# Become a better developer creating Bullhorn

Contents

[Become a better developer creating Bullhorn 1](#_Toc467677598)

[Become a Better Developer 8](#_Toc467677599)

[How to set up your development environment 9](#_Toc467677600)

[Download the Oracle Virtual Machine 9](#_Toc467677601)

[Install Eclipse in the Virtual Machine 9](#_Toc467677602)

[How to test Your Oracle Installation 9](#_Toc467677603)

[Prework: Recommended Basic Java to get you started 11](#_Toc467677604)

[Using Git to backup and share your files between different computers 12](#_Toc467677605)

[How to install Git on Redhat Linux 12](#_Toc467677606)

[Create a GitHub Account 12](#_Toc467677607)

[Make a Repository on GitHub 12](#_Toc467677608)

[Git Walkthrough 12](#_Toc467677609)

[After you make changes to your project: 13](#_Toc467677610)

[Checking Status 13](#_Toc467677611)

[A Basic Git Workflow 13](#_Toc467677612)

[What is MVC? 15](#_Toc467677613)

[Model 15](#_Toc467677614)

[View 15](#_Toc467677615)

[Controller 15](#_Toc467677616)

[Validation 16](#_Toc467677617)

[How is MVC implemented in Bullhorn? 16](#_Toc467677618)

[What is Object-Oriented Programming? 17](#_Toc467677619)

[Benefits of Object-Oriented Programming 17](#_Toc467677620)

[Object-Oriented Programming Concepts 18](#_Toc467677621)

[Work with a Database 20](#_Toc467677622)

[What is a database? 21](#_Toc467677623)

[What can we do with a database? 21](#_Toc467677624)

[What is Structured Query Language (SQL)? 21](#_Toc467677625)

[Rows and Fields 22](#_Toc467677626)

[Primary Keys and Foreign Keys 22](#_Toc467677627)

[Normalization 23](#_Toc467677628)

[Null Values 25](#_Toc467677629)

[Creating Tables 25](#_Toc467677630)

[Joining Tables 26](#_Toc467677631)

[Using Oracle and Java Together 27](#_Toc467677632)

[Connecting to Oracle from Eclipse 27](#_Toc467677633)

[Using Java Prepared Statements 29](#_Toc467677634)

[Oracle 12c Autoincrement with Identity 29](#_Toc467677635)

[Create Bullhorn Tables in Oracle 31](#_Toc467677636)

[Add Test Data to Bullhorn Database 33](#_Toc467677637)

[Commit your Database Changes 34](#_Toc467677638)

[Add your Database scripts to your Java Project 34](#_Toc467677639)

[How to create a dynamic web application using Eclipse 35](#_Toc467677640)

[Create a basic application to display form data 36](#_Toc467677641)

[Create an HTML Form 36](#_Toc467677642)

[Create a Servlet 37](#_Toc467677643)

[Create a page to display the output of your form 37](#_Toc467677644)

[Passing Parameters 37](#_Toc467677645)

[How to get a value from the form to the servlet and then to the output page 37](#_Toc467677646)

[Request Parameters 37](#_Toc467677647)

[Getting the form data into the servlet 38](#_Toc467677648)

[How will the Servlet retrieve the Parameters? 38](#_Toc467677649)

[Should I use doGet() or doPost() 38](#_Toc467677650)

[How will the Servlet set the values in your output page? 38](#_Toc467677651)

[How to Make the Servlet go to the Next Page 40](#_Toc467677652)

[Create a JSP that Displays the values from a Form 41](#_Toc467677653)

[What is a JavaBean? 41](#_Toc467677654)

[How to make a JSP secure and dynamic 41](#_Toc467677655)

[How to include JSTL as part of your JSP 41](#_Toc467677656)

[How to import a class you’ve created into your JSP? 41](#_Toc467677657)

[How to determine if a String is null or empty using JSTL? 42](#_Toc467677658)

[Other JSTL tags 42](#_Toc467677659)

[How to include a block of Java Statements in a JSP 43](#_Toc467677660)

[How to retrieve the value of parameters passed to a JSP 43](#_Toc467677661)

[Web Page Structure 45](#_Toc467677662)

[Basic Parts of a Web Page 45](#_Toc467677663)

[The Head 46](#_Toc467677664)

[The Body 46](#_Toc467677665)

[Designing Web Pages with HTML5 47](#_Toc467677666)

[HTML5 tags 47](#_Toc467677667)

[HTML5 Attributes 47](#_Toc467677668)

[Name-value pairs 47](#_Toc467677669)

[The Id Attribute 47](#_Toc467677670)

[The name Attribute 48](#_Toc467677671)

[A basic HTML5 document 48](#_Toc467677672)

[Explanation of common tags 48](#_Toc467677673)

[HTML Forms 50](#_Toc467677674)

[HTML5 Comments 50](#_Toc467677675)

[An HTML Table 50](#_Toc467677676)

[Minimal HTML Table: 50](#_Toc467677677)

[How to navigate between web pages 51](#_Toc467677678)

[Styling Web Pages with CSS 52](#_Toc467677679)

[Why use CSS? 52](#_Toc467677680)

[How CSS works 52](#_Toc467677681)

[Span and Div Tags 53](#_Toc467677682)

[Attributes 53](#_Toc467677683)

[When to use ID and when to use class 54](#_Toc467677684)

[Making Web Pages Do Something with JavaScript 55](#_Toc467677685)

[What is JavaScript? 55](#_Toc467677686)

[Where is JavaScript defined in the document? 55](#_Toc467677687)

[What is the DOM? 55](#_Toc467677688)

[Using JavaScript to Validate a Web Form 56](#_Toc467677689)

[Fun with Javascript: 56](#_Toc467677690)

[Counting and limiting the amount of text in a textbox to 141 characters 57](#_Toc467677691)

[Ensure a post contains data 57](#_Toc467677692)

[Getting started with Bootstrap 59](#_Toc467677693)

[Responsive Web Design 59](#_Toc467677694)

[Where to get Bootstrap 60](#_Toc467677695)

[Connecting to the Database from Java with JPA 62](#_Toc467677696)

[Why use JPA? 62](#_Toc467677697)

[What are entities? 62](#_Toc467677698)

[What is a persistence unit? 62](#_Toc467677699)

[Develop a JPA application which connects to the database and displays a record 62](#_Toc467677700)

[Create a DbUtilities Class 63](#_Toc467677701)

[Creating the DbUser Class 63](#_Toc467677702)

[Determining if a user is valid based on email/password 64](#_Toc467677703)

[Get a user by email address 65](#_Toc467677704)

[Selecting a user by primary key with JPQL 65](#_Toc467677705)

[Parameterized queries – select a user given their email address 65](#_Toc467677706)

[Inserting a user with JPQL 66](#_Toc467677707)

[updating a user with JPQL 66](#_Toc467677708)

[The Stateless Nature of the Web 67](#_Toc467677709)

[Understanding Sessions and Requests 67](#_Toc467677710)

[How to add a User to the Session 69](#_Toc467677711)

[To add a value to the session for use by other servlets and JSPs: 69](#_Toc467677712)

[To read a value from the session 69](#_Toc467677713)

[Getting a User from the Session 70](#_Toc467677714)

[How to use the Java mail API to send email to the support team 71](#_Toc467677715)

[Add the Java Mail API to your application 71](#_Toc467677716)

[Import some libraries 71](#_Toc467677717)

[Get a mail session 71](#_Toc467677718)

[Create a message object 72](#_Toc467677719)

[Address the message object 72](#_Toc467677720)

[Send the message object 72](#_Toc467677721)

[How to use Gravatar to display user’s avatars with posts 73](#_Toc467677722)

[How to display the Gravatar 73](#_Toc467677723)

[Setting the size of the Gravatar image 73](#_Toc467677724)

[Test Gravatars you can use 73](#_Toc467677725)

[Default image 73](#_Toc467677726)

[Site to create temporary email addresses (for creating test Gravatars) 73](#_Toc467677727)

[Random user photos for creating test Gravatars https://randomuser.me/photos 74](#_Toc467677728)

[How does Gravatar Work in Bullhorn? 74](#_Toc467677729)

[How to generate an MD5 hash in Bullhorn 74](#_Toc467677730)

[Create other Bullhorn Pages and Servlets 76](#_Toc467677731)

[Create and Style the Login Page 76](#_Toc467677732)

[Create the Login Servlet 76](#_Toc467677733)

[How to validate the user 76](#_Toc467677734)

[How the log out button works 77](#_Toc467677735)

[How to redirect the user to the next page 77](#_Toc467677736)

[Forward 77](#_Toc467677737)

[Redirect 77](#_Toc467677738)

[Passing an attribute or value to your JSP 78](#_Toc467677739)

[Create and Style the Home Page 78](#_Toc467677740)

[Move the Bootstrap links, NavBar and Footer to include files 78](#_Toc467677741)

[Create the Post Servlet 78](#_Toc467677742)

[Create and Style the Newsfeed Page 79](#_Toc467677743)

[Create the Newsfeed Servlet 79](#_Toc467677744)

[Create and Style the User Profile Page 79](#_Toc467677745)

[Create the Profile Servlet 80](#_Toc467677746)

[Create the add user page 80](#_Toc467677747)

[Create the add user servlet 80](#_Toc467677748)

[Create and Style the Support Page 80](#_Toc467677749)

[Create the Support Servlet 80](#_Toc467677750)

[Create and Style the Error Page 81](#_Toc467677751)

[Create the Error Servlet 81](#_Toc467677752)

[Send Email to the Administrator when an error occurs 81](#_Toc467677753)

[Wrapping Up 81](#_Toc467677754)

# Become a Better Developer

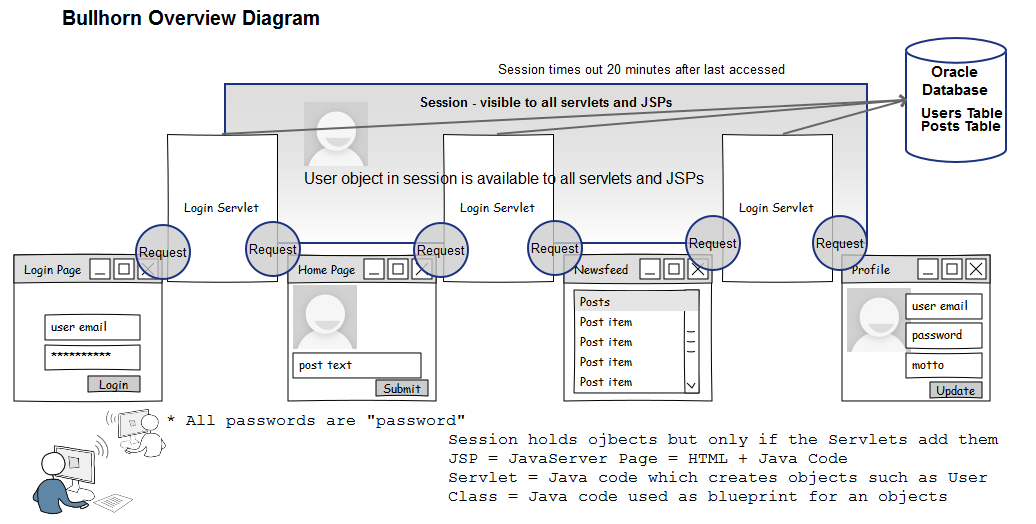
* Learn Java EE (Enterprise Edition)
* Learn Best Practices
* Learn Practical Tips and Advice from the field

Develop Bullhorn and learn how Java EE applications are structured and how they work. We’ll explore key issues in Enterprise web development.

Bullhorn is a website that allows users to post messages to each other. Each message is limited to 141 characters. Messages do not contain images. When messages are posted the user’s Gravatar is posted next to the message. Clicking on the Gravatar will take one to the public profile for the user. Users clicking on their own image can edit their profile. Site design is enhanced using the Bootstrap framework allowing you to easily create a professional looking website.The site will include a navigation bar, tables, headers, images, and other common features found in the latest websites.

Bullhorn users can read messages from all users even without logging into the site. Users who log in to the site can also post messages and edit their profile.

All posts and user data will be stored in an Oracle database. The application will access the database using JPA (Java Persistence API). This makes it easy for the developer to not only access a single database but also switch databases without rewriting code.



