**Java EE**

**(J2EE)**

**Projects**

Author: Mike Dean

Date: February 9, 2015

Table of Contents

[1. Project Overview 1](#_Toc411285737)

[2. Java EE Project Types 1](#_Toc411285738)

[2.1. Java Server Pages (JSPs) and Servlets 1](#_Toc411285739)

[2.2. Java Server Faces (JSFs) 1](#_Toc411285740)

[2.3. JSFs and Context and Dependency Injection (CDI) 1](#_Toc411285741)

[2.4. JSFs and Java Persistence 2](#_Toc411285742)

[3. Projects 2](#_Toc411285743)

[3.1. JSPs and Servlets 2](#_Toc411285744)

[3.1.1. Web Pages - JSPs 2](#_Toc411285745)

[3.1.2. Code - Servlet 3](#_Toc411285746)

[3.1.3. Testing 7](#_Toc411285747)

[3.1.4 Error Messages: 11](#_Toc411285748)

[3.1.5 Feedback Message: Winning 12](#_Toc411285749)

[3.1.6 Feedback: Number of Tries: 13](#_Toc411285750)

[3.2. JSFs 13](#_Toc411285751)

[3.2.1. Web Pages - JSFs 13](#_Toc411285752)

[3.2.2. Code - Managed Beans 17](#_Toc411285753)

[3.2.3. Testing 22](#_Toc411285754)

[3.2.4. Error Messages: 24](#_Toc411285755)

[3.2.5. Feedback Messages: 25](#_Toc411285756)

[3.3. JSFs – CDI 27](#_Toc411285757)

[3.3.1. Web Pages – JSFs 27](#_Toc411285758)

[3.3.2. Code – CDI Beans 31](#_Toc411285759)

[3.3.3. Testing 37](#_Toc411285760)

[3.3.4. Error Messages: 39](#_Toc411285761)

[3.3.5. Feedback Messages: 40](#_Toc411285762)

[3.4. JSF and Java Persistence 41](#_Toc411285763)

[3.4.1. Web Pages – JSFs 42](#_Toc411285764)

[3.4.2. Code – CDI Beans / EJB Bean 46](#_Toc411285765)

[3.4.3. Testing 54](#_Toc411285766)

[3.4.4. Error Messages: 56](#_Toc411285767)

[3.4.5. Feedback Messages: 57](#_Toc411285768)

List of Tables

[Table 1: JSP of Index.jsp – Screenshot 2](#_Toc411285769)

[Table 2: JSP of game.jsp – Screenshot 3](#_Toc411285770)

[Table 3 - Servlet - Index Processing 4](#_Toc411285771)

[Table 4 : Game Processing Part 1 5](#_Toc411285772)

[Table 5 : Game Processing Part 2 6](#_Toc411285773)

[Table 6 : Game Processing Part 3 7](#_Toc411285774)

[Table 7: main template 14](#_Toc411285775)

[Table 8: Index.xhtml 15](#_Toc411285776)

[Table 9: Game.xhtml 16](#_Toc411285777)

[Table 10: WinList.xhtml 16](#_Toc411285778)

[Table 11: GameBean.java 20](#_Toc411285779)

[Table 12: main template 27](#_Toc411285780)

[Table 13 - Index.xhtml 28](#_Toc411285781)

[Table 14 - Game.xhtml 29](#_Toc411285782)

[Table 15: WinList.xhtml 30](#_Toc411285783)

[Table 16 - WinList.xhtml - Display 30](#_Toc411285784)

[Table 17 - logic.java 31](#_Toc411285785)

[Table 18 - GameBean.java 33](#_Toc411285786)

[Table 19 – UserBean.java 34](#_Toc411285787)

[Table 20 – ResponseBean.java 36](#_Toc411285788)

[Table 21 - main\_template 42](#_Toc411285789)

[Table 22 - index.xhtml 42](#_Toc411285790)

[Table 23 - game.xhtml 43](#_Toc411285791)

[Table 24 - gameOver.xhtml 44](#_Toc411285792)

[Table 25 - List.xhtml 44](#_Toc411285793)

[Table 26 - Logic.java 46](#_Toc411285794)

[Table 27 - GameBean.java 48](#_Toc411285795)

[Table 28 - UserBean.java 49](#_Toc411285796)

[Table 29 - Winners.java 51](#_Toc411285797)

**Table of Figures**

[Figure 1: Website Wide Number Generation: 7](#_Toc411285798)

[Figure 2: Multiple Users - Same number to guess 8](#_Toc411285799)

[Figure 3 - Bad Guess - Give Hint 8](#_Toc411285800)

[Figure 4: Good Guess and new number to guess 9](#_Toc411285801)

[Figure 5 - Showing of last winner with numTries 10](#_Toc411285802)

[Figure 6 : Error Messages and Feedback 11](#_Toc411285803)

[Figure 7 - Website Wide Generation + Users can guess at the same number 22](#_Toc411285804)

[Figure 8 - Guessing the Wrong Number 22](#_Toc411285805)

[Figure 9 - Correct Guesses with previous winner displayed. 23](#_Toc411285806)

[Figure 10 - Website Wide Generation + Users can guess at the same number 37](#_Toc411285807)

[Figure 11 - Guessing the Wrong Number 37](#_Toc411285808)

[Figure 12 - Correct Guesses with previous winner displayed. 38](#_Toc411285809)

[Figure 13 - Website Wide Number + Users can guess at the same number 54](#_Toc411285810)

[Figure 14 - Guessing the wrong number 54](#_Toc411285811)

[Figure 15 - Correct Guesses with previous winner displayed. 55](#_Toc411285812)

[Figure 16 - Testing of Persistence (Adding to the Database) 55](#_Toc411285813)

# Project Overview

*Problem: Learning J2EE Concepts and using these concepts to create a Multiplayer Number Guessing Game*

*Goal: Complete each stage of the project (projects) and see the differences between each of the stages.*

*Criteria for Success: Completion of each section of the project and a completion of the section in the progress document.*

*Assumptions: Basic HTML and Java Knowledge and technical document writing skills*

*Risks include: Failing to meet requirements of each section, failure to write a concise technical document,*

# Java EE Project Types

*The project will contain these smaller project types:*

# Java Server Pages (JSPs) and Servlets

*This project type utilizes the simple intercommunication between Java Server Pages and Servlets. It will mainly display the most basic version of the game and have a basis for future improvements (further projects)*

# Java Server Faces (JSFs)

*This project type utilizes the communication between JSF files and managed beans. It will display an upgraded version of the game and be easily upgradable for the next version of the game in future projects.*

# JSFs and Context and Dependency Injection (CDI)

# JSFs and Java Persistence

# Projects

*In this section, we will go through each section of the project and give insight into the page design and implementation testing that occurred during that section.*

# JSPs and Servlets

*This portion of the project dealt with the interaction between the JSP pages of the game and the game servlet. In the following page or so, there will be code snippets from the JSP pages that were created for this portion of the project, code snippets from the servlet and testing of all the required features at this point in time.*

# Web Pages - JSPs

*This is a screenshot of the main portion of the Index.jsp. It showcases the main form that users will use to login to the main game. It sets up the userbean and gives it the scope of session.*

Table 1: JSP of Index.jsp – Screenshot

|  |
| --- |
|  |

*Moving forward to the game.jsp, the screenshot on the following page (Table 2) shows the main form for use when the user is playing the game. The user will enter their number to guess and send it back to the game servlet for processing.*

*Notice on the page that I tell the page that it will be using the UserBean and GameBean as they will be storing the User Data and Game Data Respectively. As well, I made note to set placeholders for important information that needs to be seen by the user.*

*For example:*

*${win} will display to the winner with a congratulatory message with how many tries it took.*

*<jsp: getProperty name=”gamebean” property=lastWinner”/> will display the last known winner with how many tries it too them. – Note: Experiencing an issue with it overwriting the username of the player who wins with the last player to login.*

Table 2: JSP of game.jsp – Screenshot

|  |
| --- |
|  |

# Code - Servlet

In this section, we will talk about the design choices and implementation that went into creating the game servlet for the game.

