CSCI 5448 - OBJECT ORIENTED ANALYSIS AND DESIGN
PROJECT REPORT, SPRING 2017

Title: EEZON - An Interactive EE Inventory Management
Team: Sairam Udaya Janardhana Muttavarapu, Chinmay Shah, Rahul Yamasani, Sharath Reddy Vontari

1. Features that were implemented

ID	TITLE
UR-02	As a registered user, I can login to the system
UR-14	As a user, I can view the check in and check out details of my kits in the system
UR-15	As a user, I can view kit details course wise
UR-19	As a user, I can view the penalties incurred for missing checkIN deadlines and damage/lost kits per my role
UR-16	As a professor, Admin, TA, I can update check in and check out details of course specific kits in the system
UR-17	As a student, I can request unavailable items
UR-20	As a Professor, Admin, I can override the penalties incurred for students
UR-21	As a student, I can request for checkIN extension
BR-01	Must be Implemented in Java
BR-02	Must be a Standalone Project
BR-03	All students/professors can register only using @colorado.edu
BR-04	All students/professors can login only using @colorado.edu
FR-04	The system should automatically record the date and time of Log in/ Log out
NFR: 03	System should be on 24*7 for highest grade of performance

2. Features that were not implemented from Part 2

ID	TITLE
UR-01	As a student/professor, I can signup to the system based on my role
UR-03	As a registered user, I can request for a new password when I forget my old password
UR-04	As the admin, I can add courses to the system
UR-05	As the admin, I can remove courses from the system
UR-06	As the admin, I can approve role of existing user as a Professor
UR-07	As a Student, I can request to enroll for a course
UR-08	As the Professor, TA and Admin, I can approve role of user as a student for a specific course
UR-09	As the Professor/Admin, I can add/remove TA/ (Professor & TA) into a specific course
UR-10	As a professor, Admin, I can add a new kit type for a course
UR-11	As a professor/admin, I can remove a kit type from the course
UR-12	As a Professor/Admin/TA, I can add serial numbers to the kits for a course
UR-13	As a Professor/Admin/TA, I can remove serial numbers of kits from the course
UR-18	As a professor/admin, I can approve requests for unavailable items
UR-23	As Professor and Admin, I can view Graphical Report of checkin/checkout kits to understand the popularity of a certain kit in the course
UR-24	As Professor and Admin, I can view Graphical Report of damaged kits that helps in understanding the status of kits
FR-01	System will email the auto generated password to the User when he requests for a new password.

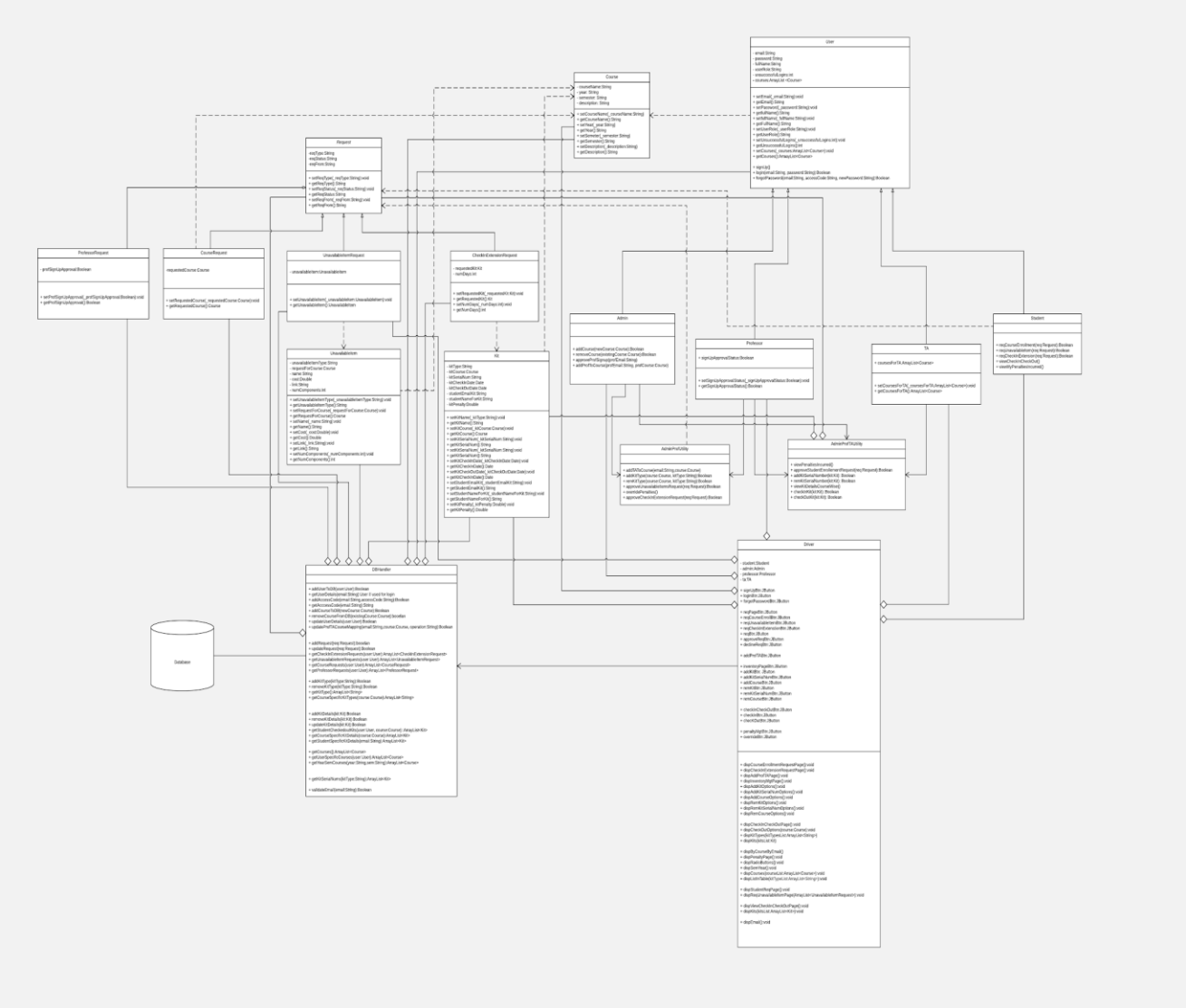
FR-02	The system should automate the email reminders of checkin deadlines to students and parallely cc to corresponding course wise professor and TA's
FR-03	The system should automate the generation of penalties and email it to the students and parallely cc to professor and TAs
NFR-01	Back up of data from database is performed once a week
NFR-02	Encryption of password to enable security