# How to set up your development environment

## Download the Oracle Virtual Machine

We use a pre-built Oracle virtual machine (VM) image to develop the site. Oracle makes this VM available for download at no cost (registration required). The VM requires the open source VirtualBox software to be installed on your computer. VirtualBox is also made by Oracle and available to you at no cost.

The VM hosts the latest version of the Oracle database (version 12c). To use it for class you need to install Eclipse and git on it and run it in Virtual Box. This page simply tells you how to find the version of the virtual machine. Since this is not a class in creating virtual machines this material will not be taught.

<http://www.oracle.com/technetwork/database/enterprise-edition/databaseappdev-vm-161299.html>

## Install Eclipse in the Virtual Machine

Eclipse is an IDE (Integrated Development Environment) that simplifies the process of developing Java applications. You can download the latest version of Eclipse as well as read the installation instructions on their website at [www.eclipse.org](http://www.eclipse.org) . The installation instructions can be found at <https://wiki.eclipse.org/Eclipse/Installation>. If you’re using the Oracle virtual machine discussed above then you already have Java installed. You simply need to download Eclipse and extract it to a directory.

## How to test Your Oracle Installation

*Note: These commands are necessary only for the virutal machine we have set up. Consult Oracle’s documentation for details about how to start the database on your particular computer.*

Run the following commands to ensure Oracle is set up properly. In the end you should be able to create a query in SQL Developer.

Open the terminal and enter the following commands:

Note: there is a space between the first dot and the tilde in the first command. Also, there is a space on either side of the / in sqlplus command.

. ~/.bash\_profile

lsnrctl start

sqlplus / as sysdba

sql>startup

* Then open SQL Developer (link on Desktop))
* Select orcl connection from the connections on the left
* From SQL Developer, enter the  password: oracle
* In  SQL Developer,  type: select user from dual;  
    
  You should see the user name

# Prework: Recommended Basic Java to get you started

If you have a strong Java background you can continue. If not, we recommend knowing the following topics prior to developing the Bullhorn application.

* Create a basic Java program in Eclipse
* Working with the Scanner object to Get input from the user
* Validate input data
* Create a program which implements sequence, decision making and repetition
* Work with data types - Strings, numbers and boolean
* Dividing your Program into Methods
* Passing parameters to methods
* Test and debug an application
* Handle exceptions
* Define and use a class
* Move your method to a separate class
* Pass a parameter to a method in a class
* Set class variables and call a method
* Create a Constructor in your Class
* Work with inheritance, Polymorphism and Encapsulation
* Work with Collections and ArrayLists
* Working with Dates
* Get today’s date in Java
* Formatting a Date
* Get the number of days in a month
* Formatting Numbers
* Create a Date and Number Utility Class
* Create and use a Unit Test with JUnit

# Using Git to backup and share your files between different computers

There are two components to using Git: a server for offsite sharing of your files and a client which allows you to use Git locally.

Clients are available for all major operating systems. The most common client is to use the Git client at the command line. You can download the software from GitHub.com which I’ll talk about more in a moment.

The Git command line client works on most platforms such as Windows, linux or Mac. You can also use Git clients from Atlassian found at Sourcetreeapp.com. To download that requires an email signup but is otherwise free.

You may also find Git to be integrated in your favorite IDE such as Eclipse. When you want to save your work remotely you’ll need a server. During this course I’ll focus on GitHub.com. It’s the largest code repository in the world.

GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project.

## How to install Git on Redhat Linux

* Open a terminal
* Use su command to become the root user
* As root, type yum install git
* Once git is installed then exit the root terminal with the exit command.

## Create a GitHub Account

* Go online at <http://www.github.com> and follow the online form to create a GitHub account

## Make a Repository on GitHub

* Click the + sign button on the upper right of your GitHub home page to create a new repository.

## Git Walkthrough

From Eclipse locate the Project directory by right-clicking on the project folder   
and selecting Properties from the menu

1. sign into github.com  
2. create new repository called <projectName>  
3. open the terminal on your pc and change to the project directory  
4. run the following commands at the prompt (indicated by #)

git init

git remote add origin https://<ghusername>@github.com/<ghusername>/<projectName>

git add --all

git commit -m "initial commit"

git push -u origin master

The project files will now be on GitHub.

### After you make changes to your project:

git add --all  
git commit -m "commit message"  
git push  
git status git push -u origin master

## Checking Status

At any time in the Git directory you can type

git status

to check the status of your repository.

Most of the time you'll see something like this:

MyProject>git status

# On branch master

nothing to commit (working directory clean)

When there are changes to your files or changes to be added to your repository you'll get messages about those as well.

## A Basic Git Workflow

Here is a detailed look at the basic workflow you may wish to follow:

1. First, initialize your repository
2. Next create, add and commit a readme file. You can use markup for formatting this file.
3. Then Create/checkout a branch. You need the previous commit for this to work.
4. Then add or modify your files.
5. Next Stage your changes.
6. Then Review/Test your changes.
7. Next you commit and finally you merge then delete the branch.
8. Pull the latest version from github to make sure you've created no conflicts
9. Resolve any conflicts/add/commit
10. Push your version to GitHub

The objective of this workflow is that you are always working on a branch. When things are working you simply merge the branch back into the master branch. When things go wrong you simply delete the branch and start over, without negatively impacting the rest of your project.

When working on a team, one person should create the project directory and add a readme file:

mkdir ProjectDir

cd ProjectDir

echo "#This is the ProjectName >>readme.md

echo "The team members are John, George, Paul and Ringo" >>readme.md

Then iteratively perform the following:

1. clean project  
2. git pull   
3. git add --all  
4. git commit - m "xxx"   
5. git push -u origin master

# What is MVC?

The model-view-controller (MVC) pattern is an architectural pattern used primarily in creating Graphic User Interfaces (GUIs). The major premise of the pattern is based on modularity and it is to separate three different aspects of the GUI: the data (model), the visual representation of the data (view), and the interface between the view and the model (controller). The primary idea behind keeping these three components separate is so that each one is as independent of the others as possible, and changes made to one will not affect changes made to the others. In this way, for instance, the GUI can be updated with a new look or visual style without having to change the data model or the controller.

Newcomers will probably see this MVC pattern as wasteful, mainly because you are working with many extra objects at runtime, when it seems like one giant object will do. But the secret to the MVC pattern is not writing the code, but in maintaining it, and allowing people to modify the code without changing much else. Also, keep in mind, that different developers have different strengths and weaknesses, so team building around MVC is easier. Imagine a View Team that is responsible for great views, a Model Team that knows a lot about data, and a Controller Team that see the big picture of application flow, handing requests, working with the model, and selecting the most appropriate next view for that client.

One of the great advantages of the Model-View-Controller Pattern is the ability to reuse the application's logic (which is implemented in the model) when implementing a different view. A good example is found in web development, where a common task is to implement an external API inside of an existing piece of software. If the MVC pattern has cleanly been followed, this only requires modification to the controller, which can have the ability to render different types of views dependent on the content type requested by the user agent.

## Model

Generally, the model is constructed first. The model has two jobs: it must both store a state and manage subscribers. The state does not need to be anything special; you simply need to define how you're going to store data, with setters and getters.

## View

Once you write a data model, the next easiest thing to write is usually a view. The view is the part of the application which subscribes to a model. Usually it presents it to a user alongside a user interface, or GUI. The GUI contains other components too, which are usually part of the controller and can be handled later.

## Controller

The rest of the GUI -- the parts which do not update when the model changes -- are the responsibility of the controller. This includes navigating around the view, as well as what you do when someone tries to edit the data in the view. Strictly speaking, a view cannot be edited and is 'read-only' -- when you try to modify a field in the view, the controller needs to pick up the editing event, process it, and send it to the model; the model will then update the view if/when the value actually changes.

## Validation

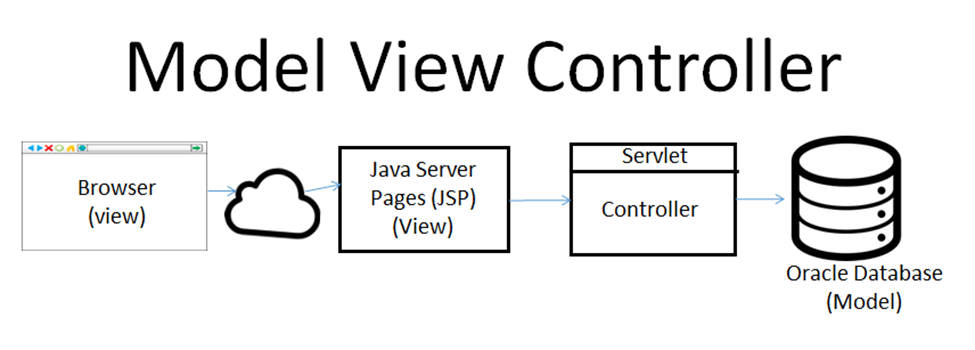
When possible, it is usually best to allow the model to do all the necessary validation of values, so that any changes to the allowed values, or changes simply to the validation process, only need to be made in one place. However, in some languages under certain circumstances, this may not be possible. For instance, if a numeric property is being edited with a text field, then the value of the property passed to the controller by the view will be text, not a number. In this case, the model could be made to have an additional method that takes text as the input value for this property, but more likely, the controller will do the initial parsing of the text to get the numeric value, and then pass it on to the model to do further validation (for instance, bounds checking). When either the controller or the model determines that a passed in value is invalid, the controller will need to tell the view that the value is invalid. In some cases, the view may then issue an error dialog or other notification, or it may simply revert the value in its editor to the older valid value.

## How is MVC implemented in Bullhorn?

**Model** – Oracle database running on the same computer but could easily be moved to a different computer

**View** – JSP Pages using Bootstrap displayed in the user’s browser

**Controllers** – Servlets running in Tomcat container. Also includes the User and Post objects which are written to the database using JPA



# What is Object-Oriented Programming?

Object-Oriented Programming is a programming methodology. It allows you to divide the functionality of your program into a collection of interacting parts. In many cases, each part may represent a real life object such as a car, a shape, a chess piece or an invoide. Often these objects will contain traits of their real life counterparts.

To allow your program to create an object you first put your code into a module called a class. A class contains variables to hold the data of your object such as color, size, name, description, etc... The class also contains methods which operate on the variables or other objects (other code). That way your object can be interactive. For example, a chess piece can move, jump, or get captured. A light can turn on or off. A car can drive at a particular speed or it can stop. These actions would be implemented by methods. The methods would change the data inside the class which represents the state of the class.

#### An class is a container your define in your code. It is the combination of:

* Some data (i.e. variables) that are related to each other
* Procedures specifically designed to work with that data
* Often the variables would be declared as private so they could only be changed by the methods. This insures all programmers interact with your object in the same manner.

### Benefits of Object-Oriented Programming

The goal of object-oriented programming is to divide the programming problem into units that are easy to comprehend, develop and test. Rather than putting all your code in one module and making that code accessible to everyone you put your code into many smaller classes and write additional code to enable the classes to communicate with each other. This produces the same output. More efficiently.

* saves you time and effort by dividing tasks nicely
* enables code reuse
* enables teamwork
* you can (and often do) use objects in your program which you do not understand fully how they were designed
* implementation details of an object are hidden from the user of the object. Furthermore, an object can have private code that is not accessable from the consumer of the object. This prevents the consumer from “getting under the hood” of your object and “breaking” it
* Design Benefits: Large programs are very difficult to write. Object Oriented Programs force designers to go through an extensive planning phase, which makes for better design.
* Larger programs are easier to develop with object-oriented principles than smaller programs.
* Software Maintenance: An Object Oriented Program is much easier to modify and maintain than a non-Object Oriented Program. So although a lot of work is spent before the program is written, less work is needed to maintain it over time.

A program that creates invoices might contain an invoice object. The invoice object might further contain things real invoices actually contain: customer, line item, date. Any of these can be objects.

The idea that something like an invoice, date or customer being an object in your program is that all the code for working with that item is encapsulated into a reusable module called a class. When you need to work with that object elsewhere in your program you simply create an instance of the object and set the properties of it. Then all the methods that work with those properties can now work in your program. And the code you wrote inside the object can be reused multiple times. If you have three customers or three-thousand, you only have to write the code once. For even greater efficiency let someone else write the object as has happened with the Java Date class.

