19AIE111 Data Structure and Algorithms 1

S$_2$ B.Tech CSE(AI)

Project Report

Submitted by

(Nithin sylesh)

(AM.EN.U4AIE19044)

June 2020

Department of Computer science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

Department of Computer Science and Engineering

Certified that this is the bonafide project report for the course 19AIE111 Data Structure and Algorithms 1 submitted on its completion by Nithin Sylesh, a second semester student of B.Tech CSE(AI)

Abstract

In the Project Phase 1, the project focused mainly on linear data structures such as stack, linked list and queue. Stack is implemented using an array and a linked list. Both these implementations must support the same basic functions. A push and pop method must be implemented in order to push a chosen element and pop elements from the top of the stack respectively. A peek method is used to print the topmost element of the stack. The show method prints all the elements in the specified stack. In the input code, different integers are assigned to different methods which, when entered, executes that particular method. In the second part of the project similarly a queue is implemented using an array and linked list. The methods implemented are enqueue, dequeue, peek and show which are activated by specific integer inputs as mentioned above.

In the first part of the Advanced questions, a linked list is initialised which had two parts: a main list and vertical lists attached the nodes in the main list. Here the aforementioned linked list is flattened using the flatten method that is created. All keys in the list are unique and positive. The input is generated by means of user input. It consists of the first line which contains keys for nodes in the main list, separated by spaces. The subsequent lines contain the keys for nodes in the down lists, one after the other, separated by spaces along including the head nodes even though they are a part of the main list. The output will be a singly linked list with all the nodes to the right and downwards arranged in increasing order.

In the Project Phase 2, we focus mainly on the hierarchical data structures in detail such as Binary Search Trees. We create a phone book which is ordered using the property of Binary Search Trees and the different methods: Search, Display_First and Display_Last are implemented using the aforementioned properties.

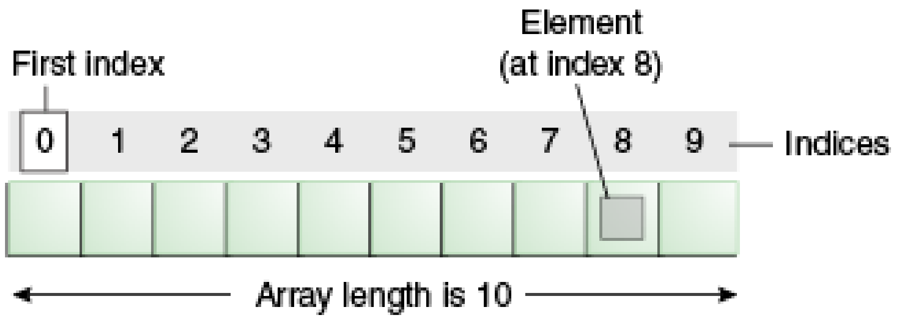
Contents page no

1.Introduction

The modern technological world revolves around Data Management. All of the services we use need to store raw data in some form and process it. Data Structures is a framework present in Java that provides an architecture that helps in storing and reading this data efficiently and in an organized manner. Data can be structured and stored in many ways but the methods that are being utilized and highlighted in this project include Array, Stack, Queue and Linked List

- ARRAY

  Arrays are basically blocks of memory. It is a container object that holds a fixed number of values of a single type stored contiguously. The length of an array is established when the array is created. After creation, its length is fixed.



Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the preceding illustration, numbering begins with 0. It can hold **primitive types** as well as **object references**. In fact most of the collection types in Java which are the part of **java.util** package use arrays internally in their functioning. Since Arrays are objects, they are created during runtime

Features of Array

1. Arrays are objects
2. They can even hold the reference variables of other objects

3. They are created during runtime
4. They are dynamic, created on the heap
5. The Array length is fixed

The declaration of array states the type of the element that the array holds followed by the identifier and square braces which indicates the identifier is array type. When processing array elements, we often use a loop because all of the elements in an array are of the same type and the size of the array is known.

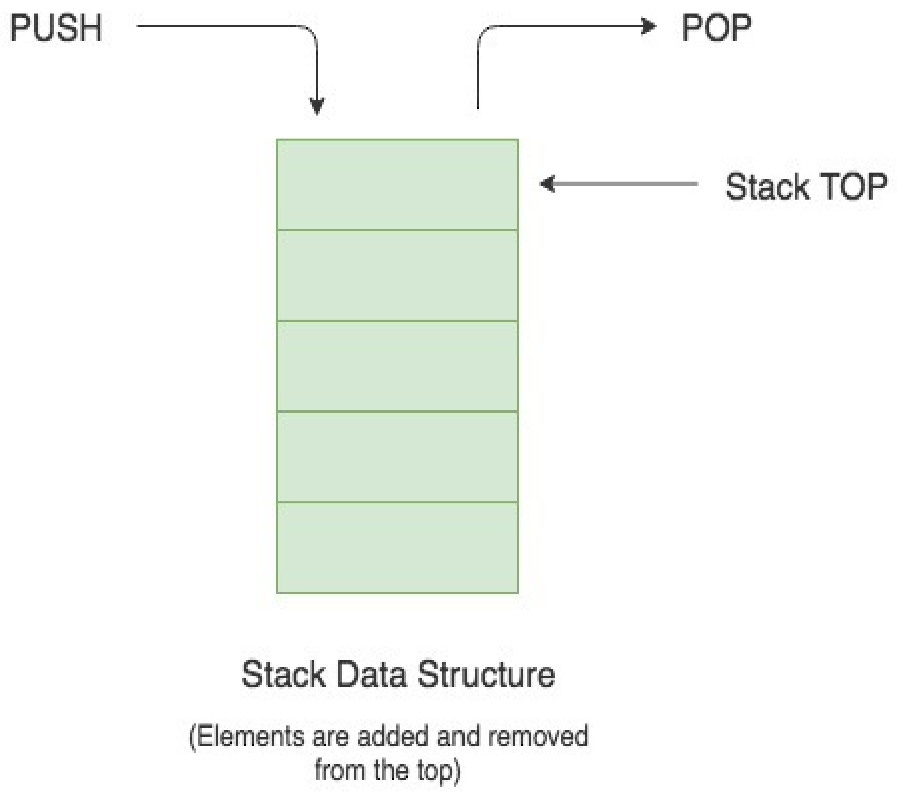
Following are the basic operations supported by an array.

- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

An array is used in some cases because the elements stored in an array can be accessed easily when compared to other data structures. But you can only insert/delete from the end of the array. You cannot increase the size of the arrays in Java; if you want to add new elements you need to create new array with extended size and assign to the array reference.

- STACK

A Stack is a Last In First Out (LIFO) data structure. A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. It supports two basic operations called **push** and **pop**. The push operation adds an element at the top of the stack, and the pop operation removes an element from the top of the stack. Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.



Two primary operations on a stack are:

