DAY- 2

QUEUE

Queue – Tryout

```
class Queue {
	private int front;
// front represents the index position of the first element in the queue
	private int rear;
// rear represents the index position of the last element in the queue
	private int maxSize;
 // maxSize represent the maximum number of elements that can be stored in the queue
	private String arr[];
	Queue(int maxSize) {
		this.front = 0;
 // front is 0 when the queue is created
		this.rear = -1;
 // rear is -1 when the queue is created
		this.maxSize = maxSize;
		this.arr = new String[maxSize];
	}
// Checking if the queue is full or not
	public boolean isFull() {
		if (rear == maxSize - 1) {
			return true;
		}
		return false;
	}
	// Adding a new element to the rear of queue
	public boolean enqueue(String data) {
```

```
        if (isFull()) {
                return false;
        } else {
                arr[++rear] = data;
                return true;
        }
}
// Displaying all the elements in the queue
public void display() {
        if (isEmpty())
                System.out.println("Queue is empty!");
        else {
                System.out.println("Displaying queue elements");
                for (int index = front; index <= rear; index++) {
                        System.out.println(arr[index]);
                }
        }
}
// Checking if the queue is empty or not
public boolean isEmpty() {
        if (front > rear)
                return true;
        return false;
}
// Removing an element from the front of queue
public String dequeue() {
        if (isEmpty()) {
                return "empty";
        } else {
                String data = arr[this.front];
                arr[front++] = null;
```

```
                return data;
            }
        }
}
class Tester {
    public static void main(String[] args) {
        Queue queue = new Queue(5);
    System.out.println("Queue created.\n");
        if (queue.enqueue("Joe"))
            System.out.println("The element is enqueued to the queue!\n");
        else
            System.out.println("Queue is full!\n");
        if (queue.enqueue("Jack"))
            System.out.println("The element is enqueued to the queue!\n");
        else
            System.out.println("Queue is full!\n");
        if (queue.enqueue("Eva"))
            System.out.println("The element is enqueued to the queue!\n");
        else
            System.out.println("Queue is full!\n");
        if (queue.enqueue("Mia"))
            System.out.println("The element is enqueued to the queue!\n");
        else
            System.out.println("Queue is full!\n");
        if (queue.enqueue("Luke"))
            System.out.println("The element is enqueued to the queue!\n");
        else
            System.out.println("Queue is full!\n");
        queue.display();
        if (queue.enqueue("Emma"))
            System.out.println("The element is enqueued to the queue!\n");
```

```
else
        System.out.println("Queue is full!\n");
String dequeuedElement = queue.dequeue();
if (dequeuedElement == "empty")
        System.out.println("Queue is empty\n");
else
System.out.println("The element dequeued is : " + dequeuedElement + "\n");
dequeuedElement = queue.dequeue();
if (dequeuedElement == "empty")
        System.out.println("Queue is empty\n");
else
System.out.println("The element dequeued is : " + dequeuedElement + "\n");

dequeuedElement = queue.dequeue();
if (dequeuedElement == "empty")
        System.out.println("Queue is empty\n");
else
System.out.println("The element dequeued is : " + dequeuedElement + "\n");
dequeuedElement = queue.dequeue();
if (dequeuedElement == "empty")
        System.out.println("Queue is empty\n");
else
System.out.println("The element dequeued is : " + dequeuedElement + "\n");
dequeuedElement = queue.dequeue();
if (dequeuedElement == "empty")
        System.out.println("Queue is empty\n");
else
System.out.println("The element dequeued is : " + dequeuedElement + "\n");
dequeuedElement = queue.dequeue();
if (dequeuedElement == "empty")
        System.out.println("Queue is empty\n");
```

```
                else
                System.out.println("The element dequeued is : " + dequeuedElement + "\n");
        }
}
```

