Homework 2

Problem 1

```
import java.util.*;
import java.lang.*;
import java.io.*;

class Problem1
{
   public static void main (String[] args) throws java.lang.Exception
   {
      int []L = {1,2,3,4,5,6,7,8,9,10};
      int []P = {2,4,6};
      int i,j;
      int lenthp,lenthl;
      lenthp = P.length;
      lenthl = L.length;

      for(i=0;i<lenthp;i++)
      {
         if(P[i] < lenthl)
         {
            System.out.println("Element of " + P[i] + "in array L is " + L[P[i]]);
         }
         else
         {
            System.out.println("Element of index " + P[i] + "is not in array L");
         }
      }
   }
}
```

Problem 2

A).Implementation of intersection(L1, L2) procedure:

```
// intersection procedure implementation

public static
List intersection(
List L1, List L2)
```

```
{
List newList =
new LinkedList();
/* declare two variables to show the index in list 2 and list 1 */
int idx1 = 0;
int idx2 = 0;
/* verify the magnitude of the index */
while(idx1 < L1.size() && idx2 < L2.size())
{
if(((Comparable) L1.get(idx1)).compareTo(L2.get(idx2)) < 0)
{
// increase the index of L1 by 1
idx1++;
}
/* verify whether the current element of L1
is greater than the current element of L2 */
else if(((Comparable) L1.get(idx1)).compareTo(L2.get(idx2)) > 0)
  {
// increase the index of L2 by 1 idx2++;
}
else
{
/* add L1 to the new list */
newList.add(L1.get(idx1));
// increase the index of L1 by 1 idx1++;
// increase the index of L2 by 1 idx2++;
}
}
// return the intersection of L1 and L2 which is the newList;
} // end of intersection procedure
```

B).Program to test the intersection(L1, L2) procedure:

```
//This is nearly the same as part A
import java.util.LinkedList;
public class SortedListsIntersection
{
public static
List intersection(
List L1, List L2)
{
/* create a new list for intersection of L1 and L2 */
List newList =
new LinkedList();
int idx1 = 0;
```

```
int idx2 = 0;
while(idx1 < L1.size() && idx2 < L2.size())
{
is less than the current element of L2 */ if(((Comparable) L1.get(idx1)).compareTo(L2.get(idx2))
< 0)
{
// increase the index of L1 by 1
idx1++;
}
else if(((Comparable) L1.get(idx1)).compareTo(L2.get(idx2)) > 0)
{
// increase the index of L2 by 1
idx2++;
}
else
  {
/* add the current element of L1 to the new list */
newList.add(L1.get(idx1));
// increase the index of L1 by 1 idx1++;
// increase the index of L2 by 1 idx2++;
}
}
// return the intersection of L1 and L2 which is the  newList;
} // end of intersection procedure

C.
public static void main(String[] args)
{
// create two lists L1 and L2 of the integers List L1 = new LinkedList();
List L2 = new LinkedList();
// add several integers to L1 in sorted order L1.add(10);
L1.add(30);
L1.add(50);
L1.add(60);
L1.add(80);
// add several integers to L2 in sorted order L2.add(20);
L2.add(40);
L2.add(50);
L2.add(60);
L2.add(70);
L2.add(90);
// display the integers of L1 System.out.println("L1: " + L1);
// display the integers of L2 System.out.println("L2: " + L2);
List L3 = intersection(L1, L2);
```

```
// display the intersection of L1 and L2 System.out.println("\nintersection(L1, L2): " + L3);
}
}
```

Problem 3

StackNode.java:

```
import java.io.*;
public int prev;
public int value;
public StackNode(int p, int v) {
value=v; prev=p; }
}
```

ArrayStack.java:

```
import java.util.*;
import java.io.*;

{
boolean[] freeIndex;
int [] stackPointers = {-1, -1};
int stackSize=10;

 StackNode[] buff=new StackNode[stackSize*2]; // Determining the size of the array
public ArrayStack(int sz)
{
buff = new StackNode[stackSize]; freeIndex = new boolean[stackSize]; for(int i=0; i
freeIndex[i] = true;
}

{
for(int i=0; i< stackSize; i++)
{
if(freeIndex[i])
return i;
}
System.out.println("Stack Array is FULL");
return -1;
}
//Insert elements to the stack
public void push(int stackNum, int value)
{
int index = getNextFreeIndex();
```

```
if(index < 0)
{
System.out.println("Insertion failed!....");
return;
}
int lastIndex = stackPointers[stackNum];
stackPointers[stackNum] = index;
freeIndex[index] = false;
buff[index] = new StackNode(lastIndex, value);
}

  //Delete elements from the stack public int pop(int stackNum)
{
if(stackPointers[stackNum] >= 0) {
int item = buff[stackPointers[stackNum]].value;
int lastIndex = stackPointers[stackNum]; stackPointers[stackNum] =
buff[stackPointers[stackNum]].prev; buff[lastIndex]= null;
Comment
   freeIndex[lastIndex]= true;
return item;
}
System.out.println("Empty Stack"); return 0;
}
//display the elements of the stack
public String toString()
{
String result = "";
for(int i=0; i
{
if(buff[i] != null)
result = result+ " \n Node Element:"+buff[i].value+"\n"; }
return result;
}
//Main method
public static void main(String[] args)
{

ArrayStack stack = new ArrayStack(1);
//Pushing the elements into the each stack
stack.push(0, 1);
stack.push(0, 2);
stack.push(1, 3);
stack.push(1, 4);
stack.push(1, 5);
```

```
System.out.println("Elements of stack");
System.out.println(stack);
stack.pop(0);
System.out.println("After deletion at stack0");
System.out.println(stack);
stack.pop(1);
stack.pop(1);
System.out.println("After deletion at stack1");
System.out.println(stack);
System.out.println("After deletion at both stacks, Elements of stack are ");
System.out.println(stack);
}
}
```