Grid

***gridLength*:** length of the grid (Here I chose it as 21 since the range is from -10 to 10)

***gridWidth*:** width of the grid (Here I chose it as 21 since the range is from -10 to 10)

grid[][]: A 2-D event array to store all the events at their respective locations based on the offset(since negative indexed are not allowed in arrays) with the grid

hasEvent[][]: A Boolean array to check if a location in the grid has an event already

generateRandomSeeds(count):

This method generates takes in an input count which represents the number of events to be created in the grid. We randomly generate x-coordinate and y-coordinate and see it the randomly generated coordinates already has an event. If not we go ahead and create an event for that coordinates by creating a few category of tickets (here I have taken this count as 10) and their count. I have taken an assumption here that the ticket price lies in a range of [$1, $100] and ticket count for each category lies in a range of [0,100]. We then sum up all the category ticket count to a total sum and assign it to the event as ***totalTicketCount.*** We now mark this coordinate as true for an event. We also reduce the ***maxEvents*** by 1 to keep track of the number of coordinates still left for adding an event. We exit out of the loop once we assign the given count of events in the grid.

getClosestLocations(location):

This method contains the core logic of checking the closest available events to the user location based on Manhattan distance between the two points. I have used a list to store all the closest events and return it to the user. The idea I used for this method is the concept of radius/Manhattan distance and incrementing it in the steps of 1 from the user location to check if there are any coordinates with the current radius having an event. I start off with a radius of 0 which is the user location itself and then increment it in steps of 1 (Manhattan distance). The coordinates count increment in steps of 4 for an increment of step 1 in radius. i.e. for radius 1, I need to check for 4 points, for radius 2 , I need to check for 8 points, for radius 3 I need to check for 12 points and so on. I have implemented a logic which takes care of covering all such points at each increment in radius. While checking such points, if I encounter a point with an event I increment my counter by 1 and add the event in that coordinates to my closest events list. Once the counter reaches 5, I break the loop and return the closest event list.

getTotalEvents(): public method to available to other classes to access the total event count in the grid.

incrementEventSize(): A subroutine for the method ***generateRandomSeeds()*** for incrementing the event size.

Location

x: x coordinate of the location

y: y coordinate of the location

Location(x,y): Constructor of the class location when invoked with both x,y coordinates

getXCoordinate(): Returns the x coordinate of the location

getYCoordinate(): Returns the y coordinate of the location

printLocationDetails(): Prints both x and y coordinates in the form (x,y)

Event

eventId: A unique id for the identification of an event

ticketPrices[]: An array with the ticket prices of all available categories(Here I have taken a size 10)

ticketCounts[]: An array with the ticket counts of all available categories corresponding to the ticket prices

Event(eventId): Constructor for this class when invoked with an eventId.

setLocation(location): Assigns a location to the given event.

getLocation(): Retrieves the location of the event

getEventID(eventId): Returns the eventId of the event after casting it to string and appending leading zeroes depending on the number of digits in the ID.

getCheapestTicketIndex(): Returns the index from the array ***ticketPrices[]*** which has the cheapest ticket price. We can use this index to get the ticket count of this category from the array ***ticketCounts[]***

displayEventDetails(): Display all the details of the event

User

Location: Location of the user

setLocation(x,y): Sets the location of the user for a given coordinates of x and y

getLocation(): Returns the location of the user

Driver

- This is the class which contains the main method.
- In the main method, we initially instantiate an object of the class Grid. We then take input from the user asking him for an event count to generate events randomly in the grid.
- Once all the events are seeded, we ask the user to enter his/her location coordinated as comma separated values.
- The program ensures that the user enters a valid input coordinates prompting them to enter valid coordinates whenever they enter invalid entries
- Once a valid coordinates are entered, the program checks for all the closest events based on the minimum Manhattan distance between the user coordinates and the event coordinates. To get this the program invokes the method ***getClosestLocations(location)*** from the class **Grid**
- Once we get the closest event list we display all the event details which consists of the Event Id, the event coordinates, Total number of tickets available for the event, cheapest ticket price for the event, number of cheapest tickets available and the Manhattan distance between the two coordinates.
- The program exits