3. Show your Part 2 class diagram and your final class diagram.
What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

In the initial implementation, we had a centralized Driver to control the various actions and a centralized DB handler to access the Database. After discussion with Professor, we have understood that it is a bad architectural practice and such classes are called blob classes. We also removed Utility classes. Then we went on to make major changes and implemented MVC architectural pattern, because of which we got a clearer class diagram. We implemented the following Design patterns as a part of refactoring:

Part 2 Class Diagram:

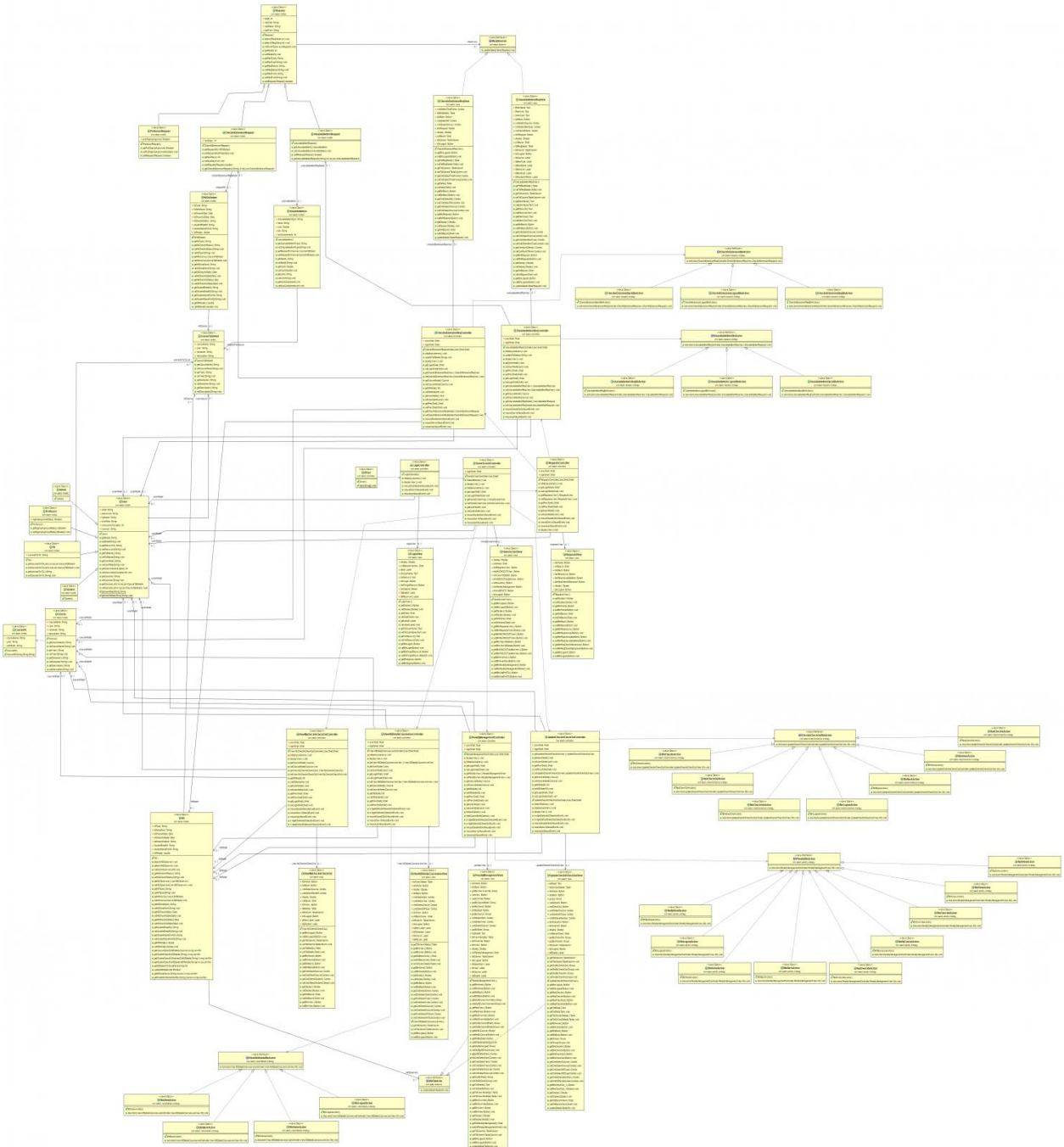
Good resolution of Part 2 class diagram can be found at this link



