```java
package stacksqueues;

// Java program for checking
// balanced brackets
import java.util.*;

public class _1BalancedBrackets {

    // function to check if brackets are balanced
    static boolean areBracketsBalanced(String expr)
    {
        // Using ArrayDeque is faster than using Stack class
        Deque<Character> stack
                = new ArrayDeque<Character>();

        // Traversing the Expression
        for (int i = 0; i < expr.length(); i++)
        {
            char x = expr.charAt(i);

            if (x == '(' || x == '[' || x == '{')
            {
                // Push the element in the stack
                stack.push(x);
                continue;
            }

            // IF current current character is not opening
            // bracket, then it must be closing. So stack
            // cannot be empty at this point.
            if (stack.isEmpty())
                return false;
            char check;
            switch (x) {
                case ')':
                    check = stack.pop();
                    if (check == '{' || check == '[')
                        return false;
                    break;

                case '}':
                    check = stack.pop();
                    if (check == '(' || check == '[')
                        return false;
                    break;

                case ']':
                    check = stack.pop();
                    if (check == '(' || check == '{')
                        return false;
                    break;
            }
        }

        // Check Empty Stack
        return (stack.isEmpty());
    }

    // Driver code
    public static void main(String[] args)
    {
        String expr = "([{}])";

        // Function call
        if (areBracketsBalanced(expr))
            System.out.println("Balanced ");
        else
            System.out.println("Not Balanced ");
    }
}
```

```java
package stacksqueues;

//Java program to print next
//greater element using stack

public class _2NGE
{
    static class stack
    {
        int top;
        int items[] = new int[100];

        // Stack functions to be used by printNGE
        void push(int x)
        {
            if (top == 99)
            {
                System.out.println("Stack full");
            }
            else
            {
                items[++top] = x;
            }
        }

        int pop()
        {
            if (top == -1)
            {
                System.out.println("Underflow error");
                return -1;
            }
            else
            {
                int element = items[top];
                top--;
                return element;
            }
        }

        boolean isEmpty()
        {
            return (top == -1) ? true : false;
        }
    }

    /* prints element and NGE pair for
    all elements of arr[] of size n */
    static void printNGE(int arr[], int n)
    {
        int i = 0;
        stack s = new stack();
        s.top = -1;
        int element, next;

        /* push the first element to stack */
        s.push(arr[0]);

        // iterate for rest of the elements
        for (i = 1; i < n; i++)
        {
            next = arr[i];

            if (s.isEmpty() == false)
            {
                // if stack is not empty, then
                // pop an element from stack
                element = s.pop();

                /* If the popped element is smaller than
                next, then a) print the pair b) keep
                popping while elements are smaller and
                stack is not empty */
                while (element < next)
                {
                    System.out.println(element + " --> " + next);
                    if (s.isEmpty() == true)
                        break;
                    element = s.pop();
                }

                /* If element is greater than next, then
                push the element back */
                if (element > next)
                    s.push(element);
            }

            /* push next to stack so that we can find next
            greater for it */
            s.push(next);
        }

        /* After iterating over the loop, the remaining
        elements in stack do not have the next greater
        element, so print -1 for them */
        while (s.isEmpty() == false)
        {
            element = s.pop();
            next = -1;
            System.out.println(element + " -- " + next);
        }
    }

    public static void main(String[] args)
    {
        int arr[] = { 13, 11,9,8, 21, 3 };
        int n = arr.length;
        printNGE(arr, n);
    }
}

// Thanks to Rishabh Mahrsee for contributing this code
```

```java
package stacksqueues;

// Java program to implement Queue using
