Task 1: Write a custom Comparator to sort a list of Employee objects by their salary and then by name if the salary is the same.

```
package Day9.Task1;

public class Employee {
    String name;
    double sal;
    public Employee( String name, double sal) {
        this.name = name;
        this.sal = sal;
    }

    @Override
    public String toString() {
        return "Employee : " + name + ", " + sal;
    }

    public String getName() {
        return name;
    }

    public double getSal() {
        return sal;
    }
}


package Day9.Task1;

import java.util.Comparator;

public class EmpComparator implements Comparator<Employee> {

    @Override
    public int compare(Employee e1, Employee e2) {
        // First compare by salary
        //int salaryComparison = Double.compare(e1.getSal(), e2.getSal());

        //if (salaryComparison != 0) {
        //    return salaryComparison;
        //} else {
            // If salary is the same, compare by name
        //    return e1.getName().compareTo(e2.getName());
        //}

        //if(e1.getSal() < e2.getSal()){
        //    return -1;
        //}
        //return 0;
        int cname = e1.getName().compareTo(e2.getName());
        if(cname != 0){
            return cname;
        }
        return Double.compare(e1.getSal(), e2.getSal());
    }
}

package Day9.Task1;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        employees.add(new Employee("Alice", 80000.0));
        employees.add(new Employee("Bob", 75000.0));
        employees.add(new Employee("Charlie", 80000.0));
        employees.add(new Employee("David", 90000.0));
        employees.add(new Employee("Alice", 45000));

        Collections.sort(employees, new EmpComparator());
        employees.forEach(System.out::println);
    }
}
```

Task2:

a) Implement a function called BruteForceSort that sorts an array using the brute force approach. Use this function to sort an array created with InitializeArray.

```
package Day9.Task2;

import java.util.Arrays;

public class BruteForceSort {
    public static void bruteForceSort(int[] arr) {
        int n = arr.length;
        boolean swapped = false;
        do {
            for (int i = 1; i < n; i++) {
                if (arr[i - 1] > arr[i]) {
                    // Swap elements
                    int temp = arr[i - 1];
                    arr[i - 1] = arr[i];
                    arr[i] = temp;
                    swapped = true;
                }
            }
        } while (swapped);
    }
    public static void main(String[] args) {
        int[] array = {5, 2, 9, 1, 5, 6};
        System.out.println("Before sorting: " + Arrays.toString(array));
        bruteForceSort(array);
        System.out.println("After sorting: " + Arrays.toString(array));
    }
}
```

b) Write a function named PerformLinearSearch that searches for a specific element in an array and returns the index of the element if found or -1 if not found.

```
package Day9.Task2;

public class LinearSearch {
    public static void main(String[] args) {
        int[] array = {5, 2, 9, 1, 5, 6};
        int target = 9;
        int index = performLinearSearch(array, target);
        if (index != -1) {
            System.out.println("Element found at index " + index);
        } else {
            System.out.println("Element not found in the array");
        }
    }

    private static int performLinearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i; // Element found, return its index
            }
        }
        return -1;
    }
}
```

Task 3: Given an array of integers, write a program that finds if there are two numbers that add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice. Optimize the solution for time complexity.

```
package Day9.Task3;

import java.util.HashMap;
import java.util.Map;

public class TwoSum {

    public static int[] findTwoSum(int[] nums, int target) {
        Map<Integer, Integer> seen = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
          int complement = target - nums[i];
          if (seen.containsKey(complement)) {
            return new int[]{seen.get(complement), i}; // Optimized order for clarity
          }
          seen.put(nums[i], i);
        }
        return null;
      }

    public static void main(String[] args) {
        int[] nums = {2, 7, 11, 15};
        int target = 9;
        int[] result = findTwoSum(nums, target);
        if (result != null) {
            System.out.println("Two numbers found that add up to " + target + ": ");
            System.out.println("[" + result[0] + ", " + result[1] + "]");
        } else {
            System.out.println("No such pair found.");
        }
    }
}
```

Task 4: Write a recursive function named SumArray that calculates and returns the sum of elements in an array, demonstarte with example.

```
package Day9.Task4;

public class RecursiveSumArray {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        int sum = sumArray(array);
        System.out.println("Sum of elements in the array: " + sum);
    }

    private static int sumArray(int[] arr) {
        return sumArrayRecursive(arr, 0);
    }

    private static int sumArrayRecursive(int[] arr, int i) {
        if (i == arr.length) {
            return 0; // Base case: sum of an empty array is 0
        }
        return arr[i] + sumArrayRecursive(arr, i + 1);
    }
}
```

Task 5:

a) Implement a method SliceArray that takes an array, a starting index, and an end index, then returns a new array containing the elements from the start to the end index.

```
package Day9.Task5;

import java.util.Arrays;

public class ArrayOperations {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        int startIndex = 1;
        int endIndex = 3;
        int[] slicedArray = sliceArray(array, startIndex, endIndex);
        System.out.println("Original array: " + Arrays.toString(array));
        System.out.println("Sliced array from index " + startIndex + " to " + endIndex + ": " + Arrays.toString(slicedArray));
    }

    private static int[] sliceArray(int[] arr, int startIndex, int endIndex) {
        if (startIndex < 0 || endIndex >= arr.length || startIndex > endIndex) {
            throw new IllegalArgumentException("Invalid start and end indices");
        }

        int[] slicedArray = new int[endIndex - startIndex + 1];
        for (int i = startIndex, j = 0; i <= endIndex; i++, j++) {
            slicedArray[j] = arr[i];
        }
        return slicedArray;
    }
}
```

b) Create a recursive function to find the nth element of a Fibonacci sequence and store the first n elements in an array.

```
package Day9.Task5;

import java.util.Arrays;

public class Fibonacci {
    public static void main(String[] args) {
        int n = 10; // Number of Fibonacci numbers to generate
        int[] fibonacciArray = new int[n];
        generateFibonacci(n, fibonacciArray);

        System.out.println("First " + n + " Fibonacci numbers:");
        System.out.println(Arrays.toString(fibonacciArray));

        int nthElement = fibonacci(n - 1); // Indexing starts from 0, so n-1 for nth element
        System.out.println("The " + n + "th Fibonacci number is: " + nthElement);
    }

    private static int fibonacci(int n) {
        if (n == 0 || n == 1) {
            return n;
        }
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    private static void generateFibonacci(int n, int[] fibonacciArray) {
        for (int i = 0; i < n; i++) {
            fibonacciArray[i] = fibonacci(i);
        }
    }
}
```

Task 6: Creating and Managing Threads

Write a program that starts two threads, where each thread prints numbers from 1 to 10 with a 1-second delay between each number

```
package Day9.Task6;

public class ThreadEx{
    public static void main(String[] args) {
        Thread th1 = new Thread(){
            @Override
            public void run(){
                for (int i = 1; i <= 10; i++) {
                    System.out.println("Thread 1 : " + i);
                    try {
                        Thread.sleep(1000); // Sleep for 1 second
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        Thread th2 = new Thread(){
            @Override
            public void run(){
                for (int i = 1; i <= 10; i++) {
                    System.out.println("Thread 2 : " + i);
                    try {
                        Thread.sleep(1000); // Sleep for 1 second
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        th1.start();
        th2.start();

        try {
            th1.join();
            th2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```