**Objective**:

To compare the three Open addressing methods: Linear probing, Quadratic probing and Double hashing by inserting the keys into the Hash table of size m for m=1223, 1831, 2447 and empirically determine the total number of collisions taken by each of the three methods thereby comparing which method performs best with fewest collisions on inputs of 1100 random natural numbers between 1 and 10,000(inclusive) as keys.

**Procedure:**

1. Create a Hashing program package with three classes (LinearProbing, QuadraticProbing, DoubleHashing) in Eclipse
2. Determine the value of total number of collisions for each open addressing method.
3. Check which method performs well for different values of m.

**Technology used:** Java

**Tool used:** Eclipse

**Source Code Filename:** LinearProbing.java, QuadraticProbing.java, DoubleHashing.java

**Explanation:**

A hash table generalizes the simpler notion of an ordinary array. When the number of keys actually stored is small relative to the total number of possible keys (here in our project, no. of keys=1100 and possible keys=size of hash table= 1223), hash tables become an effective alternative to directly addressing an array, since a hash table typically uses an array of size proportional to the number of keys actually stored. Instead of using the key as an array index directly, the array index is computedfrom the key. We can compute array indices from keys using hash functions (different hash functions for three methods here in our project, Linear probing: h(k, i) = ((k) + i) % m. Quadratic probing: h(k, i) = ( (k) + ) % m. Double hashing: h(k, i) = ((k) + i\*(k)) % m). Open addressing is a way to deal with collisions, in this method all elements occupy the hash table itself. That is, each table entry contains either an element of the dynamic set or NIL. The advantage of open addressing is that it avoids pointers altogether. Instead of following pointers, we computethe sequence of slots to be examined. The extra memory freed by not storing pointers provides the hash table with a larger number of slots for the same amount of memory, potentially yielding fewer collisions and faster retrieval.

The Hashing program is explained below.

