Notes for Lab3.

Exercise 5:

Define a recursive function to convert a Decimal number to Binary. *(The output must be an integer number. Example with input is 8, output must be an integer number 1000).*

```
    public static int Dec2Binary(int nDec){
        if(nDec==0)
            return 0;
        else
            return (nDec%2 + 10*Dec2Binary(nDec/2));
    }
```

Dec2Binary(8)

8%2 + 10*Dec2Binary(8/2) -> 1000

4%2 + 10 * Dec2Binary(4/2) -> 100

2%2 + 10 * Dec2Binary(2/2) -> 10

1%2 + 10*Dec2Binary(1/2) - > 1

0%2 + 10*Dec2Binary(0/2) -> 0

TestExer7.java ● TestExer5.java ●

D: > _Teach > 01.TDT > 08.CTDL > Lab21.22 > Lab3 > TestExer5.java > TestExer5 > Dec2Binary(int)

```
1   //Recursive Function to Convert a Decimal to Binary
2   import java.lang.*;
3   public  class TestExer5{
4       public static void main(String[] args){
5           System.out.println("Test Dec2Binary 8=" + Dec2Binary(8));
6       }
7       // Dec: 8 -> Binary:1000 , 8/2= 4 du(%) 0, 4/2= 2 du(%) 0, 2/2=1 du(%) 0
8       public static int Dec2Binary(int nDec){
9           if(nDec==0)
10              return 0;
11          else
12              return (nDec%2 + 10*Dec2Binary(nDec/2));
13      }
14  }
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
PS D:\_Teach\01.TDT\08.CTDL\Lab21.22\Lab3> java TestExer5.java
Test Dec2Binary 8=1000
```

Exercise 7

(a) Find and return the minimum element in an array. The array and its size are given as parameters.

Function : findMinArr(Arr[], n)

Arr = {7, -3, 9 , -8} , n=4

Min (Arr) = -8 (Max(Arr)= 9)

- Base case : n=1 -> return Arr[0]
- Recursive case: Min(Arr[**n-1**], findMinArr(Arr, **n-1**))

findMinArr(Arr,4): -8

Min(Arr[3], findMinArr(Arr,3)) Min(-8,-3) -> -8

Min(Arr[2], findMinArr(Arr,2)) Min(9, -3) -> -3

Min(Arr[1],findMinArr(Arr,1)) Min(-3,7) -> -3

Return Arr[0]=7

TestExer7.java

D: > _Teach > 01.TDT > 08.CTDL > Lab21.22 > Lab3 > TestExer7.java > TestExer7 > E7a_findMinArr(int[], int)

```java
import java.lang.*;
public  class TestExer7{
    public static void main(String[] args){
        int A[]={7,-3,9,-8,-5,6};
        System.out.println("Test 7a: find min of arr: " + E7a_findMinArr(A,A.length));
    }
    // 7a. Find and return the minimum element in an array
    public static int E7a_findMinArr(int Arr[],int n){
        if(n==1)
            return Arr[0];
        else
            return Math.min(Arr[n-1],E7a_findMinArr(Arr, n-1));
    }
}
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE powershell

```
PS D:\_Teach\01.TDT\08.CTDL\Lab21.22\Lab3> java TestExer7.java
Test 7a: find min of arr: -8
```

(d)

$$P(n,r) = \begin{cases} n(n-1)(n-2)\cdots(n-r+1), & n >= r > 0 \\ 1, & \text{otherwise} \end{cases}$$

Base case: n=r=1 -> P(n,r)=1

Recursive: P(n,r)= n* P(n-1)*(n-r+1)

Examples : P(3,2)=3*P(2,2)*(3-2+1)

2*P(1,2)*(2-2+1)

(4b)

(b) $\sum_{i=1}^{n}(i!)$

1! + 2! + 3! +4! + … + n!

(4c)

(c) $\prod_{i=1}^{n}(i!)$

1^1.2^2.3^3….n^n

1!.2!.3!….n!

Exercise 8

Using Linked List in **Lab 1** for this exercise.

(a) Implement method *addSortedList(E item)* to insert new element to a sorted linked list, that means we have to find the first node whose value is bigger than item and insert before it.