You develop your program by focusing on the interaction of the objects. Each object contains attributes and methods to get or set those attributes. Each object may also contain methods that allow it to perform some task or interact with other objects. You will create the object, write code in each method and also write code for each object to call the other’s methods. Each object is asking the other to ‘do something’, ‘remember something’ or ‘tell me something’. Just like people working together in real life.

The object-oriented paradigm allows very large projects to be divided into many objects. Each object has its own single responsibility or job. Various developers on a team can focus on the simultaneous development of various objects. When they all come together the work of the project is accomplished.

### Object-Oriented Programming Concepts

**Encapsulation**: hides details and direct access to your data. You hide how you are manipulating that data. Encapsulation is drawing a line between "details I'll take care of here" and "details that everyone else has to know about."

**Inheritance**: To derive something from another. Inheritance prevents code duplication.

**Polymorphism**: Changing from one type another more specific type. The more specific type can do anything the general type can - and more. If a circle object inherits a shape object then any piece of code that takes a circle object can also take a shape object. All traits and behaviors of shape are available in circle.

Procedure-oriented programming uses procedures to make code easier to write and to understand, Object-oriented programming (OOP) goes a step further and uses objects to make code easier to create and work with. The objects themselves can contain procedures. They are called methods. The procedures work with data which other objects or code can set using getter and setter methods.

# Work with a Database

* SQL Joins (venn diagram)
* Querying the Data
  + selecting all fields from one table
  + select specific fields from one table
  + adding criteria
  + adding order
  + calculated fields (gravatar url)
  + querying multiple tables
* inserting data into a table
* inserting related data
* updating records in the database
* updating related data
* working with dates in the database
* Connecting to the Database from Java with JDBC
* The difference between java.sql.Date and java.util.Date
* Create a DateUtilities class
* Summarizing data with aggregate queries

## What is a database?

A database is a place to permanently store data in a safe, efficient way which meets the principles of the ACID acronym.

* Atomicity. In a transaction involving two or more discrete pieces of information, either all of the information is saved or none is saved.
* Consistency. A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.
* Isolation. A transaction in process and not yet committed must remain isolated from any other transaction.
* Durability. Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state.

## What can we do with a database?

* Create data and save it permanently
* Retrieve previously created data
* Update existing data
* Delete existing data

Java contains features to allow us to work with the data but when we aren’t using it then we need it in a format and location that we can get back to it when we need it again. We retrieve our data and add more data using SQL or Structured Query Language. This is a specific language used to do things (query) with the data. SQL allows you to selecting records created between particular dates or updating records that match a given criteria.

## What is Structured Query Language (SQL)?

SQL (pronounced as the letters SQL or sequel) is an abbreviation for Structured Query Language. It is a specialized database language that consists of statements that are very close to English.

SQL has one purpose: to communicate with a database. We communicate with the database to add, exit or delete data. We also communicate with the database to create and modify tables and other database objects.

* SQL stands for Structured Query Language
* SQL allows you to access a database
* SQL can execute queries against a database
* SQL allows you to
  + retrieve data from a database
  + insert new records in a database
  + delete records from a database
  + update records in a database
* SQL is not a propriertary language used by a specific vendor. Almost every major DBMS supports SQL. Learning SQL will enable you to interact with every database you’ll run into.
* SQL is easy to learn. The statements consist of of descriptive English words.
* SQL is powerful. Cleverly using the language elements allows you to perform complex database operations.

**SQL is a Standard - BUT....** SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements allow you to retrieve and update data in a database. SQL works with database programs like MS Access, IBM DB2, Informix, Microsoft SQL Server, Oracle, Sybase, etc.

**... There are many different versions of the SQL language** Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

To remain in compliance with the ANSI standard each must support the major keywords in a similar manner. This means that SELECT, UPDATE, DELETE, INSERT, WHERE all work in the same way on most databases.

## Rows and Fields

Your data is stored in tables. One row of a database table is a record containing related fields of a particular record. Each field is a column describing one attribute of the data.

## Primary Keys and Foreign Keys

Databases are very powerful tools that allow us to search and sort data at incredible speeds. An essential feature of a database is making each record unique. To make each record unique either use an existing field that is unique to each record or add a field to each record that contains a unique number. Oracle will generate this number for you using a sequence.

The unique column of each record becomes the primary key. Then when another table contains related data the second table will point back to the original by including the first table’s primary key field. In the second table the field is called a foreign key.

Foreign Keys define a directed reference from one table (the child) to another table (the parent). This reference acts as long as the involved columns of the two tables contain identical values. It joins one row of the child table to a single row of the parent table - a row of the parent table may be coupled by many rows of the child table.

SQL will make it easy to locate related records when you need them again.

## Normalization

In a normalized data structure, each table contains information about a single entity and each piece of information is stored in exactly one place.

Normalization is the process of efficiently organizing data in a database. This is done by organizing the columns (attributes) and tables (relations) of a relational database to minimize data redundancy. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table).

There are three steps to follow to properly normalize your database:

**First Normal Form - Atomic Data Test**

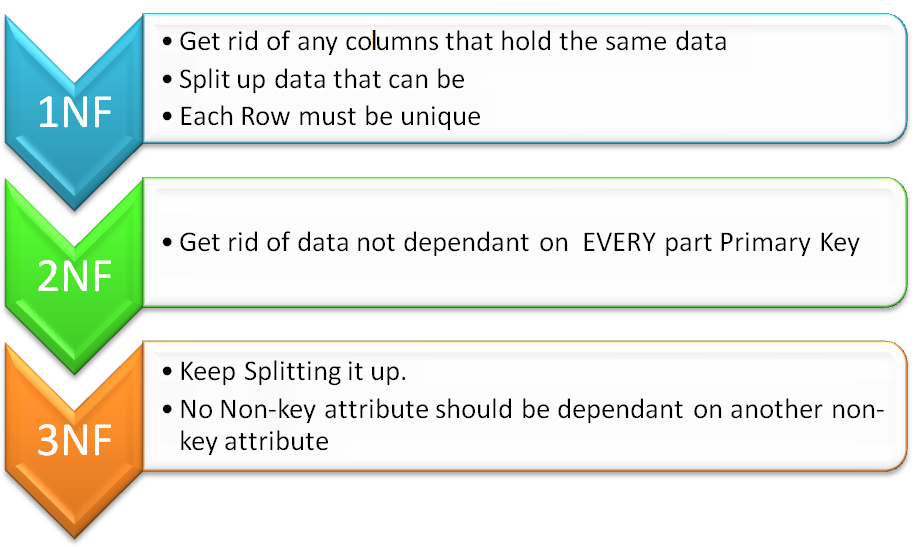
If a table has a primary key it is said to be in First Normal form if the table does not have repeating groups of attributes. All attributes within the table need to be dependent only on the [primary key](https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Operating_Systems,_Databases_and_Networking/Databases/Primary_keys). First normal form (1NF) sets the very basic rules for an organized database: Eliminate duplicate columns from the same table. Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

**Second Normal Form - Partial Dependence Test**

For a table to be in Second Normal form it must first be in First Normal (1NF) Form and then contain no data that is dependent on only part of the Primary Key. Second normal form (2NF) further addresses the concept of removing duplicate data: Meet all the requirements of the first normal form. Remove subsets of data that apply to multiple rows of a table and place them in separate tables. Create relationships between these new tables and their predecessors through the use of foreign keys.

**Third Normal Form - Non-Key Dependence Test**

For a table to be in Third Normal Form(3NF) it must be in Second Normal form and contain No data that is not dependent on the primary Key e.g. (Remove columns that are not dependent upon the primary key). Third normal form (3NF) goes one large step further: Meet all the requirements of the second normal form. Remove columns that are not dependent upon the primary key. If a remaining column ( attribute) plays a part in a functional dependency, all the attributes of the functional dependency are in the table, and the determinants of the attribute form a key, or all the dependent attributes are keying attributes. The latter means that another table which has the determinant attributes as a key, maps these determinant attributes against the dependent attributes , and these form a foreign key into this table.



## Null Values

Sometimes there is no value in a column of a row. In this case the column stores a NULL value. You can think of this as a flag to indicate the absence of data. NULL is different from the numeric value zero or a string with a length of zero characters. It is neither. It is nothing because no value has been stored in the column for this record. NULL will prove very useful. You can search for records that contain NULL.

## Creating Tables

In the introduction I said that you can do "Create Read Update Delete" operations to the data inside tables. How do you make the tables in the first place? By doing CRUD on the database schema, and the first SQL statement to learn is CREATE:

CREATE TABLE bhUser(

bhuserid number generated by default on null as identity,

username VARCHAR(50),

email VARCHAR(50),

motto VARCHAR(50),

joinDate date,  
 gravitarURL VARCHAR(50)  
);

CREATE TABLE bhPost(

bhpostid number generated by default on null as identity,

posttext VARCHAR(141),

postDate date,

bhuserid number

);

ALTER TABLE bhPost

ADD CONSTRAINT fk\_bhuserid

FOREIGN KEY (bhuserid)

REFERENCES bhUser(bhuserid);

The easiest way to run this is in SQLDeveloper. What you should see is:

Table bhUser created.

Table bhPost created.

Table bhPost altered.

## Joining Tables

Relationships area means to join data to different tables, while avoiding redundancy in the tables. Therefore, you can divide your data into different tables—using it only once—and then add it into other tables by establishing relationships.

There are three types of relationships:

• A One-to-one relationship is where each entry on table 1 can only have a relationship with one entry on table 2 and each entry on table 2 can only have a relationship with one entry on table 1. These types of relationships are rare.

• A One-to-Many relationship is where each entry on table 1 can have a relationship with multiple entries on table 2, but not vice versa.

• A Many-to-Many relationship is where each entry on table 1 can have a relationship with multiple entries on table 2 and each entry on table 2 can have a relationship with multiple entries on table 1.

The goal of referential integrity is to avoid having “orphaned” data. “Orphaned” data can happen when you are deleting or updating the data in your tables.

For example, if you have a customer on table 1 that is linked to payments on table 2 and then you delete that customer, then the payments linked to that customer on table 2 will become “orphaned” data.

Referential integrity prevents this by denying changes that will result in “orphaned” data.

Since you may have a valid need to delete or update data in your tables, you can choose to allow Access to update or delete data that have a relationship with the data you are updating or deleting.

One big reason for creating table relationships: creating relationships provides a foundation for establishing referential integrity, which will be discussed in a later section.

# Using Oracle and Java Together

## Connecting to Oracle from Eclipse

Create a Java Application and add the following code as a class. This will illustrate how to connect to Java from Eclipse. Make sure it works as shown below. Then modify the SQL Statement to retrieve the people and pets.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class TestOracleJDBC {

public static void main(String[] args) {

Connection con = null;

Statement stmt = null;

ResultSet rs = null;

String sql = "select user from dual";

try{

Class.forName("oracle.jdbc.driver.OracleDriver");  
 con = DriverManager.getConnection(

"jdbc:oracle:thin:ora1/ora1@localhost:1521:orcl");

stmt = con.createStatement();

rs = stmt.executeQuery(sql);

while(rs.next()){

System.out.println(rs.getString(1) + "\t");

//System.out.println(rs.getString(2));

}

}catch (SQLException e) {

e.printStackTrace();

}catch (ClassNotFoundException e) {

e.printStackTrace();

} finally {

try {

rs.close();

stmt.close();

con.close();

}catch(SQLException e){

e.printStackTrace();

}

}

}

}

## Using Java Prepared Statements

A PreparedStatement is an object that represents a precompiled SQL statement in your Java program.

Details about PreparedStatements can be found online at <https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html>

A SQL statement is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

In the following example of setting a parameter, con represents an active connection:

PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES

SET SALARY = ? WHERE ID = ?");

pstmt.setBigDecimal(1, 153833.00)

pstmt.setInt(2, 110592)

* The PreparedStatement class requires an import, java.sql.PreparedStatement
* Create the PreparedStatement object as shown above.
* Use question marks for the parameters. Then set the parameters in the order they appear as shown above.
* The PreparedStatement can be any valid SQL Statement.

