Software Engineering Group Project

Design Specification

|  |  |
| --- | --- |
| Author: | Group 12 |
| Config Ref: | SE\_G12\_DS\_00 |
| Date: | 2012-12-04 |
| Version: | 1.2 |
| Status: | Release |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2012

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. Decomposition Description 3

2.1 Applications in the System 3

2.2 Significant Classes 3

2.3 Requirements mapping 5

3. Dependency Description 5

3.1 Component Diagrams 5

4. Interface Description 7

4.1 Overview of Classes 7

4.2 Class Skeletons 8

5. Detailed Designs 17

5.1 Sequence Diagrams 17

5.2 Significant Algorithms 18

5.3 Significant Data structures 21

5.4 Class Diagrams 23

REFERENCES 26

DOCUMENT HISTORY 26

# Introduction

## Purpose of this Document

This document shows a detailed design of our game, including in depth class analysis, class diagrams and mapping requirements. It is to adhere to the design requirements [1] and the design specification requirements [2] QA documents, and to allow coders to understand and map out the classes that will be used during the creation of Monster Mash. The descriptions and diagrams are included to aid in the development of game and also to help in the maintenance of the project once it is completed

## Scope

This document gives access to the design information and class information. It will explain how these are linked together and which each class does. There is also discussion of the different applications in the project and the way in which the functionality of the project will be achieved.

## Objectives

* To accurately describe the classes that will be used in the project
* To show the functionality of these classes and how this will be achieved
* To create a document that aid in the creation of the prototype software
* Allow for the implementation to be completed adhering to the user requirements and efficiently

# Decomposition Description

## Applications in the System

There are two distinct applications - the client program, which runs in a web browser on the player's computer, and the server program, which will run on a server in the University.

Our client program will be what the player will see in the web browser. With this application, the player will be able to send friend requests, fight requests and other interactions with the server to be passed along to another player or other places, depending on the action. They can sell their monsters, breed their monsters with other players, and have their monsters fight with other players' monsters.

The server will generate the pages that are handed to the client program (the web browser). It will also pass data from the database to the users when required - such as the data from the marketplace, the leader board and the breeding market. Actions in the client program, such as selling a monster, will be passed back to the database on the server via the PersistenceManager.

Servers will also interact with each other, by hosting cross-server fights between players, trades and friend requests.

## Significant Classes

### Player

This class contains information about a single player/account. Email address (userID) and password instance variables are required for signing in. Each email address (userID) will be unique on our server. We need also serverID and username instance variables to work with “server to server” API. To make the application more secure, the password will be encrypted. For each player we will store their wealth as an integer. Each player will also have a list of friends and we decided to store them in an ArrayList of Players, because friend list will not be fixed size. There is an ArrayList of Monsters, which holds all monsters attached to a single player. Each player has also ArrayList of Notifications.

Player class contains four constructors. First one creates an object of player, who is not on our server, so we know only userID, username and serverID, then this object can be stored in a friend list of some player. Second constructor creates player by just taking userID, username, password, amount of money and name of initial monster and it is used for creating new account (it generates first monster and notifications). Third constructor has parameters for each instance variable and it is used for creating object of data taken from database. Last constructor takes no parameter and creates object with all null instance variables. Besides constructors, setters and getters, Player class contains method called **sortByMoney()**, which takes an ArrayList of Players and sort them by amout of money, so player with most money will be at the first position in the ArrayList.

### Monster

Monster is aclass containing information about a single monster.

Monster class contains following attributes:   
-id:int - ID of the monster (randomly generated String)  
-name:String - name of the monster  
-dob:Date - monster's date of birth  
-dod:Date - monster's date of death  
-baseStength:double - strength of the monster, used during breeding  
-currentStrength:double - strength of the monster, used during fighting  
-baseHealth:double - health of the monster, used during breeding  
-currentHealth:double - health of the monster, used during fighting  
-fertility:float - fertility of the monster  
-userID:String - ID of the owner  
-saleOffer:int - if other than 0, the monster is offered for sale  
-breedOffer:int - if other than 0, the monster is offered for breed  
-serverID:int - ID of the server on which the monster exists  
-MAX\_CHILDREN:int - maximum number of monsters that can be result of breeding

Monster class contains following methods:   
+fight(opponent:Monster):double - contains fighting algorithm. Takes the opponent monster as a parameter. Returns opponent's health.   
+breeding(other:Monster):Monster[] - contains breeding algorithm. Takes the monster to breed with as a parameter. Returns array of new monsters that are the result of breeding.   
+updateStats(strength:double, defence:double, health:double):void - updates statistics of the monster.

