

# Data Structures Homework 2 Written

Sharon Chen syc2138

October 2, 2016

## 1 Problem 1

```
public static <E> void printLots(List<E> L, List<Integer> P)
{
    Iterator<E> itr = L.iterator();
    int position = 0;
    while (itr.hasNext())
    {
        E item = itr.next();
        if (P.contains(position++))
        {
            System.out.println(item);
        }
    }
}
```

## 2 Problem 2

```
public static <E> List<E> and(List<E> list1, List<E> list2)
{
    List<E> result = new List<>();
    Iterator<E> itr = list1.iterator();
    while (itr.hasNext())
    {
        Integer value = itr.next();
        if (list2.contains(value) && !result.contains(value))
        {
            result.add(value);
        }
    }
    return result;
}
```

### 3 Problem 3

```
public class MyStacks<E>
{
    public MyStacks()
    {
        stacks = new E[];
        index1 = -1;
        index2 = -1;
    }

    public MyStacks(int length)
    {
        stacks = new E[length];
        index1 = -1;
        index2 = length + 1;
    }

    /**
     * Adds element to top of specified stack.
     * @param stack the first stack (0) or the second stack (1)
     * @param element the element to be added
     */
    public void push(int stack, int element)
    {
        if (stack == 0) index1++;
        else index2--;
        if (index1 == index2) throw new StackOverflowException;
        else
        {
            if (stack == 0) stacks[index1] = element;
            else stacks[index2] = element;
        }
    }

    /**
     * Pops element at top of specified stack.
     * @param stack the first stack (0) or the second stack (1)
     */
    public int pop(int stack)
    {
        if (stack == 0) return stacks[index1--];
        else return stacks[index2++];
    }
}
```

```

    }

    /**
     * Peeks element at top of specified stack.
     * @param stack the first stack (0) or the second stack (1)
     */
    public int top(int stack)
    {
        if (stack == 0) return stacks[index1];
        else return stacks[index2];
    }

    private E[] stacks;
    private int index1;
    private int index2;
}

```

## 4 Problem 4

### 4.1 Part A

Given input track  $I$ , output track  $O$ , and holding tracks  $A$ ,  $B$ , and  $C$ , this is a solution:

1. 4 ( $I \rightarrow C$ )
2. 3 ( $I \rightarrow C$ )
3. 1 ( $I \rightarrow O$ )
4. 8 ( $I \rightarrow B$ )
5. 2 ( $I \rightarrow O$ )
6. 7 ( $I \rightarrow B$ )
7. 6 ( $I \rightarrow B$ )
8. 9 ( $I \rightarrow A$ )
9. 3 ( $C \rightarrow O$ )
10. 4 ( $C \rightarrow O$ )
11. 5 ( $I \rightarrow O$ )
12. 6 ( $B \rightarrow O$ )
13. 7 ( $B \rightarrow O$ )

14. 8     $(B \rightarrow O)$

15. 9     $(A \rightarrow O)$

#### 4.2    Part B

A train of length 9 that cannot be rearranged in increasing order using 3 holding tracks is this:

[3,2,1,9,8,7,6,5,4]