**public** **class** SeparateChainingHashST<Key, Value> {

**int** N; // number of key-value pairs

**int** M; // hash table size

SequentialSearchST<Key, Value>[] st; // array of linked-list symbol tables

**final** **static** **int** ***INIT\_CAPACITY*** = 11; // initial default size

**final** **static** **int** ***AVG\_LENGTH*** = 7; // Threshold to determine resizing

**public** SeparateChainingHashST(**int** M) {

**this**.M = M;

st = (SequentialSearchST<Key, Value>[]) **new** SequentialSearchST[M];

**for** (**int** i = 0; i < M; i++) {

st[i] = **new** SequentialSearchST<Key, Value>();

}

}

// resize hash table to have given number of chains by rehashing all of the keys

**void** resize(**int** chains) {

SeparateChainingHashST<Key, Value> temp = **new** SeparateChainingHashST<Key, Value>(chains);

**for** (**int** i = 0; i < M; i++) {

**for** (Key key : st[i]) { // NOTE ITERATOR OVER KEYS

temp.put(key, st[i].get(key));

}

}

M = temp.M;

N = temp.N;

st = temp.st;

}

// hash value between 0 and M-1

**int** hash(Key key) { **return** (key.hashCode() & 0x7fffffff) % M; }

**public** **int** size() { **return** N; }

**public** **boolean** isEmpty() { **return** size() == 0; }

**public** **boolean** contains(Key key) { **return** get(key) != **null**; }

**public** Value get(Key key) {

**int** i = hash(key);

**return** st[i].get(key);

}

**public** **void** put(Key key, Value val) {

// double table size if average length of list >= AVG\_LENGTH

**if** (N >= ***AVG\_LENGTH***\*M) resize(2\*M);

**int** i = hash(key);

**if** (!st[i].contains(key)) N++;

st[i].put(key, val);

}

**public** **void** delete(Key key) {

**int** i = hash(key);

**if** (st[i].contains(key)) { N--; }

st[i].delete(key);

// halve table size if average length of list <= 2M

**if** (M > ***INIT\_CAPACITY*** && N <= 2\*M) resize(M/2);

}