Data Structures Homework #2

By Ishaan Agrawal

Problem 1

The method that prints the elements of List L based on the index elements provided by List P is given below:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Written1 {

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

        Iterator<AnyType> itrL = L.iterator();
        Iterator<Integer> itrP = P.iterator();

        int value;
        int oldValue = -1;

        while (itrP.hasNext()) {
            value = itrP.next();

            if (L.size() < value) {
                System.out.println("The " + value + " index is out of List L"
                        + " bounds.");
            }

            else {
                for (int i = 0; i < value - oldValue - 1; i ++) {
                    itrL.next();
                }
            oldValue = value;
            System.out.println(itrL.next());
            }
        }

    }
```

This method uses only the methods provided for by the Collections interface. It does not use any other methods provided in the List interface. However, it does make use of Iterators.

Problem 2

This algorithm assumes that since the list is sorted, then each element in the list implements Comparable and therefore, each element can be arranged in an order based on the comparable feature.

```
public static List<AnyType> intersection(List<AnyType> L1, List<AnyType> L2) {

    Iterable<AnyType> itrL1 = L1.iterator();
    Iterable<AnyType> itrL2 = L2.iterator();

    List<AnyType> intersectingList = new List<AnyType>();

    while (itrL1.hasNext()) {

        while (itrL1 >= itrL2) {

            itrL2.next();
            if (itrL1 == itrL2) {
                intersectingList.add(itrL1);
                break;
            }
        }
    }

    return intersectingList;
}
```

Problem 3

```
public class MyStack<AnyType> {

    private final int DEFAULT_CAPACITY = 50;

    Object[] theArray;
    int topOfStack1;
    int topOfStack2;

    public MyStack() {

        theArray = (AnyType[]) new Object[DEFAULT_CAPACITY];
        topOfStack1 = 0;
        topOfStack2 = DEFAULT_CAPACITY - 1;
    }

    public void push1(AnyType element) throws StackOverflowError {

        if (size1() + size2() >= DEFAULT_CAPACITY)
            throw StackOverflowError;
        else
            theArray[topOfStack1++] = element;
    }
```

The rest of the routine follows onto the next page.

```
    public void push2(AnyType element) throws StackOverflowError {

        if (size1() + size2() >= DEFAULT_CAPACITY)
            throw StackOverflowError;
        else
            theArray[topOfStack2--] = element;
    }

    public AnyType pop1() {
        return (AnyType) theArray[--topOfStack1];
    }

    public AnyType pop2() {
        return (AnyType) theArray[++topOfStack2];
    }

    public AnyType top1() {
        topOfStack--;
        return (AnyType) theArray[topOfStack1++];
    }

    public AnyType top2() {
        topOfStack++;
        return (AnyType) theArray[topOfStack2--];
    }

    public boolean isEmpty1() {
        return (topOfStack1 == 0);
    }

    public boolean isEmpty2() {
        return (topOfStack2 == DEFAULT_CAPACITY - 1);
    }

    public int size1() {
        return (topOfStack1);
    }

    public int size2() {
        return (DEFAULT_CAPACITY - topOfStack2 - 1);
    }
}
```

Problem 4

Part a

The given train car order is 5, 9, 6, 7, 2, 8, 1, 3, 4 and the desired order is 9, 8, 7, 6, 5, 4, 3, 2, 1.

Therefore, to change the current arrangement to ascending order, we take the following steps:

1. Shift Car 4 to holding track S1.
2. Shift Car 3 to holding track S1.
3. Shift Car 1 to the output track.
4. Shift Car 8 to holding track S2.
5. Shift Car 2 to the output track.
6. Shift Car 3 from holding track S1 to the output track.
7. Shift Car 4 from holding track S1 to the output track.
8. Shift Car 7 to holding track S2.
9. Shift Car 6 to holding track S2.
10. Shift Car 9 to holding track S1.
11. Shift Car 5 to the output track.
12. Shift Car 6 from holding track S2 to the output track.
13. Shift Car 7 from holding track S2 to the output track.
14. Shift Car 8 from holding track S2 to the output track.
15. Shift Car 9 from holding track S1 to the output track.

This yields the following arrangement: 9, 8, 7, 6, 5, 4, 3, 2, and 1.

Part b

The following arrangement of cars cannot be rearranged using 3 holding tracks.

$$1, 9, 8, 7, 6, 5, 4, 3, 2$$

The following steps are taken to try are arrange the train cars.

1. Shift Car 2 to holding track S1.
2. Shift Car 3 to holding track S2.
3. Shift Car 4 to holding track S3.

Now, wherever we move Car 5, it will be above either Car 2, Car 3 or Car 4. As a result, when we need to shift either of these cars onto the output track, we will not be able to access them as Car 5 is above one of them in the stack. This situation worsens when we move the cars from 6 through 9 onto the stacks. In the best case, we will be able to access Car 2 and Car 3 (if all cars from 5 through 9 are placed in S3. However, then it is impossible to access Car 4 and we as a result cannot arrange the cars in the desired order.