .hasKey(key);
}

@Override
public final int size() {
    return this.size;
}

@Override
public final Iterator<Pair<K, V>> iterator() {
    return new Map4Iterator();
}

/**
 * Implementation of {@code Iterator} interface for {@code Map4}.
 */
private final class Map4Iterator implements Iterator<Pair<K, V>> {

    /**
     * Number of elements seen already (i.e., |~this.seen|).
     */
    private int numberSeen;

    /**
     * Bucket from which current bucket iterator comes.
     */
```

```
private int currentBucket;

/**
 * Bucket iterator from which next element will come.
 */
private Iterator<Pair<K, V>> bucketIterator;

/**
 * No-argument constructor.
 */
Map4Iterator() {
    this.numberSeen = 0;
    this.currentBucket = 0;
    this.bucketIterator = Map4.this.hashTable[0].iterator();
}

@Override
public boolean hasNext() {
    return this.numberSeen < Map4.this.size;
}

@Override
public Pair<K, V> next() {
    assert this.hasNext() : "Violation of: ~this.unseen /= <>";
    if (!this.hasNext()) {
        /*
         * Exception is supposed to be thrown in this case, but with
         * assertion-checking enabled it cannot happen because of assert
         * above.
         */
        throw new NoSuchElementException();
    }
```

```
            this.numberSeen++;
            while (!this.bucketIterator.hasNext()) {
                this.currentBucket++;
                this.bucketIterator = Map4.this.hashTable[this.currentBucket]
                        .iterator();
            }
            return this.bucketIterator.next();
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException(
                    "remove operation not supported");
        }

    }

}
```