Practice 01:

Implementation of hash table using separate chaining

Code:

**package** SeperateChaining;

**import** java.io.IOException;

**class** Link {

**private** **int** data;

**public** Link next;

**public** Link(**int** d) {

data = d;

}

**public** **int** getKey() {

**return** data;

}

**public** **void** displayLink() {

System.***out***.print(data + " ");

}

};

**package** SeperateChaining;

**import** java.io.IOException;

**class** SortedList {

**private** Link first;

**public** SortedList() {

first = **null**;

}

**public** **void** insert(Link theLink){

**int** key = theLink.getKey();

Link previous = **null**; // start at first

Link current = first;

// until end of list,

//or current bigger than key,

**while** (current != **null** && key > current.getKey()) {

previous = current;

current = current.next; // go to next item

}

**if** (previous == **null**) // if beginning of list,

first = theLink;

**else**

// not at beginning,

previous.next = theLink;

theLink.next = current;

}

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

Link previous = **null**;

Link current = first;

**while** (current != **null** && key != current.getKey()) {

previous = current;

current = current.next;

}

// disconnect link

**if** (previous == **null**) // if beginning of list delete first link

first = first.next;

**else**

// not at beginning

previous.next = current.next; //delete current link

}

**public** Link find(**int** key) {

Link current = first;

**while** (current != **null** && current.getKey() <= key) { // or key too small,

**if** (current.getKey() == key) // found, return link

**return** current;

current = current.next; // go to next item

}

**return** **null**; // cannot find it

}

**public** **void** displayList() {

System.***out***.print("List: ");

Link current = first;

**while** (current != **null**){

current.displayLink();

current = current.next;

}

System.***out***.println("");

}

};

**package** SeperateChaining;

**import** java.io.IOException;

**public** **class** HashChain {

**private** SortedList[] hashArray;

**private** **int** arraySize;

**public** HashChain(**int** size) {

arraySize = size;

hashArray = **new** SortedList[arraySize];

**for** (**int** i = 0; i < arraySize; i++)

hashArray[i] = **new** SortedList();

}

**public** **void** displayTable() {

**for** (**int** j = 0; j < arraySize; j++) {

System.***out***.print(j + ". ");

hashArray[j].displayList();

}

}

**public** **int** hashFunc(**int** key) {

**return** key % arraySize;

}

**public** **void** insert(Link theLink) {

**int** key = theLink.getKey();

**int** hashVal = hashFunc(key);

hashArray[hashVal].insert(theLink);

}

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

**int** hashVal = hashFunc(key); // hash the key

hashArray[hashVal].delete(key);

}

**public** Link find(**int** key) {

**int** hashVal = hashFunc(key); // hash the key

Link theLink = hashArray[hashVal].find(key); // get link

**return** theLink;

}

**public** **static** **void** main(String[] args) **throws** IOException {

**int** aKey;

Link dataItem;

**int** size, initSize, keysPerCell = 100;

size = 100;

initSize = 10;

HashChain hashTable = **new** HashChain(size);

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

aKey = (**int**) (java.lang.Math.*random*() \* keysPerCell \* size);

dataItem = **new** Link(aKey);

hashTable.insert(dataItem);

}

hashTable.displayTable();

aKey = 100;

dataItem = **new** Link(aKey);

hashTable.insert(dataItem);

aKey = 100;

hashTable.delete(aKey);

aKey = 50;

dataItem = hashTable.find(aKey);

**if** (dataItem != **null**)

System.***out***.println("Found " + aKey);

**else**

System.***out***.println("Could not find " + aKey);

}

}