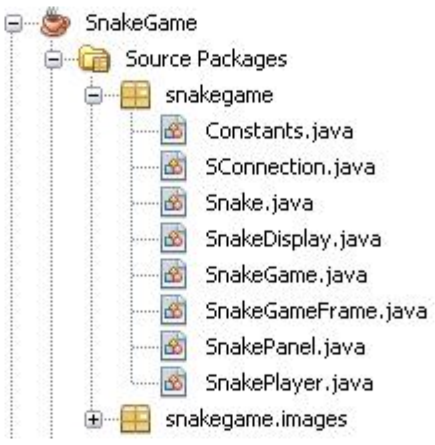
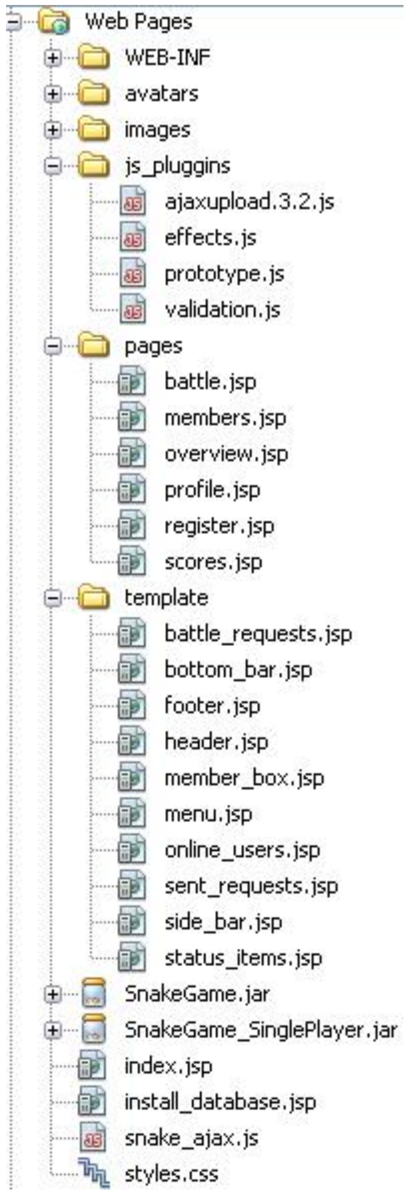
Created By
Lance Baker 2009

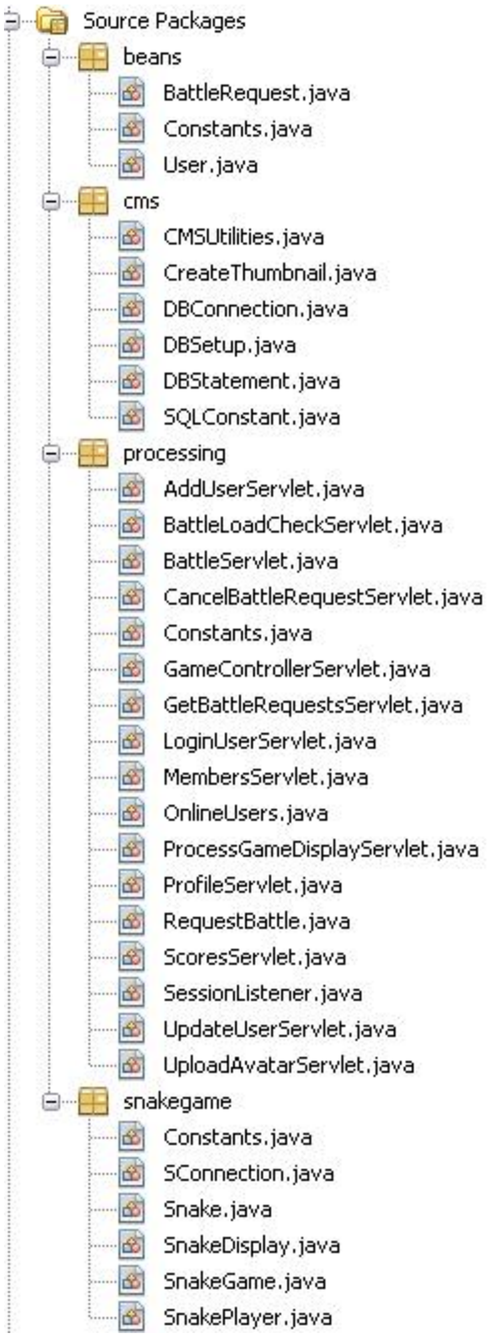
Project Content Overview



```
SnakeGame
    Source Packages
        snakegame
            Constants.java
            SConnection.java
            Snake.java
            SnakeDisplay.java
            SnakeGame.java
            SnakeGameFrame.java
            SnakePanel.java
            SnakePlayer.java
        snakegame.images
```



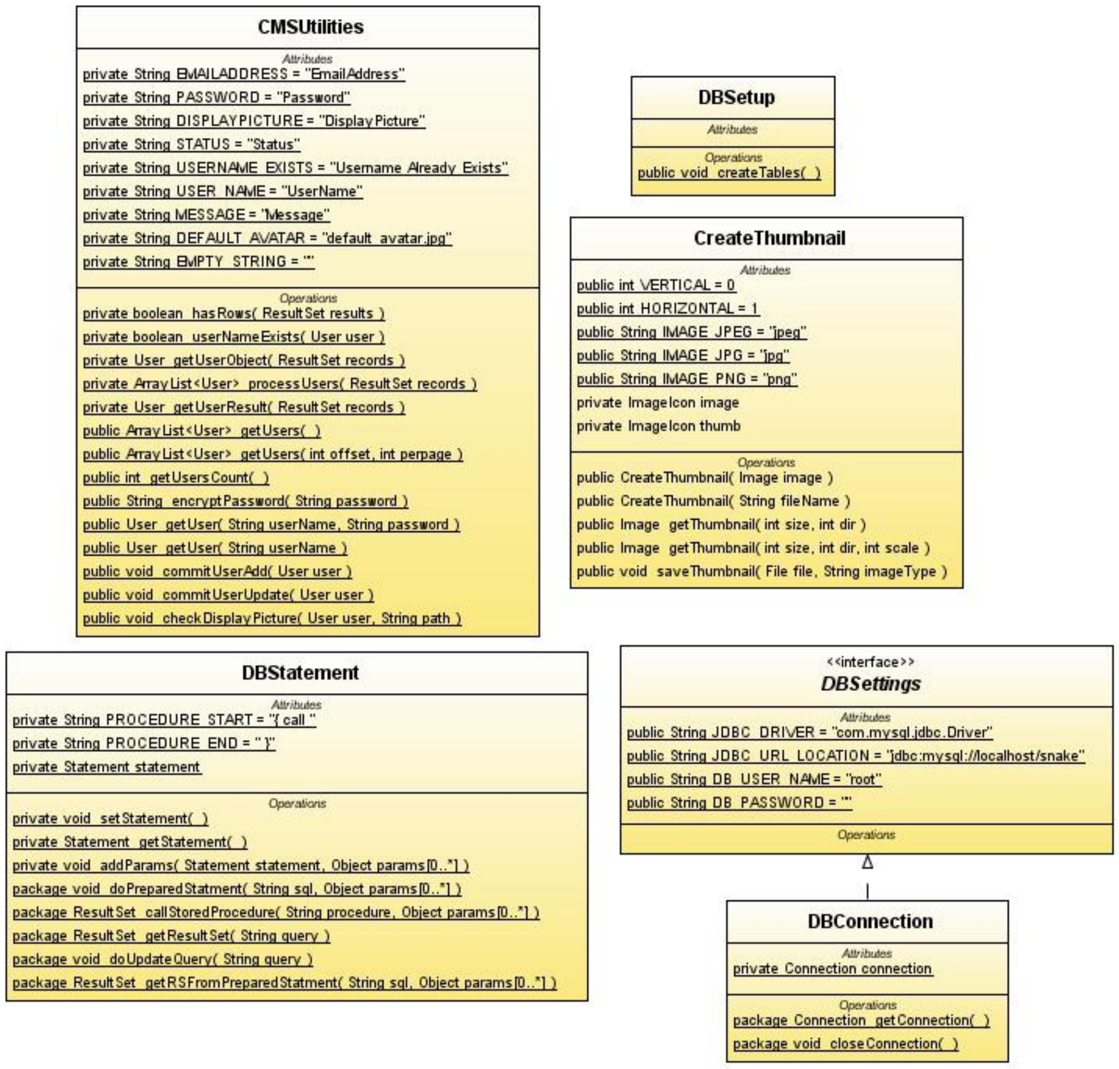
```
Web Pages
    WEB-INF
    avatars
    images
    js_pluggins
        ajaxupload.3.2.js
        effects.js
        prototype.js
        validation.js
    pages
        battle.jsp
        members.jsp
        overview.jsp
        profile.jsp
        register.jsp
        scores.jsp
    template
        battle_requests.jsp
        bottom_bar.jsp
        footer.jsp
        header.jsp
        member_box.jsp
        menu.jsp
        online_users.jsp
        sent_requests.jsp
        side_bar.jsp
        status_items.jsp
    SnakeGame.jar
    SnakeGame_SinglePlayer.jar
    index.jsp
    install_database.jsp
    snake_ajax.js
    styles.css
```



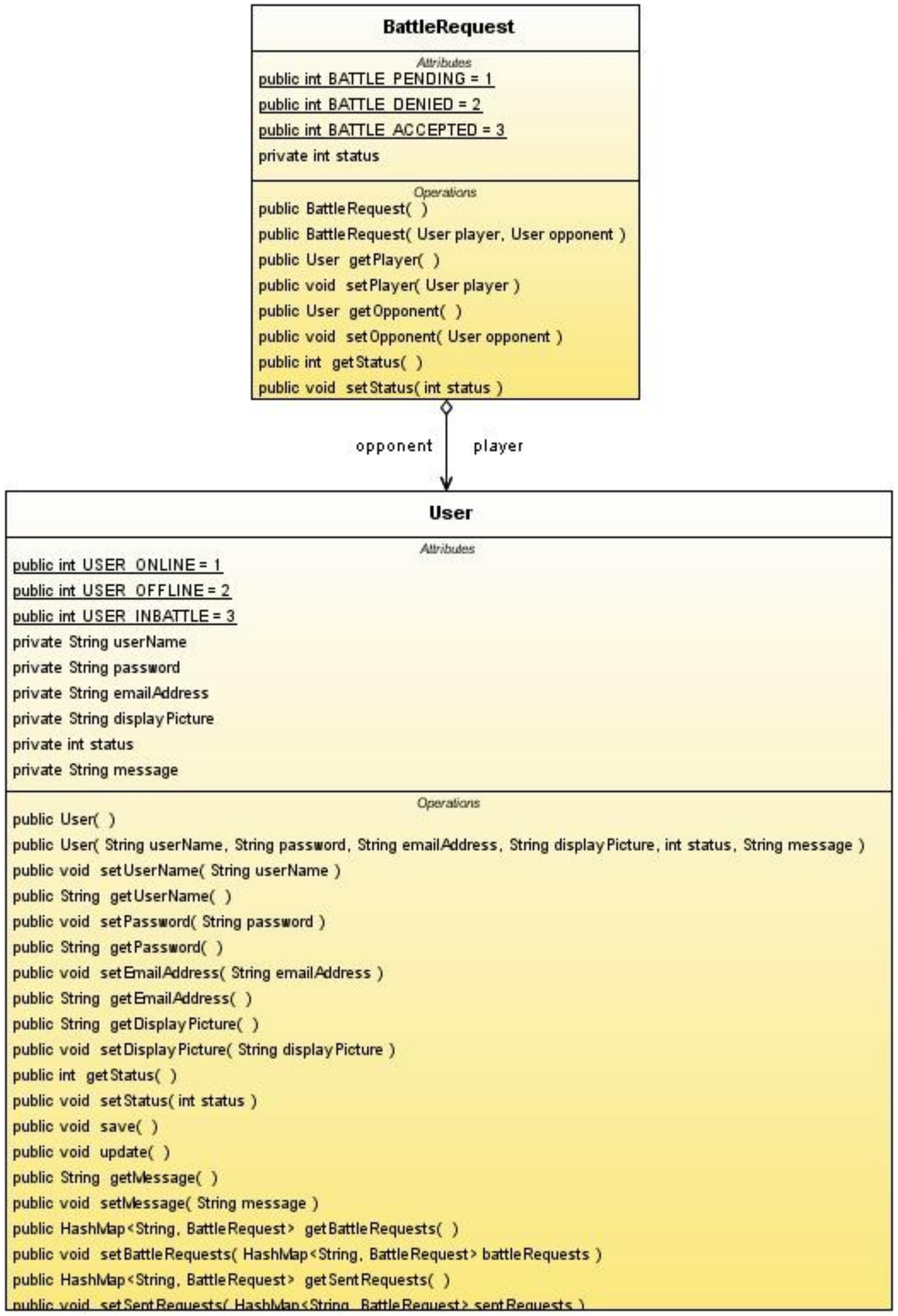
```
Source Packages
    beans
        BattleRequest.java
        Constants.java
        User.java
    cms
        CMSUtilities.java
        CreateThumbnail.java
        DBConnection.java
        DBSetup.java
        DBStatement.java
        SQLConstant.java
    processing
        AddUserServlet.java
        BattleLoadCheckServlet.java
        BattleServlet.java
        CancelBattleRequestServlet.java
        Constants.java
        GameControllerServlet.java
        GetBattleRequestsServlet.java
        LoginUserServlet.java
        MembersServlet.java
        OnlineUsers.java
        ProcessGameDisplayServlet.java
        ProfileServlet.java
        RequestBattle.java
        ScoresServlet.java
        SessionListener.java
        UpdateUserServlet.java
        UploadAvatarServlet.java
    snakegame
        Constants.java
        SConnection.java
        Snake.java
        SnakeDisplay.java
        SnakeGame.java
        SnakePlayer.java
```

