//Project 2b

*//Data structures in Java programming language are developed using a combination of interfaces, classes, and built-in data types to create efficient and flexible data storage and manipulation solutions.*

*//A simple Linked List implementation in Java:*

```java

Class Node {

Int data;

Node next;

Public Node(int data) {

This.data = data;

This.next = null;

}

}

Class LinkedList {

Node head;

Public void add(int data) {

Node newNode = new Node(data);

If (head == null) {

Head = newNode;

} else {

Node current = head;

While (current.next != null) {

Current = current.next;

}

Current.next = newNode;

}

}

Public void display() {

Node current = head;

While (current != null) {

System.out.print(current.data + “ “);

Current = current.next;

}

System.out.println();

}

}

Public class Main {

Public static void main(String[] args) {

LinkedList list = new LinkedList();

List.add(10);

List.add(20);

List.add(30);

List.display();

}

}

```

//In this program, we have a `Node` class which represents a single node in the Linked List. Each node contains an integer `data` and a reference to the next node in the list.

//The `LinkedList` class maintains the head of the Linked List and provides methods to add elements to the list and display the contents of the list.

//In the `main` method, we create a new Linked List object, add some elements to it, and then display the contents of the list.

//2. Example of an array data structure in java programming language.

```java

Public class ArrayExample {

Public static void main(String[] args) {

// Declare and initialize an array of integers

Int[] numbers = {1, 2, 3, 4, 5};

// Accessing elements of the array

System.out.println(“Element at index 0: “ + numbers[0]);

System.out.println(“Element at index 2: “ + numbers[2]);

// Changing the value of an element

Numbers[3] = 10;

System.out.println(“Element at index 3 is now: “ + numbers[3]);

// Getting the length of the array

System.out.println(“Length of the array: “ + numbers.length);

// Iterating through the array

For (int I = 0; I < numbers.length; i++) {

System.out.println(“Element at index “ + I + “: “ + numbers[i]);

}

}

}

`` `

//3 Example a stack data structure in java programming language.

```java

Import java.util.Stack;

Public class StackExample {

Public static void main(String[] args) {

Stack<Integer> stack = new Stack<>();

// Push elements onto the stack

Stack.push(10);

Stack.push(20);

Stack.push(30);

// Pop element from the stack

Int poppedElement = stack.pop();

System.out.println(“Popped element: “ + poppedElement);

// Peek at the top element of the stack

Int topElement = stack.peek();

System.out.println(“Top element: “ + topElement);

// Check if the stack is empty

Boolean isEmpty = stack.isEmpty();

System.out.println(“Is stack empty? “ + isEmpty);

// Search for an element in the stack

Int index = stack.search(20);

System.out.println(“Index of element 20: “ + index);

// Iterate through the stack using for-each loop

For (Integer element : stack) {

System.out.println(“Element: “ + element);

}

}

}

```

*//4 Example of a queue data structure in java programming language*

```java

Import java.util.LinkedList;

Import java.util.Queue;

Public class QueueExample {

Public static void main(String[] args) {

Queue<String> queue = new LinkedList<>();

// Add elements to the queue

Queue.offer(“Alice”);

Queue.offer(“Bob”);

Queue.offer(“Charlie”);

// Remove and return the head of the queue

String removedElement = queue.poll();

System.out.println(“Removed element: “ + removedElement);

// Peek at the head of the queue without removing

String peekedElement = queue.peek();

System.out.println(“Peeked element: “ + peekedElement);

// Check if the queue is empty

Boolean isEmpty