Individual Exercises for Data Structures (Laboratory) (Prelim)

Prepared by

The faculty of the IT/CS department

August 2025

Vision – Mission Statement

Saint Louis University is an excellent missionary and transformative educational institution zealous in developing human resources who are creative, competent, socially involved and imbued with the Christian spirit.

Note:

The programming exercises specified below will help you review and/or gain mastery of the fundamentals of the data structures in Java. For the required submissions, you have to submit a compressed electronic folder that contains the collection of the files (source code files) for the exercises.

Assumption:

The Java Development Kit (JDK) and IntelliJ IDEA are both installed in your computing device.

Working folder (project):

For individual work submission, you are to create a project with the naming convention as follows: **<CourseNumber>_<ClassCode>_<Lastname><Firstname><MI>** (e.g.**ICS211_9300_DelaCruzJuan)**. Use the specified package names for the programs that you are going to create for each problem.

For your submission:

Individual work submission will be the archived/compressed form of the project containing all the necessary files. This may be done by accessing the menu **File | Export | Project to Zip file…** within IntelliJ IDEA. However, partial outputs/programs will be collected during class meetings as needed.

Prelim Programming Exercises

Consider the interface below:

```
package prelim;
import java.util.NoSuchElementException;
public interface MyList<E> {
   public int getSize();
   public void insert(E data) throws ListOverflowException;
   public E getElement(E data) throws NoSuchElementException;
   public boolean delete(E data); // returns false if the data is not deleted in the list
   public int search(E data); // returns index of data, else -1 is return
}
```

Other than creating the interface shown above, the `ListOverflowException` class will have to be created. `NoSuchElementException` class is accessed from java.util package.

Place all your outputs in a package named prelim within your project.

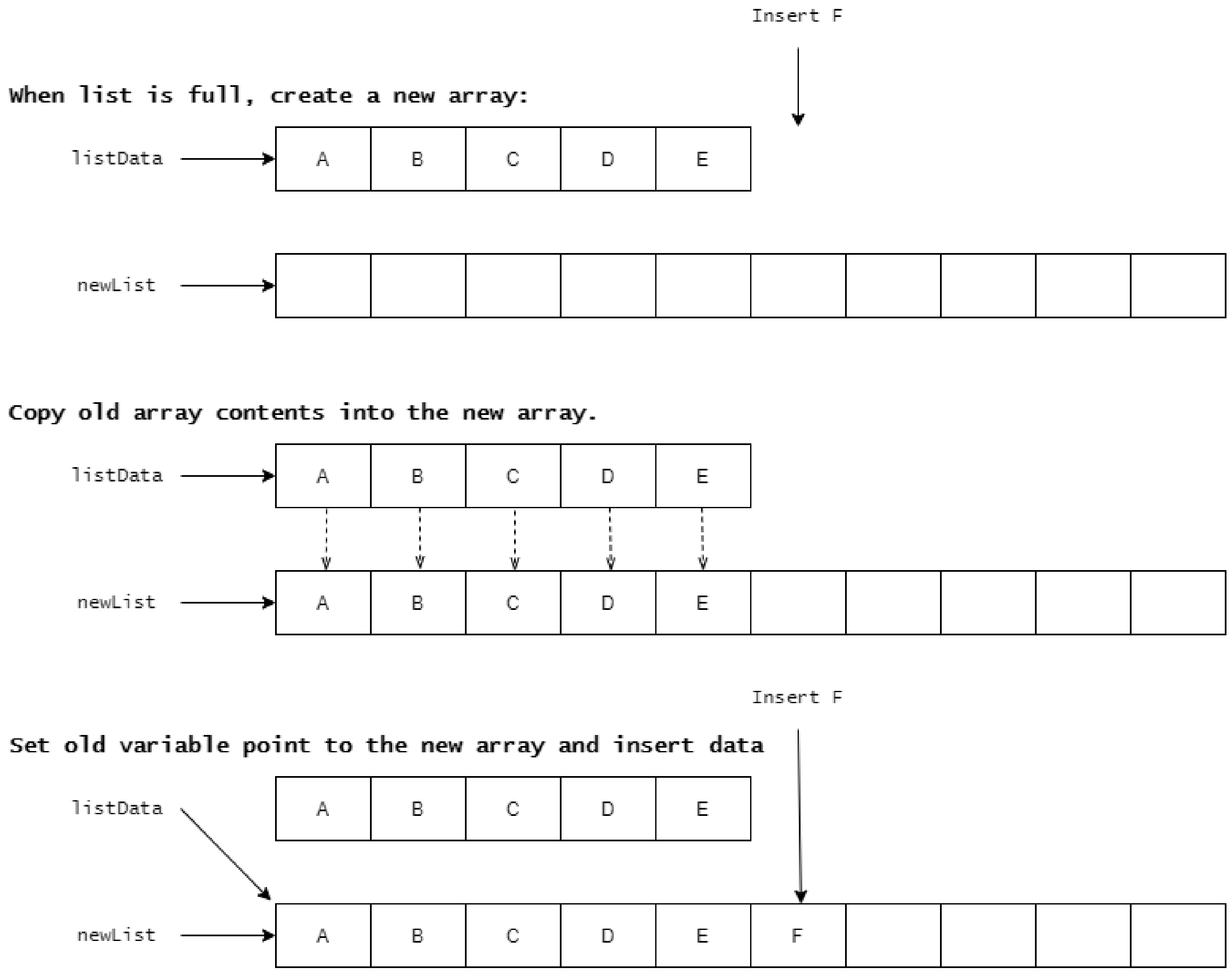
Create implementations of the above interface in different ways as stipulated below.