QUEUE EXERCISE

```
class Queue {
    private int front;
    private int rear;
    private int maxSize;
    private int arr[];
    Queue(int maxSize) {
        this.front = 0;
        this.rear = -1;
        this.maxSize = maxSize;
        this.arr = new int[this.maxSize];
    }
    public boolean isFull() {
        if (rear == maxSize - 1) {
            return true;
        }
        return false;
    }
    public boolean enqueue(int data) {
        if (isFull()) {
            return false;
        } else {
            arr[++rear] = data;
            return true;
        }
    }
    public void display() {
```

```
        if(isEmpty())
            System.out.println("Queue is empty!");
        else {
            for (int index = front; index <= rear; index++) {
                System.out.println(arr[index]);
            }
        }
    }
    public boolean isEmpty() {
        if (front > rear)
            return true;
        return false;
    }
    public int dequeue() {
        if (isEmpty()) {
            return Integer.MIN_VALUE;
        } else {
            int data = arr[this.front];
            arr[front++] = Integer.MIN_VALUE;
            return data;
        }
    }
    public int getMaxSize() {
        return maxSize;
    }
}
class Tester {
    public static void main(String[] args) {
        Queue queue = new Queue(7);
        queue.enqueue(2);
        queue.enqueue(7);
```

```
        queue.enqueue(9);
        queue.enqueue(4);
        queue.enqueue(6);
        queue.enqueue(5);
        queue.enqueue(10);
        Queue[] queueArray = splitQueue(queue);
        System.out.println("Elements in the queue of odd numbers");
        queueArray[0].display();
        System.out.println("\nElements in the queue of even numbers");
        queueArray[1].display();
    }
    public static Queue[] splitQueue(Queue queue) {
        int maxSize = queue.getMaxSize();
        Queue oddQueue = new Queue(maxSize);
        Queue evenQueue = new Queue(maxSize);
        while (!queue.isEmpty()) {
            int num = queue.dequeue();
            if (num % 2 == 0) {
                evenQueue.enqueue(num);
            } else {
                oddQueue.enqueue(num);
            }
        }
        return new Queue[]{oddQueue, evenQueue};
    }
}
```

STACK ASSIGNMENT

```
class Stack {
    private int top;
```

```
private int maxSize;
private int[] arr;
Stack(int maxSize) {
   this.top = -1;
   this.maxSize = maxSize;
   arr = new int[maxSize];
}
public boolean isFull() {
   return top >= (maxSize - 1);
}
public boolean push(int data) {
   if (isFull()) {
      return false;
   } else {
      arr[++top] = data;
      return true;
   }
}
public int peek() {
   if (isEmpty())
      return Integer.MIN_VALUE;
   else
      return arr[top];
}
public void display() {
   if (isEmpty())
      System.out.println("Stack is empty!");
   else {
      System.out.println("Displaying stack elements");
      for (int index = top; index >= 0; index--) {
         System.out.println(arr[index]); // accessing element at position index
```

```
            }
        }
    }
    public boolean isEmpty() {
        return top < 0;
    }
    public int pop() {
        if (isEmpty())
            return Integer.MIN_VALUE;
        else
            return arr[top--];
    }
}
class Tester {
    public static void main(String args[]) {
        Stack stack = new Stack(10);
        stack.push(15);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        calculateSum(stack);
        System.out.println("Updated stack");
        stack.display();
    }
    public static void calculateSum(Stack stack) {
        Stack tempStack = new Stack(stack.maxSize);
        int sum = 0;
        // Pop all elements from the original stack, calculate the sum, and push them onto a
temporary stack
        while (!stack.isEmpty()) {
            int value = stack.pop();
```

```
            sum += value;
            tempStack.push(value);
        }
        // Push the sum onto the original stack
        stack.push(sum);
        // Push all elements from the temporary stack back to the original stack
        while (!tempStack.isEmpty()) {
            stack.push(tempStack.pop());
        }
    }
}
```