### Persistence manager

This is the only class which interacts with database. It has five private final instance variables: **dbname**, **dbhost**, **dbport**, **dbusername**, **dbpassword** and also three private instance variables: **connection** (which holds database connection), **error** (error message if any occurred) and **remote** (holds object of RemoteTalker class). The **constructor** of this class, which has no parameters, opens a connection to the database and creates new object of RemoteTalker class. This class contains a plenty of methods which operate on objects from data package. To communicate with database it uses SQL queries with values taken from that objects. When some error occurred, while executing SQL query, error message can be read from **getErrorMessage()** method. There is also one private method which doesn’t interact with database – it is **randomString()** method. It takes length and generates random string with that length. It is used for storing new records in database, which needs unique field like monsterID *(Note: we couldn’t use auto increment field in database, because API specified that we have to use string instead of integer for fields like monsterID, playerID etc)*.

### Other Persistence Manager

This class holds various methods used mainly by the cross server part of the application to communicate with the database. The class extends *PersistenceManager* so one will not need to have an instance of *PersistenceManager* and *OtherPersistenceManager* at the same scope.

## Requirements mapping

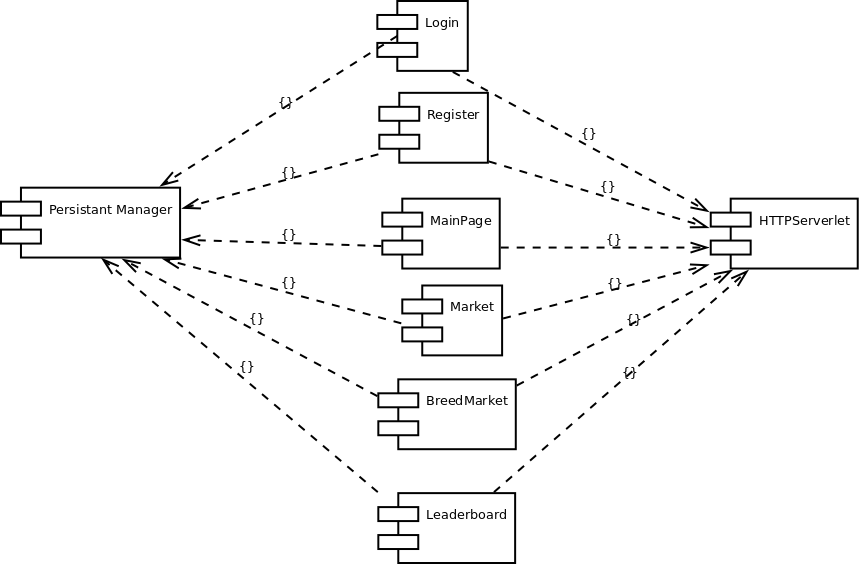
|  |  |
| --- | --- |
| Functional Requirement | Classes Providing Requirement |
| FR1 | Register, Player, Login, PersistenceManager |
| FR2 | Player, PersistenceManager |
| FR3 | Monster, Player, PersistenceManager |
| FR4 | Monster, Player, FightRequest, PersistenceManager |
| FR5 |  |
| FR6 | Register, Player, FriendRequest, Market, MarketOffer, BreedingMarket, BreedingOffer, *Monster*, MainPage, FightRequest, |
| FR7 | Player, Register, Login, PersistenceManager, OtherPersistanceManager |
| FR8 | Player, Monster, Notification, Market, FriendRequest, BreedingOffer, FightRequest, MarketOffer, PersistenceManager |
| FR9 | Player, FriendRequest, PersistenceManager, OtherPersistanceManager |
| FR10 | Notification, Player, Monster(Male), PersistenceManager |
| FR11 | Player, PersistenceManager, OtherPersistanceManager |

# Dependency Description

## Component Diagrams

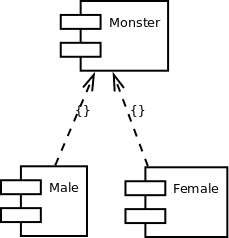
### Client

All the pages in the front-end use both the Persistence Manager and the HTTPServerlet. The PersistenceManager provides data for the webpages, and the HTTPServerlet serves the webpages to the web browser. This diagram describes the relationship between the different aspects of our project.