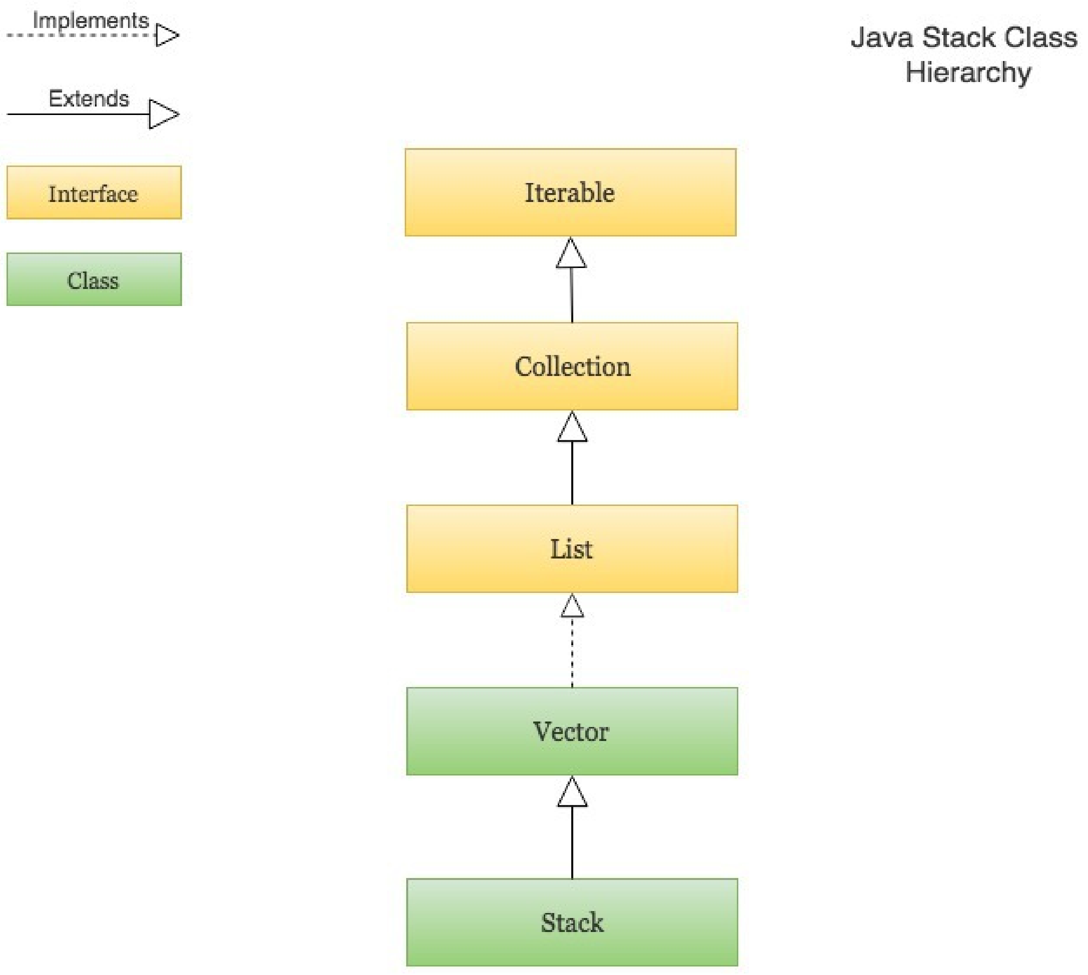
a. push() – Adding an element to the top of the stack.
b. pop()– Removing an element from the top of the stack.

Other basic functionalities that are implemented:

- peek() − get the top data element of the stack, without removing it.
- isFull() − check if stack is full.
- isEmpty() − check if stack is empty.

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size.

Java provides a Stack class which models the Stack data structure. The Stack class is part of Java's collections framework. Following is the class hierarchy of Stack in Java -



The Stack class extends Vector which implements the List interface. A Vector is a re-sizable collection. It grows its size to accommodate new elements and shrinks the size when the elements are removed.

Since the Stack class extends Vector, it also grows and shrinks its size as needed when new elements are added or removed.

The simplest application of a stack is to reverse a string. You push a given string to stack - letter by letter - and then pop letters from the stack.

There are other uses also like:

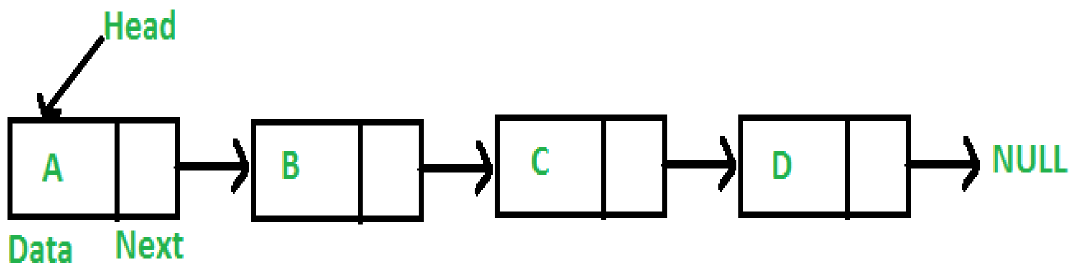
1. Parsing-analyses a string of symbols
2. Expression Conversion(Infix to Postfix, Postfix to Prefix etc)- in order to apply precedence rules to avoid the ambiguous evaluation

- Create an empty stack and an empty postfix output string/stream
- Scan the infix input string/stream left to right
- If the current input token is an operand, simply append it to the output string (note the examples above that the operands remain in the same order)
- If the current input token is an operator, pop off all operators that have equal or higher precedence and append them to the output string; push the operator onto the stack. The order of popping is the order in the output.
- If the current input token is '(', push it onto the stack
- If the current input token is ')', pop off all operators and append them to the output string until a '(' is popped; discard the '('.

3. Balancing Parenthesis
4. Stack is used to keep information about the active functions or subroutines.

- LINKED LIST

A linked list is a linear collection of data elements, whose order is not given by their physical placement in memory, Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a *link*) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing more efficient insertion or removal of nodes at arbitrary positions.



A linked list is a linear collection of data elements, whose order is not given by their physical placement in memory, Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a *link*) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing more efficient insertion or removal of nodes at arbitrary positions.

Advantages over Arrays

1)Dynamic size
2) Ease of insertion/deletion

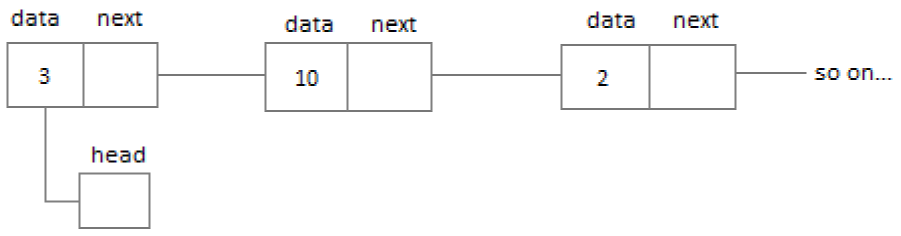
Drawbacks:
1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.
2) Extra memory space for a pointer is required with each element of the list.
3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Linked lists are often used because of their efficient insertion and deletion. They can be used to implement stacks, queues, and other abstract data types.

Singly Linked List

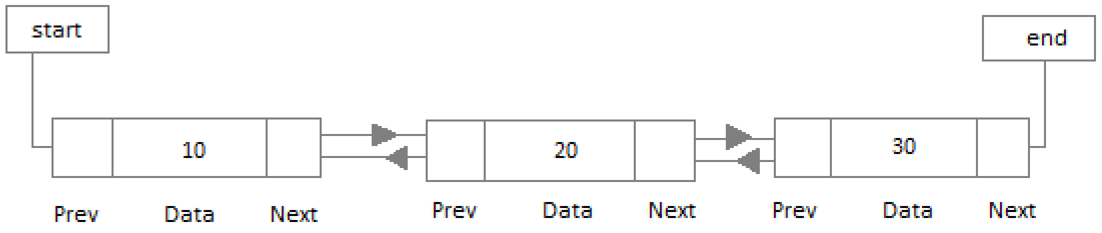
Singly linked lists contain nodes which have a **data** part as well as an **address part** i.e. next, which points to the next node in the sequence of nodes.

The operations we can perform on singly linked lists are **insertion**, **deletion** and **traversal**.



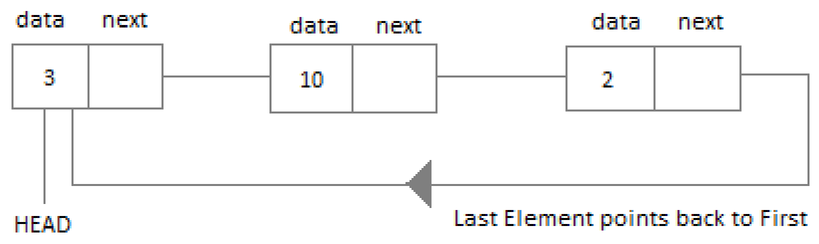
Doubly Linked List

In a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.



Circular Linked List

In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.

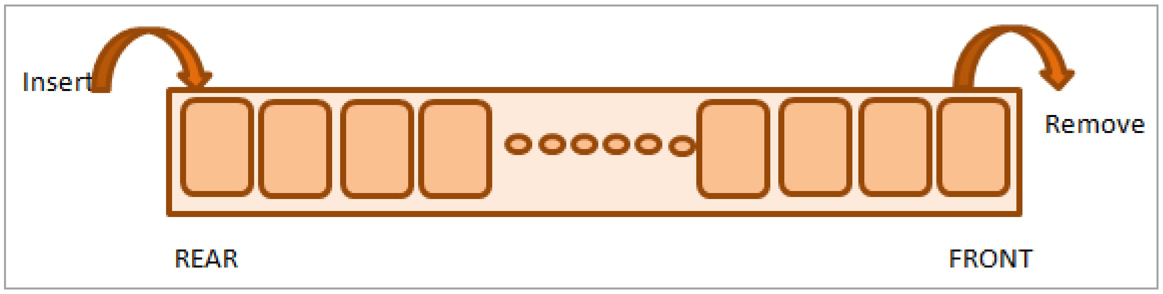


- <u>QUEUE</u>

A queue is a linear data structure or a collection in Java that stores elements in a FIFO (First In, First Out) order. Queue can be implemented using an Array, Stack or Linked List. The easiest way of implementing a queue is by using an Array.