As with Statement objects, to execute a PreparedStatement object, call an execute statement: executeQuery if the query returns only one ResultSet (such as a SELECT SQL statement), executeUpdate if the query does not return a ResultSet (such as an UPDATE SQL statement), or execute if the query might return more than one ResultSet object.

## Oracle 12c Autoincrement with Identity

Previous versions of Oracle did not have the equivalent of an AutoNumber or Identity function of other databases. The 12c database introduces the ability define an identity clause against a table column defined using a numeric type. The syntax is show below.

Using BY DEFAULT allows you to use the identity if the column isn't referenced in the insert statement, but if the column is referenced, the specified value will be used in place of the identity.

You can test the autoincrement with the following code. This won’t work in earlier versions of Oracle, just 12c. But it will generate two records with populated id fields.

create table identity\_test (  
id number generated by default on null as identity,  
description varchar(30)  
);

insert into identity\_test (description) values ('just testing 1');

insert into identity\_test (description) values ('just testing 2');

select \* from identity\_test;

drop table identity\_test;

## Create Bullhorn Tables in Oracle

CREATE TABLE "ORA1"."BHPOST"   
( "POSTID" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ,   
"POSTDATE" DATE DEFAULT NULL,   
"POSTTEXT" VARCHAR2(141) DEFAULT NULL,   
"BHUSERID" NUMBER DEFAULT NULL  
) ;  
--------------------------------------------------------  
-- DDL for Table BHUSER  
--------------------------------------------------------

CREATE TABLE "ORA1"."BHUSER"   
( "BHUSERID" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ,   
"USERNAME" VARCHAR2(50),   
"USERPASSWORD" VARCHAR2(50),   
"MOTTO" VARCHAR2(100) DEFAULT NULL,   
"USEREMAIL" VARCHAR2(100),   
"JOINDATE" DATE DEFAULT NULL  
) ;  
--------------------------------------------------------  
-- DDL for Index BHPOST\_PK  
--------------------------------------------------------

CREATE UNIQUE INDEX "ORA1"."BHPOST\_PK" ON "ORA1"."BHPOST" ("POSTID")   
;  
--------------------------------------------------------  
-- DDL for Index BHUSER\_PK  
--------------------------------------------------------

CREATE UNIQUE INDEX "ORA1"."BHUSER\_PK" ON "ORA1"."BHUSER" ("BHUSERID")   
;  
--------------------------------------------------------  
-- Constraints for Table BHPOST  
--------------------------------------------------------

ALTER TABLE "ORA1"."BHPOST" ADD CONSTRAINT "BHPOST\_PK" PRIMARY KEY ("POSTID")  
USING INDEX ENABLE;  
ALTER TABLE "ORA1"."BHPOST" MODIFY ("POSTID" NOT NULL ENABLE);  
--------------------------------------------------------  
-- Constraints for Table BHUSER  
--------------------------------------------------------

ALTER TABLE "ORA1"."BHUSER" ADD CONSTRAINT "BHUSER\_PK" PRIMARY KEY ("BHUSERID")  
USING INDEX ENABLE;  
ALTER TABLE "ORA1"."BHUSER" MODIFY ("USEREMAIL" NOT NULL ENABLE);  
ALTER TABLE "ORA1"."BHUSER" MODIFY ("USERPASSWORD" NOT NULL ENABLE);  
ALTER TABLE "ORA1"."BHUSER" MODIFY ("USERNAME" NOT NULL ENABLE);  
ALTER TABLE "ORA1"."BHUSER" MODIFY ("BHUSERID" NOT NULL ENABLE);  
--------------------------------------------------------  
-- Ref Constraints for Table BHPOST  
--------------------------------------------------------

ALTER TABLE "ORA1"."BHPOST" ADD CONSTRAINT "FK\_BHUSERID" FOREIGN KEY ("BHUSERID")  
REFERENCES "ORA1"."BHUSER" ("BHUSERID") DEFERRABLE INITIALLY DEFERRED ENABLE;

## Add Test Data to Bullhorn Database

The following sql scripts can be run in SQLDeveloper to add data to your bullhorn tables. This will be fine for getting use started and then we can see if you site is working. After that we will generate more fake data to test our site. A good source of fake data is [generatedata.com.](http://generatedata.com/). You can also look at the lorem ipsum word generator. This page explains lorem ipsum and allows you to generate some test data: [http://www.lipsum.com.](http://www.lipsum.com/)

select \* from bhpost;

insert into bhuser (USERNAME,USERPASSWORD,MOTTO,USEREMAIL) values ('user 1','password','motto for user 1', 'user1@domain.com');  
insert into bhuser (USERNAME,USERPASSWORD,MOTTO,USEREMAIL) values ('user 2','password','motto for user 2', 'user2@domain.com');  
insert into bhuser (USERNAME,USERPASSWORD,MOTTO,USEREMAIL) values ('user 3','password','motto for user 3', 'user3@domain.com');  
insert into bhuser (USERNAME,USERPASSWORD,MOTTO,USEREMAIL) values ('user 4','password','motto for user 4', 'user4@domain.com');

select \* from BHUSER;

update bhuser set joindate=sysdate where bhuserid=1;  
update bhuser set useremail = 'user2@domain.com', joindate=sysdate where bhuserid=2;  
update bhuser set useremail = 'user3@domain.com', joindate=sysdate where bhuserid=3;  
update bhuser set useremail = 'user4@domain.com', joindate=sysdate where bhuserid=4;  
-----------------------------------------------------------------------  
select posttext from bhpost where bhuserid=5;

--insert sample post data  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 1',5);  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 2',6);  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 3',7);  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 4',8);  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 5',5);  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 6',5);  
insert into bhpost (POSTDATE,POSTTEXT,BHUSERID) values (sysdate,'This is a sample post 7',5);

--query to join users with their posts. We will be doing this in the newsfeed page  
select bhuser.bhuserid, username,useremail,posttext,postdate from bhuser join bhpost on bhuser.bhuserid = bhpost.bhuserid;

--how to search for posts that contain some text  
select \* from bhpost where posttext like '%5%';

--how to determine if the user has a valid password  
--users with both valid username and password will return 1; others will return 0  
select count(bhuserid) as CountOfUserID from bhuser where useremail='user1@domain.com' and userpassword='hacker123';

--After making changes to the data you must press the commit button (F11) to save the data to the underlying database

## Commit your Database Changes

If you make changes to data in your Oracle database through SQL Developer you must commit your changes to save them to the table.

The **COMMIT** statement will end your current transaction and make permanent all changes performed in the transaction. A transaction is a sequence of **SQL**statements that Oracle Database treats as a single unit.

## Add your Database scripts to your Java Project

Create a folder in Eclipse called DBScripts. This is where you’ll store a copy of your Oracle database scripts. that way your database will also be included in your backup on GitHub. If you decide to work on a different computer you can create an exact copy of your database from the scripts.

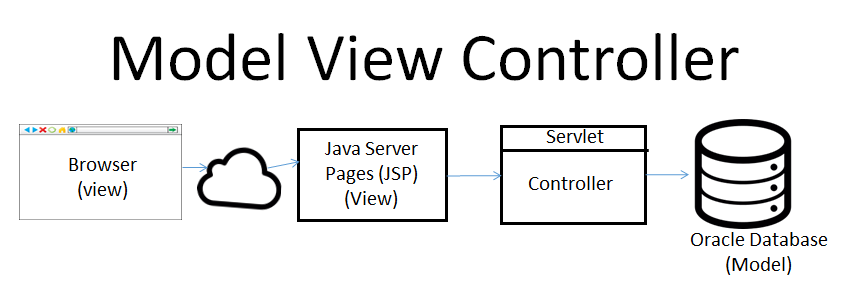
To export the scripts simply right-click on the table name and select the Export option from the menu. You can use the control button to allow you to select multiple tables.

# How to create a dynamic web application using Eclipse

We will create the Bullhorn application as a dynamic web project. That will allow us to develop Bullhorn with HTML, JSP, Servlets and using JPA to connect to the database. If you understand what all those things are then you can drop the class now. If you’re still with me, then they will be explained as we go.

Our dynamic web application will contain not only static html pages but also dynamic Java Server Pages (JSPs) and Servlets. We’ll be able to communicate between each part of the application. The Database is not actually part of the dynamic web application - it’s a separate application. The JSPs will send information to the Servlet. The Servlet will send information to (and get information from) the Database. And the Servlet will send the result back to the JSP.

It is possible to send information straight to the database from a JSP. It is also possible to send information between two JSPs. We won’t do that here. We’re putting a servlet in between every conversation. That allows us to intercept each message with some Java code in the Servlet which will make it easy to validate, evaluate and redirect each intercepted message.



**How to Create a Dynamic Weeb Project for Eclipse:**

1. From Eclipse choose File | New | Dynamic Web Project
2. Give it a project name, MVCTest01 (don’t call it Bullhorn... that comes later)
3. Select the target runtime as Tomcat v. 7.0
4. Click Finish
5. Select Yes to associate with a Java EE Perspective

## Create a basic application to display form data

We’re going to step aside from Bullhorn for a brief diversion. This will allow us to explore the components that make up Bullhorn before we create the actual application. It will also give us experience creating multiple dynamic web applications.

Java Web applications typically contain forms that collect user input and pass it to a servlet for processing. The servlet can then communicate to the database and do something with the data.

Once the servlet is finished working with the data it will send a new web page to the browser with the results of the form. All this happens in an instant on the server and out of the sight of the user.

### Create an HTML Form

HTML forms are used to get input from the user. We want to enable the user to log in with their email and a password. So we need to create a web page with an html login form. The form should contain two text boxes. One for user name and one for password. The form needs a submit button and the textboxes and button must be contained within the tags that declare the form.

Add a link to the form page to go to a servlet which we will create called ProcessForm. This is specified as the value of the action attribute. All attribute values must be in quotes and in the format of attribute="value". These values will be used by the web server to determine how the form is processed.

The form will not work at first. Until we create the servlet. The servlet is a container that can run Java code and process our form. It will receive the values from the inputs. Then we can write Java code to do something with the inputs.

<!DOCTYPE html>  
<html>  
<body><form action="ProcessForm" method="get">  
 First name:<br>  
 <input type="text" name="firstname">  
 <br>  
 Last name:<br>  
 <input type="text" name="lastname">  
 <br><br>  
 <input type="submit" value="Submit">  
</form>  
<p>If you click the "Submit" button, the form-data will be sent to a page called "process\_form.jsp".</p>  
</body>  
</html>

 Here are the elements that you can add to an HTML form. Most elements require various attributes such as id and name.

* **<form>** Defines an HTML form for user input. Be sure to set the action attribute to your servlet's @WebServlet annotation
* **<input>** Defines an input control. It must have a type attribute. And a name attribute.
* **<textarea>** Defines a multiline input control (text area)
* **<label>** Defines a label for an <input> element
* **<select>** Defines a drop-down list
* **<optgroup>** Defines a group of related options in a drop-down list
* **<option>** Defines an option in a drop-down list
* **<submit>**Defines a form submit button

## Create a Servlet

Create a new Dynamic Web Project in Eclipse. You are going to create a java servlet!

Servlets are Java classes which respond to incoming HTTP requests. Servlets reside within the web server – Tomcat and listen for requests. Then they spring into action and process the request. Think of "operators are standing by!".

### Create a page to display the output of your form

Next, we will create a JSP to display the output of the form. The form will send its data to the servlet and the servlet will send the data to the output JSP. While it’s possible to bypass the servlet there’s no good reason to do so since any application of significance will use the servlet to perform some processing.

The page to display the output will be called, simply enough, output.jsp.

### Passing Parameters

You've created a website. It contains an html page, index.html, a servlet, ProcessForm.java, and a results page, output.jsp.

## How to get a value from the form to the servlet and then to the output page

Use either request parameters or session variables.

### Request Parameters

<form action="ProcessForm" method="post">

Enter your name: <input type="text" name="userName">

<input type="submit" value="Submit">

</form>

### Getting the form data into the servlet

When the user clicks the submit button the data from the form gets sent to the servlet in the request object. The web server takes care of this. The servlet container, Tomcat, will make the request object available to your servlet. Your input named userName will contain the name that the user typed. The servlet can read that name by usingrequest.getParameter("userName"). You probably want to set it to a variable as in   
String name = request.getParameter("userName");  
Now your servlet can work with it.