Package Diagrams

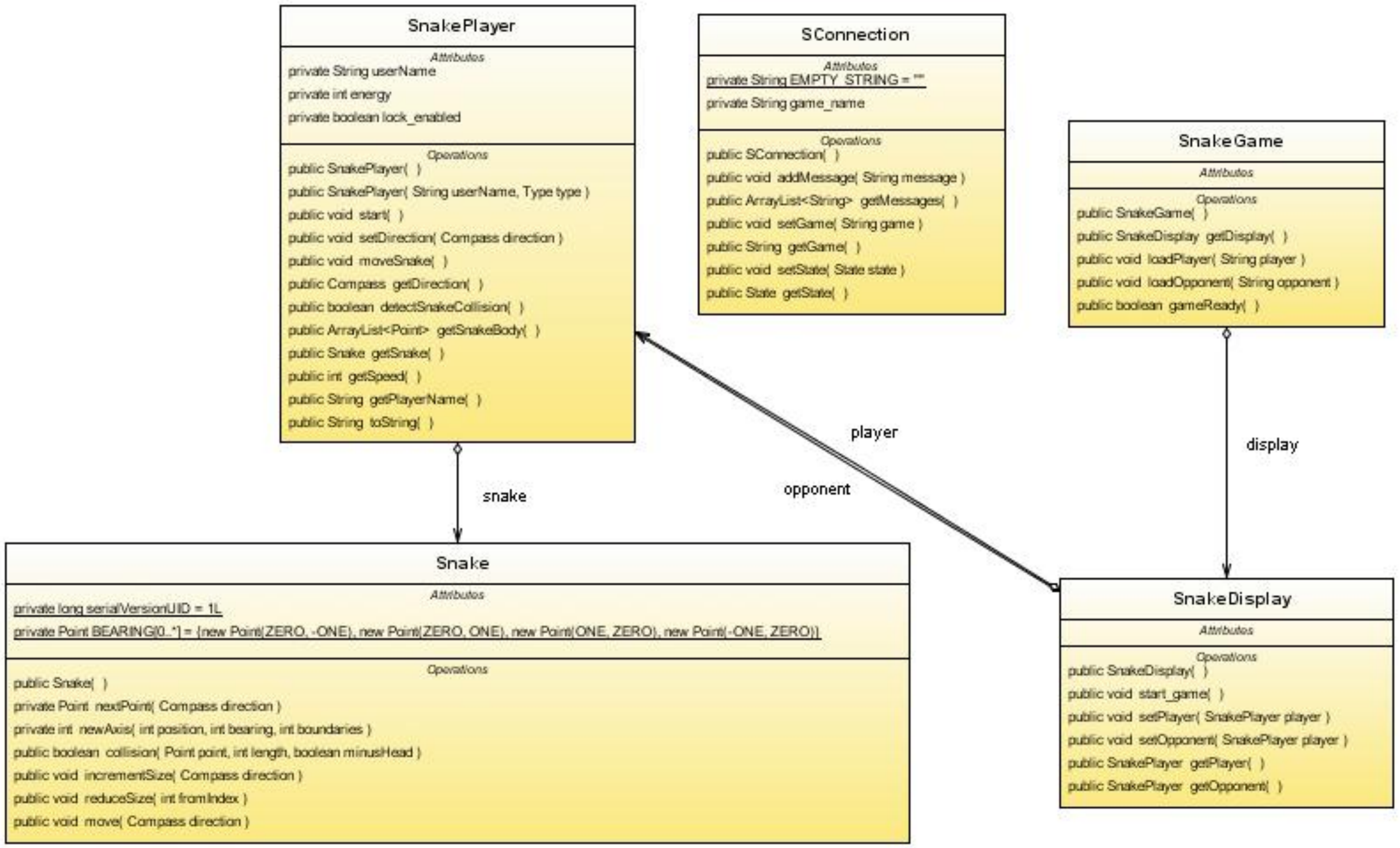
<u>CMS PACKAGE</u>


CMSUtilities
Attributes
private String EMAILADDRESS = "EmailAddress"
private String PASSWORD = "Password"
private String DISPLAYPICTURE = "DisplayPicture"
private String STATUS = "Status"
private String USERNAME_EXISTS = "Username Already Exists"
private String USER_NAME = "UserName"
private String MESSAGE = "Message"
private String DEFAULT_AVATAR = "default_avatar.jpg"
private String EMPTY_STRING = ""
Operations
private boolean hasRows(ResultSet results)
private boolean userNameExists(User user)
private User getUserObject(ResultSet records)
private ArrayList<User> processUsers(ResultSet records)
private User getUserResult(ResultSet records)
public ArrayList<User> getUsers( )
public ArrayList<User> getUsers(int offset, int perpage)
public int getUsersCount( )
public String encryptPassword(String password)
public User getUser(String userName, String password)
public User getUser(String userName)
public void commitUserAdd(User user)
public void commitUserUpdate(User user)
public void checkDisplayPicture(User user, String path)
DBSetup
Attributes
Operations
public void createTables( )
CreateThumbnail
Attributes
public int VERTICAL = 0
public int HORIZONTAL = 1
public String IMAGE_JPEG = "jpeg"
public String IMAGE_JPG = "jpg"
public String IMAGE_PNG = "png"
private ImageIcon image
private ImageIcon thumb
Operations
public CreateThumbnail(Image image)
public CreateThumbnail(String fileName)
public Image getThumbnail(int size, int dir)
public Image getThumbnail(int size, int dir, int scale)
public void saveThumbnail(File file, String imageType)
DBStatement
Attributes
private String PROCEDURE_START = "{ call "
private String PROCEDURE_END = " }"
private Statement statement
Operations
private void setStatement( )
private Statement getStatement( )
private void addParams(Statement statement, Object params[0..*])
package void doPreparedStatment(String sql, Object params[0..*])
package ResultSet callStoredProcedure(String procedure, Object params[0..*])
package ResultSet getResultSet(String query)
package void doUpdateQuery(String query)
package ResultSet getRSFromPreparedStatment(String sql, Object params[0..*])
<<interface>>
DBSettings
Attributes
public String JDBC_DRIVER = "com.mysql.jdbc.Driver"
public String JDBC_URL_LOCATION = "jdbc:mysql://localhost/snake"
public String DB_USER_NAME = "root"
public String DB_PASSWORD = ""
Operations
DBConnection
Attributes
private Connection connection
Operations
package Connection getConnection( )
package void closeConnection( )

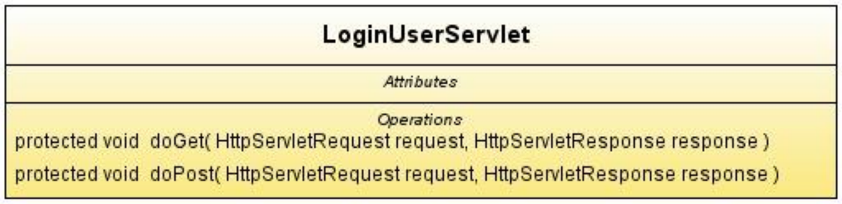
BEANS PACKAGE


BattleRequest
Attributes
public int BATTLE_PENDING = 1
public int BATTLE_DENIED = 2
public int BATTLE_ACCEPTED = 3
private int status
Operations
public BattleRequest( )
public BattleRequest(User player, User opponent)
public User getPlayer( )
public void setPlayer(User player)
public User getOpponent( )
public void setOpponent(User opponent)
public int getStatus( )
public void setStatus(int status)
opponent
player
User
Attributes
public int USER_ONLINE = 1
public int USER_OFFLINE = 2
public int USER_INBATTLE = 3
private String userName
private String password
private String emailAddress
private String displayPicture
private int status
private String message
Operations
public User( )
public User(String userName, String password, String emailAddress, String displayPicture, int status, String message)
public void setUserName(String userName)
public String getUserName( )
public void setPassword(String password)
public String getPassword( )
public void setEmailAddress(String emailAddress)
public String getEmailAddress( )
public String getDisplayPicture( )
public void setDisplayPicture(String displayPicture)
public int getStatus( )
public void setStatus(int status)
public void save( )
public void update( )
public String getMessage( )
public void setMessage(String message)
public HashMap<String, BattleRequest> getBattleRequests( )
public void setBattleRequests(HashMap<String, BattleRequest> battleRequests)
public HashMap<String, BattleRequest> getSentRequests( )
public void setSentRequests(HashMap<String, BattleRequest> sentRequests)

SNAKE SERVER PACKAGE


SnakePlayer
Attributes
private String userName
private int energy
private boolean lock_enabled
Operations
public SnakePlayer( )
public SnakePlayer(String userName, Type type)
public void start( )
public void setDirection(Compass direction)
public void moveSnake( )
public Compass getDirection( )
public boolean detectSnakeCollision( )
public ArrayList<Point> getSnakeBody( )
public Snake getSnake( )
public int getSpeed( )
public String getPlayerName( )
public String toString( )
SConnection
Attributes
private String EMPTY_STRING = ""
private String game_name
Operations
public SConnection( )
public void addMessage(String message)
public ArrayList<String> getMessages( )
public void setGame(String game)
public String getGame( )
public void setState(State state)
public State getState( )
SnakeGame
Attributes
Operations
public SnakeGame( )
public SnakeDisplay getDisplay( )
public void loadPlayer(String player)
public void loadOpponent(String opponent)
public boolean gameReady( )
player
opponent
display
snake
Snake
Attributes
private long serialVersionUID = 1L
private Point BEARING[0..*] = {new Point(ZERO, -ONE), new Point(ZERO, ONE), new Point(ONE, ZERO), new Point(-ONE, ZERO)}
Operations
public Snake( )
private Point nextPoint(Compass direction)
private int newAxis(int position, int bearing, int boundaries)
public boolean collision(Point point, int length, boolean minusHead)
public void incrementSize(Compass direction)
public void reduceSize(int fromIndex)
public void move(Compass direction)
SnakeDisplay
Attributes
Operations
public SnakeDisplay( )
public void start_game( )
public void setPlayer(SnakePlayer player)
public void setOpponent(SnakePlayer player)
public SnakePlayer getPlayer( )
public SnakePlayer getOpponent( )

PROCESSING PACKAGE (USER INVOLVED SERVLETS)

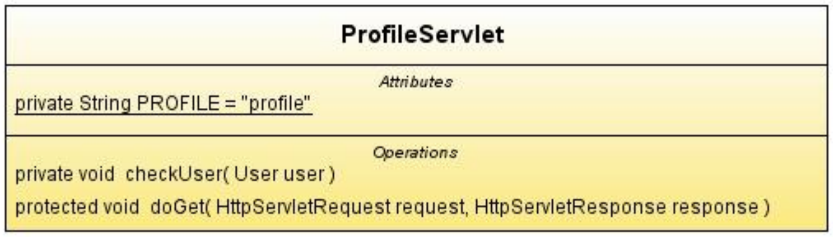


LoginUserServlet

Attributes

Operations

```
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
```



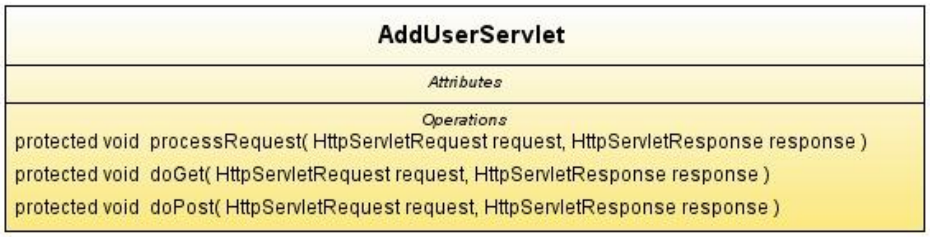
ProfileServlet

Attributes

```
private String PROFILE = "profile"
```

Operations

```
private void  checkUser( User user )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
```

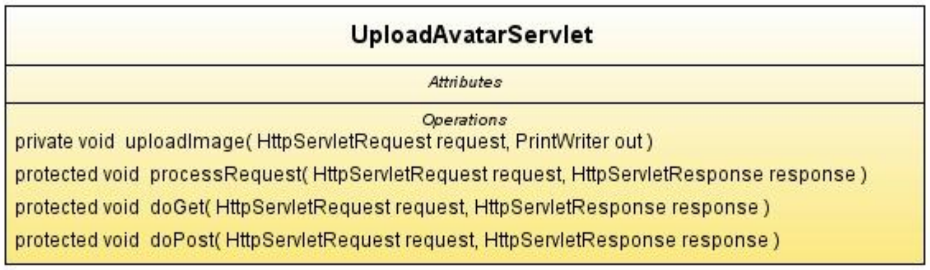


AddUserServlet

Attributes

Operations

```
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
```

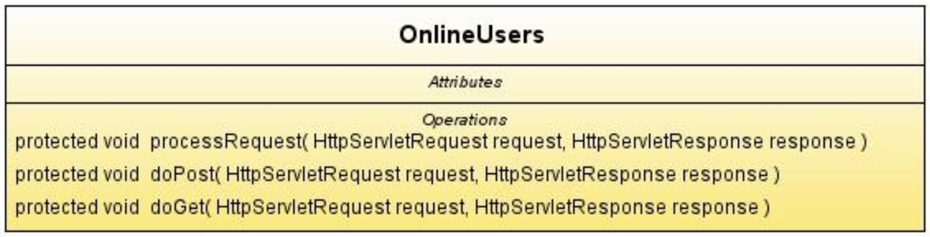


UploadAvatarServlet

Attributes

Operations

```
private void  uploadImage( HttpServletRequest request, PrintWriter out )
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
```