**Table 3:**

This portion of the servlet is focusing on processing input from the index.jsp (logging in)

Starting in order of occurance, we start by initializing the beans for usuage in the servlet by getting the attributes from the request. Next we check if the username string that gets set to the “username” parameter from the request is not null and has a length greater than 0.

If it passes that condition, the log will print out a log with the username and then set the username property of myBean to that username. Otherwise, it will return to the login-page with an error that states that a username needs to be entered.

Table 3 - Servlet - Index Processing

|  |
| --- |
|  |

**Table 3.2 Parts 1-3:**

Due to the long screenshot of the game processing portion of the Game Servlet, It was decided to break it into three parts and explain each code section.

**Part 1:**

Firstly, we get the guess in the form of a string from the request which will be tested to make sure it is not null and has a length greater than zero.

If it fails the condition, an error message will appear on the page, letting the user know that there hasn’t been a guess entered yet.

Otherwise it sets the error attribute to be empty and parses the integer guess from the guess string and then logs the guess.

Table 4 : Game Processing Part 1

|  |
| --- |
|  |

**Part 2:**

This portion mainly deals with the random number generation and what occurs when the guess is equal to the number to guess.

At first, the number to guess is set to 0 so that the servlet knows to randomize the number at the beginning of the game.

It does this by checking if the number to guess is set to zero, then randomizes the number to guess between 1 and 1000. Then sets the property num2guess in the application wide GBean.

Before any checking if the num2Guess is going to be checked against the user’s guess. It is logged what the num2Guess is.

Checking the guess is quite simple, all that needs to be done is to check if the guess the user has made is equal to the num2guess that is inside the GBean (GameBean).

If that condition has been made, we set the win attribute of the request to a message designated for the user who won. This message contains a simple congrats with their username and number of tries it took them to guess correctly.

We then set the LastWinner property of the GBean with a message that will be shown to all other users when they make another guess, notifying them that the number has changed. A log is created of the Last Winner and then the num2Guess is randomly set and set inside the GBean once again.

To make sure that the number of tries is accurate for each player, I reset their number of tries when a user wins the game and forward it back to the game.jsp page.

Table 5 : Game Processing Part 2

|  |
| --- |
|  |

**Part 3:**

In this part, we will be going into the logic that occurs if the user has not guessed the number correctly.

If the guess they made was less than the number to guess, we increment the number of tries using the setNumTries method in the userbean myBean. We then set the numTries attribute of the request to the number of tries from the myBean and log that number of tries.

We then set the hint for the user (in this case, “You need to guess higher”) and then set the request attribute hint equal to that hint and forward back to the game.jsp to inform the user.

In the other case where the guess was higher than the num2guess, it states to the user to guess lower and the same code logic as before is run.

Table 6 : Game Processing Part 3

|  |
| --- |
|  |

# Testing

*In this section, we will be testing the web application to see if it adheres to the requirements that were set at the beginning of this project section.*

Figure 1: Website Wide Number Generation:

|  |
| --- |
| <jsp:useBean id="gamebean" class="Web\_Game.GameBean" scope = "application" />  GameBean GBean = (GameBean)request.getServletContext()  .getAttribute("gamebean");  log # 5 # GBean -num2Guess=660 |

Since GBean is application scope it will be available to all users who access the game page. The log shows that the current number to guess is 660.

Figure 2: Multiple Users - Same number to guess

|  |
| --- |
| log # 1 # username=Mike  log # 2 # username=Jake  log # 4 # Guess=456 – user – Mike guess  log # 6 # GBean -num2Guess=859 – stated number to guess.  log # 9 # Guess=678 – user - Jake guess  log # 11 # GBean -num2Guess=859 – never changed through the guess |

Here we see that there are two users, Mike and Jake.

Mike makes a guess of 456 and then the log shows that he is not correct and the number to guess is 859.

Jake makes a guess of 678 and then the log shows he is not correct as well and the number to guess is still 859.

|  |
| --- |
|  |

Figure 3 - Bad Guess - Give Hint

Number to guess: 356

User Enters 889 as a Guess.

|  |
| --- |
|  |

Hint Displayed: 889 is larger than 356 – Needed to guess lower

|  |
| --- |
|  |

|  |
| --- |
|  |

User Enters 45 as a guess

Hint Displayed: 45 is smaller than 356 – Needed to guess higher.

Figure 4: Good Guess and new number to guess

|  |
| --- |
|  |

Screenshot of my Ubuntu Virtual Machine Playing the game and getting the new Winner message when they try to guess next.

|  |  |
| --- | --- |
| Num2guess before win | Num2guess after win |
| log # 55 # GBean -num2Guess=927 | log # 61 # GBean -num2Guess=983 |

As you can see at log 55, the number to guess was 927. But after it was successfully guessed it changed to 983 via this code snippet:

|  |
| --- |
| num2Guess = Randomizer.Randomize(1, 1000)  GBean.setNum2Guess(num2Guess); |

After guessing correctly it resets the num2Guess by using my random class to generate a new random number.

It then sets the Num2Guess in the bean to be that new randomly generated number.

Figure 5 - Showing of last winner with numTries

|  |
| --- |
|  |

Upon entry of a new guess, the user sees that the last Winner was Jake with 2 tries at guessing the number.

The code snippets that shows how this is done is:

Once the number has been guessed correctly, we set the last winner text in the GBean for use in printing on the main game page.

|  |
| --- |
| GBean.setLastWinner(myBean.getUsername() + " with " + myBean.getNumTries() + " tries"); |

This is printed to the screen by using the getProperty using jsp tags and noting the property we want is “last winner”

|  |
| --- |
| <p>Last Winner: <jsp:getProperty name="gamebean" property="lastWinner"/></p> |

## 3.1.4 Error Messages:

**Login:**

Figure 6 : Error Messages and Feedback

|  |
| --- |
|  |

Code Snippets that explains how this is achieved

**Game Servlet:**

|  |
| --- |
| // Required Error message if any of the previous checking failed.  request.setAttribute("error", "You need to enter a username!"); request.getRequestDispatcher("/index.jsp").forward(request, response); |

If the username is not valid, we set the request attribute to be our error message “You need to enter a username!”

**Index.jsp**

|  |
| --- |
| <span class="error">${error}</span></center> |

This is the placeholder where the error message will be placed if an error occurs.

**Game:**

|  |
| --- |
|  |

Code snippets that show how this is achieved:

**Game Servlet:**

|  |
| --- |
| // Gets the guess form the request  String guessStr = request.getParameter("num2Guess");    // checks if the guessStr is null or less than or equal to 0.  if((guessStr == null)|| (guessStr.length() <= 0)) {    request.setAttribute ("error", "You have not entered a guess!");  request.getRequestDispatcher("/game.jsp")  .forward(request, response); |

**Game.jsp:**

|  |
| --- |
| <span class="error">${error}</span> |

This is the placeholder where the error message will be placed if an error occurs.

## 3.1.5 Feedback Message: Winning

|  |
| --- |
|  |

Code Snippets that show how this is achieved:

**Game Servlet:**

|  |
| --- |
| request.setAttribute ("win", "Congrats” + myBean.getUsername () + " You guessed right! after " + myBean.getNumTries() + " tries"); |

This sets the attribute named win in the request to be a string that contains the winning username and the number of tries it took them to answer correctly.

**Game.jsp:**

|  |
| --- |
| p>${win}</p> |

This is the placeholder on the page where this string will be deposited if the user wins.

## 3.1.6 Feedback: Number of Tries:

|  |
| --- |
|  |

Code Snippets that show how this is done:

**Game Servlet:**

|  |
| --- |
| myBean.setNumTries(); |

Calls this method whenever a try at guessing was attempted.

**UserBean.Java**

|  |
| --- |
| public void setNumTries() { this.numTries++; } |

This method increments the numTries variable when called.

