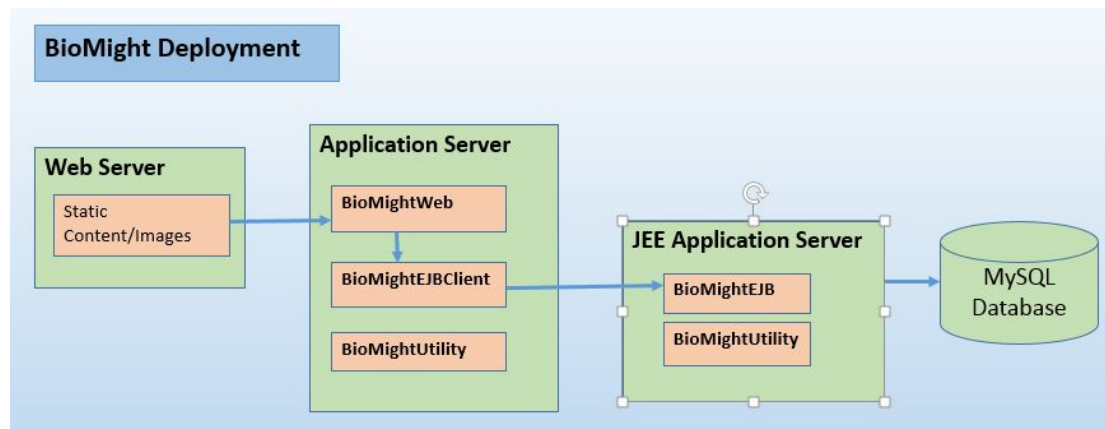


# **BioMight Development Guide**

**Version 1.2**

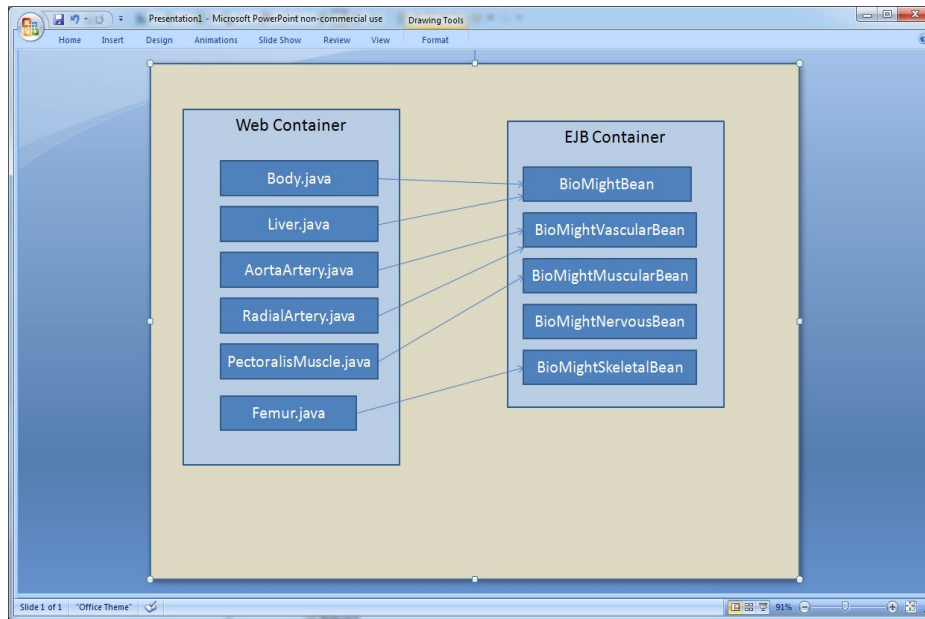
## Introduction

**BioMight** is a Web JEE (Java Enterprise Edition) application that is currently set up to run on the TomEE application server. It uses *MySQL* as its database and uses a JDBC connection as the data conduit. Below is a typical, but simplified, deployment model that could be used for any Web application. *TomEE* provides the Application Server and a lightweight JEE Application server under one program.



**BioMight's** Java classes mimics biological component. They run in the Web Container (Client Code) that runs on the Application server. These Java class get data from the user via the web page, package it up, and then send it to the EJB component for processing.

The EJB component runs under the JEE Application server and contains the core programming logic that creates the biological components and stores the information in the database. This model allows one to distribute work between the *Web Container* and the *EJB Container*.



## BioMight Process Model

Below, we show the Java projects, or libraries that comprise the application. There are 4 projects. The **BioMightWeb** project contains the Java client classes such as Arm, Leg, Stomach, Cell, DNA, etc. These classes define the starting coordinates for the biological component and attributes such as color, shape, texture that are gathered from the user via the web pages.

The **BioMightEJBClient** contains the glue that allows the client code to talk with the code on the JEE Application Server. It contains declarations for each method that exists in the corresponding EJBs. It is a way to present a listing of what's available.

The **BioMightUtility** project has methods that are used by the Web and EJB projects such as drawing graphics, and defining common data objects that are used to pass information between the two application layers. Lastly, the **BioMightEJB** project which contains the code to create the objects by getting their database info.

## BioMight Application Projects

### BioMightWeb (Web Client)

**Heart**  
Heart()  
Create()  
Generate()  
getX3D()  
executeMethods()

**Femur**  
Femur()  
Create()  
Generate()  
getX3D()  
executeMethods()

**NerveCell**  
NerveCell()  
Create()  
Generate()  
getX3D()  
executeMethods()

### BioMightEJBClient

**BioMightVascularBeanLocal**  
generateXXXX()

**BioMightMuscularBeanLocal**  
generateXXXX()

**BioMightCellularBeanLocal**  
generateXXXX()

### BioMightEJB

**BioMightVascularBean**  
generateXXXX()

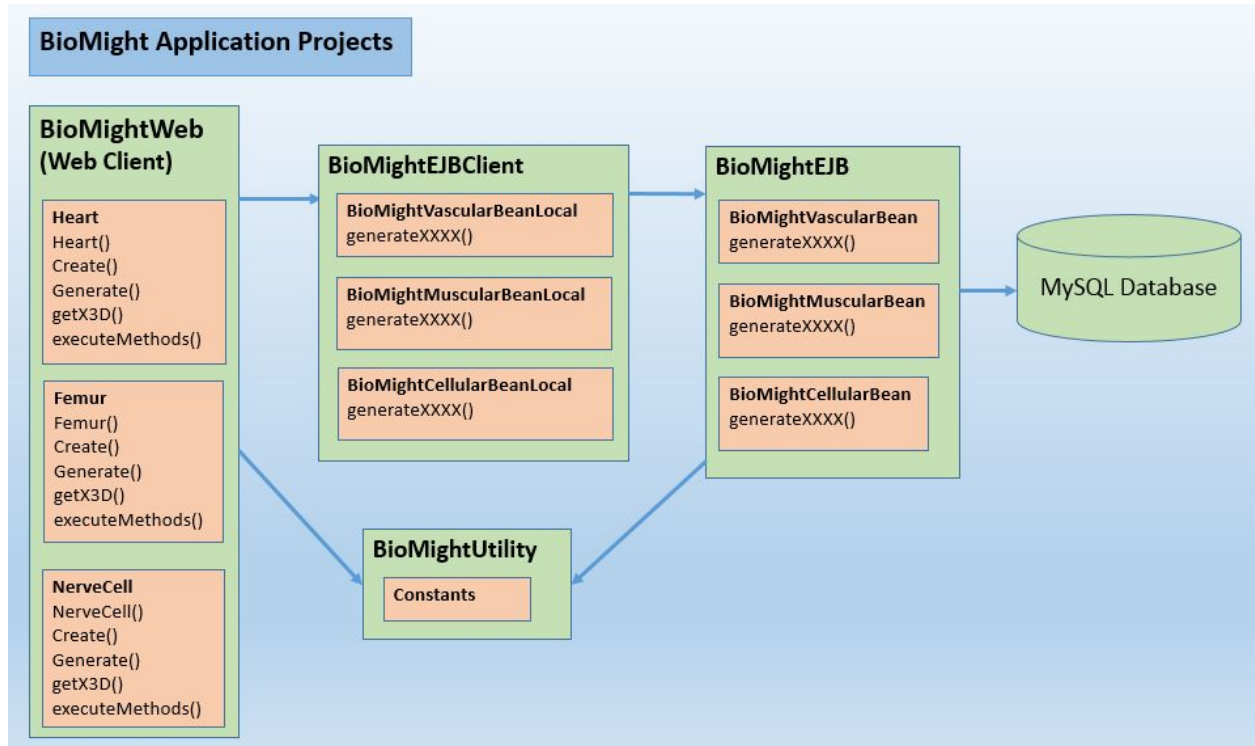
**BioMightMuscularBean**  
generateXXXX()

**BioMightCellularBean**  
generateXXXX()

MySQL Database

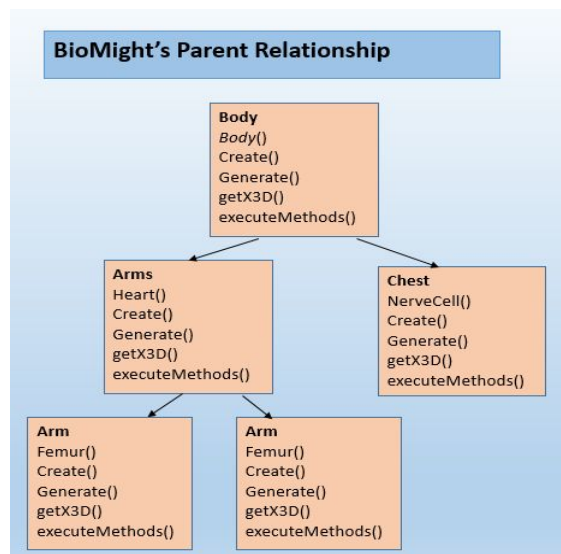
### BioMightUtility

Constants



BioMight organizes its components into parent-child relationship. In the Body view, the Body.java class sits at the top, with head, neck, chest, stomach as child objects. The Head.java class has child objects named eyes, ears, nose, etc.