QUEUE ASSIGNMENT

```
class Queue {
    private int front;
    private int rear;
    private int maxSize;
    private int[] arr;
    Queue(int maxSize) {
        this.front = 0;
        this.rear = -1;
        this.maxSize = maxSize;
        this.arr = new int[this.maxSize];
    }
    public boolean isFull() {
        return rear >= (maxSize - 1);
    }
    public boolean enqueue(int data) {
        if (isFull()) {
            return false;
        } else {
```

```
            arr[++rear] = data;
            return true;
        }
    }

    public void display() {
        if (isEmpty())
            System.out.println("Queue is empty!");
        else {
            for (int index = front; index <= rear; index++) {
                System.out.println(arr[index]);
            }
        }
    }
    public boolean isEmpty() {
        return front > rear;
    }
    public int dequeue() {
        if (isEmpty())
            return Integer.MIN_VALUE;
        else {
            int data = arr[this.front];
            arr[front++] = Integer.MIN_VALUE;
            return data;
        }
    }
    public int getMaxSize() {
        return maxSize;
    }
}
class Tester {
```

```
    public static void main(String[] args) {
        Queue queue = new Queue(7);
        queue.enqueue(13983);
        queue.enqueue(10080);
        queue.enqueue(7113);
        queue.enqueue(2520);
        queue.enqueue(2500);
        Queue outputQueue = findEvenlyDivisibleNumbers(queue);
        System.out.println("Evenly divisible numbers");
        outputQueue.display();
    }
    public static Queue findEvenlyDivisibleNumbers(Queue queue) {
        int maxSize = queue.getMaxSize();
        Queue resultQueue = new Queue(maxSize);
        while (!queue.isEmpty()) {
            int num = queue.dequeue();
            if (isDivisibleByAll(num)) {
                resultQueue.enqueue(num);
            }
        }
        return resultQueue;
    }
    private static boolean isDivisibleByAll(int num) {
        for (int i = 1; i <= 10; i++) {
            if (num % i != 0) {
                return false;
            }
        }
        return true;
    }
}
```

Generic Types – Tryout

```
class Container<T> {
	private T t;
	public void set(T t) {
		this.t = t;
	}
	public T get() {
		return t;
	}
}
class Tester {
	public static void main(String[] args) {
		Container<Integer> integerContainer = new Container<>();
		integerContainer.set(1);
	//integerContainer.set("Jeo");	//Uncomment the code and check if String can be passed to the set() method
		System.out.println("Inside Integer Container : "+integerContainer.get());
		Container<String> stringContainer = new Container<>();
	//stringContainer.set(1);	//Uncomment the code and check if Integer can be passed to the set() method
		stringContainer.set("Jeo");
		System.out.println("Inside String Container : "+stringContainer.get());
	}
}
```

Generic Method – Tryout

```
class GenericDemo{
  //Generic Method
	public static <E> void display(E[] arr) {
		for (E element : arr) {
			System.out.println(element);
```

```
        }
    }
    public static void main(String[] args) {
        String[] names= { "Luke", "Mia", "Mathew" };
        display(names);
        System.out.println();
        Integer[] numbers = { 1, 2, 3, 4, 5 };
        display(numbers);
    }
}
```

ArrayList - Tryout 1

```
import java.util.ArrayList; // Importing the ArrayList class
import java.util.List;
class Tester {
    public static void main(String[] args) {
        List<String> food = new ArrayList<String>(); // Creating a list of String elements
        food.add("Pizza"); // Adding elements
        food.add("Burger");
        food.add("Pasta");
        food.add("Sandwich");
        System.out.println("Food items: " + food);
    }
}
```