# JSFs

*This portion of the project dealt with the interaction between the JSF pages of the game and the managed beans. In the following pages, there will be code snippets from the JSF pages that were created for this portion of the project, code snippets from the managed beans and testing of all the required features at this point in time.*

# Web Pages - JSFs

*This screenshot shows the main template that is being used for the game website. It highlights three sections, the header, the content and the footer. When a page uses this template, it can overwrite the sections by declaring the new section in the coding, otherwise any sections that aren’t declared will use the default from the template.*

Table 7: main template

|  |
| --- |
|  |

Moving onto the Index page, we use the template from Table 7 and declare a new content section that will be displayed when the user hits the index page. They will be shown a form to enter their username and begin playing the game. See Table 8.

Table 8: Index.xhtml

|  |
| --- |
|  |

When the user begins playing the game, the player’s content changes to a form that shows them their username and a spot to begin guessing the number to guess. The page includes input validation to make sure that the number guessed is within the range of 1-1000.

See Table 9.

Table 9: Game.xhtml

|  |
| --- |
|  |

The next page is an addon for the project which will display the list of previous winners. The user can go to this page by clicking the appropriate button on the game page to see a longer list of the previous winners. See Table 10 for Code.

Table 10: WinList.xhtml

|  |
| --- |
|  |

Table 10-2: WinList - Display Page

|  |
| --- |
|  |

# Code - Managed Beans

*The main managed bean that is being used in the game is the UserBean. It includes the many methods that are managed by the user such as:*

* *Setting the username.*
* *Setting the guess.*
* *Setting the hint when applicable.*
* *Incrementing the number of tries when applicable.*
* *Creating the application scope GameBean and setting the instance.*
* *Getting the min and max that the user can guess.*
* *Checking the guess against a randomly generated number in the GameBean.*
* *Informing user of winners and the amount of tries.*
* *Allowing users to see more than one previous winner*

*See Table 11.*

Table 11: UserBean.Java

|  |
| --- |
| package Web\_GameV2;  import java.io.Serializable;  import javax.faces.bean.ManagedBean;  import javax.faces.bean.ManagedProperty;  import javax.faces.bean.SessionScoped;  @ManagedBean(name="UserBean")  @SessionScoped // this scope lasts as long as the user is logged in  public class UserBean implements Serializable {  // Validate within JSF  private String Username = "";  private int numTries = 0;  private int guess;  private String hint = "";    @ManagedProperty("#{gameBean}")  private GameBean gameBean;  public void setGameBean(GameBean gameBean) {  this.gameBean = gameBean;  }    public GameBean getGameBean(){  return this.gameBean;  }    /\*\*  \* Max and min for JSF Validation  \*  \* @return  \*/  public int getMinGuess() {  return 1;  }  public int getMaxGuess() {  return 1000;  }  public String checkguess() {  logit.log("Before Guess=" + this.Username + " Guess=" + this.guess);  if (this.guess < gameBean.getRandom\_number()) {  this.hint = "Guess Higher";  updateNumTries();  logit.log("Number to guess = " + gameBean.getRandom\_number());  logit.log(this.Username + " guessed " + this.guess);  }  if (this.guess > gameBean.getRandom\_number()) {  this.hint = "Guess Lower";  updateNumTries();  logit.log("Number to guess = " + gameBean.getRandom\_number());  logit.log(this.Username + " guessed " + this.guess);  }  if (this.guess == gameBean.getRandom\_number()) {  this.hint = "Correct!";  updateNumTries();  gameBean.setLastWinner(this.Username);  gameBean.setNumTries(this.numTries);  logit.log("Number to guess = " + gameBean.getRandom\_number());  logit.log("last winner = " + gameBean.getLastWinner());  gameBean.addWinnerToList(gameBean.getLastWinner());  gameBean.resetRandom\_number();  resetNumTries();    }    logit.log("After Guess=" + this.Username + " Guess=" + this.guess);    return "game.xhtml";  }    public UserBean() {    }  public String startplayer() {  return "game.xhtml";  }  /\*\*  \* @return the name  \*/  public String getUserName() {  return Username;  }  /\*\*  \* @param name the name to set  \*/  public void setUserName(String Username) {  this.Username = Username;  }  /\*\*  \* @return the guess  \*/  public int getGuess() {  return guess;  }  /\*\*  \* @param guess the guess to set  \*/  public void setGuess(int guess) {  this.guess = guess;  }  /\*\*  \* @return the hint  \*/  public String getHint() {  return hint;  }  /\*\*  \* @return the numTries  \*/  public int getNumTries() {  return numTries;  }  /\*\*  \* @param numTries the numTries to set  \*/  public void updateNumTries() {  this.numTries++;  }  public void resetNumTries(){  numTries = 0;  }    } |

To implement the features to display the last winner and generate the random number for guessing plus show the previous winners list. The GameBean was created to hold application scope values that will be displayed to or used by each user that plays the game. See Table 11.

Table 11: GameBean.java

|  |
| --- |
| package Web\_GameV2;  import java.io.Serializable;  import java.util.ArrayList;  import javax.faces.bean.ManagedBean;  import javax.faces.bean.ApplicationScoped;  /\* @author Mike  \*/  @ManagedBean(eager=true)  @ApplicationScoped  public class GameBean implements Serializable{    private String lastWinner="";  private int numTries;  ArrayList<String> Winners = new ArrayList<String>();    // The random number to guess  private int random\_number;    /\*\*  \* Creates a new instance of GameBean  \*/  public GameBean() {  this.random\_number = Randomizer.Randomize(1,1000);  }  /\*\*  \* @return the lastWinner  \*/  public String getLastWinner() {  return lastWinner + " with " + numTries + " tries";  }  /\*\*  \* @param lastWinner the lastWinner to set  \*/  public void setLastWinner(String lastWinner) {  this.lastWinner = lastWinner;  }  /\*\*  \* @return the numTries  \*/  public int getNumTries() {  return numTries;  }  /\*\*  \* @param numTries the numTries to set  \*/  public void setNumTries(int numTries) {  this.numTries = numTries;  }  /\*\*  \* @return the random\_number  \*/  public int getRandom\_number() {  return random\_number;  }  /\*\*  \* @param random\_number the random\_number to set  \*/  public void resetRandom\_number() {  this.random\_number = Randomizer.Randomize(1, 1000);  }  // Adds last winner to an array of winners.  public void addWinnerToList(String winner){  Winners.add(winner);  }  public ArrayList getWinnerList(){    return Winners;  }  } |

# Testing

*In this section, we will be testing the web application to see if it still adheres to the requirements that were set for the beginning project phrase.*

*When the GameBean constructor is called, it sets the random\_number to be a randonmly generated by my Randomizer class (used in previous project). Users can guess at this number and will be guessing against the same number as the GameBean is an application scoped bean. See Logs below in Figure 7.*

Figure 7 - Website Wide Generation + Users can guess at the same number

|  |
| --- |
| public GameBean() {  this.random\_number = Randomizer.Randomize(1,1000);  }  Info: log # 1 # Before Guess=Mike Guess=456  Info: log # 2 # Number to guess = 210  Info: log # 3 # Mike guessed 456  Info: log # 4 # After Guess=Mike Guess=456  ----------------------------------------------------------  Info: log # 5 # Before Guess=Jake Guess=100  Info: log # 6 # Number to guess = 210  Info: log # 7 # Jake guessed 100  Info: log # 8 # After Guess=Jake Guess=100 |

Figure 8 - Guessing the Wrong Number

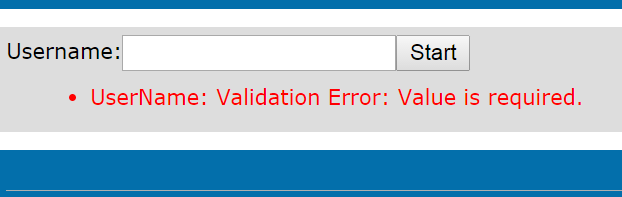
|  |
| --- |
| Info: log # 22 # Number to guess = 828 <- Correct Number to Guess    Bob guessed higher than the number to guess and was greeted with a hint to try guessing lower.    Bob guessed lower than the number to guess and was greeted with a hint to try guessing higher. |

Figure 9 - Correct Guesses with previous winner displayed.