### Data

This diagram represents how the classes depend on another within the project.



# Interface Description

## Overview of Classes

### BreedingOffer

This class holds the information concerning a breeding offer including the players and monsters involved, the money required for a player to accept the offer and the date upon which it was placed.

### Monster

This is the class that models the monsters that user own and use, it holds all the genetic data of the monster such as strength and fertility rating as well as personal stats (name, date of birth, age, owner, age rate). It also contains the methods required to breed and fight with other monsters

### Fight Request

This is the class that stores the fight requests given it will store links to the players and monsters involve

### Friend Request

This is the class which models the sent requests and will store the players it is between and whether it has been accepted yet. It will also store the date that it was sent..

### MarketOffer

This class models the current monster sale offers within our server, this will contain the relevant monster and player and also the price of the monster.

### Notification

This Class is responsible for notification events. Players will get notifications if someone decides to fight them or buy their monster. There will also be notifications for breeding and adding the player as a friend.

### Player

This holds all player’s details that player enters at registration: username, password, email. Also it lists their friends and monsters, their money, notifications created by the Notification class. This class will be used to create player’s very first initial monster. The monsters will be stored in here as a list of Monster objects.

### Breeding Market

This class processes breed requests and actually sends or shows them.

### Leader Board

This class models the leaderboard and will store the list of the wealthiest players and the amount of money they have.

### Login

This class models the login data of a username/email address and password.

### Main Page

This class models the main page shown to the user in their browser.

### Market

This class is responsible for displaying the market which is a hash table of monsters for sale and also of those that you put for sale. It also processes buy and sell requests.

### Persistence Manager

is a class that manages all persistent data and communicates with database. Its responsibility is object manipulation. It can get friends, monsters, players, notifications; as well as add these things.

### Other Persistence Manager

### Register

This class is for registration it will create a player with their attributes (username, password, email) which will be saved in a player class.

## Class Skeletons

This sections shows some code skeletons automatically generated from our UML designs, these adhere to the Java coding specifications [3].

### BreedServlet

package ServerCom**;**

**import** data**.**Monster**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles the part of the API that allows a remote user to breed with

\* a monster that is offered for breeding on our server.

\* **@author** sis13

\*/

public class BreedServlet **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### BuyServlet

package ServerCom**;**

**import** data**.**Monster**;**

**import** data**.**NameGenerator**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles our part of the API that allows a remote server to buy

\* a monster that is offered for sale on our server.

\* **@author** sis13

\*/

public class BuyServlet **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### CONFIG

package data**;**

/\*\*

\* Configuration file

\*/

public class CONFIG

**{**

/\*\*

\* ID of the local server

\*/

public static final int OUR\_SERVER**;**

**}**

### CreateAccountPage

**import** data**.\*;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** javax**.**mail**.**internet**.**AddressException**;**

**import** javax**.**mail**.**internet**.**InternetAddress**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Servlet for creating a new account.

\*/

public class CreateAccountPage **extends** HttpServlet **{**

/\*\* SET INITIAL MONEY AMOUNT \*\*/

private final int MONEY\_AMOUNT **=** 10**;**

/\*\*

\* Check if email address is correct.

\* **@param** email user's email address

\* **@return** true when email address is correct

\*/

private boolean isValidEmailAddress**(**String email**)** **{**

**}**

/\*\*

\* Encode password using MD5.

\* **@param** md5 password

\* **@return** encoded password

\*/

public String MD5**(**String md5**)** **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

**}**

### FightAccept

**import** ServerCom**.**RemoteTalker**;**

**import** data**.\*;**

**import** database**.**OtherPersistenceManager**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**json**.**JSONException**;**

/\*\*

\* This servlet is called when a fight request is accepted, runs fight

\* algorithm and determinates who won and who lost.

\*/

public class FightAccept **extends** HttpServlet **{**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### FightingPage

**import** ServerCom**.**RemoteTalker**;**

**import** data**.**CONFIG**;**

**import** data**.**FightRequest**;**

**import** data**.**Monster**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**annotation**.**WebServlet**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

**import** org**.**json**.**JSONException**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Servlet for displaying page that allows user to select to fight with another

\* user's monsters.