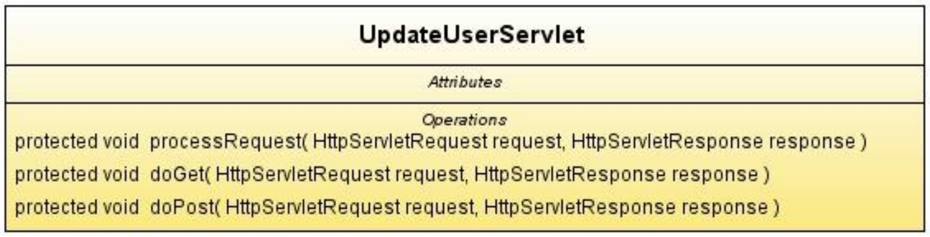


OnlineUsers

Attributes

Operations

```
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
```

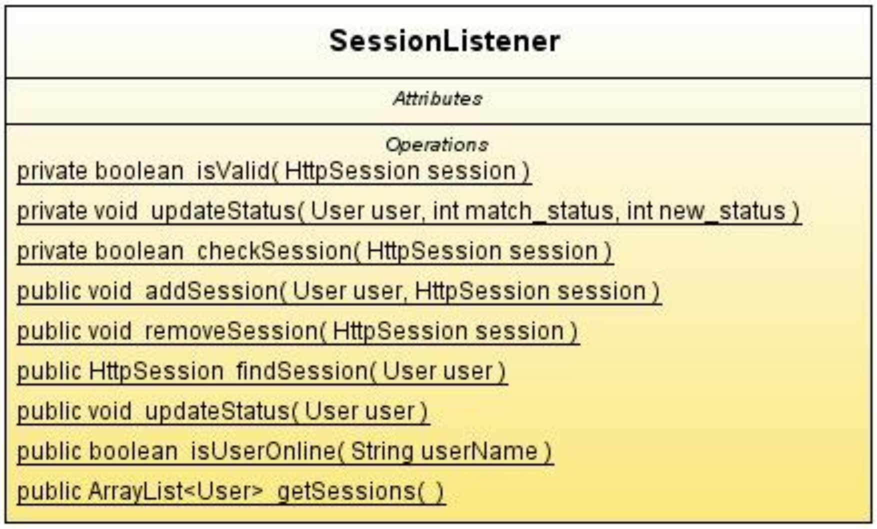


UpdateUserServlet

Attributes

Operations

```
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
```

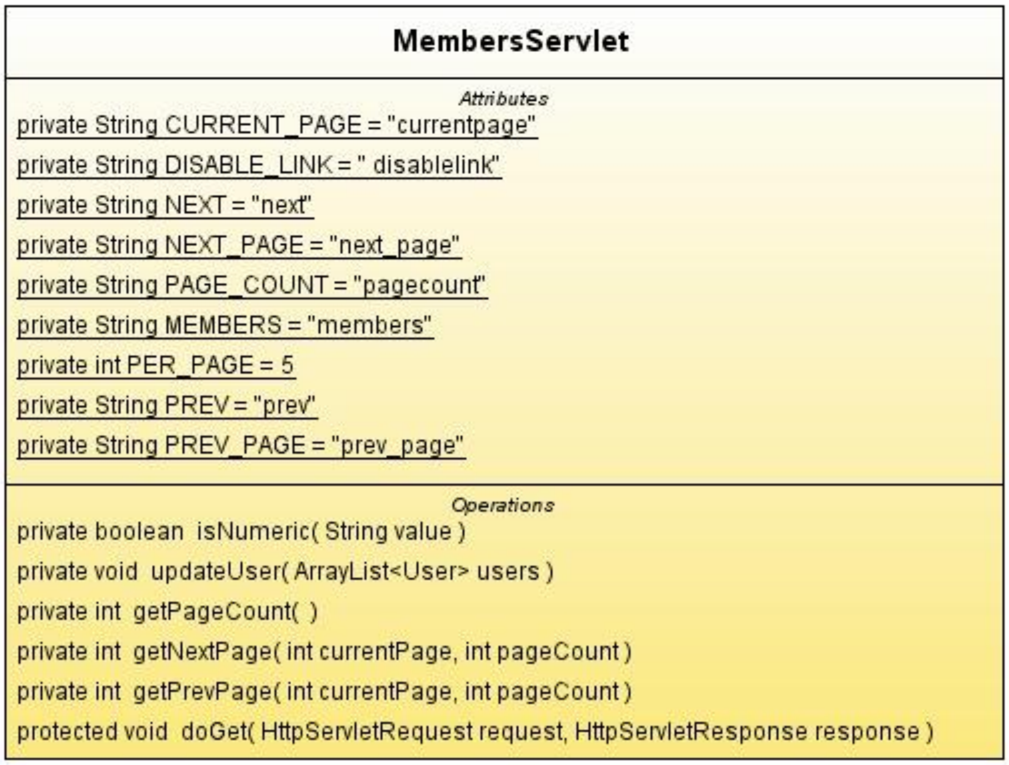


SessionListener

Attributes

Operations

```
private boolean  isValid( HttpSession session )
private void  updateStatus( User user, int match_status, int new_status )
private boolean  checkSession( HttpSession session )
public void  addSession( User user, HttpSession session )
public void  removeSession( HttpSession session )
public HttpSession  findSession( User user )
public void  updateStatus( User user )
public boolean  isUserOnline( String userName )
public ArrayList<User>  getSessions( )
```



MembersServlet

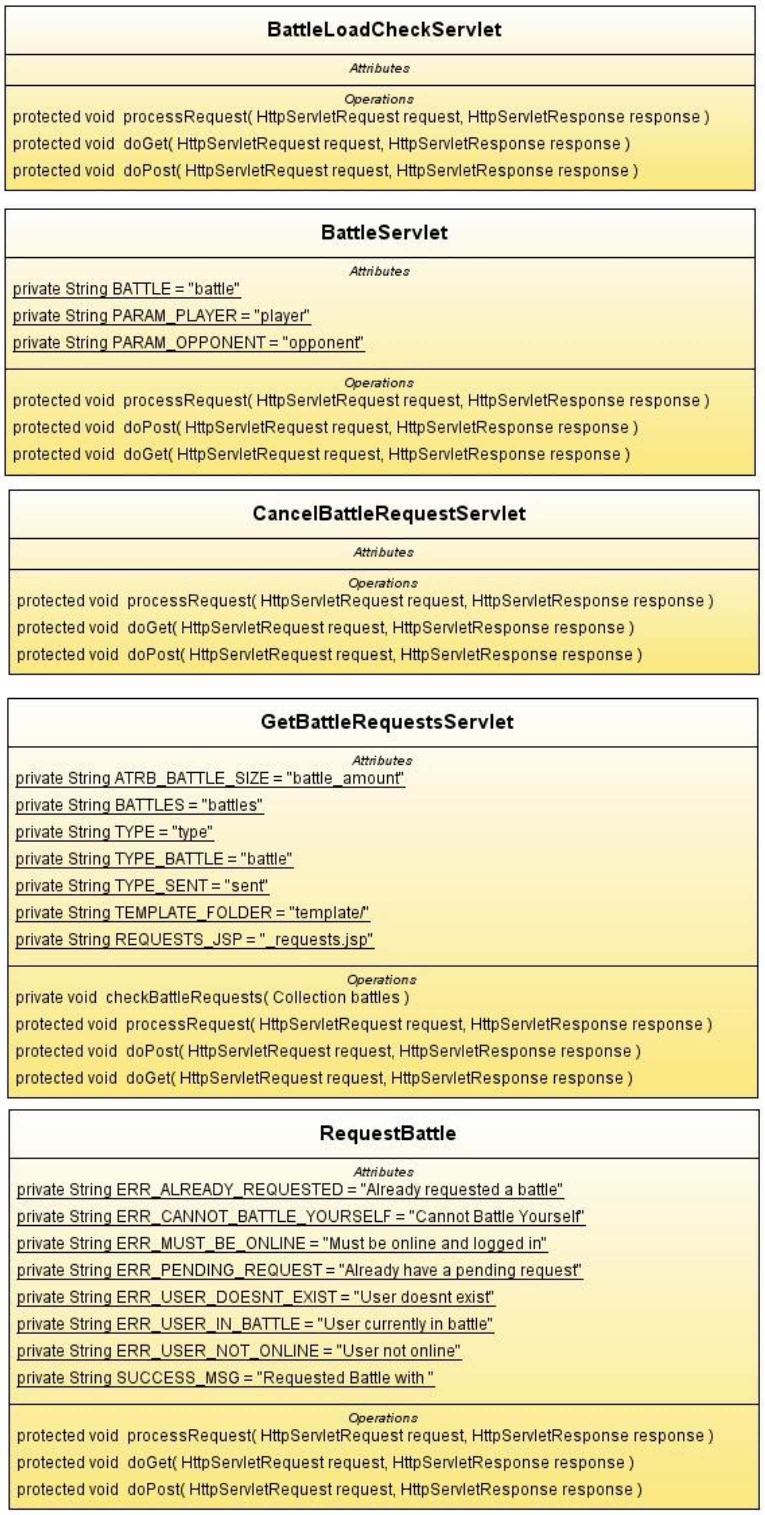
Attributes

```
private String CURRENT_PAGE = "currentpage"
private String DISABLE_LINK = " disablelink"
private String NEXT = "next"
private String NEXT_PAGE = "next_page"
private String PAGE_COUNT = "pagecount"
private String MEMBERS = "members"
private int PER_PAGE = 5
private String PREV = "prev"
private String PREV_PAGE = "prev_page"
```

Operations

```
private boolean  isNumeric( String value )
private void  updateUser( ArrayList<User> users )
private int  getPageCount( )
private int  getNextPage( int currentPage, int pageCount )
private int  getPrevPage( int currentPage, int pageCount )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
```

PROCESSING PACKAGE (BATTLE REQUESTING)



BattleLoadCheckServlet

Attributes

Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)

BattleServlet

Attributes
private String BATTLE = "battle"
private String PARAM_PLAYER = "player"
private String PARAM_OPPONENT = "opponent"

Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)

CancelBattleRequestServlet

Attributes

Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)

GetBattleRequestsServlet

Attributes
private String ATRB_BATTLE_SIZE = "battle_amount"
private String BATTLES = "battles"
private String TYPE = "type"
private String TYPE_BATTLE = "battle"
private String TYPE_SENT = "sent"
private String TEMPLATE_FOLDER = "template/"
private String REQUESTS_JSP = "_requests.jsp"

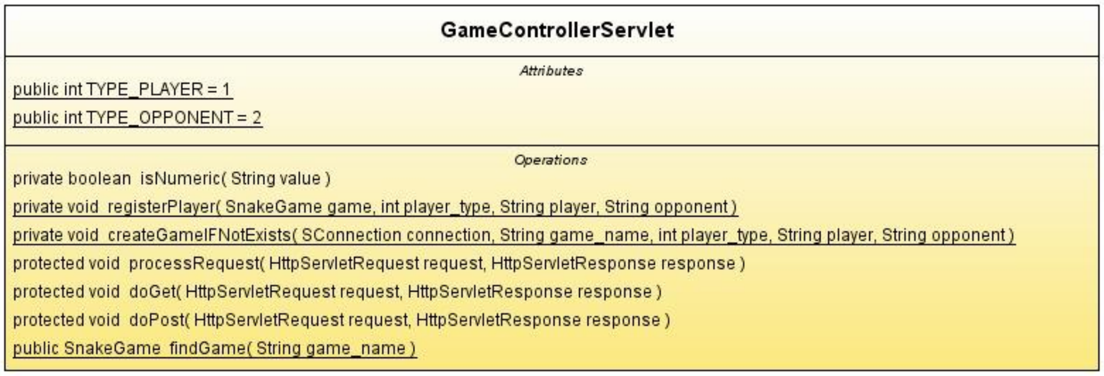
Operations
private void checkBattleRequests(Collection battles)
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)

RequestBattle

Attributes
private String ERR_ALREADY_REQUESTED = "Already requested a battle"
private String ERR_CANNOT_BATTLE_YOURSELF = "Cannot Battle Yourself"
private String ERR_MUST_BE_ONLINE = "Must be online and logged in"
private String ERR_PENDING_REQUEST = "Already have a pending request"
private String ERR_USER_DOESNT_EXIST = "User doesnt exist"
private String ERR_USER_IN_BATTLE = "User currently in battle"
private String ERR_USER_NOT_ONLINE = "User not online"
private String SUCCESS_MSG = "Requested Battle with "

Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)

PROCESSING PACKAGE (GAME COMMUNICATION)



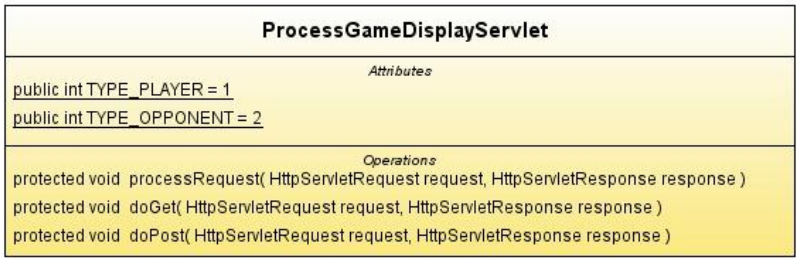
GameControllerServlet

Attributes

public int TYPE_PLAYER = 1
public int TYPE_OPPONENT = 2

Operations

private boolean isNumeric(String value)
private void registerPlayer(SnakeGame game, int player_type, String player, String opponent)
private void createGameIFNotExists(SConnection connection, String game_name, int player_type, String player, String opponent)
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public SnakeGame findGame(String game_name)



ProcessGameDisplayServlet

Attributes

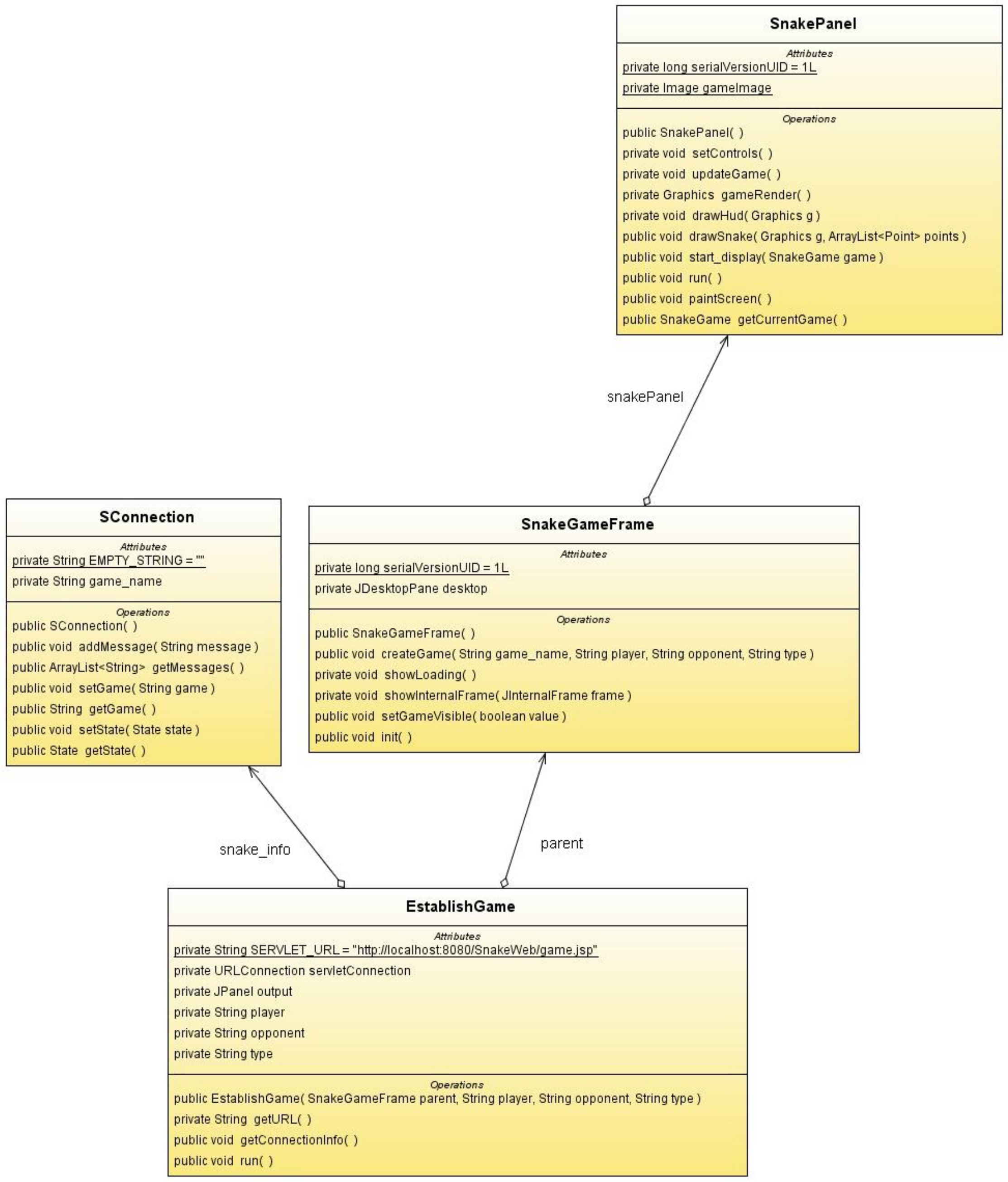
public int TYPE_PLAYER = 1
public int TYPE_OPPONENT = 2