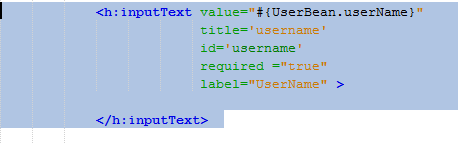
|  |
| --- |
| Info: log # 6 # Number to guess = 516  See below that Mike answered the number correctly and the feedback spot on the page now displays that he won with 2 tries.    When Jake makes his next Guess, the Last Winner field is updated and gives the hint that he should guess higher. That is correct as the number to guess has been randomized once again and is higher than before. See Log #10.    Info: log # 10 # Number to guess = 988 |

## 3.2.4. Error Messages:

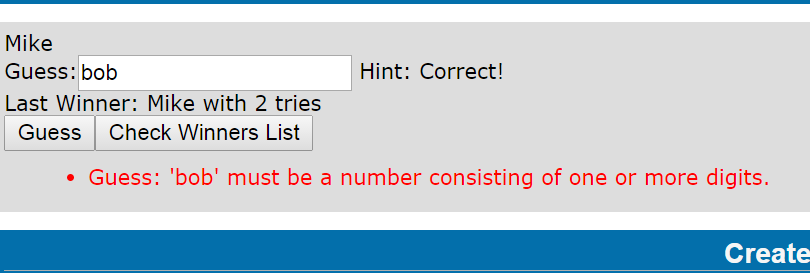
**Username Error:**



This error occurs because the username is a required field for the form to be submitted as shown in the following code snippet:

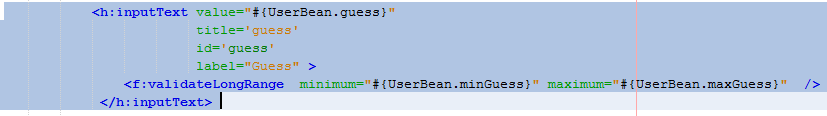


**Guess Error: Invalid type**

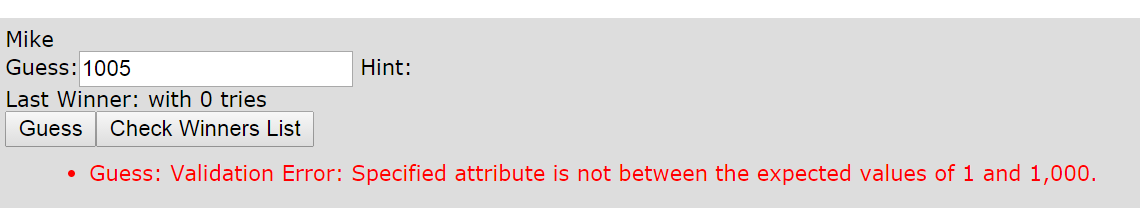


As you can see here, the field in the form does not allow any input that is not a number consisting of more than one or more digits.

This error occurs because of the intercommunication with the managed beans, since it is being put into an int variable inside the UserBean directly, there is no way that a string will be allowed to be placed within the variable. See the following code snippet below to see how this is achieved:



**Guess Error: Invalid Range**

****

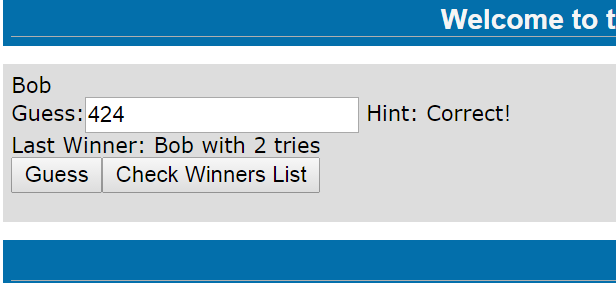
If the user guessed outside the appropriate range of 1-1000, the user is shown a validation error which tells them that they should only guess between 1 -1000.

This is achieved in the JSF by the following code snippet which validates using a validateLongRange tag:

****

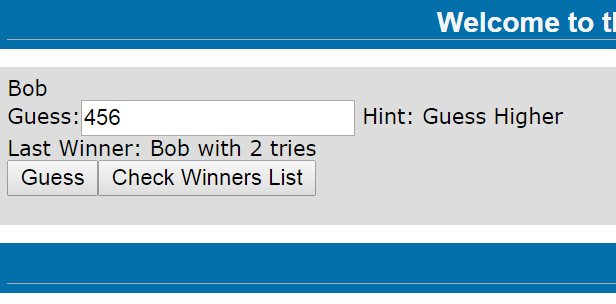
## 3.2.5. Feedback Messages:

Correct:



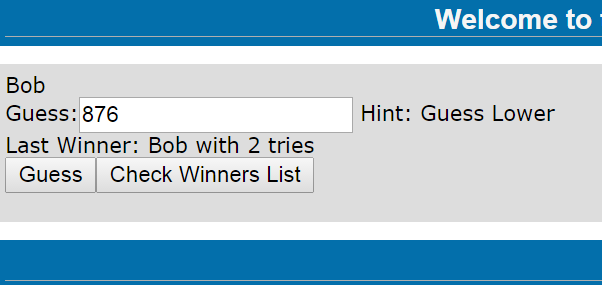
This user correctly guessed the number which was 424 and was greeted with the feedback message that the guess given was correct.

Higher:



The user made a guess that was lower than the number to be guessed and was greeted with the feedback message that he needs to guess higher than 456.

Lower:



The user made a guess that was higher than the number to be guessed and was greeted with the feedback message that he needs to guess lower than 876.

# JSFs – CDI

*This portion of the project deals with the interaction between the JSF pages of the game and the CDI injection beans. In the following pages, there will be code snippets/screenshots from the JSF pages that were created for this portion of the project, code snippets from the CDI beans and testing of all the required features at this point in time*.

# Web Pages – JSFs

*This screenshot shows the main template that is being used for the game website. It highlights three sections, the header, the content and the footer. When a page uses this template, it can overwrite the sections by declaring the new section in the coding, otherwise any sections that aren’t declared will use the default from the template.*

Table 12: main template

|  |
| --- |
|  |

Moving onto the Index page, we use the template from Table 12 and declare a new content section that will be displayed when the user hits the index page. They will be shown a form to enter their username which will be placed in the session scoped bean called UserBean. See Table 13.

Table 13 - Index.xhtml

|  |
| --- |
|  |

*.*

When the user begins playing the game, the player’s content changes to a form that shows them their username and a spot to begin guessing the number to guess. The page includes input validation to make sure that the number guessed is within the range of 1-1000.

As well the page contains a spot for a hint to be shown to the user that lets them know if they are higher or lower than the answer or gotten the correct answer. Upon submission of the form, the user calls the logic bean which will check the guess they have given.

See Table 14.