In this hierarchy there are also pairs. For instance, most humans have a pair of arms and legs, and a multitude of thoracic vertebrae. When there is one instance of an object, that object does all the needed work, for example, retrieving and writing the data for the object to/from the database. Where there is a pair, or a collection of objects, the collection class gets the data and then calls upon the Java object to be created using that data.



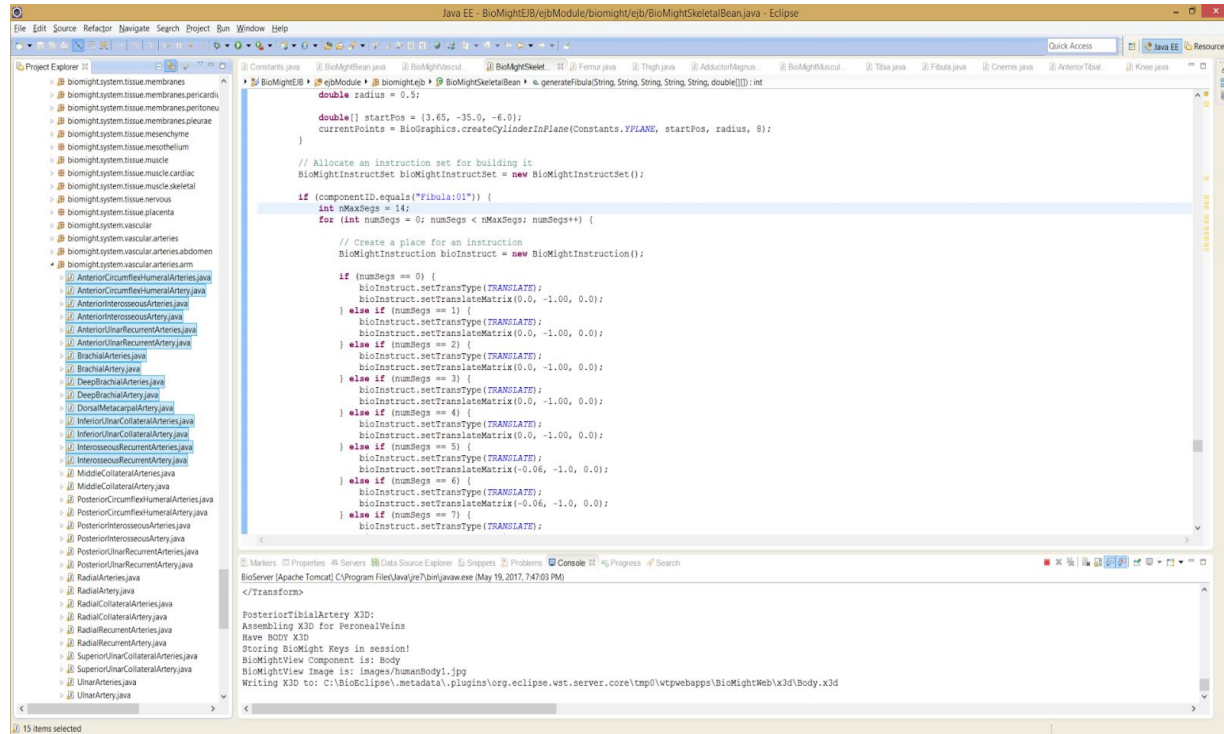
There are just over five-thousand Java client-based classes in the **BioMightWeb** library. Fortunately, they all work the same. Each has a set of methods; *constructors()*, *create()*, *generate()*, *getX3D()* and an *execute()* methods/functions.

All Java classes in the Web Client library contain a call to the *getComponents()* method that is contained in the **BioMightBean** EJB. This method gets the data from the database that it uses to construct itself and its X3D representation.

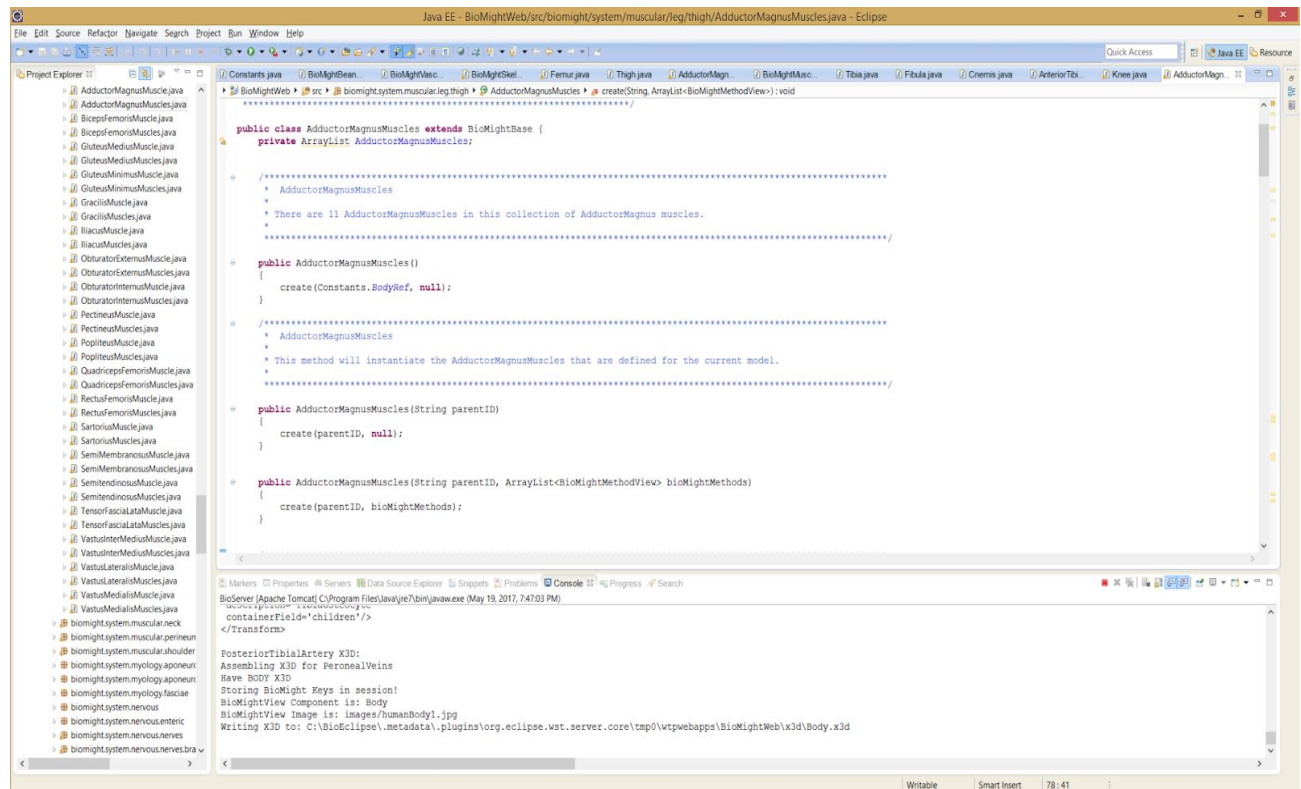
All classes also have a *generate()* method. This calls upon the associated *generateXXXX()* method that resides in one of the Java objects (Server Code) that run in the EJB container such as the **BioMightVascularBean**, **BioMightSkeletalBean**, or the **BioMightMuscularBean**.

# BioMight Web Details

In the screen capture below, we see some examples of BioMight's client Java classes. Highlighted is a sampling of the arteries that reside in the arm, such as **BrachialArtery.java**.



As there are a pair of **AdductorMagnus** Muscles in the human body, we create a collection and make the call to the EJB (Enterprise Java Bean) from the collection class.



## The Create Method

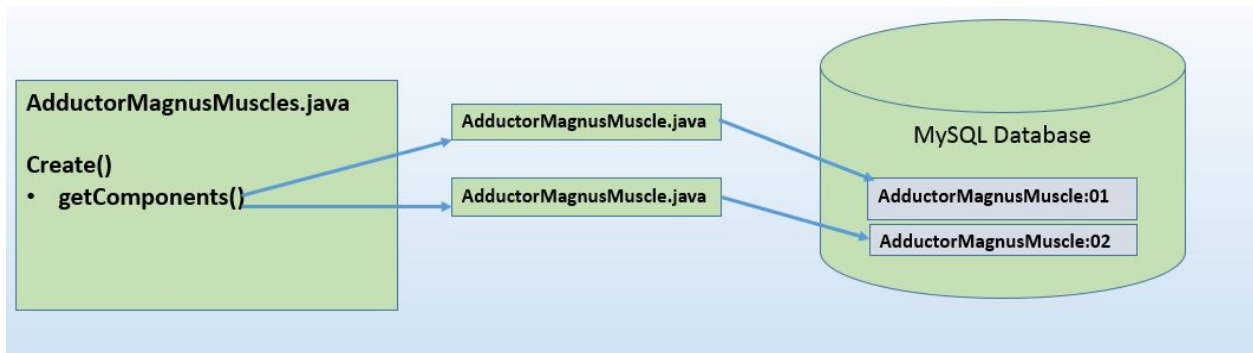
The *create()* method calls upon the *getComponents()* method in the **BioMightBean** (the base EJB) to get its information from the database. Every Java class (in the web client) uses to get its X3D data to draw the object.

Notice the `getComponents()` method has a parameter ***Constants.AdductorMagnusMuscleRef***. This is used as the primary key to retrieve the data from the database. Ensure that this constant is properly set in the **Constants.java** file, or your data will not be returned from the database.

We see that the Java class then uses a loop to process the data that has been returned from the database, and constructs **AdductorMagnusMuscle** objects to represent them.

An instance of **AdductorMagnusMuscle** will be created to represent the muscles in the right and left leg. These are represented in the database as *AdductorMagnusMuscle:01* and *AdductorMagnusMuscle:02* records.





Next we highlight the portion of the *create()* class that runs through the records that were retrieved from the database. The records are in the form of **BioMightTransform** objects. These objects model the *Transform* object defined in the X3D specification. A transform object defines a basic X3D object and when put into an X3D viewer will render a 3D image.