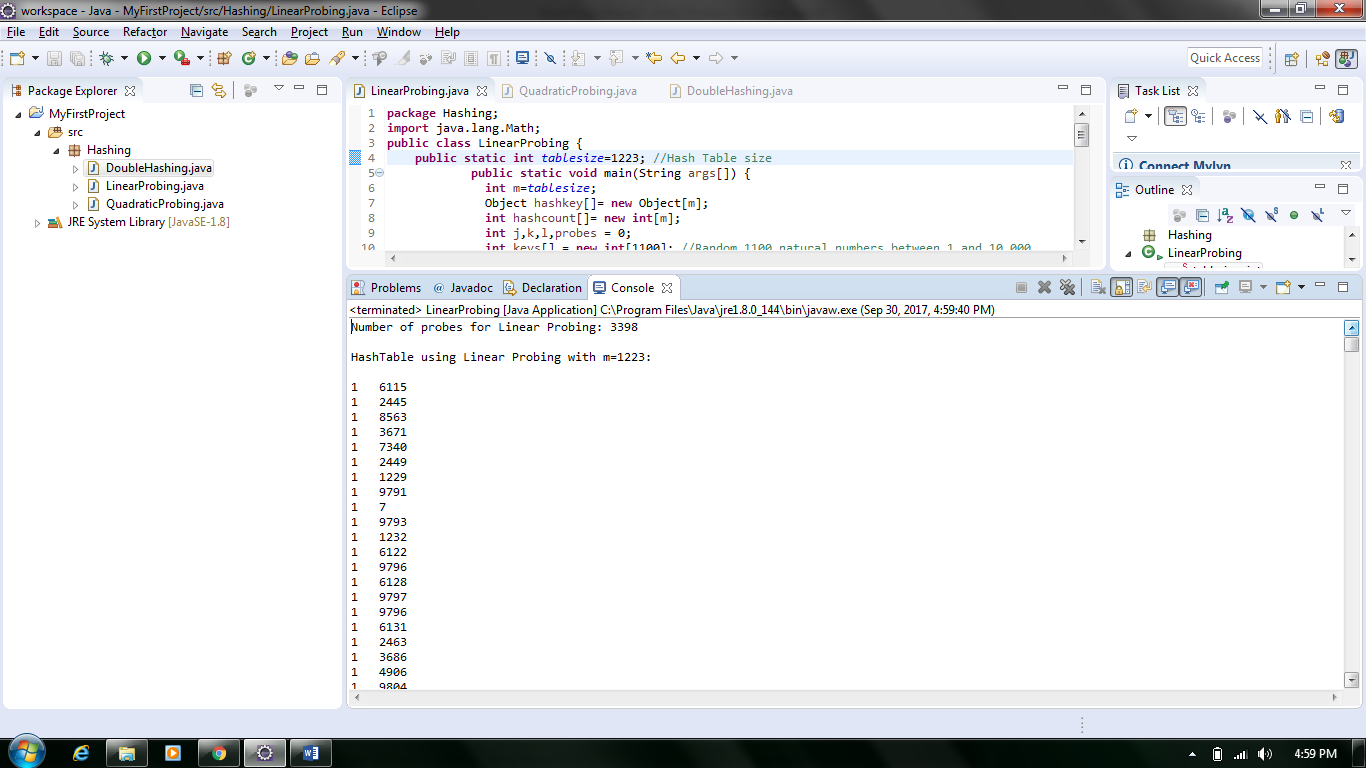
**Program Explanation:**

1. Create a java package “hashing”
2. Create three different classes for each of the addressing methods i.e., LinearProbing, QuadraticProbing, DoubleHashing
3. Create a main method.
4. Initialize the variables.(Here the Hash key is assumed as the Object type array because if the table element doesn’t contain any key then it should be displayed as NULL instead of zero).
5. Using Random function (call Math.Random()) generate 1100 keys between 1 and 10,000.
6. All the steps till here are common for all the three methods but the main difference is the hash function used to insert the keys into the table. Define the hash function.
7. If the position in the table is NIL then insert the key and break from the while loop or else if the position contains the same value of the key to be inserted then increase the count of the key to 1 and break from the while loop or else the position contains value other than the key to be inserted then increase the collisions count to 1 and continue with while loop.
8. After all keys are inserted in to the Hash table (of size m=1223). Now print the number of collisions and the Hash table
9. Repeat the Same procedure by changing the table size (static variable value) to m=1831, m=2447.
10. The method for which the fewest number of collisions generated is considered as the effective method.

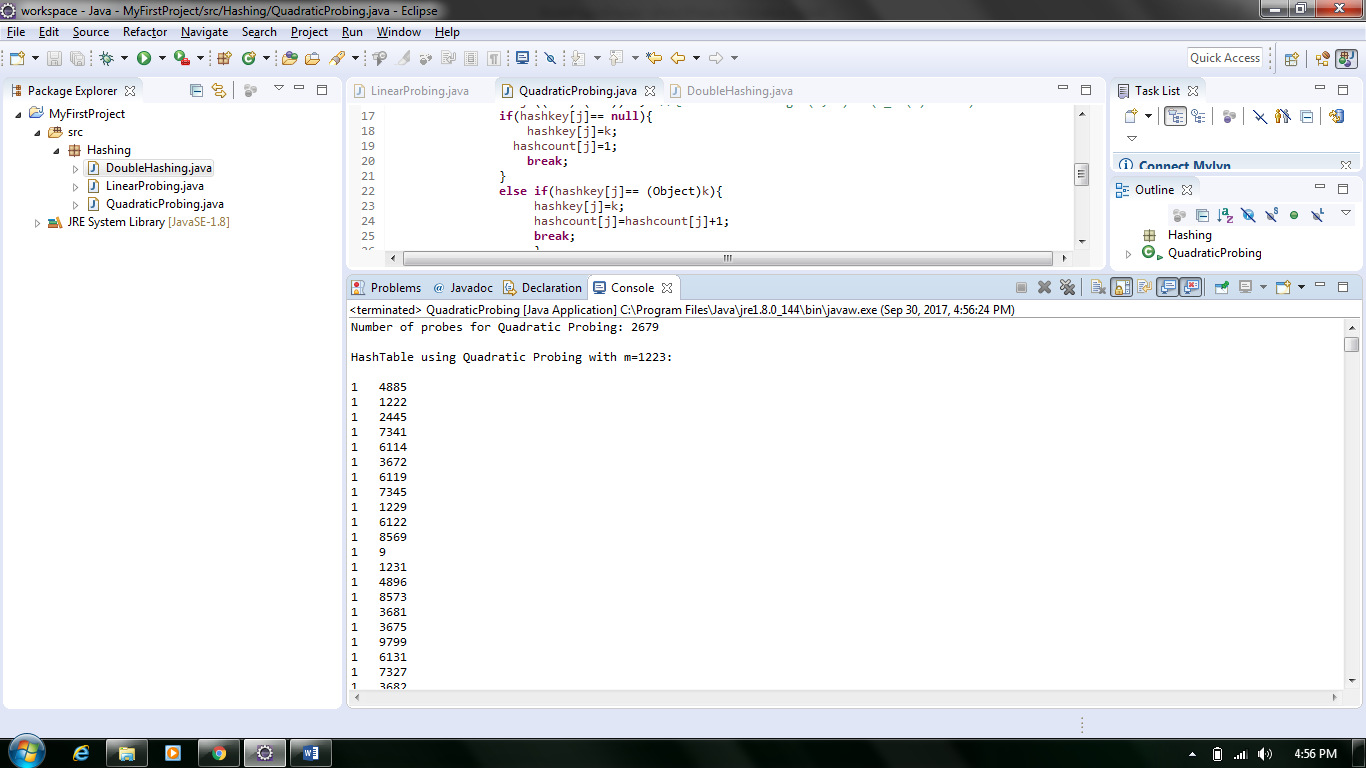
**Screen Shots of output:**

I have used three different classes for the three methods, as the array size is large the screenshots show only the first 20 array elements.

