Peer Review of Group 1 (New Five) by Group 15 (Alpha)

Do the design and implementation follow design principles?
High cohesion, low coupling: Somewhat. The model module/package only has internal dependencies and there are not too many dependencies between packages.

Single Responsibility: Many of the view classes both have the responsibility of instantiating visual components and handling interaction.

Liskov Substitution: No model classes properly follow the contract of the Object class since they do not properly implement hashCode and equals.

Dependency Inversion: The observer pattern, which is used, reverses the dependency between whoever is observing and who is being observed.

Polymorphism: Not directly but sort of when extending the JavaFX containers in the controller classes. (For example, extending the AnchorPane and using JavaFX and standard library methods). Otherwise no.

Does the project use a consistent coding style?
The code uses a mix of FXML documents and instantiated JavaFX components which in some cases work well but in many places make it messy. The different tabs are, for example, loaded using separate methods and controller classes. This might be a good practice if the different tabs contained a lot, but currently, they are mostly empty and it seems that if they were properly implemented most likely would not require this intense separation of resources. Otherwise, the code consistently uses the same syntax style in all classes.

Is the code reusable?
Many of the views are "instantiated" in their own Java classes. They could have been instantiated using a general method for loading FXML documents instead making the code much more reusable. The model part of the code is fairly reusable since it is self-contained. However, the use of singletons and instances of components which a class relies on being instantiated inside the class' constructor limits the extensibility of many classes. Dependency injection would be preferred.

Is it easy to maintain?
Hard to say as not much functionality is implemented as of yet. Given the extensive amount of model related classes there is clearly a future perspective on the implementation of the application.

Can we easily add/remove functionality?
It looks fairly easy to add more functionality to the program. New screens/views should be fairly easy to add and other parts as well.

Are design patterns used?
The observer pattern is used in the Admin class. The implementation of the pattern seems correct. However, we are not sure about the correctness of how it is used. In the notifyObserver method in the Admin class (the observer), observers "to be removed" are removed, then the remaining observers are notified, and *after* observers "to be added" are added. This seems a little strange, but there may be a reason for it.

The Model View Controller (MVC) design pattern is also used. However, many classes that should be controllers to the FXML documents, the view, (FXML is the markup language of JavaFX) are not located in the controller package but rather together with the views. For example, the view StartPage.fxml's controller is in View/StartPage.java. That is, it is assigned correctly but not located in the proper structure. Many of the "controller" files act as both the view and the controller. There is a lack of separation between the two (should use high cohesion and low coupling).

In the controller package there are two controllers, the AdminController and the EmployeeController. These are more overarching authority-specific interactions with the application so should be located in another package.

The singleton pattern is used for multiple classes, like Admin, OurCalender and CertificateHandler. This should be avoided since it makes testing harder and couples every class which uses the singletons to the singleton class. It would have been ok to use if the singleton object represented, for example, a single value object or a system resource that there was only one of, preferably abstracted behind an interface. Instead of singletons, dependency injection would be preferred.

Is the code documented?
The code contains hardly any comments explaining the implementation. A majority of the model classes have some methods commented through Javadoc. Classes in the Controller and View packages have no comments for documentation.

Are proper names used?
It is common practice to attach a "Controller" at the end of a controller class name tied to a FXML document, e.g. StartPage.fxml would have the corresponding StartPageController. In this case they are named the same (except for the different file types). The AdminController and EmployeeController are correctly named if they are to be considered controllers.

In a couple of model classes variables are named with uppercase letters even if they are not constants. They should be lower camel case like all other variables. Similarly, all test classes should be named with camel case like all other classes and all package names should be lowercase.

Is the design modular? Are there any unnecessary dependencies?
The project's classes are split up into three packages: Model, View and Controller. This provides a clear separation between different responsibilities according to the MVC pattern. However, further separation into smaller modules/packages grouped by feature would increase the cohesion of each module and allow for package-private classes and members with a more narrow scope. The latter would also decrease the risk of any accidental unwanted dependencies

Does the code use proper abstractions?
There are two interfaces, Observable and Observer. Otherwise, no abstraction is found.

Is the code well tested?
The Model package has 48% statement test coverage. The other two packages, View and Controller, have no tests. The documentation specifies that all implemented classes have tests. They are quite rudimentary, for example, the add/delete employee test only adds a hard-coded employee to a list and then deletes it from the list. In the user interface that functionality does not exist.

Are there any security problems, are there any performance issues?
We would have thought that there would be authentication problems due to the impression that all employees would have access to the application. But we can see in the description that it is explained that only the scheduler would have access to the application and that would make it more secure.

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

It has an MVC structure from looking at the different packages but, as said previously, the model and the view classes are only somewhat correctly specified, and the controller classes are bundled together with the views. These should ideally be separated. Many of the views (FXML documents) do not have an assigned controller class to them, as is common good practice for JavaFX projects.

The model classes are clearly isolated from the rest of the project's classes. No outward dependencies exist from the model to other packages.

Can the design or code be improved? Are there better solutions?

Project setup

The project does have a POM file, but there are no dependencies in it and no configuration of Java version, and does not follow the standard Maven directory structure, so it is not really a Maven project. All external libraries (The JavaFX) are handled using local libraries (VM-options). External users could therefore not just "plug and play" but had to assign the correct library file path to the project. There is no SDK assignment which made the program default to Java 5 which didn't exist and therefore threw an error. The dependencies should be handled using Maven (the POM) so that it can run regardless of the libraries on the personal devices.

Making the application more modular

A good example would be to, for example, use a borderpane as the mainlayout and load different fxml documents into its center using a standardized method accepting a string for the location of the fxml file considered. (see second video playlist for examples). Below are two tips for YouTube channels on how to use Maven to build a JavaFX project and the other is a good playlist for how to implement JavaFX using FXML files and controllers.

- Easiest JavaFX Setup, IntelliJ and Maven (2020)
  https://www.youtube.com/watch?v=Ri6No63fl-A&ab_channel=ByteSmyth
- JavaFX with Scene Builder : Lecture 1: Create our Main Layout FXML and load it In our Main Class
  https://www.youtube.com/watch?v=q5A-qW2eGKs&ab_channel=SoftwareDevelopmentTutorials

Domain model

The arrows in the domain model should be more easily distinguishable from each other. Also take a look so that the arrows actually reflect how it is implemented in the code.

References

There are no references in the text but there is a bibliography.
The bibliography should have actual references to the Maven and JavaFX (OpenJFX) sites. Not just what they are called.