\*

\* **@author** $Author: fiz$

\* **@version** $Id$

\*/

@WebServlet**(**name **=** "FightingPage"**,** urlPatterns **=** **{**"/fight"**})**

public class FightingPage **extends** HttpServlet **{**

/\*\* Encoder \*\*/

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Gets all data from DB, which will be displayed on fighting page.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void getDataFromDB**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

public void doRemoteRequest**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** IOException **{**

**}**

**}**

### FightLost

package ServerCom**;**

**import** data**.**FightRequest**;**

**import** data**.**Monster**;**

**import** data**.**NameGenerator**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**json**.**JSONException**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles our part of the API that allows remote servers to notify

\* us when a fight request originating form our server has been fought and lost.

\* **@author** sis13

\*/

public class FightLost **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### FightReject

package ServerCom**;**

**import** data**.**FightRequest**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**annotation**.**WebServlet**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Servlet for handling incoming fight rejections.

\* **@author** FZajac

\*/

@WebServlet**(**name **=** "FightReject"**,** urlPatterns **=** **{**"/fight/reject"**})**

public class FightReject **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### FightRequestServlet

package ServerCom**;**

**import** data**.\*;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**json**.**JSONException**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles incoming fight request from a remote server.

\* **@author** sis13

\*/

public class FightRequestServlet **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException**,** JSONException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### FightWon

package ServerCom**;**

**import** data**.**FightRequest**;**

**import** data**.**Monster**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles the part of the server to server API that allows a remote

\* Server to say that a fight request that originated from our server has been done

\* on the remote server, and that our user won the fight.

\* **@author** sis13

\*/

public class FightWon **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### Friend

package data**;**

/\*\*

\* The friend class represents a friendship between two players. It stores data

\* needed to identify the players and the server they belong too.

\* **@author** sjk4

\*/

public class Friend **{**

private String friendshipID**;**

private String remoteUserID**,** localUserID**;**

private int localServerID**,** remoteServerID**;**

private boolean friendshipConfirmed**;**

/\*\*

\* Friend constructor

\* **@param** friendshipID Unique friendship ID

\* **@param** remoteUserID The userID of the remote player (Sender)

\* **@param** localUserID The userID of the local player (Receiver)

\* **@param** localServerID The server ID of the local user.

\* **@param** remoteServerID The server ID of the remote user.

\* **@param** confirmed Boolean to decide if the friendship is confirmed.

\*/

public Friend**(**String friendshipID**,** String remoteUserID**,** String localUserID**,** int localServerID**,** int remoteServerID**,** String confirmed**){**

**}**

**}**

### FriendAccept

package ServerCom**;**

**import** data**.**Friend**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles the API that allows a remote server to accept a friend

\* request originally sent from our server.

\* **@author** sis13

\*/

public class FriendAccept **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### FriendReject

package ServerCom**;**

**import** data**.**Friend**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**annotation**.**WebServlet**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* The part of the API that allows remote servers to reject a friend request

\* originally sent from our server.

\* **@author** sis13

\*/

@WebServlet**(**name **=** "FriendReject"**,** urlPatterns **=** **{**"/friends/reject"**})**

public class FriendReject **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### FriendRequest

package ServerCom**;**

**import** data**.**CONFIG**;**

**import** data**.**Friend**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* This servlet handles our part of the API that receives friend requests from

\* a remote server.

\*

\* **@author** sis13

\*/

public class FriendRequest **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### HighscoresPage

**import** ServerCom**.**RemoteTalker**;**

**import** data**.**Player**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

/\*\*

\* Displays the highscores page.

\*/

public class HighscoresPage **extends** HttpServlet **{**

/\*\*

\* Gets all data from DB, which will be displayed on highscores page.

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void getDataFromDB**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

**}**

### JSONManager

package ServerCom**;**

**import** data**.**Monster**;**

**import** data**.**Player**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**json**.**JSONArray**;**

**import** org**.**json**.**JSONException**;**

**import** org**.**json**.**JSONObject**;**

/\*\*

\* The JSONManager turns data objects into JSON so they can be sent over the wire.

\* **@author** sis13

\*/

public class JSONManager **{**

/\*\*

\* Turns a player object into JSON.

\* **@param** player The layer object you want to turn into JSON.

