**Java Collection Assignment**

**Java Collection: ArrayList Exercises**

1. Write a Java program to create a new array list, add some Movie names (string) and print out the

collection.

-Write a Java program to insert an element into the array list at the first -position.

-Write a Java program to retrieve an element (at a specified index) from a given array list.

-Write a Java program to update specific array elements by given element.

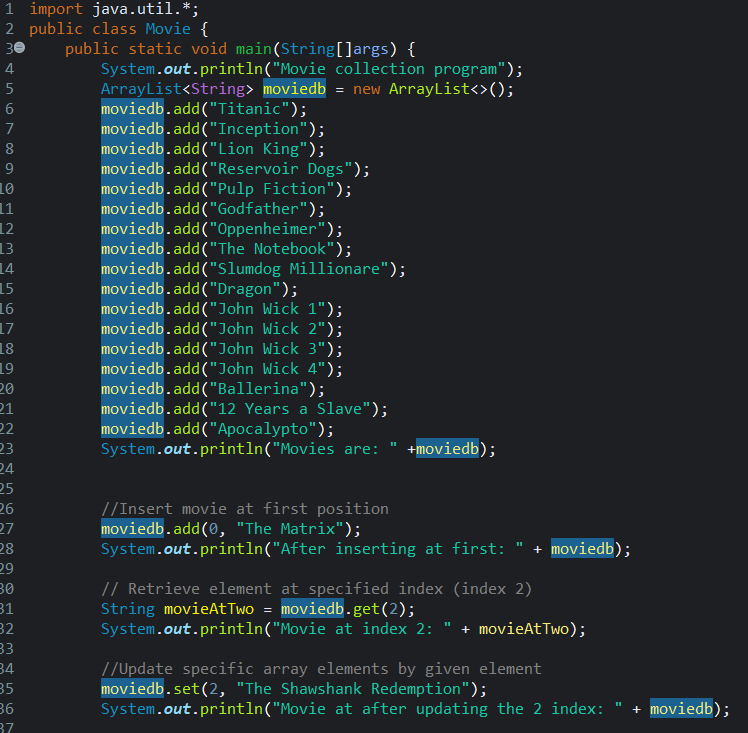
-Write a Java program to remove the third element from an array list.

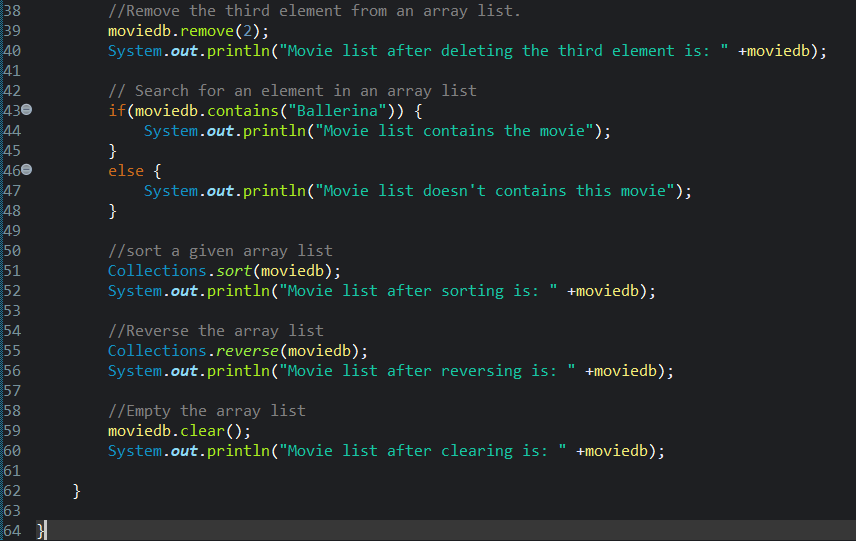
-Write a Java program to search for an element in an array list.

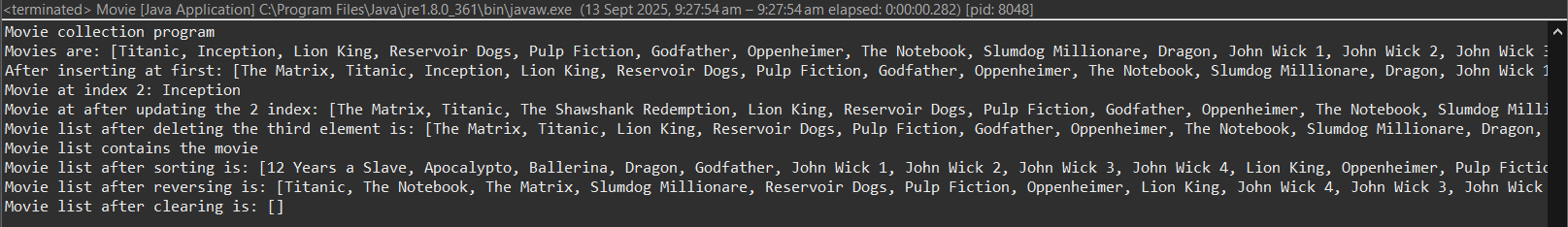
-Write a Java program to sort a given array list.