Operations

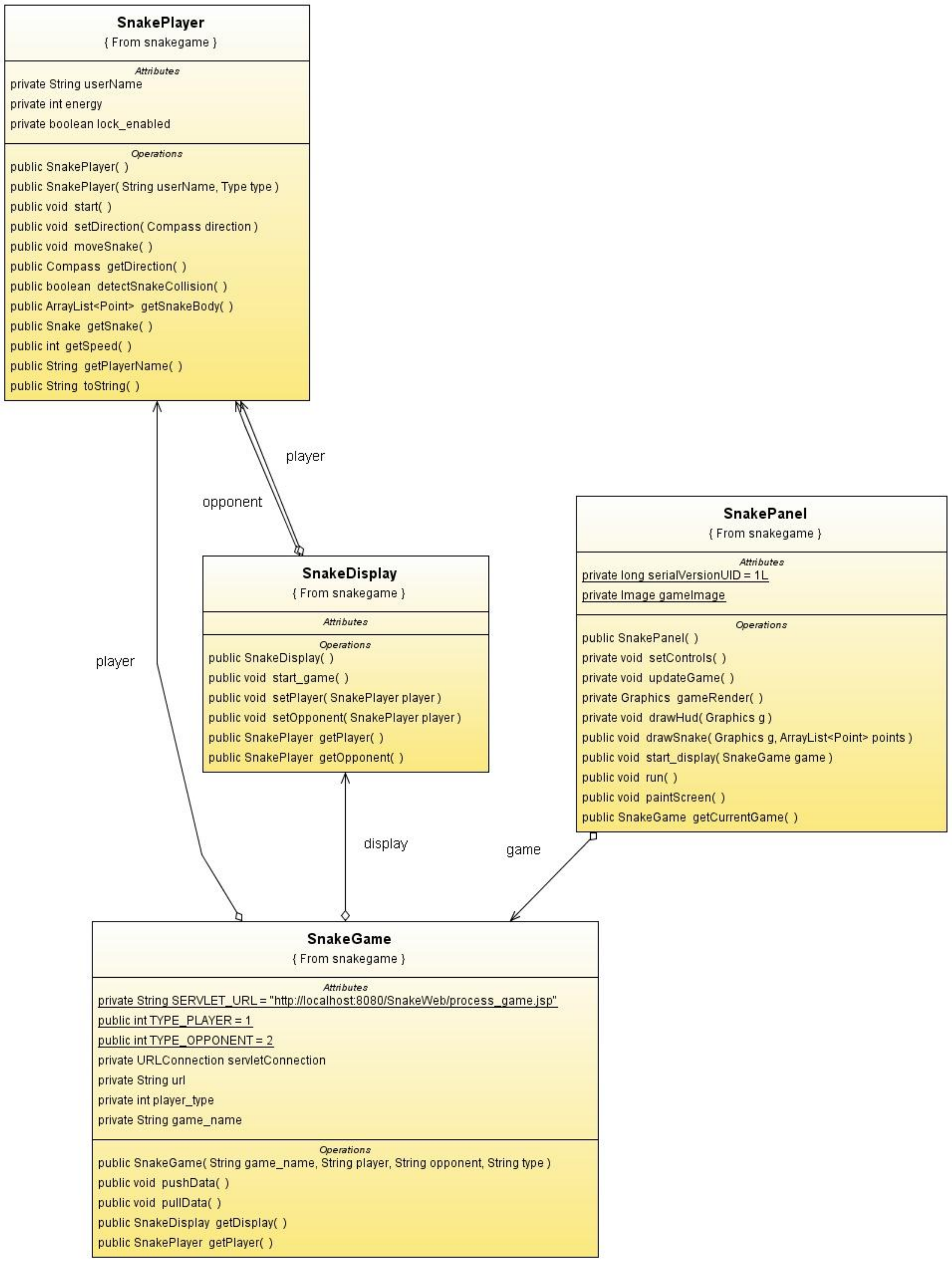
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)

Snake Applet

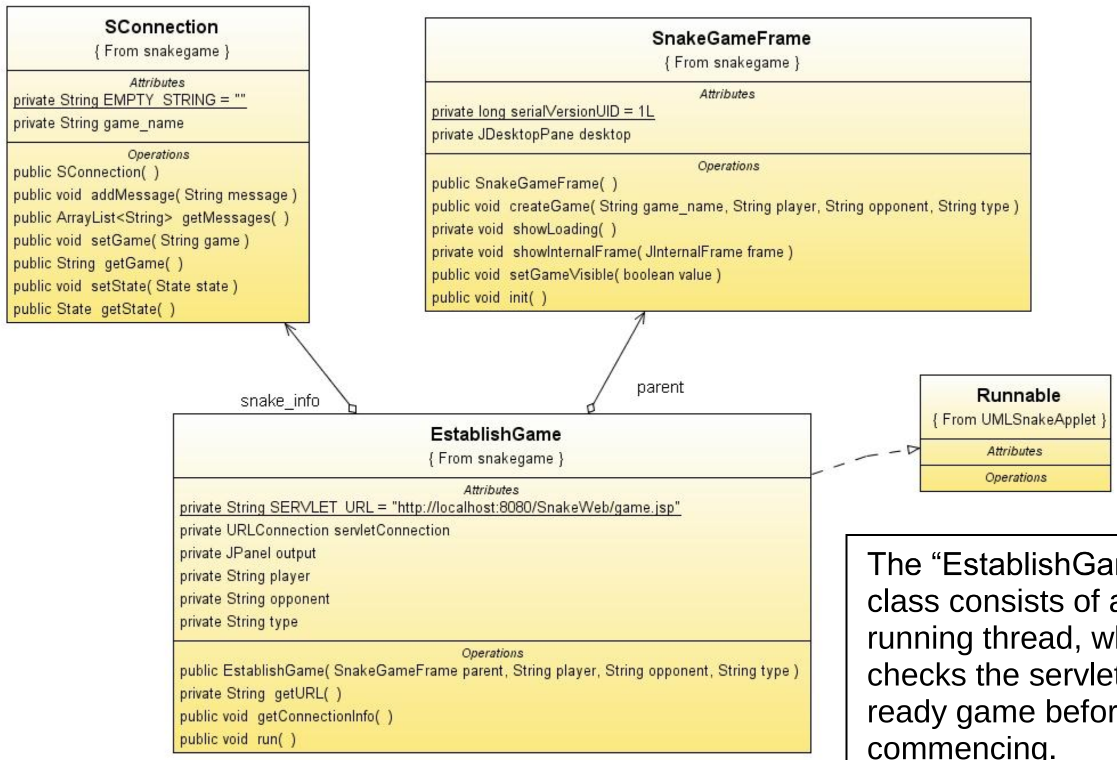
<u>Domain Class Diagram (Part 1)</u>


SnakePanel
Attributes
private long serialVersionUID = 1L
private Image gameImage
Operations
public SnakePanel( )
private void setControls( )
private void updateGame( )
private Graphics gameRender( )
private void drawHud(Graphics g)
public void drawSnake(Graphics g, ArrayList<Point> points)
public void start_display(SnakeGame game)
public void run( )
public void paintScreen( )
public SnakeGame getCurrentGame( )
snakePanel
SConnection
Attributes
private String EMPTY_STRING = ""
private String game_name
Operations
public SConnection( )
public void addMessage(String message)
public ArrayList<String> getMessages( )
public void setGame(String game)
public String getGame( )
public void setState(State state)
public State getState( )
SnakeGameFrame
Attributes
private long serialVersionUID = 1L
private JDesktopPane desktop
Operations
public SnakeGameFrame( )
public void createGame(String game_name, String player, String opponent, String type)
private void showLoading( )
private void showInternalFrame(JInternalFrame frame)
public void setGameVisible(boolean value)
public void init( )
snake_info
parent
EstablishGame
Attributes
private String SERVLET_URL = "http://localhost:8080/SnakeWeb/game.jsp"
private URLConnection servletConnection
private JPanel output
private String player
private String opponent
private String type
Operations
public EstablishGame(SnakeGameFrame parent, String player, String opponent, String type)
private String getURL( )
public void getConnectionInfo( )
public void run( )

Domain Class Diagram (Part 2)

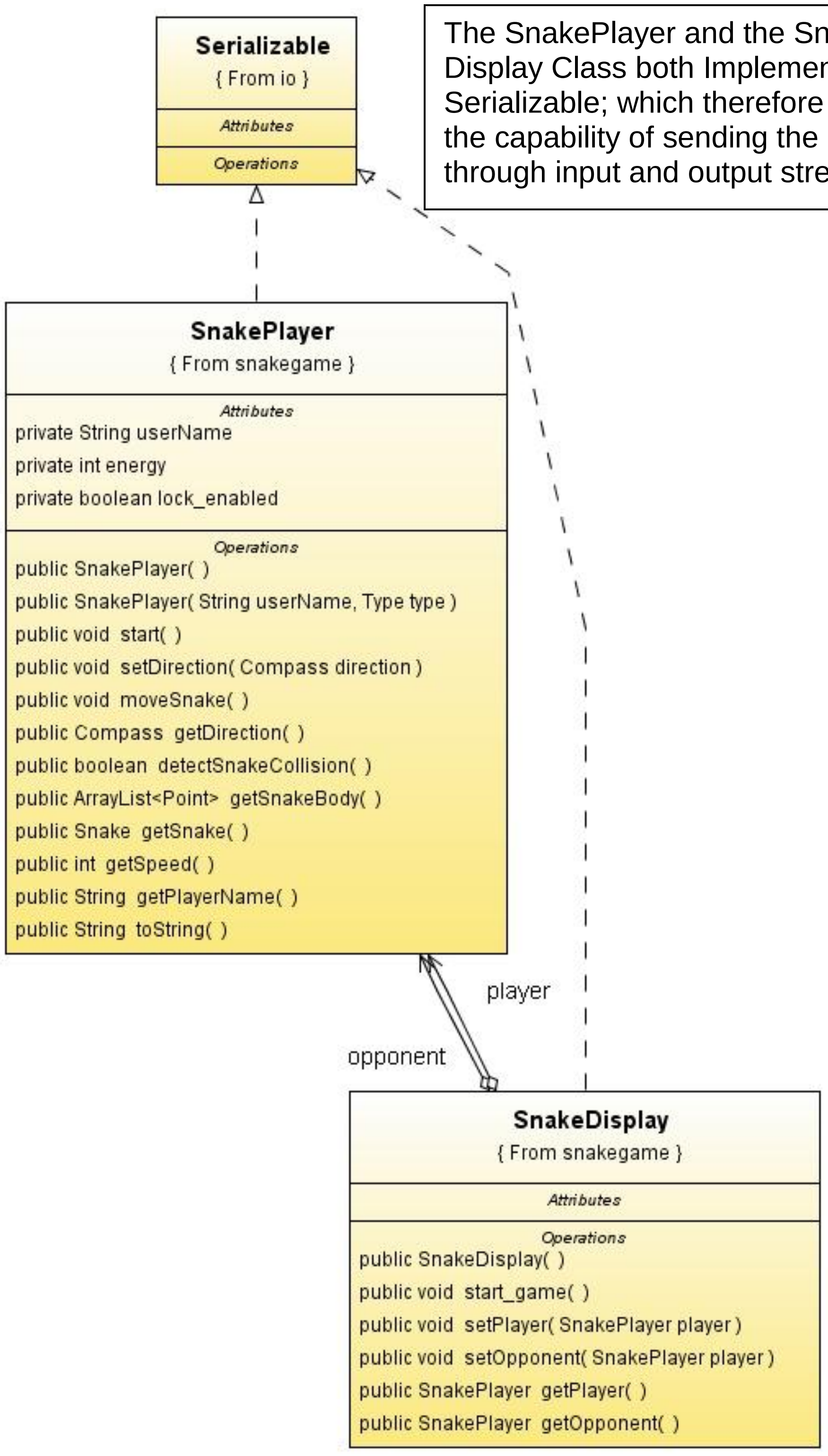

SnakePlayer
{ From snakegame }
Attributes
private String userName
private int energy
private boolean lock_enabled
Operations
public SnakePlayer( )
public SnakePlayer(String userName, Type type)
public void start( )
public void setDirection(Compass direction)
public void moveSnake( )
public Compass getDirection( )
public boolean detectSnakeCollision( )
public ArrayList<Point> getSnakeBody( )
public Snake getSnake( )
public int getSpeed( )
public String getPlayerName( )
public String toString( )
player
opponent
player
SnakeDisplay
{ From snakegame }
Attributes
Operations
public SnakeDisplay( )
public void start_game( )
public void setPlayer(SnakePlayer player)
public void setOpponent(SnakePlayer player)
public SnakePlayer getPlayer( )
public SnakePlayer getOpponent( )
SnakePanel
{ From snakegame }
Attributes
private long serialVersionUID = 1L
private Image gameImage
Operations
public SnakePanel( )
private void setControls( )
private void updateGame( )
private Graphics gameRender( )
private void drawHud(Graphics g)
public void drawSnake(Graphics g, ArrayList<Point> points)
public void start_display(SnakeGame game)
public void run( )
public void paintScreen( )
public SnakeGame getCurrentGame( )
display
game
SnakeGame
{ From snakegame }
Attributes
private String SERVLET_URL = "http://localhost:8080/SnakeWeb/process_game.jsp"
public int TYPE_PLAYER = 1
public int TYPE_OPPONENT = 2
private URLConnection servletConnection
private String url
private int player_type
private String game_name
Operations
public SnakeGame(String game_name, String player, String opponent, String type)
public void pushData( )
public void pullData( )
public SnakeDisplay getDisplay( )
public SnakePlayer getPlayer( )

Establish Game Dependency Diagram


SConnection
{ From snakegame }
Attributes
private String EMPTY_STRING = ""
private String game_name
Operations
public SConnection( )
public void addMessage(String message)
public ArrayList<String> getMessages( )
public void setGame(String game)
public String getGame( )
public void setState(State state)
public State getState( )
SnakeGameFrame
{ From snakegame }
Attributes
private long serialVersionUID = 1L
private JDesktopPane desktop
Operations
public SnakeGameFrame( )
public void createGame(String game_name, String player, String opponent, String type)
private void showLoading( )
private void showInternalFrame(JInternalFrame frame)
public void setGameVisible(boolean value)
public void init( )
snake_info
parent
Runnable
{ From UMLSnakeApplet }
Attributes
Operations
EstablishGame
{ From snakegame }
Attributes
private String SERVLET_URL = "http://localhost:8080/SnakeWeb/game.jsp"
private URLConnection servletConnection
private JPanel output
private String player
private String opponent
private String type
Operations
public EstablishGame(SnakeGameFrame parent, String player, String opponent, String type)
private String getURL( )
public void getConnectionInfo( )
public void run( )

The "EstablishGame" class consists of an inner running thread, which checks the servlet for a ready game before commencing.

