Refactoring List

1. **<u>CLI/View Organized:</u>** Originally, the entire View was handled in the CLI Class. This made for a very large file that was easy to get lost in and search when having to make specific changes to methods. Specific methods were taken from this Class and distributed into other CLI Classes that held similar functions or related to specific classes/logic in the Gym System. For instance, the Hard Coding for the features and users to initialize the Gym System was moved into a specific Class because these methods only ran when the System started and in general were not needed to scroll past or investigate again when making other enhancements to the System.
2. **<u>Exercise, Routine and WorkoutClass Classes:</u>** All of these methods originally employed a very complicated equals() method that is worth noting. This was simplified in all of them to simply compare them by Name as comparing them by all of their attributes was making equals very complicated and ultimately unlikely to even occur because it was so specific what had to make any two of these Objects equal to one another. A simplified comparison by Name meets the requirements of the Project and made for a better experience.
3. **<u>Domain and Handler Organization:</u>** All of these classes were put into respective packages for their types to make them easier to navigate. There wasn't a need for most of the Person classes to be located with the Gym Entities so these were moved into separate packages.
4. **<u>All Extra Features Fully Implemented:</u>** This didn't require the implementation of any new classes, but did require logic to be added to existing Controllers and Domain classes. The major change was modifying the Membership enum to hold the various Membership types (Basic, Regular, Premium and Inactive) and then applying logic in the CLI to restrict accessing those features to the User. Fitness Classes were already created with the intent of finishing and now can be accessed by both Trainers and Customers for teaching/enrolling respectively. Additionally, enrolling with a Private Trainer wasn't initially done and now functions fully.
5. **<u>Renaming/Bug Fixes:</u>** Method names were modified in certain classes to make more logical sense, along with unused methods removed for being duplicates of others or not being needed. For instance, the Equipment Class originally had an updateQuantity class, but what this actually did was add more to the existing Equipment and the name was confusing, so this was renamed to addMoreQuantity. Additionally, ExerciseController had duplicate methods that weren't referenced so were deleted in favor of the others ones that were being used. As mentioned above, the Membership enum was updated to include the different Membership types. Lastly, the Trainer and Customer were modified so users could be partnered with one another as necessitated by the Extra Features. The tracked list includes these changes:
    a. Duration: getDuration to getMinutes, setDuration to setMinutes to make easier.

b. Equipment: udpateQuantity to addMoreQuantity to avoid confusion.
c. Exercise: setWorkoutSet method added to make functionality work correctly.
d. ExerciseController: Removed 3 addExercise classes as they were redundant once the equals() method was overridden to only use the name as an equals method.
e. Exercise: equals method re-written.
f. Routine: equals method re-written.
g. WorkoutClass: equals method re-written.
h. Person: Phone number attribute was altered from an int type to a String type. Int type was causing errors in creation of phone numbers and in hindsight was a bad choice, it should have been String all along.
i. CLICreatorScreens: from the CLI Class, the following methods were moved to this new Class to avoid clutter: createUser, createPerson, createEquipment, createRoutine, createWorkoutClass, createExercise.
j. CLIHardCode: from the CLI Class, the following methods were moved to this new Class to avoid clutter: hardCodedUsers, hardCodeTrainers, hardCodeCustomers, hardCodeEquipment, hardCodeWorkoutClass, hardCodeExercise, hardCodeRoutines.
k. CLIManageScreens: from the CLI Class, the following methods were moved to this new Class to avoid clutter: managePerson, manageEquipmentScreen, manageRoutineScreen, manageWorkoutClassesScreen, manageExerciseScreen,
l. CLIUserScreens: from the CLI Class, the following methods were moved to this new Class to avoid clutter: managerScreen, trainerScreen, customerScreen, customerClassesScreen, customerTrainerScreen.
m. Membership: Basic, Regular and Premium Membership status was added, in addition to a getStatus() method for comparing the types.
n. Trainer: The following methods were added to employ functionality for Customers enrolling with a Private Trainer: getCustomers, addCustomerToTrainer, removeCustomer.
o. Customer: The following methods were added to employ functionality for Customers enrolling with a Private Trainer: enrollWithTrainer, unenrollFromTrainer, getPrivateTrainer. Additionally, the following methods were also added to employ functionality for Customers to enroll in Classes: unenrollFromClass. Lastly, the getRoutines, addRoutine, removeRoutine and routinesToString methods were added to complete functionality for Customers to have Routines.
p. CustomerController: The following methods were added so that the Routine logic would function as expected and maintain Controller Design Pattern standards: assignRoutine, unassignRoutine, getCustomerRoutines.

q. TrainerController: The following methods were added to employ functionality for Customers enrolling with a Private Trainer and keep logic for handling access to this from the View here: enrollCustomerWithTrainer, unenrollCustomerFromTrainer.
r. RountineController: Method getRoutine removed as it wasn't needed anymore once the equals method in Routine was refactored for easier use.
s. WorkoutClassController: Ability for Customers to enroll/unenroll from Classes was added here to keep functionality out of View and maintain better design standards: enrollCustomerInWorkoutClass, unenrollCustomerFromWorkoutClass.

6. **Validation Logic Removed from Views:** Some logic for enrolling Customers in classes or signing up for Private Trainers was being validated in the View classes which wasn't good implementation. This resulted in some of the above changes that were needed in creating new Methods for certain Controllers and Users.
7. **Test Methods:** Additionally, with all of these changes came many, many updates to the applicable Testing methods.

Design Patterns List

1. **Controller Pattern**: The Controller Classes all were implemented as a means to provide a centralized location where handling of System mechanisms could be. The Dispatchers were implemented with Handlers that worked with the View and were employed to keep decoupling low so that View methods were not performing changes directly on Domain classes at minimum.
2. **MVC**: There was an implementation of a classic Model View Controller Pattern. Model included all of the Classes in the Domain package (Gym Entities and Users), the View included all of the Classes in the View package (Various CLI-named Classes), and the Controller included the handling of the System functionality between the Domain and the View. The Domain was kept at minimum to work with the View as little as possible and the logic was attempted to be relegated to the Controller Classes.
3. **Builder Pattern**: The PersonBuilder Class was created to simplify the creation of the various User types in the Gym System (Manager, Trainer and Customer). All User types shared attributes for creating them in their most basic form so this was an easy place to centralize the logic. In the future, it would be better to employ a better Builder Pattern as currently there are specific features of users that are being accessed in the View methods and this breaks the standards of both MVC and the Builder Pattern, but time and resources haven't allowed this to be fully fixed yet.
4. **Factory Pattern**: Similar to the Builder Pattern, the Factory Pattern is implemented to ease creation of User types through the PersonFactory Class. This methodology kept decoupling low between the View and the Domain Classes so that User Types could be created in a centralized location.