For this example, more than one row will come back from the database. As you can see from the Java code it builds a collection of **AdductorMagnusMuscle** (an ArrayList). Please note the *initProperty()* method is called to set up an object that allows it to be seen in the JSP (web interface) and access it (it holds database keys for lookup).

```

// AdductorMagnusMuscles.java

try {
    // Get the information from the database via the Enterprise Bean
    System.out.println("Getting AdductorMagnusMusclesInfo for ParentID: " + parentID);
    InitialContext ctx = new InitialContext();
    BiomightBeanLocal biomightBean = (BiomightBeanLocal) ctx.lookup(Constants.BioMightBeanRef);
    //biomightTransforms = biomightBean.getComponents(Constants.AdductorMagnusMuscleRef, Constants.AdductorMagnusMuscleRef+"0");
    biomightTransforms = biomightBean.getComponents(Constants.AdductorMagnusMuscleRef, parentID);
} catch (Exception e) {
    System.out.println("Exception Getting Components - AdductorMagnusMuscles");
    throw new ServletException("Remote Exception getComponents()", e);
}

// This is a collection, so set the parent of the children
// to that of the aggregation object
componentID = parentID;

// Run through the collection of AdductorMagnusMuscles and build them into the model
// In the Default case, we get two instances of the hip, one
// positioned on the right and one positioned on the left
ArrayList transforms = biomightTransforms.getTransforms();
System.out.println("AdductorMagnusMuscles NumTransforms: " + transforms.size());
for (int i = 0; i < transforms.size(); i++) {
    // Get the information for the hip we are creating
    BioMightTransform biomightTransform = (BioMightTransform) transforms.get(i);
    System.out.println("Creating AdductorMagnusMuscle: " + biomightTransform.getName() + " " + biomightTransform.getId());

    // Create an instance of the AdductorMagnusMuscle for each transform specified for the organism
    AdductorMagnusMuscle adductorMagnusMuscle = new AdductorMagnusMuscle(biomightTransform.getId(), biomightTransform);
    System.out.println("AdductorMagnusMuscle CreatedId: " + biomightTransform.getName() + " " + biomightTransform.getId());
    AdductorMagnusMuscles.add(adductorMagnusMuscle);
    System.out.println("Added AdductorMagnusMuscle to Collection: " + biomightTransform.getName() + " " + biomightTransform.getId());

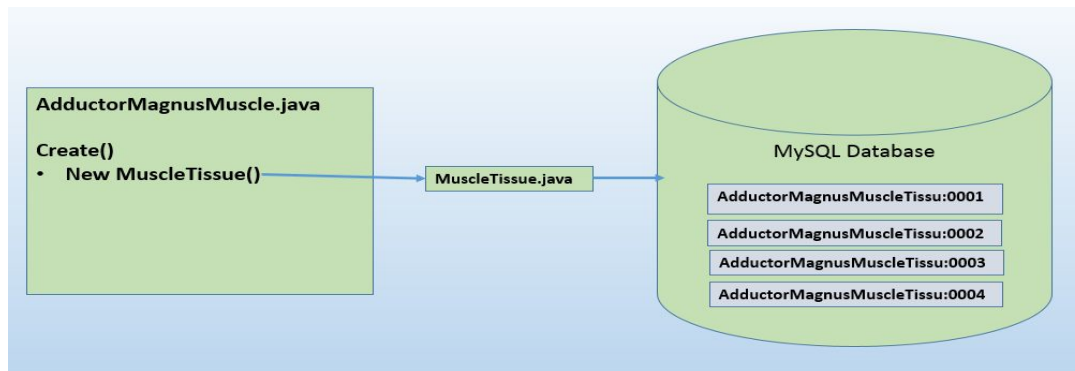
    // Initialize the Properties
    // In this case, rather than pass canonical name, we pass the name found in the collection
    initProperty(biomightTransform.getName(), Constants.AdductorMagnusMuscleRef, biomightTransform.getId());

    // Set up methods that will be available to the AdductorMagnusMuscles
    initMethods();
}

```

## AdductorMagnus Object Creation

Whereas, the **AdductorMagnusMuscles** class creates a bunch of **AdductorMagnus** Muscle objects. The **AdductorMagnusMuscle** class builds the actual muscle by calling upon the **MuscleTissue** class.



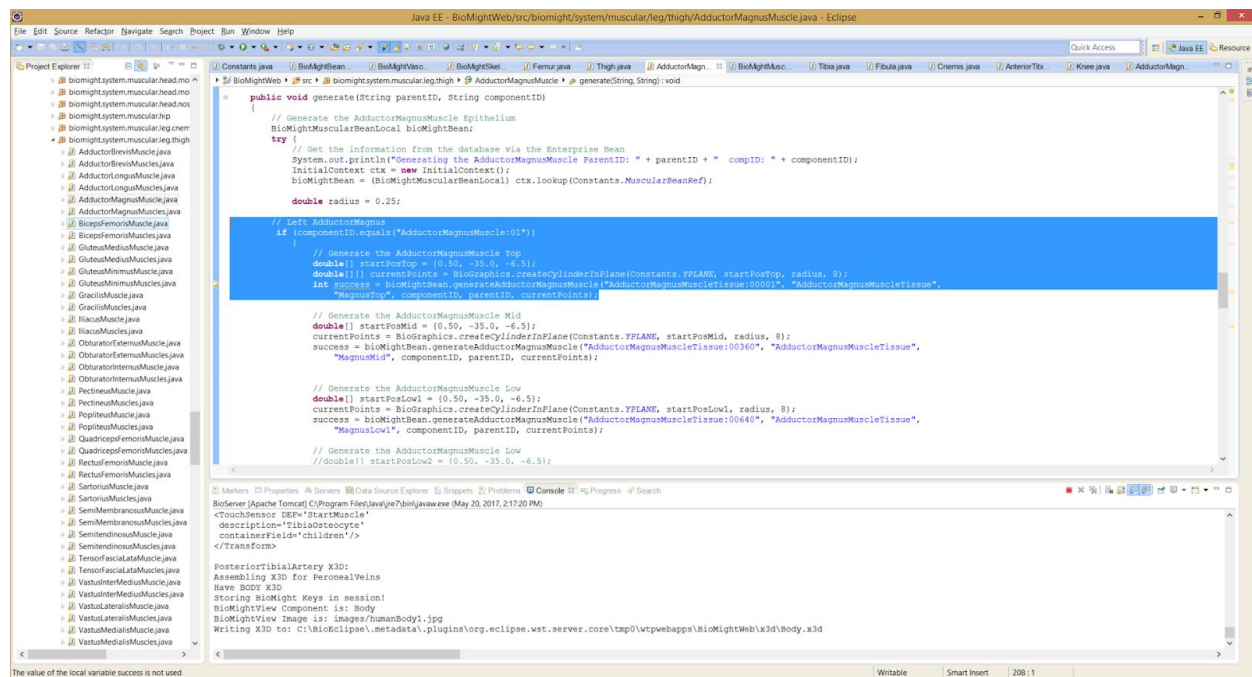
We see the *create()* method for this class below. It calls upon the *MuscleTissue()* class, passing in parameter for the current Muscle, AdductorMagnus.

*Note: Before we get the data from the muscle tissue data from the database, the generate() method can be called to create the data for the MuscleTissue, based on the parameters the user passes in through the web interface. (For now, I turn it on until the component looks how I want*

it, and then I set the flag to false – it saves all those updates to the database and makes running the application much faster)

The *generate()* method calls upon the corresponding *generateMagnusMuscle()* method in the **BioMightMuscularBean** EJB. In the example below, we first set up the coordinates for the initial points by calling the *createCylinderInPlane()* method. (For now, we are mostly using octogons. As the model is perfected, we can produce better shapes.) (Give me an octagon and a starting point, and I can create a tube by pushing the points and connecting them)

The **AdductorMagnus** muscles attach to the pelvis and attach to the associated femur in either the left or right leg. Please note the **BioMightBean** points to the appropriate EJB, **BioMightMuscularBeanRemote**. Arteries and Veins generate methods will call upon the **BioMightVascularBeanRemote**.



```
public void generate(String parentID, String componentID)
{
    // generate the AdductorMagnusMuscle Epithelium
    BioMightMuscularBeanLocal bioMightBean;
    try {
        // Get the information from the database via the Enterprise Bean
        System.out.println("Generating the AdductorMagnusMuscle ParentID: " + parentID + " compID: " + componentID);
        InitialContext ctx = new InitialContext();
        bioMightBean = (BioMightMuscularBeanLocal) ctx.lookup(Constants.MuscularBeanRef);

        double radius = 0.25;

        // Left AdductorMagnus
        if (componentID.equals("AdductorMagnusMuscle:01"))
        {
            // Generate the AdductorMagnusMuscle Top
            double[] startPosTop = {0.50, -35.0, -6.5};
            double[] currentPoints = BioGraphics.createCylinderInPlane(Constants.YPLANE, startPosTop, radius, 8);
            int success = bioMightBean.generateAdductorMagnusMuscle("AdductorMagnusMuscleTissue:00001", "AdductorMagnusMuscleTissue",
                "MagnusTop", componentID, parentID, currentPoints);

            // Generate the AdductorMagnusMuscle Mid
            double[] startPosMid = {0.50, -35.0, -6.5};
            currentPoints = BioGraphics.createCylinderInPlane(Constants.YPLANE, startPosMid, radius, 8);
            success = bioMightBean.generateAdductorMagnusMuscle("AdductorMagnusMuscleTissue:00360", "AdductorMagnusMuscleTissue",
                "MagnusMid", componentID, parentID, currentPoints);

            // Generate the AdductorMagnusMuscle Low
            double[] startPosLow1 = {0.50, -35.0, -6.5};
            currentPoints = BioGraphics.createCylinderInPlane(Constants.YPLANE, startPosLow1, radius, 8);
            success = bioMightBean.generateAdductorMagnusMuscle("AdductorMagnusMuscleTissue:00440", "AdductorMagnusMuscleTissue",
                "MagnusLow1", componentID, parentID, currentPoints);

            // Generate the AdductorMagnusMuscle Low
            double[] startPosLow2 = {0.50, -35.0, -6.5};
            currentPoints = BioGraphics.createCylinderInPlane(Constants.YPLANE, startPosLow2, radius, 8);
            success = bioMightBean.generateAdductorMagnusMuscle("AdductorMagnusMuscleTissue:00440", "AdductorMagnusMuscleTissue",
                "MagnusLow2", componentID, parentID, currentPoints);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Markers Properties Servers Data Source Explorer Snippets Problems Console Search

BiOServer [Apache Tomcat] C:\Program Files\Java\jre7\bin\java.exe (May 20, 2017, 2:17:20 PM)