The queue collection has two ends i.e. front & rear. The elements are added at the rear and removed from the front.

As shown in the below diagram, a queue is a structure having two points i.e. start (front) and end (rear). Elements are inserted into the queue at the rear end and removed from the queue at the front.



The methods in a queue are:

1. add: Adds element e to the queue at the end (tail) of the queue without violating the restrictions on the capacity. Returns true if success or IllegalStateException if the capacity is exhausted.
2. peek :Returns the front of the queue without removing it.
3. element: Performs the same operation as the peek () method. Throws NoSuchElementException when the queue is empty.
4. remove: Removes the head of the queue and returns it. Throws NoSuchElementException if queue is empty.

5. poll: Removes the head of the queue and returns it. If the queue is empty, it returns null.

Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

In the project phase two, we use hierarchal data structures like Trees.

- BINARY SEARCH TREE

A binary tree is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.

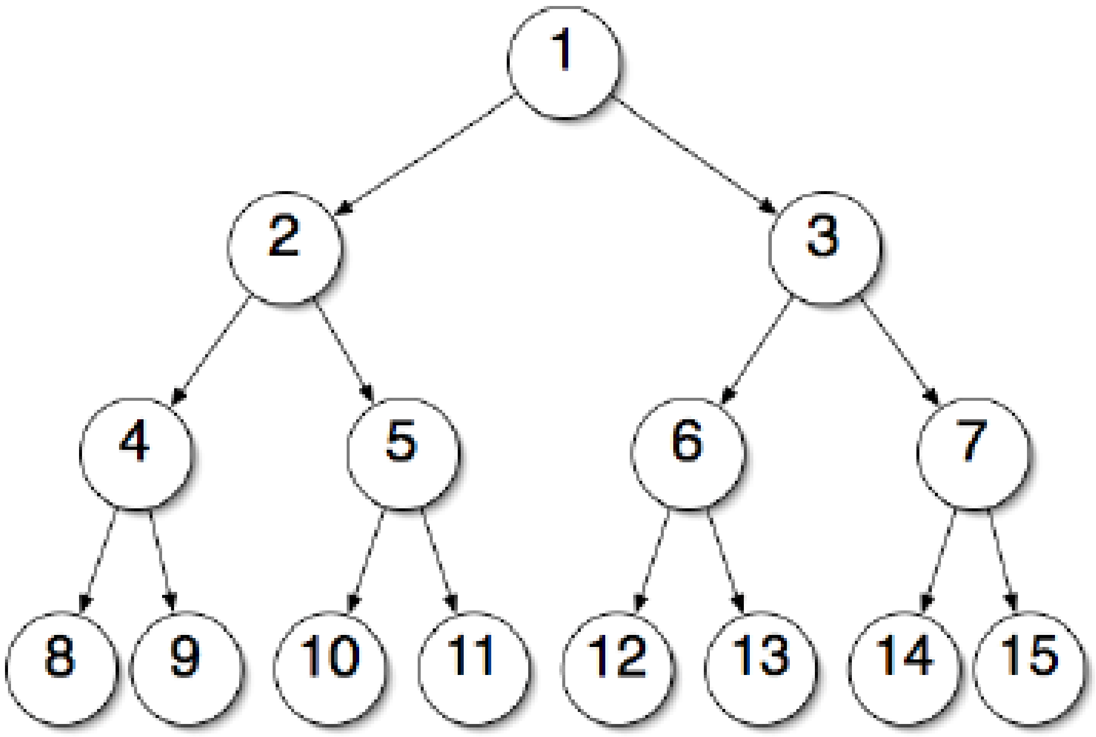
Each node contains three components:

1. Pointer to left subtree
2. Pointer to right subtree
3. Data element

The topmost node in the tree is called the root. An empty tree is represented by **NULL** pointer.

For a binary tree to be a binary search tree, the data of all the nodes in the left sub-tree of the root node should be less than the data of the root. The data of all the nodes in the right subtree of the root node should be greater than equal to the data of the root. As a result, the leaves on the farthest left of the tree have the lowest values, whereas the leaves on the right of the tree have the greatest values.

A representation of binary search tree is shown:



Basic operations on a BST

- Create: creates an empty tree.
- Insert: insert a node in the tree.
- Search: Searches for a node in the tree.
- Delete: deletes a node from the tree.
- Inorder: in-order traversal of the tree.
- Preorder: pre-order traversal of the tree.

- Postorder: post-order traversal of the tree.

2. Software and Hardware Requirements:

The OS used for this project is Windows 10. I prefer Windows 10 because it is compatible with all programming environments. It has better scheduling of tasks and processes. Windows 10 offers much more control over the tasks and processes without hassles. It also catalogues much better and is user-friendly, especially for an amateur programmer.

The IDE that I have used for this project is Eclipse. Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications.

Advantages of IDE:

- Using IDE will cost you less time and effort .
- Navigation is made easier.
- Auto completion- one of the best features , you don’t have to remember all.
- Refactoring
- Error debugging is easy , you can easily navigate to Error line.
- All files can be viewed and managed at same screen.
- Organizing you imports.
- Downloading requires packages at ease.
- It is free and open source.
- Industrial level of development
- It supports many other languages other than JAVA.
- Since Eclipse tools are open source, they are quickly updated with the latest technology that can be integrated into existing code.

4. Problem Statement

<u>Project Phase 1 – LISTS, STACKS, QUEUES</u>

<u>Question 1 : 10 Marks</u>

STACK

a. Implement a stack using an array.

b. Implement a stack using a linked list.

Your program must support the following functions:

• push(element)

• pop()

• peek()

• show(stk)

<u>Question 2 : 10 Marks</u>

QUEUE

a. Implement a queue using an array.

b. Implement a queue using a linked list.

Your program must support the following functions:

• enqueue(element)

• dequeue(q)

• peek(q)

• show(q)

<u>Question 3 : 7 Marks</u>

Flattening a Linked List

Write a function flatten() to flatten the lists into a singly linked list. The flattened linked list should also be sorted.

<u>Question 4 : 8 Marks</u>

Flatten a multilevel linked list

Given a linked list where in addition to the next pointer, each node has a child pointer, which may or may not point to a separate list. You are given the head of the first level of the list. Flatten the list so that all the nodes appear in a single-level linked list. You need to flatten the list in way that all nodes at first level should come first, then nodes of second level, and so on.

Project Phase 2 – Phone Directory using BST ADT

Implement a Phone Directory

Develop a phone directory which load and store contact information which are name, phone number and email from an input text.file. Your java program should function like a phone book, loading information from an input file into a Binary search tree(call insert() method of BST) and performing the following operations on said information.

Search(String name): Should return all the contacts with the given name. If no contacts exists then give an appropriate message.

Display(): Display the contents of the entire Directory.

Display_first(): Display the Contact detail that comes first in the ordering of your phone Book.

Display_last(): Display the Contact detail that comes last in the ordering of your phone Book.

The above functions can be tested from Command line. The initial input to the program should be taken from the file. (input format is : Name phone number email). At least 30 entries should be there in your file.

4.Modules

PROJECT PHASE I

STACK

Array Implementation of Stack

A stack can be implemented using a one-dimensional array which stores only a fixed number of data values. It is implemented according to the principle LIFO (Last In First Out) i.e. the element that gets pushed last onto the stack get popped first. Push, pop, peek and show are the functions implemented in this program. Generate an array of fixed length whose size should be greater than zero and less than 100 as specified in the problem statement. Then a top pointer needs to be initialised. This pointer will point to the top element of the stack. In the implementation using an array, this top pointer will be a non-negative integer which will always be equal to the index of the topmost element of the stack and if the stack is not initialized then the top pointer is made to be equal to -1.

The push method is used to insert an element at the top of the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as parameter and inserts that value into the stack. Initially we check whether the stack is full. If the top is pointing to the last index of the array(size-1), that means that the stack is full. In this case, if an element is made to enter the stack, it would activate overflow condition and the function will be terminated. If the stack is not full, then the top pointer gets incremented and the new element gets pushed to the incremented position which becomes the new top of the stack.

