COLLECTION

1) Write a Java program to
   a. search for an element in an array list.
   b. to join two array lists (use iterator)

```
import java.util.*;
public class SetA1 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        List<String> c1 = new ArrayList<String>();
        List<String> c2 = new ArrayList<String>();
        System.out.println("How many values you want to insert in list 1 & 2: ");
        int m = sc.nextInt();
        System.out.println("Enter "+m + "values to insert into List 1");
        for(int i=0 ;i<m ; i++){
            c1.add(sc.next());
        }
        System.out.println("LinkedList 1 elements:"+c1);
        System.out.println("Enter "+m + "values to insert into List 2");
        for(int i=0 ;i<m ; i++){
            c2.add(sc.next());
        }
        System.out.println("LinkedList 2 elements:"+c2);
        System.out.println("Enter elements to Search:");
        String str=sc.next();
        if (c1.contains(str)) {
            System.out.println("Found the element");
        } else {
            System.out.println("There is no such element");
        }
        ArrayList<String> a = new ArrayList<String>();
        a.addAll(c1);
        a.addAll(c2);
        System.out.println(“Join Array c1 & c2”);
        Iterator it = a.iterator();
        while(it.hasNext()){
            System.out.println( it.next());
        }
    }
}
```

2) Write a Java program to
   a. iterate a linked list in reverse order.
   b. to insert elements into the linked list at the first and last positions.

```
import java.util.LinkedList;
import java.util.Iterator;
  public class SetA2 {
  public static void main(String[] args) {
  LinkedList<String> l_list = new LinkedList<String>();
```

```
        l_list.add("Red");
        l_list.add("Green");
        l_list.add("Black");
        l_list.add("Pink");
        l_list.add("orange");
        System.out.println("Original linked list:" + l_list);
        Iterator it = l_list.descendingIterator();
        System.out.println("Elements in Reverse Order:");
        while (it.hasNext()) {
           System.out.println(it.next());
        }
        l_list.addFirst("White");
        l_list.addLast("Purple");
        System.out.println("Final linked list after adding elements at first and last position:\n" +
l_list);
    }
}
```

3) Write a Java program
   a. to empty a hash set.
   b. to clone a hash set to another hash set.

```
import java.util.*;
public class SetB1 {
   public static void main(String[] args) {
      HashSet<String> h_set = new HashSet<String>();
      h_set.add("Red");
      h_set.add("Green");
      h_set.add("Black");
      h_set.add("White");
      h_set.add("Pink");
      h_set.add("Yellow");
      System.out.println("Original Hash Set: " + h_set);
      HashSet <String> chs = new HashSet <String> ();
      chs = (HashSet)h_set.clone();
      System.out.println("Cloned Hash Set: " + chs);
      h_set.removeAll(h_set);
      System.out.println("Hash Set after removing all the elements "+h_set);
  }
}
```

4) Write a Java program
   a. to retrieve and remove the first and last element of a tree set.
   b. to find numbers less than 7 in a tree set.

```
import java.util.TreeSet;
import java.util.Iterator;
public class SetB2 {
   public static void main(String[] args) {
      TreeSet <Integer>ts = new TreeSet<Integer>();
      ts.add(10);
```

```
        ts.add(2);
        ts.add(24);
        ts.add(5);
        ts.add(16);
        ts.add(6);
        ts.add(1);
        ts.add(15);
        ts.add(7);
        System.out.println("Original tree set: "+ts);
        System.out.println("Removes the first(lowest) element: "+ts.pollFirst());
        System.out.println("Tree set after removing first element: "+ts);
        System.out.println("Removes the last element(highest): "+ts.pollLast());
        System.out.println("Tree set after removing last element: "+ts);
        TreeSet <Integer>head = new TreeSet<Integer>();
        head = (TreeSet)ts.headSet(7);
        Iterator itr = head.iterator();
        System.out.println("Tree set data: ");
        while (itr.hasNext()){
           System.out.println(itr.next() + " ");
        }
    }
}
```

5) Write a Java program
   a. to test if a map contains a mapping for the specified value.
   b. to get a shallow copy of a HashMap instance

```
import java.util.*;
public class SetC1 {
    public static void main(String args[]) {
    HashMap < Integer, String > hm = new HashMap < Integer, String > ();
    hm.put(1, "Red");
    hm.put(2, "Green");
    hm.put(3, "Black");
    hm.put(4, "White");
    hm.put(5, "Blue");
    System.out.println("The Original map: " + hm);
    System.out.println("1. Is value \'Green\' exists?");
    if (hm.containsValue("Green")) {
        System.out.println("Yes! ");
    } else {
        System.out.println("no!");
    }
    System.out.println("\n2. Is value \'orange\' exists?");
    if (hm.containsValue("orange")) {
        System.out.println("yes! - " );
    } else {
        System.out.println("No!");
    }
    HashMap<Integer,String> hm1= new HashMap<Integer,String>();
    hm1 = (HashMap)hm.clone();
    System.out.println("Cloned map: " + hm1);
 }
}
```

6) Write a Java program
   a. to get a key-value mapping associated with the greatest key and the least key in a map.
   b. to get the portion of a map whose keys range from a given key to another key.

```
import java.util.*;
import java.util.Map.Entry;
public class SetC2 {
    public static void main(String args[]) {
    TreeMap < Integer, String > tm = new TreeMap < Integer, String > ();
    tm.put(10, "Red");
    tm.put(20, "Green");
    tm.put(30, "Black");
    tm.put(40, "White");
    tm.put(50, "Pink");
    System.out.println("Orginal TreeMap content: " + tm);
    System.out.println("Greatest key: " + tm.firstEntry());
    System.out.println("Least key: " + tm.lastEntry());
    System.out.println("Sub map key-values: " + tm.subMap(20, true, 40, true));
    }
}
```