### How will the Servlet retrieve the Parameters?

In either the doGet or doPost method of the servlet you will use the getParameter method of the request object to retrieve data from the HTML form and return it to a variable.

Sting fname = request.getParameter("first\_name");

Sting lname = request.getParameter("last\_name");

### Should I use doGet() or doPost()

Your servlet defines two methods, doGet and doPost. They are shown below. The former runs when the form being submitted uses a method of get. The latter when the form uses a method of post.

Officially GET is used for viewing something, without changing it, while POST is used for changing something. For example, a search page should use GET, while a form that changes your password should use POST.

When you submit with Post you can't see the data, however, it is still accessible so don't think this is more secure. I generally teach get first so you can see the data as it goes from the JSP to the Servlet and so on. In a professional environment you should use Post all the time until you have a compelling reason to use get.

### How will the Servlet set the values in your output page?

Create a jsp page called output.jsp.

Add the following code to your page so it will read the values of the parameters from the servlet:

<html>

<head>

<title>The results of my form</title>

</head>

<body>

<center>

<h1>Using GET Method to Read Form Data</h1>

<ul>

<li><p><b>First Name:</b>

${firstName}

</p></li>

<li><p><b>Last Name:</b>

${lastName}

</p></li>

</ul>

</body>

</html>

### How to Make the Servlet go to the Next Page

After the servlet has validated and processed all the incoming data you want to tell it to take the user to the next page. Here's the way Java allows you to do that:

Add this line of code as the last line in your servlet's doPost or doGet method.

Then when the servlet comes to this line of code it will take the user there.

//redirect to next page as indicated by the value of the nextURL variable

String nextURL = "new\_page.jsp";  
 getServletContext().getRequestDispatcher(nextURL).forward(request,response);

# Create a JSP that Displays the values from a Form

## What is a JavaBean?

You can use a JavaBean to pass objects between the servlet and the JSP. A JavaBean or bean, is a POJO (Plain old Java object) Java class that

* provides a zero-argument constructor,
* provides get and set methods for all its private member variables that follow standard Java naming conventions, and
* implements the Serializable or Externalizable interface.

## How to make a JSP secure and dynamic

Markup languages are designed for the processing, definition and presentation of text. The language specifies code for formatting, both the layout and style, within a text file. The code used to specify the formatting are called tags. HTML is a markup language for creating web documents. It is a set of markup tags. Each HTML tag describes the document content contained between the opening and closing tag.

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags you can add to your JavaServer Page (JSP). These tags add functionality common to many JSP applications.

JSTL adds support for common, structural tasks such as iteration and conditionals.

EL (Expression Language) is a subset of JSTL that makes it easy to use JavaClasses (called beans) in your JSP. EL has a compact syntax and allows you to access the nested properties of objects.

EL allows you to access a bean using syntax such as ${name} for a simple variable or ${name.foo.bar} for a nested property such as the property of an object withing your bean.

### How to include JSTL as part of your JSP

* Before you can use JSTL tags in your page you need to import two Java archive (jar) files and include the taglib directive.
* Copy both jar files to the WEB-INF/lib folder of a dynamic web application.
  + jstl-impl.jar
  + jstl-api.jar
* Add a taglib directive to the top of every JSP that uses JSTL

<%@ taglib prefix=”c” uri=<http://java.sun.com/jsp/jstl/core> %>

### How to import a class you’ve created into your JSP?

You’ll need to specify the package name and the class name. Your class should also be a bean. A bean is a regular Java class that meets certain standards. Beans were discussed elsewhere in this document. To make your class available for use in your JSP you specify your class in the import attribute of the page directive. The page directive goes at the top of your JSP.

You can import just one class:

<%@ page import=”package1.class1, package2.class2” %>

Or multiple classes:

<%@ page import=”package1.class1, package2.class2” %>

### How to determine if a String is null or empty using JSTL?

<c:if test="${empty var1}">

var1 is empty or null.

</c:if>

<c:if test="${not empty var1}">

var1 is NOT empty or null.

</c:if>

### Other JSTL tags

#### Escape output to prevent XSS (Cross Site Scripting)

<c:out value="${item.value}"></c:out>

#### Repeat the nested body content a fixed number of times

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:forEach var="number" begin="5" end="10">

<c:out value="${number}"></c:out>

</c:forEach>

#### Repeat the nested body content over a collection

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:forEach var="item" items=”${cart.items}">

<c:out value="${item.value}"></c:out>

</c:forEach>

#### Test a condition

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:set var="number" value="${250}">

<c:if test="${number<1000}">

<c:out value="number is less than 1000"></c:out>

</c:if>

#### Test a condion and choose an alternative

#### The c:when and c:otherwise works like if-else statement when placed inside c:choose tag.

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<h1>c:when, c:otherwise, c:choose</h1>

<c:set value="21" var="i"></c:set>

<c:choose>

<c:when test="${i%2==0}">

<c:out value="Even Number"></c:out>

</c:when>

<c:otherwise>

<c:out value="Odd Number"></c:out>

</c:otherwise>

</c:choose>

### How to include a block of Java Statements in a JSP

You can use the Scriptlet tags <% %> to embed Java code in your JSP.

<% %>

Many developers frown on this. Before you jump ahead and start putting all manner of Java code in your JSP consider how much easier it would be to use only the JSTL EL tags. They are optimized for the purpose and probably perform most of the tasks you would do in your code anyway. Also consider what code could go in the servlet and in any objects that are passed to the JSP. If, after considering all that, you believe you still need to write your own code in the JSP then place it between the scriptlet tags.

### How to retrieve the value of parameters passed to a JSP

Normally the servlet would set these values but if you call the JSP with a url via GET then you may acccess them as parameters.

<% String message = (String) request.getAttribute(“Message”)%>

Top of Form

Bottom of Form

# Web Page Structure

* Web pages are text files with an ".html" suffix and HTML commands / tags
* HTML commands are written in angle brackets < >. They are called tags or elements.
* Most HTML tags consist of a beginning tag and an ending tag. The ending tag contains a forward slash /
* Most HTML tags use attributes to customize them
* In HTML, the "Enter" key will not make a new line in the web page no matter how many times you it it. Use the <p/> or <br/> tags.
* In HTML, more than one space will not be visible unless you use the special character &nbsp;
* Carriage returns and spaces will be visible if you put them inbetween <PRE> and </PRE> tags

## Basic Parts of a Web Page

Each web page begins and ends with a special command: <HTML> and the ending </HTML>. These commands tell the browser that everything in-between is the web page.

Inside the web page, there are two basic parts: the HEAD and the BODY.

The HEAD is used to give the title of the web page (visible in the Title Bar). It can also contain information about the web page, such as keywords, what the page is for, where people will go after this page, and so on. In this class, we will use it just for the TITLE of the web page. The HEAD area of the web page begins and ends with the commands <HEAD> and </HEAD>.

The BODY is where the real web page is. That's the area which holds the information which we think of as the "web page." The BODY area of the web page begins and ends with the commands <BODY> and </BODY>.

Here is how a web page looks with basic structure:



## The Head

The head contains the title tag, link and script tags. The link and script tags allow

When the HEAD ends the BODY begins.

## The Body

The BODY area will of course have a lot more information, because that is where the code that displays in the browser will go. This includes the form and any text displayed to the user. Also links to other pages and possibly images.

# Designing Web Pages with HTML5

HTML is a markup language for creating web documents (web pages).

* HTML stands for hypertext markup language
* HTML documents are composed of elements called tags
* Tags are used to identify document content and structure
* Tags often contain attributes which provide parameters for the element
* The HTML document which contains content and tags is rendered by the browser to display the nice version of the web page

## HTML5 tags

In HTML a tag is created by putting the tag names in angle brackets like this: *<Tag>* The word in brackets, in this case Tag, is the Tag name. Tags should contain an opening tag and a closing tag. An opening tag contains only the tag name in angle brackets. A closing tag precedes the tag name with a forward slash. For example: <table></table>. If a tag contains no data then the opening and closing tag can be combined, as in <br/>.

Tags can contain attributes which give further information about them. We’ll discuss attributes and then look at some of the tags we’ll use for developing Bullhorn.

## HTML5 Attributes

HTML5 attributes are created using a name-value pair and are usually put alongside the tag name.

### Name-value pairs

Name-value pairs are represented by a set of text strings in which name="value" and are usually seperated by commas,semicolons,space or newline character.

HTML5 attributes are written inside the element's tag and seperated by spaces. For example:

<**input** type="text" value="hello Web"/>

In this code the attributes are type and value and their values are "text" and "hello Web" respectively.

Attributes give extra information about an element for example in the input element the attribute type gives the type of input .

### The Id Attribute

The Id attribute is a unique Identifier for the element.The Id property becomes very useful when using CSS and JavaScript.An example of the Id Attribute.

<**form** id="htm\_form">

</**form**>

### The name Attribute

The name Attribute specifies a name for an element. Groups of elements such as checkboxes can have the same name. In this case the value of all selected checkboxes will become the value of the named group.

## A basic HTML5 document

<!DOCTYPE html>  
<html>  
<head>  
<title>This is the Page Title</title>  
</head>  
<body>  
  
<h1>This is a Heading</h1>

<h2>This is a Subheading</h2>  
<p>This is a paragraph.Any text in the body of the page (within the body tags) should be further enclosed in tags.</p>  
<!--This is a comment -->

<!-- all the interesting stuff goes in the body of the page -->  
</body>  
</html>

## Explanation of common tags

<!DOCTYPE html> - Identifies the document as an HTML5 document. This must be the first thing at the top of the page.

<head></head> - All data in the head section of an HTML document is considered "meta-data", meaning "data about data". The information in this section is not normally displayed directly. Instead elements such as style affect the appearance of other elements in the document. Some items in the head section may be used by programs such as search engines to learn more about the page for reference purposes.

The head element should always contain a title element which sets the title commonly displayed by the web browser in the title bar of the window.

<title></title> - belongs in the head section of the document and sets the title that is displayed in the browser tab.

<script></script> - belongs in the head section of the document and contains either a JavaScript of a link to a JavaScript.

<body></body> - The entire document body is contained within these two tags.

<h1></h1> - any text contained within these tags is displayed as a heading, bold and centered. There are six heading tags… h1 (largest) to h6 (smallest).

<p></p> - any content with the paragraph tags is considered a paragraph. You can add attributes to the paragraph tags to control the display of the paragraphs.

<img></img> - the image tag is used to display images. It has two attributes you need to use: src and alt. The src attribute contains the path to the image file. The path can be either a filename or a url. The alt attribute contains the alternate text to be displayed when the images don't show or can't be seen. It's also used by screen readers to describe the image. A complete image tag would look like:

<img src="path/to/filename.png" alt="Picture of my cat"/>

<form></form> - the form tag contains all the elements of a user input form that gets data from the user and sends it to the servlet. The form tag contains two required attributes, method and action. The method attribute can be either "get" or "post" and it determines how the data is sent to the servlet. The action attribute contains the url of the servlet which processes the form data. A completed form tag looks like this:

<form action="processForm" method="get"></form>

<input></input> - The purpose of a form is to get input from the user. The way you get input from the user is with the input tag. It will create a textbox on the web page. The contents of the input tag will be sent to the servlet when the form’s submit button is clicked. Submit itself is an input. An input tag becomes a submit button when the type attribute is set to “submit”.

Example input tags:

<input id="email" name=”email” type="text" value=""/> - an input tag that displays as a textbox and collects the email address of the user

<input type="submit" value="Submit" id="submit"/> - an input tag that displays as button and calls the form’s action when clicked

<input type="reset" value="Clear"/> - an input tag that displays as a button with a label that says Clear and causes all the form’s input boxes to clear

<textarea></textarea> - an input that contains multiple rows

example: <textarea name=”posttext” id=”posttext” rows=”2” maxlength=”141”></textarea>

## HTML Forms

A form’s action sends the request package back to the servlet.

## HTML5 Comments

Comments can be included in your web page.

