Conder Shou

cs3544

Data Structures in Java

Homework 2: Written Portion

1.)

```
public void printLots(List<AnyType> L, List<Integer> P) {

	Iterator<AnyType> it2 = L.iterator();

	int previous = 0; // keeps track of the position of L

	for (int elem: P) {

		for (int i = previous; i < elem; i++) {
			it2.next();
			previous++;
		}

		System.out.println(it2.next());
		previous++;

	}
 }
```

2.)

The idea is to see if the two elements in each list matches, and then check if it equals the value of the element obtained beforehand. If it passes both these tests, then the current element is added to the third list.

If the two elements are not equal, then the list whose element is lowest advances further.

PSEUDOCODE:

```
ListIterator iterator1  = list1.iterator();
ListIterator iterator2 = list2.iterator();
LinkedList<AnyType> list3 = newLinkedList<AnyType>();

if (iterator1.hasNext() && iterator2.hasNext()) {
	list1Node = iterator1.next();
	list2Node = iterator2.next();
}
while (list1Node != null && list2Node != null) {
	int comparison = list1Node.compareTo(list2Node);

	if (comparison == 0) {			// 0 would indicate that elements are equal
```

```
                    if (iterator1.hasNext() ) {
                        list1NodeTrial = iterator1.next();
                            if (list1NodeTrial.compareTo(list1Node) == 0) { //duplicate
                                list1Node = list1NodeTrial;

                            } else if (list1NodeTrial.compareTo(list1Node) < 0 ||
list1NodeTrial.compareTo(list1Node)  > 0 )
                                list1Node = list1NodeTrial;
                                list3.add(list1Node);

                    } else {
                        list1Node = null;

                    if (iterator2.hasNext() ) {
                        list2Node = iterator2.next();
                    } else {
                        list2Node = null;



                } else if (comparison < 0 ) {       // list1Node is less than list2Node
                    if (iterator1.hasNext() ) {
                        list1Node = iterator1.next();
                    } else if {
                        list1Node = null;

                } else if (comparison > 0) {        // list1Node is greater than list2Node
                    if (iterator2.hasNext() ) {
                        list2Node = iterator2.next();
                    } else if {
                        list2Node = null;
                    }
            }
```

3.)

```
public class Written3 {

	public static class MyStack<AnyType> {
		private AnyType array[];
		private int counter1 = -1;
		private int counter2;

		public MyStack(int capacity) {
			array = (AnyType[]) new Object[capacity];
			counter2 = array.length;
		}

		public void push1(AnyType elem) {
			array[++counter1] = elem;
		}

		public AnyType pop1() {
			return array[counter1--];
		}

		public void push2(AnyType elem) {
			array[--counter2] = elem;
		}

		public AnyType pop2() {
			return array[counter2++];
		}
	}
	public static void main(String[] args) {
		MyStack<Integer> aStack =  new MyStack<>(10);
		aStack.push1(5);
		aStack.push2(3);
		aStack.pop1();
		aStack.pop2();
	}
}
```

4.)

a.

i. Move 4 to S2
ii. Move 3 to S2
iii. Move 1 to Output
iv. Move 8 to S1
v. Move 2 to Output
vi. Move 7 to S1
vii. Move 6 to S1
viii. Move 9 to S3
ix. Move 5 to S1

Now we can move these cars from the holding tracks to the output for the sorted arrangement like so:

x. Move 3 to Output
xi. Move 4 to Output (Emptying S2)
xii. Move 5 to Output
xiii. Move 6 to Output
xiv. Move 7 to Output
xv. Move 8 to Output (Emptying S1)
xvi. Move 9 to Output (Emptying S3)

Resulting Output: 9,8,7,6,5,4,3,2,1

b.

An example of a train of a length 9 that cannot be rearranged in increasing order using 3 holding tracks is:

259781436