Final Class Diagram:

For good resolution of final class diagram: link

The final class diagram of the implementation is created using Object Aid UML Explorer for Eclipse [2].



4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).

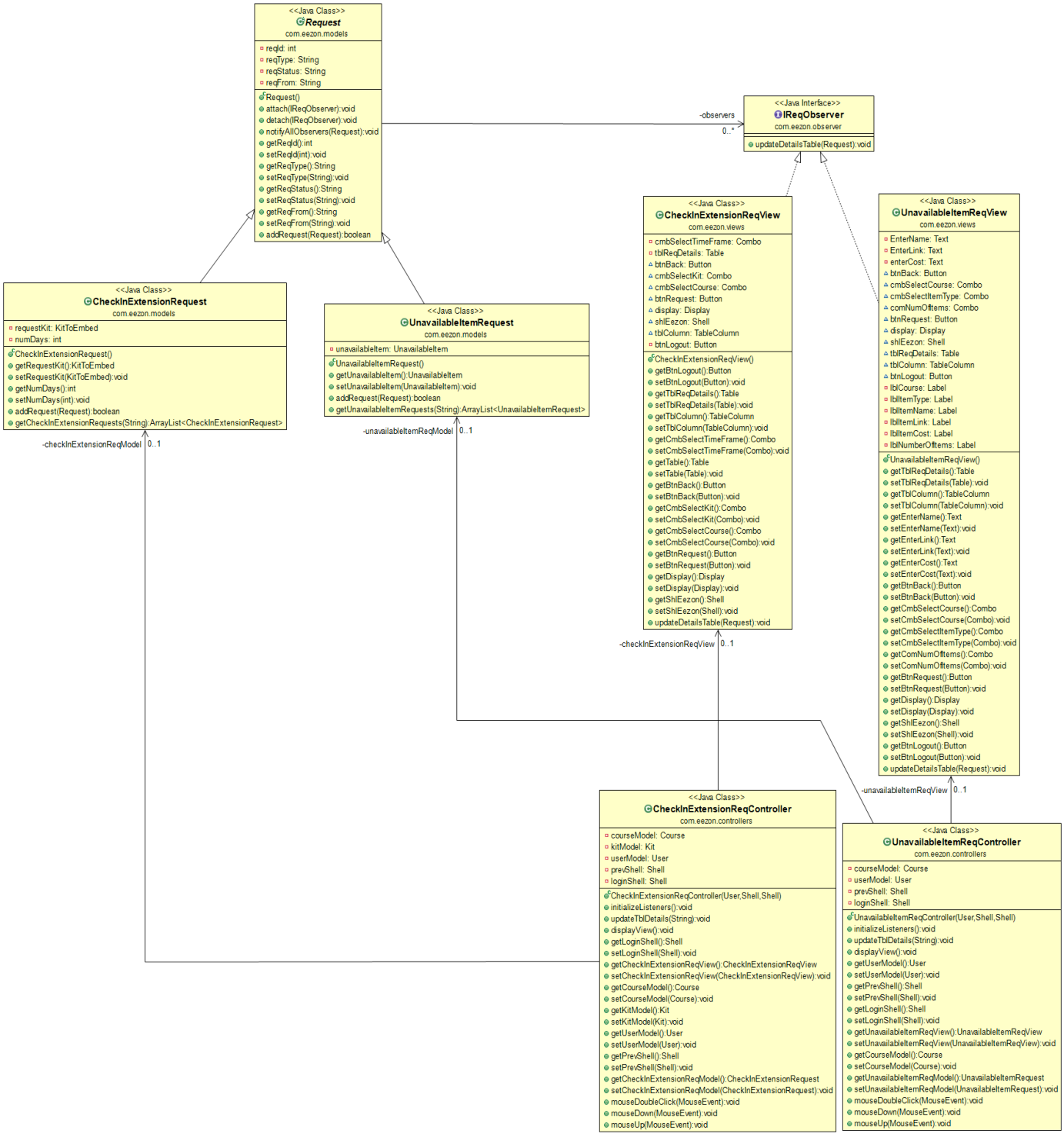
Each member of our project group implemented Observer Pattern and Strategy Pattern for their respective use cases as part for our final prototype.

