Java question

1: sol

```
 List<String> responses = new ArrayList<>();


            for (List<String> request : requests) {
               String requestType = request.get(0);
               String url = request.get(1);

               Map<String, String> paramsDict = parseParameters(url);

               if (!paramsDict.containsKey("token")) {
                  responses.add("INVALID");
               } else {
                  String authToken = paramsDict.get("token");

                  if (!valid_auth_tokens.contains(authToken)) {
                     responses.add("INVALID");
                  } else if (requestType.equals("POST") &&
!isValidCsrf(paramsDict.get("csrf"))) {
                     responses.add("INVALID");
```

```
                } else {
                    paramsDict.remove("token");  // Remove token from
parameters
                    paramsDict.remove("csrf");   // Remove csrf from parameters
                    String paramsStr = formatParameters(paramsDict);
                    responses.add("VALID," + paramsStr);
                }
            }
        }

        return responses;
    }

    private static boolean isValidCsrf(String csrfToken) {
    return csrfToken != null && csrfToken.matches("[a-zA-Z0-9]{8,}");
    }

    private static Map<String, String> parseParameters(String url) {
        Map<String, String> paramsDict = new LinkedHashMap<>();
        String[] urlParts = url.split("\\?");
        if (urlParts.length > 1) {
            String[] params = urlParts[1].split("&");
            for (String param : params) {
                String[] keyValue = param.split("=");
                paramsDict.put(keyValue[0], keyValue[1]);
```

```
            }
        }
        return paramsDict;
    }

    private static String formatParameters(Map<String, String> paramsDict) {
        List<String> formattedParams = new ArrayList<>();
        for (Map.Entry<String, String> entry : paramsDict.entrySet()) {
            formattedParams.add(entry.getKey() + "," + entry.getValue());
        }
        return String.join(",", formattedParams);
```

2: sol

```
int count = 0;
        for (int i = 0; i < a.size(); i++) {
            for (int j = i + 1; j < a.size(); j++) {
                for (int k = j + 1; k < a.size(); k++) {
                    if ((a.get(i) + a.get(j) + a.get(k)) % d == 0) {
                        count++;
                    }
                }
            }
        }
        return count;
```

3:sol

```
List<Integer> exclusiveTimes = new ArrayList<>();
                Stack<FunctionLog> stack = new Stack<>();
                int[] totalTime = new int[n];

                for (String log : logs) {
                   String[] parts = log.split(":");
                   int id = Integer.parseInt(parts[0]);
                   String type = parts[1];
                   int timestamp = Integer.parseInt(parts[2]);

                   if (type.equals("start")) {
                      stack.push(new FunctionLog(id, type, timestamp));
                   } else {
                      FunctionLog startLog = stack.pop();
                      int executionTime = timestamp - startLog.timestamp + 1;
                      totalTime[id] += executionTime;

                      // Subtract any overlapping time from the parent function
                      if (!stack.isEmpty()) {
                         totalTime[stack.peek().id] -= executionTime;
                      }
                   }
                }

                for (int time : totalTime) {
                   exclusiveTimes.add(time);
                }

                return exclusiveTimes;
```

```
class FunctionLog {
        int id;
        String type;
        int timestamp;

        public FunctionLog(int id, String type, int timestamp) {
           this.id = id;
           this.type = type;
           this.timestamp = timestamp;
        }
      }
```

4:sol

```
    List<String> result = new ArrayList<>();

    for (String triangle : triangleToy) {
       String[] sidesStr = triangle.split(" ");
       int[] sides = Arrays.stream(sidesStr).mapToInt(Integer::parseInt).toArray();
       Arrays.sort(sides);

       if (sides[0] + sides[1] > sides[2]) {
          if (sides[0] == sides[1] && sides[1] == sides[2]) {
             result.add("Equilateral");
          } else if (sides[0] == sides[1] || sides[1] == sides[2]) {
             result.add("Isosceles");
          } else {
             result.add("None of these");
          }
       } else {
```

```
        result.add("None of these");
    }
}

return result;
```