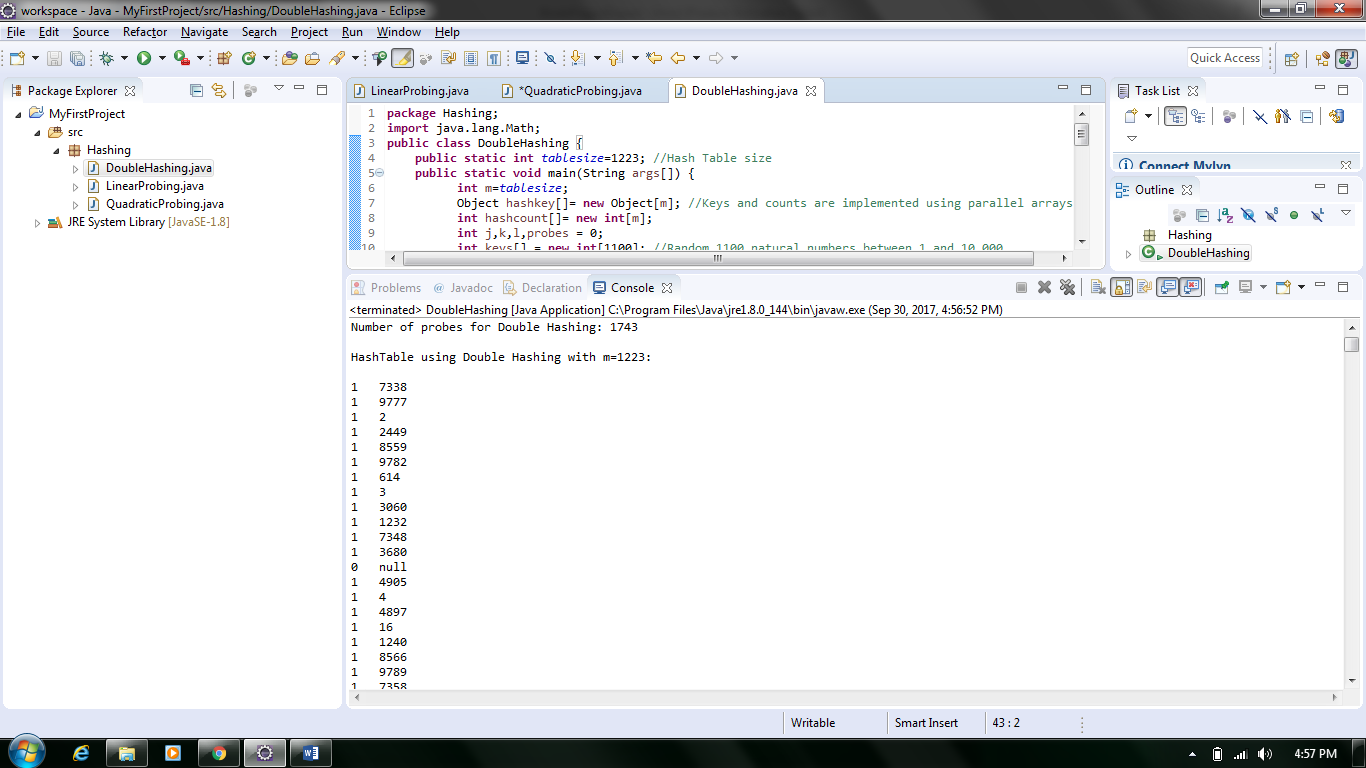
LinearProbing (m=1223):