Any text between <!-- and --> is a comment and is ignored by the browser. The text will still be sent to the browser but the user will not see it unless they view the source of the page.

## An HTML Table

A table starts with <table> and ends with </table> each table is made of of table rows which start with <tr> and end with </tr>. Each row is made up of headers which start with <td> and end with </td>.

The first row of a table can be used as the header row. In this case change the <td> tags to <th> for the first row.

### Minimal HTML Table:

<**table**>

<**caption**>Formulas and Results</**caption**>

<**tr**><**th**>Formula</**th**><**th**>Result</**th**></**tr**>

<**tr**><**td**>1 + 1</**td**><**td**>2</**td**></**tr**>

<**tr**><**td**>3 \* 5</**td**><**td**>15</**td**></**tr**>

</**table**>

**Captions** are useful for defining or describing the content of the table. Captions are optional.

To add a caption to a table, add the caption element after the opening table tag, with the text of the caption inside the element

Captions are usually displayed outside the border of the table, at the top. The exact appearance and placement of captions is subject to CSS styling.

Table **headings** are a way of defining the contents of the table columns. They are usually only used in the first <tr>, table row.

Instead of using a <td> for the cell, we use a <th>.

By default the text in table headings is displayed bold and centered.

## How to navigate between web pages

Links are found in nearly all web pages. Links allow users to click their way from page to page.

HTML links are hyperlinks. In HTML, links are defined with the **<a>** tag:

<a href="*url*">*link text*</a>

A hyperlink is a text or an image you can click on, and jump to another document.

Example:

<a href="http://www.google.com/">Visit Google</a>

A local link (link to the same web site) is specified with a relative URL (without http://www....)

<a href="index2.html">My other page</a>

# Styling Web Pages with CSS

CSS (Cascading Style Sheets) describe the visual style and presentation your web appication. CSS alows you to separate the style from the structure. This means you are looking through less code when working with your page. This makes it easier to develop your page. Separation of style from structure and content increases maintainability.

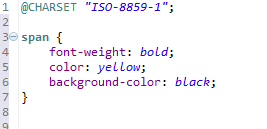
The styles described by CSS include the colors, fonts, layout, and other presentation aspects of a document. A single CSS file can describe a common style applicable to many documents. Typically, a particular element in a XHTML file has a "cascade" of CSS style rules that can be applied to it. The highest priority style rule is applied to each element.

## Why use CSS?

CSS allows you to define a format rule for any element your site may use. This allows you to maintain a consistent appearance for every title, sub-title, paragraph and sentence of your site. And with CSS you can modify the format rule in only one place and impact the site on every page.

## How CSS works

When you create a style sheet you code a rule for each element by name, class or id. These values are set in the element’s attributes. The browser will apply the CSS rule when the page is rendered.



Each rule has two parts: a selector and a group of one or more declarations surrounded by braces. Each declaration consists of a property name and value pair. There can be several declarations in one rule.

## Span and Div Tags

Span and Div are container tags that define parts of your document. They are used to apply styles to a section of a document. They are also helpful for dividing your document into parts such as header, body and footer.

* The <span> tag is used to group inline-elements in a document.
* The <span> tag provides no visual change by itself.
* The <span> tag provides a way to control the style of part of your document.
* The <div> tag defines a section of an HTML document.
* The <div> tag is used to group elements to format them with CSS.
* You can make a <span> or <div> point to a style sheet using the id or class attribute.
* The id must be unique. There can only be one element with that particular id.
* The class attribute can apply to in different elements.

<span style="color:ff0000">This text is red</span> this text isn't red.<div style="color:#0000ff"> <h3>This is a heading</h3> <p>This is a paragraph.</p></div>

## Attributes

The <span> and <div> tags have no *required* attributes but the most common attributes used are:

* **style** - specifies a style that applies to all content and elements up to the end tag
* **class** - specifies a CSS classs that applies to all content and elements up to the end tag
* **id** - identifies the tag so you can select it with jQuery or JavaScript

In general, use id whenever you want to refer to a specific element and class when you have a number of things that are alike.

* The <span> tag is used to group inline-elements in a document.
* The <span> tag provides no visual change by itself.
* The <span> tag provides a way to add a hook to a part of a text or a part of a document.

<span style="color:blue;font-weight:bold">blue</span>

* The <div> tag defines a division or a section in an HTML document.
* The <div> tag is used to group block-elements to format them with CSS.

<div style="color:#0000FF">  
 <h3>This is a heading</h3>  
 <p>This is a paragraph.</p>  
</div>

The <span> and <div> tags have no required attributes, but the three that are the most useful are:

* style
* class
* id

You can make a <span> or <div> point to a style sheet using id or class.

ids must be unique where as class can be applied to many things. In CSS, ids look like #elementID and class elements look like .someClass

### When to use ID and when to use class

In general, use id whenever you want to refer to a specific element and class when you have a number of things that are all alike.

* Style all elements of a particular type by using the element name
* Syle the element that contains a particular id by using a # followed by the id
* Style all elements that contain a particular class attribute by a . (period) followed by the class name



would have a matching entry in the .css file for



# Making Web Pages Do Something with JavaScript

## What is JavaScript?

* JS is object-oriented
* JS is supported by all major browsers
* JS is the programming language of the browser - it’s embedded in an HTML page
* works with text, arrays, dates, regular expressions
* does not include I/O, networking, storage or graphics
* must operate in a browser
* not related to Java
* developed 1995 Netscape
* Like Java, JS is based on C syntax
* primitive types are numbers, strings and booleans

## Where is JavaScript defined in the document?

* Include JavaScript in your web page by putting the script between <script> tags
* Generally place script tags in head section or at the bottom of page
* JavaScript may also be saved in a text file (no script tags needed) and referenced in the head section of your page. This allows you to reuse the same JavaScript in multiple pages.

## What is the DOM?

The *Document-Object Model*, or DOM, is one of JavaScript's more powerful uses. With DOM, you can navigate through and modify an entire page, ranging from simply adding an element to rearranging several areas on the page.

DOM breaks up a document into a *tree* of *nodes*. For example, take a look at the following HTML snippet:

<**div** id="exampleDiv">This is an<**br**>example HTML snippet.</**div**>

Through DOM, JavaScript sees this snippet as four nodes. The div, from its start tag through its end tag, is one node. This div happens to have a *property* assigned inside its start tag. This property is named "id" and has the *value* "exampleDiv".

The three other nodes in this example are inside the div. They are called *child nodes* of the div, because the div encompasses them. Conversely, the div is their *parent node*. The first child of the div is a *text node*, with the value "This is an". Text nodes contain only text; they never contain tags, which is why the node stops here. The <br> tag is another node, and the rest of the text is another text node. Since the text nodes and the <br> tag all share the same parent, they are said to be *sibling nodes*.

## Using JavaScript to Validate a Web Form

The form below is used to submit a post to Bullhorn. Before the user can submit the post it should be validated. JavaScript allows us to do this at the browser. The JavaScript should be placed between script tags at the end of the page.

**function** validateForm**(**str**)**

**{**

*// first name is required*

**var** fn**=**document**.**forms**[**str**][**"first\_name"**].**value**;**

**if** **(**fn **==** ""**)**

**{**

alert**(**"Please Fill In First name"**);**

**return** **false;**

**}**

*// last name is required*

**var** fn**=**document**.**forms**[**str**][**"last\_name"**].**value**;**

**if** **(**fn **==** ""**)**

**{**

alert**(**"Please Fill In Last name"**);**

**return** **false;**

**}**

*// verify email if provided*

**var** em**=**document**.**forms**[**str**][**"email"**].**value**;**

**if** **(**em **!=** ""**)** **{**

**var** atpos**=**em**.**indexOf**(**"@"**);**

**var** dotpos**=**em**.**lastIndexOf**(**"."**);**

**if** **(**atpos**<1** **||** dotpos**<**atpos**+2** **||** dotpos**+2>=**em**.**length**)**

**{**

alert**(**"You have entered an invalid e-mail address"**);**

**return** **false;**

**}**

**}**

**}**

### Fun with Javascript:

Go to google email and enter a name and password on the form. You won’t be able to see the password because it will show up as a bunch of dots or asterisks. Enter the following code in the address bar and press enter:

javascript:alert(document.getElementById('Passwd').value)

We can use JavaScript and JQuery in Bullhorn...

$(document).ready(function() {…. is an anonymous function that is loaded when the document is ready. The $(document).ready() method is part of jQuery which we included when we included Bootstrap. So once the page is loaded this function is added to the element.

|  |
| --- |
| <div class="form-group"> |
| <label for="post">Create New Post (141 char):</label> |
| <textarea name= "posttext" id="posttext" class="form-control" rows="2"  placeholder= "Express yourself!" maxlength="141"></textarea> |
| <div id="textarea\_feedback"></div> |
| </div> |
| <div class = "form-group"> |
| <input type="submit" value="Submit" id="submit"/> |
| <input type="reset" value="Clear"/> |
| </div> |
| </form> |
|  |
|  |
| <jsp:include page="footer.jsp"></jsp:include> |

## Counting and limiting the amount of text in a textbox to 141 characters

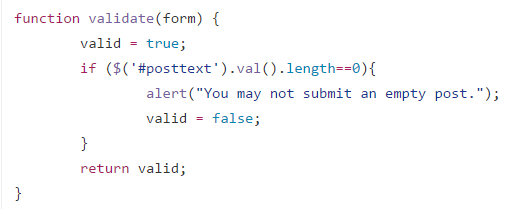
This JavaScript function will load when the document is ready. The document is ready after is has been fully rendered and all the DOM has been downloaded from the web server to the browser. Then the function will be created. This function will set the html property of the element with the id of textarea\_feedback to “XX characters remaining” where XX is the number of remaining characters from the max length of 141.

Within document.ready the keyup event of the element with an id of posttext is modified to include another function which counts the number of remaining characters and displays them in the textarea\_feedback element.

|  |
| --- |
| $(document).ready(function() { |
| var text\_max = 141; |
| $('#textarea\_feedback').html(text\_max + ' characters remaining'); |
|  |
| $('#posttext').keyup(function() { |
| var text\_length = $('#posttext').val().length; |
| var text\_remaining = text\_max - text\_length; |
|  |
| $('#textarea\_feedback').html(text\_remaining + ' characters remaining'); |
| }); |
| }); |

## Ensure a post contains data

This JavaScript function is called validate. It gets a reference to a tag which has an id of posttext. This is accomplished by the jQuery $ function which gets the element by id. If the length of the value of the posttext element is 0 then a JavaScript alert will pop up with a message. If that’s the case then the function will return false. This will cause the form to cancel the submission.



# Getting started with Bootstrap

<https://www.codeofaninja.com/2014/05/bootstrap-tutorial-beginners-step-step.html>

## Responsive Web Design

The way you're going to adapt to different websites is using something called responsive web design. This allows you to adapt your site to desktops, laptops and different mobile devices. You can accomplish this by writing an application which will respond to the width of the web page.

A computer, phone or tablet screen is composed of pixels. The most popular screen resolution for a computer is 1366×768. That means the screen is 1366 pixels wide and 768 pixels high. Screen resolution determines the clarity with which text and images are displayed. Items appear sharper at higher resolutions. They also appear smaller which enables more items to fit on a screen.

When viewed on a tablet the screen may only have 1024 pixels across, on a phone maybe 480 pixels across. Determining how to display your web page on different width devices is known as making your website responsive.

Responsive web design (RWD) refers to the approach of developing a web application that displays well on any size screen from desktop computer to mobile phone.



*Content is like water*, a saying that illustrates the principles of RWD.

Bootstrap is an HTML, CSS, JavaScript framework that you can use as basis for creating responsive web applications.

Bootstrap allows you to benefit from the well written and well tested work of dozens (if not hundreds) of other developers. Bootstrap allows you to develop your application interface rapidly. You won't have to spend time focusing on the HTML, CSS or JavaScript.

You can get more done in less time, and know that the framework you're using or extending is likely to be done better than you doing it all yourself.

As quoted from their site at [getbootstrap.com](http://getbootstrap.com/), "Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web".