```
<TouchSensor ID="StartMuscle"
description="TibiaOsteocyte"
containerField="children"/>
</Transform>

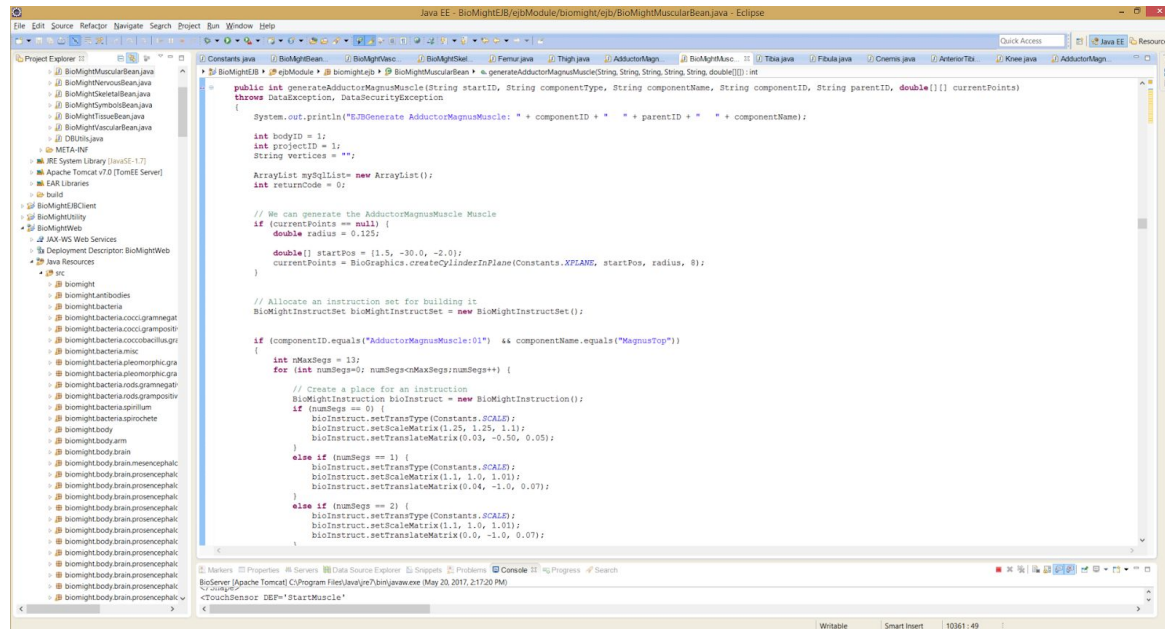
PosteriorTibialArtery X3D:
Assembling X3D for PersonnelVeins
Have BODY X3D
Storing BioMight Keys in session!
BioMightView Component is: Body
BioMightView Image is: images/humanBody1.jpg
Writing X3D to: C:\BioEclipse\metadata\plugins\org.eclipse.wst.server.core\tmp0\vtwebapps\BioMightWeb\X3D\Body.X3D
```

The value of the local variable success is not used

Writeable Smart Insert 208:1

# The EJB Methods

In the **BioMightMuscularBean** EJB, we see the *generateAdductorMagnus()* method that the **AdductorMagnus.java** client class calls to build the actual muscle. The EJB method first performs validations and if the objects are null, it sets up default values. The *startPos* (start position) that is defined in the Client Java class and here in the EJB, should have the same value.



```
public int generateAdductorMagnusMuscle(String startID, String componentType, String componentName, String componentID, String parentID, double[][] currentPoints) throws DataException, DataSecurityException {
    System.out.println("EJBGenerate AdductorMagnusMuscle: " + componentID + " * " + parentID + " * " + componentName);

    int bodyID = 1;
    int projectID = 1;
    String vertices = "";

    ArrayList myArrayList = new ArrayList();
    int returnCode = 0;

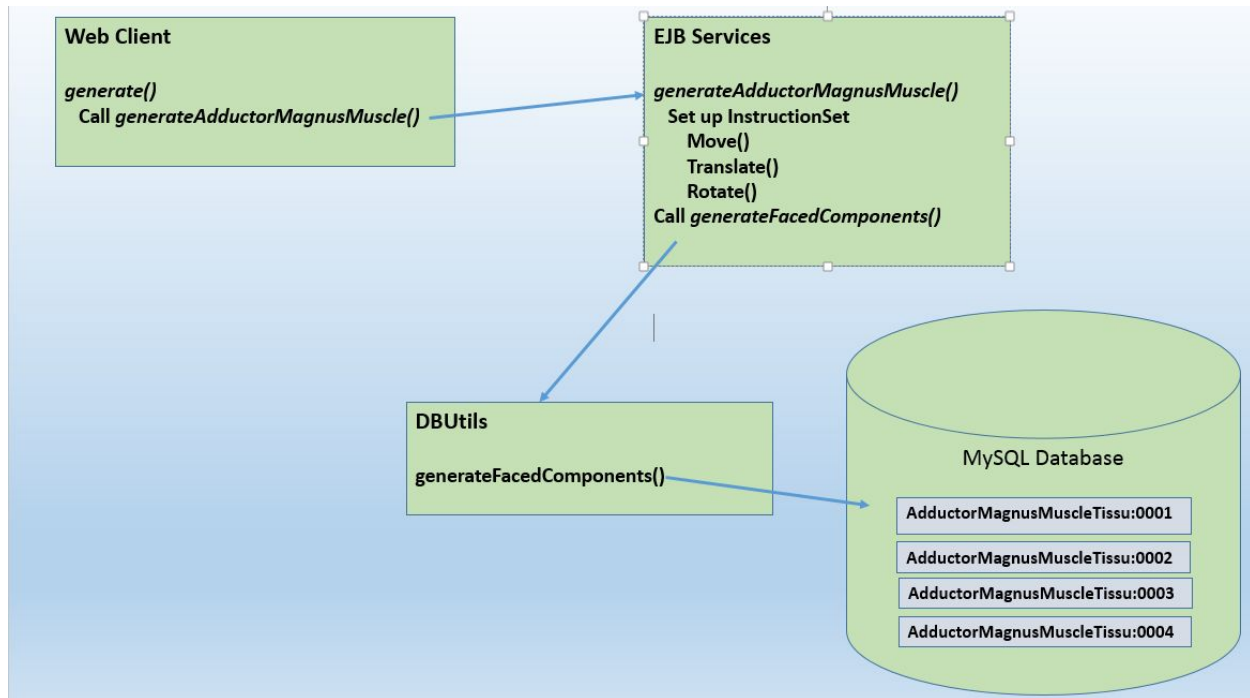
    // We can generate the AdductorMagnusMuscle Muscle
    if (currentPoints == null) {
        double radius = 0.125;

        double[] startPos = {1.5, -30.0, -2.0};
        currentPoints = BioGraphics.createCylinderInPlane(Constants.XPLANE, startPos, radius, 8);
    }

    // Allocate an instruction set for building it
    BioMightInstructionSet bioMightInstructionSet = new BioMightInstructionSet();