ArrayList - Tryout 2

```
import java.util.ArrayList;
import java.util.List;
class Tester {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<Integer>();  // Creating an ArrayList
object
```

```
		// Adding the elements to the list
		numbers.add(1);
		numbers.add(2);
		numbers.add(3);
		numbers.add(4);
		numbers.add(5);
		numbers.add(6);
		System.out.println("numbers list: " + numbers);
		// Adding the number 15 at a particular index (index: 3) in the ArrayList
		numbers.add(3, 15);
		System.out.println("Observe the index position 3: " + numbers);
		// Finding the size of the ArrayList
		System.out.println("Size of the ArrayList: " + numbers.size());
		// Retrieving the element at a specified index
		System.out.println("The number present at the fifth index position is " +
numbers.get(5));
		// Modifying the element at a specified index (index: 2)
		numbers.set(2, 200);
		System.out.println("The number at the 2nd index position is changed from 3 to
200");
	}
}
```

ArrayList - Tryout 3

```
import java.util.ArrayList;
import java.util.List;
class Tester {
	public static void main(String[] args) {
		List<String> names = new ArrayList<String>();
		names.add("Brian");
		names.add("Ross");
		names.add("Steve");
		names.add("Rachel");
		names.add("Steve");
```

```
		//Checking whether any element is present or not
		if (names.isEmpty()) {
			System.out.println("No names are present!!");
		}
		//Displaying the number of names
		System.out.println("Number Of names: " + names.size());
		//Creating newNames list
		List<String> newNames = new ArrayList<String>();
		newNames.add("Emily");
		newNames.add("Melissa");
		// Adding elements of newNames list into names
		names.addAll(newNames);
		//Displaying all names
		System.out.println("The list of names after adding all the names from newNames
to names: ");
	System.out.println("=====================================");
		for (String name : names) {
			System.out.println(name);
		}
	System.out.println("=====================================");

		// Checking whether the name Ross is present or not
		if (names.contains("Ross")) {
			System.out.println("This name is already present!");
		} else {
			System.out.println("This name is not present!");
		}
		//Converting list to array
		Object[] namesArray = names.toArray();
		// Deleting all the names from the names list
		names.clear();
	System.out.println("=====================================");
	System.out.println("Checking whether the names list is empty or not : ");
	//Confirming whether all the elements are deleted or not
		System.out.println(names.isEmpty());
```

```
	}
}
```

Iterating through ArrayList using for-each – Tryout

```
import java.util.ArrayList;
import java.util.List;
class Student {
	private int studentId;
	private String studentName;
	private boolean courseRegistered;
	public Student(int studentId, String studentName, boolean courseRegistered) {
		this.studentId = studentId;
		this.studentName = studentName;
		this.courseRegistered = courseRegistered;
	}
	public int getStudentId() {
		return studentId;
	}
	public void setStudentId(int studentId) {
		this.studentId = studentId;
	}
	public String getStudentName() {
		return studentName;
	}
	public void setStudentName(String studentName) {
		this.studentName = studentName;
	}
	public boolean getCourseRegistered() {
		return courseRegistered;
	}
	public void setCourseRegistered(boolean courseRegistered) {
		this.courseRegistered = courseRegistered;
	}
}
```

```
class Tester {
	public static void main(String[] args) {
		List<Student> students = new ArrayList<Student>();
		students.add(new Student(1001, "Steve", true));
		students.add(new Student(1002, "Rachel", false));
		students.add(new Student(1003, "Monica", true));
		students.add(new Student(1004, "David", true));
		List<String> studentNames = new ArrayList<String>();
		for (Student student : students) {
			studentNames.add(student.getStudentName());
			System.out.println("Student Id: " + student.getStudentId());
			System.out.println("Student Name: " + student.getStudentName());
			System.out.println("Course Registered: " +
student.getCourseRegistered());
		}

	System.out.println("========================================");
		System.out.println("Student Names: " + studentNames);
	}
}
```

iterator() method – Tryout

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
class Student {
	private int studentId;
	private String studentName;
	private boolean courseRegistered;

	public Student(int studentId, String studentName, boolean courseRegistered) {
		this.studentId = studentId;
		this.studentName = studentName;
		this.courseRegistered = courseRegistered;
	}
```