// two stacks with costly enQueue()
import java.util.*;

class GFG
{
    static class Queue
    {
        static Stack<Integer> s1 = new Stack<Integer>();
        static Stack<Integer> s2 = new Stack<Integer>();

        static void enQueue(int x)
        {
            // Move all elements from s1 to s2
            while (!s1.isEmpty())
            {
                s2.push(s1.pop());
                //s1.pop();
            }

            // Push item into s1
            s1.push(x);

            // Push everything back to s1
            while (!s2.isEmpty())
            {
                s1.push(s2.pop());
                //s2.pop();
            }
        }

        // Dequeue an item from the queue
        static int deQueue()
        {
            // if first stack is empty
            if (s1.isEmpty())
            {
                System.out.println("Q is Empty");
                System.exit(0);
            }

            // Return top of s1
            int x = s1.peek();
            s1.pop();
            return x;
        }
    };

    // Driver code
    public static void main(String[] args)
    {
        Queue q = new Queue();
        q.enQueue(1);
        q.enQueue(2);
        q.enQueue(3);

        System.out.println(q.deQueue());
        System.out.println(q.deQueue());
        System.out.println(q.deQueue());
    }
}

// This code is contributed by Prerna Saini
```

```java
package stacksqueues;

/* Java Program to implement a stack using
two queue */
import java.util.*;

class GfG {

    static class Stack {
        // Two inbuilt queues
        static Queue<Integer> q1 = new LinkedList<Integer>();
        static Queue<Integer> q2 = new LinkedList<Integer>();

        // To maintain current number of
        // elements
        static int curr_size;

        Stack()
        {
            curr_size = 0;
        }

        static void push(int x)
        {
            curr_size++;

            // Push x first in empty q2
            q2.add(x);

            // Push all the remaining
            // elements in q1 to q2.
            while (!q1.isEmpty()) {
                q2.add(q1.peek());
                q1.remove();
            }

            // swap the names of two queues
            Queue<Integer> q = q1;
            q1 = q2;
            q2 = q;
        }

        static void pop()
        {

            // if no elements are there in q1
            if (q1.isEmpty())
                return;
            q1.remove();
            curr_size--;
        }

        static int top()
        {
            if (q1.isEmpty())
                return -1;
            return q1.peek();
        }

        static int size()
        {
            return curr_size;
        }
    }

    // driver code
    public static void main(String[] args)
    {
        Stack s = new Stack();
        s.push(1);
        s.push(2);
        s.push(3);

        System.out.println("current size: " + s.size());
        System.out.println(s.top());
        s.pop();
        System.out.println(s.top());
        s.pop();
        System.out.println(s.top());

        System.out.println("current size: " + s.size());
    }
}
// This code is contributed by Prerna
```

```java
package stacksqueues;

// Java program to implement a stack that supports
// getMinimum() in O(1) time and O(1) extra space.
import java.util.*;

// A user defined stack that supports getMin() in
// addition to push() and pop()
class MyStack
{
    Stack<Integer> s;
    Integer minEle;

    // Constructor
    MyStack() { s = new Stack<Integer>(); }

    // Prints minimum element of MyStack
    void getMin()
    {
        // Get the minimum number in the entire stack
        if (s.isEmpty())
            System.out.println("Stack is empty");

            // variable minEle stores the minimum element
            // in the stack.
        else
            System.out.println("Minimum Element in the " +
                    " stack is: " + minEle);
    }

    // prints top element of MyStack
    void peek()
    {
        if (s.isEmpty())
        {
            System.out.println("Stack is empty ");
            return;
        }

        Integer t = s.peek(); // Top element.

        System.out.print("Top Most Element is: ");

        // If t < minEle means minEle stores
        // value of t.
        if (t < minEle)
            System.out.println(minEle);
        else
            System.out.println(t);
    }

    // Removes the top element from MyStack
    void pop()
    {
        if (s.isEmpty())
        {
            System.out.println("Stack is empty");
            return;
        }

        System.out.print("Top Most Element Removed: ");
        Integer t = s.pop();

        // Minimum will change as the minimum element
        // of the stack is being removed.
