Day 1 – Built-in Data Structures & Libraries in JAVA

1. **Java I/O**
2. **Sequence Containers**
    a. **Array / ArrayList**
    b. **LinkedList (Doubly-Linked)**
3. **Associative Containers**
    a. **HashSet**
    b. **HashMap**
4. **Container Adaptors**
    a. **Stack/Queue/Deque (all actually LinkedList)**
    b. **PriorityQueue**

1. Java I/O

```
// Java Output
System.out.println(“Hello World”);
System.out.print(“No newline”);
System.out.print(String.format(“%d %s\n”, 2, “hi”));

// Java Input (Scanner)
Scanner in = new Scanner(System.in);
[var] v = in.[function]();    //[function] -> type [var] return
String line = in.nextLine();  //Reads one line till ‘\n’
String[] line_tokens = line.split(“ “); //Splits into String[]

// Stub answer file with I/O
import java.util.Scanner;
public class STL{
      public static void main(String[] args){
            Scanner in = new Scanner(System.in);
            // impt: nextLine() after nextInt() to consume trailing newline
            int a = in.nextInt(); in.nextLine();
            // Consume an entire line of contents
            String line = in.nextLine();
            // Splits with specified delimiter
            String[] tokens = line.split(",");
            // Prints with argument formatting
            System.out.print(String.format("%d\n", a));
            // ForEach loop through tokens array and prints
            for (String s: tokens) System.out.println(s);
      }
}
```

2a. Array/ArrayList

```
String[] s_arr = new String[s_size];    // Static num. elements

// Declaration + Initialization ArrayList<T>
import java.util.ArrayList;
ArrayList<String> s_li = new ArrayList<String>();

s_li.add("element");          // O(1)
s_li.get(0);                  // O(1)
s_li.set(idx, "new");         // O(1)
s_li.size();                  // O(1)

s_li.remove(idx);             // O(n)
s_li.subList(from, to);       // O(n)

// Sorting on ArrayList
import java.util.Collections;
Collections.sort(s_li);
```

BONUS: Sorting on ArrayList with custom objects

```
// Sorting on ArrayList with Custom comparator/object
// In this case, we build a Pair<T1, T2> generic-typed object
class Pair<T1, T2>{
	public T1 first;
	public T2 second;
	public Pair(T1 first, T2 second){
		this.first = first; this.second = second;
	}
	public String toString(){
		return String.format("(%s,%s)", first.toString(), second.toString());
	}
}
// Ascending order of sort
class comp_pair implements Comparator<Pair<String, Integer>>{
	public int compare(Pair<String, Integer> a, Pair<String, Integer> b){
		if (a.first.compareTo(b.first) == 0 &&
			a.second.compareTo(b.second) == 0) return 0;
		else if (a.first.compareTo(b.first) == 0)
			return a.second.compareTo(b.second);
		else return a.first.compareTo(b.first);
	}
}
class main{
	public static void main(String[] args){
		Pair<String, Integer> p1 = new Pair<String, Integer>("b", 2);
		Pair<String, Integer> p2 = new Pair<String, Integer>("a", 5);
		Pair<String, Integer> p3 = new Pair<String, Integer>("b", -1);
		Pair<String, Integer> p4 = new Pair<String, Integer>("a", 3);
		ArrayList<Pair<String, Integer>> p_li = new ArrayList<~>();
		p_li.add(p1); p_li.add(p2); p_li.add(p3); p_li.add(p4);
		Collections.sort(p_li, new comp_pair());
		for(Pair<String, Integer> p: p_li) System.out.println(p.toString());
	}
}
// Note: We can also have Pair implement the Comparable interface
```

2b. LinkedList (Doubly Linked implementation)

```
import java.util.LinkedList;
LinkedList<Integer> ll = new LinkedList<Integer>(); // Integer

ll.add(1); ll.addLast(1);          // O(1) - Identical
ll.addFirst(0);                    // O(1)
ll.getFirst(); ll.getLast();       // O(1) - Access
ll.removeFirst(); ll.removeLast(); // O(1) - Removal from ends
ll.size();                         // O(1)

ll.addAll(Collection<E> c);        // O(n) - Extends the list
```

3. Associative Containers: HashSet / HashMap

```
HashSet<String> hs = new HashSet<String>();
HashMap<String, String> hm = new HashMap<String, String>();

hs.add("one");                  // O(1)
hs.contains("one");             // O(1)
hs.remove("one");               // O(1) - removes if present

// Iterating through a HashSet
for (Stirng s: hs) System.out.println(s);

hm.put("key", "value");         // O(1) – replaces if present
hm.get("key");                  // O(1)
hm.containsKey("key");          // O(1)
hm.remove("key");               // O(1)
hm.isEmpty(); hm.size();        // O(1)
hm.replace("key", "n_val");     // O(1) – only replace if present

// Iterating through a HashMap
for (Map.Entry<String, String> e: hm.entrySet()){
     String key = e.getKey();
     String val = e.getValue();
}
```

4a. Stack/Queue/Deque (using ArrayDeque)

```
import java.util.ArrayDeque;
ArrayDeque<String> stack = new ArrayDeque<String>();
ArrayDeque<String> queue = new ArrayDeque<String>();
ArrayDeque<String> dequeue = new ArrayDeque<String>();

// Common functions
.isEmpty(); .size();          // O(1)

// Exposing the Stack interface
stack.push("ele");            // O(1)
stack.pop();                  // O(1)

// Queue interface
queue.add("ele");             // O(1) - Adds to end of queue
queue.poll();                 // O(1) - Returns front else null
queue.peek();                 // O(1) - Get but not remove

// Dequeue interface
dequeue.addFirst("ele");      // O(1) - Add to front of queue
dequeue.pollLast();           // O(1) - Remove from back
dequeue.peekLast();           // O(1) - Peek at end of queue
```

4b. PriorityQueque (heap)

```
import java.util.PriorityQueue;
PriorityQueue<String> pq = new PriorityQueue<String>();

pq.add("z");              // O(log n)
pq.peek();                // O(1) - Looks at top element
pq.poll();                // O(log n) - Remove & return top
pq.size();                // O(1)
```