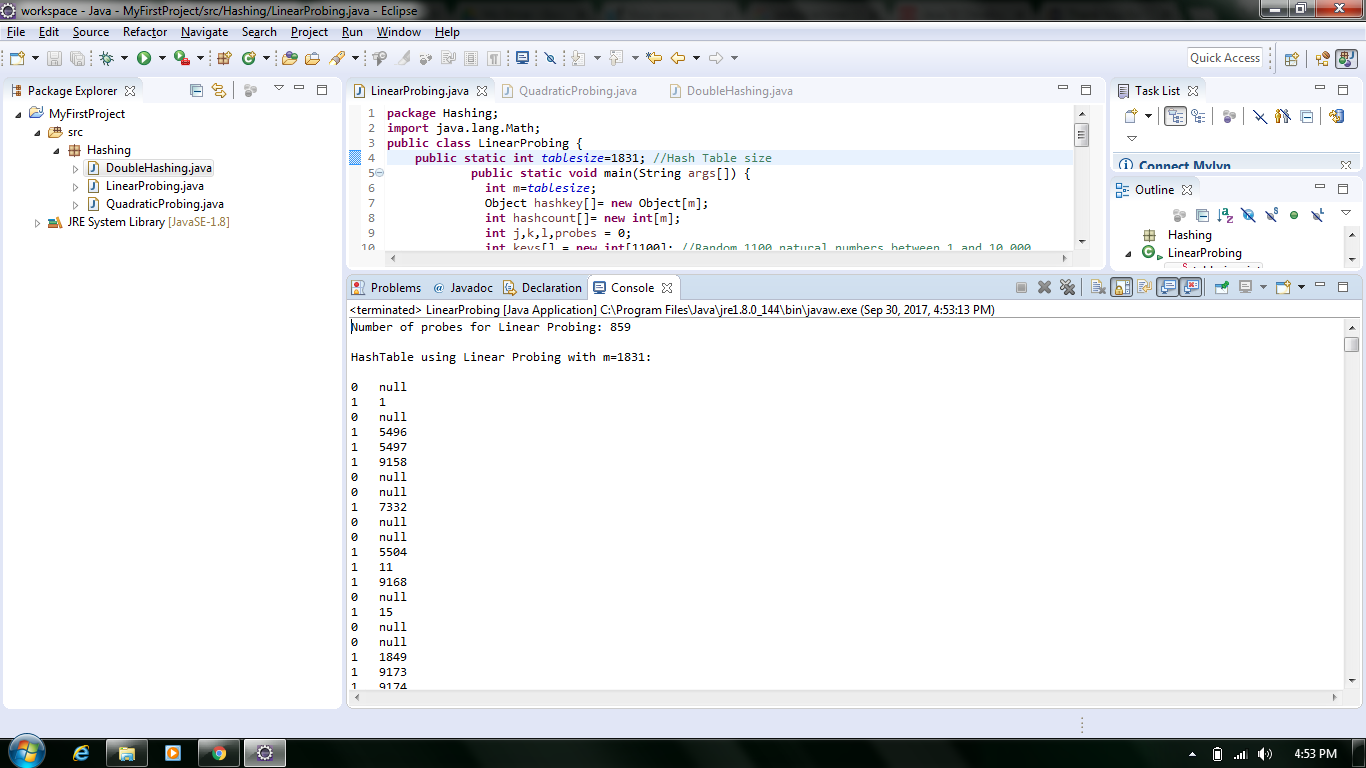
QuadraticProbing (m=1223):



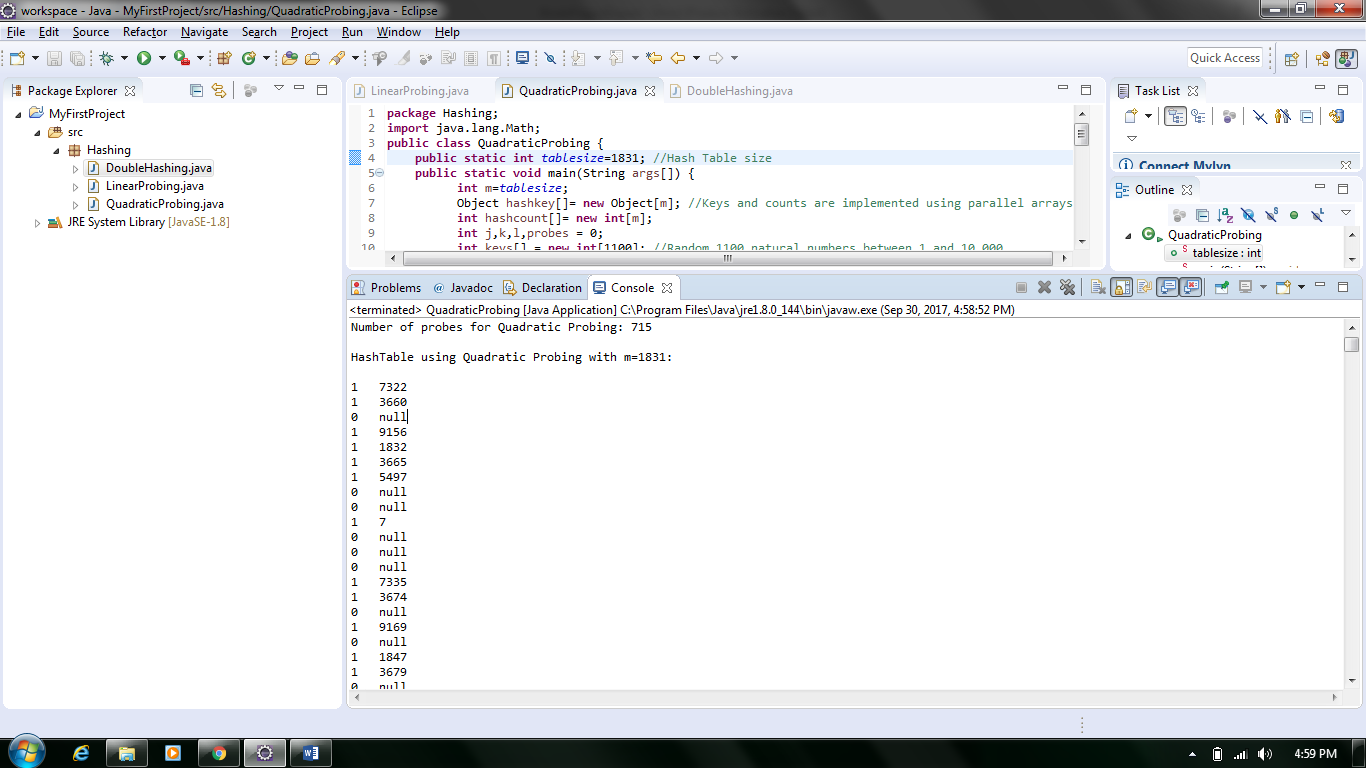
DoubleHashing (m=1223):



