Q1/Q2:

I began by creating an AbstractEvents class that extends an Events interface. The Concert, Gala, Workshop, and Screening events extend this class as they all share the same features. I added additional features to the subclasses classes if they had additional features. The Festival event implemented the Event interface separately (it's acts as a composite class).

Q3:

For filtering, I used the strategy design pattern. Bob is able to create any new class which represents a filter (implement the filter interface), and simply input this filter into the FilterResult constructor and the applyFilter method gets overridden depending on the filter he passes through. This makes it easy to add strategies (filters) in the future.

For filtering by price, the client creates a filter object and must indicate the min and max price to be in the filtered list. I also sort the prices from lowest to max price using an anonymous comparator.

For filtering by location, the client creates a filter object and must indicate a list of locations to be filtered (this can be one or many locations). I sort the locations in the filtered list if they include more than 1 location with an anonymous comparator.

For filtering by location and price, I simply created a multipleFilters class which takes in a list of filters and applies the filters in the order of the list. The client can include more filters/different filters using this class. The client can also create more strategies (filters) by simply creating a class and implementing the applyFilter method. They do not have to change anything in the FilterObject class.

Q4:

I implemented the Visitor Design pattern. I added acceptVisitor(Visitor) to the Event interface, which is overridden in every Event class. The acceptVisitor(Visitor) method calls the visit*Event*(*Event)* method in the ExpectedProfitVisitor class which calculates the expected profit. (Note: in the Festival class, I loop through the list of events and call visit*Event*(*Event)* on each individual Event in the festival). The percentages are hard-coded in the ExpectedProfitVisitor class which was indicated on an ED post.

I created another class NumberOfVIPsVisitor which follows the same logic as above. When visit*Event*(*Event)* is called in the acceptVisitor(Visitor) methods in each Event class, the NumberOfVIPsVisitor methods will return the number of VIPs attending each event.

Q5:

My initial idea was to use Optional<> for the coming soon events, but this created a problem in future calculations and would have made my code messy if I had to constantly use isPresent() checks. Therefore, I chose to set the fields to a specific value in the case where Location, Price per Ticket, and number of tickets was not provided. I added ComingSoon to my Location enum, and I set the price and number of tickets to 0. This makes it easy to understand the code and it doesn't affect the future calculations of events. My program doesn't break when calling the getters because the fields are set to a specific values and will not return null. I added an additional constructor to each Event class for the ComingSoon events.

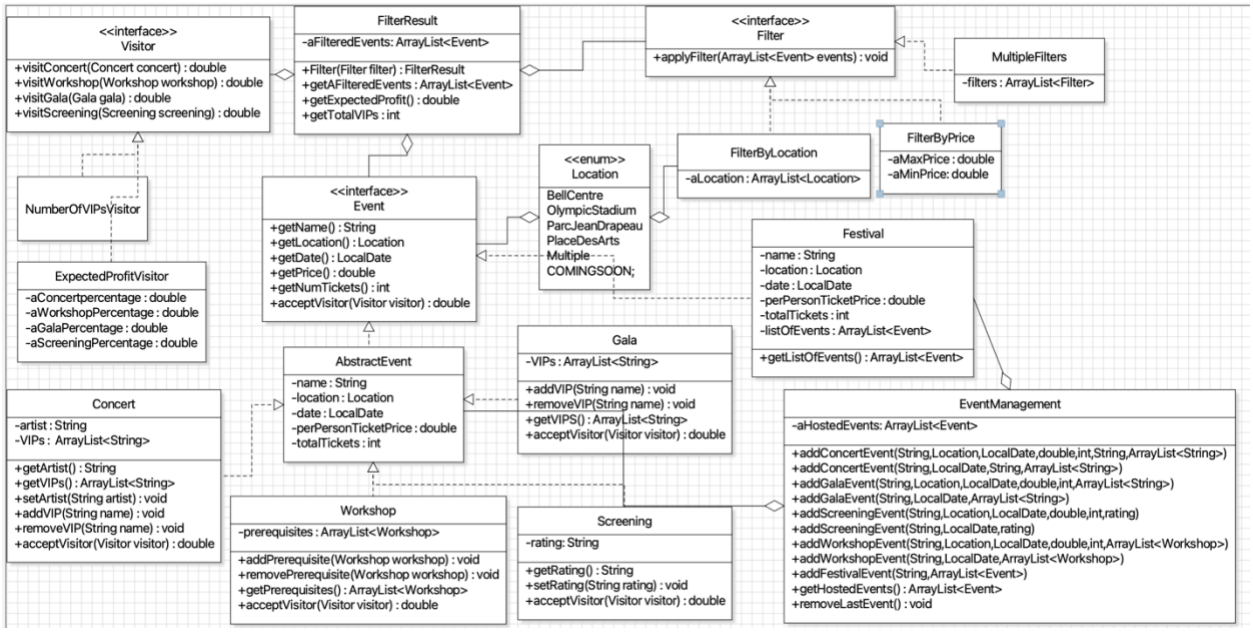
Q6:

Bob creates events in the EventManagement class, therefore, I verify that there are no duplicate location/dates in the current list before adding a new event.