Snake Display Dependency Diagram


Serializable
{ From io }
Attributes
Operations
SnakePlayer
{ From snakegame }
Attributes
private String userName
private int energy
private boolean lock_enabled
Operations
public SnakePlayer( )
public SnakePlayer(String userName, Type type)
public void start( )
public void setDirection(Compass direction)
public void moveSnake( )
public Compass getDirection( )
public boolean detectSnakeCollision( )
public ArrayList<Point> getSnakeBody( )
public Snake getSnake( )
public int getSpeed( )
public String getPlayerName( )
public String toString( )
player
opponent
SnakeDisplay
{ From snakegame }
Attributes
Operations
public SnakeDisplay( )
public void start_game( )
public void setPlayer(SnakePlayer player)
public void setOpponent(SnakePlayer player)
public SnakePlayer getPlayer( )
public SnakePlayer getOpponent( )

The SnakePlayer and the Snake Display Class both Implement Serializable; which therefore enables the capability of sending the objects through input and output streams.

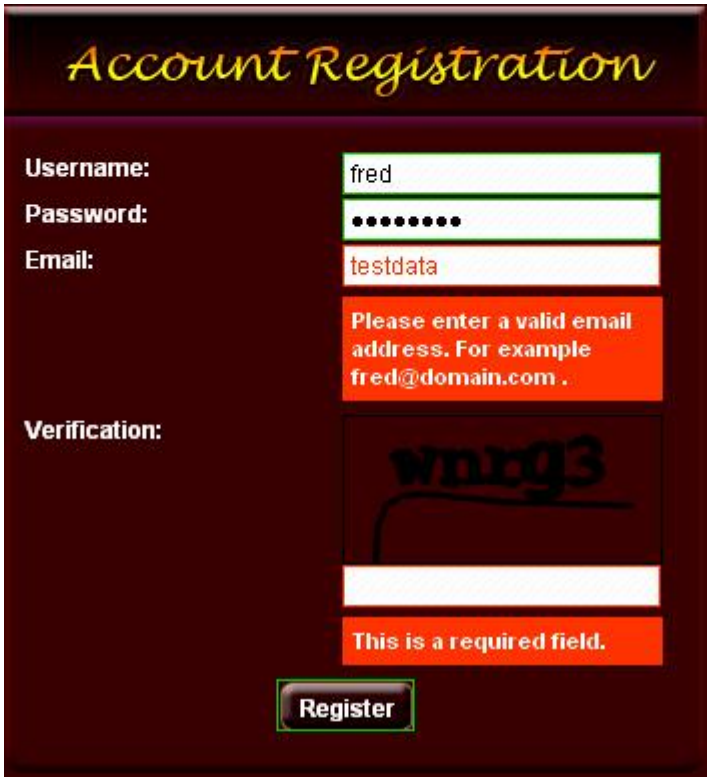
User Interactivity

The multiplayer snake game website supports the ability for users to dynamically interact using implement Ajax communication technology for sending and receiving requests from the servlets. The technology was implemented to increase the interactivity levels of the website, and to allow users to see others with an online status instantaneously, receiving battle requests, and also to dynamically load the 'player' into the game once a battle has been accepted.

<u>Registration</u>

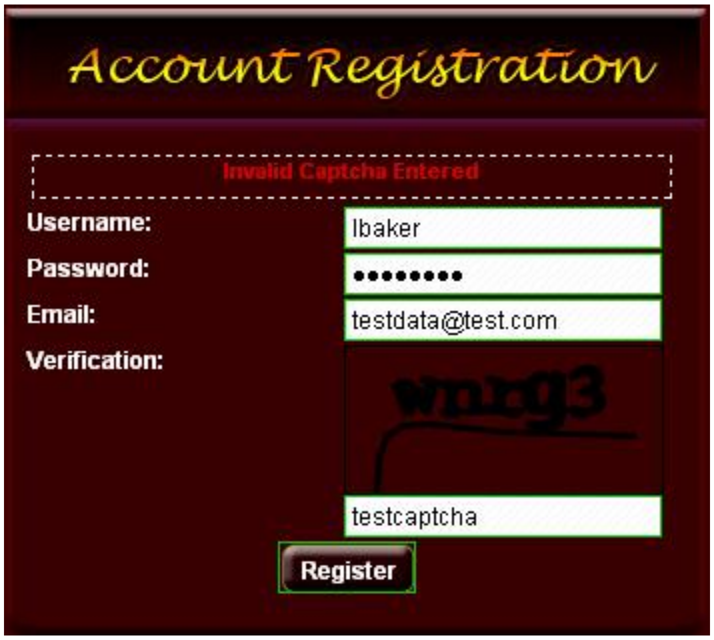
The website contains a registration page, which is severely guarded by JavaScript validation and also a captcha code; used to reduce spam submissions, and to verify that the user is indeed human.

a. The JavaScript submits the form to the servlets for processing, so therefore it is a requirement for the user to have JavaScript enabled; which is deliberately done to prevent the form submitting without JavaScript support.
b. The registration request is processed and sent to the servlet through an Ajax transmission; the reply is then received, and sent back to the user's browser and outputted to the corresponding div.
c. The user registration servlet validates whether the username specified is unique within the system; preventing users from creating an account with the same user name.


Account Registration

Username:	fred
Password:	●●●●●●●●●
Email:	testdata
	Please enter a valid email address. For example fred@domain.com .
Verification:	
	This is a required field.

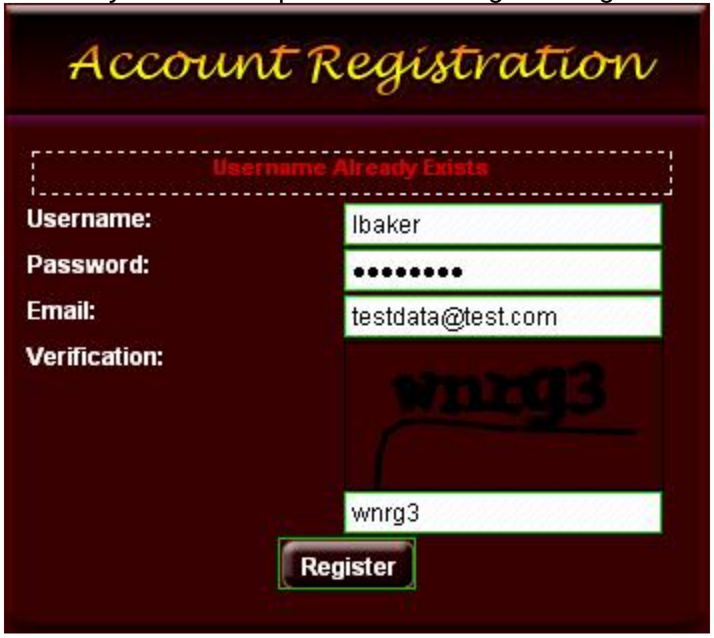
d. The user has entered a invalid captcha code, the JavaScript sends the request via Ajax to the servlet for processing, checks the contents of the captcha parameter and if incorrect; it outputs the following message.



Invalid Captcha Entered

Username:	lbaker
Password:	•••••••••
Email:	testdata@test.com
Verification:	testcaptcha

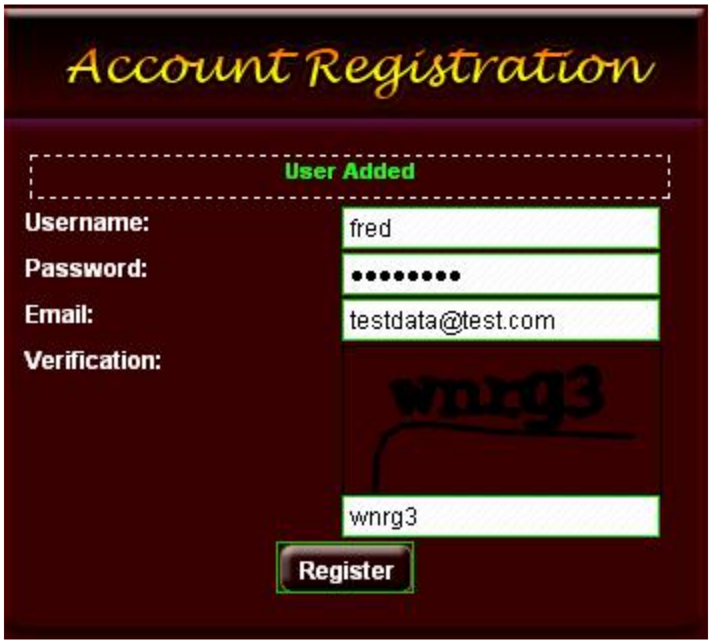
e. If the captcha is validated successfully, it then will proceed to validate the user name; by checking whether it already exists within the users table. If it already exists it outputs the following message.



Username Already Exists

Username:	lbaker
Password:	•••••••••
Email:	testdata@test.com
Verification:	wnrg3

f. If the data was sent, and validated successfully; it will add the new user as a new record within the users table, and output the following message for success verification. The user will now have the ability to log into the website.



<u>JavaScript Processing</u>

The website uses some ready made libraries for the input validation, and for the transmission of Ajax communication messages to the servlet.

```
function CheckForm(form) {
        var valid = new Validation(form, {onSubmit: false, useTitles: true});
        return valid.validate();
}

function RegisterForm() {
    if (CheckForm('registerForm')) {
        var params = "userName="+$('userName').value+"&password="+$('password').value+
            "&emailAddress="+$('emailAddress').value+"&answer="+$('verification').value;
        new Ajax.Updater("info_message", "AddUser.ajax", {
            method: 'post',
            parameters: params
        });
        $('info_message').style.display = '';
    }
}

Snake_ajax.js
```

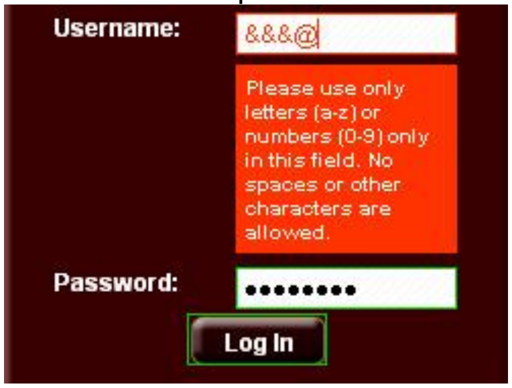
Servlet Processing
The servlet receives the sent request, and validates the parameters given.

```
        Captcha captcha = (Captcha)request.getSession().getAttribute(Captcha.NAME);

        request.setCharacterEncoding("UTF-8");
        String answer = request.getParameter("answer");
        try {
            if (!captcha.isCorrect(answer)) {
              throw new Exception(ERR_INVALID_CAPTCHA);
            }

            new User(request.getParameter(USER_NAME),
                    request.getParameter(PASSWORD),
                    request.getParameter(EMAIL_ADDRESS),
                    DEFAULT_AVATAR, User.USER_OFFLINE, DEFAULT_MESSAGE).save();


            out.println(SPAN_SUCCESS + MSG_USER_ADDED + END_SPAN);
        } catch (Exception ex) {
          out.println(SPAN_ERROR + ex.getMessage() + END_SPAN);
        }

    processing.AddUserServlet.java
```

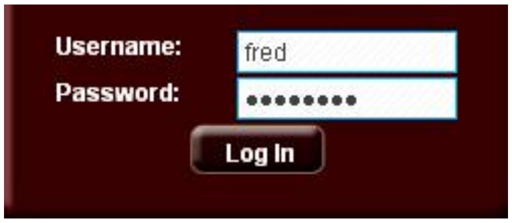
User Login

Once the user has been successfully created, the guest now has the ability to login from inputting their credentials into the login form located on the side bar.

g. The login form is validated through JavaScript, and is only submitted through JavaScript as well; so therefore if they have disabled support for JavaScript they will not be able to login, as it is a detrimental requirement for the website to function.



Please use only letters (a-z) or numbers (0-9) only in this field. No spaces or other characters are allowed.



h. Once the user has been successfully validated, a session will be created; which also stores the session into a SessionListener class – for later use.

<u>JavaScript Processing</u>

```
function LoginAction() {
    var form = $('UserLoginForm');
    if (CheckForm(form)) {
      form.submit();
    }
}


Event.observe('loginAction', 'click', function() {
    LoginAction();
});



Snake_ajax.js
```

<u>Servlet Processing</u>

The parameters are not checked for null values, due to the fact that the parameters have to be sent. The validation is kept to a minimum, as JavaScript does the hard yards already.

- The parameters received are checked using a method created for the retrieval of the User bean object based on the username. If the user is not found it will return a null value; which is checked before proceeding.
- The user can have display pictures uploaded, but due to netbeans deleting the build contents every time the application gets built (which is obvious I know) I figured it would be best to check whether the display picture actually exists.
- The session is now stored in a created session listener class; which will be explained on the next page. The class will be used for the "Wow" affect later on for sending battle and receiving battle requests.

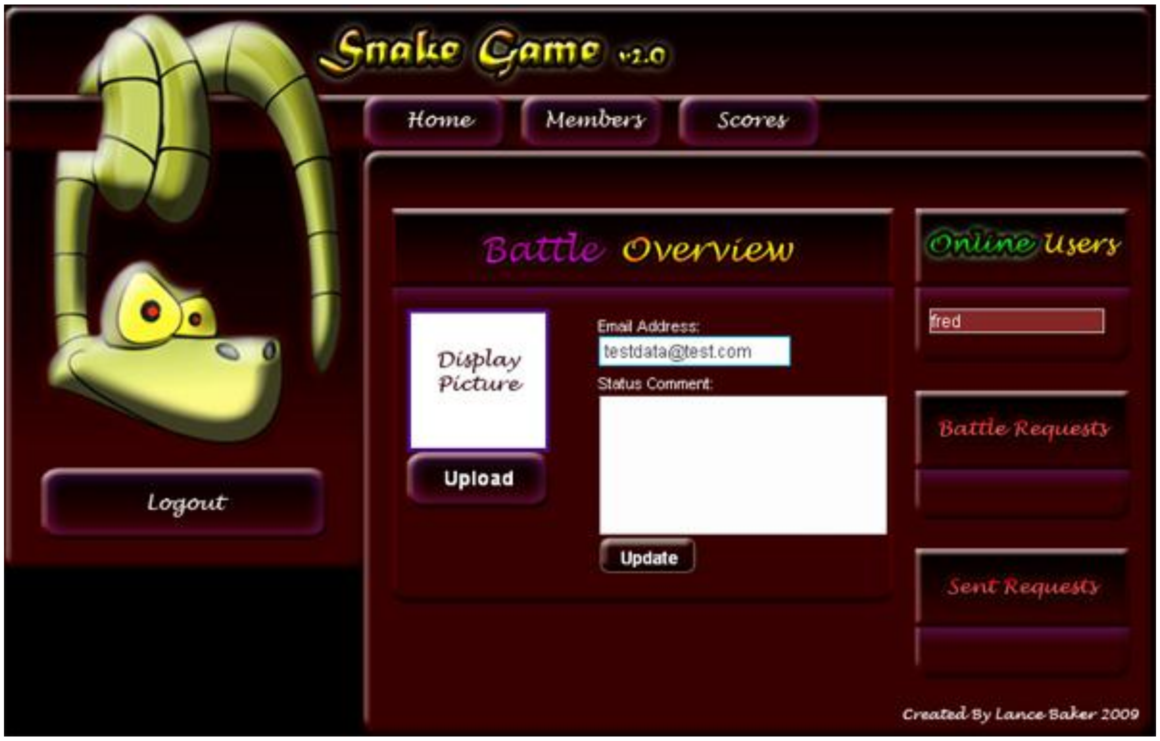
```
User user = CMSUtilities.getUser(request.getParameter(LOGIN_USER_NAME),
request.getParameter(LOGIN_PASSWORD));

if (user != null) {
            request.getSession().setAttribute(LOGGED_IN_USER, user);
            String path = this.getServletContext().getRealPath(USER_UPLOAD_PATH) + File.separator;
            cms.CMSUtilities.checkDisplayPicture(user, path);
            SessionListener.addSession(user, request.getSession());
}
response.sendRedirect(HOME_PAGE);


LoginUserServlet.java
```

```
private static HashMap<String, HttpSession> activeSessions = new HashMap<String, HttpSession>();
```

The HttpSession hashmap will allow for the ability of checking the current users logged in on the website. The sessions could also be received from another user, and an attribute could be applied; which could output or do an action to that received user.

```
public static void addSession(User user, HttpSession session) {
		if (activeSessions.containsKey(user.getUserName())) activeSessions.remove(user.getUserName());
		updateStatus(user, User.USER_OFFLINE, User.USER_ONLINE);
		activeSessions.put(user.getUserName(), session);
	}

SessionListener.java
```

<u>Screenshot</u>



Email Address: testdata@test.com
Status Comment:
fred

The java server page files use JavaServer Pages Standard Tag Library (JSTL) developed by sun, which encapsulates conditions as simple tags, and can be used in conjunction with EL.

