Practical 10

1. HashCode

```
//Ming Yeu
private static int h;

public int hashCode() { // for testing ***************
    // this hash code causes collisions
    //int h = 0;

    for (int i = 0; i < first.length(); i++) {
      h = h + first.charAt(i) * i;
    }
    return h;
} // end hashCode
```

2. Modify the class **Student** such that the **id** is type **long** and comprises a 16-digit number. Then, modify the implementation of the method **hashCode** such that the *folding technique* is applied as follows:

- break the key into groups of digits
- then combine the groups using addition

```
//Raphael

private String id;

public Student(){
      String id= “”;
}
```

3. Modify the class **HashedDictionary** such that the collision-resolution scheme used is *open addressing with double hashing*. Use the following primary and secondary hash functions:

```
private int getHashIndex(K key){
      int hashIndex= key.hashCode() % hashTable.length;
      if(hashIndex < 0)
            hashIndex = hashIndex + hashTable.length;

      return hashIndex;
}

private int getSecondHashIndex(K key){
      return 7 - key.hashCode() % 7;
}


//Yeu Yang
private int probe(int index, K key) {
  boolean found = false;
  int removedStateIndex = -1; // index of first location in
```

```
    // removed state

    int h2 = getSecondHashIndex(key);

    while (!found && (hashTable[index] != null)) {
        if (hashTable[index].isIn()) {
            if (key.equals(hashTable[index].getKey())) {
                found = true; // key found
            } else // follow probe sequence
            {
                    //double hashing
                    index = (index + h2) % hashTable.length;
            }
        } else { // skip entries that were removed
            // save index of first location in removed state
            if (removedStateIndex == -1) {
                removedStateIndex = index;
            }
              //double hashing
            index = (index + h2) % hashTable.length;
        } // end if
    } // end while
    // Assertion: either key or null is found at hashTable[index]

    if (found || (removedStateIndex == -1)) {
        return index;             // index of either key or null
    } else {
        return removedStateIndex; // index of an available location
    }
} // end probe
```