The pop method is used to remove an element from the top of the stack. If the value of top is -1, then the stack is empty. If we try to use the pop function on an empty stack it will invoke underflow condition which would, in turn, terminate the operation. Otherwise, we return the topmost element and then decrease the value of the ‘top’ by 1 which will detach the last element from the stack.

The peek method displays the topmost element in the stack without removing it. If the value of top is -1, then there are no elements to display and it will print a message to indicate that the stack is empty. If the top pointer is not at -1, it prints the element the top pointer is currently pointing to.

The show method prints all the elements of the specified stack. It takes a stack that is specified by the user as an input parameter. As mentioned above, when the stack is empty, i.e. top=-1; it will show an appropriate empty message. Otherwise define a variable 'i' and initialize with top. Then run a for loop to print the element at the i^{th} position and decrement i after each iteration until i reaches 0.

<u>Linked List Implementation of Stack</u>

A stack data structure can be implemented by using a singly linked list data structure. The stack implemented using linked list can work for an unlimited number of values. Therefore, size need not be specified in this implementation. In linked list implementation of stack, the nodes are maintained non-contiguously in the memory. All of the basic operations should be carried out by a Node variable top whose position in the linked list indicates the top of the stack. For that, a Node class must be created, which contains a data part that stores a particular value stored in a node and an address part that points to the successor node in the stack. When a stack object is created, top gets initialised to null. The functions- push, pop, peek and show are implemented here.

When the push method is called, the provided data is stored on a Node variable and a specific memory is allocated. Then we make this new node point to the address of the previous node at the top of the list. Then we assign top to the new node thus making it the top of the list.

The pop method is used to pop the topmost element of the stack. Here I have implemented it as an integer method as it returns a data value which is to be removed. If the top is equal to null, it indicates that the stack is empty and -1 is returned as an integer has to be returned and a message indicating the empty stack is printed as well. If not, update the top pointer to point to the next node and then return the element pointed to previously by the top, thereby removing it from the stack.

The peek method prints the element the top pointer is pointing to in the stack. To check whether a stack is empty, the peek method checks if the top is null and displays a message to indicate an empty stack.

In the show method, initially we check if whether the stack is empty by checking if the top is equal to null. The top pointer is saved to a variable that is referred to as temp in the code. This temp variable is used to traverse through the stack and print all the elements using a while loop until temp becomes equal to null.

QUEUE

Array Implementation of Queue

Queue implements the FIFO mechanism i.e. the element that is inserted first is also deleted first. In other words, the least recently added element is removed first in a queue. To define a queue using an array, we need to define three variables: a variable named front which is used to store the position of the first element, a variable named rear which stores the position of the last element and an array to store elements of the queue and whose size is predetermined. In the constructor the front and rear variables of the queue are set to -1 which denotes that the queue is empty and the size of the array is initialised. Common operations or function calls associated with Queue which include enqueue, dequeue, peek and show are implemented in the code.

In the enqueue method, we take the element that is to be entered as an input parameter. We check whether the queue is full by checking if the value of rear is one less than length of the array which is denoted by a message. If the queue is not full, we increment the values of front and rear such that the value of front and rear is zero and then the element is inserted at the position of the rear pointer. Otherwise, we insert the element onto the position denoted by the incremented rear pointer on the queue.

The dequeue method removes elements from the front of the queue. A queue is taken as an input parameter. If the front pointer is equal to the rear pointer, then both the pointers are assigned a value of -1. Otherwise, the front position is incremented in order to remove the element from the queue.

The peek method displays the element at the front of the queue. It takes a queue as the input parameter. If the front and rear pointers are at 1, a message is displayed to indicate an empty queue. Otherwise the value at the front position of the queue is returned.

The show method displays all the elements in the queue if the queue is empty. If the queue is not empty, then the front position is assigned to a variable i. This variable is incremented until it reaches the end of the queue and prints the elements at the ith position in the queue stored in the array.

Linked List Implementation of Queue

Linked list implementation of queue is more efficient for large scale operations that queue tackles. In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory. In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue In the constructor, front and rear both are null, which is the empty condition.

The methods implemented are enqueue, dequeue, peek, and show.

The enqueue method creates a node with the value given as the only input parameter. If the queue is empty, then the new node is assigned to the location of the front and rear node pointers. If the queue is not empty, then the rear pointer's next points to the new node and the rear is now this new node.

The dequeue method first checks if the queue is empty, and returns if it is, then it assigns the front pointer to a variable. Then the front pointer is repositioned to the node after the initial front node, and the value of the initial front node is returned.

The peek method displays a message if the queue is empty, otherwise it returns the value of the front node of the queue.

The show method is used to print all the elements in a queue. It checks if the queue is empty or not, and if it is not, then the front node is assigned to a temporary variable called temp as specified in the code. Then the queue is traversed through until the end of the queue as the values in the queue are printed.

PROJECT PHASE II

A phone directory is developed by using 4 classes: Phonebook, Person,filereading,binarysearch tree class. Phonebook class is used to manage the information of multiple individuals. This class should contain a binary search tree as a data field. This tree contains the people in the book. Person manages the details of a single person. The Driver class runs all the methods.filereading **class** is used to **read** data from the **file**. It returns data in byte format like FileInputStream **class**. It is character-oriented **class**

In the Phonebook class, only one class variable is created which is Person root. In the Person class, six variables are created: three are of the string type which include *name*, *phonenumber* and *email* a Person object *left* and.

In the insert method, the input file that contains the details of a group of 8 people is generated and is read in the Phonebook class using the File

Class and is stored into a string array. Then we read the file and split the string so that the name, phonenumber and email gets stored into separate arrays which will represent the individual parameters of a person. We allocate the first name on the list as the root and the subsequent entries as the children of that root. In the insert method in Person, compare each name entry to the name variable present and arrange it lexicographically to either the left or the right of the root according to the ASCII number irrespective of case using the compareToIgnoreCase method.

We can use the search method to search for a particular name and print out the entries that matches that name. It can also print out the details of people having similar first names. The search method in Phonebook recursively calls the search method in Person until all the names are found.

5. Implementation :

PROJECT PHASE I

Stack using array

```
import java.util.*;

import java.lang.String;


public class Projectphase1B {

    public static void main(String[] main) {

        Stack_array st1 = new Stack_array();

        Scanner in = new Scanner(System.in);
        int choice = -1;
        int x = 0;
        while (choice != 0) {
            System.out.println("Enter your choice:");
            String st[] = in.nextLine().split(" ");
            choice = Integer.parseInt(st[0]);
            if (st.length > 1) {
                x = Integer.parseInt(st[1]);
            }
            switch (choice) {
                case 0:
                    break;
                case 1:
                    if (x < 0) {
                        System.out.println("Integer must be positive");
                        break;
                    }
                    if (st.length > 1) {
                        st1.push(x);
                    } else {
                        System.out.println("Please enter value to be pushed with choice. i.e
- \"1 2\"");
                    }
                    break;
                case 2:
                    st1.pop();
                    break;
                case 3:
                    st1.peek();
                    break;
                case 4:
                    st1.show();
                    break;
                default:
                    System.out.println("Error, please stick to instructions mentioned in
pdf.");
            }
        }
    }
}

class Node {
    int data;
    Node next;

    Node(int item) {
        data = item;
        next = null;
    }
}

class Stack_array {
    static int MAX;
    int top;
    int[] ar;

    Stack_array() {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter size of stack.");
        top = -1;
        boolean flag = false;
        while (!flag) {
            MAX = in.nextInt();
            if (MAX > 0 && MAX < 100) {
                ar = new int[MAX];
                flag = true;
            } else {
                System.out.println("Array size has to be in between 0 and 100. Enter
value again.");
```

```
        }
    }
}

void push(int item) {
    if (top == MAX - 1) {
        System.out.println("OVERFLOW");
        return;
    }
    top++;
    ar[top] = item;
}

void pop() {
    if (top < 0) {
        System.out.println("EMPTY");
        return;
    }
    System.out.println(ar[top]);
    top--;
}

void peek() {
        if (top < 0) {
            System.out.println("EMPTY");
            return;
        }
        System.out.println(ar[top]);
    }

    void show() {
        if (top < 0) {
            System.out.println("EMPTY");
            return;
        }
        for (int i = top; i >= 0; i--) {
            System.out.print(ar[i] + " ");
        }
        System.out.println();
    }
}
```