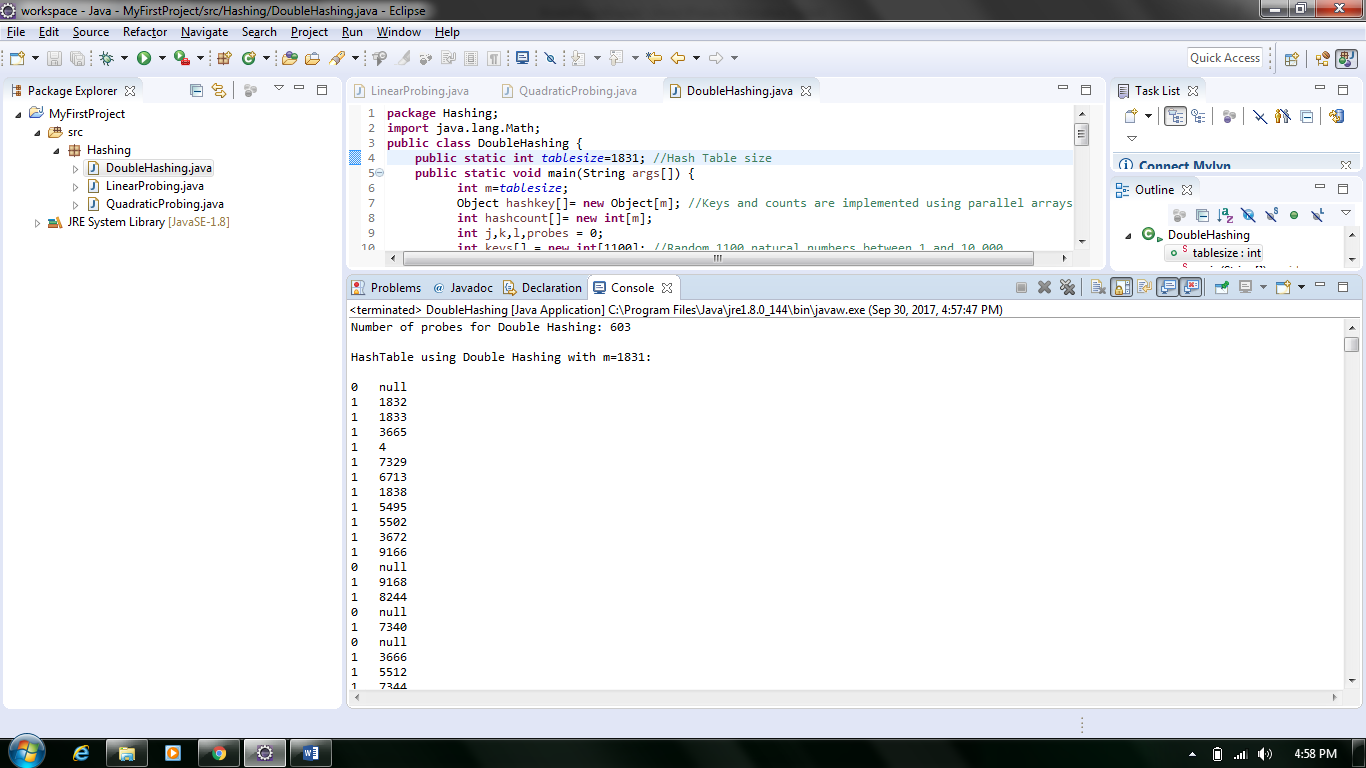
LinearProbing (m=1831):



