```java
import com.sun.org.apache.xalan.internal.xsltc.compiler.util.MultiHashtable;
import com.sun.org.apache.xpath.internal.operations.Mult;
import java.util.Arrays;

/**
 * Created by joey on 11/19/16.
 */
public class Main {

    public static void main(String[] args){

        // ------------------------------Part 1 - Highest Number ----------------------------------//
        //helper methods at bottom of page;


        int[] myNumbers = randList(15);

        // Inheritance Implementation                   Array , # of threads
        Thread[] threadList = makeThreadsInheritance(myNumbers, 4);
        startAll(threadList);
        joinAll(threadList);
        System.out.println("Highest number (Inheritance):"  + MultiInherit.highestNumber);


        // Interface Implementation                     Array , # of threads
        Thread[] threadList2 = makeThreadsInterface(myNumbers, 4);
        startAll(threadList2);
        joinAll(threadList2);
        System.out.println("Highest number (Interface):"  + MultiInterface.highestNumber);
        System.out.println("Array of random Numbers" + Arrays.toString(myNumbers));

        //-------------------------------- Part 2 - Stacks and Queues ---------------------------------------

        //To keep the code more readable and to adhere to the instructions
        //directly I left out some try-catch blocks
        //Note: exceptions are thrown in various stacks and queues
        //This was done intentionally for readability not design, otherwise they'd be there.

        Stack myStack1 = new LinkedListStack();
        Stack myStack2 = new GrowableArrayStack();

        System.out.println("Pushing items into stack:");

        for(int i = 0; i < 5; i++){
            myStack1.Push(i);
            myStack2.Push(i);
            System.out.println(i); //Alternatively, you can output this message in your push method
        }

        System.out.println("Popping items out of stack:");

        for(int i = 0; i < 5; i++){
            System.out.println(myStack1.Pop());
            System.out.println(myStack2.Pop());
        }

        Queue myQueue1 = new LinkedListQueue();
        Queue myQueue2 = new GrowableArrayQueue();

        System.out.println("Enqueueing items into queue:");

        for(int i = 0; i < 5; i++){
            myQueue1.Enqueue(i);
            myQueue2.Enqueue(i);
            System.out.println(i); //Alternatively, you can output this message in your enqueue method

        }

        System.out.println("Dequeuing items from queue:");

        for(int i = 0; i < 5; i++){
            System.out.println(myQueue1.Dequeue());
            System.out.println(myQueue2.Dequeue());

        }

        // ---------------------------------------- Part 3 - Generics --------------------------------//

        //This class is a Generic class which allows it to work with various types of objects
        // <Integer>, <String>, <Node>, etc.

        GenericGrowableArray<String> arr = new GenericGrowableArray<String>();
        GenericGrowableArray<Integer> arr2 = new GenericGrowableArray<Integer>();

        //implementing generic container
        // new arrays must begin as Object arrays and then casted to type (T[]).
        // See GenericGrowableArray class for full implementation.
        //methods and properties must be set to take a generic type;

        String[] str = {"Hello", "Professor", "Lee", "!"};
        Integer[] ints = {1, 2, 3, 4};

        for(String string: str)
            arr.add(string);

        for(Integer integer: ints)
            arr2.add(integer);

        for(int i = 0; i < 4; i++){
            System.out.print(arr.atIndex(i) + " ");
            System.out.print(arr2.atIndex(i) + " ");
        }
    }
    //----------------------------------------- Helper Methods --------------------------------------//

    private static void startAll(Thread[] threads){
        for(Thread thread : threads){
            thread.start();
        }
    }


    private static void joinAll(Thread[] threads){
        for(Thread thread: threads){
            try {
                thread.join();
            }catch(Exception ex){}
        }
    }


    private static int[] randList(int length){
        int[] out = new int[length];
        for (int i = 0; i < length; i++){
            out[i] = (int)(Math.random() * 100);
        }
        return out;
    }


    private static Thread[] makeThreadsInterface(int[] arr, int numThreads){
        Thread[] threads = new Thread[numThreads];

        for (int i = 0; i < numThreads; i++){
            int length = arr.length / numThreads;
            int begin = i == 0 ? 0 : i * length;
            int end = (i == numThreads - 1 ? arr.length : begin + length) - 1;
            threads[i] = new Thread(new MultiInterface(arr, begin, end));
        }
        return threads;
    }


    private static Thread[] makeThreadsInheritance(int[] arr, int numThreads) {
        Thread[] threads = new Thread[numThreads];

        for (int i = 0; i < numThreads; i++){
            int length = arr.length / numThreads;
            int begin = i == 0 ? 0 : i * length;
            int end = (i == numThreads - 1 ? arr.length : begin + length) - 1;
            threads[i] = new MultiInherit(arr, begin, end);
        }

        return threads;
    }

}
```

```java
/**
 * Created by joey on 11/20/16.
