Assumptions considered-

Grid details –

Dimensions – 21 X 21

X axis range: [-10, 10]

Y axis range: [-10, 10]

- Each coordinate can take a maximum of one event.
- The code written can only be expanded to higher grid sizes provided the grid is centered at origin and the dimensions are symmetric
  Eg – It can work for dimensions like 51 X 71 with X axis in range of [-25, 25] and Y axis in range of [-35, 35]. However the code needs to be tweaked to work with dimensions like 52 X 63, 51 X 64 (Even dimensions not centered at origin) or when the dimensions are not symmetric with respect to origin i.e. when x lies in range [-24, 26] etc.
- The user location should be present inside the grid. If the user is outside the grid the current state of code will not work and need some further tweaking.
- Another shortcoming of the program is that I'm randomly choosing a grid location and check if that location already has an event. If it doesn't, then I go ahead and add an event. This could be a bottle neck for higher grid sizes as creating such numbers which don't have an event could require higher complexities as the coordinates chosen are random and there is a probability that a coordinate may be chosen after a high number of trials which could lead to time complexity issues

Ways to overcome the shortcomings –

1) How to modify the code if a location can handle multiple events?

A) In my current code I have filled the grid with just one event. If I'm allowed to add multiple events I can keep a list of events in the grid instead of a single event. I could use a syntax something like the below to handle such lists.

```
List<Event>[][] eventList = (List<Event>[][]) new List[gridLength][gridWidth];
```

2) How to scale up my program to handle bigger grid sizes?

A) Like I have mentioned earlier my current code is capable of handling higher grid sizes as long as the grid is symmetric centering at origin. In case if this is not the case, I would need to use a **HashMap<Location, List<Event>>** so that I can store all the locations and the list of events in that location as key value pairs. This way while searching for the closest event from the user location I could check for the nearest coordinates by increasing the radius/Manhattan distance in the increments of 1 and check if such a location exists in the Hashmap with an event. If so I add the event to the closest event list for the user. Using a hashmap will work for cases when the user is not present in the grid (Of course with time complexity issues as we don't know how far the user is from the grid and we only increment the coordinates in the steps of 1. We can overcome this case by storing the boundary coordinates of the grid and modify the code such that the user starts looking from the boundaries of the grid and go inside)

P.S – In the above questions I have only considered the application as a single threaded application with it being hosted on a single node/server. In case if this app needs to scaled up as a multiple threaded application, then several use cases come into picture requiring the program to use completely different data structures like priority queues and maintaining back up nodes to handle node failures etc. to name a few. I have deliberately omitted all such cases above as cases involving such scenarios are aplenty and would be a topic of much larger discussion.