\* **@return** Returns a JSONObject.

\*/

public static JSONObject jsonPlayer**(**Player player**)** **{**

**}**

/\*\*

\* Turns a Monster into JSON, so it can be sent over the wire.

\*

\* **@param** monster The monster you want to turn to JSON.

\* **@return** Returns a JSONObject of the monster.

\*/

public static JSONObject jsonMonster**(**Monster monster**)** **{**

**}**

/\*\*

\* Turns a ArrayList of monster into a JSON array of monsters.

\* **@param** monsters The monsters you want to turn into JSON.

\* **@return** A JSONArray of monsters.

\*/

public static JSONArray jsonMonsterList**(**ArrayList**<**Monster**>** monsters**)** **{**

**}**

/\*\*

\* Turns a ArrayList of players into a JSON array of monsters.

\* **@param** players ArrayList of players you want to turn to JSON.

\* **@return** A JSONArray of players.

\*/

public static String jsonUsers**(**ArrayList**<**Player**>** players**)** **{**

**}**

**}**

### LoginPage

**import** data**.**Player**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Servlet used for the login page.

\*/

public class LoginPage **extends** HttpServlet **{**

/\*\*

\* Encode password using MD5.

\* **@param** md5 password

\* **@return** encoded password

\*/

public String MD5**(**String md5**)** **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

**}**

### LogoutPage

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

/\*\*

\* Servlet used for logging out.

\*/

public class LogoutPage **extends** HttpServlet **{**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

**}**

### MainPage

**import** ServerCom**.**RemoteTalker**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

**import** data**.\*;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**util**.**Collections**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Servlet used for displaying the main page after logging in.

\*/

public class MainPage **extends** HttpServlet **{**

/\*\*

\* Gets all data from DB, which will be displayed on main screen (list of

\* friends, monsters and notifications).

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void getDataFromDB**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Check if user sent form with new friend request if so process it and send

\* friend request with proper notifications.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void sendFriendRequest**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Accept or cancel friend request.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void respondToFriendRequest**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

**}**

### MarketPage

**import** ServerCom**.**RemoteTalker**;**

**import** data**.**CONFIG**;**

**import** data**.**Monster**;**

**import** data**.**Notification**;**

**import** data**.**Player**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

/\*\*

\* Servlet used for displaying the market page.

\*/

public class MarketPage **extends** HttpServlet **{**

/\*\*

\* Gets all data from DB, which will be displayed on market page.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void getDataFromDB**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* After a user has offered a monster for sell, he can cancel the offer.

\* **@param** request servlet request

\* **@param** response servlet response

\* **@param** pm persistence manager

\* **@param** current object of current player

\* **@throws** ServletException

\* **@throws** IOException

\*/

private void cancelOffer**(**HttpServletRequest request**,** HttpServletResponse response**,** PersistenceManager pm**,** Player current**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Enables player to buy a monster available on the market.

\* **@param** request servlet request

\* **@param** response servlet response

\* **@param** pm persistence manager

\* **@param** current object of current player

\* **@throws** ServletException

\* **@throws** IOException

\*/

private void buyMonster**(**HttpServletRequest request**,** HttpServletResponse response**,** PersistenceManager pm**,** Player current**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Checks if the offered money is valid (an integer larger than 0)

\* **@param** offerAmount

\* **@return**

\*/

private boolean validOffer**(**String offerAmount**){**

**}**

**}**

### MatingPage

**import** ServerCom**.**RemoteTalker**;**

**import** data**.\*;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

/\*\*

\* Servlet used for displaying the mating page.

\*/

public class MatingPage **extends** HttpServlet **{**

/\*\*

\* Gets all data from DB, which will be displayed on mating page.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

private void getDataFromDB**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* After a user has offered a monster for breeding, he can cancel the offer.

\* **@param** request servlet request

\* **@param** response servlet response

\* **@param** pm persistence manager

\* **@param** current object of current player

\* **@throws** ServletException

\* **@throws** IOException

\*/

private void cancelOffer**(**HttpServletRequest request**,** HttpServletResponse response**,** PersistenceManager pm**,** Player current**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Enables player to breed with a monster available for breeding.