 */
public class Node {
    private int value = 0;
    private Node next = null;

    public Node(int value){
        this.value = value;
    }
    public Node(int value, Node next){
        this.value = value;
        this.next = next;
    }

    public int getValue(){
        return this.value;
    }
    public Node getNext(){
        return this.next;
    }
    public void setValue(int value){
        this.value = value;
    }
    public void setNext(Node next){
        this.next = next;
    }
}
```

```
/**
 * Created by joey on 11/20/16.
 */
public interface Queue {
    void Enqueue(int n);
    int Dequeue();
    int length();
}
```

```java
/**
 * Created by joey on 11/20/16.
 */
public interface Stack {
    void Push(int n);
    int Pop();
    int length();
}
```

```java
/**
 * Created by joey on 11/20/16.
 */
public class LinkedList{
    protected Node headNode = null;
    protected Node endNode = null;
    protected int listLength = 0;

    //construct a linked list with no nodes
    public LinkedList(){
    }

    //construct a linked list with one node
    public LinkedList(int value){
        Node newnode = new Node(value);
        headNode = newnode;
        endNode = newnode;
        listLength++;
    }

    //I did not implement any insert or remove functions in this class
    //this design choice was due to the fact that my stack and queue classes
    //have their own names (push vs enqueue).  Also I did think that a stack or queue
    //should be able to remove or insert to arbitrary locations in the linkedlist

    //These functionalities were put in a subclass of LinkedList called LinkedListExtra

}
```

```java
/**
 * Created by joey on 11/19/16.
 */
public class MultiInherit extends Thread{
    private int begin;
    private int end;
    private int[] theArr;
    public static int highestNumber = 0;


    public MultiInherit(int[] arr, int begin, int end) {
        this.begin = begin;
        this.end = end;
        this.theArr = arr;
    }


    public void run(){
        for (int i = begin; i <= end; i++) {
            int numToCheck = theArr[i];
            if(numToCheck > highestNumber) {
                this.setHighestNumber(numToCheck);
            }
        }
    }


    synchronized void setHighestNumber(int numToCheck){
        if(numToCheck > highestNumber){
            MultiInherit.highestNumber = numToCheck;
        }
    }

}
```

```java
/**
 * Created by joey on 11/20/16.
 */
public class GrowableArray {
    protected int[] innerArr = new int[10];
    protected int nextEmptyIndex = 0;
    protected int expandBy = 10;

    public GrowableArray(){
    }

    public GrowableArray(int size){
        this.innerArr = new int[size];
    }

    public void add(int item){
        if(nextEmptyIndex >= innerArr.length - 1){
            increaseSize();
        }
        innerArr[nextEmptyIndex] = item;
        nextEmptyIndex++;
    }

    public void increaseSize(){
        int[] newArr = new int[innerArr.length + expandBy];
        for(int i = 0; i < innerArr.length; i++){
            newArr[i] = innerArr[i];
        }
        innerArr = newArr;
    }

    public int atIndex(int num) throws IndexOutOfBoundsException{
            if (num >= nextEmptyIndex)
                throw new IndexOutOfBoundsException();
            else return innerArr[num];
    }
}
```

```java
/**
 * Created by joey on 11/19/16.
 */
public class MultiInterface implements Runnable{

    private int begin;
    private int end;
    private int[] theArr;
    public static int highestNumber = 0;


    public MultiInterface(int[] arr, int begin, int end) {
        this.begin = begin;
        this.end = end;
        this.theArr = arr;
    }


    public void run(){
        for (int i = begin; i <= end; i++) {
            int numToCheck = theArr[i];
            if(numToCheck > highestNumber) {
                this.setHighestNumber(numToCheck);
            }
        }
    }


    synchronized void setHighestNumber(int numToCheck){
        if(numToCheck > highestNumber){
            MultiInterface.highestNumber = numToCheck;
        }
    }

}
```

```java
/**
 * Created by joey on 11/20/16.