```
	public int getStudentId() {
		return studentId;
	}

	public void setStudentId(int studentId) {
		this.studentId = studentId;
	}

	public String getStudentName() {
		return studentName;
	}

	public void setStudentName(String studentName) {
		this.studentName = studentName;
	}

	public boolean isCourseRegistered() {
		return courseRegistered;
	}

	public void setCourseRegistered(boolean courseRegistered) {
		this.courseRegistered = courseRegistered;
	}

}

class Tester {
	public static void main(String[] args) {
		List<Student> students = new ArrayList<Student>();
		students.add(new Student(1001, "Steve", true));
		students.add(new Student(1002, "Rachel", false));
		students.add(new Student(1003, "Monica", true));
		students.add(new Student(1004, "David", true));

		ListIterator<Student> item = students.listIterator();
		System.out.println("Student names");
		while (item.hasNext()) {
			System.out.println(item.next().getStudentName());
		}

      System.out.println();

		System.out.println("Student names in reverse order");
		while (item.hasPrevious()) {
			System.out.println(item.previous().getStudentName());
```

```
                }
        }
}
```

ARRAYLIST EXERCISE

```
import java.util.List;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;
class Order {
    private int orderId;
    private List<String> itemNames;
    private boolean cashOnDelivery;
    public Order(int orderId, List<String> itemNames, boolean cashOnDelivery) {
        this.orderId = orderId;
        this.itemNames = itemNames;
        this.cashOnDelivery = cashOnDelivery;
    }
    public int getOrderId() {
        return orderId;
    }
    public void setOrderId(int orderId) {
        this.orderId = orderId;
    }
    public List<String> getItemNames() {
        return itemNames;
    }
    public void setItemNames(List<String> itemNames) {
        this.itemNames = itemNames;
    }
    public boolean isCashOnDelivery() {
        return cashOnDelivery;
    }
```

```
    public void setCashOnDelivery(boolean cashOnDelivery) {
        this.cashOnDelivery = cashOnDelivery;
    }
    @Override
    public String toString() {
        return "Order Id: " + getOrderId() + ", Item names: " + getItemNames() + ", Cash on
delivery: " + isCashOnDelivery();
    }
}

class Tester {
    public static List<String> getItems(List<Order> orders) {
        Set<String> itemSet = new HashSet<>();
        for (Order order : orders) {
            itemSet.addAll(order.getItemNames());
        }
        return new ArrayList<>(itemSet);
    }
    public static void main(String[] args) {
        List<Order> orders = new ArrayList<Order>();
        List<String> items1 = new ArrayList<String>();
        items1.add("FriedRice");
        items1.add("Pasta");
        items1.add("Tortilla");
        orders.add(new Order(101, items1, true));
        List<String> items2 = new ArrayList<String>();
        items2.add("Pizza");
        items2.add("Pasta");
        orders.add(new Order(102, items2, true));
        List<String> items3 = new ArrayList<String>();
        items3.add("Burger");
        items3.add("Sandwich");
        items3.add("Pizza");
        orders.add(new Order(103, items3, true));
```

```
        List<String> items = getItems(orders);
        System.out.println("List of Items:");
        for (String item : items) {
            System.out.println(item);
        }
    }
}
```

ARRAYLIST ASSIGNMENT

```
import java.util.ArrayList;
import java.util.List;
class Participant {
    private String participantName;
    private String participantTalent;
    private double participantScore;
    public Participant(String participantName, String participantTalent, double participantScore) {
        this.participantName = participantName;
        this.participantTalent = participantTalent;
        this.participantScore = participantScore;
    }
    public String getParticipantName() {
        return participantName;
    }
    public void setParticipantName(String participantName) {
        this.participantName = participantName;
    }
    public String getParticipantTalent() {
        return participantTalent;
    }
    public void setParticipantTalent(String participantTalent) {
        this.participantTalent = participantTalent;
    }
    public double getParticipantScore() {
        return participantScore;
    }
```