<u>Stack using Linked List</u>

```
import java.util.*;
import java.lang.String;
```

```
public class projectphase1A {
    public static void main(String[] main) {
        Stack_linkedlist st1 = new Stack_linkedlist();
        Scanner in = new Scanner(System.in);
        int choice = -1;
        int x = 0;
        while (choice != 0) {
            System.out.println("Enter your choice:");
            String[] st = in.nextLine().split(" ");
            choice = Integer.parseInt(st[0]);
            if (st.length > 1) {
                x = Integer.parseInt(st[1]);
            }

            switch (choice) {
                case 0:
                    break;
                case 1:
                    if (x < 0) {
                        System.out.println("Integer must be positive");
                        break;
                    }
                    if (st.length > 1) {
                        st1.push(x);
                    } else {
                        System.out.println("Please enter value to be pushed with choice. i.e
- \"1 2\"");
                    }
                    break;
                case 2:
                    st1.pop();
                    break;
                case 3:
                    st1.peek();
                    break;
                case 4:
                    st1.show();
                    break;
                default:
                    System.out.println("Error, please stick to instructions mentioned in
pdf.");
            }
        }
    }
}
```

```
class Node {
    int data;
    Node next;

    Node(int item) {
        data = item;
        next = null;
    }
}

class Stack_linkedlist {
    Node top = null;

    void push(int item) {
        Node next_node = new Node(item);
        if (isEmpty()) {
            top = next_node;
            return;
        }
        next_node.next = top;
        top = next_node;
    }
```

```
void pop() {
    if (isEmpty()) {
        System.out.println("EMPTY");
        return;
    }
    System.out.println(top.data);
    top = top.next;
}

void show() {
    if (isEmpty()) {
        System.out.println("EMPTY");
        return;
    }
    Node temp = top;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}
```

```
    int peek() {
        if (isEmpty()) {
            System.out.println("EMPTY");
            return -1;
        }
        System.out.println(top.data);
        return top.data;
    }

    boolean isEmpty() {
        return (top == null);
    }
}
```

Queue using array

```
import java.util.*;

public class projectphase2B {
    public static void main(String[] main) {
        Queue_array q2 = new Queue_array();
        Scanner in = new Scanner(System.in);
        int choice = -1;
        int x = 0;

        while (choice != 0) {

            System.out.println("Enter your choice:");

            String st[] = in.nextLine().split(" ");

            choice = Integer.parseInt(st[0]);

            if (st.length > 1) {

                x = Integer.parseInt(st[1]);

            }


            switch (choice) {

                case 0:

                    break;

                case 1:

                    if (st.length > 1) {

                        q2.enqueue(x);

                    } else {

                        System.out.println("Please enter value to be pushed with choice. i.e
- \"1 2\"");

                    }

                    break;

                case 2:

                    q2.dequeue();

                    break;
                case 3:
                    q2.peek();
                    break;
                case 4:
                    q2.show();
                    break;
                default:
                    System.out.println("Error, please stick to instructions mentioned in
pdf.");
            }
        }
    }
}

class Node {
    int data;
    Node next;

    Node(int item) {
        data = item;
        next = null;
    }
}
```

```
class Queue_array {
    int[] ar;
    int front;
    int rear;
    int current_size;
    static int MAX;

    Queue_array() {
        front = -1;
        rear = -1;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter size of queue.");
        boolean flag = false;
        while (!flag) {
            MAX = in.nextInt();
            if (MAX > 0 && MAX < 100) {
                ar = new int[MAX];
                flag = true;
            } else {
                System.out.println("Array size has to be in between 0 and 100. Enter
value again.");
            }
        }
        current_size = 0;
    }

    boolean isEmpty() {
        return (front == -1 && rear == -1);
    }

    void show() {
        if (front == -1) {
            System.out.println("EMPTY");
            return;
        }
        for (int i = front; i != rear + 1; i = (i + 1) % MAX) {
            System.out.print(ar[i] + " ");
        }
        System.out.println();
    }

    void enqueue(int item) {
        if (current_size == MAX) {
            System.out.println("OVERFLOW");
        } else if (isEmpty()) {
            front++;

            rear++;

            ar[rear] = item;

            current_size++;

        } else {

            rear = (rear + 1) % MAX;

            ar[rear] = item;

            current_size++;

        }

    }


    int peek() {

        if (front == -1) {

            System.out.println("EMPTY");

            return -1;

        } else {

            System.out.println(ar[front]);

            return ar[front];

        }

    }


    int dequeue() {

        int temp;
```

```
        if (isEmpty()) {
            System.out.println("EMPTY");
            return -1;
        } else if (front == rear) {
            temp = ar[front];
            front = -1;
            rear = -1;
            current_size--;
            System.out.println(temp);
            return temp;
        } else {
            temp = ar[front];
            front = (front + 1) % MAX;
            current_size--;
            System.out.println(temp);
            return temp;
        }
    }
}
```

Queue using Linked List

```
import java.util.*;
```

```
public class projectphase2A {
    public static void main(String[] main) {
        Queue_linkedlist q1 = new Queue_linkedlist();
        Scanner in = new Scanner(System.in);
        int choice = -1;
        int x = 0;
        while (choice != 0) {
            System.out.println("Enter your choice:");
            String st[] = in.nextLine().split(" ");
            choice = Integer.parseInt(st[0]);
            if (st.length > 1) {
                x = Integer.parseInt(st[1]);
            }

            switch (choice) {
                case 0:
                    break;
                case 1:
                    if (st.length > 1) {
                        q1.enqueue(x);
                    } else {
                        System.out.println("Please enter value to be pushed with choice. i.e
- \"1 2\"");
```

```
                    }
                    break;
                case 2:
                    q1.dequeue();
                    break;
                case 3:
                    q1.peek();
                    break;
                case 4:
                    q1.show();
                    break;
                default:
                    System.out.println("Error, please stick to instructions mentioned in
pdf.");
            }
        }
    }
}

class Node {
    int data;
    Node next;
    Node(int item) {
        data = item;
        next = null;
    }
}

class Queue_linkedlist {
    Node front, rear;

    void enqueue(int item) {
        Node temp = new Node(item);
        if (this.rear == null) {
            this.front = this.rear = temp;
            return;
        }
        this.rear.next = temp;
        this.rear = temp;
    }

    Node dequeue() {
        if (isEmpty()) {
            System.out.println("EMPTY");
            return null;
        }
        Node temp = front;
        front = front.next;
        if (front == null) {
            rear = null;
        }
        System.out.println(temp.data);
        return temp;
    }

    Node peek() {
        if (isEmpty()) {
            System.out.println("EMPTY");
            return null;
        }
        System.out.println(front.data);
        return front;
    }

    void show() {
        if (isEmpty()) {
            System.out.println("EMPTY");
            return;
        }
        Node temp = front;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    boolean isEmpty() {
        return (front == null);
    }
}
```

<u>Phase1 advanced question</u>

Flattern a linkedlist

```
import java.util.Scanner;

class Flatten_OLL {
    static int cnt = 0;
    Node head;

    static class Node {
        int data;
    Node next, down;


    Node(int data) {
        this.data = data;
        next = null;
        down = null;
    }
}

Node flatten(Node first) {
    if (first == null || first.next == null)
        return first;


    first.next = flatten(first.next);
    first = join(first, first.next);


    return first;
}


Node push(Node temp, int data) {
    Node node = new Node(data);
    node.down = temp;
    temp = node;

    return temp;

}


Node join(Node a, Node b) {

    if (a == null) return b;

    if (b == null) return a;


    Node res;

    if (a.data < b.data) {

        res = a;

        res.down = join(a.down, b);

    } else {

        res = b;

        res.down = join(a, b.down);

    }


    res.next = null;

    return res;

}


void printList() {

    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.down;
    }
    System.out.println();
}

static Flatten_OLL makeList(String[] ar) {
    Flatten_OLL L = new Flatten_OLL();
    L.head = L.push(L.head, Integer.parseInt(ar[0]));
    Node temp = L.head;
    for (int i = 1; i < ar.length; i++) {
        temp.next = L.push(temp.next, Integer.parseInt(ar[i]));
        temp = temp.next;
    }
    return L;
}

static void addList(Flatten_OLL L, String[] ar) {
    Node temp = L.head;
    for (int j = 0; j < cnt; j++) {
        temp = temp.next;
    }
```

```
        for (int i = 1; i < ar.length; i++) {
            temp.down = L.push(temp.down, Integer.parseInt(ar[i]));
            temp = temp.down;
        }
        cnt = cnt + 1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Flatten_OLL L;
        System.out.println("Enter the main list (horizontal)");
        String[] ar1 = sc.nextLine().split(" ");
        int count = ar1.length;
        L = makeList(ar1);
        int i = 0;
        while (count != i) {
            System.out.println("Enter the list under " + ar1[i] + " (including " + ar1[i] +
")");
            String[] ar2 = sc.nextLine().split(" ");
            addList(L, ar2);
            i++;
        }
        L.head = L.flatten(L.head);
        L.printList();
    }
}
```

