```java
//Console Inputs in one line : "465657656 fygy hjkhkh khkh  "
public static void main(String[] args) throws IOException {
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT_PATH")));

    String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

    int n = Integer.parseInt(firstMultipleInput[0]);

    int d = Integer.parseInt(firstMultipleInput[1]);

    List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))
            .map(Integer::parseInt)
            .collect(toList());

    // To read the input in multiple lines
        465657656
        fygy
        hjkhkh
        khkh

    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
   ArrayList<String> g = new ArrayList<>();
   String line;
   while (true) {
        line = bufferedReader.readLine();
        if (line.isEmpty()) {
            break;  // Exit if an empty line is entered
        }
        g.add(line);
    }
    System.out.println("You entered: " + g.toString());

    bufferedReader.close();
}

// Foreach
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.forEach(name -> {
    if (name.startsWith("A")) {
        System.out.println(name + " starts with A");
    } else {
        System.out.println(name + " does not start with A");
    }
});
            or
names.stream()
     .filter(name -> name.startsWith("A"))
     .forEach(name -> System.out.println(name + " starts with A"));

HashMap<String, Integer> map = new HashMap<>();
map.forEach((key, value) ->
        System.out.println(key + " is " + value + " years old")
    );

map.keySet().forEach(key ->
    System.out.println("Key: " + key)
);

map.values().forEach(value ->
    System.out.println("Value: " + value)
);

for (Map.Entry<String, Integer> entry : map.entrySet()) {
    System.out.println(entry.getKey() + " is " + entry.getValue() + " years old");
}

// To print the 2D array
int[][] array = new int[rows][columns];
System.out.println(Arrays.deepToString(array));
```

```java
// return 1d array
return new int[]{4,5};

// return empty 2d array
 return new int[0][];

// Sort intervals by the starting value of each interval
Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));


// To covert ArrayList to array
String[] array = list.toArray(new String[0]);

// Convert list to 2D array and return
return merged.toArray(new int[merged.size()][]);

//Convert 1d array to array list
int[] array = {1, 2, 3, 4, 5};
ArrayList<Integer> arrayList = new ArrayList<>();
arrayList = new ArrayList<>(Arrays.asList(array));

//Convert 2d array to array list
int[][] array2D = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
ArrayList<ArrayList<Integer>> arrayList2D = new ArrayList<>();
for (int[] row : array2D) {
    ArrayList<Integer> innerList = new ArrayList<>(Arrays.asList(row));
    arrayList2D.add(innerList);
}

@FunctionalInterface
public interface Comparator<T> {
    int compare(T o1, T o2);
}

// use of Comparator for the class object
List<Person> personList = new ArrayList<>();
personList.add(new Person("Alice", 25));
personList.add(new Person("Bob", 30));
personList.add(new Person("Charlie", 20));

// Create a Comparator to sort Person objects based on age
Comparator<Person> ageComparator = new Comparator<Person>() {
    @Override
    public int compare(Person p1, Person p2) {
        return Integer.compare(p1.getAge(), p2.getAge());
    }
};
Comparator<Person> reversedAgeComparator = Comparator.comparingInt(Person::getAge).reversed();

// Sort the personList using the ageComparator
Collections.sort(personList, ageComparator);

If we need min heap then use :
PriorityQueue<Integer> pq = new PriorityQueue<>();

If we need max heap
PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.reverseOrder());

//Leet code learning
int sum = Integer.MIN_VALUE;
sum = Math.max(sum,t);

Checked Exceptions: [Complie time exception]
Must be caught or declared in the method signature.[like Throws IOException]
Represent conditions that a program should handle explicitly.
Example: IOException, SQLException.

Unchecked Exceptions: [Run Time exception]
Do not need to be caught or declared.
Represent programming errors or bugs that are often not anticipated.
Example: NullPointerException, ArrayIndexOutOfBoundsException.
```

```
1. Try-With-Resources (try () { ... })
When to Use:
1) When working with resources that implement the AutoCloseable or java.io.Closeable interface.
2) To ensure automatic resource management and avoid potential resource leaks.

public class BufferedReaderWriterExample {
    public static void main(String[] args) {
        // Writing to a file
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("example.txt"))) {
            bw.write("BufferedWriter example");
            bw.newLine();
            bw.write("BufferedWriter is efficient for writing large data.");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Reading from a file
        try (BufferedReader br = new BufferedReader(new FileReader("example.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

2. Traditional Try-Catch-Finally (try { ... })
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class TraditionalTryCatchFinallyExample {
    public static void main(String[] args) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader("example.txt"));
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // here we have to write close logoic
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

Throws Clause: The main method declares that it throws IOException. This means that if an IOException
occurs, it will propagate up to the JVM, which will terminate the program if not handled.
public class BufferedReaderWriterExample  {
    public static void main(String[] args) throws IOException {
        // Reading from a file
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader("exampsle.txt"));
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
```

```java
        }
        catch (Exception e) {
            e.printStackTrace();
            System.out.println("ghjgj");
        }

        // Writing to a file
        BufferedWriter bw = null;
       try {
            bw = new BufferedWriter(new FileWriter("example.txt"));
           bw.write("BufferedWritevvbnbnbr example");
           bw.newLine();
           bw.write("BufferedWriter is efficient for writing large data.");
       }
       catch (Exception e) {
           e.printStackTrace();
       }

       finally {
           // Ensure that the BufferedWriter is closed properly
           if (bw != null) {
               bw.close();
           }
       }
    }
}

// Why it is not working
 public static void main(String[] args) throws IOException,FileNotFoundException
FileNotFoundException is a subclass of IOException, which means that when you declare a method to
throw IOException, it also covers FileNotFoundException. Therefore, it's sufficient and often
preferred to declare IOException in the throws clause if you are dealing with file operations or other
 I/O operations.

In Java, the throws keyword is used to declare that a method can throw certain exceptions. It can only
 be used at the method level, not at the class level.

// it will not work : public class Example throw IOexception {}


public class Traditionaltrycatchfinallyexample {
    public static void main(String[] args) throws IOException{
        BufferedReader br = null;
        // here filerEDADER THROW THE exception so we have already declared using throws
        br = new BufferedReader(new FileReader("example.txt"));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        br.close();
    }
}

Question 4: Exception Handling with return in finally
public class FinallyReturn {
    public static void main(String[] args) {
        System.out.println(method());
    }

    public static int method() {
        try {
            return 1;
        } finally {
            return 2;
        }
    }
}

//Output
2
```