Table 14 - Game.xhtml

|  |
| --- |
|  |

The next page is an addon for the project which will display the list of previous winners. The user can go to this page by clicking the appropriate button on the game page to see a longer list of the previous winners. See Table 15 for Code.

|  |
| --- |
|  |

Table 15: WinList.xhtml

Possible Display of WinList:

Table 16 - WinList.xhtml - Display

|  |
| --- |
|  |

# Code – CDI Beans

*This file contains most of the game logic that is applied during the runtime of the number guessing game.*

*During the startup, three beans are injected into this logic bean:*

* *Singleton Bean – gameBean*
* *Session Bean – userBean*
* *Request Bean – responseBean*

*When the user submits a guess, it is checked against the num2guess and depending on how close the guess is to the correct answer, a hint is set and the user is notified of the situation.*

Table 17 - logic.java

|  |
| --- |
| *package com.NexusX.web\_project;*  *import javax.enterprise.context.RequestScoped;*  *import javax.inject.Inject;*  *import javax.inject.Named;*  */\*\**  *\* @author Mike Dean*  *\*/*  *@RequestScoped*  *@Named*  *public class Logic {*  *@Inject*  *GameBean gameBean;*  *@Inject*  *UserBean userBean;*  *@Inject*  *ResponseBean responseBean;*  *public Logic() {*  *}*  *public String checkguess() {*  *logit.log("Before Guess=" + userBean.getUsername() + " Guess=" + userBean.getGuess());*  *if (userBean.getGuess() < gameBean.getNum2Guess()) {*  *responseBean.setGuessResponse("Guess Higher");*  *userBean.updateTries();*  *logit.log("Number to guess = " + gameBean.getNum2Guess());*  *logit.log(userBean.getUsername() + " guessed " + userBean.getGuess());*  *}*  *if (userBean.getGuess() > gameBean.getNum2Guess()) {*  *responseBean.setGuessResponse("Guess Lower");*  *userBean.updateTries();*  *logit.log("Number to guess = " + gameBean.getNum2Guess());*  *logit.log(userBean.getUsername() + " guessed " + userBean.getGuess());*  *}*  *if (userBean.getGuess() == gameBean.getNum2Guess()) {*  *responseBean.setGuessResponse("Correct!");*  *userBean.updateTries();*  *gameBean.setLastWinner(userBean.getUsername());*  *gameBean.setNumTries(userBean.getNumTries());*  *logit.log("Number to guess = " + gameBean.getNum2Guess());*  *logit.log("last winner = " + gameBean.getLastWinner());*  *gameBean.addWinnerToList(gameBean.getLastWinner());*  *gameBean.resetnum2Guess();*  *userBean.resetTries();*  *}*  *logit.log("After Guess=" + userBean.getUsername() + " Guess=" + userBean.getGuess());*    *return "game.xhtml";*  *}*  *}* |

This is the GameBean that is a Singleton bean that persists application wide and every user can interact with this bean.

The properties that are set and get in this bean are:

* Last Winner – Last Winner of the Game.
* Num2Guess – Number to be guessed.
* numTries – Number of Tries of User who has won.
* Winners Array – Array of Winners.

Table 18 - GameBean.java

|  |
| --- |
| package com.NexusX.web\_project;  import java.util.ArrayList;  import javax.inject.Named;  import javax.inject.Singleton;  @Singleton  @Named  public class GameBean {    private String lastWinner="";  ArrayList<String> Winners = new ArrayList<String>();  private int num2Guess;  private int numTries;  /\*\*  \* Creates a new instance of GameBean  \*/  public GameBean() {  this.num2Guess = Randomizer.Randomize(1,1000);  }  /\*\*  \* @return the lastWinner  \*/  public String getLastWinner() {  return lastWinner + " with " + numTries + " tries";  }  /\*\*  \* @param lastWinner the lastWinner to set  \*/  public void setLastWinner(String lastWinner) {  this.lastWinner = lastWinner;  }  /\*\*  \* @return the numTries  \*/  public int getNumTries() {  return numTries;  }  /\*\*  \* @param numTries the numTries to set  \*/  public void setNumTries(int numTries) {  this.numTries = numTries;  }  /\*\*  \* @return the num2Guess  \*/  public int getNum2Guess() {  return num2Guess;  }  /\*\*  \* @param num2Guess the num2Guess to set  \*/  public void resetnum2Guess() {  this.num2Guess = Randomizer.Randomize(1, 1000);  }    // Adds last winner to an array of winners.  public void addWinnerToList(String winner){  Winners.add(winner);  }  //Gets the arraylist  public ArrayList getWinnerList(){    return Winners;  }  } |

The UserBean is a session-scoped bean that is created every time a new user connects to the game website.

The UserBean contains and sets the following properties:

* Username – Name of the User
* Guess – The guess of the user
* numTries – Number of tries user takes to guess the number.

Table 19 – UserBean.java

|  |
| --- |
| package com.NexusX.web\_project;  import java.io.Serializable;  import javax.enterprise.context.SessionScoped;  import javax.inject.Named;  /\*\*  \*  \* @author Mike Dean  \*/  @Named  @SessionScoped  public class UserBean implements Serializable {  private String Username = "";  private int guess;  private int numTries;  public UserBean() {  }  /\*\*  \* @return the username  \*/  public String getUsername() {  return Username;  }  /\*\*  \* @param name the name to set  \*/  public void setUsername(String userName) {  this.Username = userName;  }  /\*\*  \* @return the guesses  \*/  public int getNumTries() {  return numTries;  }  /\*\*  \* @param resetTries reset the number of tries.  \*/  public void resetTries() {  this.numTries = 0;  }  public void updateTries() {  this.numTries++;  }  /\*\*  \* Max and min for JSF Validation  \*  \* @return  \*/  public int getMinGuess() {  return 1;  }  public int getMaxGuess() {  return 1000;  }  /\*\*  \* @return the guess  \*/  public int getGuess() {  return guess;  }  /\*\*  \* @param guess the guess to set  \*/  public void setGuess(int guess) {  this.guess = guess;  }  } |

The responseBean is a very lightly used request-scoped which gets and sets the one property of the guessResponse.

It is used only when the user has sent a guess via the form and the game page requests a response to the guess.

See Table 20.

Table 20 – ResponseBean.java

|  |
| --- |
| package com.NexusX.web\_project;  import javax.enterprise.context.RequestScoped;  import javax.inject.Named;  @RequestScoped  @Named  public class ResponseBean {  private String guessResponse = "";  public ResponseBean() {  }  /\*\*  \* @return the guessResponse  \*/  public String getGuessResponse() {  return guessResponse;  }  /\*\*  \* @param guessResponse the guessResponse to set  \*/  public void setGuessResponse(String guessResponse) {  this.guessResponse = guessResponse;  }  } |

# Testing

*In this section, we will be testing the web application to see if it still adheres to the requirements that were set for the beginning project phrase.*

*When the GameBean constructor is called, it sets the random\_number to be a randonmly generated by my Randomizer class (used in previous project). Users can guess at this number and will be guessing against the same number as the GameBean is an application scoped bean. See Logs below in Figure 10.*

Figure 10 - Website Wide Generation + Users can guess at the same number

|  |
| --- |
| public GameBean() {  this.random\_number = Randomizer.Randomize(1,1000);  }  Info: log # 1 # Before Guess=Mike Guess=867  Info: log # 2 # Number to guess = 400  Info: log # 3 # Mike guessed 867  Info: log # 4 # After Guess=Mike Guess=867  ----------------------------------------------------------  Info: log # 5 # Before Guess=Jake Guess=150  Info: log # 6 # Number to guess = 867  Info: log # 7 # Jake guessed 150  Info: log # 8 # After Guess=Jake Guess=150 |

Figure 11 - Guessing the Wrong Number

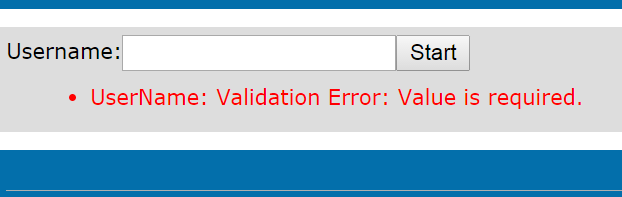
|  |
| --- |
| Info: log # 22 # Number to guess = 828 <- Correct Number to Guess    Desiree guessed higher than the number to guess and was greeted with a hint to try guessing lower.    Desiree guessed lower than the number to guess and was greeted with a hint to try guessing higher. |

Figure 12 - Correct Guesses with previous winner displayed.