Flattern a mll

```
import java.util.Scanner;


class Flatten_MLL {
    static Node head;
    Scanner sc = new Scanner(System.in);


    public static void main(String[] args) {
        new Scanner(System.in);
        Flatten_MLL list = new Flatten_MLL();
        System.out.println("Enter the first list");
        head = list.createArray();
        head = list.recList(head);
        list.flatten(head);
        list.printList(head);
    }


    Node createArray() {
        System.out.println("Enter the numbers in the list as space separated digits");
        String[] s = sc.nextLine().split(" ");
    int[] a = new int[s.length];
    for (int i = 0; i < s.length; i++) {
        a[i] = Integer.parseInt(s[i]);
    }
    return createList(a, a.length);
}

Node createList(int[] arr, int n) {
    Node node = null;
    Node p = null;
    int i;
    for (i = 0; i < n; ++i) {
        if (node == null) {
            node = p = new Node(arr[i]);
        } else {
            p.next = new Node(arr[i]);
            p = p.next;
        }
        p.next = p.child = null;
    }
    return node;
}
```

```
void printList(Node node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
    System.out.println("");
}



void flatten(Node node) {
    if (node == null) {
        return;
    }

    Node tmp;
    Node tail = node;
    while (tail.next != null) {
        tail = tail.next;
    }

    Node cur = node;
    while (cur != tail) {
        if (cur.child != null) {
            tail.next = cur.child;
            tmp = cur.child;
            while (tmp.next != null) {
                tmp = tmp.next;
            }
            tail = tmp;
        }
        cur = cur.next;
    }
}

Node recList(Node node) {
    if (node == null)
        return null;

    Node last = node;

    Node next = node.next;

    System.out.println("Does " + node.data + " have a child? (y/n)");
    String ch = sc.nextLine();
    if (ch.equals("y")) {
        node.child = createArray();
```

```
            node.child = recList(node.child);
        }

        if (next != null)
            last.next = recList(next);

        return node;
    }

    static class Node {
        int data;
        Node next, child;

        Node(int d) {
            data = d;
            next = child = null;
        }
    }

}
```

Phase2

```
public class BinarySearchTree{
    static class Node {
```

```
    Person data;

    Node left;

    Node right;


    public Node(Person item) {

        data = item;

        left = right = null;

    }

}


Node root;


BinarySearchTree() {

    root = null;

}


void insert(Person p) {

    root = insertRec(root, p);

}


Node insertRec(Node root, Person p) {

    if (root == null) {

        root = new Node(p);
        return root;
    }
    int result = p.getName().compareTo(root.data.getName());
    if (result < 0)
        root.left = insertRec(root.left, p);
    else if (result > 0)
        root.right = insertRec(root.right, p);
    else {
        System.out.println("This name has already been entered.");
    }
    return root;
}

public Person search(String name) {
    Node s = searchRec(root, name);
    if(s!=null){
        System.out.println("Search completed successfully");
        return s.data;
    }
    else {
        System.out.println("Search failed");
        return null;
    }
```

```
}

public Node searchRec(Node root, String name) {
    if (root == null || root.data.getName().equals(name))
        return root;
    int result = name.compareTo(root.data.getName());
    if (result < 0)
        return searchRec(root.left, name);
    return searchRec(root.right, name);
}

void Display() {
    DisplayRec(root);
}

public void DisplayRec(Node root) {
    if (root != null) {
        DisplayRec(root.left);
        root.data.display();
        DisplayRec(root.right);
    }
}
```

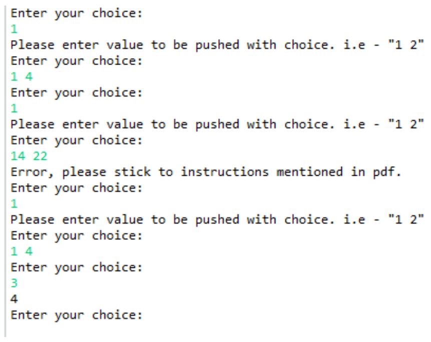
```
    void Display_first() {
        Person temp = root.data;
        while (root.left != null) {
            temp = root.left.data;
            root = root.left;
        }
        temp.display();

    }

    void Display_last() {
        Person temp = root.data;
        while (root.right != null) {
            temp = root.right.data;
            root = root.right;
        }
        temp.display();
    }

}
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;


public class FileReading{
    public ArrayList<Person> input(String path) {
        Person per;
        ArrayList<Person> p = new ArrayList<Person>();
        try {
            BufferedReader br = new BufferedReader(new FileReader(path));
            String currentLine = br.readLine();
            while (currentLine != null) {
                String[] arr = currentLine.split(" ");
                per = new Person(arr[0], arr[1], arr[2]);
                p.add(per);
                currentLine = br.readLine();
            }
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return p;
    }
}
public class Person {
    String name;
    String phonenumber;
    String email;

    public Person(String personname, String personphonenumber, String
personemail) {
        name = personname;
        phonenumber = personphonenumber;
        email = personemail;
    }

    public String getName() {
        return name;
    }

    public void display() {
        System.out.print(this.name + " ");
        System.out.print(this.phonenumber + " ");
        System.out.println(this.email);
    }
}
public class phoneBook {
    BinarySearchTree phonebook = new BinarySearchTree();
    public BinarySearchTree getphonebook() {
        return phonebook;
    }
}
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class Main {
    public static phoneBook book = new phoneBook();
    public static BinarySearchTree bst = book.getphonebook();
    public static FileReading fil = new FileReading();



    public static void main(String[] args) throws IOException {

        BufferedReader sc = new BufferedReader(new
InputStreamReader(System.in));

        while (true) {
            System.out.println("Please choose a option :");
            System.out.println("1.Input data using text file");

            System.out.println("2.Add a person");

            System.out.println("3.Find a Person");

            System.out.println("4.Display all contacts");

            System.out.println("5.Display first contact");

            System.out.println("6.Display last contact");

            System.out.println("7.Quit");


            int choice = Integer.parseInt(sc.readLine());


            switch (choice) {

                case 1 : {

                    System.out.println("Enter path of text file including filename");

                    String path = sc.readLine();

                    ArrayList<Person> per = fil.input(path);

                    for (Person p : per) {

                        bst.insert(p);

                    }

                }

                case 2 : {

                    System.out.println("Please enter person name, phone number and
email to add");

                    String name = sc.readLine();
                    String phone = sc.readLine();
                    String email = sc.readLine();


                    String personname;
                                                Person p = new Person(name, phone ,  email);
                    bst.insert(p);
                }
                case 3 : {
                    System.out.println("Please enter a person name to find");
                    String name = sc.readLine();
                    Person p = bst.search(name);
                    System.out.println("Details of " + name + " is : ");
                    if(p!=null)
                        p.display();
                }
                case 4 : {
                    bst.Display();
                }
                case 5 : {
                    bst.Display_first();
                }
                case 6 : {
                    bst.Display_last();
```

```
                }
                case 7 : {
                }
                default : System.out.println("Please enter a valid choice");
            }
            if (choice == 7)
                break;
        }
    }
}
```

