Lab 5. Linked Structures

Theme. In this practical, you will:

- use a linear linked structure to model a linear collection of objects
- practise how to add a node to a linear linked structure
- practise how to traverse the nodes within a linear linked structure

Key concepts: linear linked structure, manipulation of a linked structure

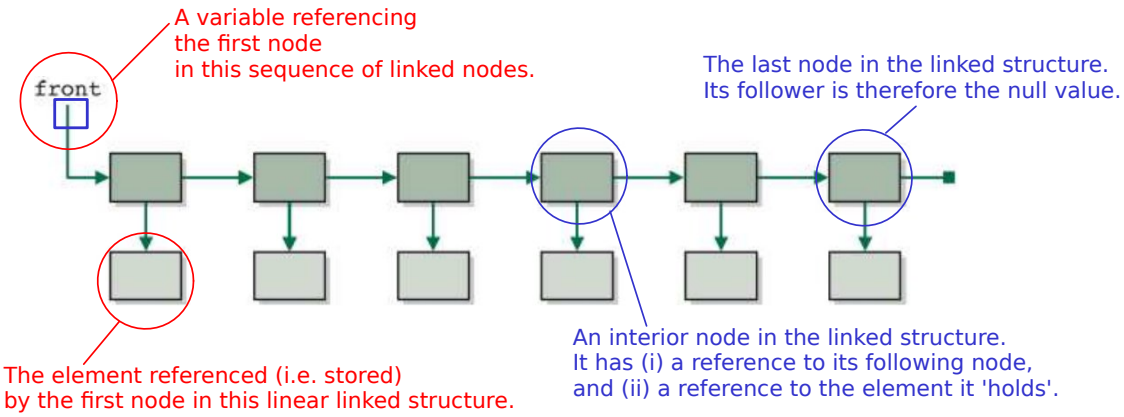
Required file(s): `lab5.zip`

1 Getting started...

1. In a web browser, download the archive `lab5.zip` from Blackboard and extract its contents into your default `Eclipse` workspace for this module.
2. Start up `Eclipse`.
3. Making use of the contents of your extracted archive, create a new Java project named **mother_linked_structure** in your `Eclipse` workspace using the contents of your extracted archive.

2 Working with Linked Structures

A *linked structure* uses object reference variables to link one object to another. Each object in a linked structure is known as a **node**. A linear linked structure (also commonly known as a *linked list*) is suitable for modelling a *linear* collection of objects. Using such a structure, there is no need to use any array for storing the elements of a collection.



The fundamental building blocks of a linear link structure can be represented using a Java class called `LinearNode` which has two fields: `element` and `next`. A `LinearNode` object represents a node within a linked list. While a node can be added to anywhere within a linked list, adding a node to the front of a linked list is relatively straight-forward and can be done in constant time. To obtain an element within a linked list, simply locate the node where the element is stored and return its contents. The following exercises give you a chance to put these concepts in writing using Java code.

The `Eclipse` Java project `mother_linked_structure` has two packages.

1. Package `dsa` contains:
   - `LinearNode` – A class for modelling a node in a linear linked structure.
2. Package `ancestry` contains:
   - `Female` – A class for modelling a female person who has a name.
   - `MotherLink` – A class to model a family line that involves females only. The idea is to form a link based on a daughter and mother relationship. The ancestral information is kept in a *reversed* order. The last in line in a mother link is the *youngest* within the family line.

     This class uses a linear linked structure to model the ancestral information. The first node in the linear linked structure holds the female in the most recent generation. The last node holds the ultimate ancestor. For example, if Clare's mother is Belle and Belle's mother is Angela, the linear structure should look like the following:

     Clare ➟ Belle ➟ Angela

Your Tasks

Your task is to complete the given implementation of `LinearNode` and `MotherLink` according to the specification. A successful completion of the implementation should enable the ancestral information of the last in line to be written to the standard output.