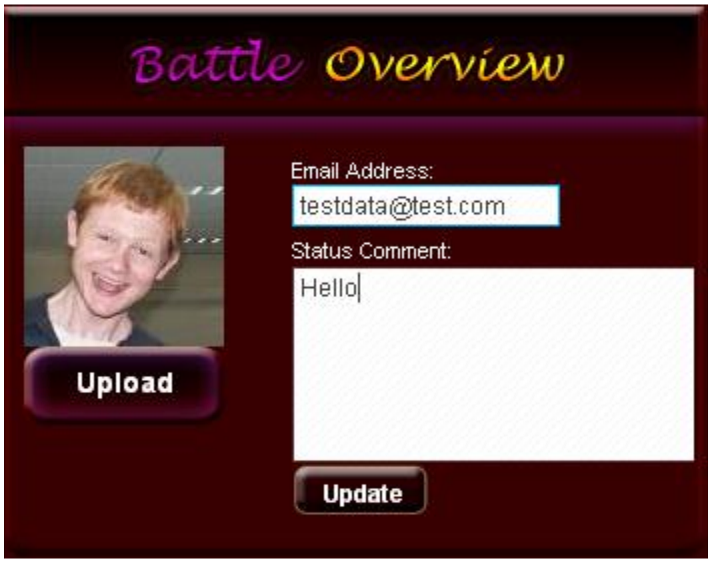
JSP Menu JTSL Functionality Example

```
<div id="menu_buttons">
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:choose>
   <c:when test="${empty sessionScope.LoggedInUser}">
      <form id="UserLoginForm" method="POST" action="LoginUser">
            // taken out for the example, as the main focus is to see the conditional tags demonstrated.
      </form>
   </c:when>
   <c:otherwise>
      <a id="Logout_Button" href="LoginUser?action=logout" title="Logout"></a>
   </c:otherwise>
</c:choose>
</div>
Web/ template/ menu.jsp
```

Modifying User Information

The user has the ability of updating their profile details. The form details are validated through the use of JavaScript, and sent to the corresponding servlet via an Ajax request.

i. The user has the ability of uploading a display picture; the display picture is uploaded through the use of JavaScript – it is impossible though to send the image via Ajax, so instead it creates an inline invisible frame, processes the upload to the servlet, and displays the new picture in the img on the page.
j. The uploaded image is then resized immediately to a fixed 100px in height (width may vary depending on the scale of the image, but it's placed within an img with a fixed width and height of 100px).
k. The user can update their email address, and also change their "Status Comment" on their account; which is viewable from the members listing page, and also their individual profiles.



Email Address: testdata@test.com

Status Comment: Hello

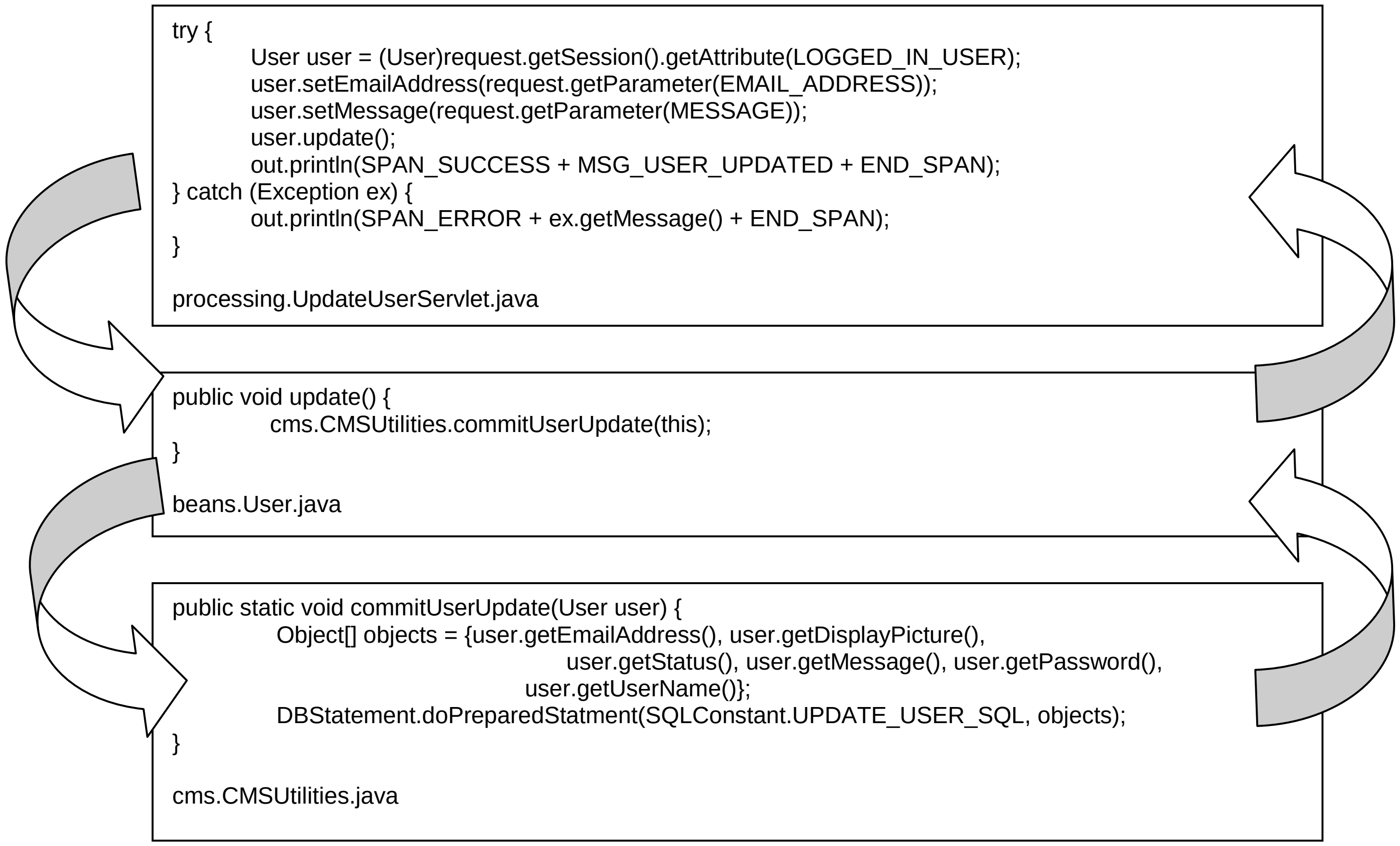
JavaScript Processing

The form is validated with JavaScript with the same CheckForm function to ensure that the user input is valid. The request is then sent to the servlet using Ajax for the transmission, and the result is then updated in the 'info_message' div.

```
function UpdateDetails() {
        if (CheckForm('UserUpdateForm')) {
            var params = "emailAddress="+$('emailAddress').value+"&message="+$('message').value;
            new Ajax.Updater("info_message", "UpdateUser.ajax", {
                method: 'post',
                parameters: params
            });
            $('info_message').style.display = '';
        }
}


Snake_ajax.js
```

Servlet Processing

- The stored session for the login, is grabbed from the request (javascript still using the current opened session) and thrown into a User bean object. The user bean is then updated using the methods for the corresponding parameters received, and then the update() method is called – which is binded to the user record in the user table. The output is then sent back to the client through javascript and placed inside the info_message div.



```
try {
            User user = (User)request.getSession().getAttribute(LOGGED_IN_USER);
            user.setEmailAddress(request.getParameter(EMAIL_ADDRESS));
            user.setMessage(request.getParameter(MESSAGE));
            user.update();
            out.println(SPAN_SUCCESS + MSG_USER_UPDATED + END_SPAN);
} catch (Exception ex) {
            out.println(SPAN_ERROR + ex.getMessage() + END_SPAN);
}

processing.UpdateUserServlet.java
```

```
public void update() {
              cms.CMSUtilities.commitUserUpdate(this);
}

beans.User.java
```

```
public static void commitUserUpdate(User user) {
              Object[] objects = {user.getEmailAddress(), user.getDisplayPicture(),
                                                   user.getStatus(), user.getMessage(), user.getPassword(),
                                              user.getUserName()};
              DBStatement.doPreparedStatment(SQLConstant.UPDATE_USER_SQL, objects);
}

cms.CMSUtilities.java
```

```
static void doPreparedStatment(String sql, Object[] params) {
               try {
                    PreparedStatement p = DBConnection.getConnection().prepareStatement(sql);
                    addParams(p, params);
                    p.executeUpdate();
                    p.close();
               } catch (SQLException e) {
                    System.err.println(e.getMessage());
               }
 }
```

cms.DBStatement.java

```
static Connection getConnection() {
        try {
            if (connection == null) {
                Class.forName(JDBC_DRIVER);
                DBConnection.connection = DriverManager.getConnection(JDBC_URL_LOCATION,
                DB_USER_NAME, DB_PASSWORD);
            }
        } catch (Exception ex) {
            System.err.println(ex.getMessage());
        }
        return DBConnection.connection;
}
```

cms.DBConnection.java

Overview JSP File

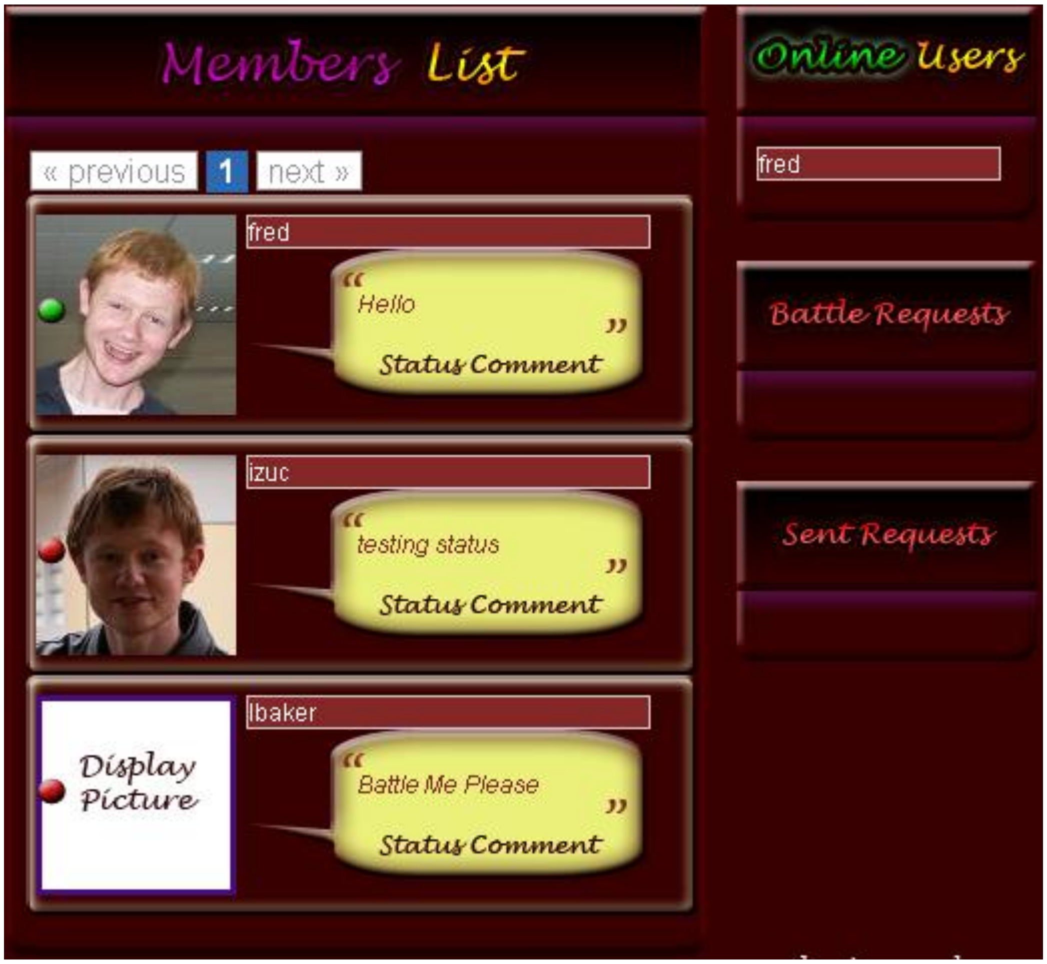
```
    <div id="content_box">
        <div id="overview_header"></div>
         <div id="content_repeat">
                <div id="profile_container">
                        <div id="left_section">
                            <img id="profile_image" src="avatars/${LoggedInUser.displayPicture}" />
                            <div id="uploadImage" class="Blank_Button">Upload</div>
                        </div>
                        <form id="UserUpdateForm" action="index.jsp" method="post">
                            <div id="right_section">
                                <div id="info_message" style="width: 98%; display: none;"></div>
                                <div class="group">
                                      <span class="label">Email Address:</span>
                                      <span class="content"><input type="text" id="emailAddress" class="required validate-email"
                                      value="${LoggedInUser.emailAddress}"/></span>
                                </div>
                                <div class="group">
                                      <span class="label">Status Comment:</span>
                                      <span class="content">
                                            <textarea id="message" class="validate-message"
                                             rows="5">${LoggedInUser.message}</textarea>
                                      </span>
                                 </div>
                                 <input id="updateUserButton" type="button" class="buttonSubmit" style="color: #FFFFFF;"
                                  value="Update" />
                            </div>
                        </form>
                  </div>
         </div>
         <div id="content_footer"></div>
    </div>
```

Web/ Pages/ Overview.jsp

Viewing Other Members

The members section is used to display a list of the current users; showing users in both online and offline state. The green dot inside the user's display picture represents a currently online status. The users name also appears within the Online Users box on the right bar immediately; through the use of running JavaScript requests in the background to the servlet.