```
class ListNode <E> {
    protected E element;
    protected ListNode <E> next;

    /* constructors */
    public ListNode(E item) { element = item; next = null; }
    public ListNode(E item, ListNode <E> n) { element = item; next = n; }

    /* get the next ListNode */
    public ListNode <E> getNext() {
        return this.next;
    }

    /* get the element of the ListNode */
    public E getElement() {
        return this.element;
    }
}

import java.util.*;
public interface ListInterface <E> {

    public boolean isEmpty();
    public int     size();
    public E       getFirst() throws NoSuchElementException;
    public boolean contains(E item);
    public void    addFirst(E item);
    public E       removeFirst() throws NoSuchElementException;

    public void    print();

    // ....etc....
    // ....etc....
}

import java.util.*;

class BasicLinkedList <E> implements ListInterface <E> {
```

```java
protected ListNode <E> head = null;
protected int num_nodes = 0;

public boolean isEmpty() {
    return (num_nodes == 0);
}

public int size() {
    return num_nodes;
}

public E getFirst() throws NoSuchElementException {
    if (head == null)
        throw new NoSuchElementException("can't get from an empty list");
    else return head.element;
}

public boolean contains(E item) {
    for (ListNode <E> n = head; n!= null; n=n.next)
        if (n.getElement().equals(item)) return true;

    return false;
}

public void addFirst(E item) {
    head = new ListNode <E> (item, head);
    num_nodes++;
}

public E removeFirst() throws NoSuchElementException {
    ListNode <E> ln;
    if (head == null)
        throw new NoSuchElementException("can't remove from an empty list");
    else {
        ln = head;
        head = head.next;
        num_nodes--;
        return ln.element;
    }
}

public void print2() throws NoSuchElementException {
    if (head == null)
        throw new NoSuchElementException ("Nothing to print...");
```

```
        ListNode <E> ln = head;
        System.out.print ("List is: " + ln.element);
        for (int i=1; i < num_nodes; i++) {
            ln = ln.next;
            System.out.print(", " + ln.element);
        }
        System.out.println(".");
    }

    public void print() throws NoSuchElementException {
        if (head == null)
            throw new NoSuchElementException("Nothing to print...");

        Iterator <E> itr = iterator();
        System.out.print("List is: " + itr.next());
        while (itr.hasNext())
            System.out.print(", " + itr.next());
        System.out.println(".");
    }

    public Iterator<E> iterator() {
        return new LinkedListIterator();
    }

    private class LinkedListIterator implements Iterator<E> {
        private ListNode<E> current = head;

        public boolean hasNext() {
            return current != null;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }

        public E next() {
            if (!hasNext())
                throw new NoSuchElementException();
            E element = current.element;
            current = current.next;
            return element;
        }
    }
}
```

```
class SortedLinkedList <T extends Comparable<? super T>>
                                                extends BasicLinkedList<T> {

    private ListNode<T> insert(ListNode<T> p, T v) {
        //System.out.println("inside insert(ListNode<T> p, T v)");
        if (p == null || v.compareTo((T) p.element) < 0) {
            return new ListNode<T>(v, p);
        } else {
            p.next = insert(p.next, v);
            return p;
        }
    }

    public void insert(T v) {
        head = insert(head, v);
        num_nodes++;  // need to do this one ourselves
    }

    public void printLL() {
        System.out.print("Sorted List in order: ");
        printLL(head);
        System.out.println();
    }

    private void printLL(ListNode<T> n) {
        if (n != null) {
            System.out.print(n.element + " ");
            printLL(n.next);
        }
    }

    public void printRev() {
        System.out.print("Sorted List in reversed order: ");
        printRev(head);
        System.out.println();
    }

    private void printRev(ListNode<T> n) {
        if (n != null) {
            printRev(n.next);
            System.out.print(n.element + " ");
        }
    }
```

```
}
```

```
public class TestSortedList {

    public static void main(String[] args) {
        SortedLinkedList<Integer> sl = new SortedLinkedList<Integer> ();
        sl.insert(5);
        sl.insert(4);
        sl.insert(10);
        sl.insert(2);
        sl.insert(3);
        sl.insert(1);
        sl.insert(8);
        sl.print();

        sl.printLL();
        sl.printRev();
    }
}
```

(b) Suppose we have a linked list contains integer numbers, do the following requirements:
- Count all even numbers.
- Sum all numbers.

Yêu cầu: Make use of SortLinkedList, you are required to conduct the (b) with recurision.