ASSIGNMENT: 1 LAST DATE : 27july2025

- For today do the following:

- 1. Create a class
  class Student {
  }

- 2. Create a static method

  class Student {
      static void schoolName() {
          System.out.println("School Name: ABC Public School");
      }
  }

  Call it using:  Student.schoolName();

- 3. Create a non static method

  class Student {
      void displayName() {
          System.out.println("Student Name: Rahul");
      }
  }

4. Create a static data member

```
class Student {
    static String school = "ABC Public School";
}
```

- 5. Create a non static data member

```
class Student {
    int rollNo;
}
```

- Read about Tokens, JDK, JVM , Inheritance and come

**Tokens** are the smallest units in Java.
Examples:

- Keywords: `class`, `int`
- Identifiers: `Student`
- Literals: `10`, `"Hello"`
- Operators: `+`, `=`
- Separators: `{ } ;`

### JDK (Java Development Kit)

- Used to **develop Java programs**
- Contains compiler (`javac`), JRE, tools

### JVM (Java Virtual Machine)

- Executes Java bytecode
- Makes Java **platform independent**

You are given a list of integers representing a sequence of moves in a game.

- Each integer represents a move (like a key)
- You need to keep track of how many times each move has appeared using a `HashMap`.

- After processing the whole list, remove all moves from the map whose frequency is **less than the average frequency** of all moves
- Return the list of remaining moves sorted by their frequencies in descending order
- If frequencies are equal, sort by the move value ascending

**Example:**

Input: `[2, 3, 2, 4, 3, 2, 5, 4]`

Frequencies
2 → 3 times
3 → 2 times
4 → 2 times
5 → 1 time

- Average frequency = (3 + 2 + 2 + 1) / 4 = 2
- Remove moves with frequency < 2 → remove move 5
- Remaining: 2, 3, 4
- Sort by frequency desc and then move asc → `[2 (3), 3 (2), 4 (2)]`

Output: `[2, 3, 4]`

## Inheritance

Inheritance means **one class acquiring properties of another class** using extends.

```
class A {

    int x = 10;

}


class B extends A {

    void show() {

        System.out.println(x);

    }

}
```

ASSIGNMENT: 2  LAST DATE : 27july2025

- **Write a program for Type Safety**

```
public class TypeSafety {
   public static void main(String[] args) {
      int a = 10;
      // a = "Hello"; ❌ Error (Type mismatch)
      System.out.println(a);
   }
}
```

- Write a program for Upcasting and Downcasting

## Upcasting (Automatic)

Parent reference → Child object

```
class Parent {

   void show() {
      System.out.println("Parent class");
   }
}

class Child extends Parent {
   void display() {
      System.out.println("Child class");
   }

   public static void main(String[] args) {
      Parent p = new Child(); // Upcasting
      p.show();
   }
}
```

**Downcasting**

Child reference → Parent object (needs casting)

```
Parent p = new Child();
Child c = (Child) p; // Downcasting
c.display();
```

- Write a program for Overriding

```
class Parent {
   void show() {
      System.out.println("Parent Method");
   }
}

class Child extends Parent {
   @Override
   void show() {
      System.out.println("Child Method");
   }

   public static void main(String[] args) {
```

```
        Parent p = new Child();
        p.show(); // Child Method
    }
}
```

ASSIGNMENT: 3  LAST DATE : 3july2025

- **Solve Two Sum problem in O(N) time complexity**

```
public class TwoSum {
    public static int[] twoSum(int[] nums, int target) {

        HashMap<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int diff = target - nums[i];

            if (map.containsKey(diff)) {
                return new int[]{map.get(diff), i};
            }

            map.put(nums[i], i);
        }
        return new int[]{};
    }

    public static void main(String[] args) {
        int[] arr = {2, 7, 11, 15};
        int target = 9;

        System.out.println(Arrays.toString(twoSum(arr, target)));
    }
}
```

- **Solve Height Checker (1051) in O(N) time complexity**

```
public class HeightChecker {
        public static int heightChecker(int[] heights) {

            int[] count = new int[101];

            for (int h : heights)
               count[h]++;

            int index = 0, result = 0;

            for (int i = 0; i < count.length; i++) {
               while (count[i]-- > 0) {
                  if (heights[index] != i)
                     result++;
                  index++;
               }
            }
            return result;
         }

         public static void main(String[] args) {
            int[] heights = {1, 1, 4, 2, 1, 3};
            System.out.println(heightChecker(heights));
         }
      }
```

**Assignment: 4**

You are given a list of integers representing a sequence of moves in a game.

- Each integer represents a move (like a key)
- You need to keep track of how many times each move has appeared using a `HashMap`.
- After processing the whole list, remove all moves from the map whose frequency is **less than the average frequency** of all moves
- Return the list of remaining moves sorted by their frequencies in descending order
- If frequencies are equal, sort by the move value ascending

**Example Given**

**Input**

```
[2, 3, 2, 4, 3, 2, 5, 4]
```

---

## 🔹 Step 1: Count Frequency Using HashMap

We store:

- **Key** → Move
- **Value** → Number of times the move occurs

```
2 → 3 times
3 → 2 times
4 → 2 times
5 → 1 time
```

---

## 🔹 Step 2: Calculate Average Frequency

Total frequency:

```
3 + 2 + 2 + 1 = 8
```

Number of unique moves:

```
4
```

Average frequency:

```
8 / 4 = 2
```

---

## 🔹 Step 3: Remove Moves with Frequency < Average

Average frequency = **2**

Remove moves with frequency less than 2:

```
5 → 1 time ❌ removed
```

Remaining moves:

```
2 → 3

3 → 2

4 → 2
```

---

## 🔹 Step 4: Sort Remaining Moves

Sorting rules:

1. **Frequency descending**
2. **Move value ascending (if frequency same)**

Sorted result:

```
2 (3 times)

3 (2 times)

4 (2 times)
```

---

**Final Output**

```
[2, 3, 4]
```

---

**Complete Java Program**

```
public class GameMoveAnalysis {

    public static List<Integer> analyzeMoves(int[] moves) {

        // Step 1: Count frequency of each move
        HashMap<Integer, Integer> map = new HashMap<>();

        for (int move : moves) {
            map.put(move, map.getOrDefault(move, 0) + 1);
        }

        // Step 2: Calculate average frequency
        int total = 0;
        for (int freq : map.values()) {
            total += freq;
        }
```

```
        double average = (double) total / map.size();

        // Step 3: Remove moves with frequency less than
average
        map.entrySet().removeIf(entry -> entry.getValue() <
average);

        // Step 4: Sort remaining moves
        List<Map.Entry<Integer, Integer>> list =
                new ArrayList<>(map.entrySet());

        list.sort((a, b) -> {
            if (!a.getValue().equals(b.getValue())) {
                return b.getValue() - a.getValue(); //
frequency descending
            }
            return a.getKey() - b.getKey(); // move ascending
        });

        // Step 5: Store result
        List<Integer> result = new ArrayList<>();
        for (Map.Entry<Integer, Integer> entry : list) {
            result.add(entry.getKey());
        }
```

```
        return result;
    }

    public static void main(String[] args) {

        int[] input = {2, 3, 2, 4, 3, 2, 5, 4};
        System.out.println(analyzeMoves(input));
    }
}
```

---

🔹 **Output**

```
[2, 3, 4]
```