MULTITHREADING

1) Write a program that create 2 threads – each displaying a message (Pass the message as a parameter to the constructor). The threads should display the messages continuously till the user presses ctrl-c. Also display the thread information as it is running.

```
public class ContinuousMessages implements Runnable {
    private String message;
    public ContinuousMessages(String message) {
        this.message = message;
    }
    public void run() {
        while (true) {
            System.out.println(message);
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) {
        Thread thread1 = new Thread(new ContinuousMessages("Thread 1: Hello, World!"));
        Thread thread2 = new Thread(new ContinuousMessages("Thread 2: Hello, Universe!"));
        thread1.start();
        thread2.start();
        System.out.println("Thread 1: " + thread1.getName() + ", ID: " + thread1.getId());
        System.out.println("Thread 2: " + thread2.getName() + ", ID: " + thread2.getId());
    }
}
```

2) Write a java program which will display name and priority of current thread. Change name of Thread to MyThread and priority to 2. Display the details of Thread.

```
public class MainThread
{
    public static void main(String arg[])
    {
        Thread t=Thread.currentThread();
        System.out.println("Current Thread:"+t);
//Change Name
        t.setName("My Thread ");
        System.out.println ("After the name is Changed:"+t);
        try      {
            for(int i=2;i>0;i--)
            {
                System.out.println(i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

3) Write a java program to calculate the sum and average of an array of 1000 integers (generated randomly) using 10 threads. Each thread calculates the sum of 100 integers. Use these values to calculate average. [Use join method]

```
import java.util.*;
class thread implements Runnable{
            Thread t;
                        int i,no,sum;
                        int a[]=new int[1000];
                        thread(String s,int n){
                                    Random rs = new Random();
                                                t=new Thread(this,s);
                                                no=n;
                                                int j=0;
                                                for(i=1;i<=1000;i++){
                                                            a[j]=rs.nextInt()%100;;
                                                             j++;
                                                }
                                    t.start();
```

```
                    }
                     public void run() {
                    for(i=0;i<100;i++){
                              sum=sum+a[no];
                                        no++;
                    }
                    System.out.println("Sum = "+sum);
                    System.out.println("Avg ="+sum/100);
          }
}
public class CalculateSum{
          public static void main(String[] arg) throws InterruptedException {
                    thread t1=new thread("g",1);
                    t1.t.join();
                    thread t2=new thread("r",100);
                    t2.t.join();
                    thread t3=new thread("s",200);
                    t3.t.join();
                    thread t4=new thread("t",300);
                    t4.t.join();
                    thread t5=new thread("p",400);
                    t5.t.join();
                    thread t6=new thread("p",500);
                    t5.t.join();
                    thread t7=new thread("p",600);
                    t5.t.join();
                    thread t8=new thread("p",700);
                    t5.t.join();
                    thread t9=new thread("p",800);
                    t5.t.join();
                    thread t10=new thread("p",900);
                    t5.t.join();
          }
}
```

4) Define a thread called “PrintText_Thread” for printing text on command prompt for n number of times. Create three threads and run them. Pass the text and n as parameters to the thread constructor. Example:
i. First thread prints “I am in FY” 10 times
ii. Second thread prints “I am in SY” 20 times
iii. Third thread prints “I am in TY” 30 times

```
public class PrintTextThread implements Runnable {
   private String text;
   private int n;
   public PrintTextThread(String text, int n) {
      this.text = text;
      this.n = n;
```

```
    }
    public void run() {
        for (int i = 0; i < n; i++) {
            System.out.println(text);
        }
    }
    public static void main(String[] args) {
        PrintTextThread thread1 = new PrintTextThread("I am in FY", 2);
        PrintTextThread thread2 = new PrintTextThread("I am in SY", 4);
        PrintTextThread thread3 = new PrintTextThread("I am in TY", 6);
        Thread t1 = new Thread(thread1);
        Thread t2 = new Thread(thread2);
        Thread t3 = new Thread(thread3);
        t1.start();
        t2.start();
        t3.start();
    }
}
```

5) Write a java program to create a class called FileWatcher that can be given several filenames. The class should start a thread for each file name. If the file exists, the thread will write a message to the console and then end. If the filw does not exist, the thread will check for the existence of its file after every 5 seconds till the file gets created.

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class FileWatcher {
    private List<String> fileNames;

    public FileWatcher(List<String> fileNames) {
        this.fileNames = fileNames;
    }
    public void start() {
        List<Thread> threads = new ArrayList<>();
        for (String fileName : fileNames) {
            Thread thread = new Thread(() -> {
                while (!new File(fileName).exists()) {
                    try {
                        Thread.sleep(5000); // Sleep for 5 seconds
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("File " + fileName + " exists.");
            });
            threads.add(thread);
            thread.start();
        }
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) {
        List<String> fileNames = new ArrayList<>();
        fileNames.add("file1.txt");
        fileNames.add("file2.txt");
        fileNames.add("file3.txt");
        FileWatcher fileWatcher = new FileWatcher(fileNames);
        fileWatcher.start();
    }
}
```