6. Experimentation Result

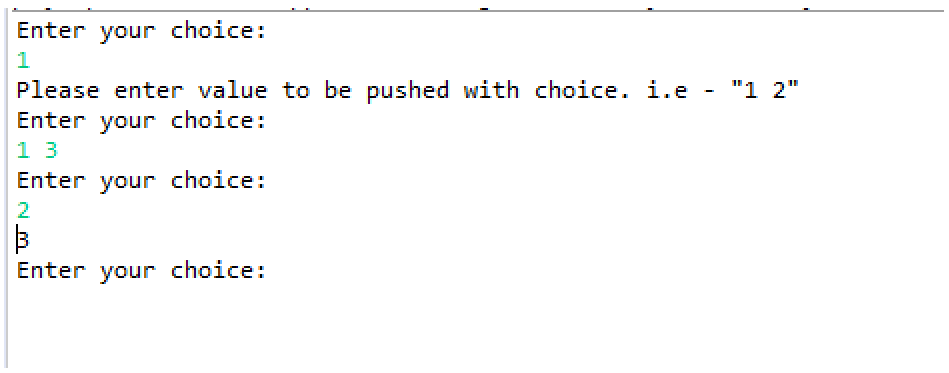
Phase 1

Stack using array



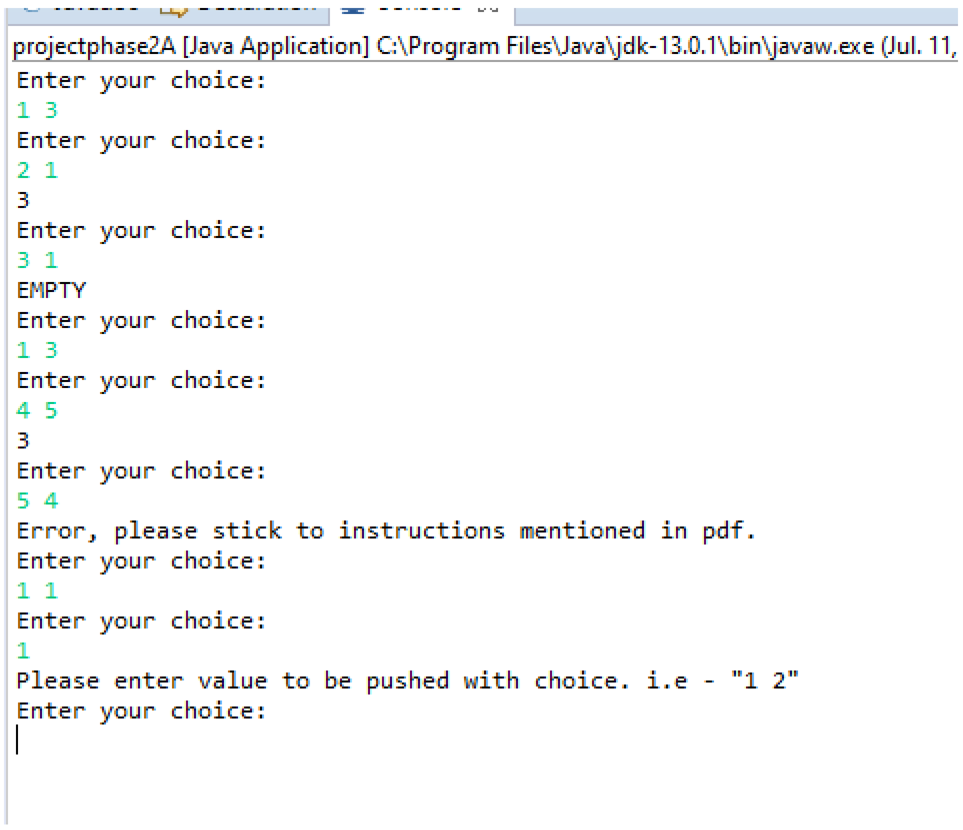
```
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
1 4
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
14 22
Error, please stick to instructions mentioned in pdf.
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
1 4
Enter your choice:
3
4
Enter your choice:
```

Stack using linkedlist



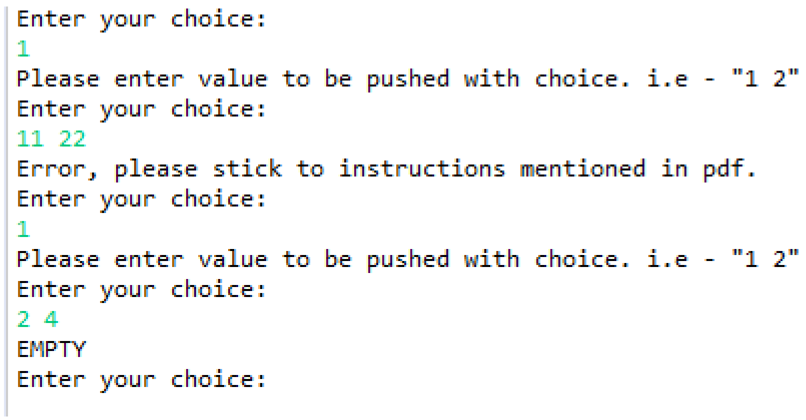
```
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
1 3
Enter your choice:
2
3
Enter your choice:
```

Queue using array



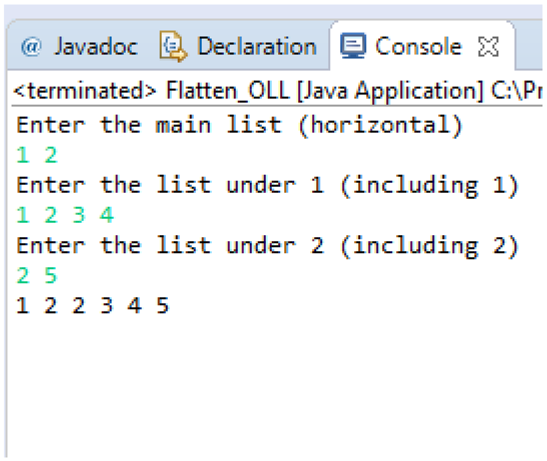
```
projectphase2A [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Jul. 11,
Enter your choice:
1 3
Enter your choice:
2 1
3
Enter your choice:
3 1
EMPTY
Enter your choice:
1 3
Enter your choice:
4 5
3
Enter your choice:
5 4
Error, please stick to instructions mentioned in pdf.
Enter your choice:
1 1
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
```

Queue using linked list



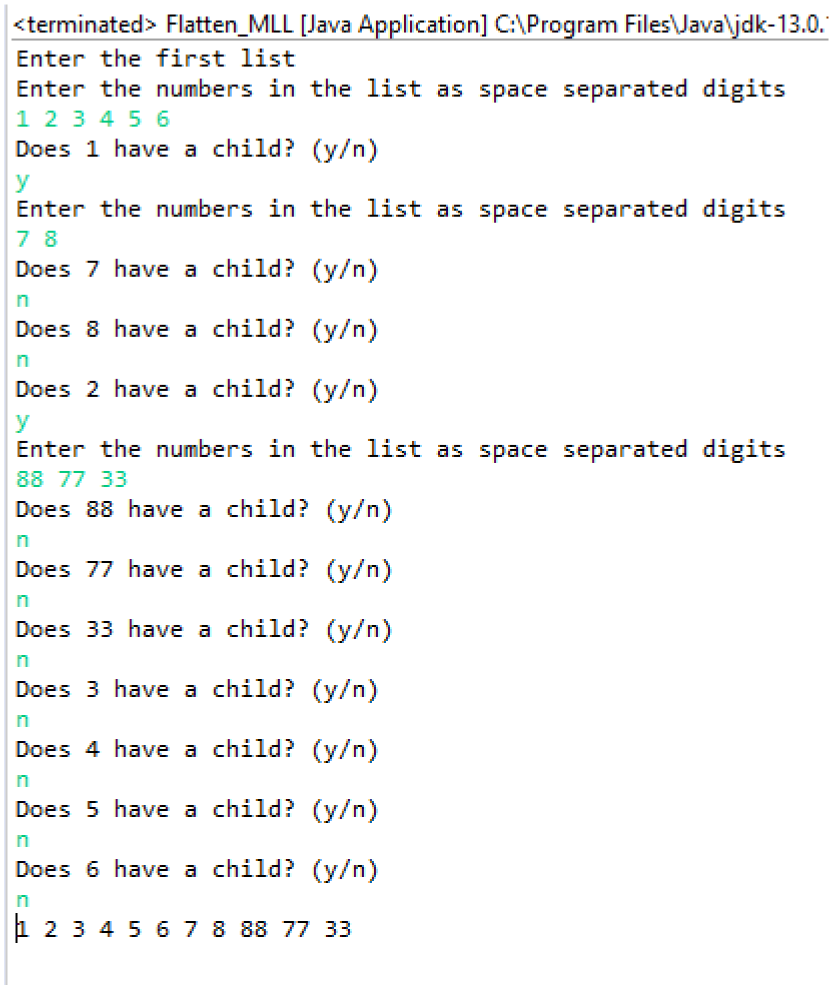
```
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
11 22
Error, please stick to instructions mentioned in pdf.
Enter your choice:
1
Please enter value to be pushed with choice. i.e - "1 2"
Enter your choice:
2 4
EMPTY
Enter your choice:
```