Observer Pattern:

In general, Observer Pattern is implemented between the model and view classes in MVC. In the use cases of 'Checkin extension Request' and 'Unavailable Item Request', the IReqObserver interface acts as the observer for the Request class and the Request class is the subject. The addRequest() method is implemented by the sub classes of Request class which are CheckInExtensionRequest, UnavailableItemRequest. When these classes call the addRequest() method, foreach object 'o' in the observers o.UpdateDetailsTable() method is called in all the observers. The observer classes implement the UpdateDetailsTable() method according to their need. Similarly, Observer pattern is implemented for the use cases 'Update Checkin Checkout details and 'Override Kit Penalties'. The class diagrams for these use cases that implement design patterns are shown below.

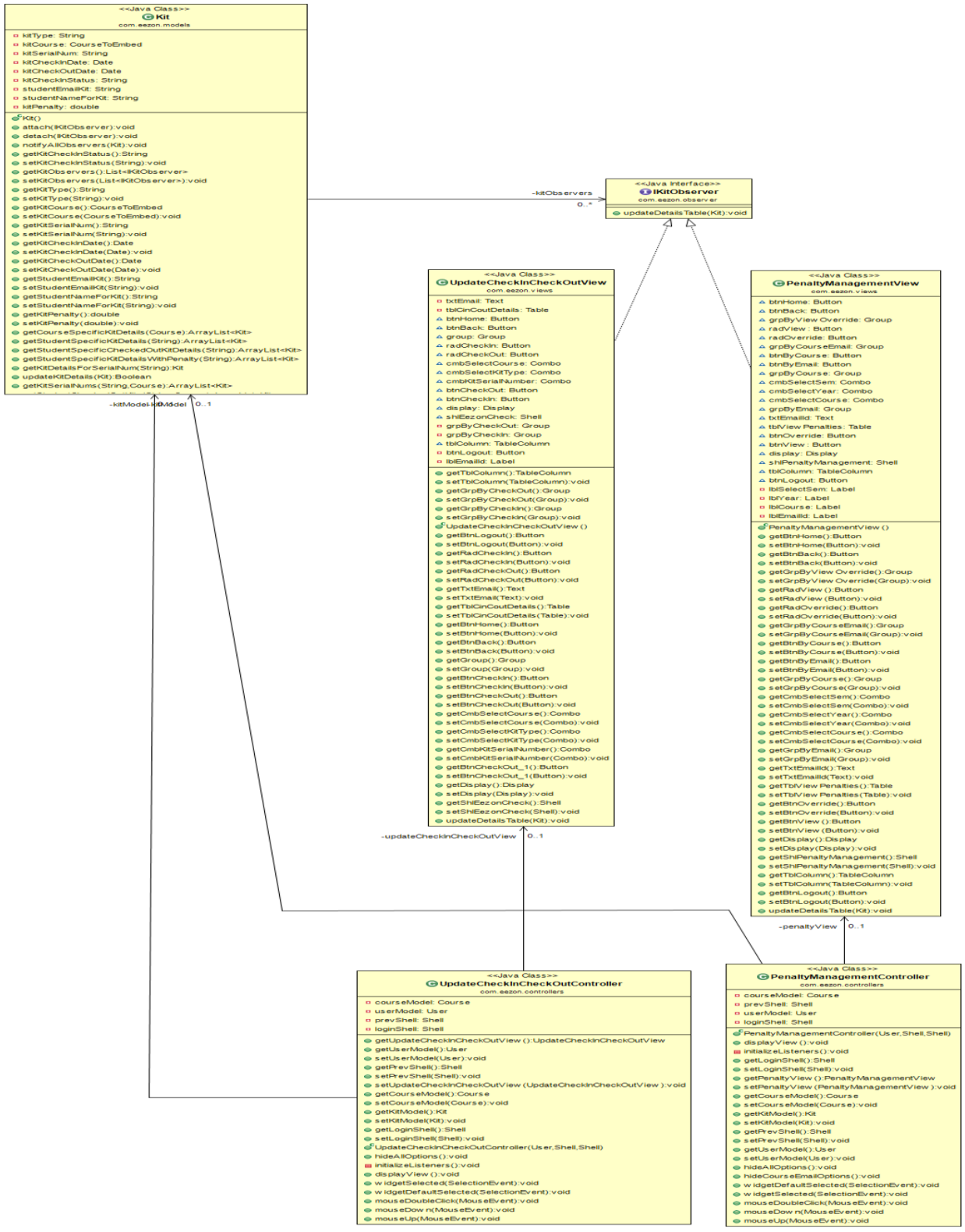
Request Observer:

Better resolution image can be found in this [link.](#)



Kit Observer:

Better resolution image can be found in this [link.](#)

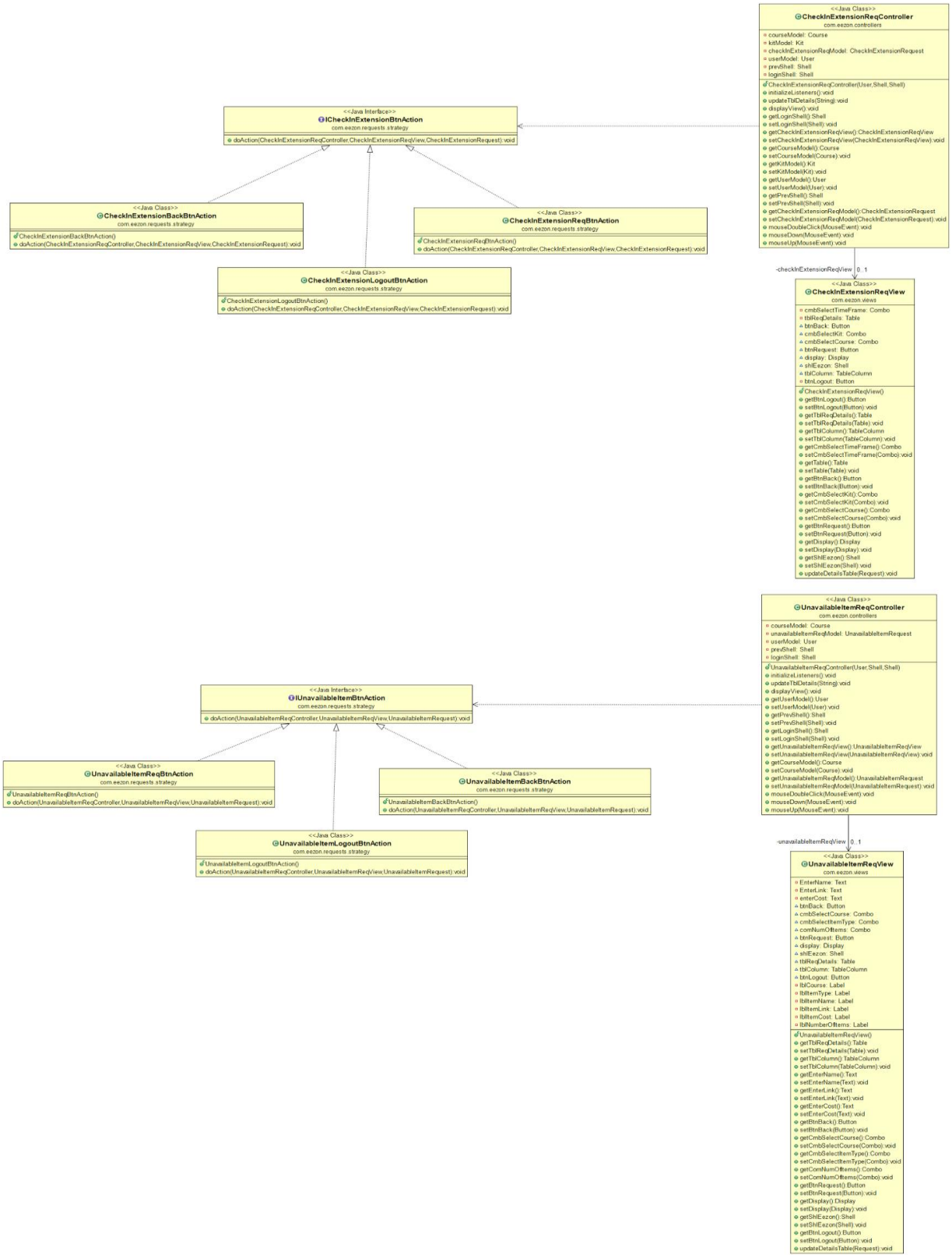


Strategy Pattern:

