It looks like you're working on an analytics system that tracks user actions and stores them. I'll help you complete the Analytics class methods. Here's a possible implementation:

1. **Constructor**: Initialize the analyticsStore and K variables.
2. **registerAction**: Register an action and store it if the number of actions reaches K.
3. **getNumberOfActionRegisteredButNotSentToAnalyticsStore**: Return the number of actions registered but not yet sent to the store.
4. **getTotalNumberOfLoggedActions**: Return the total number of logged actions.
5. **getMostFrequentlyUsedActions**: Return the most frequently used actions.

Here's the complete code:

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

enum ActionEnum {

appliedFilter,

feature1Button,

feature2Button,

feature3Button,

proceedButton

}

interface IAnalytics {

public void registerAction(ActionEnum action);

public int getNumberOfActionRegisteredButNotSentToAnalyticsStore();

public int getTotalNumberOfLoggedActions();

public List<ActionEnum> getMostFrequentlyUsedActions();

}

interface IAnalyticsStore {

public void storeActions(Queue<ActionEnum> q);

public Queue<ActionEnum> getAllStoredActions();

}

class Analytics implements IAnalytics {

private IAnalyticsStore analyticsStore;

private int K;

private Queue<ActionEnum> actionQueue;

private Map<ActionEnum, Integer> actionCountMap;

private int totalLoggedActions;

public Analytics(IAnalyticsStore analyticsStore, int K) {

this.analyticsStore = analyticsStore;

this.K = K;

this.actionQueue = new LinkedList<>();

this.actionCountMap = new HashMap<>();

this.totalLoggedActions = 0;

‘; @Override

public void registerAction(ActionEnum action) {

actionQueue.add(action);

actionCountMap.put(action, actionCountMap.getOrDefault(action, 0) + 1);

totalLoggedActions++;

if (actionQueue.size() >= K) {

analyticsStore.storeActions(new LinkedList<>(actionQueue));

actionQueue.clear();

}

}

@Override

public int getNumberOfActionRegisteredButNotSentToAnalyticsStore() {

return actionQueue.size();

}

@Override

public int getTotalNumberOfLoggedActions() {

return totalLoggedActions;

}

@Override

public List<ActionEnum> getMostFrequentlyUsedActions() {

List<ActionEnum> mostFrequentActions = new ArrayList<>();

int maxCount = Collections.max(actionCountMap.values());

for (Map.Entry<ActionEnum, Integer> entry : actionCountMap.entrySet()) {

if (entry.getValue() == maxCount) {

mostFrequentActions.add(entry.getKey());

}

}

return mostFrequentActions;

}

}

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

PrintWriter out = new PrintWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String inp[];

inp = br.readLine().split(" ");

int totalNumberOfRequests = Integer.parseInt(inp[0]),

K = Integer.parseInt(inp[1]);

IAnalyticsStore analyticsStore = new AnalyticsStore();

IAnalytics analytics = new Analytics(analyticsStore, K);

for (int queryNumber = 1; queryNumber <= totalNumberOfRequests; queryNumber++) {

inp = br.readLine().split(" ");

switch (inp[0]) {

case "registerAction":

ActionEnum actionEnum = ActionEnum.valueOf(inp[1]);

analytics.registerAction(actionEnum);

break;

case "getTotalNumberOfLoggedActions":

int totalNumberOfLoggedEvents = analytics.getTotalNumberOfLoggedActions();

out.println(totalNumberOfLoggedEvents);

break;

case "getNumberOfActionRegisteredButNotSentToAnalyticsStore":

int x = analytics.getNumberOfActionRegisteredButNotSentToAnalyticsStore();

out.println(x);

break;

case "getMostFrequentlyUsedActions":

List<ActionEnum> li = analytics.getMostFrequentlyUsedActions();

for (ActionEnum a : li) {

out.print(a + " ");

}

out.print("\n");

break;

}

}

Queue<ActionEnum> q = analyticsStore.getAllStoredActions();

for (ActionEnum a : q) {

out.print(a.toString() + " ");

}

out.print("\n");

out.flush();

out.close();

}

}

class AnalyticsStore implements IAnalyticsStore {

private Queue<ActionEnum> storedEventEnums = new LinkedList<>();

@Override

public void storeActions(Queue<ActionEnum> q) {

while (!q.isEmpty()) {

storedEventEnums.add(q.poll());

}

}

@Override

public Queue<ActionEnum> getAllStoredActions() {

return new LinkedList<>(storedEventEnums);

}

}

This implementation should cover the functionality you need. If you have any questions or need further adjustments, feel free to ask!