# *Programming IV (420-B42-HR)*

# *Assignment 3 – Comprehensive*

Date assigned: March 13, 2017

Date due: April 3, 2017

**Learning Objectives**

Upon successful completion of this assignment, the student will be able to:

* Create a class library
* Create a test library

To do:

Part 1 – The Set Up

1. This is a group project. Only one solution needs to be submitted. You will be given some class time to work on this project. You are also required to provide feedback on the other members of your team. That feedback will form part of YOUR mark as well as the team members’ marks.
2. In this assignment you will be creating a solution which will contain three projects. The solution name should be *groupname*ClassLibrary where *groupname* is your group name. As you know, the solution will create a default project inside with the same name. You are going to create a unit test project called *groupname*UnitTests. You are also going to create two class library projects one called *groupname*hvkBLL and the other *groupname*hvkDB) Once these three libraries are complete DELETE the default project that was created when you created the solution. Make sure that the unit test project references BOTH the class libraries and that the BLL references the DB.

Part 2 – Copying Other Assignments

1. Move your (likely updated) classes from assignment 1 to the BLL class library.
2. Create or add matching DB classes with the same name as the BLL class, but with the suffix DB to your DB class library (that is Pet.cs in the BLL class library will have a PetDB.cs class in the DB library). Leave these classes empty for now.
3. Add the methods from assignment 2 to the appropriate classes in this class library (that is, listOwners would go in the Owner class, etc).
4. Add one or more dataset(s) to the DB class library. There can be one dataset of all the tables or multiple datasets containing some of the tables. The design is up to you. With so many relationships between tables I know how I would do it (subtle hint). You should connect to the team database; make sure that database is up-to-date.
5. Add queries as required to the datasets so that you can access the data matching the seven queries from assignment 2. Then add methods to the appropriate classes in the DB class library to access and return that data. The data should be returned in data tables and converted to lists in the BLL class library.
6. Move the corresponding unit tests from assignment 2 to the unit test library in this project. If you set them up properly in assignment 2 they should run correctly right away; if not you may have to update them. Run the unit tests to make sure that you are getting back the correct data from the database.
7. Move the methods from Part B of assignment 2 to the appropriate class in the BLL class library. Do NOT update them yet…that is coming.

Part 3 – Making Reservations

1. Now comes the tricky part…we are going to start creating reservations. I STRONGLY recommend that you break this problem down and get a piece working before adding more logic. If you attempt all of this at the same time you will complicate things much more than necessary.
2. You will be using the business rules from Part B of Assignment 2 so make sure you have them handy at this point.
3. You are going to be responsible for scenario based testing for add reservation. These scenarios should run from simple to complex. So, for example, the simplest reservation would be adding a reservation for one pet with no services when there are no conflicts in the database. The most complex might be 4 pets, two each sharing a run, one needing a large-door run, with different services, multiple feedings for some and not others, medication for some and discounts for own food or something like that. You will also need to consider reservations that have conflicts such as no runs available or runs for part of the time or something like that.

Document these test scenarios in detail with the data from the database you will be using and review the test scenarios with me. A lot of this information should come from the work done in assignment 2, but it will likely have to be re-worked a little bit.

1. Add methods to the DB class library in the appropriate classes that will retrieve information required to determine if a reservation can be made and insert/update records in the appropriate database tables. These methods can be overloaded if you want to use the same method with differing sets of parameters. Once again, these methods in the DB class library should be suffixed with DB.
2. Add tests to your unit test library which will call these DB methods directly (NOT THROUGH THE BLL CLASS LIBRARY) with data to make sure that they work.
3. Now add the BLL methods which will call the database methods and return appropriate return values to the calling methods (the web application eventually).
4. Using the unit tests from Part B Assignment 2 as a base, add tests to the unit test library that tests that the BLL classes work correctly and return appropriate return values to the calling function. Here is a potential list of errors (probably incomplete):

| **Return Value** | **Type** | **Meaning** |
| --- | --- | --- |
| 0 | Success | Reservation successful |
| -1 | Warning | Expired or missing vaccinations |
|  |  |  |
| -10 | Error | Invalid pet number |
| -11 | Error | Start date before today |
| -12 | Error | Start date after end date |
| -13 | Error | Dog already has a reservation for all or part of period |
| -14 | Error | No run available for reservation period |
| -15 | Error | Reservation insert failed |
|  |  |  |
|  |  |  |
| -16 | Error | Pet number not in reservation |
| -17 | Error | Invalid reservation number |
|  |  |  |
|  |  |  |
| -18 | Error | Dog is already included in the reservation |
| -19 | Error | Dog has a different owner from other dog(s) in reservation |
|  |  |  |

Do:

1. Build the assignment in pieces and get each piece working.
2. Think about design and efficiency.
3. Make sure to perform error checking and handle errors well.

Don’t:

1. Write this all as one massive class.

**To submit**

1. A ZIP format (*username*B42A03.zip) containing all submitted files on Moodle.