In general, the strategy pattern is implemented between the Controller classes and View classes. Here UnavailableItemReqController and UnavailableItemReqView classes are involved in strategy pattern. The IUnavailableItemBtnAction interface is implemented by several other Button Action classes, which implement the doAction() method according to their algorithm. For example when btnRequest:Button is clicked then the doAction() method inside UnavailableItemReqBtnAction will be executed. Similarly, strategy pattern is implemented to the other controller and view classes of the use cases, 'ViewKitDetailsCoursewise', 'CheckINCheckOUT' and 'PenaltyManagement'. The implementation of strategy pattern for these use cases is shown below.

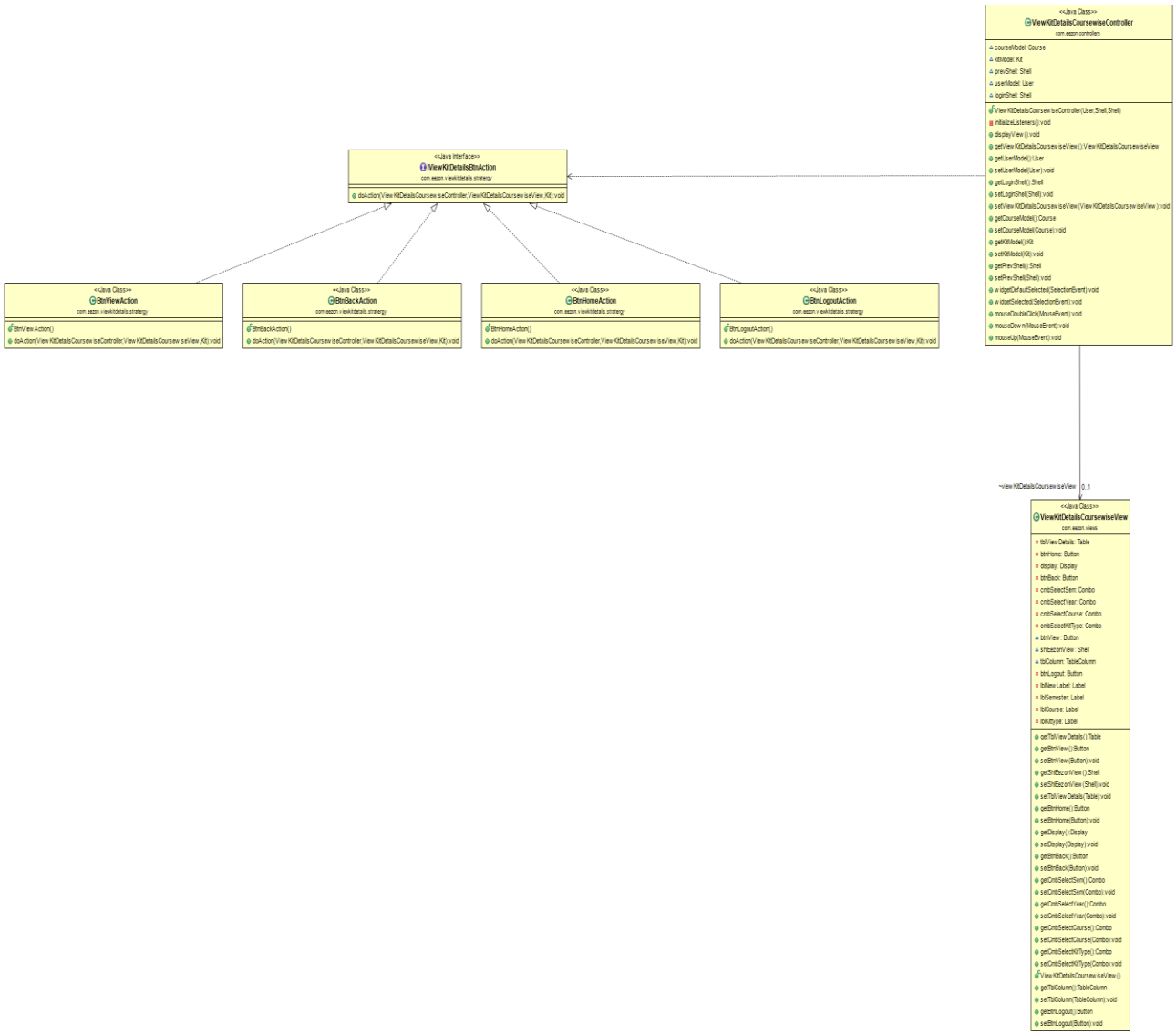
Request Strategy:

Better resolution image can be found in this [link.](#)



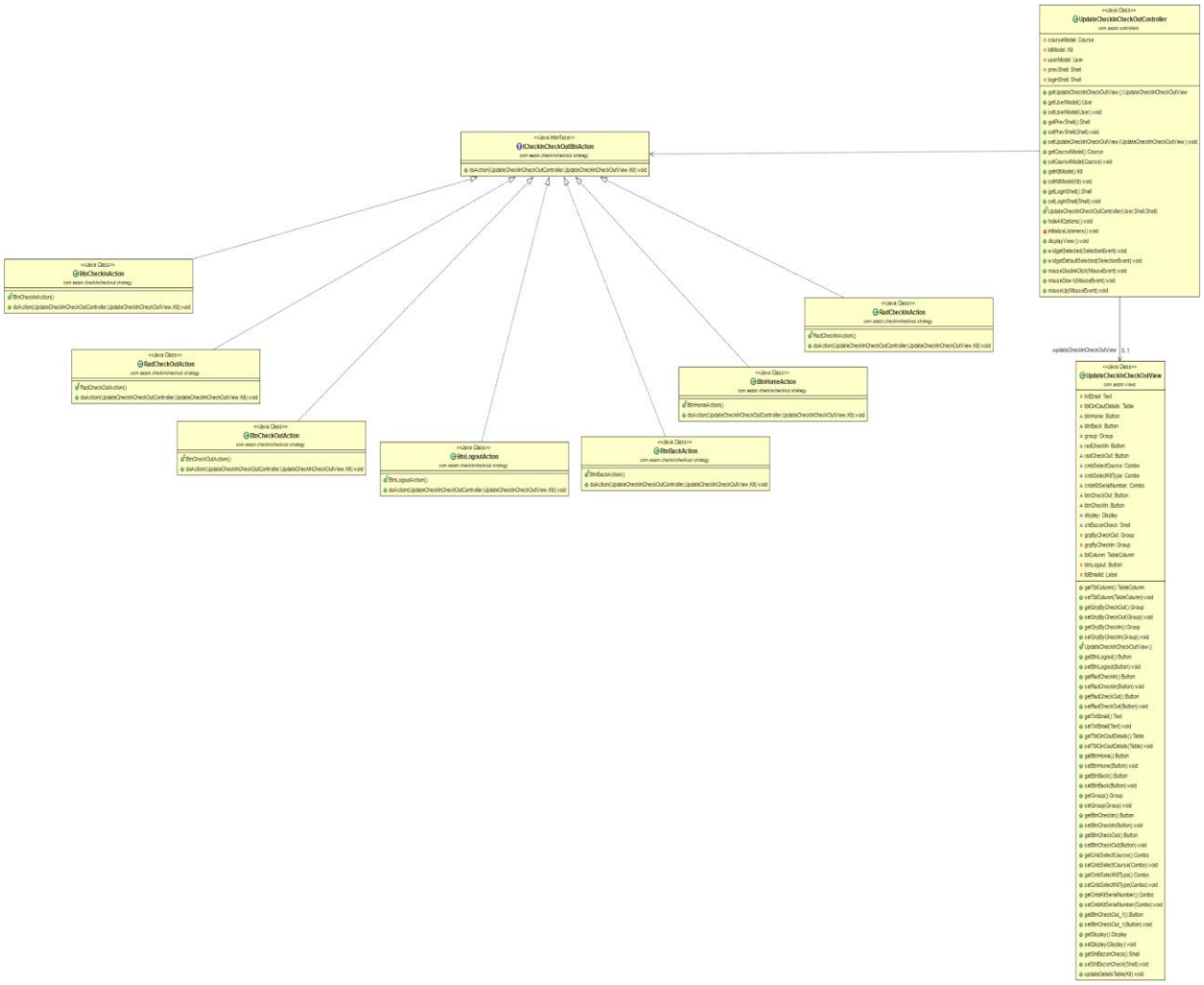
View Kit Details Strategy:

Better resolution image can be found in this [link.](#)



