1. Scenario
    1) Entity:
        (1) (x, y) co-ordinate, where x∈[-10,10] and y∈[-10,10]
        (2) Event, in format of (id, x, y, state), where state indicates whether this event have tickets or not
        (3) Tickets, in format of (id, price)
    2) Relationship
        (1) One co-ordinate have 0 or 1 event
        (2) One event have 0 or more tickets
2. Assumptions
    1) If one ticket occur in this program, we consider this ticket as available. In other words, let's assume a situation that one event have two kinds of tickets: the first one have 5 tickets left, and each cost $100; the other kind have 0 tickets left and each cost $150. In this case, this program will only consider the $100 tickets and skip the $150 tickets. Also, based on instruction, there is no need to consider the number of tickets.
    2) Since each event could has zero or more tickets, it's possible that closer events has no tickets left. In this situation, this program will show these events, and also mention there is no tickets left instead of showing the cheapest ticket.
    3) Distance still could be the same even though the two events have different locations, and I assume customers are more interested in cheaper event. Therefore, if there are several events, which have exactly the same distance from the given location, this program will arrange the event which have cheaper tickets ahead of the others.
    4) Since seed data should be randomly generated, I assume no more than 5 kinds of tickets will are left for a specific event, and its amount is randomly picked.
    5) I assume the price is not lower than $30 and not higher than $200. Also, if you want to use the initial requirement of price scale, you will find the code as a code annotation at Line 38 of 'Functions.java' file
3. Questions / Answers:
    1) How might you change your program if you needed to support multiple events at the same location?

        Since assumption 3 already considered the situation that several events have the same distance from the input location and the program will show the cheapest alternatives as result, therefore I only need to remove the Vector (line 22, 'Functions.java' file) and the same-location events checking code at about line 34, 'Functions.java' file to enable multiple events at the same location.

        The code still choose to show the cheapest events if some of the alternatives, no matter they have the same location or not, have the same distance from the input location.
    2) How would you change your program if you were working with a much larger world size?

        In terms of larger world size, I was thinking about two aspects:
        (1) Larger frame of axes, like the real world.

            In this case, we have to use a new way to measure distance, e.g. Euclidean distance. Also, we have to solve issues such as crossing prime meridian and coordinate system distortion.
        (2) Larger size of events.

If we thousands of events and millions of users, it will be very time consuming and computing power consuming to go through all the events for every single request. My current idea is to apply K-Means algorithms to achieve pruning. First of all, cluster the events into several clusters based on their latitude and longitude. For each cluster, there will be a centroid which is the central point of all the event locations among that cluster.

Then, when user send a request to search for closest events, compare the input location with every centroids of different clusters, choose the closest cluster and compare the events' locations among the chosen cluster with input location to provide the result.