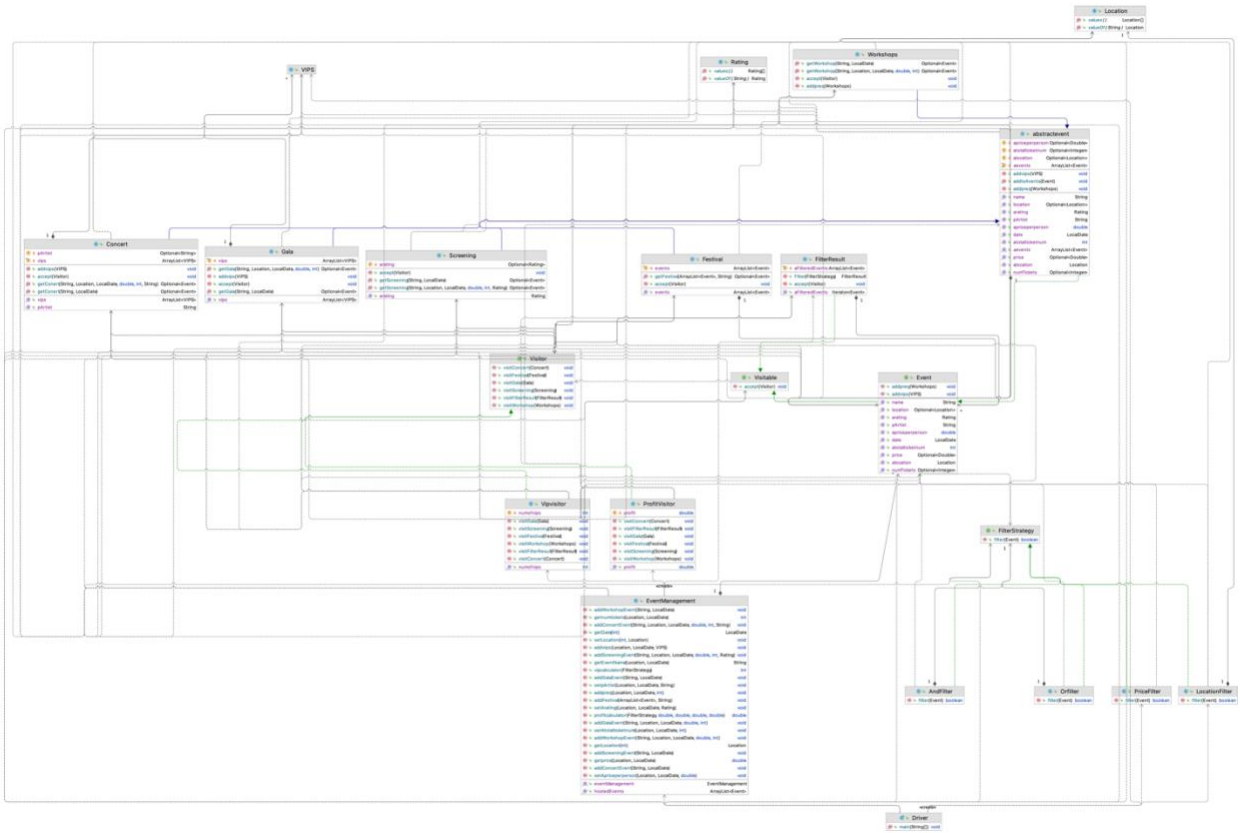
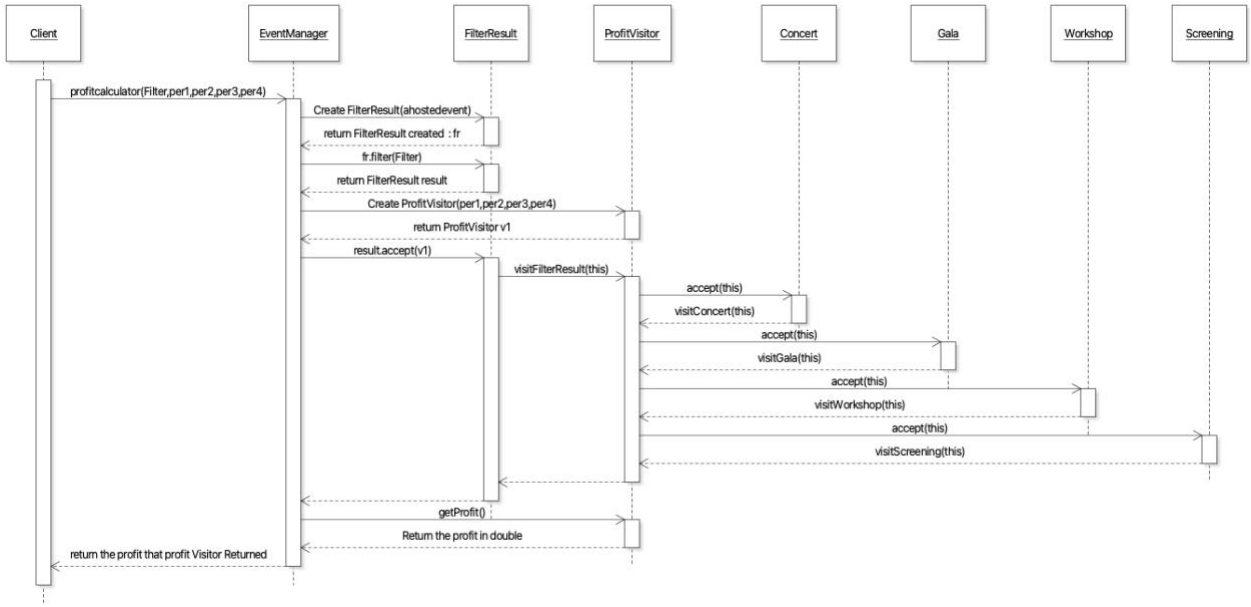
Class diagram:



Sequence diagram:



Readme:
First, for different events I used the concept of Inheritance since there are a lot of share fields and methods between them. Then for Festivals I used the Composition design pattern where I am allowed to treat a bunch of events like one regular one. After, In order to create several ways of Filtering I used the strategy design pattern and designed the two required strategy namely location and price Filter. Then in order to combine two filters I also created an and filter and a or filter so the client can combine those Filters and create a big filter with several aspect to it. After that, for calculating the profit and number of VIPs I used visitor design pattern and explore polymorphism for the method accept such that different kind of event will map the visitor to different visit where they can calculate it's profit from there which the functionality coupling was reduced to minimum. To provide the user to create a Coming soon event I used the idea of optional for each concrete event. On ED the TA explicitly said we can assume that Festival will not contain all coming soon events, so I take full advantage of it and reduce the complexity of code by making such assumption. I also overload each concrete event's constructor so it allows them to create a concrete event with some field missing. Last but not least, in order to stop Bob creating duplicate event with same date and location flyweight design pattern were used, since we are not doing flyweight on each individual concrete design pattern, we have to store all the event instances in an Array list within the abstract event. So that all event is able to access it and add to it. Lastly, I update the Event management class so the user only needs to access the event management and the filter Strategy Constructor to do everything Eventbrite is designated to do. I also make Event management a singleton so that when filtering it can filter through all the events that has ever been created. I tested pretty thoroughly the code I wrote through unit test and test the event management class through driver (aka. the functionality test). The coverage for most classes is 100% which makes me confident that these code are correct and the design is on point. In conclusion: I used in total of 5 design patterns: Flyweight, Singleton, Composition, Visitor, Strategy. Below is a photo of coverage test.

100% classes, 90% lines covered in package 'src'

Element	Class, %	Method, %	Line, %
abstractevent	100% (1/1)	100% (17/17)	100% (39/39)
AndFilter	100% (1/1)	100% (2/2)	100% (4/4)
Concert	100% (1/1)	100% (9/9)	100% (28/28)
Driver	100% (1/1)	100% (1/1)	79% (85/107)
Event	100% (0/0)	100% (0/0)	100% (0/0)
EventManagement	100% (1/1)	100% (27/27)	83% (119/142)
Festival	100% (1/1)	100% (5/5)	100% (35/35)
FilterResult	100% (1/1)	100% (4/4)	100% (11/11)
FilterStrategy	100% (0/0)	100% (0/0)	100% (0/0)
Gala	100% (1/1)	100% (8/8)	100% (24/24)
Location	100% (1/1)	100% (2/2)	100% (2/2)
LocationFilter	100% (1/1)	100% (2/2)	100% (6/6)
Orfilter	100% (1/1)	100% (2/2)	100% (4/4)
PriceFilter	100% (1/1)	100% (3/3)	100% (9/9)
ProfitVisitor	100% (1/1)	100% (9/9)	100% (25/25)
Rating	100% (1/1)	100% (2/2)	100% (2/2)
Screening	100% (1/1)	100% (7/7)	100% (25/25)
VIPS	100% (1/1)	100% (0/0)	100% (1/1)
Vipvisitor	100% (1/1)	100% (8/8)	100% (12/12)
Visitable	100% (0/0)	100% (0/0)	100% (0/0)
Visitor	100% (0/0)	100% (0/0)	100% (0/0)
Workshops	100% (1/1)	100% (7/7)	100% (23/23)