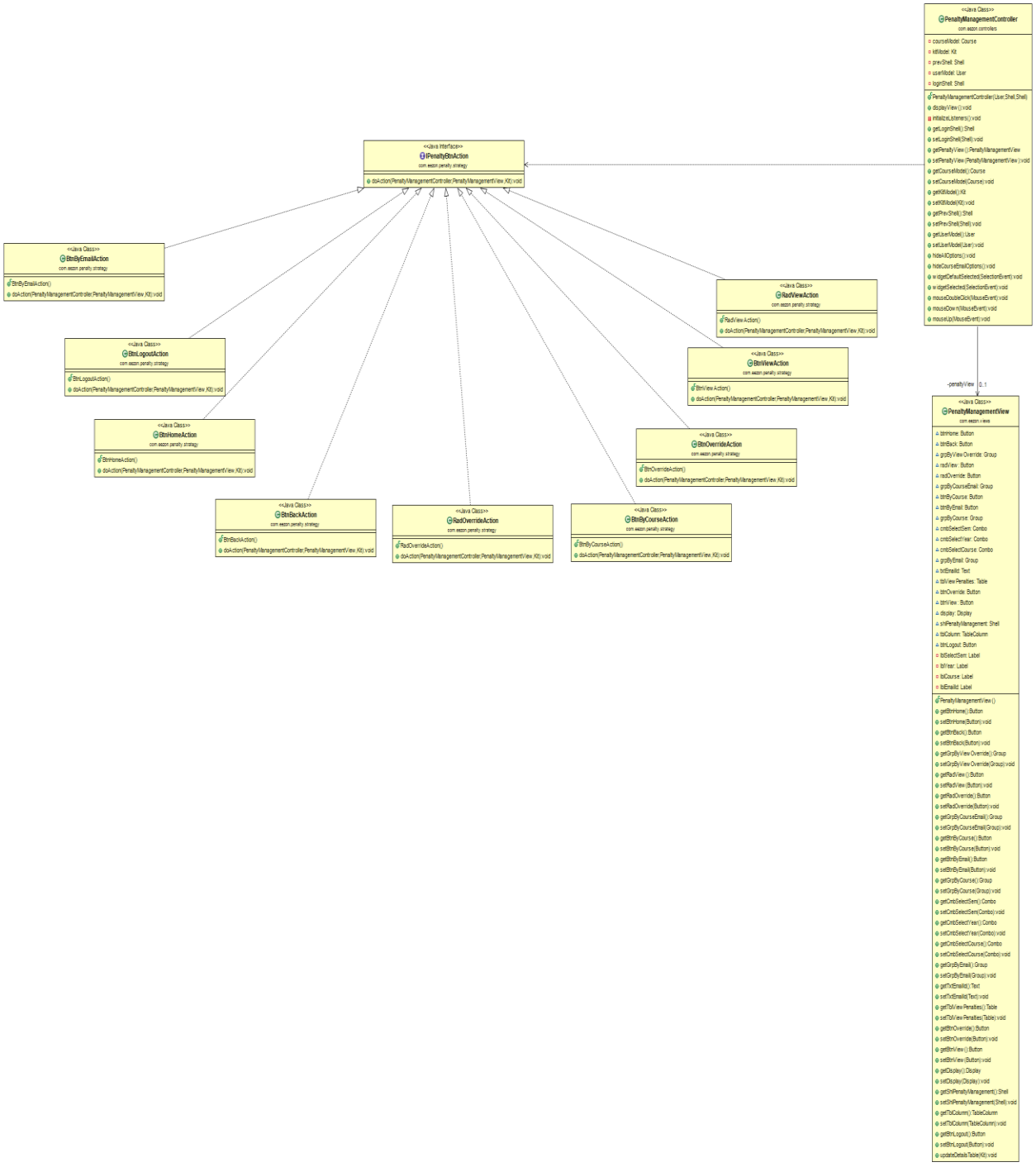
CheckIn CheckOUT Strategy:

Better resolution image can be found in this [link.](#)



Penalty Strategy:

Better resolution image can be found in this [link.](#)



5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Initially, we learned that taking requirements from client is very important for any project. Good requirements will help in understanding various use cases and identifying the actors in the project. We refined our requirements which helped us identify various use cases and the actors involved in the application. Once actors and use cases are identified, we drew Use Case Diagrams, Activity Diagrams, Sequence and Class Diagrams.

We learnt that using MVC Architectural pattern we can decouple model, view and controller into individual classes which helps in implementing a system easily. It helps in developing model, view and controller classes independently without any dependency on one another. Also, by using Design Patterns such as Observer and Strategy in MVC architectural pattern we can simplify the implementation.

Learning a new framework is always a challenging task and we successfully explored Java SWT GUI Application framework, Hibernate frameworks this semester. We also learned the advantages of Hibernate over plain SQL queries. Apart from the frameworks, exploring tools like Eclipse EGIT, Object AID UML Explorer, Java Decompiler also gave us a good learning curve of overall Software Development Life Cycle experience involving Object Oriented Analysis and Design.

References:

[1] https://github.com/sairam-muttavarapu/DistributedSystemsProject

Java SWT GUI Basic Implementation, Hibernate Basic configuration and implementation is referred from the above project.

[2] http://objectaid.com/home