# Assignment 2 - Design Choices

*Nisarag Bhatt*

## 1 Design Explanation

- For this project, we had to implement a server (TBS) that takes care of managing the tickets for performances at the theatre.
- After reading the brief, I gathered that the booking system must ultimately take care of **Performances** which take place in a **Theatre** and have **Acts**, and **Artists** perform these acts. The booking system must also issue **Tickets** accordingly to the theatre for every unique performance. It also made sense for there to be a location for the server to access information like what Artists are currently in the system at the moment and how many Acts there are and so forth. Due to this, it would be a good idea for a **Database** to exist where this information is stored.
- Due to the reasoning above, the following classes were then produced
  - **Theatre** - Since there will be many unique theatres, each with different dimension, area, it makes sense for there to be a theatre class.
  - **Artist** - Since in a theatre there will be a lot of artists performing, each artist will also be unique so it makes sense for the booking system to know which artist to handle at any point in time. Hence an artist class would be a good since we can create many artist objects.
  - **Act** - In a theatre, there will be many acts being performed by artists, each act will also be unique from another. Due to this it makes sense for a booking system to handle all these unique acts and hence a class would be appropriate.
  - **Performance** - In a theatre, there will be performances where people will go to which consist of acts performed by artists. Each performance will be at a certain time and hence will be unique to one another and therefore having a class for a performance would be a good idea.
  - **Ticket** - In every performance, people will buy tickets and therefore the booking system must manage all these tickets. Since each ticket is unique and related to a performance it makes sense for there to be a ticket class.
  - **Database** - When the server is instantiated, it needs a place to store all this data about the classes above. Therefore it would be a good idea to have a single database for each server since the server will always be querying the database for vital information to make sure the server is running smoothly. The database will also have utility methods which will make the job of the server much easier (for example the server can ask for a theatre relating to a certain performance, it doesn't really make sense for a server to do that since the database will be storing most of the information - it also provides a level of abstraction since the server just wants a theatre

relating to a performance, the server does not care how the database got it).

- From the above justification for each class, I then went and wrote my project with all these design choices in mind and tried to minimize the amount of work the server has to do since it is good that the server constantly talks to the database (for information, and storing information) and also talks to the other classes (Theatre, Artist, Act, Performance, Ticket) making sure that when producing an object of any of those types there are no errors and if there is - it gives feedback in the best way which is user friendly.
- One another thing I would like to talk about is why my (Theatre, Artist, Act, Performance, Ticket) classes all contain a private static field (ID-Counter), this justification for this is that every time we instantiate an object of any of those classes we want to assign that object an ID relating to its class. Since I did this by simply having a counter and appending characters to that object relating to its class.
  - For example consider the Ticket class, every time the server issues a ticket - the ticket gets a unique ID. The first ticket will have an ID of "$TI0$", however when the server wants to issue the next ticket - how does that ticket know the last create ID number? It doesn't (if the counter was private), one way of getting around this is making the variable static however the server doesn't need to know the counter number at all times so I made the counter variable have the **private static** modifier.
  - This is good because the ID Counter variables can be used share the counter across all instances of that particular class.