**GROUP: 27**

Unnayan Amitabh

Justinas Lingys

[Farhad Bin Siddique](https://www.facebook.com/farhad.b.siddique?fref=ufi)

#### [Anurag Sahoo](https://www.facebook.com/profile.php?id=100008204951226)

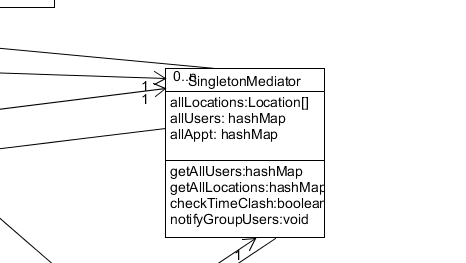
**ARCHITECTURE DOCUMENTATION**

**Name of the Pattern:** Singleton Design Pattern

**Motivation:** Without the singleton design pattern it would be extremely difficult to manage multiple instances of the same class which manages the same resource. For example in the case of our calendar project, the entire GUI package named as “hkust.cse.calender.gui” uses this pattern and prevents us from making multiple instances of classes within the package. Without Singleton class, if a class has inheritance to derive from, it would be very difficult, it would also be hard to specify creation logic.

**Solution:** Using the singleton architecture, we create a private constructor of the singleton class that prevents us from making instances of the class nut at the same time helps us to access a static property or function of the class for a preconfigured instance. In our implementation of the Singleton Class we use a single thread to access the properties of instances. From the point of view of our code then, the singleton class helps us manage all the functions from creating appointments, to group appointments, managing locations, deleting appointments to managing time conflicts etc

**Class Diagram:**

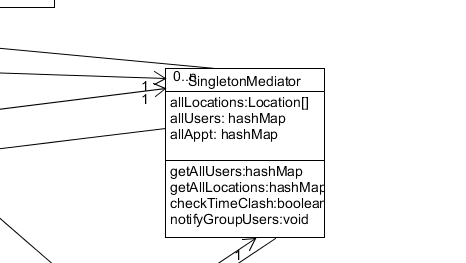
****

**Name of the Pattern:** Mediator Design Pattern

**Motivation:** Without the Mediator Class Pattern, it would be very inconvenient for us to get the classes in our code to interact efficiently. For example in our code if we were implementing a group event and we needed to check for appointment conflicts and location conflicts, separate classes and functions would have to be concurrently called and in order to manage many to many relationships, it would be very hard to see which rules reside within which class.

**Solution:** By creating the Mediator Class, we facilitate the interaction between classes by creating an intermediary path for the classes to interact. It functions like the brain of the program wherein all the functions that are invoked first interact with the intermediary mediator class and the mediator class interacts with the target classes. Mediator class helps us in maintaining our code by providing an interacting terminal between several functions.

**Class Diagram:**

****

**Name of new application that could improve our Design:** Observer Deign Pattern

**Motivation:** Without the Observer Design Pattern, when information in classes is updated or changed the observers can be updated with the changes. For Example in the case of changing appointment scheduling in the calendar project, this design pattern could be useful in implementing the update feature.

**Solution:** In case of group appointments, when the administrator changes the appointment time or date using the observer class design pattern can help in creating the update function that notifies members of the group about the necessary change.

**Final Class Diagram:**