## Where to get Bootstrap

#### How to use Bootstrap from a CDN (Content Delivery Network)

* Create a three-column page with image gallery and menu of links
* Walkthrough bootstrap features:
  + The 12 column grid system and how the attributes are set for different devices
  + typography: **Bootstrap** includes simple and easily customized **typography** for headings, body text, lists, and more. For even more control, check out the textual utility classes.
    - http://v4-alpha.getbootstrap.com/content/typography/#display-headings
  + tables: Due to the widespread use of tables across third-party widgets like calendars and date pickers, we’ve designed our tables to be opt-in. Just add the base class .table to any <table>.
    - http://v4-alpha.getbootstrap.com/content/tables/
  + images: Opt your images into responsive behavior (so they never become larger than their parent elements) and add lightweight styles to them—all via classes.
    - http://v4-alpha.getbootstrap.com/content/images/
  + Jumbotron: A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.
    - http://v4-alpha.getbootstrap.com/components/jumbotron/
  + Wells: The .well class adds a rounded border around an element with a gray background color and some padding
    - http://www.w3schools.com/bootstrap/bootstrap\_wells.asp
  + alerts: Provide contextual feedback messages for typical user actions with the handful of available and flexible alert messages. Alerts don't have default classes, only base and modifier classes.
    - http://getbootstrap.com/components/#alerts
  + button-dropdowns: Use any button to trigger a dropdown menu by placing it within a .btn-group and providing the proper menu markup.
    - http://getbootstrap.com/components/#btn-dropdowns
  + navbar: Navbars are responsive meta components that serve as navigation headers for your application or site. They begin collapsed (and are toggleable) in mobile views and become horizontal as the available viewport width increases.
    - http://getbootstrap.com/components/#navbar
  + form: Bootstrap provides several form control styles, layout options, and custom components for creating a wide variety of forms.
    - http://v4-alpha.getbootstrap.com/components/forms/
  + inputs
    - http://www.w3schools.com/bootstrap/bootstrap\_forms\_inputs2.asp
  + carousel
  + badges – to show #posts per user
  + pagination
  + validate email

# Connecting to the Database from Java with JPA

## Why use JPA?

JPA allows you to use your object-oriented programming skills to work with a database. Furthermore, it makes all databases look the same to your program. So as the programmer you develop your application to work with JPA and JPA will take care of the details of connecting to the database.

JPA is an object-relational mapping specification. When you set up JPA it will examine your database tables and create a class for each table. The class name is the table name and the class fields are the table fields. Each class represents one record in your database. When you want to work with all the records in your table then you will store the collection of entities in a typed ArrayList.

JPA will create getters and setters for each field in your table. JPA will handle sequences and identity keys and even relationships between tables.

If you have a table which contains a foreign key to another table the class representation of that table (known as a model) will contain a class of the foreign key table rather than the key. Since the posts table in our database contains a userID to identify the owner of the posts the Post class in our JPA model will contain a User object. This makes it easy to work with the user of the Post class.

The Java Persistence API (JPA) was introduced as part of the Java EE 5 platform to simplify the development of Java applications using data persistence and to unify the community behind a standard lightweight persistence API. We can query the objects using the Java Persistence Query Language (JPQL). This allows us to work with the objects in a way that is similar to SQL (Structured Query Language).

An advantage of using JPA is that we can change the database without changing our Java code. The database information is stored in an XML file which can be edited without recompiling our application. You may start off writing your application using MySQL then as it grows move up to Oracle without any changes to your code.

## What are entities?

The business classes to be used with JPA. Each entity instance contains the data for a single record in the database

## What is a persistence unit?

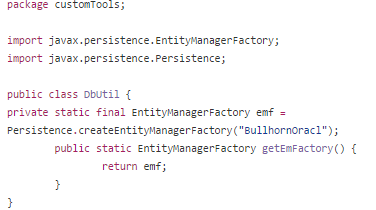
JPA uses an xml configuration file which contains configuration information JPA requires to connect to your database. We’ll learn how to edit and read this file so we can understand how JPA connects to the database.

## Develop a JPA application which connects to the database and displays a record

* Click on the checkbox next to JPA and you will be asked to do some configuration. Click the link and get the following screen. Set the following:
* **Platform**: to EclipseLink
* **JPA Implementation Type:** Disable Library Configuration
* **Connection:** to the name of your connection.
* Once this is done click the connect link immediately under and to the right of the connection drop down.
* And then click **OK**.

### Create a DbUtilities Class

To simplify the process of callling the EntityManager when we need to persist (or merge) data to the database we’ll create a DbUtilities class. This class will be static so we don’t need to create an instance of it each time we use it. It will have one method, getEmFactory() will will return an instance of EntityManagerFactory identified in the persistence.xml.



### Creating the DbUser Class

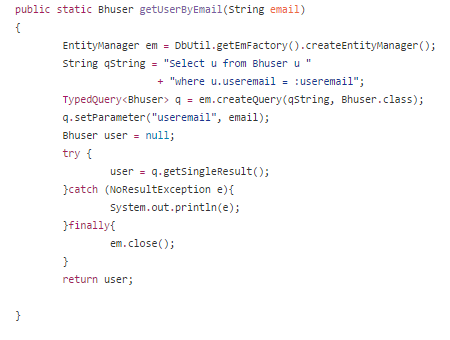
The DbUser class contains methods for working with the user.

Each table in your application will have a corresponding class in the model package. For example, you’ll add a new user to the table in the database by creating an instance of the Bhuser class and populating it with the required data. Then you need to use the persist method of the EntityManager to write the data to the database. If you’re updating a user then use the merge method of the EntityManager.

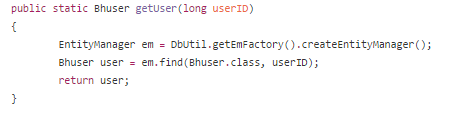
### Determining if a user is valid based on email/password



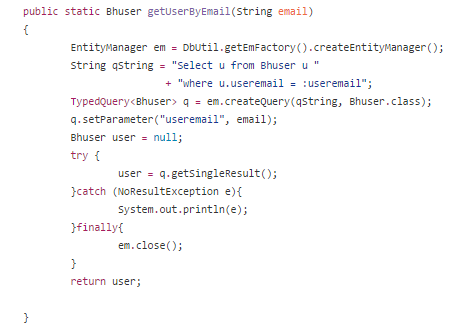
### Get a user by email address



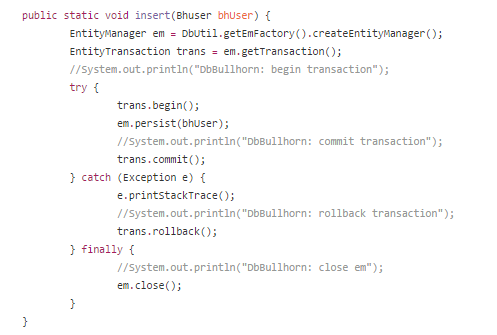
### Selecting a user by primary key with JPQL



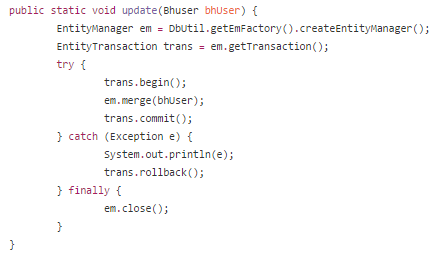
### Parameterized queries – select a user given their email address



### Inserting a user with JPQL



### updating a user with JPQL



# The Stateless Nature of the Web

## Understanding Sessions and Requests

A web application does not maintain state. That means that each request to the web server is an independent request and does not know about any previous related requests.

When you send your username and password the web server views this as an independent event. It does not keep track of what you’re logging in to. If you submit a form then all the information for what to do with the form data must be send along with the form because each request is an independent transaction.

In real life this is what it would be like if you went to the bank and after each question you would get a new teller.

You: I would like my balance, please

Teller #1: I can give you your balance, please show me your ID and account number

You: submits ID and account number along with a note that says I want my balance

Teller #2: processes request for balance and returns balance, ID and account number then forgets about you

You: hands next teller your ID, account number, and a request to withdraw money

Teller #3: processes request for with draw and returns money, ID, account number and receipt

So how does a web application maintain state? Either session variables or passing parameters.

Before we go further understand that data is sent between the client (web browser) and the server via either the URL or as other information sent to the server as part of the request. I call this the request packet.

Session variables exist in the memory of the web server and each request includes a session ID which ties the request to all the variables for that session.

Parameters can be passed via to methods: get or post. Get passes them via the URL, post passes them via the request packet.

So, it’s the request packet and the session that ties everything together. In any event a session is almost always needed for the server to connect one request to another.

* The form passes the request
* Servlet receives the request
* the servlet processes the request with request.getParameter()
* the servlet generates a response based on the data in the request
* the servlet constructs a response in an object that will be sent to the JSP
* the JSP contains an attribute ${user}
* the servlet sets the attribute request.setAttribute(“user”,myUser);
* The servlet sends the JSP back to the originating URL (aka, your browser)
* getServletContext().getRequestDispatcher(url).forward(request, response);

# How to add a User to the Session

A user object that is stored in the session is easily available to every servlet or JSP in your site.

On the login page we validated the user and sent them to the next page. But the next page doesn't know anything about the user. Let's put the user in the session so the next page, and any other pages in our application, can have access to the user. This will allow every page to display the user's name and Gravatar.

Java Servlets provide a variable called HttpSession to identify a user across more than one page request. The session persists for a specified time period across more than one connection or page request from the user.

You would get a reference to the HttpSession object by calling the public method **getSession()** of HttpServletRequest, as below:

 HttpSession session = request.getSession();

By adding this line of code to your servlet you now have access to a variable that can be viewed across any page the user visits.

## To add a value to the session for use by other servlets and JSPs:

User user = new User();  
user.setUserName("Larry");  
user.setEmail("larry12345@gmail.com");  
//add the user to the session

session.setAttribute("user", user);

## To read a value from the session

User user = (User) session.getAttribute("user");  
//now we can get values out of the class  
String username = user.getUserName();  
String email = user.getEmail();

Note: (User) is known as a cast. It converts the session attribute from an Object to a User.

## Getting a User from the Session

Create another JSP called home.jsp. This will eventually become the user's home page. It will show all their posts. The first thing we want to do on the home page is show the user's name. Make her or him feel welcome.

How do we get the user's name?

When the user logged in the login servlet validated their email and password. Then the user was added to the session. On the home page we can get the user out of the session and use that information to display the user's name on the page.

The user's information will be stored in a class called User.

First, let's create a reference to the session:

HttpSession session = request.getSession();

Next, create a variable to hold the user information. This assumes  you've already created the User class:

User u = new User();

Then, get the user out of the session. The (User) casts the object in the session to a User object. Otherwise the object in the session will remain a generic object:

u = (User) session.getAttribute("user");

Last, display the user name on the home page.

This name will change for each user. Close your application and log in as a different user.

Did it work? Don't go to the next assignment  until you can log in a one user and see their name on the home page. And then log in as a second user and see their name on the home page.

# How to use the Java mail API to send email to the support team

To create an application to send email you need a mail server. If you don't have a mail server available then you might want to set up a fake email server. This has already been done.

You can download fakeSMTP from here:

<https://nilhcem.github.io/FakeSMTP/index.html>

And run it by executing he JAR as:

 java -jar fakeSMTP-2.0.jar -o /home/oracle/Downloads  &

 This java archive will run as a Java Swing application and listen on the selected port for your application to send an email. It will then collect the email in the folder you specify above. You can read more details on its web page above.

Your application will require the JavaMail API which you can download from here: [https://java.net/projects/javamail/pages/Home#Download\_JavaMail\_Release](https://java.net/projects/javamail/pages/Home)

You should create a Dynamic Web application with a servlet and JSP that allows a user to enter all the fields for sending an email.

The following code illustrates how to send a message. Notice that the server properties are set up to match the fakeSMTP server. Also note that the session variable refers to the javax.mail.Session which is not the same as HttpSession.