1. Use an array object to store the elements of the list object. This array object will have a fixed size of five (5) and that an attempt to insert an element into the array when full will result to a ListOverflowException.
   Filename: `MyFixedSizeArrayList.java`

   Expected Additional Task
   Create an executable class(class with a main method) such that the executable class will apply/use the MyFixedArrayList class. A simple application will do. [e.g. An application that will allow the user to store their valuable personal properties in a List. The application may facilitate the viewing of the details about the personal property (e.g. model number, color, status, etc.)]

2. Use an array object to store the elements of the list object. Although the initial size of the array is 5, this differs from the first situation because if the array is full and an element is to be inserted, a new array object with a size that is twice the size of the "old" array will be created. All the elements of the "old" array will be copied to the new array before inserting the new element. You may refer to the illustration below.
   Filename: `MyGrowingArrayList.java`

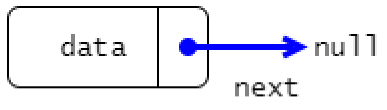
   Expected Additional Task
   Create an executable class(class with a main method) such that the executable class will apply/use the MyGrowingArrayList class. A simple application will do. [e.g. An application that will allow the user to store their completed tasks for one course in a List. The application may facilitate the viewing of the details about the completed tasks (e.g. project name, date assigned, date submitted, etc.)]


Insert F
When list is full, create a new array:
listData
A
B
C
D
E
newList
Copy old array contents into the new array.
listData
A
B
C
D
E
newList
A
B
C
D
E
Insert F
Set old variable point to the new array and insert data
listData
A
B
C
D
E
newList
A
B
C
D
E
F

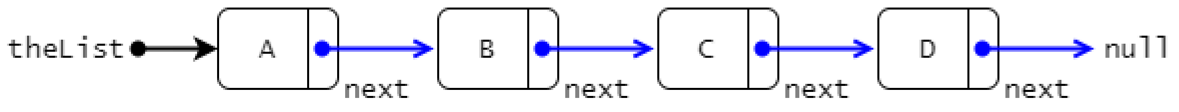
3. Use a singly-linked node class to implement the linked list implementation. Details of the Node class to be used are as follows.
   Filename: `MySinglyLinkedList.java`

```
Node<T>
- data: T
- next: Node<T>
+ Node(T data)
+ getData(): T
+ setNext(node: Node<T>): void
+ getNext(): Node<T>
```

Node Illustration



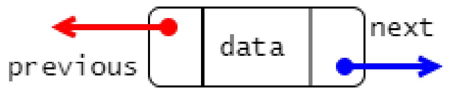
The Linked list using a singly-linked node:



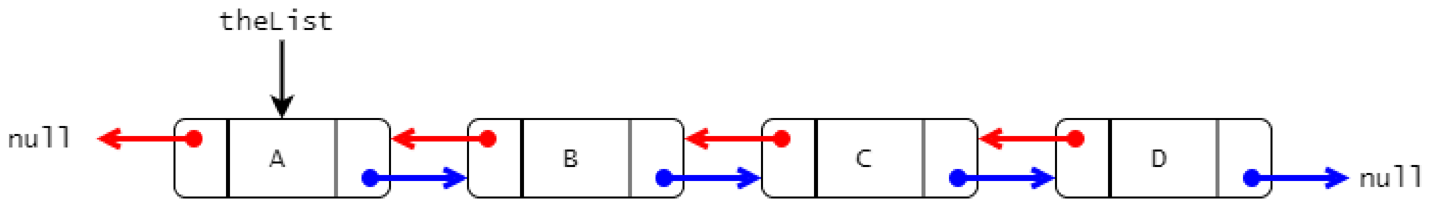
4. Use a doubly-linked node class to implement the list. Details of the DoublyLinkedNode class are shown below.
   Filename: `MyDoublyLinkedList.java`

```
DoublyLinkedNode<T>
- data: T
- next: Node<T>
+ Node(T data)
+ getData(): T
+ setNext(node: Node<T>): void
+ getNext(): Node<T>
```

DoublyLinkedNode Illustration

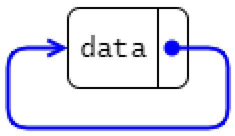


The doubly linked list illustration:

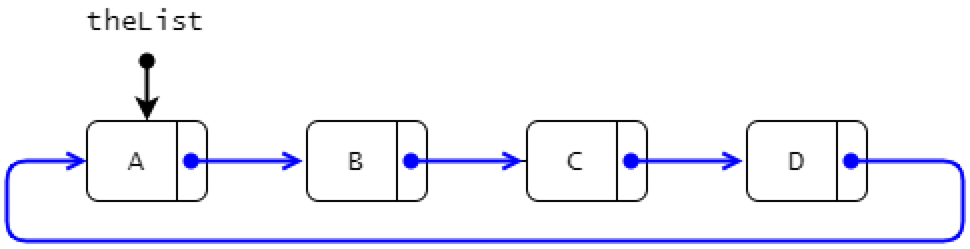


5. Use a singly-linked circular node class to implement the list. Refer to the illustration below.
   Filename: `MySinglyLinkedCircularList.java`

The singly-linked circular node:

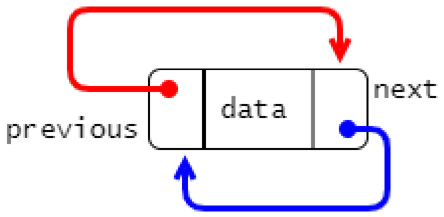


The Linked list using a singly-linked circular node:

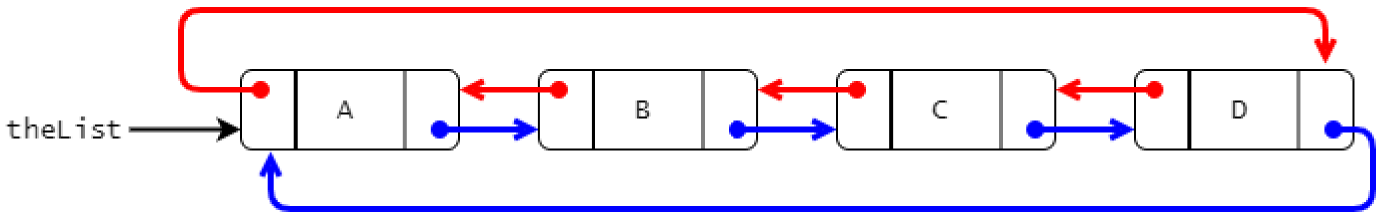


6. Use a doubly-linked circular node class to implement the list. Refer to the illustration below.
   Filename: `MyDoublyLinkedCircularList.java`

The doubly-linked circular node:



The linked list illustration using a doubly-linked circular node:



Test each data structure that you have created by using the class in an executable class(i.e. main class). Use the data structure to organize values and subject the data structures to appropriate manipulations/processing. You may have as many executable classes as needed.