1.Food nutrition :

```
abstract class Food{
double proteins;
double fats;
double carbs;
double tastyScore;
public abstact void getMacroNutrients();
}
class Egg extends Food{
int tastyScore =7;
String type="non-vegetarian";
public Egg(double proteins, double fats, double carbs){
this.proteins=proteins;
this.fats=fats;
this.carbs=carbs;
}
public void getMacroNutrients(){
System.out.println("An egg has " + this.proteins + " gms of proteins, " +this.fats + " gms of fats
and " +this.carbs + " gms of carbohydrates.");
}
}
class Bread extends Food{
int tastyScore =8;
String type="vegetarian";
public Bread(double proteins, double fats, double carbs){
this.proteins=proteins;
this.fats=fats;
this.carbs=carbs;
}
public void getMacroNurients(){
System.out.println("A slice of bread has "+this.proteins+" gms of proteins, " +this.fats + " gms of
fats and "+this.carbs+ " gms of carbohydrates.");
}
}
```

2. Abstract class employee

```
import java.util.Scanner;
class Employee {
private int salary;
private String grade;

void setSalary(int salary) {
this.salary = salary;
}
int getSalary() {
return salary;
}
```

```
void setGrade(String grade) {
this.grade = grade;
}

String getGrade() {
return grade;
}

void label() {
System.out.println("Employee's data:");
}
}
class Engineer extends Employee {
private int salary;
private String grade;

@Override
void setSalary(int salary) {
this.salary = salary;
}
@Override
int getSalary() {
return salary;
}
@Override
void setGrade(String grade) {
this.grade = grade;
}
@Override
String getGrade() {
return grade;
}
@Override
void label() {
super.label();
}
  void displayInfo() {
    label();
    System.out.println("GRADE: " + getGrade());
    System.out.println("SALARY: " + getSalary());
  }
}
class Manager extends Employee {
  private int salary;
  private String grade;
  @Override
  void setSalary(int salary) {
    this.salary = salary;
  }
  @Override
  int getSalary() {
    return salary;
  }
  @Override
  void setGrade(String grade) {
    this.grade = grade;
  }
  @Override
  String getGrade() {
    return grade;
  }
  @Override
  void label() {
    super.label();
  }
  void displayInfo() {
    label();
    System.out.println("GRADE: " + getGrade());
    System.out.println("SALARY: " + getSalary());
  }
}
public class Solution {

public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
int numOfEmployees = scanner.nextInt();

for (int i = 0; i < numOfEmployees; i++) {
String type = scanner.next();
String grade = scanner.next();
int salary = scanner.nextInt();

if (type.equals("ENGINEER")) {
Engineer engineer = new Engineer();
engineer.setGrade(grade);
engineer.setSalary(salary);
engineer.displayInfo();
} else if (type.equals("MANAGER")) {
Manager manager = new Manager();
manager.setGrade(grade);
manager.setSalary(salary);
manager.displayInfo();
}
}
scanner.close();
}
```

```
}

3.import java.util.Scanner;

class ItemSeparator {
    String name;
    double price;
    int quantity;

    public ItemSeparator(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}

public class Solution {
    public static void main(String args[]) throws Exception {
        Scanner sc = new Scanner(System.in);
        String sub = sc.nextLine();

        // Split the input string based on the delimiter
        String[] parts = sub.split("\\$\\$##");
        // Parse the parts into the appropriate types
        String itemName = parts[0];
        double itemPrice = Double.parseDouble(parts[1]);
        int itemQuantity = Integer.parseInt(parts[2]);

        // Create an ItemSeparator object
        ItemSeparator itemData = new ItemSeparator(itemName, itemPrice, itemQuantity);

        // Output the data
        System.out.println("Item Name: " + itemData.getName());
        System.out.println("Item Price: " + itemData.getPrice());
        System.out.println("Item Quantity: " + itemData.getQuantity());
    }
}
```

4. StudentRank(DS)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
class StudentRank {
private String[] students;
private int[] ranks;
public StudentRank(String[] studentArray, int[] ranksArray) {
this.students = studentArray;
this.ranks = ranksArray;
}

public String highestRank() {
int index = 0;
int currentMax = Integer.MIN_VALUE;
for (int i = 0; i < ranks.length; i++) {
if (currentMax < ranks[i]) {
currentMax = ranks[i];
index = i;
}
}
return students[index];
}

public String lowestRank() {
int index = 0;
int currentMax = Integer.MAX_VALUE;
for (int i = 0; i < ranks.length; i++) {
if (currentMax > ranks[i]) {
currentMax = ranks[i];
index = i;
}
}
return students[index];
}
}
public class Rank {
public static void main(String[] args) {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
try {
String studentsInput = br.readLine();
String ranksInput = br.readLine();