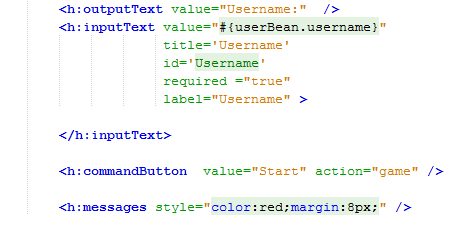
|  |
| --- |
| Info: log # 2 # Number to guess = 829  See below that Mike answered the number correctly and the feedback spot on the page now displays that he won with 6 tries.    When Desiree makes her next Guess, the Last Winner field is updated and gives the hint that she should guess higher. That is correct as the number to guess has been randomized once again and is higher than before. See Log #46.    Info: log # 46 # Number to guess = 113 |

## 3.3.4. Error Messages:

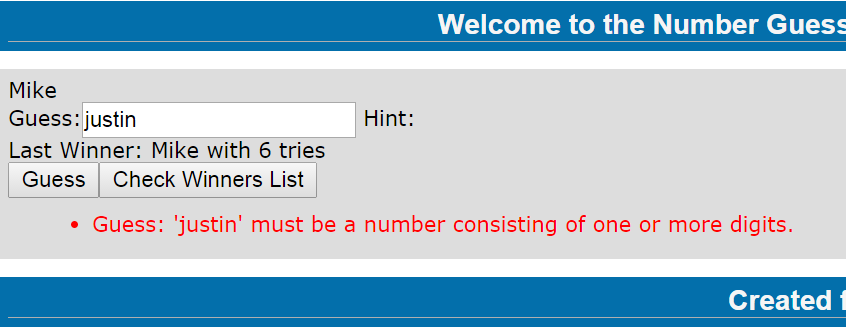
**Username Error:**



This error occurs because the username is a required field for the form to be submitted as shown in the following code snippet:

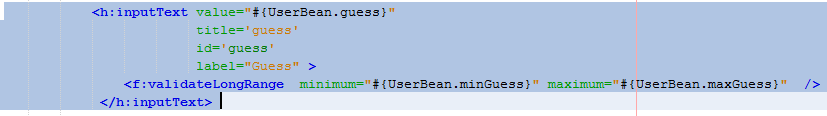


**Guess Error: Invalid type**

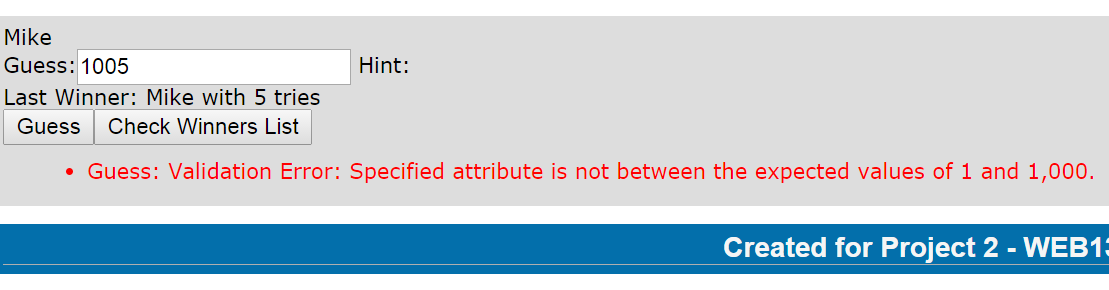


As you can see here, the field in the form does not allow any input that is not a number consisting of more than one or more digits.

This error occurs because of the intercommunication with the managed beans, since it is being put into an int variable inside the UserBean directly, there is no way that a string will be allowed to be placed within the variable. See the following code snippet below to see how this is achieved:



**Guess Error: Invalid Range**

****

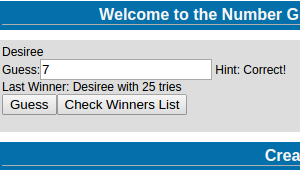
If the user guessed outside the appropriate range of 1-1000, the user is shown a validation error which tells them that they should only guess between 1 -1000.

This is achieved in the JSF by the following code snippet which validates using a validateLongRange tag:

****

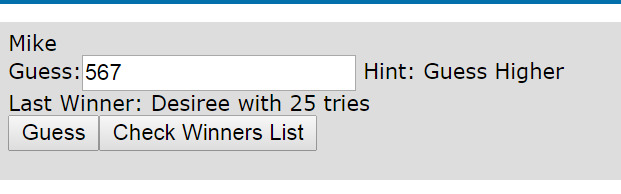
## 3.3.5. Feedback Messages:

Correct:



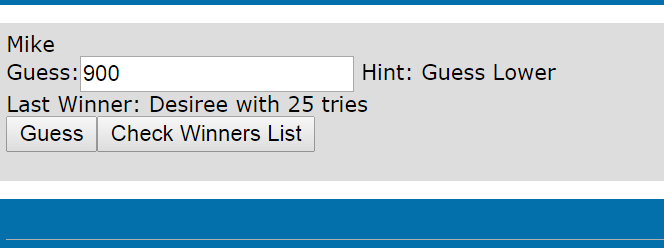
Desiree correctly guessed the number which was 7 and was greeted with the feedback message that the guess given was correct.

Higher:



Mike made a guess that was lower than the number to be guessed and was greeted with the feedback message that he needs to guess higher than 567.

Lower:



Mike made a guess that was higher than the number to be guessed and was greeted with the feedback message that he needs to guess lower than 900.

# JSF and Java Persistence

In the final section of the project, we will deal with the inner workings of Persistence using a Java database and Enterprise Java Beans. In this section, we have added a few features and reworked the game logic for the number guessing game that has developed during this project of projects.

Moving forward, we will go through each of the following sections and describe any changes or updates that occurred in this section.

# Web Pages – JSFs

The main template of the game has not really changed since Project. The only updated changes were to the greeting message that is displayed via the template.

Table 21 - main\_template

|  |
| --- |
|  |

The index page of the game has not changed from project 2. The basic premise of this page is to guide the user to enter their username and begin playing the game.

Table 22 - index.xhtml

|  |
| --- |
| <?xml version='1.0' encoding='UTF-8' ?>  <!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"  template="./main\_Template.xhtml"  xmlns:h="http://xmlns.jcp.org/jsf/html"  xmlns:f="http://xmlns.jcp.org/jsf/core">  <!-- Only define the content section, and  missing parts are taken from the template -->  <ui:define name="content">  <h:form>  <h:outputText value="Username:" />  <h:inputText value="#{userBean.username}"  title='Username'  id='Username'  required ="true"  label="Username" >    </h:inputText>  <h:commandButton value="Start" action="game" />  <h:messages style="color:red;margin:8px;" />  </h:form>  </ui:define>  </ui:composition> |

This is the main page that the users will frequent when playing the game. Compared to the game page of previous projects, a couple new items have been added/changed:

* Added Points Remaining Item (Out of 100)
* Changed Link to the Winner’s List
* Placed command button linking to the winner’s list in separate form to separate gameplay from checking the winner’s list.

See Table 23.

Table 23 - game.xhtml

|  |
| --- |
|  |

This page was added in Project 4 to be an Output Page for the user if they failed to guess the number correctly before running out of points. The page displays the message of game over and allows the user to retry playing the game and is transferred to the login page again. See Table 24.

Table 24 - gameOver.xhtml

|  |
| --- |
|  |

The List page was automatically generated when NetBeans created JSF Files from the Data source (my Database). However I changed a few values to get the correct result that was being aimed for.

* Added Title of “Winner Database Entries”
* Changed some of the (CRUD) generated headings.
* Made this page the result of clicking the Winner List Button on the game page.