## Add the Java Mail API to your application

Add the javax.mail.jar file to WEB-INF\lib

## Import some libraries

import java.util.Properties;  
import javax.mail.\*;  
import javax.mail.internet.\*;

## Get a mail session

        Properties props = new Properties();  
        props.put("mail.transport.protocol", "smtp");  
        props.put("mail.smtp.host", "localhost");  
        props.put("mail.smtp.port", 25);  
        props.put("mail.smtp.auth", "true");  
        props.put("mail.smtp.quitwait", "false");  
        //NOTE: Session object is part of javax.mail.Session  
        javax.mail.Session session = javax.mail.Session.getDefaultInstance(props);

## Create a message object

Message message = new MimeMessage(session);  
 message.setSubject(subject);  
 if (bodyIsHTML) {  
 message.setContent(body, "text/html");  
 } else {  
 message.setText(body);  
 }

## Address the message object

        Address fromAddress = new InternetAddress(from);  
        Address toAddress = new InternetAddress(to);  
        message.setFrom(fromAddress);  
        message.setRecipient(Message.RecipientType.TO, toAddress);

## Send the message object

        Transport transport = session.getTransport();  
        transport.connect();  
        transport.sendMessage(message, message.getAllRecipients());  
        transport.close();

# How to use Gravatar to display user’s avatars with posts

Gravatar is a service for providing globally unique avatars. Gravatar allows users to register an account using their email address. Users then upload an avatar to be associated with the account. Other websites, including Bullhorn, can freely use Gravatar to display the user’s image given their email address.

When the user posts a comment on a site that uses Gravatar, the site checks whether that email address has an associated avatar at Gravatar. If so, the Gravatar is shown along with the comment.

## How to display the Gravatar

The site takes the email address and creates an MD5 hash from that address. The MD5 hash becomes part of the Gravatar url that returns the image. This url can be used in an image tag to display the image on an HTML page.

For example, the fake email address [ymu@z.zgrco.com](mailto:ymu@z.zgrco.com) has an MD5 hash of eb7e8a51c9991b29e55aa29ec68c5702.

Combine the MD5 hash with the Gravatar URL:

<http://gravatar.com/avatar/eb7e8a51c9991b29e55aa29ec68c5702>

Use the URL in an IMG tag as the src:

<img src=”<http://gravatar.com/avatar/eb7e8a51c9991b29e55aa29ec68c5702>”/>

### Setting the size of the Gravatar image

Simply append the url with ?s=150 where 150 is the length in pixels of the square image returned by the url. the value of s can range from 1 to 2048. Lower values will look better.

### Test Gravatars you can use

[ymu@z.zgrco.com](mailto:ymu@z.zgrco.com)  
[lpr@s.rv55.com](mailto:lpr@s.rv55.com)  
[xdwr@my.qsl.ro](mailto:xdwr@my.qsl.ro)  
[kqy@g.gtvtech.com](mailto:kqy@g.gtvtech.com)

### Default image

If a user does not have a Gravatar set up then the default image will be displayed:



### Site to create temporary email addresses (for creating test Gravatars)

[https://mytemp.email](https://mytemp.email/)

### Random user photos for creating test Gravatars <https://randomuser.me/photos>

## How does Gravatar Work in Bullhorn?

Each Bullhorn user is uniquely identified by their email address. That email address is used to retrieve the Gravatar. Since the Gravatar site expects to receive an MD5 hash of the email address as part of the url, Bullhorn includes the capability to hash the email address and construct the URL then request the Gravatar as an image.

## How to generate an MD5 hash in Bullhorn

The following class will provide you with a static method that returns the hex format md5 of an input string. See: <http://en.gravatar.com/site/implement/images/java/> for more details.

package util;

import java.io.\*;

import java.security.\*;

public class MD5Util {

public static String hex(byte[] array) {

StringBuffer sb = new StringBuffer();

for (int i = 0; i < array.length; ++i) {

sb.append(Integer.toHexString((array[i]

& 0xFF) | 0x100).substring(1,3));

}

return sb.toString();

}

public static String md5Hex (String message) {

try {

MessageDigest md = MessageDigest.getInstance("MD5");

return hex (md.digest(message.getBytes("CP1252")));

} catch (NoSuchAlgorithmException e) {

} catch (UnsupportedEncodingException e) {

}

return null;

}

}

Call this from your application as shown below.

String email = "someone@somewhere.com";

String hash = MD5Util.md5Hex(email);

You may wish to write a method which actually creates the url taking the email address and size as parameters.

<http://gravatar.com/avatar/eb7e8a51c9991b29e55aa29ec68c5702>?s=150

# Create other Bullhorn Pages and Servlets

## Create and Style the Login Page

The login page is called login.jsp. This page contains a form which allows the user to enter their username/password and click submit to log in. If the user is a valid user they will be redirected to their home page. If the user is not a valid user they will be sent back to the login page with no feedback.

* Create the jsp page
* add a form to the page
* add an input for email
* add an input for password
* add a hidden input indicating the form being used
* add a submit button
* set the method and action for the form
  + method is post
  + action is /LoginServlet
* add classes for bootstrap
* add a link to the adduser.jsp page
* add an include tag to include footer.jsp

## Create the Login Servlet

The job of the login servlet is to validate the user and add them to the session. Then direct the user to the home page. If the user fails validation then go back to the login page.

Secondly, the login servlet is called when the user clicks the logout button. At that point the login servlet ends the session and displays the login page.

* set the attribute that is used as the action in the login.jsp page
* remove doGet and put all your code in doPost
* get a reference to the session
* if user is logging out then invalidate session and redirect to login page
* validate user
* if user is valid then add them to the session and redirect to their home page
* if user is invalid then redirect them to the login page

## How to validate the user

look up the user in the database by email. Each email address can only be used once and each user must have an email address to log in. So if the user’s password and email match then we can conclude that their email address is valid.

Valid users are added to the session. Invalid users are redirected back to the login page.

## How the log out button works

When the user wants to log out then you simply end their session. That makes logging out easy. Most users won’t click the logout button. For them the session ends when the session timeout is reached. you can set the session timeout property or use the default of 20 minutes.

Create a form that will pass a parameter called “action” with a value of “logout” to your login servlet. To pass that parameter you’ll create a hidden input and give it a name and id of action with a value of logout. The form’s action will be the name of the login servlet. When the servlet receives the parameter it will invalidate the session and redirect the user to the login page.

 The Java code to end a session is:

session.invalidate();

## How to redirect the user to the next page

#### Use response.sendRedirect or getServletContext().getRequestDispatcher() ?

getServletContext().getRequestDispatcher() is more efficient as it takes place on the server. Otherwise you would have to send a redirect request back to the browser. The browser would then make the request.

String nextURL = "home.jsp";  
 //redirect to next page as indicated by the value of the nextURL variable  
 //getServletContext().getRequestDispatcher(nextURL).forward(request,response);  
 response.sendRedirect(request.getContextPath() + nextURL);

### Forward

* a forward is performed internally by the servlet
* the browser is completely unaware that it has taken place, so its original URL remains intact
* any browser reload of the resulting page will simple repeat the original request, with the original URL

### Redirect

* a redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original
* a browser reload of the second URL will not repeat the original request, but will rather fetch the second URL
* redirect is marginally slower than a forward, since it requires two browser requests, not one
* objects placed in the original request scope are not available to the second request

### Passing an attribute or value to your JSP

If you want to set a value in your jsp then create a placeholder for it on the jsp called ${message} and set it as follows below:

request.setAttribute("messages", messages);  
The value you sent in messages can be text or it can be an object.

## Create and Style the Home Page

* Add Bootstrap CDN
* Create the NavBar
* Add Gravatar to NavBar
* Add Search form to Navbar
* Add logout to the navbar
* Add Table of posts
* Styling your table with Bootstrap
* Add Footer

## Move the Bootstrap links, NavBar and Footer to include files

Include files allow you to move code out of your page and into a file that will be merged with your page before the user accesses it. That way you can re-use the code in several pages. So you don’t have to create the NavBar again we’ll move it to an include file. Then if you need to change the NavBar in any way those changes will apply to all pages that use it. You’ll copy the code for the NavBar to a file called navbar.jsp and replace it with the following tag:

<jsp:include page="navbar.jsp"></jsp:include>

## Create the Post Servlet

* remove doGet() and put all your code in doPost()
* Get the current date
  + Date postdate = new Date();
* Get the post text from the request parameter
* Get the user from the session
* Bhuser user = (Bhuser) session.getAttribute(“user”);
* Create an empty Bhpost
* Add the user to the new post
* add the current date to the new post
* add the text from the user to the new post
* insert the new post in the database
* redirect to the user’s home page

## Create and Style the Newsfeed Page

The newsfeed page is called by the newsfeed servlet and displays posts. The posts it displays depend on the parameters sent to the newsfeed servlet. See the section on the newsfeed servlet for more details.

* add tag library links

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> |
| <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %> |
| <fmt:setLocale value="en\_US" /><!-- fixes date not displaying correctly in Eclipse browser --> |

* Add bootstrap include file
* add navbar include file
* add html table with three columns: user, post, date
* how to format table with bootstrap
* How to use JSTL and EL to populate the table of posts
  + Use forEach
* How does the c:out tag protect from hackers
  + surround fields with c:out
* Add the footer include file

## Create the Newsfeed Servlet

The user does not need to be logged in to view this page. The newsfeed page shows all the posts from all the users. The posts that will be displayed will depend on the parameters that are passed. If no parameters are passed then all the posts will show. If the userid is passed then the page will only show posts for that user. If the searchtext parameter is passed then the page will show posts that contain the words in the search text.

* call doPost from doGet and write all your code in doPost
* check if the user exists in the session
* how to add code to filter posts by that userid
* how to filter posts by the search text
* redirect to newsfeed page

## Create and Style the User Profile Page

* include the bootstrap links
* include the navbar
* add the page directives for the tag libraries
* how the user is sent from the servlet
* how to display the user name, email, image (from GravatarURL) and join date
* create a form for the user to update their profile
* use jstl to allow user to edit fields when the user is viewing their own profile
* add the footer include tag

## Create the Profile Servlet

* call doPost from doGet
* verify the user exists in the session or redirect to the login page
* if the parameter indicates update profile then
  + get Bhuser object from session (so user can’t pass it to the servlet without logging in)
  + get user email from jsp form
  + get user motto from jsp form
  + set the values in the Bhuser object
  + update the Bhuser object in the database
  + put the updated user back in the session

## Create the add user page

* add the page directives for the tag libraries
* include the bootstrap links
* include the navbar
* create a form to get the Name, Email, Passoword, and Motto from the user
* add hidden input to form to indicate the form is from the AddUser page
* add submit button to form
* set the action to “AddUser” and the method to “post”
* add the footer include link

## Create the add user servlet

This page does not require the user to be logged in

remove doGet and put all your code in doPost

* check if the user exists in the database
* if user does not already exist then
  + create empty Bhuser object
  + set the user name
  + set the user email
  + set the password
  + set the motto
  + insert the user in the database

## Create and Style the Support Page

The support page allows the user to email the website administrator. This is great for reporting problems or just collecting feedback. The support page will contain a form that resembles an email but the subject will be limited to one of several categories so the administrator can take appropriate action on the message before reading it.

## Create the Support Servlet

The support servlet sends the email generated by the support page. It then redirects to a confirmation page stating that the message has been sent. The confirmation page should also allow the user to return to their home page or the page they were viewing before they sent the email.

## Create and Style the Error Page

When a page is not found or there is a server error or any other Java error you want to inform your users nicely.We’ll use the same error page for all errors so it doesn’t give any useful information to the user. If you let Tomcat handle the error then Tomcat will spill the beans and give out all kinds of information about your site. That could be helpful to hackers who want to create trouble.

* Create a JSP
* Add links to the bootstrap, navbar and footer include files
* add a message such as something’s wrong
* add an image

## Create the Error Servlet

An error servlet gives you the chance to log the error or even receive an email when an error occurs. At the least, someone should know that something has gone wrong.

## Send Email to the Administrator when an error occurs

When you get an error in the catch section of your try/catch block call the method to send an email to the administrator with details of the error.

# Wrapping Up

Congratulation! You’ve completely built a Java EE application

#### What you can do next…

* Show my posts on my home page.
* Add followers so the user’s home page displays their own posts plus those of their followers.
* Add email and direct messaging from user to user
* Add a cookie to save the user from having to log in on a known pc
* Create test data
* publish website online