- The members list has a paging system created, which breaks the results from the query to a set static limit of 5 – and generates page numbers for the maximum amount of calculated pages.
- The status comment is displayed for the member (which means they could post stuff like "Hey, Please Battle Me" to draw more attention, and if the user is interested to battle; they can click on their username which brings up their user profile – with the battle request functionality.



Servlet Processing

- The page parameter is only supplied when the user has clicked on a navigational hyperlink, so therefore the first page is initially 1. If a page parameter is supplied; it will check whether the number is of valid range, and calculate the offset for the query limit.
- The use of RequestDispatcher is demonstrated in the following code, which will therefore forward the request (and also any attributes added) to the home template page; which includes the members page when triggered.
- The .jsp members viewing page does not contain scriptlets, but it still processes the list of users received. It makes use of the iteration supported by the JSTL and allows iterating for each member using the forEach tag, and outputs the corresponding EL value in the html.

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    int page = 1, offset = 0, page_count = this.getPageCount();
    if (request.getParameter(PAGE) != null) {
        page = (isNumeric(request.getParameter(PAGE))) ? Integer.parseInt(request.getParameter(PAGE)): 1;
        if (request.getParameter(ACTION) != null) {
            page = (request.getParameter(ACTION).equalsIgnoreCase(NEXT)) ? this.getNextPage(page, page_count) :
                  (request.getParameter(ACTION).equalsIgnoreCase(PREV)) ? this.getPrevPage(page, page_count) : 1;
        }
        page = (page > page_count) ? page_count :
                    (page <= 1) ? 1 : page;
    }
    offset = (page - 1) * PER_PAGE;
    RequestDispatcher view = request.getRequestDispatcher(HOME_PAGE);

    request.setAttribute(PAGE, MEMBERS);
    request.setAttribute(CURRENT_PAGE, page);
    request.setAttribute(PAGE_COUNT, page_count);
    request.setAttribute(NEXT_PAGE, (page >= page_count) ? DISABLE_LINK : EMPTY_STRING);
    request.setAttribute(PREV_PAGE, (page <= 1) ? DISABLE_LINK : EMPTY_STRING);

    ArrayList<User> users = CMSUtilities.getUsers(offset, PER_PAGE);
    this.updateUser(users);
    request.setAttribute(MEMBERS, users);
    view.forward(request, response);
}


MembersServlet.java
```

JSP View

- The request is dispatched the index.jsp page, which does a JSTL when test; checking the value of the EL page attribute, and it includes the desired page. The member's page then loops through the amount pages of members available and creates the pagination.
- The members list is then iterated, and the attributes are forwarded to the member_box.jsp template (which therefore provides reuse for the viewing of the individual profiles).

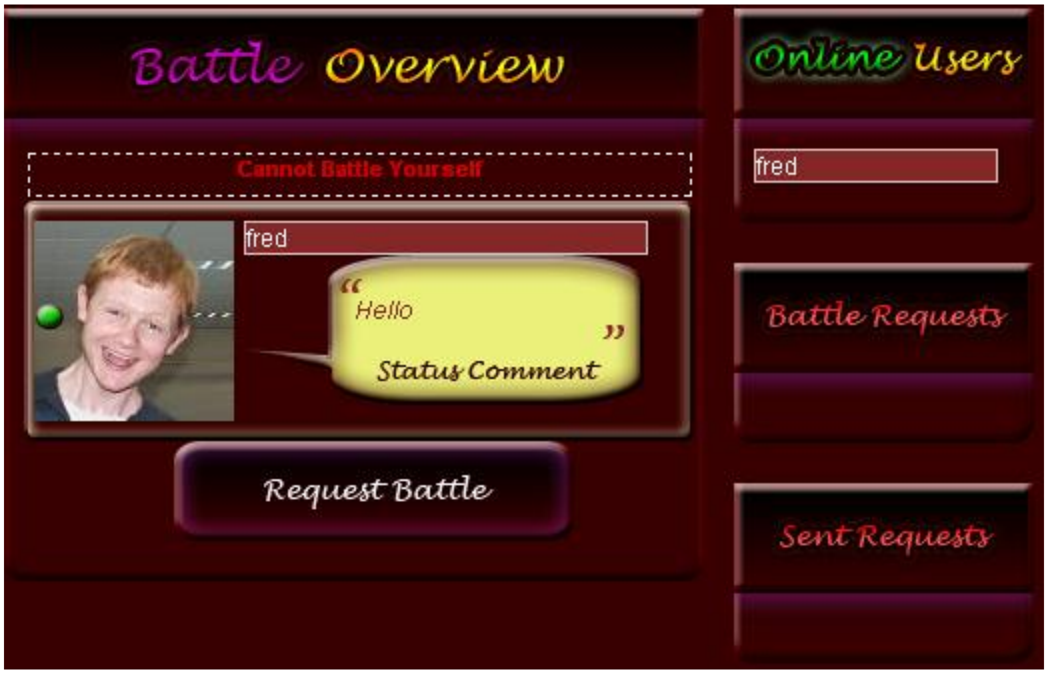
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<div id="content_box">
    <div id="members_header"></div>
    <div id="content_repeat">
        <div class="pagination">
            <ul>
                  <li><a href="members.jsp?page=${currentpage}&action=prev" class="prevnext${prev_page}">« previous</a></li>
                      <c:forEach var="i" begin="${1}" end="${pagecount}">
                           <li>
                                <c:choose>
                                    <c:when test="${currentpage eq i}">
                                        <a href="members.jsp?page=${i}" class="currentpage">${i}</a>
                                    </c:when>
                                    <c:otherwise>
                                        <a href="members.jsp?page=${i}">${i}</a>
                                    </c:otherwise>
                                </c:choose>
                            </li>
                      </c:forEach>
                  <li><a href="members.jsp?page=${currentpage}&action=next" class="prevnext${next_page}">next »</a></li>
            </ul>
        </div>
          <c:forEach var="member" items="${members}">
                <jsp:include page="/template/member_box.jsp" flush="true">
                    <jsp:param name="displayPicture" value="${member.displayPicture}"/>
                    <jsp:param name="status" value="${member.status}"/>
                    <jsp:param name="userName" value="${member.userName}"/>
                    <jsp:param name="message" value="${member.message}"/>
                </jsp:include>
          </c:forEach>
     </div>
     <div id="content_footer"></div>
</div>
```

pages / members.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
                <div class="member_box_container">
                    <div class="member_box_header"></div>
                    <div class="member_box_repeat">
                        <div class="member_box_picture">
                            <img class="member_image" src="avatars/${param.displayPicture}" />
                            <c:choose>
                                <c:when test="${param.status eq 1}">
                                    <div class="member_online"></div>
                                </c:when>
                                <c:otherwise>
                                    <div class="member_offline"></div>
                                </c:otherwise>
                            </c:choose>
                        </div>
                        <div class="member_box_content">
                            <a href="profile.jsp?user=${param.userName}" class="member_box_username">${param.userName}</a>
                            <div class="messagebox">
                                 <div class="message_header"></div>
                                 <div class="message_repeat">
                                     <div class="message_text">
                                          ${param.message}
                                     </div>
                                 </div>
                                 <div class="message_footer"></div>
                            </div>
                        </div>
                    </div>
                    <div class="member_box_footer"></div>
                </div>
     </div>
     <div id="content_footer"></div>
</div>
```

Includes **template / member_box.jsp (which also is included in the individual profile pages)**

The individual members profile servlet receives a user parameter, checks whether it is valid, and passes the value to the getUser(String username) method of the CMSUtilities class. It then attempts to retrieve the user, and if it has been found it will set the request with a member attribute and dispatch the request to the same member_box.jsp file used within the members section.



Servlet Processing

```
    private void checkUser(User user) {
            String path = this.getServletContext().getRealPath(USER_UPLOAD_PATH) + File.separator;
            cms.CMSUtilities.checkDisplayPicture(user, path);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
             RequestDispatcher view = request.getRequestDispatcher(HOME_PAGE);
             request.setAttribute(PAGE, PROFILE);
             if (request.getParameter("user") != null) {
                 User user = CMSUtilities.getUser(request.getParameter("user"));
                 if (user != null) {
                     request.setAttribute("member", user);
                     view.forward(request, response);
                 } else {
                     response.sendRedirect(HOME_PAGE);
                 }
             } else {
                 response.sendRedirect(HOME_PAGE);
             }
    }
```

ProfileServlet.java

JSP View

```
    <div id="content_box">
       <div id="overview_header"></div>
        <div id="content_repeat">
           <div id="profile_container">
             <div id="info_message" style="width: 330px; display: none;"></div>
             <jsp:include page="/template/member_box.jsp" flush="true">
                <jsp:param name="displayPicture" value="${member.displayPicture}"/>
                <jsp:param name="status" value="${member.status}"/>
                <jsp:param name="userName" value="${member.userName}"/>
                <jsp:param name="message" value="${member.message}"/>
             </jsp:include>
           </div>
           <div id="RequestBattle_Button"></div>
        </div>
        <div id="content_footer"></div>
    </div>

  pages / profile.java
```

Battle Requesting & Loading Games

The battle request feature makes use of the stored sessions within the session listener class. The concept of the design is to be able to send a battle request to a specific user, and that user can actually receive that request automatically through Ajax without having to refresh the page. The idea is, once the other reciprocant has accepted the battle – the sender will get automatically and instantaneously loaded into the game at the same time (without having to touch the browser).

1. The sender has the option of canceling the sent request.
2. The battle requests received are in real time, and appear instantaneously.
3. The action for requesting a battle is processed through Ajax:
    a. That both users exist within the system, and are currently online.
    b. Checks to ensure that the user submitting the request has not already received a request from the sending user.
        i. It also validates whether the sending user is not requesting a battle with their self.
    c. If the user goes offline, the sent and received requests are cleared on for all users.

4. Once the request has been sent, the sender will receive a new item within the sent requests section of the right bar. It will contain an outputted real time list of the sent requests which are pending. If the user wishes to stop a request, they have the option of canceling. Once it has been cancelled, the corresponding request is cleared from the receivers account.

5. The receiver of the request, will receive the new request in real time and have the option of accepting it via the accept button; which processes a servlet "*BattleServlet.java*". The servlet grabs the user information through the user parameter and the opponent parameter; both values stay the same for each user playing, it's just a way of defining the stored game name.

6. In the "*BattleServlet* "both users are checked to ensure that they exist within the database, and then checked to see whether they have an online status. The player's session is grabbed from the *SessionListener* and has a session attribute set; which is constantly checked via Ajax, sending requests to the *BattleLoadChecked* servlet to verify whether that specific session has been set.

The following servlet code is executed when the opponent has accepted the battle request. The yellow highlighted code demonstrates applying attributes to other online users.

```
if (SessionListener.isUserOnline(player.getUserName()) &&
(SessionListener.isUserOnline(opponent.getUserName()))) {
                        int type = (player.getUserName().equals(current_user.getUserName())) ? 1 : 2;
                        request.getSession().setAttribute("player_input", player.getUserName());
                        request.getSession().setAttribute("opponent_input", opponent.getUserName());
                        request.getSession().setAttribute("type_input", type);
                        if (opponent.getUserName().equals(current_user.getUserName())) {
                            HttpSession session = SessionListener.findSession(player);
                            session.setAttribute(ATRB_LOAD_GAME, current_user);
                        }
                    }
}

processing.BattleServlet.java
```

The javascript method BattleLoadCheck() is checks the BattleLoadCheckServlet every one second. The browser is only redirected if the response text from the servlet is greater than 0. If it is, it will redirect the page to the battle.jsp file and passing the parameters (the battle.jsp file is actually a mapping to the BattleServlet.

```
function LoadBattle(text) {
        if (text.length > 0) {
            window.location = "battle.jsp?"+text
        }
}

function BattleLoadCheck() {
        var url = "BattleLoadCheck.ajax";
        new Ajax.PeriodicalUpdater("battle_load", url, {
            method: 'POST',
            frequency: 1,
            onSuccess:  function(response) {
                LoadBattle(response.responseText);
            }
        });
}

Snake_ajax.js
```

The BattleLoadCheck servlet is accessed through ajax with the url pattern mapping of BattleLoadCheck.ajax (it isn't required, but I just figured it would be a nice touch when looking through the web.xml file; which identifies the ajax used servlets).

- The user checking for a battle request to be initiated is first checked to determine whether their session is still valid, and then checked whether they have the load attribute assigned. If it is there, it means that someone has accepted their request; which then grabs the sender's user name from the battle requests received hashmap stored in the user object. If it is there it means that the game should be initiated; so it therefore quickly removes the battle request from that user, and the sender, changes the session load game attribute to null – then outputs the ready made parameters with the standard PrintWriter. The response is picked up from JavaScript with the onSuccess event, and passed to a function which loads the battle page.

  The player doesn't even need to touch the browser!

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        if (request.getSession().getAttribute(LOGGED_IN_USER) != null) {
            User current_user = (User)request.getSession().getAttribute(LOGGED_IN_USER);
            if (request.getSession().getAttribute(ATRB_LOAD_GAME) != null) {
                User sender = (User)request.getSession().getAttribute(ATRB_LOAD_GAME);
                if (sender.getBattleRequests().containsKey(current_user.getUserName())) {
                    out.println("player="+current_user.getUserName()+"&opponent="+sender.getUserName());
                    sender.getBattleRequests().remove(current_user.getUserName());
                    current_user.getSentRequests().remove(sender.getUserName());
                    request.getSession().setAttribute(ATRB_LOAD_GAME, null);
                }
            }
        }
    }

processing.BattleLoadCheckServlet
```

7. The game applet receives the parameters from the passed in values as highlighted below, and then loads the game. The applet initially needs to communicate to the server side, via establishing a game. This is done in the GameControllerServlet.

```
<div id="snake_box">
    <div id="snake_header"></div>
     <div id="snake_repeat">
        <applet code="snakegame/SnakeGameFrame.class" archive="SnakeGame.jar" height="400" width="480">
            <param name="player" value="${player_input}">
            <param name="opponent" value="${opponent_input}">
            <param name="type" value="${type_input}">
        </applet>
     </div>
     <div id="snake_footer"></div>
</div>

pages / battle.jsp
```