-Write a Java program to reverse elements in an array list.

-Write a Java program to empty an array list.







**Java Collection: LinkedList**

**1.** Write a Java program to append the specified element to the end of a linked list of names.

-Write a Java program to iterate through all elements in a linked list starting at the specified position.

-Write a Java program to iterate a linked list in reverse order.

-Write a Java program to insert the specified element at the specified position in the linked list.

-Write a Java program to insert elements into the linked list at the first and last position.

-Write a Java program to insert the specified element at the front of a linked list.

-Write a Java program to insert some elements at the specified position into a linked list.

-Write a Java program to get the first and last occurrence of the specified elements in a linked list.

-Write a Java program to remove the first and last element from a linked list.

-Write a Java program to swap two elements in a linked list.

-Write a Java program to join two linked lists.

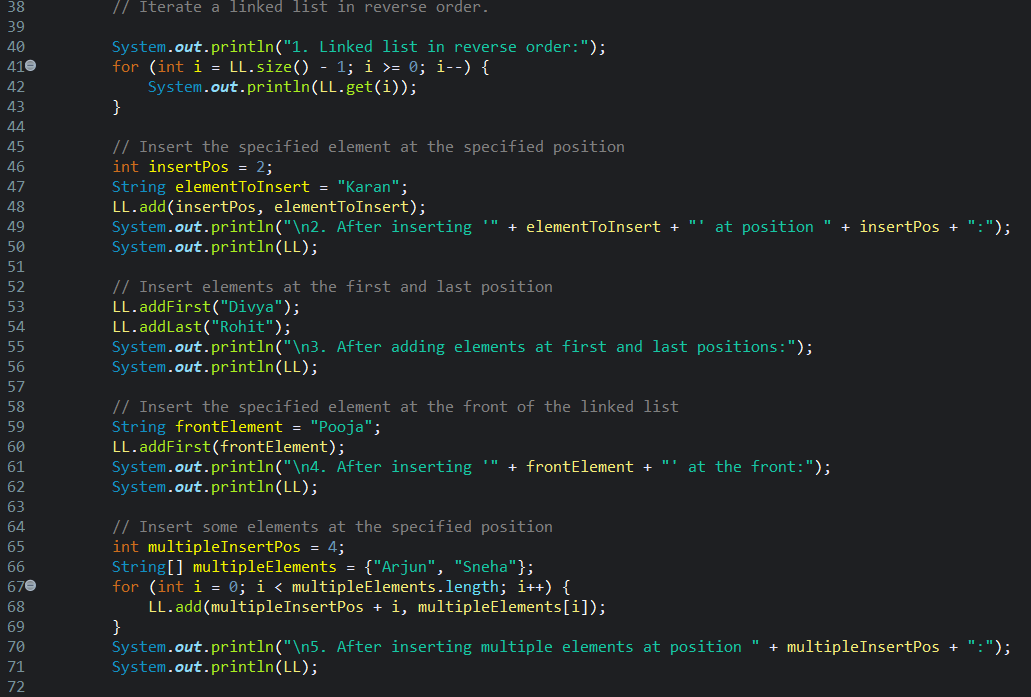
-Write a Java program to check if a particular element exists in a linked list.

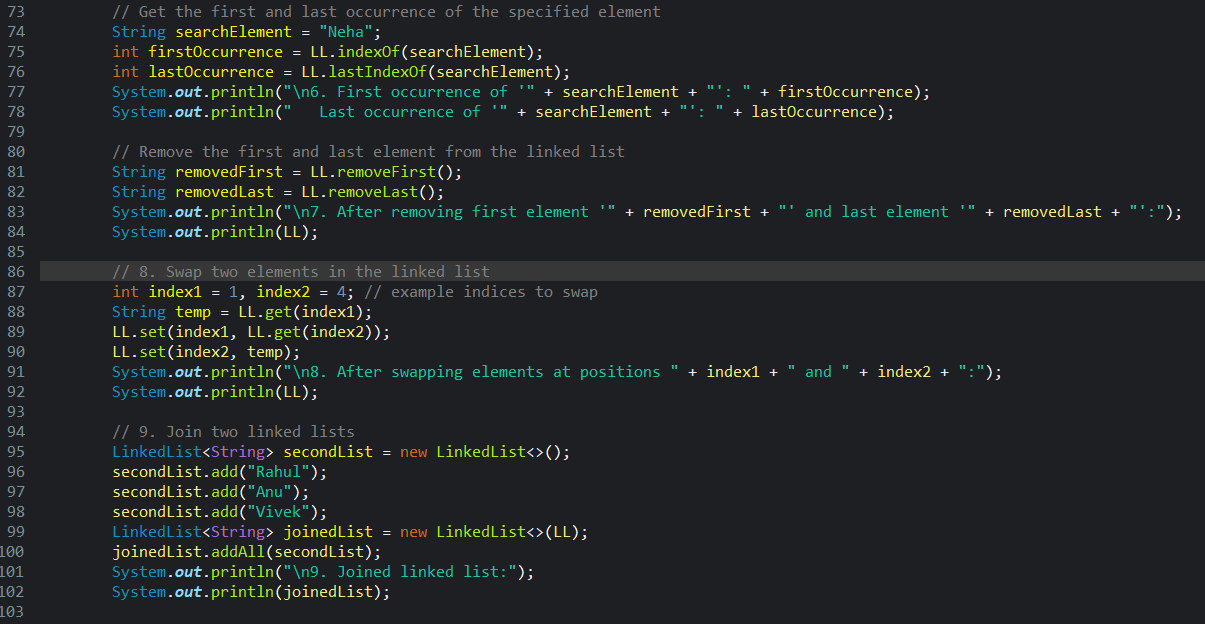
-Write a Java program to convert a linked list to an array list.