    if (componentID.equals("AdductorMagnusMuscle:01") && componentName.equals("MagnusTop")) {
        int numSegs = 13;
        for (int numSegs=0; numSegs<=MaxSegs;numSegs++) {
            // Create a place for an instruction
            BioMightInstruction bioInstr = new BioMightInstruction();
            if (numSegs == 0) {
                bioInstr.setTransType(Constants.SCALE);
                bioInstr.setScaleMatrix(1.25, 1.25, 1.1);
                bioInstr.setTranslateMatrix(0.03, -0.56, 0.05);
            }
            else if (numSegs == 1) {
                bioInstr.setTransType(Constants.SCALE);
                bioInstr.setScaleMatrix(1.1, 1.0, 1.01);
                bioInstr.setTranslateMatrix(0.04, -1.0, 0.07);
            }
            else if (numSegs == 2) {
                bioInstr.setTransType(Constants.SCALE);
                bioInstr.setScaleMatrix(1.1, 1.0, 1.01);
                bioInstr.setTranslateMatrix(0.0, -1.0, 0.07);
            }
        }
    }
}
```

Below, we see how the component gets generated by the various Java classes.

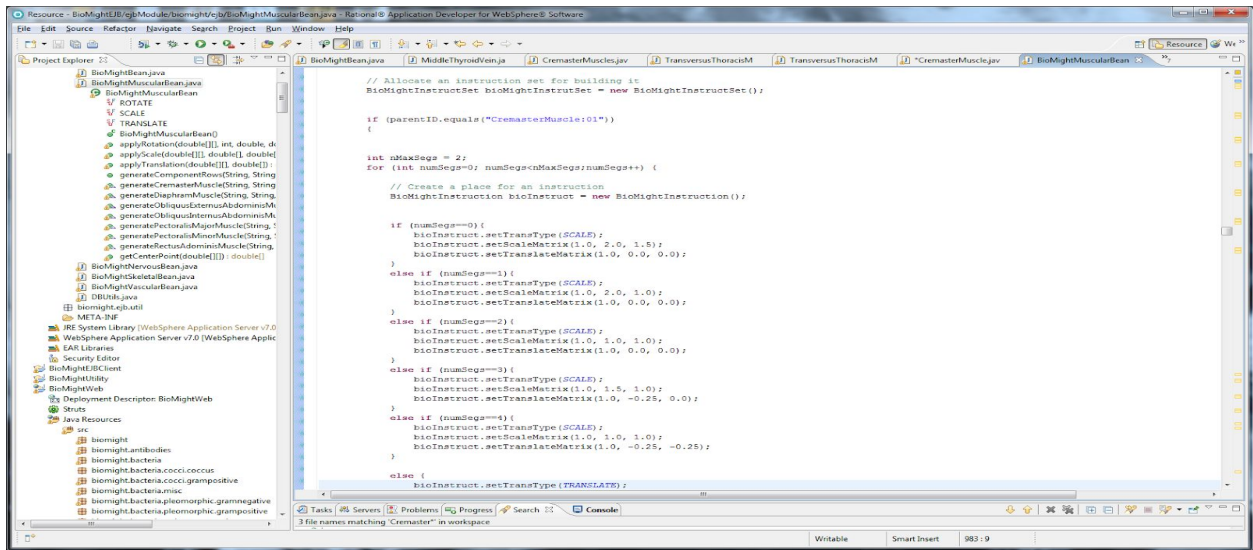


## Declare Instruction Set

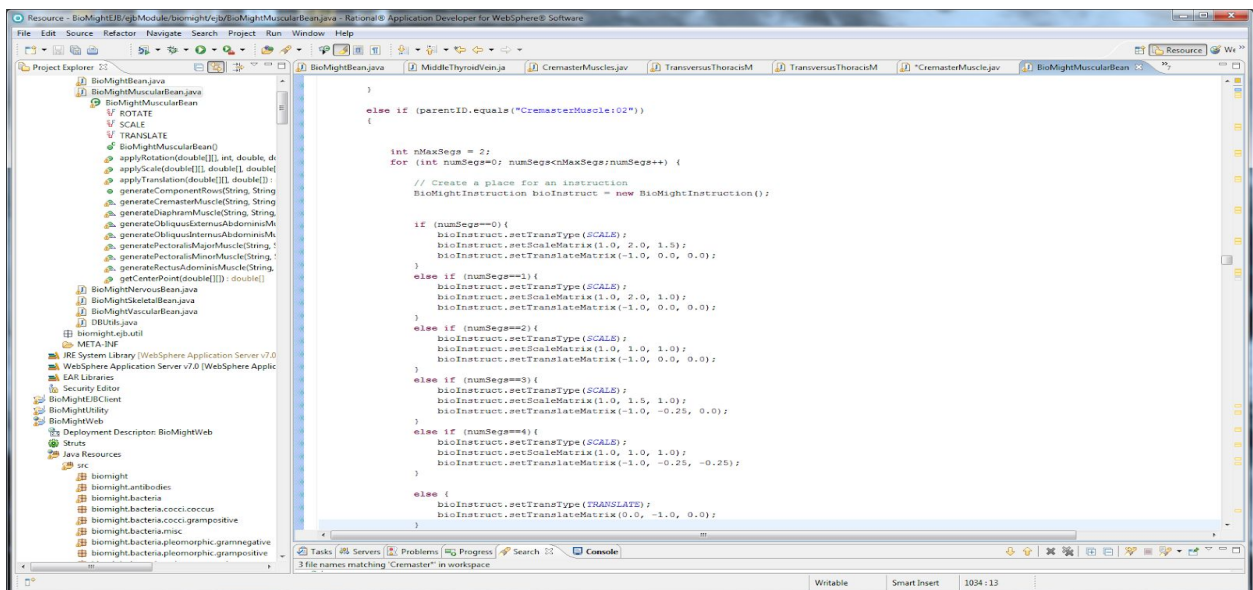
In the EJB, we set up a starting point and then supply a set of instructions to create the biological component. Looking up Adductor Magnus muscles via Google web search we see that there are a pair of AdductorMagnus muscles, one for each leg. When the Client class calls `getComponents()`, the database will find two references for the AdductorMagnus muscles.

In the database their key (*comp\_id*) will appear as *AdductorMagnus:01* and *AdductorMagnus:02* and we look for those specific references here to create it. We will provide instructions for creating each one by looking at the parent ID.



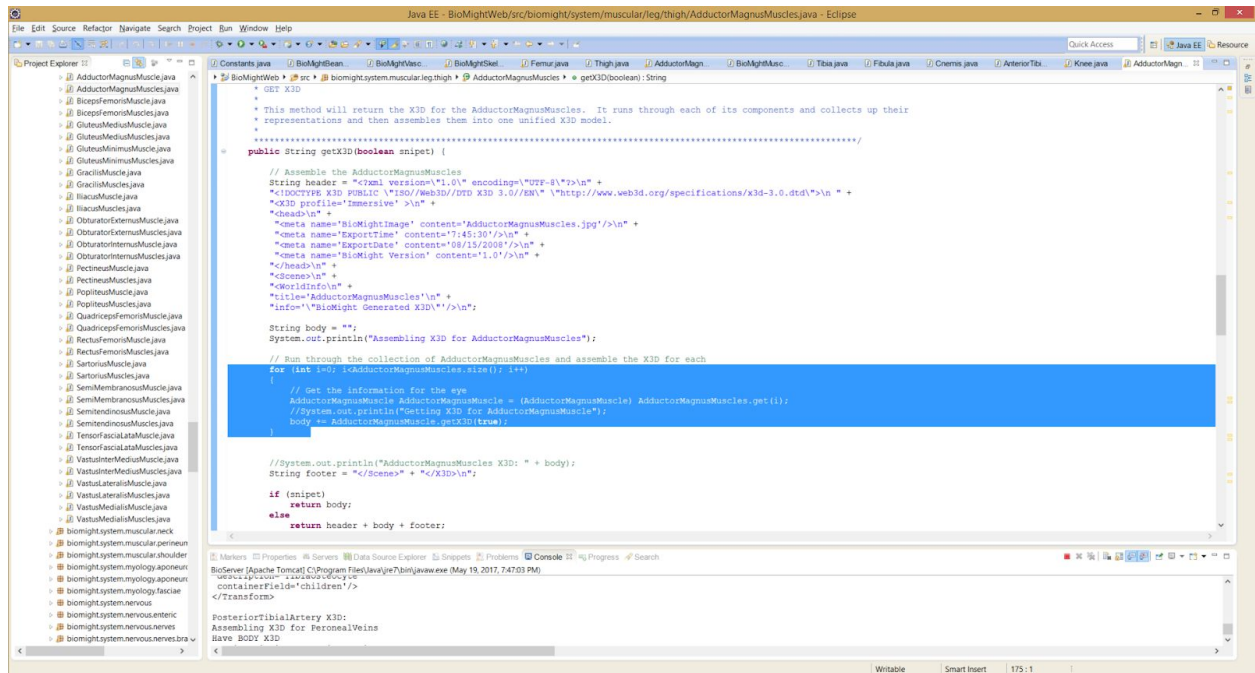


The code below handles the second AdductorMagnus muscle. There are commands for move, scale, and rotate.



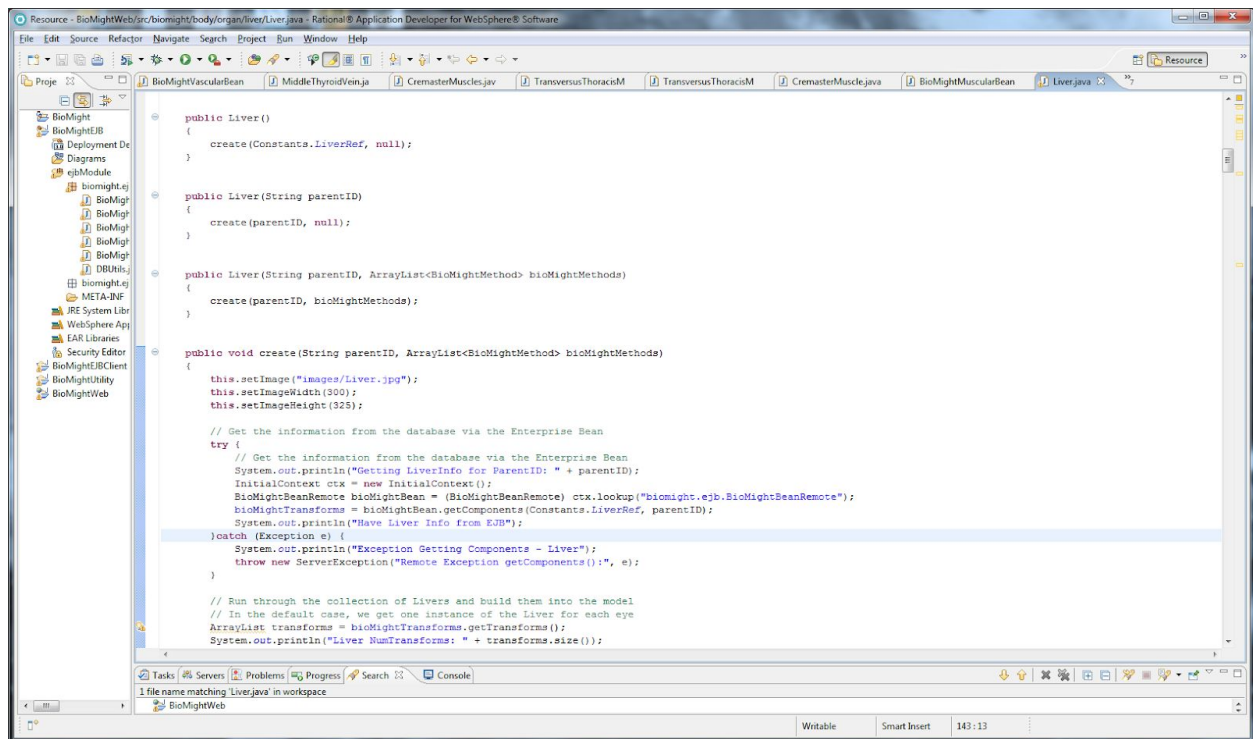
## Get X3D

Every Java class also has a *getX3D()* method. In this example, the X3D is created by getting the X3D from the individual **AdductorMagnus** muscles. The *getX3D()* method will ultimately work in a few ways. Based on the constant passed in, it may get the Transform XML data directly and return it, or it will drill down deeper into the biological models to get the X3D from the objects that comprise it.



## Example2

Below we see the constructors and *create()* method for the **Liver**. There is only one liver in the body, so this method gets its data from via the database directly. There is not a collection class for the Liver. As in most all cases, the method calls the *getComponents()* method in the **BioMightBean** to get the data to construct the Liver object. This brings back a list of *Transform* objects.