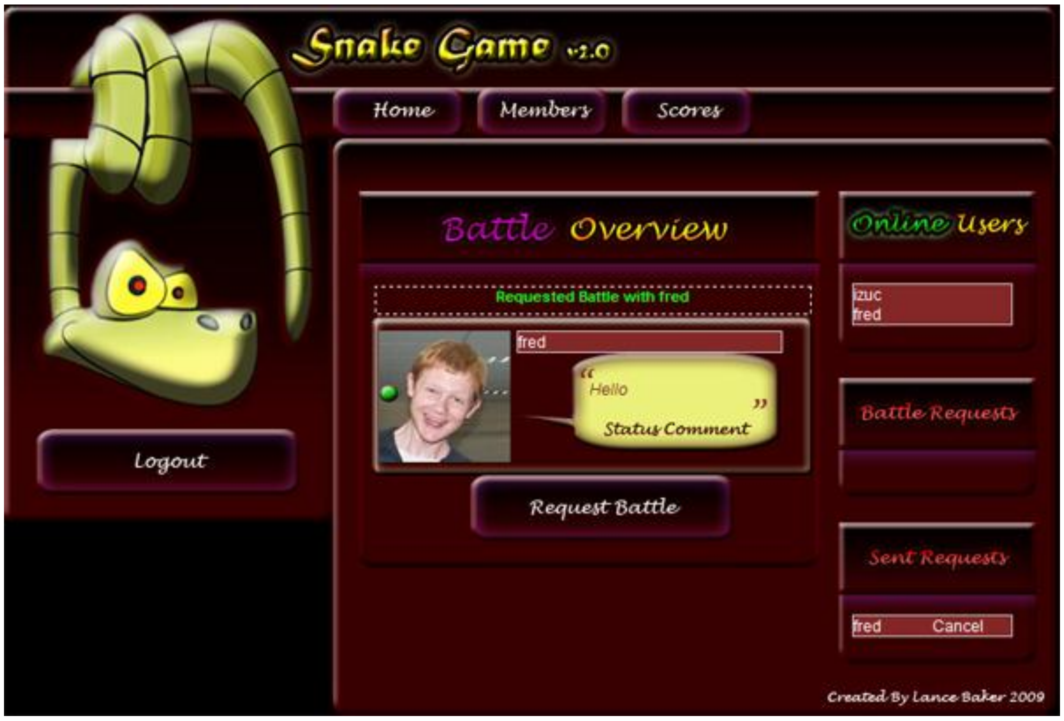
8. The game controller servlet is first called from the EstablishGame Class within the applet, which creates loads the game in a thread. It passes the parameters received from the applet using just the normal “GET” by just accessing the servlet with the query string following.
   a. It checks whether both users are currently online
   b. It instantiates a SConnection (Snake Connection) object which contains the loading messages, the game name, and also the status.
   c. It creates the game (usually it is done by the opponent since they are the first ones who get loaded into applet first). The game is created and added to a hashmap of SnakeGame with the key of the player’s name “underscore” opponent’s name.
   d. It registers the current user to the game, and constantly goes through the same process (skipping a lot, since it tests whether the game exists) until the other user enters.
   e. Once both users are registered, the game status is set to connected, and the SConnection object is once again sent back.
9. The running thread in the EstablishGame class evaluates to false, as the game is connected. Once it is connected, the game is created on the applet, and both the game name, player name, and opponent name is passed into the instantiated SnakeGame on the applet.

The applets establish game class contains a thread, which performs the getConnectionInfo request to the GameControllerServlet each second. The thread is terminated once the game has been created, and both users have registered for the game. It is presumed that the game will now work, so it creates the game on the SnakeGameFrame which then loads the SnakePanel. Please see the GameControllerServlet for a better understanding on how the game creation is done.

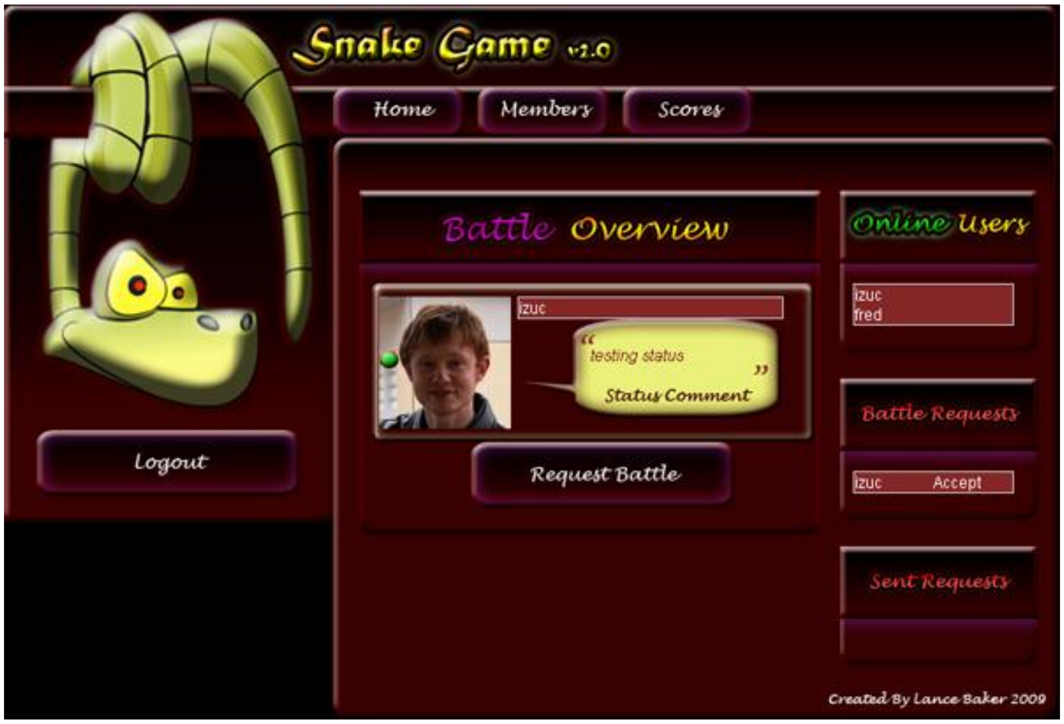
```
private String getURL() {
      return SERVLET_URL + "?player="+this.player+"&opponent="+this.opponent+"&type="+this.type;
}

public void getConnectionInfo() {
          try {
              URL servletURL = new URL(this.getURL());
              this.servletConnection = servletURL.openConnection();
              this.servletConnection.setDoInput(true);
              this.servletConnection.setUseCaches (false);
              this.servletConnection.setRequestProperty("Content-Type","multipart;application/octet-stream");
              ObjectInputStream inputFromServlet = new ObjectInputStream(servletConnection.getInputStream());
              this.snake_info = (SConnection)inputFromServlet.readObject();
              for (String message : this.snake_info.getMessages()) {
                  this.output.add(new JLabel(message), BorderLayout.CENTER);
              }
              if (this.snake_info.getState() == SConnection.State.CN_Connected) {
                  parent.createGame(this.snake_info.getGame(), this.player, this.opponent, this.type);
                  this.dispose();
              }
          } catch (Exception ex) {
              JOptionPane.showMessageDialog(parent, ex.getMessage());
          }
   }
```

SnakeGame Applet – snakegame.SnakeGameFrame. EstablishGame class

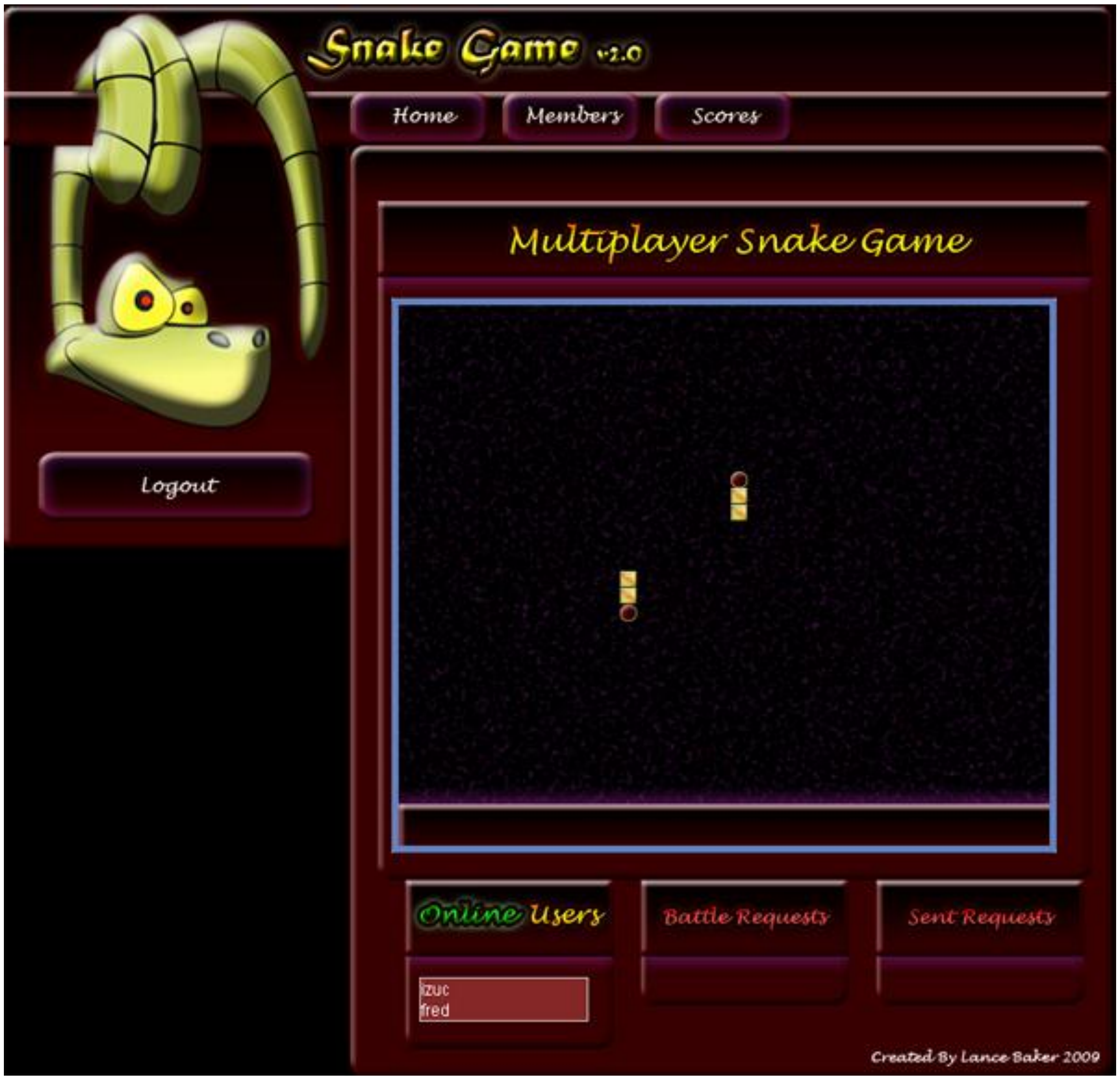
The izuc account has requested a battle with the Fred user. The izuc account user will now have to wait until Fred accepts the battle request. If the user accepts, izuc will be automatically entered into the game; following Fred.



The user Fred is currently logged in and viewing the profile page of the izuc account. The Fred user has just received a new battle request from Izuc, and decides to accept the invitation.



Once Fred has accepted the battle request, the account will be directed to the battle page, and sends a dynamic request to the initial sender; loading their game automatically.



The features/ functionality of the game are extremely trimmed down, but it demonstrates the ability to communicate to each other through servlets; by sending an ObjectOutputStream, processing it, and then receiving an ObjectInputStream back into the applet itself. Both players send their updated SnakePlayer object with the new snake Point items, and added to their section of the SnakeDisplay object. The SnakeDisplay object is then outputted to the applet, and the game renders both snakes.

The snake game applet now has the established game name, the player name, and the opponent. The processing for the updating of the snake players and for the retrieval of the snake display object is done within the “Process *GameDisplayServlet”.*

The following is two main methods in the applet for the sending of the SnakePlayer object, and for the retrieval of the SnakeDisplay object containing both players’ positions within the game.

```
    public void pushData() {
        try {
            URL servletURL = new URL(this.url);
            this.servletConnection = servletURL.openConnection();
            this.servletConnection.setDoOutput(true);
            this.servletConnection.setUseCaches (false);
            this.servletConnection.setDoInput(true);
            this.servletConnection.setRequestProperty("Content-Type","multipart;application/octet-stream");
            ObjectOutputStream outputToServlet = new ObjectOutputStream(this.servletConnection.getOutputStream());
            outputToServlet.writeObject(this.player);
            outputToServlet.flush();
        } catch (Exception ex ) {
        }
    }

    public void pullData() {
        try {
            InputStream in = this.servletConnection.getInputStream();
            ObjectInputStream ois = new ObjectInputStream(in);
            Object object = ois.readObject();
            this.display = (SnakeDisplay)object;
            this.player = (this.player_type == TYPE_PLAYER) ? this.display.getPlayer() : this.display.getOpponent();
        } catch (Exception ex ) {
        }
    }
```

Snake Game Applet – SnakeGame

The game pushData (which pushes the SnakePlayer) and the pullData method (which pulls the SnakeDisplay) are called within a running thread on the applets.

```
this.game.pushData();
this.game.pullData();
if (gameImage == null) gameImage = this.getContentPane().createImage(PANEL_WIDTH, (PANEL_HEIGHT + HUD_HEIGHT));
if (gameImage != null) {
        Graphics g =    this.gameRender();
                        this.game.getPlayer().moveSnake();
                        this.drawSnake(g, this.game.getDisplay().getPlayer().getSnakeBody());
                        this.drawSnake(g, this.game.getDisplay().getOpponent().getSnakeBody());
                        this.drawHud(g);
                        this.paintScreen();
}
```

Snake Game Applet – SnakePanel .updateGame() method (it gets called within a thread).

Servlet Processing

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
            try {
                    if ((request.getParameter("type") != null) && (request.getParameter("game") != null)) {
                        int type = Integer.parseInt(request.getParameter("type"));
                        SnakeGame game = GameControllerServlet.findGame(request.getParameter("game"));
                        if (game != null) {
                            if (game.gameReady()) {

                            }
                            ObjectInputStream inputFromApplet = new ObjectInputStream(request.getInputStream());
                            SnakePlayer player = (SnakePlayer)inputFromApplet.readObject();
                            if ((player != null) && (!player.toString().equalsIgnoreCase("empty"))) {
                                    if (type == GameControllerServlet.TYPE_PLAYER) {
                                        game.getDisplay().setPlayer(player);
                                    } else {
                                        game.getDisplay().setOpponent(player);
                                    }
                            }
                            SnakeDisplay display = game.getDisplay();
                            response.setContentType("java-internal/" + SnakeDisplay.class.getName());
                            OutputStream out = response.getOutputStream();
                            ObjectOutputStream oos = new ObjectOutputStream(out);
                            oos.writeObject(display);
                            oos.flush();
                            oos.close();
                        }
                    }
            } catch(Exception ex) {
                System.out.println(ex.getMessage());
            }
    }


processing.ProcessGameDisplayServlet
```