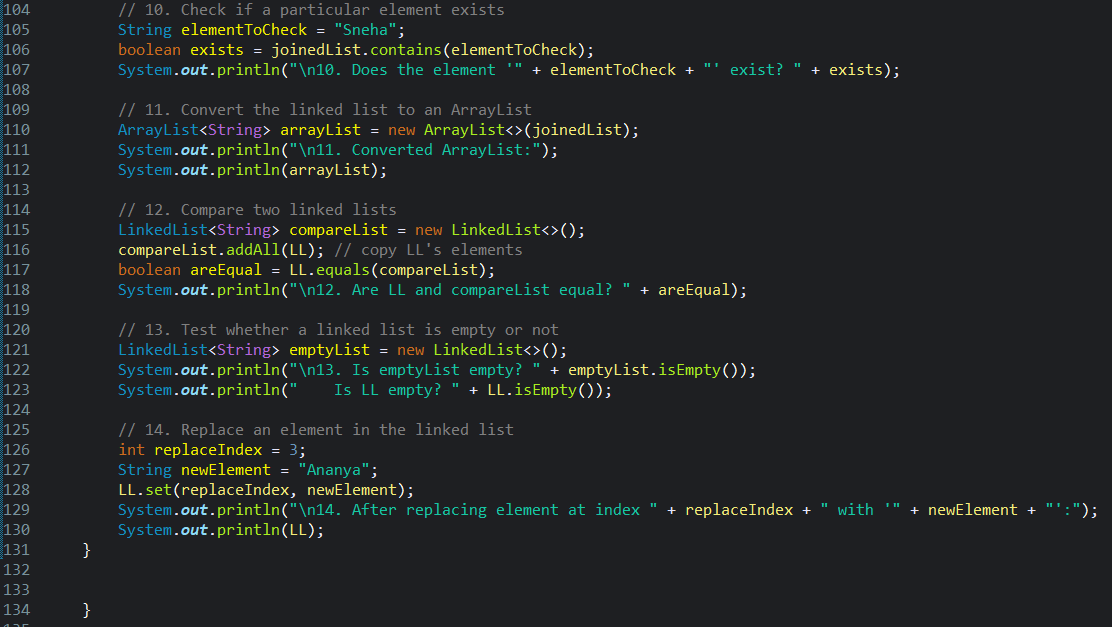
-Write a Java program to compare two linked lists.

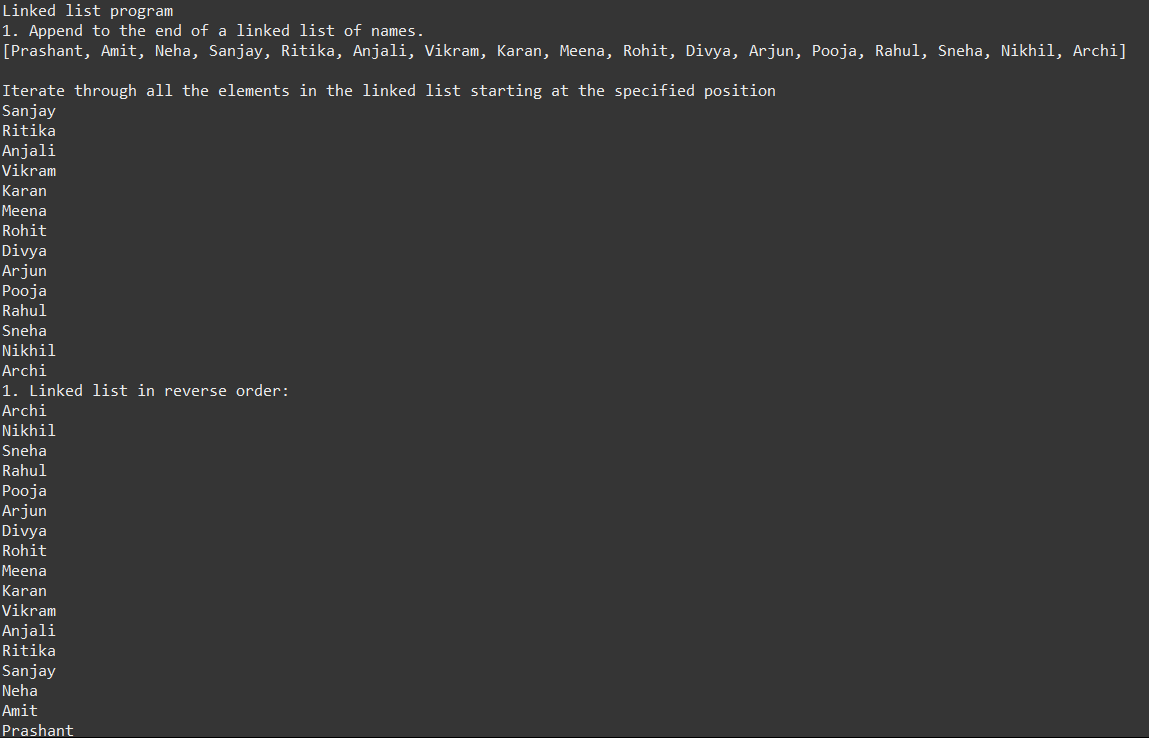
-Write a Java program to test whether a linked list is empty or not. -Write a Java program to replace an element in a linked list.

