Input:

#Five types of command
performance [Date] [Hour] [NumSeats]
book [TimeStamp] [PerformanceDate] [Name] [Phone] [NumAttendees]
cancel [TimeStamp] [ReservationNum] [Name] [Phone]
pickup [TimeStamp] [ReservationNum] [Name] [Phone]
time [TimeStamp]

note1:
time [TimeStamp] means update time
input TimeStamps are in non-decreasing order

note2:
TimeStamp format is [Date],[Hour] e.g. 365,23
1 <= Date <= 365, value of Date will be Integer
0 <= Hour <= 23, value of Hour will be Integer
In performance command, 8 <= Hour <= 23

note3:
ReservationNum should be generated after each success book command.
ReservationNum start from 1, each generation increase by 1
e.g.
book … // Assume success, ReservationNum = 1
book … // Assume success, ReservationNum = 2
pickup ...
cancel …
(two hours prior, do cancel) ...
(performance start, do clear) ...
book … // Assume fail (not enough seats)
book … // Assume success, ReservationNum = 3

Output:

```
After read TimeStamp in each command:
    if performance.startTime - TimeStamp <= 2hr:
        // cancel all non-picked up reservation process in performance
        // customer cancelled reservation does not need to be cancelled again.
        // output all non-picked reservations you cancelled in "ReservationNum" order
        => Cancel #[ReservationNum]. name: [Name], tickets: [NumAttendees]

    if performance.startTime == TimeStamp:
        // clear all no-show reservation process in performance
        // no-show reservation include non-picked up and customer cancelled
        // output all no-show reservations you cleared in "ReservationNum" order
        => Clear #[ReservationNum]. name: [Name], tickets: [NumAttendees]

After book command:
    Success => Reserve #[ReservationNum]. name: [Name] tickets: [NumAttendees]
    Fail (not enough seats) => Book Fail. Not enough seats in requested performance.

After cancel command:
    Success => Cancel #[ReservationNum]. name: [Name], tickets: [NumAttendees]
    Fail (reservation non-exist) => Cancel fail. No such reservation.
    Fail (wrong ID) => Cancel fail. Identification check fail.
    Fail (already cancelled) => Cancel fail. Reservation already cancelled.
    Fail (already picked up) => Cancel fail. Reservation already picked up.

After pickup command:
    Success => Pickup #[ReservationNum]. name: [Name], tickets: [NumAttendees]
    Fail (reservation non-exist) => Pickup fail. No such reservation.
    Fail (wrong ID) => Pickup fail. Identification check fail.
    Fail (already cancelled) => Pickup fail. Reservation already cancelled.
    Fail (already picked up) => Pickup fail. Reservation already picked up.
```

note4:
The right side of "=>" means the output format

note5:
Before processing command, check the TimeStamp first. (cancel non-picked up, clear no-show)

note6:
After picking up a reservation, reservation still exist but can no longer be canceled or picked up.
After canceling a reservation, reservation still exist but can no longer be canceled or picked up.
After clearing a reservation, reservation does not exist.

note7:
wrong ID means that name or phone do not match the reservation

Comment:

There will be only one performance for each day, and customers will only book existing performance. The input order of all book, cancel and pickup command will follow the given timestamp of each command.

Please don’t ignore the periods in output.

Please implement your main function in Class Main.
We'll test your program through "java Main inputFile"
e.g java Main sampleInput

Do not read input from System.in or hard code input file, or your program won’t pass any test case.

Upload:

Please zip your source code and upload it.
The file name should be [studentID].zip. e.g. r06944012.zip
The folder structure should be:

```
unzip r06944012.zip
=> [dir] r06944012
=>       r06944012/Main.java
=>       r06944012/*.java (optional)
=>       r06944012/group_design.jpg (one for each group, soft copy or hard copy)
=>       r06944012/individual_dsgin.jpg (optional, soft copy or hard copy)
```

You won’t receive any point if you didn’t follow the directory structure or main class name or compressed format!

#**SampleInput:**

```
performance 2 23 10
performance 3 8 5
book 1,1 2 CustomerA 0911111111 5
book 1,3 2 CustomerB 0922222222 3
book 1,5 2 CustomerC 0933333333 1
book 1,6 2 CustomerD 0944444444 4
pickup 1,7 2 CustomerA 0911111111
pickup 1,8 1 CustomerA 0911111111
cancel 1,9 2 CustomerB 0922222222
cancel 1,10 4 CustomerC 0933333333
time 2,21
book 2,23 3 CustomerE 0955555555 1
```

#**SampleOutput:**

```
Reserve #1. name: CustomerA, tickets: 5
Reserve #2. name: CustomerB, tickets: 3
Reserve #3. name: CustomerC, tickets: 1
Book fail. Not enough seats in requested performance.
Pickup fail. Identification check fail.
Pickup #1. name: CustomerA, tickets: 5
Cancel #2. name: CustomerB, tickets: 3
Cancel fail. No such reservation.
Cancel #3. name: CustomerC, tickets: 1
Clear #2. name: CustomerB, tickets: 3
Clear #3. name: CustomerC, tickets: 1
Reserve #4. name: CustomerE, tickets: 1
```