1. Download jar files from myCourses, unzip and add to classpath
2. Open jGrasp, Project->New, check “create directory”, click “next”, uncheck “add files to project now” and click “create”
3. Add all of the jar files to the classpath for the workspace. You can do this by adding the folder that contains the jar files and then put /\* at the end before saving and applying it.
4. Create a new Java file, add a package statement for the service layer package (e.g. service), add a class definition (e.g. class LAMSService), save the file and have jGrasp create a directory for it, making sure it is added to the project.
5. Create a new Java file, copy and paste the DBSingleton.java code from the zip file on MyCourses into the file. Add a package statement at the top to match what you did in step 4. Save and compile the file making sure it is added to the project.
6. Create a new Java file, add a package statement for the business layer package (e.g. business), add a class definition (e.g. class BusinessLayer), save the file and have jGrasp create a directory for it.
7. Go back to the file created in Step 4, add an import for java.util.\*, components.data.\* and business.\*.
8. Add four method stubs for the methods described in the assignment, all returning a String. Add a property to the service and business classes (or any class that requires use of the database): DBSingleton dbSingleton;
9. In the initialize() method and make it:

dbSingleton = DBSingleton.getInstance();

dbSingleton.db.initialLoad("LAMS");

return “Database Initialized”;

1. In the getAllAppointment method, copy and paste lines 20 to 26 from the code from the TestDB.java file downloaded from my MyCourses. Insert a line to get an instance of the singleton object at the start of the method. Change it so instead of System.out’s you create and return a string. You might also want to add code to initialize the database if the the List returned is empty and re-fetch all of the data before creating the string.
2. Do something similar for the getAppointment method (add the getInstance line, copy and paste lines 29 to 38 from TestDB.java, change to return a String instead of System.out’s). Then change the code to use the appointment number passed in (instead of patient id and make sure you include the single quotes around the appointment number when you concatenate the string) and return an error string if the appointment doesn’t exist. You can also comment out lines 35-37 from TestDB.java for now, they are used for adding an appointment later. You will also have to add List<Object> at the start of the getData line.
3. Do something similar for the addAppointment method (lines 40 to 57). You will have to do some additional methods to get the psc, patient and phlebotomist (have the methods get the object from the database and return that object) using the data passed in via the xml, you can hard code for now. You may want to move some of the get methods to the business layer to be used in checking preconditions, etc. You may also want to move the code to add the appointment to the business layer as well.

//these ids would be from the xml...

Patient patient = getPatient("210");

Phlebotomist phleb = getPhleb("100");

PSC psc = getPSC("500");

Appointment newAppt = new Appointment("800",java.sql.Date.valueOf("2017-09-01"),java.sql.Time.valueOf("10:15:00"));

//extra steps here due to persistence api and join, need to create objects in list

List<AppointmentLabTest> tests = new ArrayList<AppointmentLabTest>();

AppointmentLabTest test = new AppointmentLabTest("800","86900","292.9");

test.setDiagnosis(getDiagnosis("292.9"));

test.setLabTest((LabTest)dbSingleton.db.getData("LabTest","id='86900'").get(0));

tests.add(test);

newAppt.setAppointmentLabTestCollection(tests);

newAppt.setPatientid(patient);

newAppt.setPhlebid(phleb);

newAppt.setPscid(psc);

DBSingleton dbSingleton = DBSingleton.getInstance();

boolean good = dbSingleton.db.addData(newAppt);

List<Object> objs = dbSingleton.db.getData("Appointment", "id='800'");

String string = "";

for (Object obj : objs){

string += obj.toString()+"\n";

}

return string;

1. Create a new Java file with a main method in it to test your service layer. You will need to import the service package. Add lines to call and print out the results of all of the methods you created. You will have to create an object of your service class on which to call all of the methods. The initialize method may take a while to run, so you may want to comment out the call to that method unless you want to reset the database. If you get some other exceptions like “Hibernate Dialect must be explicitly set” or no connection, just re-run. This happens especially when running in debug mode.
2. Go back and return xml strings instead of plain strings and test again.
3. Process xml in the addAppointment method and add in business layer code to check preconditions, etc. (change the test class to pass in the various xml from the project description for adding an appointment)
4. Once it is all working, add your annotations and follow the instructions for the structure of the WEB-INF folder (make sure you put just the .jar files into the lib folder), create the war file and deploy. Test using generated test client or SoapUI.