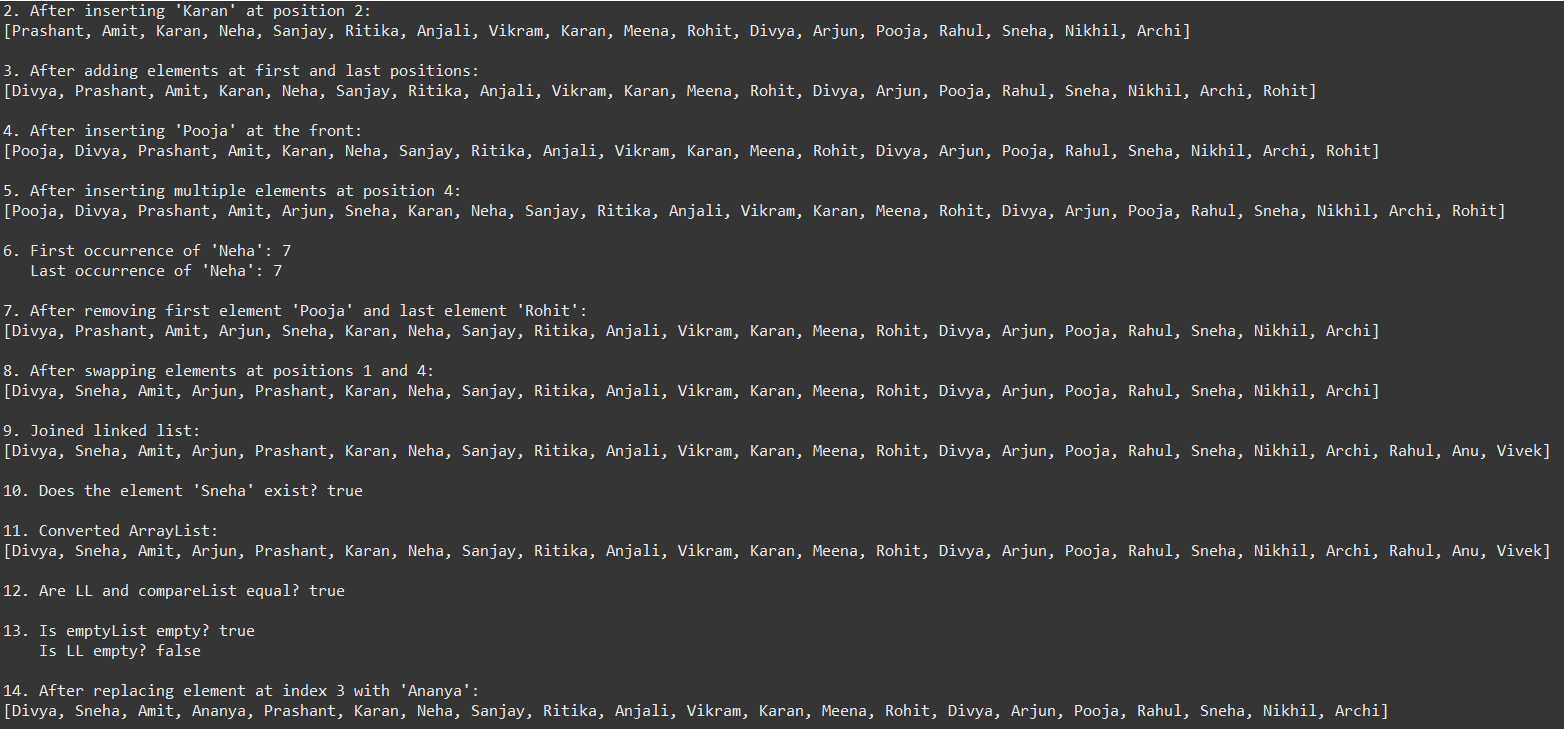












**Java Collection: HashSet Exercises**

1. Write a Java program to append the specified element to the end of a hash set for Employee Id and

Employee name.

-Write a Java program to get the number of elements in a hash set.

-Write a Java program to convert a hash set to an array.