String[] students = studentsInput.split(",");
String[] ranksArray =ranksInput.split(",");
int[] ranks = new int[ranksArray.length];
for(int i =0; i<ranksArray.length; i++) {
ranks[i] = Integer.parseInt(ranksArray[i]);
}
StudentRank s = new StudentRank(students, ranks);
System.out.println(s.highestRank()+","+s.lowestRank());
}
catch(IOException ioe){
System.out.println(ioe);
}
}
}
```

5.
```
import java.io.*;
import java.util.*;
import java.util.stream.*;

import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;

class Result {

    /*
     * Complete the 'getTotalExecutionTime' function below.
     *
     * The function is expected to return a LIST of INTEGERS.
     * The function accepts following parameters:
     *  1. INTEGER n
     *  2. LIST of STRING logs
     */

    public static List<Integer> getTotalExecutionTime(int n, List<String> logs) {
        int[] result = new int[n];
        Stack<int[]> stack = new Stack<>(); // [function_id, start_time]
        for (String log : logs) {
            String[] parts = log.split(":");
            int functionId = Integer.parseInt(parts[0]);
            String type = parts[1];
            int timestamp = Integer.parseInt(parts[2]);

            if (type.equals("start")) {
                stack.push(new int[]{functionId, timestamp});
            } else {
                int[] startInfo = stack.pop();
                int startId = startInfo[0];
                int startTime = startInfo[1];
                int duration = timestamp - startTime + 1;
                result[startId] += duration;

                if (!stack.isEmpty()) {
                    result[stack.peek()[0]] -= duration;
                }
            }
        }

        return Arrays.stream(result).boxed().collect(toList());
    }
}

public class Solution {
    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")));

        int n = Integer.parseInt(bufferedReader.readLine().trim());
        int logsCount = Integer.parseInt(bufferedReader.readLine().trim());

        List<String> logs = IntStream.range(0, logsCount).mapToObj(i -> {
            try {
                return bufferedReader.readLine();
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }
        }).collect(toList());

        List<Integer> result = Result.getTotalExecutionTime(n, logs);

        bufferedWriter.write(
                result.stream()
                        .map(Object::toString)
                    .collect(joining("\n"))
                    + "\n"
        );

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```

6. Grocery Receipt
****

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

class Grocery {
    String fruit;
    double price, total;

    Grocery(String fruit, double price, double total) {
        this.fruit = fruit;
        this.price = price;
        this.total = total;
    }
}

class Node {
    String fruit;
    int count;

    Node(String fruit, int count) {
        this.fruit = fruit;
        this.count = count;
    }
}

abstract class GroceryReceiptBase {
    private Map<String, Double> prices;
    private Map<String, Integer> discounts;

    public GroceryReceiptBase(Map<String, Double> prices, Map<String, Integer> discounts) {
        this.prices = prices;
        this.discounts = discounts;
    }

    public abstract List<Grocery> Calculate(List<Node> shoppingList);
    public Map<String, Double> getPrices() {
        return prices;
    }

    public Map<String, Integer> getDiscounts() {
        return discounts;
    }
}

class GroceryReceipt extends GroceryReceiptBase {

    public GroceryReceipt(Map<String, Double> prices, Map<String, Integer> discounts) {
        super(prices, discounts);
    }

    @Override
    public List<Grocery> Calculate(List<Node> shoppingList) {
        Map<String, Double> totalPricePerFruit = new HashMap<>();
        Map<String, Integer> totalQuantityPerFruit = new HashMap<>();

        for (Node item : shoppingList) {
            String fruit = item.fruit;
            double price = getPrices().getOrDefault(fruit, 0.0);
            int quantity = item.count;
            double discount = getDiscounts().getOrDefault(fruit, 0);
            double total = price * quantity * (1 - discount / 100.0);

            totalPricePerFruit.put(fruit, totalPricePerFruit.getOrDefault(fruit, 0.0) + total);
            totalQuantityPerFruit.put(fruit, totalQuantityPerFruit.getOrDefault(fruit, 0) + quantity);
        }

        List<Grocery> result = new ArrayList<>();
        for (Map.Entry<String, Double> entry : totalPricePerFruit.entrySet()) {
            String fruit = entry.getKey();
            double totalPrice = entry.getValue();
            int quantity = totalQuantityPerFruit.get(fruit);
            double price = getPrices().getOrDefault(fruit, 0.0);
            result.add(new Grocery(fruit, price, totalPrice));
        }
        return result;
    }
}

public class SolutionMain {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        Map<String, Double> prices = new HashMap<>();
        Map<String, Integer> discounts = new HashMap<>();
        List<Node> boughtItems = new ArrayList<>();

        System.out.println("Enter number of items and their prices:");
        int n = Integer.parseInt(reader.readLine().trim());
        for (int i = 0; i < n; i++) {
            String[] a = reader.readLine().trim().split(" ");
            prices.put(a[0], Double.parseDouble(a[1]));
        }

        System.out.println("Enter number of items and their discounts:");
        int m = Integer.parseInt(reader.readLine().trim());
        for (int i = 0; i < m; i++) {
            String[] a = reader.readLine().trim().split(" ");
            discounts.put(a[0], Integer.parseInt(a[1]));
        }

        System.out.println("Enter number of items bought:");
        int b = Integer.parseInt(reader.readLine().trim());
        for (int i = 0; i < b; i++) {
            String[] a = reader.readLine().trim().split(" ");
            boughtItems.add(new Node(a[0], Integer.parseInt(a[1])));
        }

        GroceryReceipt g = new GroceryReceipt(prices, discounts);
        List<Grocery> result = g.Calculate(boughtItems);

        for (Grocery x : result) {
            System.out.printf("%s %d %.1f%n", x.fruit, (int) x.price, x.total);
        }
    }
}
```