> **Hint**: As usual, the rough locations where you are expected to add your Java code and relevant hints for accomplishing the tasks have been marked throughout the given Java programs. Look out for **block comments** that include a sequence of *four* exclamation marks, i.e.:
>
> ```
> /* !!!!  ...  ...  */
> ```
>
> The locations where you are expected to pay particular attention on the given Java code have also been annotated. Look out for **block comments** that include a sequence of *four* plus signs, i.e.:
>
> ```
> /* ++++ ...  ...  */
> ```

1. Define *two* fields (i.e. instance variables) in class `LinearNode`.
2. Complete the implementation of the constructor of `LinearNode` which enables a new `LinearNode` object to be created. Such a `LinearNode` object does not hold any an element nor have a "follower".
3. Complete the implementation of the constructor of `LinearNode` which enables a new `LinearNode` object with an element, but no "follower", to be created.
4. In class `MotherLink`, define a field named `lastInLine` for keeping a `LinearNode<Female>` object.
5. Complete the implementation of the constructor of `MotherLink`.
6. Complete the implementation of the method `addNextInLine` in `MotherLink`. This method should add a new node to the front of the mother link.
7. Complete the implementation of the method `size` in `MotherLink`. This method should return the number of nodes within the mother link. For practice at traversing, linked lists im-

plement this using a loop to count the nodes (rather than maintaining an instance field `count` which keeps track of the number of nodes created so far).

8. Complete the implementation of the method `toString` in `MotherLink`. This method should return a `String` representation of the complete mother link.

3 Testing

Now test your implementation to see if it meets the above requirements. Use the given `main(String[])` method in `MotherLink` to perform a quick unit testing. If your implementation is correct, the output should be:

```
This mother link has 5 people.
 They are:
      Emma who is the daughter of...
              Dorothy who is the daughter of...
                      Clare who is the daughter of...
                              Belle who is the daughter of...
                                      Angela.
```

4 Further Challenges

1. Suppose the following Java statement has just been executed:

   ```
   MotherLink link = new MotherLink(new Female("Angela"));
   ```

   Using an *object diagram*, describe the **state** of object `link`.

> **Hint**: The state of an object refers to the fields of the object and their values at a specified point during the runtime of the application.
>
> If you are not familiar with drawing object diagrams, see `https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/` for a brief introduction.

2. Suppose the following Java statements have just been executed:

   ```
   MotherLink link = new MotherLink(new Female("Angela"));
   link.addNextInLine(new Female("Belle"));
   ```

   Using an *object diagram*, describe the **state** of object `link`.

3. The present implementation of an ancestry link provides the implementation basis for modelling an ADT. Name this ADT.

4. Let's assume that a single female child policy is to be adopted. Each family is therefore allowed to have one daughter only.

   Rather than keeping track of the youngest female in the family line, the ancestry link will now keep track of the oldest female in the family line, i.e. the first in line.

(a) Describe the modification to class `MotherLink` required to support the addition of a new female to the family line.

(b) Which ADT would the new implementation of class `MotherLink` be suitable for modelling?

5. **Array** and **linear linked structure** are two types of data structures that can be used to implement an ADT. Which *one* of these data structures is better for implementing a **bounded map**? Making reference to the characteristics of these data structures and the time efficiency of typical **bounded map** operations, justify your answer.

5 Further Programming Exercises

1. The current output of the class `MotherLink` is not ideal. It assumes that there are at least two people in the mother link. However, the implementation of this class does not endorse this assumption. Modify the method `toString` so that appropriate output could be produced even when the mother link:

   - is empty, i.e.:

     ```
     This mother link is empty.
     ```

   - contains *only* one member, i.e.:

     ```
     This mother link has one person.
      She is:
     Angela.
     ```

2. The present implementation keeps track of the youngest female within a mother link, i.e. the last in line. Implement another version of mother link which keeps track of the oldest female in the family line, i.e. the first in line.