\* **@param** request servlet request

\* **@param** response servlet response

\* **@param** pm persistence manager

\* **@param** current object of current player

\* **@throws** ServletException

\* **@throws** IOException

\*/

private void breedMonster**(**HttpServletRequest request**,** HttpServletResponse response**,** PersistenceManager pm**,** Player current**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)** **throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Checks if the offered money is valid (an integer larger than 0)

\* **@param** offerAmount

\* **@return**

\*/

private boolean validOffer**(**String offerAmount**)**

**{**

**}**

**}**

### Monster

package data**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**Random**;**

/\*\*

\* Encapsulation of the Monster data. The data is stored in the database and

\* represents one of the the player's Monster.

\*/

public class Monster **{**

private final int START\_HEALTH **=** 100**;**

/\*\* lifespan of a monster \*\*/

public static final int LIFESPAN **=** 1000**\***60**\***60**\***60**\***24**\***7**;**

private String id**;**

private String name**;**

private Date dob**;**

private Date dod**;**

private double baseStrength**;**

private double currentStrength**;**

private double baseDefence**;**

private double currentDefence**;**

private double baseHealth**;**

private double currentHealth**;**

private float fertility**;**

private String userID**;**

private int saleOffer**;**

private int breedOffer**;**

private int serverID**;**

private final int MAX\_CHILDREN **=** 10**;**

/\*\*

\* Constructor taking parameters. This constructor is used for creating

\* Monster objects about remote Monsters before we get all the data.

\*

\* **@param** id The Monsters ID.

\* **@param** name The Monster's name.

\* **@param** userID The ID of the owner of the Monster.

\*

\* **@see** Player

\*/

public Monster**(**String id**,** String name**,** String userID**){**

**this.**id **=** id**;**

**this.**name **=** name**;**

**this.**userID **=** userID**;**

**}**

/\*\*

\* Constructor setting all the Monsters fields, the stats are random.

\* **@param** name

\* **@param** userID

\*/

public Monster**(**String name**,** String userID**){**

**}**

public Monster**(**String id**,** String name**,** Date dob**,** Date dod**,** Double baseStrength**,** Double currentStrength**,** Double baseDefence**,** Double currentDefence**,** Double baseHealth**,** Double currentHealth**,** float fertility**,** String userID**,** int saleOffer**,** int breedOffer**)** **{**

**}**

/\*\*

\* Breeding class to breed new monsters

\*

\* **@param** other Monster that is being bred with

\* **@return** Monster[] and array of the children

\*/

public Monster**[]** breeding**(**Monster other**)** **{**

**}**

/\*\*

\* Updates statistics of the mosnter

\* **@param** strength monster's updated strength

\* **@param** defence monster's updated defence

\* **@param** health monster's updated health

\*/

public void updateStats**(**Double strength**,** Double defence**,** Double health**)** **{**

**}**

/\*\*

\* Enrolls two Monsters in an epic battle.

\* **@param** opponent The Monster this monster will fight versus.

\* **@return** The opponent is returned with new stats..

\*/

public double fight**(**Monster opponent**){**

**}**

**}**

### Monsters

package ServerCom**;**

**import** data**.**Monster**;**

**import** database**.**OtherPersistenceManager**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**json**.**JSONObject**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Monster servlet handling our part of the API handling request used to retrieve

\* monster information from our database.

\* **@author** sis13

\*/

public class Monsters **extends** HttpServlet **{**

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

/\*\*

\* Get the information about all the monsters of a user.

\* **@param** userID The ID of the user you want to retrieve.

\* **@param** response Response object for error handling.

\* **@return** Returns null if it fails, if successful it returns a JSON string

\* of all the monsters belonging to the user.

\* **@throws** IOException

\*/

public String usersMonsters**(**String userID**,** HttpServletResponse response**)** **throws** IOException **{**

**}**

/\*\*

\* This method looks up a single monster from the database and returns

\* JSONed monster.

\* **@param** monsterID The ID of the monster the remote client wants.

\* **@param** response Response object for error handling.

\* **@return** Returns null if the method fails, and a JSON string of the monster

\* if successful.

\* **@throws** IOException

\*/

public String singleMonster**(**String monsterID**,** HttpServletResponse response**)** **throws** IOException **{**

**}**

**}**

### NameGenerator

package data**;**

**import** java**.**util**.\*;**

/\*\*

\* NameGenerator to generate random name for monsters. The name is a combination

\* of a front name and last name.