flattern linked list



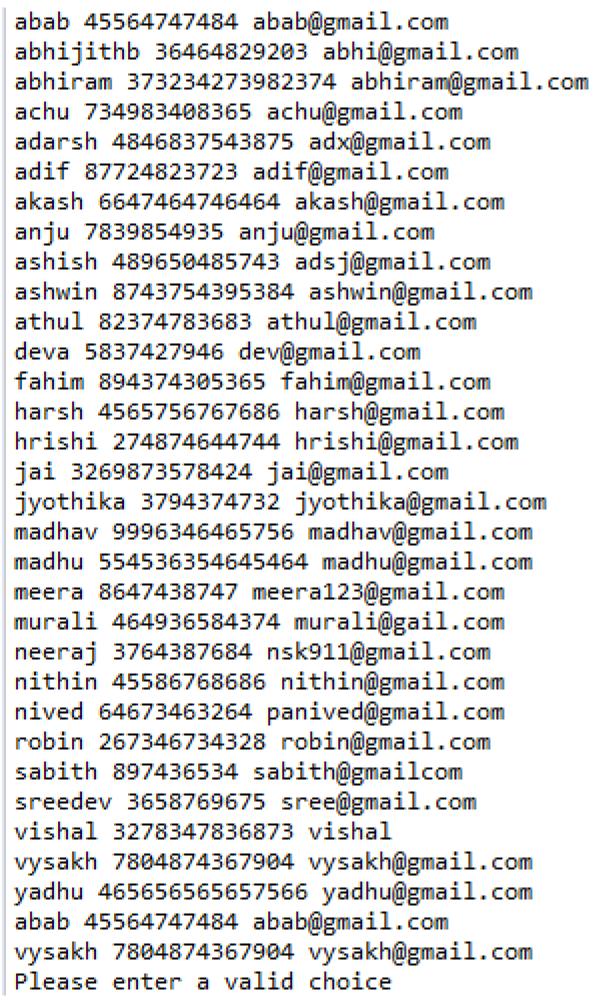
```
<terminated> Flatten_OLL [Java Application] C:\Pr
Enter the main list (horizontal)
1 2
Enter the list under 1 (including 1)
1 2 3 4
Enter the list under 2 (including 2)
2 5
1 2 2 3 4 5
```

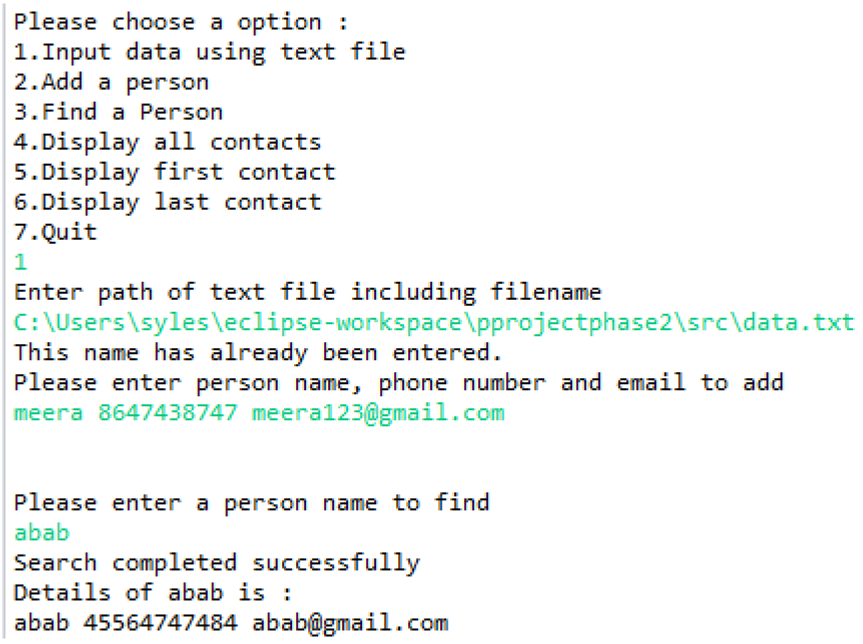
Flattern mll

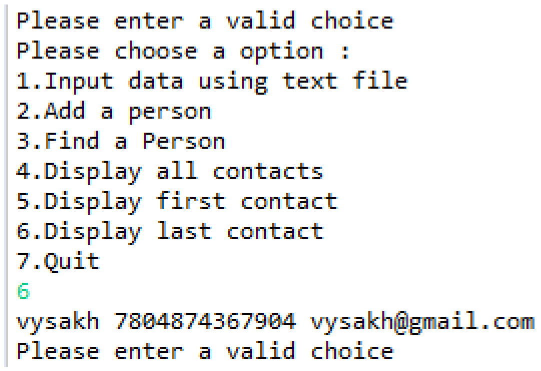


```
<terminated> Flatten_MLL [Java Application] C:\Program Files\Java\jdk-13.0.
Enter the first list
Enter the numbers in the list as space separated digits
1 2 3 4 5 6
Does 1 have a child? (y/n)
y
Enter the numbers in the list as space separated digits
7 8
Does 7 have a child? (y/n)
n
Does 8 have a child? (y/n)
n
Does 2 have a child? (y/n)
y
Enter the numbers in the list as space separated digits
88 77 33
Does 88 have a child? (y/n)
n
Does 77 have a child? (y/n)
n
Does 33 have a child? (y/n)
n
Does 3 have a child? (y/n)
n
Does 4 have a child? (y/n)
n
Does 5 have a child? (y/n)
n
Does 6 have a child? (y/n)
n
1 2 3 4 5 6 7 8 88 77 33
```

Phase 2


```
abab 45564747484 abab@gmail.com
abhijithb 36464829203 abhi@gmail.com
abhiram 373234273982374 abhiram@gmail.com
achu 734983408365 achu@gmail.com
adarsh 4846837543875 adx@gmail.com
adif 87724823723 adif@gmail.com
akash 6647464746464 akash@gmail.com
anju 7839854935 anju@gmail.com
ashish 489650485743 adsj@gmail.com
ashwin 8743754395384 ashwin@gmail.com
athul 82374783683 athul@gmail.com
deva 5837427946 dev@gmail.com
fahim 894374305365 fahim@gmail.com
harsh 4565756767686 harsh@gmail.com
hrishi 274874644744 hrishi@gmail.com
jai 3269873578424 jai@gmail.com
jyothika 3794374732 jyothika@gmail.com
madhav 9996346465756 madhav@gmail.com
madhu 554536354645464 madhu@gmail.com
meera 8647438747 meera123@gmail.com
murali 464936584374 murali@gail.com
neeraj 3764387684 nsk911@gmail.com
nithin 45586768686 nithin@gmail.com
nived 64673463264 panived@gmail.com
robin 267346734328 robin@gmail.com
sabith 897436534 sabith@gmailcom
sreedev 3658769675 sree@gmail.com
vishal 3278347836873 vishal
vysakh 7804874367904 vysakh@gmail.com
yadhu 465656565657566 yadhu@gmail.com
abab 45564747484 abab@gmail.com
vysakh 7804874367904 vysakh@gmail.com
Please enter a valid choice
```


```
Please choose a option :
1.Input data using text file
2.Add a person
3.Find a Person
4.Display all contacts
5.Display first contact
6.Display last contact
7.Quit
1
Enter path of text file including filename
C:\Users\syles\eclipse-workspace\pprojectphase2\src\data.txt
This name has already been entered.
Please enter person name, phone number and email to add
meera 8647438747 meera123@gmail.com


Please enter a person name to find
abab
Search completed successfully
Details of abab is :
abab 45564747484 abab@gmail.com
```



```
Please enter a valid choice
Please choose a option :
1.Input data using text file
2.Add a person
3.Find a Person
4.Display all contacts
5.Display first contact
6.Display last contact
7.Quit
6
vysakh 7804874367904 vysakh@gmail.com
Please enter a valid choice
```

7. Conclusion :

By working on this project I have a deeper understanding of the linear data structures which includes stacks, arrays and linked lists and the hierarchal data structures which include Binary Search Trees. I've also had an in-depth understanding of the implementation of these various data structures using other structures; for instance the implementation of stack and queue using linked list and arrays. The main questions in project phase one were already covered in our classes. But I struggled with the advanced questions and could only obtain a proper output for them.

In the phase two of the project, we had to implement a phonebook which gave me an insight into the real-life application of the data structures that we have studied about. By doing this project, an amateur programmer such as myself, was able to grasp the nuances of programming much more easier.

By doing this project I have understood many ways in which these data structures can be used and it helped me think outside the box and gain confidence in my programming knowledge

8. References:

www.geeksforgeeks.org

https://www.studytonight.com/data-structures/binary-search-tree

https://beginnersbook.com/2013/05/java-arrays/

https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html