        if (t < minEle)
        {
            System.out.println(minEle);
            minEle = 2*minEle - t;
        }

        else
            System.out.println(t);
    }

    // Insert new number into MyStack
    void push(Integer x)
    {
        if (s.isEmpty())
        {
            minEle = x;
            s.push(x);
            System.out.println("Number Inserted: " + x);
            return;
        }

        // If new number is less than original minEle
        if (x < minEle)
        {
            s.push(2*x - minEle);
            minEle = x;
        }

        else
            s.push(x);

        System.out.println("Number Inserted: " + x);
    }
};

// Driver Code
 class Main
{
    public static void main(String[] args)
    {
        MyStack s = new MyStack();
        s.push(3);
        s.push(5);
        s.getMin();
        s.push(2);
        s.push(1);
        s.getMin();
        s.pop();
        s.getMin();
        s.pop();
        s.peek();
    }
}
```

```
package stacksqueues;

/* We can use Java inbuilt Deque as a double
ended queue to store the cache keys, with
the descending time of reference from front
to back and a set container to check presence
of a key. But remove a key from the Deque using
remove(), it takes O(N) time. This can be
optimized by storing a reference (iterator) to
each key in a hash map. */
import java.util.Deque;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Iterator;

 class LRUCache {

    // store keys of cache
    private Deque<Integer> doublyQueue;

    // store references of key in cache
    private HashSet<Integer> hashSet;

    // maximum capacity of cache
    private final int CACHE_SIZE;

    LRUCache(int capacity) {
        doublyQueue = new LinkedList<>();
        hashSet = new HashSet<>();
        CACHE_SIZE = capacity;
    }

    /* Refer the page within the LRU cache */
    public void refer(int page) {
        if (!hashSet.contains(page)) {
            if (doublyQueue.size() == CACHE_SIZE) {
                int last = doublyQueue.removeLast();
                hashSet.remove(last);
            }
        }
        else {/* The found page may not be always the last element, even if it's an
            intermediate element that needs to be removed and added to the start
            of the Queue */
            doublyQueue.remove(page);
        }
        doublyQueue.push(page);
        hashSet.add(page);
    }

    // display contents of cache
    public void display() {
        Iterator<Integer> itr = doublyQueue.iterator();
        while (itr.hasNext()) {
            System.out.print(itr.next() + " ");
        }
    }

    public static void main(String[] args) {
        LRUCache cache = new LRUCache(4);
        cache.refer(1);
        cache.refer(2);
        cache.refer(3);
        cache.refer(1);
        cache.refer(4);
        cache.refer(5);
        cache.refer(2);
        cache.refer(2);
        cache.refer(1);
        cache.display();
    }
}
// This code is contributed by Niraj Kumar
```

```java
package stacksqueues;

//Java program to find circular tour for a truck

 class Petrol
{
    // A petrol pump has petrol and distance to next petrol pump
    static class petrolPump
    {
        int petrol;
        int distance;

        // constructor
        public petrolPump(int petrol, int distance)
        {
            this.petrol = petrol;
            this.distance = distance;
        }
    }

    // The function returns starting point if there is a possible solution,
    // otherwise returns -1
    static int printTour(petrolPump arr[], int n)
    {
        int start = 0;
        int end = 1;
        int curr_petrol = arr[start].petrol - arr[start].distance;

        // If current amount of petrol in truck becomes less than 0, then
        // remove the starting petrol pump from tour
        while(end != start || curr_petrol < 0)
        {

            // If current amount of petrol in truck becomes less than 0, then
            // remove the starting petrol pump from tour
            while(curr_petrol < 0 && start != end)
            {
                // Remove starting petrol pump. Change start
                curr_petrol -= arr[start].petrol - arr[start].distance;
                start = (start + 1) % n;

                // If 0 is being considered as start again, then there is no
                // possible solution
                if(start == 0)
                    return -1;
            }
            // Add a petrol pump to current tour
            curr_petrol += arr[end].petrol - arr[end].distance;

            end = (end + 1)%n;
        }

        // Return starting point
        return start;
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {

        petrolPump[] arr = {new petrolPump(6, 4),
                new petrolPump(3, 6),
                new petrolPump(7, 3)};