```
Resource - BioMightWeb/arc/biomight/body/organ/liver/Liver.java - Rational® Application Developer for WebSphere® Software
File Edit Source Refactor Navigate Search Project Run Window Help
Project: BioMight
  BioMight
  BioMightEJB
  Deployment Descriptor
  Diagrams
  ejbModule
  biomight.ej
  BioMight
  BioMight
  BioMight
  BioMight
  BioMight
  DBUtils
  biomight.ej
  META-INF
  JRE System Lib
  WebSphere App
  EAR Libraries
  Security Editor
  BioMightEJBClient
  BioMightUtility
  BioMightWeb

public Liver()
{
    create(Constants.LiverRef, null);
}

public Liver(String parentID)
{
    create(parentID, null);
}

public Liver(String parentID, ArrayList<BioMightMethod> bioMightMethods)
{
    create(parentID, bioMightMethods);
}

public void create(String parentID, ArrayList<BioMightMethod> bioMightMethods)
{
    this.setImage("images/Liver.jpg");
    this.setImageWidth(300);
    this.setImageHeight(325);

    // Get the information from the database via the Enterprise Bean
    try {
        // Get the information from the database via the Enterprise Bean
        System.out.println("Getting LiverInfo for ParentID: " + parentID);
        InitialContext ctx = new InitialContext();
        BioMightBeanRemote bioMightBean = (BioMightBeanRemote) ctx.lookup("biomight.ejb.BioMightBeanRemote");
        bioMightTransforms = bioMightBean.getComponents(Constants.LiverRef, parentID);
        System.out.println("Have Liver Info from EJB");
    } catch (Exception e) {
        System.out.println("Exception Getting Components - Liver");
        throw new ServletException("Remote Exception getComponents():", e);
    }

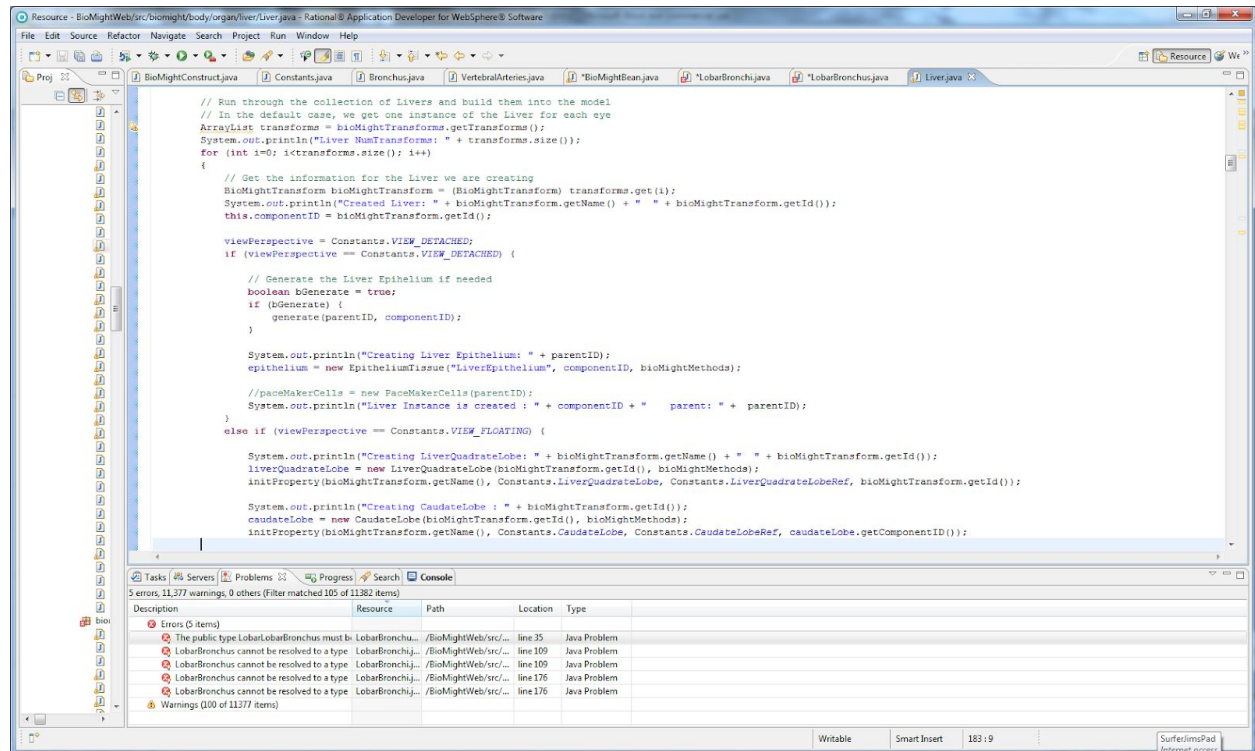
    // Run through the collection of Livers and build them into the model
    // In the default case, we get one instance of the Liver for each eye
    ArrayList transforms = bioMightTransforms.getTransforms();
    System.out.println("Liver NumTransforms: " + transforms.size());
}
```

Tasks Servers Problems Progress Search 1 file name matching 'Liver.java' in workspace Console

Writable Smart Insert 143:13

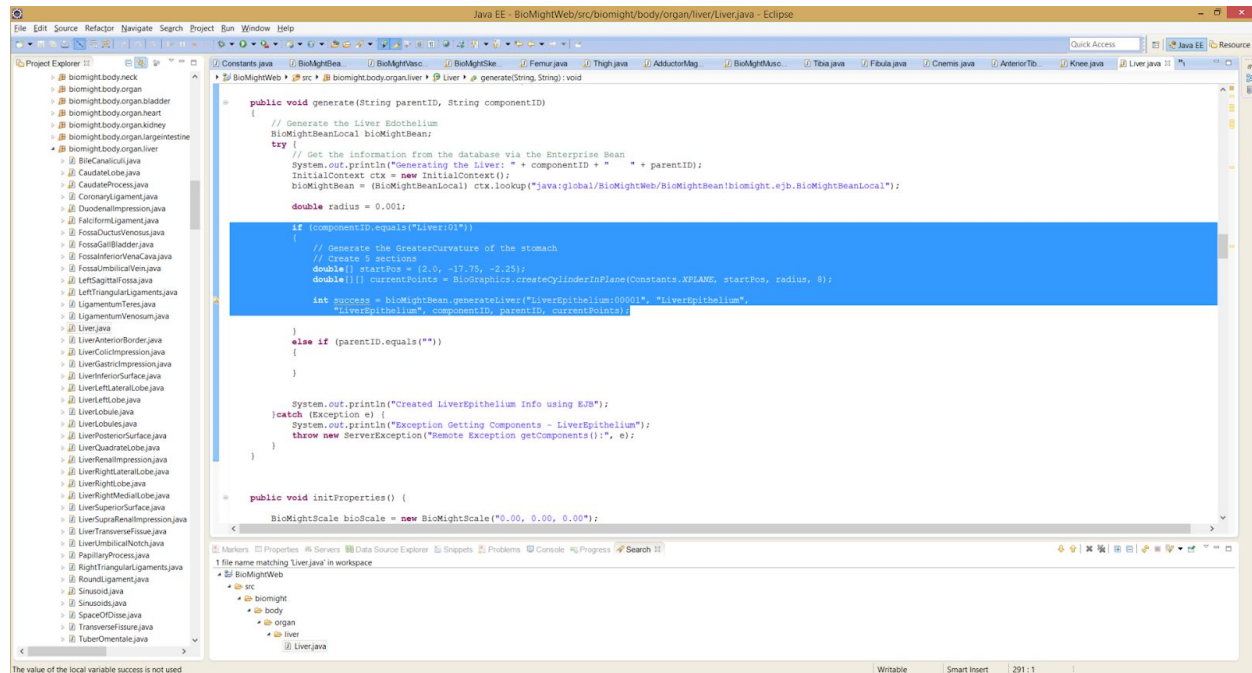


We run through the Transform objects that are returned from the database. One row will be returned by default (Liver:01), and for that object we are going to create the **EpitheliumTissue**. If the view level asks for more detail (as seen in the Else-Block), we will drill down to get the X3D from the subcomponent objects. Again, we the *generate()* method that will be called when we want to create or update an object.



# Generate the Liver

Below we see the *generateLiver()* method that calls upon the BioMightBean (EJB) to create or update the object. The liver starts at -17.75in on the Y axis for me. After setting up the octagon, the *generateLiver()* method in the EJB is called.



```
public void generate(String parentID, String componentID)
{
    // Generate the Liver Edothelium
    BioMightBeanLocal bioMightBean;
    try {
        // Get the information from the database via the Enterprise Bean
        System.out.println("Generating the Liver: " + componentID + " " + parentID);
        InitialContext ctx = new InitialContext();
        bioMightBean = (BioMightBeanLocal) ctx.lookup("java:global/BioMightWeb/BioMightBean!biomight.ejb.BioMightBeanLocal");

        double radius = 0.001;

        if (componentID.equals("Liver01"))
        {
            // Generate the GreaterCurvature of the stomach
            // Create 5 sections
            double[] startPos = {2.0, -17.75, -2.25};
            double[][] currentPoints = BioMightBean.createCylinderInPlane(Constants.XPLANE, startPos, radius, 8);

            int success = bioMightBean.generateLiver("LiverEpithelium00001", "LiverEpithelium",
                "LiverEpithelium", componentID, parentID, currentPoints);

        }
        else if (parentID.equals(""))
        {
        }

        System.out.println("Created LiverEpithelium info using EJB");
    } catch (Exception e) {
        System.out.println("Exception Getting Components - LiverEpithelium");
        throw new ServletException("Remote Exception getComponents():", e);
    }
}

public void initProperties() {
    BioMightScale bioScale = new BioMightScale("0.00, 0.00, 0.00");
}
```

The screenshot shows the BioMight IDE with the following details:

- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard IDE icons for file operations, editing, and running.
- Project Explorer:** Shows the project structure with folders like 'src' and 'test'.
- Editor:** Displays the code for `BioMightBeanRemote.java`. The `generate()` method is highlighted in blue.
 