Table 25 - List.xhtml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml"  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"  xmlns:h="http://xmlns.jcp.org/jsf/html"  xmlns:f="http://xmlns.jcp.org/jsf/core">  <ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"  template="../main\_Template.xhtml"  xmlns:h="http://xmlns.jcp.org/jsf/html"  xmlns:f="http://xmlns.jcp.org/jsf/core"  xmlns="http://www.w3.org/1999/xhtml">    <ui:define name="content">  <h:outputText value="Winner Database Entries:"/>  <h:form styleClass="jsfcrud\_list\_form">  <h:panelGroup rendered="#{winnersController.items.rowCount > 0}">  <h:commandLink action="#{winnersController.previous}" value="#{bundle.Previous} #{winnersController.pagination.pageSize}" rendered="#{winnersController.pagination.hasPreviousPage}"/>&nbsp;  <h:commandLink action="#{winnersController.next}" value="#{bundle.Next} #{winnersController.pagination.pageSize}" rendered="#{winnersController.pagination.hasNextPage}"/>&nbsp;  <h:dataTable value="#{winnersController.items}" var="item" border="0" cellpadding="2" cellspacing="0" rowClasses="jsfcrud\_odd\_row,jsfcrud\_even\_row" rules="all" style="border:solid 1px">  <h:column>  <f:facet name="header">  <h:outputText value="Username"/>  </f:facet>  <h:outputText value="#{item.username}"/>  </h:column>  <h:column>  <f:facet name="header">  <h:outputText value="Number To Guess"/>  </f:facet>  <h:outputText value="#{item.numbertoguess}"/>  </h:column>  <h:column>  <f:facet name="header">  <h:outputText value="Number of Tries"/>  </f:facet>  <h:outputText value="#{item.tries}"/>  </h:column>  <h:column>  <f:facet name="header">  <h:outputText value="Points Remaining (Out of 100)"/>  </f:facet>  <h:outputText value="#{item.pointsremain}"/>  </h:column>  </h:dataTable>  </h:panelGroup>  </h:form>  </ui:define>  </ui:composition>  </html> |

# Code – CDI Beans / EJB Bean

The main difference between this Logic.java and the previous version in Project 3 is changing this bean into an Enterprise Java Bean by adding few lines and changing the game logic.

The lines that changed were:

* Added statement “import javax.ejb.Stateless;” - Made bean stateless Enterprise Bean.
* Added statement “import javax.persistence.EntityManager; - Allowed usage of EntityManager.
* Added statement “import javax.persistence.PersistenceContext;” – Allowed usage of PersistenceContext.
* Created Instance of EntityManager to be used to persist
* Changed Logic to include checking points before continuing on with logic of game. If points = 0, reroute user to the gameOver page.
* Added call to method to decrement points from user if they failed to answer correctly.
* When user correctly guesses the right now, an instance of the entity class is created and each of the user’s details are written into an object of the entity class then persisted using the EntityManager instance.
* After being persisted, number being guessed is reset and the number of tries/points are reset so that a new game can begin.

Table 26 - Logic.java

|  |
| --- |
| package com.nexusx.webproject;  import javax.ejb.Stateless;  import javax.inject.Inject;  import javax.inject.Named;  import javax.persistence.EntityManager;  import javax.persistence.PersistenceContext;  /\*\*  \* @author Mike Dean  \*/  @Stateless  @Named  public class Logic {  @Inject  GameBean gameBean;  @Inject  UserBean userBean;  @Inject  ResponseBean responseBean;    @PersistenceContext  EntityManager em;  public Logic() {  }  public String checkguess() {    if (userBean.getNumPoints() == 0)  return "gameOver.xhtml";    else{  logit.log("Before Guess=" + userBean.getUsername() + " Guess=" + userBean.getGuess());  if (userBean.getGuess() < gameBean.getNum2Guess()) {  responseBean.setGuessResponse("Guess Higher");  userBean.updateTries();  userBean.decrementPoints();  logit.log("Number to guess = " + gameBean.getNum2Guess());  logit.log(userBean.getUsername() + " guessed " + userBean.getGuess());  }  if (userBean.getGuess() > gameBean.getNum2Guess()) {  responseBean.setGuessResponse("Guess Lower");  userBean.updateTries();  userBean.decrementPoints();  logit.log("Number to guess = " + gameBean.getNum2Guess());  logit.log(userBean.getUsername() + " guessed " + userBean.getGuess());  }  if (userBean.getGuess() == gameBean.getNum2Guess()) {  responseBean.setGuessResponse("Correct!");  userBean.updateTries();  gameBean.setLastWinner(userBean.getUsername());  gameBean.setNumTries(userBean.getNumTries());  logit.log("Number to guess = " + gameBean.getNum2Guess());  logit.log("last winner = " + gameBean.getLastWinner());  logit.log("Number of Points Remaining = " + userBean.getNumPoints());    // Add guess winner in DB...first make guess object  Winners winner = new Winners();  winner.setUsername(userBean.getUsername());  winner.setNumbertoguess(gameBean.getNum2Guess());  winner.setTries(gameBean.getNumTries());  winner.setPointsremain(userBean.getNumPoints());    //Save the new object as a row in the table  em.persist(winner);    gameBean.resetnum2Guess();  userBean.resetTries();  userBean.resetNumPoints();    }  logit.log("After Guess=" + userBean.getUsername() + " Guess=" + userBean.getGuess());    return "game.xhtml";  }  }  } |

This version of GameBean has not changed much from the previous version of GameBean except for the following change.

* Removed need for ArrayList as Database became a viable solution for storing winner user data.

Table 27 - GameBean.java

|  |
| --- |
| package com.nexusx.webproject;  import javax.inject.Named;  import javax.inject.Singleton;  @Singleton  @Named  public class GameBean {  private String lastWinner="";  private int num2Guess;  private int numTries;  /\*\*  \* Creates a new instance of GameBean  \*/  public GameBean() {  this.num2Guess = Randomizer.Randomize(1,1000);  }  /\*\*  \* @return the lastWinner  \*/  public String getLastWinner() {  return lastWinner + " with " + numTries + " tries";  }    /\*\*  \* @param lastWinner the lastWinner to set  \*/  public void setLastWinner(String lastWinner) {  this.lastWinner = lastWinner;  }  /\*\*  \* @return the numTries  \*/  public int getNumTries() {  return numTries;  }  /\*\*  \* @param numTries the numTries to set  \*/  public void setNumTries(int numTries) {  this.numTries = numTries;  }  /\*\*  \* @return the num2Guess  \*/  public int getNum2Guess() {  return num2Guess;  }  /\*\*  \* @param num2Guess the num2Guess to set  \*/  public void resetnum2Guess() {  this.num2Guess = Randomizer.Randomize(1, 1000);  }    } |

This version of UserBean does not differ much from the previous version that was used in Project 3. However in this project, a new value was added for innovation for this updated project.

* numPoints – Implemented a points system that starts from 100 and is decremented by 5 each time a user fails to guess the number correctly.

Table 28 - UserBean.java

|  |
| --- |
| package com.nexusx.webproject;  import java.io.Serializable;  import javax.enterprise.context.SessionScoped;  import javax.inject.Named;  /\*\*  \* @author Mike Dean  \*/  @Named  @SessionScoped  public class UserBean implements Serializable {  private String Username = "";  private int guess;  private int numTries;  private int numPoints = 100;  public UserBean() {  }  /\*\*  \* @return the name  \*/  public String getUsername() {  return Username;  }  /\*\*  \* @param name the name to set  \*/  public void setUsername(String userName) {  this.Username = userName;  }  /\*\*  \* @return the guesses  \*/  public int getNumTries() {  return numTries;  }  /\*\*  \* @param resetTries reset the number of tries.  \*/  public void resetTries() {  this.numTries = 0;  }  public void updateTries() {  this.numTries++;  }    /\*\*  \* Max and min for JSF Validation  \*  \* @return  \*/  public int getMinGuess() {  return 1;  }  public int getMaxGuess() {  return 1000;  }    /\*\*  \* @return the guess  \*/  public int getGuess() {  return guess;  }  /\*\*  \* @param guess the guess to set  \*/  public void setGuess(int guess) {  this.guess = guess;  }  /\*\*  \* @return the numPoints  \*/  public int getNumPoints() {  return numPoints;  }  /\*\*  \* Rest Points to 100.  \*/  public void resetNumPoints() {  numPoints = 100;  }  /\*\*  \* @param numPoints the numPoints to set  \*/  public void decrementPoints() {  this.numPoints -= 5;  }  } |

This class is the result of creating the Entity Classes from the data source which contains the database which will store the winning user data. It contains methods to set and get storable values from the game and is serializable/persistable.