\*/

public class NameGenerator **{**

private static List startWords **=** **new** ArrayList**();**

private static List endWords **=** **new** ArrayList**();**

/\*\*

\* Get a random name, the random name is of a for name and last name.

\* **@return** The name.

\*/

public static String getName**()** **{**

**}**

/\*\*

\* Private function to get a random int between a range.

\* **@param** min Min

\* **@param** max Max

\* **@return** A random number between min and max.

\*/

private static int randomInt**(**int min**,** int max**)** **{**

**}**

/\*\*

\* Private method to get the a random element from a list.

\* **@param** v The list of strings

\* **@return** a random element.

\*/

private static String getRandomElementFrom**(**List**<**String**>** v**)** **{**

**}**

**}**

### Notification

package data**;**

**import** java**.**util**.**Date**;**

/\*\*

\* Encapsulation of data used to send notifications to Players who has gotten

\* request like friend/fight request, or has sold or breed a monster.

\*

\* $Author sis13 $

\*

\* **@see** Player

\* **@see** ServerCom.RemoteTalker

\* **@see** database.PersistenceManager

\*/

public class Notification**{**

/\*\* Attributes \*/

private int id**;**

private String shortText**;**

private String longText**;**

private Player player**;**

private Date timeSent**;**

/\*\*

\*

\* **@param** shortText Short title.

\* **@param** longText Longer description.

\* **@param** player Player to receive the notification.

\*/

public Notification**(**String shortText**,** String longText**,** Player player**){**

**this.**id **=** 0**;**

**this.**shortText **=** shortText**;**

**this.**longText **=** longText**;**

**this.**player **=** player**;**

**this.**timeSent **=** **new** Date**();**

**}**

/\*\*

\*

\* **@param** shortText Short title.

\* **@param** longText Longer description.

\* **@param** player Player to receive the notification.

\* **@param** timeSent Date object of the time when the notification was sent.

\*/

public Notification**(**int id**,** String shortText**,** String longText**,** Date timeSent**){**

**this.**id **=** id**;**

**this.**shortText **=** shortText**;**

**this.**longText **=** longText**;**

**this.**player **=** **null;**

**this.**timeSent **=** timeSent**;**

**}**

**}**

### OtherPersistenceManager

package database**;**

**import** data**.**FightRequest**;**

**import** data**.**Friend**;**

**import** data**.**Monster**;**

**import** data**.**Player**;**

**import** java**.**sql**.**Connection**;**

**import** java**.**sql**.**DriverManager**;**

**import** java**.**sql**.**ResultSet**;**

**import** java**.**sql**.**SQLException**;**

**import** java**.**sql**.**Statement**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**logging**.**Level**;**

/\*\*

\* **@see** PersistanceManager

\* **@author** sis13

\*/

public class OtherPersistenceManager **extends** PersistenceManager **{**

private final String dbname **=** "MonsterMash"**;**

private final String dbuser **=** "root"**;**

private final String dbpassword **=** "root"**;**

private final String dbhost **=** "localhost"**;**

private final String dbport **=** "1527"**;**

private Connection connection**;**

private String error**;**

public OtherPersistenceManager**()** **{**

String driver **=** "org.apache.derby.jdbc.EmbeddedDriver"**;**

String connectionURL **=** "jdbc:derby://" **+** dbhost **+** ":" **+** dbport **+** "/" **+** dbname **+** ";create=true;user=" **+** dbuser **+** ";password=" **+** dbpassword**;**

**try** **{**

Class**.**forName**(**driver**);**

**}** **catch** **(**java**.**lang**.**ClassNotFoundException e**)** **{**

e**.**printStackTrace**();**

**}**

**try** **{**

connection **=** DriverManager**.**getConnection**(**connectionURL**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