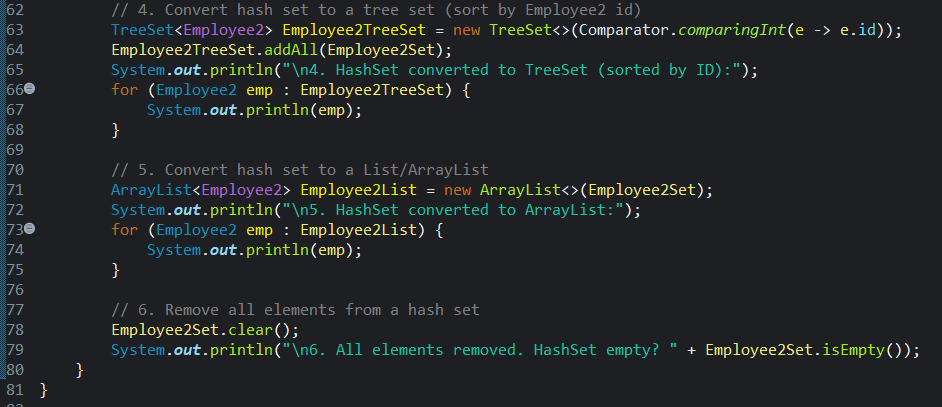
-Write a Java program to convert a hash set to a tree set.

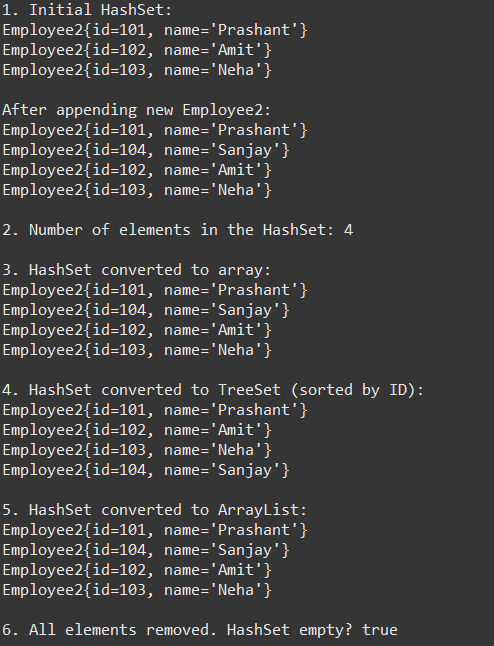
-Write a Java program to convert a hash set to a List/ArrayList.

-Write a Java program to remove all of the elements from a hash set.









**Java Collection: TreeSet**

1. Write a Java program to create a new tree set, add some fruits (string) and print out the tree set.

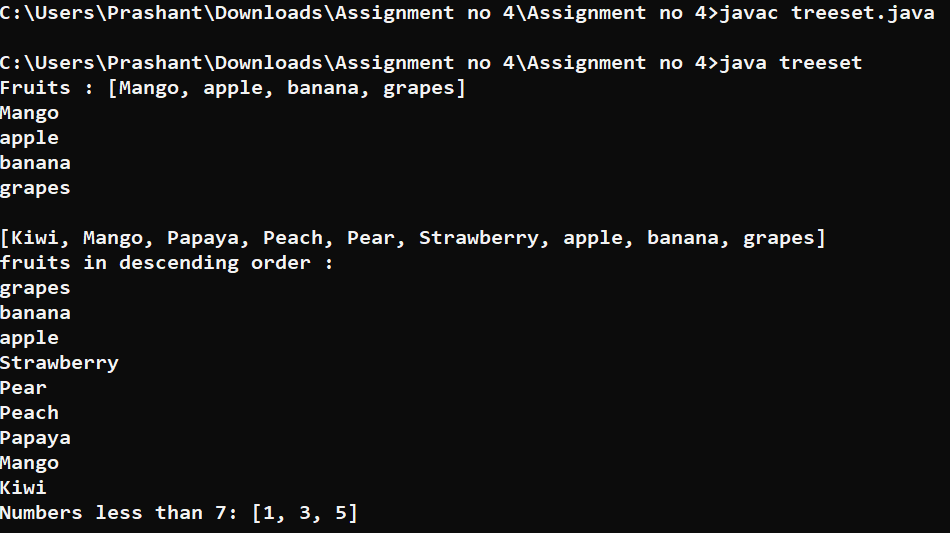
-Write a Java program to iterate through all elements in a tree set.

-Write a Java program to add all the elements of a specified tree set to another tree set.

-Write a Java program to create a reverse order view of the elements contained in a given tree set.

-Write a Java program to find the numbers less than 7 in a tree set.