```

      /*****
      *
      * GENERATE LIVER
      *
      * @param parentID
      * @param componentID
      *****/
      public void generate(String parentID, String componentID)
      {
          // Generate the Liver: Endothelium
          BioMightBeanRemote bioMightBean;
          try {
              // Get the information from the database via the Enterprise Bean
              System.out.println("Generating the Liver: " + componentID + " " + parentID);
              InitialContext ctx = new InitialContext();
              bioMightBean = (BioMightBeanRemote) ctx.lookup("biomight.ejb.BioMightBeanRemote");

              double circumference = 0.25;

              if (componentID.equals("Liver:01"))
              {
                  // Generate the GreaterCurvature of the stomach
                  // Create 5 sections
                  double[] startPos = {3.5, -17.5, -1.75};

                  // Create a equilateral octogon
                  double x = startPos[0];
                  double y = startPos[1];
                  double z = startPos[2];

                  double[][] currentPoints = {
                      {x, y, z},
                      {x, y-circumference, z-circumference},
                      {x, y-(circumference*2), z-circumference},
                      {x, y-(circumference*3), z},
                      {x, y-(circumference*3), z+(circumference)},
                      {x, y-(circumference*2), z+(circumference*2)},
                      {x, y-circumference, z+(circumference*2)},
                      {x, y, z+(circumference*2)}
                  };

                  System.out.println("Calling Generate Liver: " + componentID + " " + parentID);
                  int success = bioMightBean.generateLiver("LiverEpithelium:00001", "LiverEpithelium",
                      "LiverEpithelium", componentID, parentID, currentPoints);
              }
          }
      }
      
```
- Bottom Panel:**
  - Tasks:** #6 Servers, #5 Problems (22), #4 Progress, #3 Search, #2 Console.
  - Console:** Shows 3 errors, 11,377 warnings, 0 others (Filter matched 105 of 11382 items).
  - Table:** A table with columns: Description, Resource, Path, Location, Type.

The screenshot shows the Eclipse IDE interface. The top bar includes the menu bar (File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help) and the toolbar. The left sidebar contains the Project Explorer, which shows the project structure of BioMightWeb. The main editor window displays the source code of the BioMightWeb class. The code includes a main method and a generateLive method. The generateLive method contains a complex loop that generates a live matrix based on the number of components. The code is written in Java and uses the BioMightWeb class for generating live matrices.



## Putting it together

Below, are the steps needed to create, or overwrite a new BioMight component.

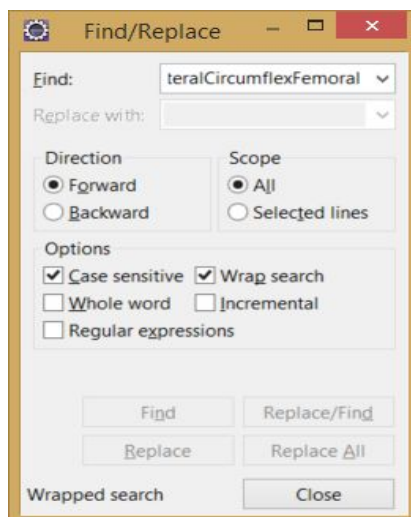
### BioMight Development Steps

1. Make a copies of **GreatSaphenousVeins** and **GreatSaphenousVein** and rename to new Veins in proper location.
2. Use Find/Replace to update the code to new names. Case Sensitive Search and Replace, do upper and lower.
3. Update references in **Constants.java** if needed. (**BioMightUtility** project)
4. Set the *bGenerate* flag to true, so that it generates the 3D model
5. Set the Starting Coordinates and Cylinder
6. Add the *generateXXXX()* method **BioMightVascularBeanLocal** (Copy and paste from corresponding Artery)
7. Add the *generateXXXX()* method to **BioMightVascularBean** (Copy and paste from corresponding Artery)
8. Rename the references in the *generateXXXX()* method for the new vein
9. Set up some initial instructions (Translate, Rotate, Scale, etc)
10. "Seed the database" Use the scripts to add the records to the **BioComp**, **BioGroup** table
11. Add the new object to the Veins class. Do a search and create using the **GreatSaphenousVeins** as a guide
12. Export the EJB and/or Utility packages as JAR files using the **Export** Command while mousing over project
13. Run it and look for results
14. When done, set the bGenerate flag back to false.

## ***BioMight Web Code***

Pick the Java class that you are going to make your copy, or duplicate from. Copy and paste the Collection class and the Child class using the Ctrl-C and Ctrl-V or mouse-based copy and paste. This will give you two new files. If older versions of the files exist, delete them.

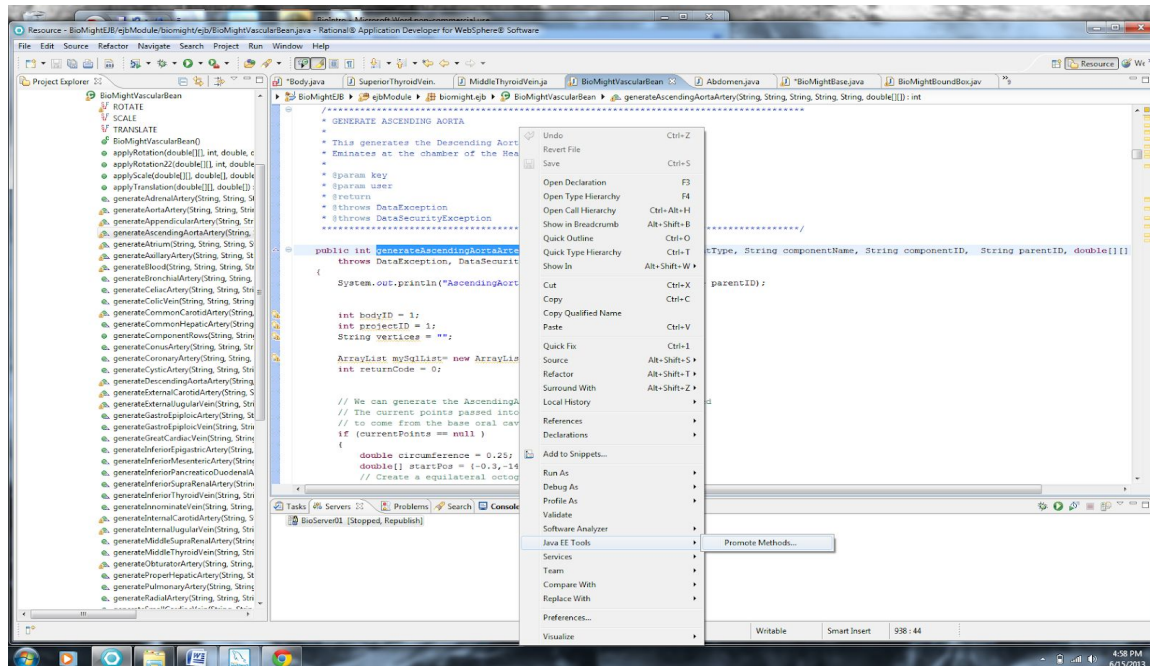
Perform a rename operation by using the Ctrl-F option which brings up the following Dialog window. Put the name of the Old Class in the Find box, and the new Java Class name in the Replace box. Use the “Replace All” feature and will do everything at once. Always make sure the “*Case Sensitive Flag*” is on as shown below. You will have to perform a find and replace on the lowercase name in the Collection classes only.



Each BioMight component has a *generate()* method. Ensure that you set the *bGenerate* flag to **true** when working on a component so that its *generate()* method is called. Also make sure the call to *getComponents()* has the proper **Constant** defined. As a later note, the constant, defined in **Constants.java**, will match up with the *comp\_type* field/element in the SQL **biocomp** database table

## EJB Client Code

Set up the remote method call in the **MuscularEJBClient**, **VascularEJBClient**, or **SkeletalEJBClient**. This mechanism allows the Web Client Java classes to talk to EJB beans.



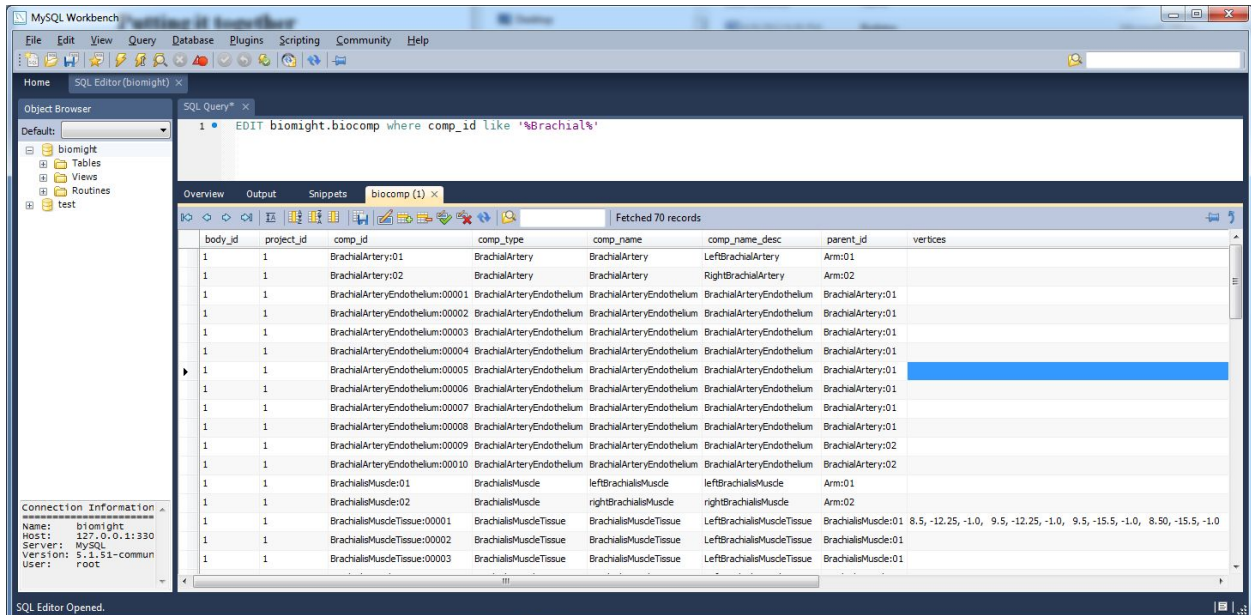
## EJB Code

Add the generateXXXX() method to the proper EJB class whether it be **MuscularBean**, **VascularBean**, **SkeletalBean**, etc. If you are working on a vein, make a copy of the generateArtery as veins and arteries pretty much follow the same path.



## Database Rows

**Component Table** - There needs to be a record for each instance of BioMight component that you are creating. Below we see an example of the Brachial Artery. Also, we see the Endothelium row that the *generate()* method constructed.



MySQL Workbench

SQL Editor (biomight) x

SQL Query\*