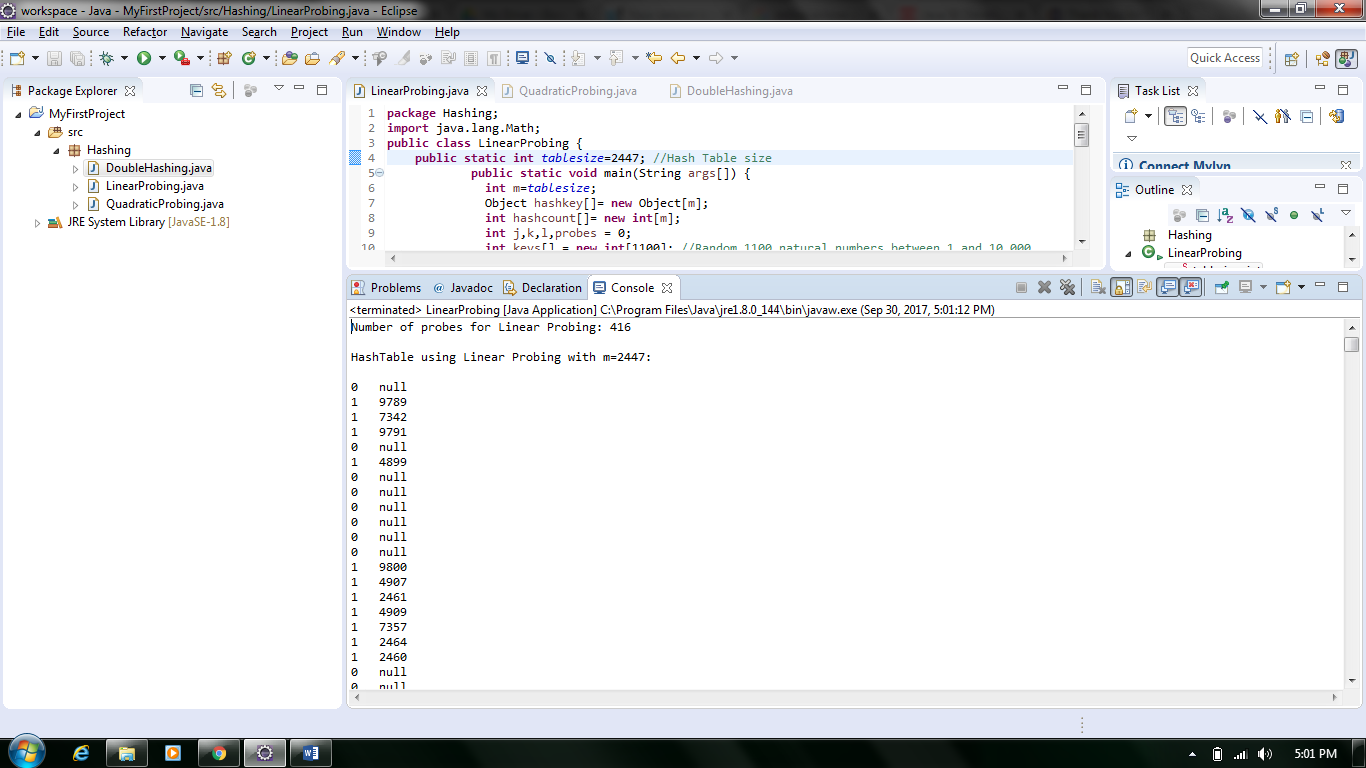
QuadraticProbing (m=1831):