Java Collection: HashMap

1. Write a Java program to associate the specified value with the specified key in a HashMap.

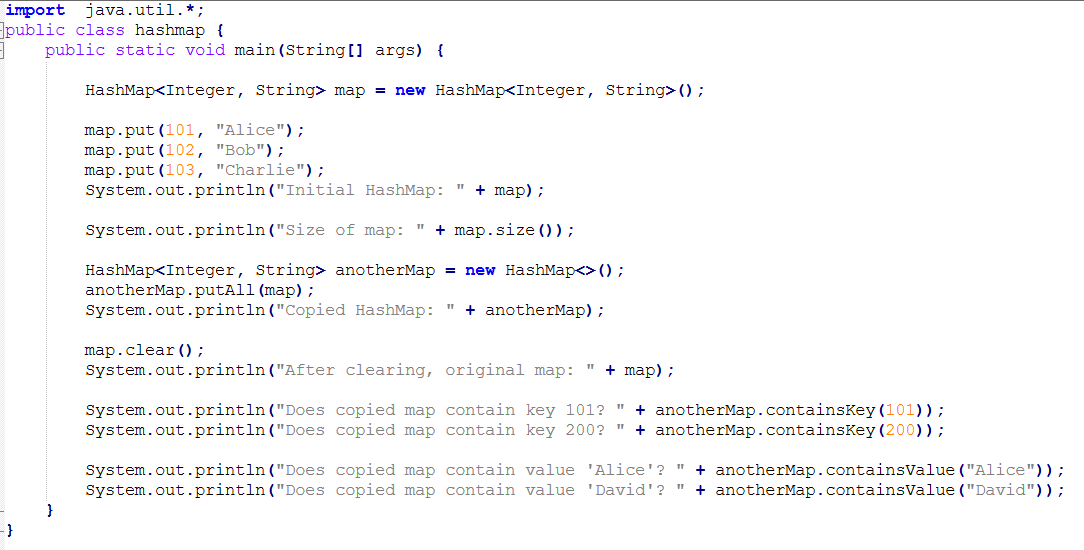
-Write a Java program to count the number of key-value (size) mappings in a map.

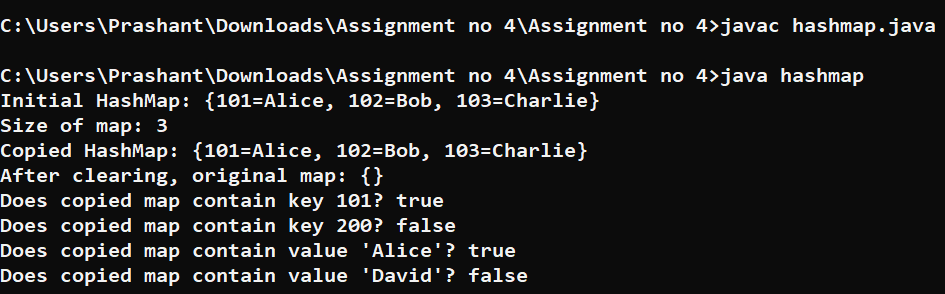
-Write a Java program to copy all of the mappings from the specified map to another map.

-Write a Java program to remove all of the mappings from a map.

-Write a Java program to test if a map contains a mapping for the specified key.

-Write a Java program to test if a map contains a mapping for the specified value.

****

****

**Practice Problem: Ex:**1

Implement different operations on an ArrayList A.

**Input**:

The first line of input contains an integer **T** denoting the no of test cases. Then T test cases follow. The first line of input contains an integer **Q** denoting the no of queries. Then in the next line

are **Q** space separated queries .

A query can be of five types

1. a x (Adds an element x to the ArrayList A at the end)

2. b (Sorts the ArrayList A in ascending order)

3. c (Reverses the ArrayList A)

4. d (prints the size of the ArrayList)

5. e (prints space separated values of the ArrayList)

5. f (Sorts the ArrayList A in descending order)

**Output:**

The output for each test case will be space separated integers denoting the results of each query

**Constraints:**

**1<=T<=**100

**1<**=Q<**=**100

**Example**:

**Input**

2

6

a4 a6 a7bce

**4**

a 55 a 11 de

**Output**

764

2 55 11

**Explanation:**

**For the first test case**

There are six queries. Queries are performed in this order

1. a 4 { ArrayList has 4 }

2. a 7 {ArrayList has 7 }

3. a 6 {ArrayList has 6}

4. b {sorts the ArrayList in ascending order, ArrayList now is 5 6 7}

5. c {reverse the ArrayList}

6. e {prints the element of the ArrayList 7 6 4}

**For the sec test case**

There are four queries. Queries are performed in this order

1. a 55 (ArrayList A has 55}

(prints the size of the ArrayList A ie. 2)