        int start = printTour(arr, arr.length);
        System.out.println(start == -1 ? "No Solution" : "Start = " + start);

    }

}
//This code is contributed by Sumit Ghosh
```

```java
package stacksqueues;

// A Java program to find first non-repeating character
// from a stream of characters

import java.util.ArrayList;
import java.util.List;

 class NonReapeatingC {
    final static int MAX_CHAR = 256;

    static void findFirstNonRepeating()
    {
        // inDLL[x] contains pointer to a DLL node if x is
        // present in DLL. If x is not present, then
        // inDLL[x] is NULL
        List<Character> inDLL = new ArrayList<Character>();

        // repeated[x] is true if x is repeated two or more
        // times. If x is not seen so far or x is seen only
        // once. then repeated[x] is false
        boolean[] repeated = new boolean[MAX_CHAR];

        // Let us consider following stream and see the
        // process
        String stream = "geeksforgeeksandgeeksquizfor";
        for (int i = 0; i < stream.length(); i++) {
            char x = stream.charAt(i);
            System.out.println("Reading " + x
                    + " from stream \n");

            // We process this character only if it has not
            // occurred or occurred only once. repeated[x]
            // is true if x is repeated twice or more.s
            if (!repeated[x]) {
                // If the character is not in DLL, then add
                // this at the end of DLL.
                if (!(inDLL.contains(x))) {
                    inDLL.add(x);
                }
                else // Otherwise remove this character from
                // DLL
                {
                    inDLL.remove((Character)x);
                    repeated[x]
                            = true; // Also mark it as repeated
                }
            }

            // Print the current first non-repeating
            // character from stream
            if (inDLL.size() != 0) {
                System.out.print(
                        "First non-repeating character so far is ");
                System.out.println(inDLL.get(0));
            }
        }
    }

    /* Driver code */
    public static void main(String[] args)
    {
        findFirstNonRepeating();
    }
}
// This code is contributed by Sumit Ghosh
```

```
package stacksqueues;

// Java Program to find the maximum for
// each and every contiguous subarray of size k.
import java.util.Deque;
import java.util.LinkedList;

 class SlidingWindow
{

    // A Dequeue (Double ended queue)
    // based method for printing
    // maximum element of
    // all subarrays of size k
    static void printMax(int arr[], int n, int k)
    {

        // Create a Double Ended Queue, Qi
        // that will store indexes of array elements
        // The queue will store indexes of
        // useful elements in every window and it will
        // maintain decreasing order of values
        // from front to rear in Qi, i.e.,
        // arr[Qi.front[]] to arr[Qi.rear()]
        // are sorted in decreasing order
        Deque<Integer> Qi = new LinkedList<Integer>();

        /* Process first k (or first window)
        elements of array */
        int i;
        for (i = 0; i < k; ++i)
        {

            // For every element, the previous
            // smaller elements are useless so
            // remove them from Qi
            while (!Qi.isEmpty() && arr[i] >=
                    arr[Qi.peekLast()])

                // Remove from rear
                Qi.removeLast();

            // Add new element at rear of queue
            Qi.addLast(i);
        }

        // Process rest of the elements,
        // i.e., from arr[k] to arr[n-1]
        for (; i < n; ++i)
        {

            // The element at the front of the
            // queue is the largest element of
            // previous window, so print it
            System.out.print(arr[Qi.peek()] + " ");

            // Remove the elements which
            // are out of this window
            while ((!Qi.isEmpty()) && Qi.peek() <=
                    i - k)
                Qi.removeFirst();

            // Remove all elements smaller
            // than the currently
            // being added element (remove
            // useless elements)
```

```java
            while ((!Qi.isEmpty()) && arr[i] >=
                    arr[Qi.peekLast()])
                Qi.removeLast();

            // Add current element at the rear of Qi
            Qi.addLast(i);
        }

        // Print the maximum element of last window
        System.out.print(arr[Qi.peek()]);
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 12, 1, 78, 90, 57, 89, 56 };
        int k = 3;
        printMax(arr, arr.length, k);
    }
}
// This code is contributed by Sumit Ghosh
```