```
    public void setParticipantScore(double participantScore) {
        this.participantScore = participantScore;
    }
    @Override
    public String toString() {
        return "Participant Name: " + getParticipantName() + ", Participant Talent: " +
getParticipantTalent() + ", Participant Score: " + getParticipantScore();
    }
}
class Tester {
    public static List<Participant> generateListOfFinalists(Participant[] finalists) {
        List<Participant> finalistsList = new ArrayList<>();
        for (Participant finalist : finalists) {
            finalistsList.add(finalist);
        }
        return finalistsList;
    }
    public static List<Participant> getFinalistsByTalent(List<Participant> finalists, String talent) {
        List<Participant> finalistsByTalent = new ArrayList<>();
        for (Participant finalist : finalists) {
            if (finalist.getParticipantTalent().equalsIgnoreCase(talent)) {
                finalistsByTalent.add(finalist);
            }
        }
        return finalistsByTalent;
    }
    public static void main(String[] args) {
        Participant finalist1 = new Participant("Hazel", "Singing", 91.2);
        Participant finalist2 = new Participant("Ben", "Instrumental", 95.7);
        Participant finalist3 = new Participant("John", "Singing", 94.5);
        Participant finalist4 = new Participant("Bravo", "Singing", 97.6);
        Participant[] finalists = { finalist1, finalist2, finalist3, finalist4 };
        List<Participant> finalistsList = generateListOfFinalists(finalists);
        System.out.println("Finalists");
        for (Participant finalist : finalistsList)
```

```
        System.out.println(finalist);
    String talent = "Singing";
    System.out.println("Finalists in " + talent + " category");
    List<Participant> finalistsCategoryList = getFinalistsByTalent(finalistsList, talent);
    for (Participant finalist : finalistsCategoryList)
        System.out.println(finalist);
  }
}
```

LINKED LIST

Methods in LinkedList – Tryout

```
import java.util.List;
import java.util.LinkedList;
class EuropeTrip {
        public static void main(String args[]) {
                // Creating a LinkedList
                List<String> cities = new LinkedList<String>();
                // Adding elements
                cities.add("Milan");
                cities.add("Venice");
                cities.add("Munich");
                cities.add("Vienna");
                // Displaying elements
                System.out.println(cities);
                // Inserting elements
                cities.add(3, "Prague");
                System.out.println(cities);
                // Removing elements
                cities.remove("Munich");
                System.out.println(cities);
                // Replacing element
                cities.set(2, "Berlin");
                System.out.println(cities);
```

```
        // Displaying size
        System.out.println(cities.size());
        // Checking if an element is present
        System.out.println(cities.contains("Paris"));
        // Getting element at specific position
        System.out.println(cities.get(0));
        // Clearing the elements from the LinkedList
        cities.clear();
        System.out.println(cities);
        // Try to test the other methods of the LinkedList class
    }
}
```

LINKEDLIST EXERCISE

```
import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;
class Tester {
   public static List<Object> concatenateLists(List<Object> listOne, List<Object> listTwo) {
      List<Object> concatenatedList = new LinkedList<Object>(listOne);

      Iterator<Object> iterator = ((LinkedList<Object>) listTwo).descendingIterator();
      while (iterator.hasNext()) {
         concatenatedList.add(iterator.next());
      }
      return concatenatedList;
   }
   public static void main(String args[]) {
      List<Object> listOne = new LinkedList<Object>();
      listOne.add("Hello");
      listOne.add(102);
      listOne.add(25);
      listOne.add(38.5);
```

```
        List<Object> listTwo = new LinkedList<Object>();
        listTwo.add(150);
        listTwo.add(200);
        listTwo.add('A');
        listTwo.add("Welcome");
        List<Object> concatenatedList = concatenateLists(listOne, listTwo);
        System.out.println("Concatenated linked list:");
        for (Object value : concatenatedList) {
            System.out.print(value + " ");
        }
    }
}
```