 */

//This implementation include methods for adding pushing, inserting and removing nodes
public class LinkedListExtra extends LinkedList{

    public int length(){
        return listLength;
    }

    public Node getLast(){
        return endNode;
    }

    public void insertNode(int index, int newNode){
        Node newnode = new Node(newNode);
        if(index > listLength -1){
            System.out.println("OOB");
            return;
        }
        if(index == 0){
            Node oldhead = headNode;
            headNode = newnode;
            headNode.setNext(oldhead);
        }else{
            Node tempNode = headNode;
            while(index - 1 != 0){
                tempNode = tempNode.getNext();
                index--;
            }
            newnode.setNext(tempNode.getNext());
            tempNode.setNext(newnode);
        }
        listLength++;
    }

    public void removeNode(int index){
        if(index > listLength -1){
            System.out.println("OOB");
            return;
        }
        if(index == 0){
            headNode = headNode.getNext();
        }
        else{
            Node tempNode = headNode;
            while(index - 1 != 0){
                tempNode = tempNode.getNext();
                index--;
            }
            if(index == listLength - 1) {
                tempNode.setNext(null);
                endNode = tempNode;
            }else{
                tempNode.setNext(tempNode.getNext().getNext());
            }
        }
        listLength--;
    }

    public void pushNode(int value){
        Node newNode = new Node(value);
        endNode.setNext(newNode);
        endNode = newNode;
        listLength++;
    }


}
```

```
/**
 * Created by joey on 11/20/16.
 */
public class LinkedListQueue extends LinkedList implements Queue{


    public void Enqueue(int value){
        Node newNode = new Node(value);
        //if linked list is empty
        if(headNode == null){
            headNode = newNode;
            endNode = newNode;
        }
        else {
            endNode.setNext(newNode);
            endNode = newNode;
        }
        listLength++;
    }


    public int Dequeue() {
        int toreturn = headNode.getValue();
        if(headNode == null)
            throw new IndexOutOfBoundsException();
        else if(listLength == 1){
            headNode = null;
            endNode = null;
        }
        else{
            headNode = headNode.getNext();
        }
        listLength--;
        return toreturn;
    }


    public int length() {
        return listLength;
    }
}
```

```
/**
 * Created by joey on 11/20/16.
 */
public class LinkedListStack extends LinkedList implements Stack {

    public void Push(int value){
        Node newNode = new Node(value);
        //if linked list is empty
        if(headNode == null){
            headNode = newNode;
            endNode = newNode;
        }
        else {
            endNode.setNext(newNode);
            endNode = newNode;
        }
        listLength++;
    }


    public int Pop() throws IndexOutOfBoundsException{
        int toreturn = endNode.getValue();
        //if only no elements in linked list throw exception
        if(length() < 1) throw new IndexOutOfBoundsException();

        //if list has 1 elem
        else if(length() == 1){
            headNode = null;
            endNode = null;

        }
        //point second to last node to null
        //and set that to endnode
        else {
            Node tempNode = headNode;
            int secondToLast = listLength - 2;
            for (int i = 0; i < secondToLast; i++) {
                tempNode = tempNode.getNext();
            }
            tempNode.setNext(null);
            endNode = tempNode;
        }
        listLength--;
        return toreturn;
    }

    public int length(){
        return listLength;
    }




}
```

```
/**
 * Created by joey on 11/20/16.
 */
public class GrowableArrayQueue extends GrowableArray implements Queue {
    private int nextOnQueue = 0;
    private int Arrlen = 0;


    public void Enqueue(int n) {
        add(n);
        Arrlen++;
    }


    public int Dequeue() {
        int toreturn = innerArr[nextOnQueue];
        Arrlen--;
        nextOnQueue++;
        return toreturn;
    }


    public int length() {
        return Arrlen;
    }
}
```

```
/**
 * Created by joey on 11/20/16.
 */
public class GrowableArrayStack extends GrowableArray implements Stack {


    public void Push(int n) {
        add(n);
    }


    public int Pop() {
        int toreturn = atIndex(nextEmptyIndex - 1);
        nextEmptyIndex--;
        return toreturn;
    }


    public int length() {
        return nextEmptyIndex;
    }
}
```

```
/**
 * Created by joey on 11/20/16.
 */
public class GenericGrowableArray<T> {


    protected T[] innerArr;
    protected int nextEmptyIndex = 0;
    protected int expandBy = 10;


    public GenericGrowableArray(){
        T[] inputarr = (T[])new Object[10];
        innerArr = inputarr;
    }


    public GenericGrowableArray(int size){
        T[] inputarr = (T[])new Object[size];
        innerArr = inputarr;
    }


    public void add(T item){
        if(nextEmptyIndex >= innerArr.length - 1){
            increaseSize();
        }
        innerArr[nextEmptyIndex] = item;
        nextEmptyIndex++;
    }


    public void increaseSize(){
        T[] newArr = (T[])new Object[innerArr.length + expandBy];
        for(int i = 0; i < innerArr.length; i++){
            newArr[i] = innerArr[i];
        }
        innerArr = newArr;
    }


    public T atIndex(int num) throws IndexOutOfBoundsException{
            if (num >= nextEmptyIndex)
                throw new IndexOutOfBoundsException();
            else return innerArr[num];
    }
}
```