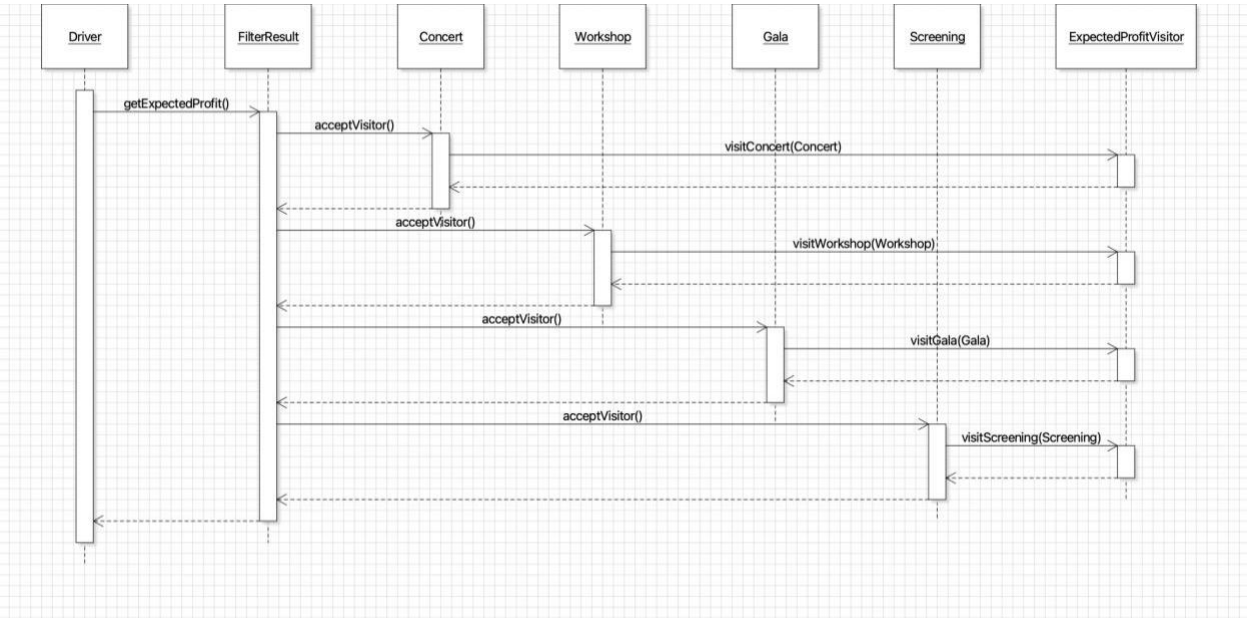
Q7:

In the EventManagement class, I created two methods for the creation of each event: one with all required fields and one for coming soon events. I chose to throw an exception if the location and time of an event already exists because this is a rare occurrence and I want an error message to be displayed to the user.

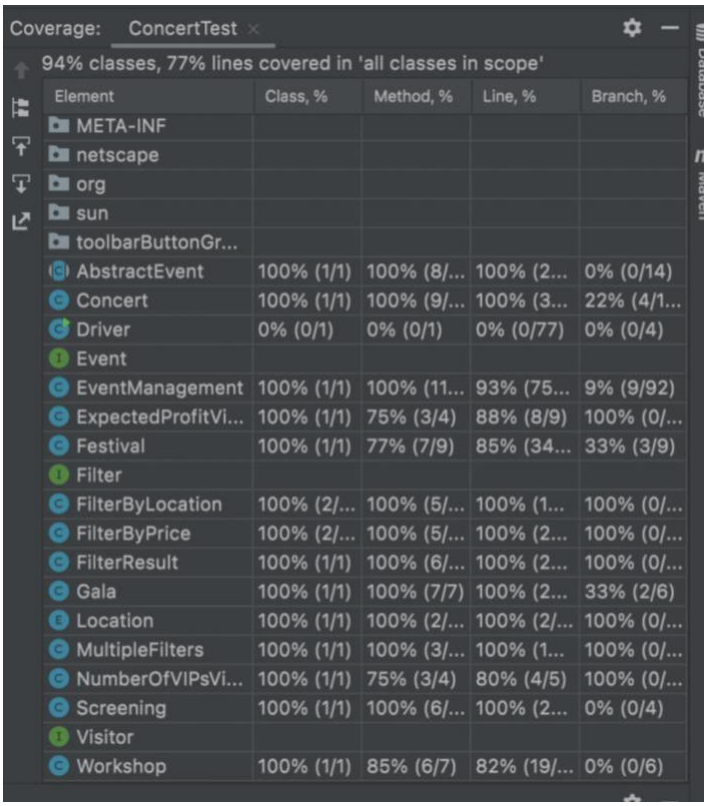
Class Diagram:



Sequence Diagram:



Test Coverage:



Coverage: ConcertTest

94% classes, 77% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
META-INF				
netscape				
org				
sun				
toolbarButtonGr...				
AbstractEvent	100% (1/1)	100% (8/...	100% (2...	0% (0/14)
Concert	100% (1/1)	100% (9/...	100% (3...	22% (4/1...
Driver	0% (0/1)	0% (0/1)	0% (0/77)	0% (0/4)
Event				
EventManagement	100% (1/1)	100% (11...	93% (75...	9% (9/92)
ExpectedProfitVi...	100% (1/1)	75% (3/4)	88% (8/9)	100% (0/...
Festival	100% (1/1)	77% (7/9)	85% (34...	33% (3/9)
Filter				
FilterByLocation	100% (2/...	100% (5/...	100% (1...	100% (0/...
FilterByPrice	100% (2/...	100% (5/...	100% (2...	100% (0/...
FilterResult	100% (1/1)	100% (6/...	100% (2...	100% (0/...
Gala	100% (1/1)	100% (7/7)	100% (2...	33% (2/6)
Location	100% (1/1)	100% (2/...	100% (2/...	100% (0/...
MultipleFilters	100% (1/1)	100% (3/...	100% (1...	100% (0/...
NumberOfVIPsVi...	100% (1/1)	75% (3/4)	80% (4/5)	100% (0/...
Screening	100% (1/1)	100% (6/...	100% (2...	0% (0/4)
Visitor				
Workshop	100% (1/1)	85% (6/7)	82% (19/...	0% (0/6)