5:sol

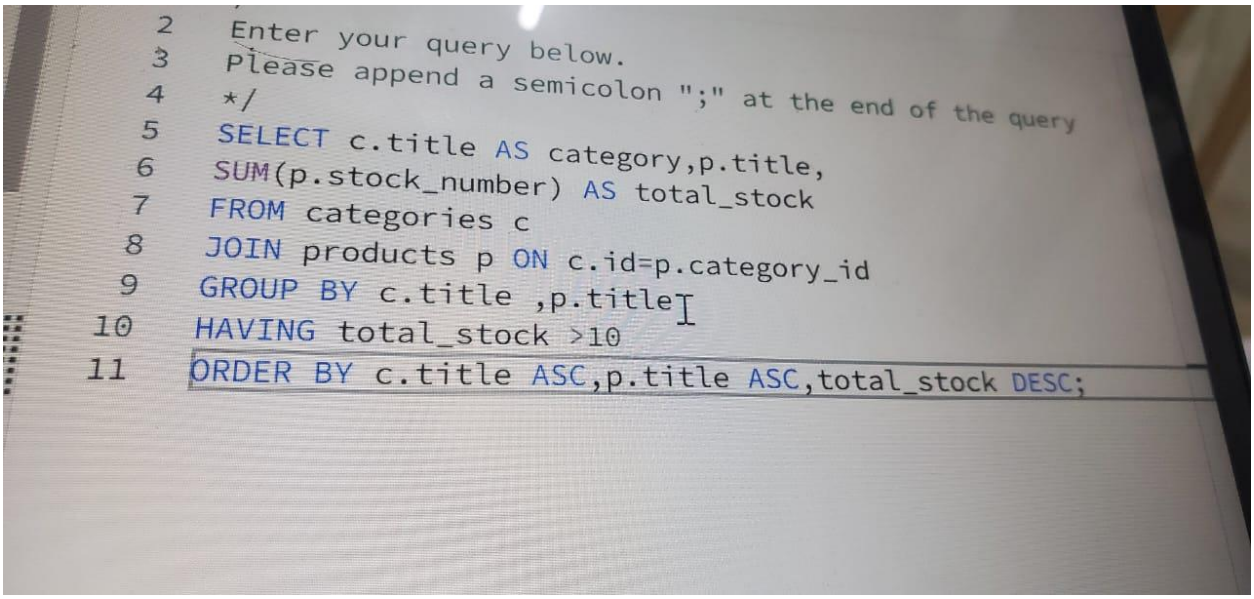
```
int compEdges = cFrom.size(); // Calculate the number of edges

Map<Integer, List<Integer>> adjList = new HashMap<>();

for (int i = 1; i <= compNodes; i++) {
    adjList.put(i, new ArrayList<>());
}

for (int i = 0; i < compEdges; i++) {
    adjList.get(cFrom.get(i)).add(cTo.get(i));
    adjList.get(cTo.get(i)).add(cFrom.get(i));
}

Set<Set<Integer>> connectedComponents = new HashSet<>();

// DFS to find connected components
for (int node = 1; node <= compNodes; node++) {
    Set<Integer> component = new HashSet<>();
    dfs(node, component, adjList);
    if (!component.isEmpty()) {
```

```
            connectedComponents.add(component);
        }
    }

    // Check if all nodes are in the same connected component
    if (connectedComponents.size() == 1) {
        return 0; // All computers are already connected
    }

    // Determine the minimum number of operations needed to connect all computers
    int operations = connectedComponents.size() - 1;

    return (operations > 0) ? operations : -1;
}

private static void dfs(int node, Set<Integer> component, Map<Integer, List<Integer>> adjList) {
    if (!component.contains(node)) {
        component.add(node);
        for (int neighbor : adjList.get(node)) {
            dfs(neighbor, component, adjList);
        }
    }
}

public static void main(String[] args) {
    int compNodes = 4;
    List<Integer> cFrom = Arrays.asList(1, 1, 3);
    List<Integer> cTo = Arrays.asList(2, 3, 2);
```

```
    int result = minOperations(compNodes, cFrom, cTo);
    System.out.println(result);
}
```

sql

6: sol



```
2   Enter your query below.
3   Please append a semicolon ";" at the end of the query
4   */
5   SELECT c.title AS category,p.title,
6   SUM(p.stock_number) AS total_stock
7   FROM categories c
8   JOIN products p ON c.id=p.category_id
9   GROUP BY c.title ,p.title
10  HAVING total_stock >10
11  ORDER BY c.title ASC,p.title ASC,total_stock DESC;
```

7:sol

```
SELECT DISTINCT S.NAME AS
STUDENT_NAME
FROM STUDENT S
JOIN BACKLOG B ON S.ID=B.STUDENT_ID;
```