```
1 • EDIT biomight.biocomp where comp_id like '%Brachial%'
```

Overview Output Snippets biocomp (1) x

Fetches 70 records

body_id	project_id	comp_id	comp_type	comp_name	comp_name_desc	parent_id	vertices
1	1	BrachialArtery:01	BrachialArtery	BrachialArtery	LeftBrachialArtery	Arm:01	
1	1	BrachialArtery:02	BrachialArtery	BrachialArtery	RightBrachialArtery	Arm:02	
1	1	BrachialArteryEndothelium:00001	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00002	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00003	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00004	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00005	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00006	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00007	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00008	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:01	
1	1	BrachialArteryEndothelium:00009	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:02	
1	1	BrachialArteryEndothelium:00010	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArteryEndothelium	BrachialArtery:02	
1	1	BrachialisMuscle:01	BrachialisMuscle	leftBrachialisMuscle	leftBrachialisMuscle	Arm:01	
1	1	BrachialisMuscle:02	BrachialisMuscle	rightBrachialisMuscle	rightBrachialisMuscle	Arm:02	
1	1	BrachialisMuscleTissue:00001	BrachialisMuscleTissue	BrachialisMuscleTissue	LeftBrachialisMuscleTissue	BrachialisMuscle:01	8.5, -12.25, -1.0, 9.5, -12.25, -1.0, 9.5, -15.5, -1.0, 8.50, -15.5, -1.0
1	1	BrachialisMuscleTissue:00002	BrachialisMuscleTissue	BrachialisMuscleTissue	LeftBrachialisMuscleTissue	BrachialisMuscle:01	
1	1	BrachialisMuscleTissue:00003	BrachialisMuscleTissue	BrachialisMuscleTissue	LeftBrachialisMuscleTissue	BrachialisMuscle:01	

Connection Information

Names: biomight  
Host: 127.0.0.1:3306  
Server: MySQL  
Version: 5.1.51-community  
User: root

SQL Editor Opened.

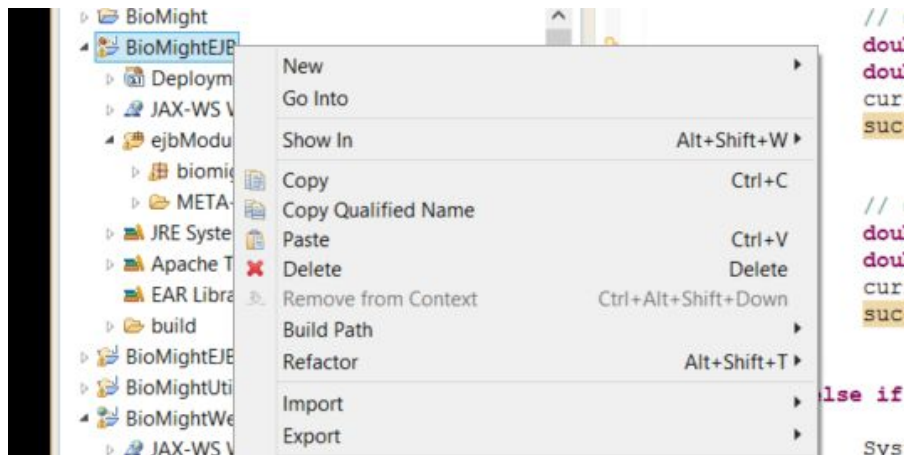
**Group Table** – This stores the relationship from a view perspective. For instance, there is only one Brachial Artery in the left arm of the human body. But, we can access the Brachial Artery from the Arm.java or from Arteries.java. The parent to child relationship is stored in the group table. If the rows are not in the group table, the component database table rows will not be picked up in the *getComponents()* method query in the base **BioMightBean** EJB.



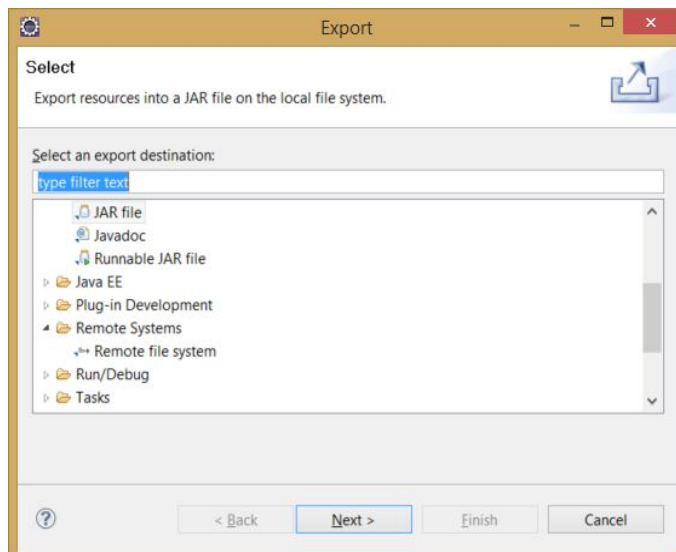
## Exporting the Libraries

Whenever you make changes to the **BioMightEJB** methods, such as *generateFemoralArtery()*, where you would update the instructions for graphic operations such as Rotate, Scale, Translate, etc. you will need to export and deploy the library so that your changes appear in the **BioMight** application.

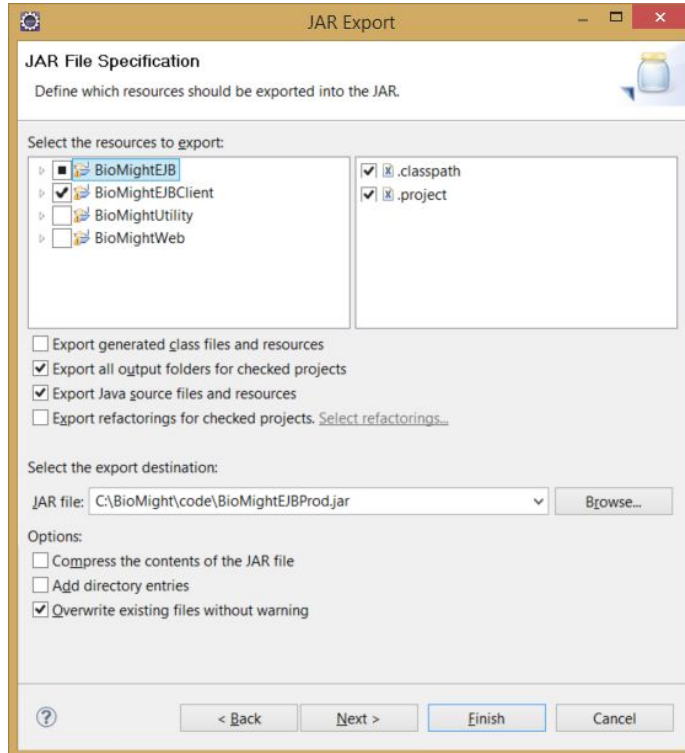
Start by moving your mouse over the BioMightEJB project and click the mouse to select the Export option as shown below.



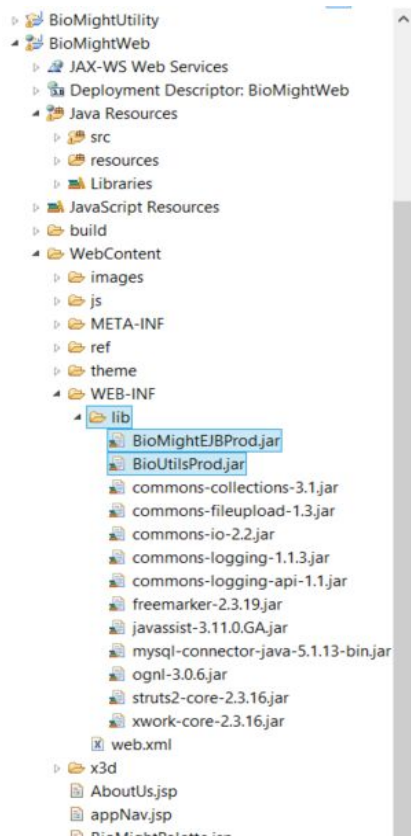
Select **JAR file**. It should be preselected, so you may just press “Next”.



Click on the **BioMightEJBClient** library as that needs to be exported as well.



Grab the file using Windows Explorer and copy it to here. They reside in the **Lib** folder.



After making changes to the code, or updating a library, wait until the server has been updated with the latest code. Watch for the little green status bar that says “working” to complete. Below, we see that our server is not in the proper state, so a restart is needed.



Now, we see that the server is Started and Synchronized. You are ready to test your code.

