Faculty of Science and Engineering

COMP125 Fundamentals of Computer Science
Workshop Week 11

Learning outcomes

By the end of this session, you will have learnt about traversing custom-built linked lists.

The all-important `Node` class

The `Node` class is one that holds,

a. some data (could be as simple as an integer, and as complex as an entire XML object ... or even more complex)

b. a **reference** to one or more `Node` objects.

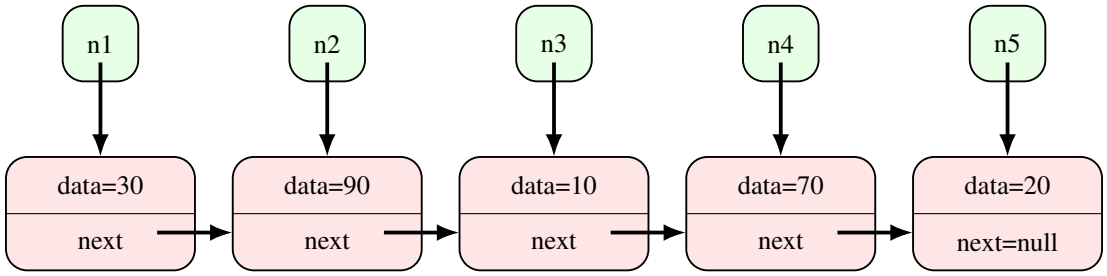
In the simplest case, we can have a `Node` with reference to only one another `Node`

```
1  public class Node {
2          private Node next;
3          private int data;
4
5          public int getData() {
6                  return data;
7          }
8          public Node getNext() {
9                  return next;
10         }
11
12         public void setData(int data) {
13                 this.data = data;
14         }
15         public void setNext(Node next) {
16                 this.next = next;
17         }
18
19         public Node(int data, Node next) {
20                 setData(data);
21                 setNext(next);
22         }
23 }
```

Take a look at the following client code:

```
class Client {
        public static void main(String[] args) {
                Node n5 = new Node(20, null);
                Node n4 = new Node(70, n5);
                Node n3 = new Node(10, n4);
                Node n2 = new Node(90, n3);
                Node n1 = new Node(30, n2);
        }
}
```

The memory diagram for the above code is given below:



Traversing a collection of nodes linked together

```
Node current = n1;
while(current != null) {
        //use current.getData() as you wish
        current = current.getNext();
}
```

Example 1 - Adding up all the items

```
Node current = n1;
int total = 0;
while(current != null) {
        total = total + current.getData();
        current = current.getNext();
}
```

Example 2 - Adding up all the positive items

```
Node current = n1       ;
int total = 0;
while(current != null) {
        if(current.getData() > 0) {
                total = total + current.getData();
        }
        current = current.getNext();
}
```

Example 3 - Two in a row?

This version assumes there are at least two connected nodes starting at `a`.

```
1
2  Node current = n1;
3  boolean twoInARow = false;
4  while(current.getNext() != null) { //we need next one too
5          int item1 = current.getData();
6          int item2 = current.getNext().getData();
7          if(item1 == item2) {
8                  twoInARow = true;
9          }
10         current = current.getNext();
11 }
```

Example 4 - Two in a row? Version 2

This version makes no assumptions.

```
1
2  Node current = n1;
3  boolean twoInARow = false;
4  while(  current != null
5                  && current.getNext() != null
6                  && twoInARow == false ) {
7          int item1 = current.getData();
8          int item2 = current.getNext().getData();
9          if(item1 == item2) {
10                 twoInARow = true;
11         }
12         current = current.getNext();
13 }
```

Questions: Traversing `Node` class

Consider the same client ($n1 \rightarrow n2 \rightarrow n3 \rightarrow n4 \rightarrow n5 \rightarrow null$) for all questions in this section.

1. Write a piece of code that counts the number of even numbers in the "list".
2. Write a piece of code that adds up all the even numbers in the "list".
3. Write a piece of code that determines the first index at which an item between 10 and 18 exists in the "list". The first node in the list is said to have index 0.
4. Write a piece of code that stores `true`, in a variable *isAscending*, if the "list" is in ascending order (and `false` if not).

Linked List class

We can formally keep a track of linked nodes by storing the first node in a class.

```
class MyLinkedList {
        private Node head; //head refers to first node - ALWAYS

        public void addToFront(int data) {
                Node n = new Node(data);
                n.setNext(head); //the current first node is next to n
                head = n; //now n is the first node
        }
}
```

Questions: Custom-built linked list

1. Define an instance method that returns the number of negative items in the list. A template is for this example.

```
public int countNegatives() {
        /*
                no need to pass anything
                as the first node in the
                list (head) is an instance
                variable
        */
        return 0; //to be completed
}
```

2. Define an instance method that returns `true` if every single item in the list is within a given range, `false` otherwise.

```
/**
* @param min
* @param max (assume min <= max)
* @return true if all items in list
* starting at head are in the range
* [min, max], false otherwise
*/
public boolean allInRange(int min, int max) {
        return false; //to be completed
}
```

3. Define an instance method that returns `true` if all the items in the list are the same (for example 16 -> 16 -> 16 -> 16 -> null) and `false` otherwise.

4. Define an instance method that returns `true` if every item is different from other items, and `false` otherwise.

5. Define an instance method that returns `true` if the list remains unchanged when reversed, `false` otherwise.

6. Define an instance method that returns a reference to the starting node beyond which the list is sorted in ascending order. If the list is empty, return `null`. If the list is not empty, in the worst case scenario (for example, 40 -> 30 -> 20 -> 10 -> null), it will return a reference to the last node (containing 10)