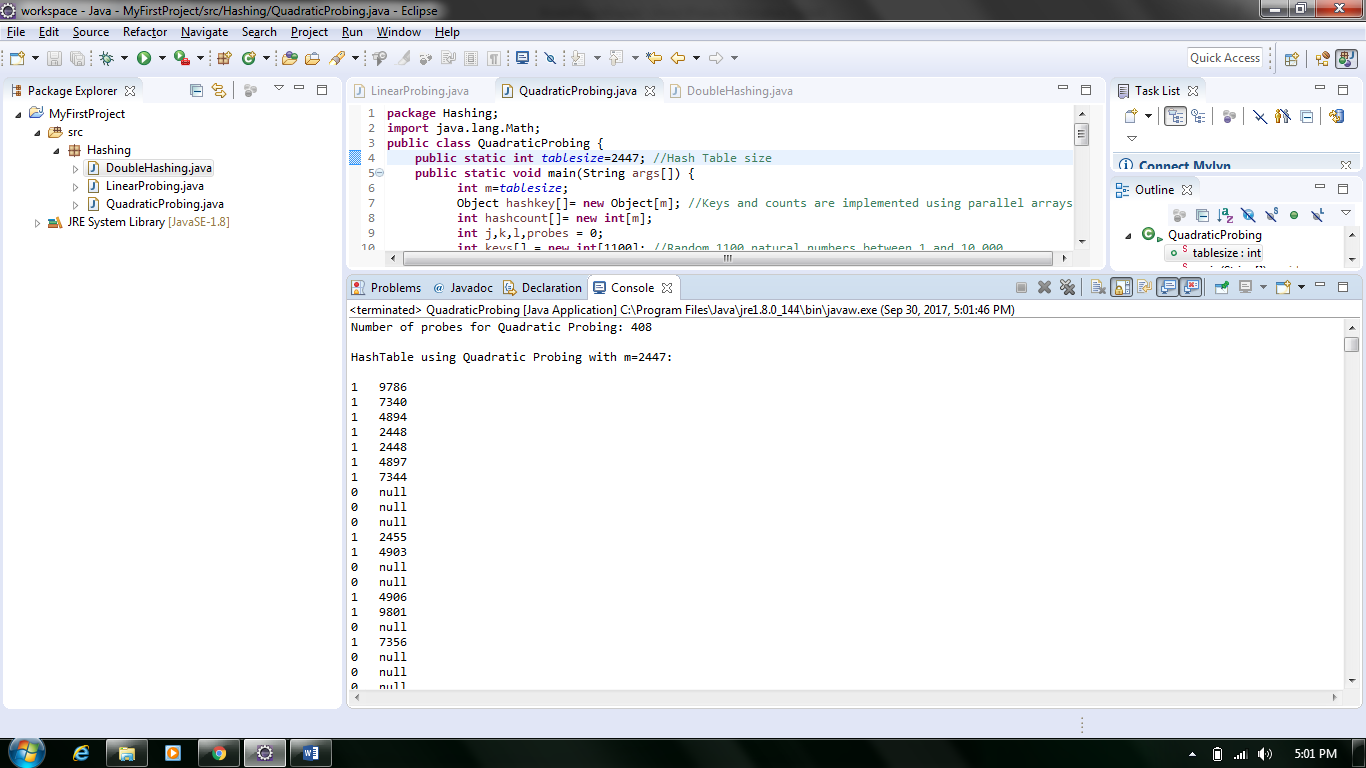
DoubleHashing (m=1831):



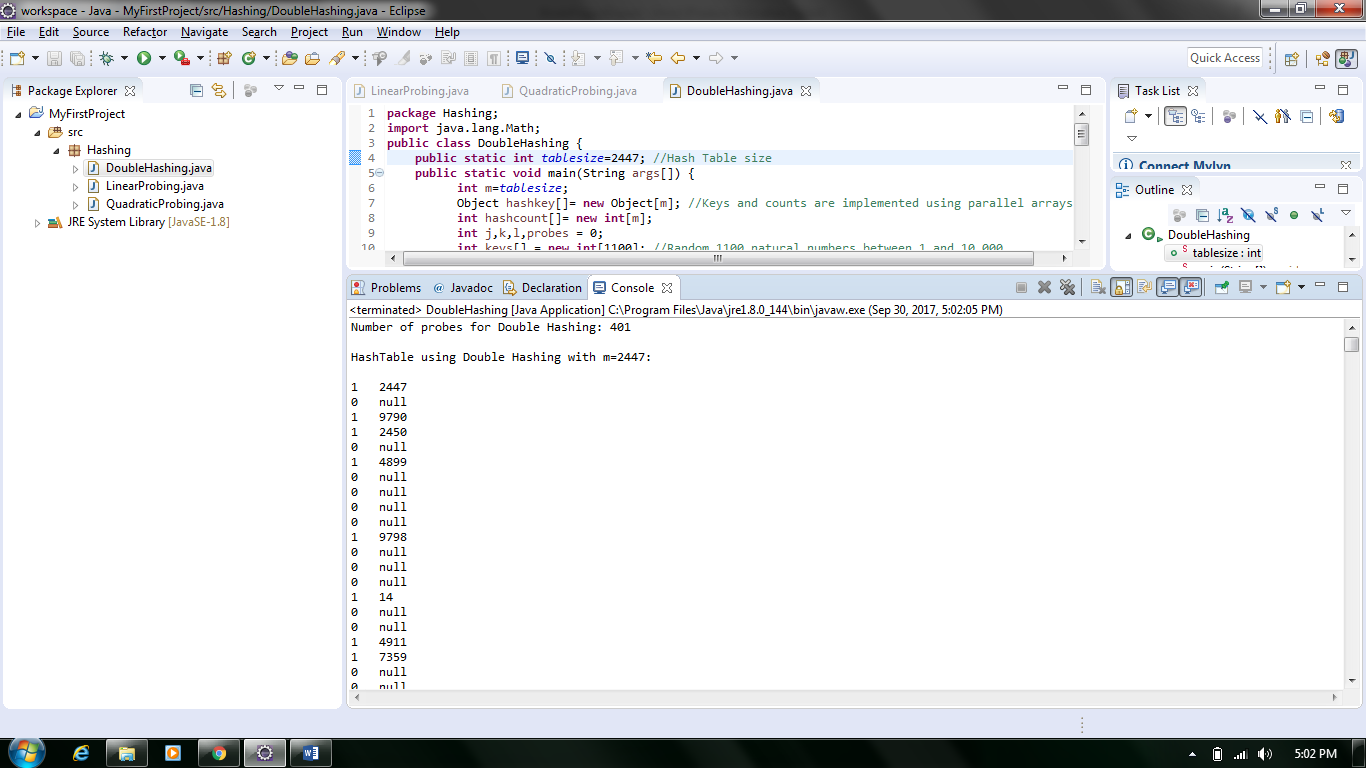
LinearProbing (m=2447):



