Natalie Sayegh (nys2112)
Data Structures: Problem Set #2
October 10, 2016

1. Problem 1

```
//Assume: L and P are sorted and have valid elements (ex no negative integers on P)
void printLots(List<Integer> L, List<Integer> P) {
	Iterator<Integer> itrL = L.iterator(); //get iterators for lists L and P
	Iterator<Integer> itrP = P.iterator();
	int posOnL = -1; //tracks position of iterator on list L
	int valAtIndex = 0; //stores value of L at given index, arbitrarily initialized to 0
	while(itrP.hasNext() && itrL.hasNext()) {
		int index = itrP.next();
		for(int i = posOnL; i < index; i++) { //advance iterator to specified index on L
			if(itrL.hasNext()) {
				valAtIndex = itrL.next();
				posOnL++;
			}
		}
		if(posOnL == index) { //L didn't run out of elements inside the for loop
			System.out.println(valAtIndex);
		}
	}
}
```

2. Weiss 3.4

//Assume: L1 and L2 are sorted and contain elements of the same Comparable type

```
LinkedList<AnyType> findIntersection(LinkedList<AnyType> L1, LinkedList<AnyType> L2)
{
        LinkedList<AnyType> intersection = new LinkedList<>(); //construct empty list
        Iterator<AnyType> itr1 = L1.iterator(); //get iterators for L1 and L2
        Iterator<AnyType> itr2 = L2.iterator();
        while(itr1.hasNext() && itr2.hasNext()) { //while elements are left in both lists
                AnyType el1 = itr1.next();
                AnyType el2 = itr2.next();

                /*Since the lists are sorted: if el2 < el1, iterate across L2 until el2 >= el1, or the
                *end of the list is reached.
                *If el1 < el2, iterate across L1 until el1>=el2, or the end of the list is reached*/

                while(!(el1.compareTo(el2) == 0) && itr1.hasNext() && itr2.hasNext()) {
                        while(itr2.hasNext() && el1.compareTo(el2) > 0) { //el2<el1
                                el2 = itr2.next();
                        }
                        while(itr1.hasNext() && el1.compareTo(el2) < 0) { //el1<el2
                                el1 = itr1.next();
                        }
                }

                /*if the two elements match and the element isn't yet in the intersection list,
                *add the common element to the intersection list*/

                if(el1.compareTo(el2) == 0 && !(intersection.contains(el1))) {
                        intersection.add(el1);
                }
        }
        return intersection;
}
```

3. Weiss 3.24

```
public class TwoStackArray<AnyType> {
	AnyType[] array;
	int top1; //index of topmost element in stack 1
	int top2; //index of topmost element in stack 2
	int capacity; //max size of array

	public TwoStackArray(int maxSize) {
		array = (AnyType[]) new Object[maxSize];
		top1 = -1;  //stack 1 starts at the front and grows forward
		top2 = maxSize; //stack 2 starts at the back and grows backward
		capacity = maxSize;
	}

	void push1(AnyType x) { //push method for stack 1
		if(top2 - top1 > 1) { //if there are empty spaces left in the array
			top1++;
			array[top1] = x;
		}
		else {
			System.out.println("Overflow error!");
			System.exit(0);
		}
	}

	AnyType pop1() { //pop method for stack 1
		if(top1 == -1) { //no elements in stack 1
			return null;
		}
		else {
			AnyType val = array[top1];
			top1--;
			return val;
		}
	}

	void push2(AnyType x) { //push method for stack 2
		if(top2 - top1 > 1) {
			top2--;
			array[top2] = x;
		}
		else {
			System.out.println("Overflow error!");
			System.exit(0);
		}
	}
	AnyType pop2() { //pop method for stack 2
		if(top1 == capacity) { //no elements in stack 2
			return null;
		}
		else {
			AnyType val = array[top2];
			top2++;
			return val;
		}
	}
}
```

4. Problem 4
(a)
Given:
Input track I = [5, 9, 6, 7, 2, 8, 1, 3, 4]
Holding tracks S1 = [], S2 = [], S3 = []
Output track O = []

1 Move 4 from I to S1
  I = [5, 9, 6, 7, 2, 8, 1, 3]
  S1 = [4], S2 = [], S3 = []
  O = []
2 Move 3 from I to S1
  I = [5, 9, 6, 7, 2, 8, 1]
  S1 = [4, 3], S2 = [], S3 = []
  O = []
3 Move 1 from I to O
  I = [5, 9, 6, 7, 2, 8]
  S1 = [4, 3], S2 = [], S3 = []
  O = [1]
4 Move 8 from I to S2
  I = [5, 9, 6, 7, 2]
  S1 = [4, 3], S2 = [8], S3 = []
  O = [1]
5 Move 2 from I to O
  I = [5, 9, 6, 7]
  S1 = [4, 3], S2 = [8], S3 = []
  O = [2, 1]
6 Move 3 from S1 to O
  I = [5, 9, 6, 7]
  S1 = [4], S2 = [8], S3 = []
  O = [3, 2, 1]
7 Move 4 from S1 to O
  I = [5, 9, 6, 7]
  S1 = [], S2 = [8], S3 = []
  O = [4, 3, 2, 1]
8 Move 7 from I to S2
  I = [5, 9, 6]
  S1 = [], S2 = [8, 7], S3 = []
  O = [4, 3, 2, 1]
9 Move 6 from I to S2
  I = [5, 9]
  S1 = [], S2 = [8, 7, 6], S3 = []
  O = [4, 3, 2, 1]
10 Move 9 from I to S1
  I = [5]
  S1 = [9], S2 = [8, 7, 6], S3 = []

O = [4, 3, 2, 1]

11 Move 5 from I to O
I = []
S1 = [9], S2 = [8, 7, 6], S3 = []
O = [5, 4, 3, 2, 1]

12 Move 6 from S2 to O
I = []
S1 = [9], S2 = [8, 7], S3 = []
O = [6, 5, 4, 3, 2, 1]

13 Move 7 from S2 to O
I = []
S1 = [9], S2 = [8], S3 = []
O = [7, 6, 5, 4, 3, 2, 1]

14 Move 8 from S2 to O
I = []
S1 = [9], S2 = [], S3 = []
O = [8, 7, 6, 5, 4, 3, 2, 1]

15 Move 9 from S1 to O
I = []
S1 = [], S2 = [], S3 = []
O = [9, 8, 7, 6, 5, 4, 3, 2, 1]

(b)
I = [1, 9, 8, 7, 6, 5, 4, 3, 2]
The first eight cars (#s 2-9) would have to be moved to the holding stacks in order to free car #1 to the output. During this process, the lower-numbered cars—which should be moved to output after car #1—get buried at the bottom of the holding stacks. Since cars cannot be moved between holding stacks, there is no way to transfer the cars to the output in the desired order.