

# CS1020: DATA STRUCTURES AND ALGORITHMS I

## Tutorial 6 – Exception, Linked List Variants

(Week 8, starting 7 March 2016)

### 1. Exceptions

Examine the following code, especially how `calculate()` works:

```
class DivideBy4Exception extends Exception {}
class DivideBy100Exception extends DivideBy4Exception {}
class DivideBy400Exception extends DivideBy100Exception {}

/* Part (e): Poor design! DO NOT EVER Use Exceptions this way!! */

public class Mystery {

    public int calculate(int input) {
        int value = 0;
        for (int index = 1; index <= input; index++)
            value += buzz(index);
        return value;
    }

    public int buzz(int value) {
        try {
            if (value % 400 == 0)
                throw new DivideBy400Exception();
            if (value % 100 == 0)
                throw new DivideBy100Exception();
            if (value % 4 == 0)
                throw new DivideBy4Exception();
            return 0;
        } catch (DivideBy4Exception ex) {
            return 1;
        } catch (DivideBy100Exception ex) {
            return 0;
        } catch (DivideBy400Exception ex) {
            return 1;
        } finally {
            return 3;
        }
    }

    public static void main(String[] arr) { ... }
}
```

- (a) Why can't the program be compiled? Identify and **fix the error**, **THEN** attempt (b) to (e).
- (b) What do `calculate()` and `buzz()` do? What is the result of `calculate(2000)`, why?
- (c) **IF the finally clause is removed**, what will the result of `calculate(2000)` be, why?
- (d) If we do not want to handle Exceptions within `buzz()`, what alternatives are there?
- (e) Why is this program poorly designed? Why should we **NOT** write a program like this?

## 2. List ADT Implementations

In lectures, we have learned two List implementations – array-based and reference-based. ArrayList and Vector are array-based list implementations, while LinkedList is a reference-based implementation. Let us compare and contrast both implementations.

For a list containing N elements, around how many elements would be accessed/modified when:

- (a) Adding to end of the List (new index == N / tail)
- (b) Adding to front of the List (index == 0 / head)
- (c) Removing from front of the List (index == 0 / head)
- (d) Getting (accessing) any element, on average (from index == 0 to index == N-1)

## 3. Circular Linked List

Implement a method `swap(int index)` in the `CircularLinkedList<E>` class given to you, to swap the node at the given index with the next node.

If the index is larger than the size of the list, then the index wraps around. For example, if the list has 13 elements, then `swap(15)` will swap nodes at indexes 2 and 3.

```
public class CircularLinkedList<E> {  
  
    public int _size;  
    public ListNode<E> _head;  
    public ListNode<E> _tail;  
  
    public void addFirst(E element) {  
        _size++;  
        _head = new ListNode<E>(element, _head);  
        if (_tail == null)  
            _tail = _head;  
        _tail.setNext(_head);  
    }  
  
    public String toString() { ... }  
    public void swap(int index) { ... }  
}
```

A pre-condition is that index will be non-negative ( $\text{index} \geq 0$ ). **Restriction:** You are **NOT allowed** to create any **new nodes**, and you are NOT allowed to **modify the element** in any node.

[Hint: Consider all cases, and remember to update the necessary instance attributes!]