QuadraticProbing (m=2447):



DoubleProbing (m=2447):



**Observations:**

As per the tabular column below, the minimum number of collisions for table size ‘m’ value in this case is 1223 is for the Double Hashing method. As the table size is more close to the no. of keys to be inserted the Double Hashing method is effective (if the table is size is limited but greater than the no. of keys).

No. of collisions for LinearProbing < No. of collisions for QuadraticProbing < No. of collisions for DoubleHashing.

From the above observation we can say that for every change in value of table size m the output trends are different. If the table size is m=1831, All methods perform slightly in a same way i.e., their no. of collisions are more or less comparable. No. of collisions for LinearProbing is slightly less than or equal to No. of collisions for QuadraticProbing and is slightly less than No. of collisions for DoubleHashing.

If the table size is large enough (2447) then the three methods perform equally efficient.

Below is the listing of the total number of collisions for the all the different values of m (1223, 1831, 2447) for the three methods.

**Tabular column:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | No. Of Collisions |  |
| Table size(m) | Linear Probing | Quadratic Probing | Double Hashing |
| 1223 | 3398 | 2679 | 1743 |
| 1223 | 3727 | 3899 | 1841 |
| 1223 | 9264 | 3692 | 1680 |
| 1223 | 3650 | 3499 | 1609 |
| 1223 | 3813 | 4729 | 1744 |
| 1223 | 5690 | 7502 | 1766 |
| 1223 | 5365 | 4952 | 1918 |
| 1223 | 6333 | 3949 | 1630 |
| 1223 | 7756 | 4172 | 1756 |
| 1223 | 3657 | 4640 | 1735 |
| 1223 | 3224 | 8254 | 1633 |
| 1223 | 7145 | 7884 | 1798 |
| 1223 | 3688 | 3772 | 1679 |
| 1223 | 4237 | 3883 | 1829 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | No. Of Collisions |  |
| Table size(m) | Linear Probing | Quadratic Probing | Double Hashing |
| 1831 | 859 | 715 | 589 |
| 1831 | 826 | 810 | 607 |
| 1831 | 781 | 893 | 603 |
| 1831 | 796 | 735 | 606 |
| 1831 | 978 | 908 | 579 |
| 1831 | 776 | 699 | 562 |
| 1831 | 698 | 806 | 586 |
| 1831 | 808 | 743 | 613 |
| 1831 | 826 | 780 | 579 |
| 1831 | 918 | 688 | 529 |
| 1831 | 769 | 768 | 626 |
| 1831 | 890 | 727 | 614 |
| 1831 | 940 | 737 | 583 |
| 1831 | 829 | 781 | 578 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | No. Of Collisions |  |
| Table size(m) | Linear Probing | Quadratic Probing | Double Hashing |
| 2447 | 416 | 408 | 401 |
| 2447 | 406 | 358 | 387 |
| 2447 | 440 | 423 | 349 |
| 2447 | 448 | 430 | 384 |
| 2447 | 437 | 471 | 380 |
| 2447 | 445 | 392 | 353 |
| 2447 | 428 | 371 | 359 |
| 2447 | 443 | 401 | 369 |
| 2447 | 481 | 390 | 381 |
| 2447 | 441 | 446 | 325 |
| 2447 | 391 | 416 | 364 |
| 2447 | 506 | 496 | 326 |
| 2447 | 477 | 434 | 342 |
| 2447 | 507 | 482 | 390 |

**Conclusion/Inference:**

From the above observation it can be inferred that as the number of collisions decreases the efficiency of the open addressing method increases.