Table 29 - Winners.java

|  |
| --- |
| package com.nexusx.webproject;  import java.io.Serializable;  import javax.persistence.Basic;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.NamedQueries;  import javax.persistence.NamedQuery;  import javax.persistence.Table;  import javax.validation.constraints.Size;  import javax.xml.bind.annotation.XmlRootElement;  /\*\*  \*  \* @author Mike  \*/  @Entity  @Table(name = "WINNERS")  @XmlRootElement  @NamedQueries({  @NamedQuery(name = "Winners.findAll", query = "SELECT w FROM Winners w"),  @NamedQuery(name = "Winners.findById", query = "SELECT w FROM Winners w WHERE w.id = :id"),  @NamedQuery(name = "Winners.findByUsername", query = "SELECT w FROM Winners w WHERE w.username = :username"),  @NamedQuery(name = "Winners.findByNumbertoguess", query = "SELECT w FROM Winners w WHERE w.numbertoguess = :numbertoguess"),  @NamedQuery(name = "Winners.findByTries", query = "SELECT w FROM Winners w WHERE w.tries = :tries"),  @NamedQuery(name = "Winners.findByPointsremain", query = "SELECT w FROM Winners w WHERE w.pointsremain = :pointsremain")})  public class Winners implements Serializable {  private static final long serialVersionUID = 1L;  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Basic(optional = false)  @Column(name = "ID")  private Integer id;  @Size(max = 50)  @Column(name = "USERNAME")  private String username;  @Column(name = "NUMBERTOGUESS")  private Integer numbertoguess;  @Column(name = "TRIES")  private Integer tries;  @Column(name = "POINTSREMAIN")  private Integer pointsremain;  public Winners() {  }  public Winners(Integer id) {  this.id = id;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getUsername() {  return username;  }  public void setUsername(String username) {  this.username = username;  }  public Integer getNumbertoguess() {  return numbertoguess;  }  public void setNumbertoguess(Integer numbertoguess) {  this.numbertoguess = numbertoguess;  }  public Integer getTries() {  return tries;  }  public void setTries(Integer tries) {  this.tries = tries;  }  public Integer getPointsremain() {  return pointsremain;  }  public void setPointsremain(Integer pointsremain) {  this.pointsremain = pointsremain;  }  @Override  public int hashCode() {  int hash = 0;  hash += (id != null ? id.hashCode() : 0);  return hash;  }  @Override  public boolean equals(Object object) {  // TODO: Warning - this method won't work in the case the id fields are not set  if (!(object instanceof Winners)) {  return false;  }  Winners other = (Winners) object;  if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {  return false;  }  return true;  }  @Override  public String toString() {  return "com.nexusx.webproject.Winners[ id=" + id + " ]";  }  } |

# Testing

Figure 13 - Website Wide Number + Users can guess at the same number

|  |
| --- |
| public GameBean() {  this.random\_number = Randomizer.Randomize(1,1000);  }  Info: log # 11 # Before Guess=Mike Guess=765  Info: log # 12 # Number to guess = 659  Info: log # 13 # Mike guessed 765  Info: log # 14 # After Guess=Mike Guess=765  ----------------------------------------------------------  Info: log # 15 # Before Guess=Jake Guess=450  Info: log # 16 # Number to guess = 659  Info: log # 17 # Jake guessed 450  Info: log # 18 # After Guess=Jake Guess=450 |

Figure 14 - Guessing the wrong number

|  |
| --- |
| Info: log # 2 # Number to guess = 917 <- Correct Number to Guess    George Dean guessed higher than the number to guess and was greeted with a hint to try guessing lower.    Mike Dean guessed lower than the number to guess and was greeted with a hint to try guessing higher. |

Figure 15 - Correct Guesses with previous winner displayed.

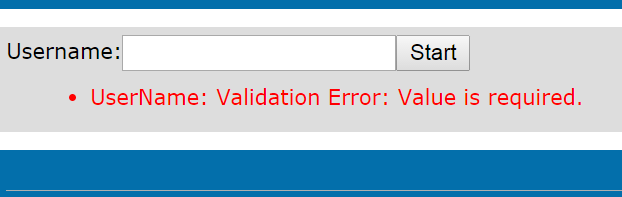
|  |
| --- |
| Info: log # 14 # Number to guess = 36  See below Sly answered the number correctly and the feedback spot on the page now displays that he won with 4 tries.    When Jake Jollimore makes his next Guess, the Last Winner field is updated and gives the hint that she should guess higher. That is correct as the number to guess has been randomized once again and is higher than before. See Log #19.    Info: log # 19 # Number to guess = 803 |

Figure 16 - Testing of Persistence (Adding to the Database)

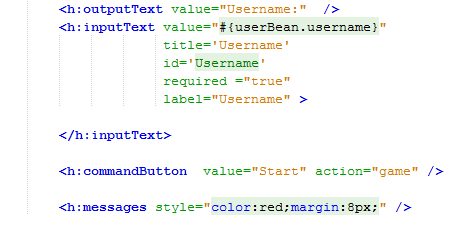
|  |
| --- |
| Bob Thorton correctly answers the number correctly. See Log 67#    Info: log # 67 # Number to guess = 803  After being added to the database. His entry shows up on the database list on the winning page. See Below: |

## 3.4.4. Error Messages:

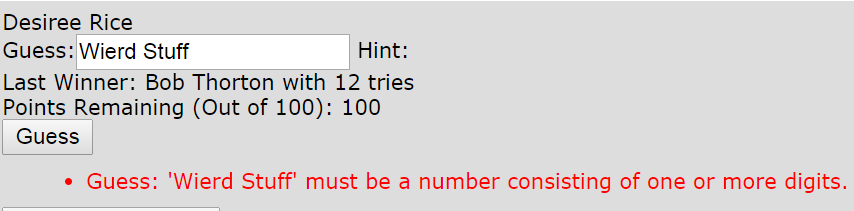
**Username Error:**



This error occurs because the username is a required field for the form to be submitted as shown in the following code snippet:

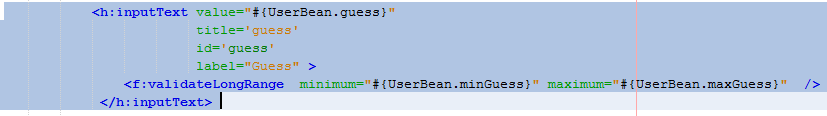


**Guess Error: Invalid type**



As you can see here, the field in the form does not allow any input that is not a number consisting of more than one or more digits.

This error occurs because of the intercommunication with the managed beans, since it is being put into an int variable inside the UserBean directly, there is no way that a string will be allowed to be placed within the variable. See the following code snippet below to see how this is achieved:



**Guess Error: Invalid Range**

****

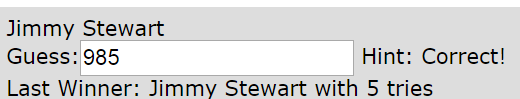
If the user guessed outside the appropriate range of 1-1000, the user is shown a validation error which tells them that they should only guess between 1 -1000.

This is achieved in the JSF by the following code snippet which validates using a validateLongRange tag:

****

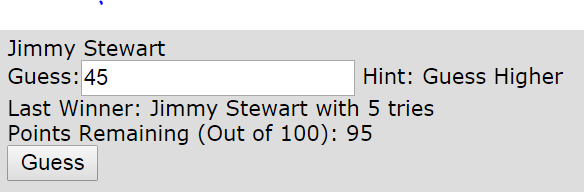
## 3.4.5. Feedback Messages:

Correct:



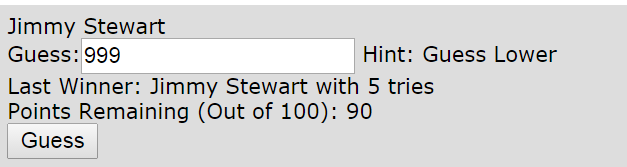
Jimmy Stewart correctly guessed the number which was 985 and was greeted with the feedback message that the guess given was correct.

Higher:



Jimmy Stewart made a guess that was lower than the number to be guessed and was greeted with the feedback message that he needs to guess higher than 45.

Lower:



Jimmy Stewart made a guess that was higher than the number to be guessed and was greeted with the feedback message that he needs to guess lower than 999.