Façade design pattern:

- We have a CalendarDataFacade class that is responsible for loading and saving data to a Json file, in order to avoid the class becoming too long and complex. We created two new classes, one responsible to load data from the file, and the other responsible to save data to the file. The façade class holds an instance of each of these two classes, and calls their methods; thus masking complex code.

- The classes involved were: CalendarDataFacade (the façade), CalendarDataLoader (responsible for loading data), and CalendarDataSaver (responsible for saving data).

-CalendarManager can also be thought of as a facade in the bigger picture, it delegates its task to CalendarDataFacade, UserManager, and OverallManager holds an instance of each of these three classes .

Dependencies:

Dependencies are all as they should be, things that are less likely to change don't depend on the ones that are more likely, just like the diagram in clean architecture.
For example, Entities(Events, Memos, Alerts..) don't depend on Use case classes(Eventmanager, AlertManger,...). This goes all the way up to the interface. Refer to the hierarchy diagram in the Extras folder for a visual representation. (Note to fit everything only some of the classes at the each level have been put on the diagram)

SOLID:

Solid principles are also in play, as almost every class has a single responsibility, thus only has one reason to change.