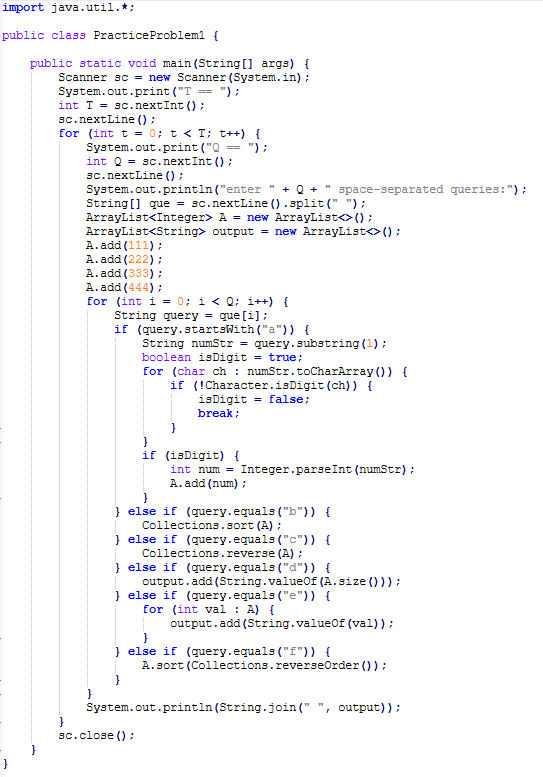
2. a 11

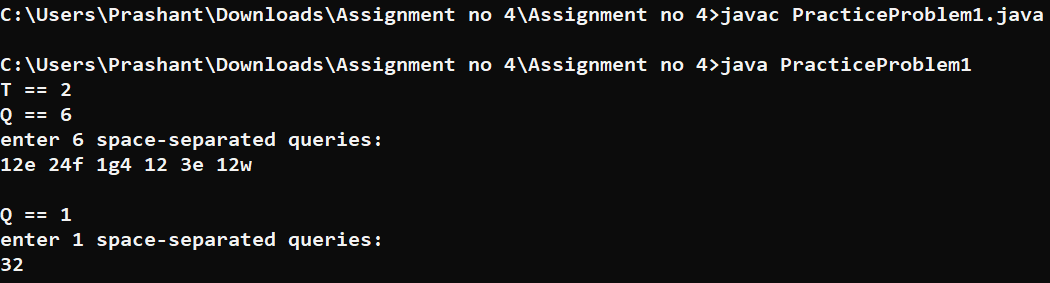
(ArrayList A has 55,11**}**

3. d

**4.** e

(prints the elements of the ArrayList A ie 55 11)





**Practice Problem: Ex:2**

ArrayLis

t are dynamic size arrays. Try this problem using ArrayList.

Given an ArrayList of **N** elements and an integer **Q** defining the type of query(which will be either 1 or 2): **Q = 1** includes two integers **p** and **r**. Which means insert the value r at index p in the ArrayList and print the whole updated ArrayList.

**Q = 2** includes one integer **p**. In this query print the index at which the value p is last found in the ArrayList. If the value p is not found in the ArrayList then print **"-1"**.

**NOTE: Assume 0 based indexing**

**Example 1:**

**Input:**

N

=

5, Q

= 1

A[] = {1, 4, 5, 9, 3}

Query []

**Output:**

= {2,6}

**1** 4 6 5 9 3

**Explanation:**

p=Query [0]=2

r=Query [1]=6

After inserting the element r=6 at index p=2, the updated arraylist ={1,4,6,5,9,3}

**Example 2**:

**Input:**

N = 4, Q

= 2

A[]= {1, 9, 2, 4}

Query[] = {4}

**Output:**

**Explanation:**

3

р

= 4

The element 4 is last found

in A at index = 3

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function **solve()** which takes the **N** (number of elements in Array A),ArrayList **A, Q**(Type of the of query) and the ArrayList **Query**. If the Q = 1 then return the updated ArrayList of integers. else return the ArrayList which contains the index at which the value p is last found in the ArrayList A (where p = Query[0]),If the value of p is not found then return the ArrayList which contains -1.

**Expected Time Complexity:** O(N)

**Expected Auxiliary Space:** O(N) **Constraints:**

1 **<=** N **<=** 104

1 **<=**Q **<= 2**

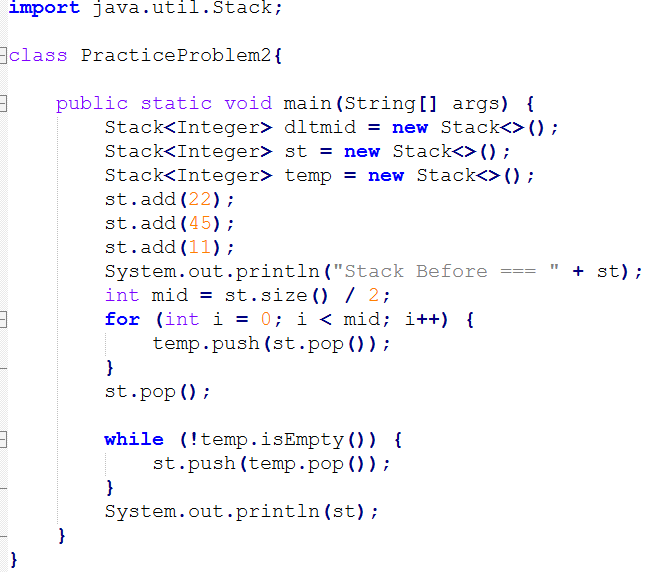
If Q = 1 then size of Query is 2,

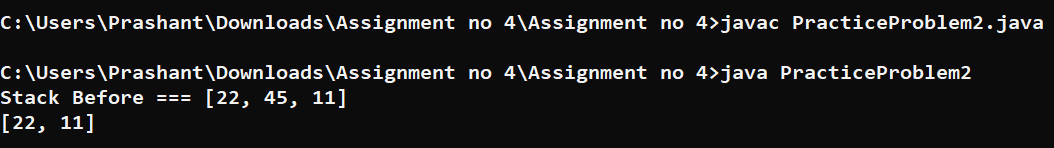
where Query[0] represents the value of p and Query[0] represents the value of r.

If Q = 2 then size of Query is 1,

where Query[0] represents the value of p.

1 **<**= A[i] **<**= 103





**Practice Problem: Ex:3**

Java provides an inbuilt object type called **Stack**. It is a collection that is based on the last in first out (LIFO) principle. Try this problem using Stack.

Given **n** elements of a stack **st** where the first value is the bottom-most value of the stack and the last one is the element at top of the stack, delete the middle element of the stack without using any additional data structure.

**Example 1**:

**Input:** n = 5

st

=

{1**,** 2, 3, 4, 5}

**Output:** 5 4 2 1

**Explanation:** The middle element is 3. If

**it** is deleted and then the values are seen from top, this will be the order.

**Example 2**:

**Input:** n = 6

st **=**

{1**,** 4, 9, 2, 6, 5}

**Output:** 5 6 2 4 1

**Explanation**: The middle element is 9 and **if**

**it** is deleted this will be the stack traversal.

**Your Task:**

You do not need to read input or print anything. Your task is to complete the function **deleteMid()** which takes n and st as input parameters and returns a stack where the middle element is deleted.

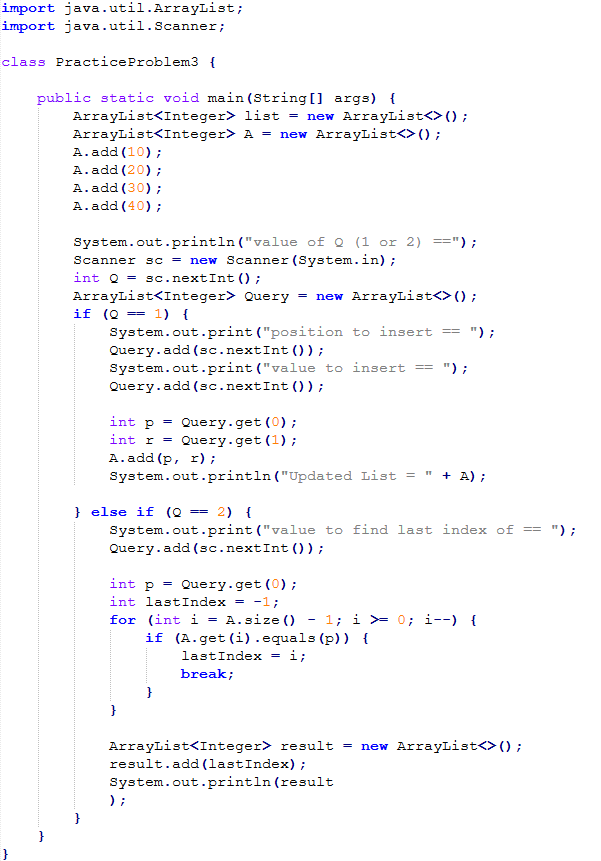
**Expected Time Complexity:** O(n)

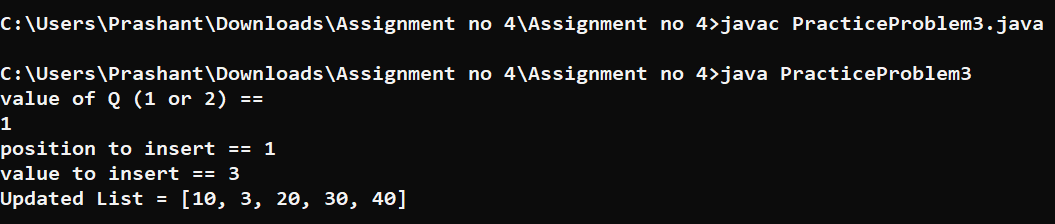
**Expected Auxiliary Space:** O(n)

**Constraints:**

2 ≤ n ≤ 103

1 ≤ st[i] ≤ 104





**Practice Problem: Ex:4**

Implement different operations on a set s.

**Input**:

The first line of input contains an integer **T** denoting the no of test cases. Then T test cases follow. The first line of input contains an integer **Q** denoting the no of queries. Then in the next line are **Q** space separated queries .

A query can be of four types

**1.** a x (inserts an element x to the set s)

**2.** b (prints the contents of the set s in increasing order)

**3.** cx (erases an element x from the sets)

**4.** d x (prints 1 if the element x is present in the set else print **-1)**

**5.** e (prints the size of the set s)

**Output:**

The output for each test case will be space separated integers denoting the results of each query. **Constraints:**

1 **<=** T **<=** 100

1 **<=** Q **<=** 100

**Example:**

**Input**:

2

6

a 1 a 2 a 3 bc2b

5

a 1a5ed5d2

**Output**:

12313

21-1

**Explanation:**

**Testcase 1:**

There are six queries. Queries are performed in this order

1. a 1

2. a 2 3. a 3 4. b

5. c **2**

6. b

{ insert 1 to set now set has {1}}

{inserts 2 to set now set has {1,2} } {inserts 3 to set now set has {1,2,3}} {prints the set contents ie 1,2,3}

{removes 2 from the set}

{prints the set contents ie 1,3}

**Testcase 2:**

There are five queries. Queries are performed in this order

1. a 1

{inserts 1 to set now set has {1}}

2. a 11

{inserts 11 to set now set has {1,11}}

3. e

{prints the size of the set ie 2}

4. d 5

{since five is present prints 1}

5. d 2

{since 2 is not present in the set prints **-1}**