/\*\*

\* Accepts friend request

\* **@param** friend

\*/

public void acceptFriendRequest**(**Friend friend**)** **{**

**}**

/\*\*

\* Gets Friend object

\* @pram friendID id of the friend

\* **@return** Friend object

\*/

public Friend getFriend**(**String friendID**)** **{**

**}**

/\*\*

\* Rejects friend request

\* **@param** friend

\*/

public void rejectFriend**(**Friend friend**)** **{**

**}**

/\*\*

\* Adds a friend

\* **@param** friend

\*/

public void addFriend**(**Friend friend**)** **{**

**}**

/\*\*

\* Gets list of players

\* **@return** list of players

\*/

public ArrayList**<**Player**>** getPlayers**()** **{**

**}**

/\*\*

\* Gets a list of monsters owned by this player

\* **@param** playerID id of player

\* **@return** list of monsters owned

\*/

@Override

public ArrayList**<**Monster**>** getMonsterList**(**String playerID**){**

**}**

/\*\*

\* Stores fight request in the database

\* **@param** fr fight request

\*/

public void storeFightRequest**(**FightRequest fr**)** **{**

**}**

/\*\*

\* Gets monster

\* **@param** monsterID id of the monster

\* **@return** Monster object

\*/

public Monster getMonster**(**String monsterID**)** **{**

**}**

/\*\*

\* Gets fight request

\* **@param** fightID id of the fight

\* **@return** fight request

\*/

public FightRequest getFightRequest**(**String fightID**)** **{**

**}**

/\*\*

\* Updates monster's statistics

\* **@param** monster

\*/

public void updateMonster**(**Monster monster**)** **{**

**}**

/\*\*

\* Removes fight request

\* **@param** fr fight request

\*/

public void removeFightRequest**(**FightRequest fr**)** **{**

**}**

/\*\*

\* Updates player's money

\* **@param** player

\*/

public void updateMoney**(**Player player**)** **{**

**}**

/\*\*

\* Removes monster from the database

\* **@param** senderMonsterID id of the monster

\*/

public void removeMonster**(**String senderMonsterID**)** **{**

**}**

/\*\*

\* Gets friends of given player

\* **@param** player

\* **@return** list of friends

\*/

public ArrayList**<**Friend**>** getFriends**(**Player player**)** **{**

**}**

/\*\*

\* Removes user from the database

\* **@param** userID id of the user

\*/

public void removeUser**(**String userID**)** **{**

**}**

/\*\*

\* Gets player from another server

\* **@param** userID id of the player

\* **@return** Player object

\*/

public Player getPlayerSafe**(**String userID**)** **{**

**}**

**}**

### PersistenceManager

package database**;**

**import** ServerCom**.**RemoteTalker**;**

**import** data**.**Monster**;**

**import** data**.\*;**

**import** java**.**security**.**SecureRandom**;**

**import** java**.**sql**.\*;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Random**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**json**.**JSONException**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Manages persistent data.

\*/

public class PersistenceManager **{**

private final String dbname **=** "MonsterMash"**;**

private final String dbuser **=** "root"**;**

private final String dbpassword **=** "root"**;**

private final String dbhost **=** "localhost"**;**

private final String dbport **=** "1527"**;**

private Connection connection**;**

private String error**;**

private RemoteTalker remote**;**

public PersistenceManager**(){**

**}**

/\*\*

\* Generates random string, which will be used as IDs for monsters/players etc.

\* **@param** length length of a string

\* **@return** random string with specified length

\*/

private String randomString**(**int length**){**

**}**

/\*\*

\* Checks if there is an account with specified email address.

\* **@param** email user's email address

\* **@return** true if account exists

\*/

public boolean accountExists**(**String userID**){**

**}**

/\*\*

\* Store player with monsters and notifications in DB.

\* **@param** email user's email address

\* **@param** password encrypted password (md5)

\* **@param** money start amount of money

\* **@return** true when account created successfully

\*/

public void storePlayer**(**Player p**){**

**}**

/\*\*

\* Stores all notifications with ID = 0 (which haven't been saved in DB)

\* **@param** p object of Player class

\*/

public void storeNotifications**(**Player p**){**

**}**

/\*\*

\* Stores all monsters with ID = 0 (which haven't been saved in DB)

\* **@param** p object of Player class

\*/

public void storeMonsters**(**Player p**){**

**}**

/\*\*

\* Gets player object of DB, returns null when user doesn't exist

\* **@param** email user's email address

\* **@param** password encrypted password using MD5

\* **@return** object of player class with all monsters, notifications and friends

\*/

public Player doLogin**(**String email**,** String password**){**

**}**

/\*\*

\* Gets all friends from DB of specified player

\* **@param** playerID ID of player

\* **@return** list of friends and friend requests

\*/

public ArrayList**<**Player**>** getFriendList**(**String playerID**){**

**}**

/\*\*

\* Gets all fight requests from DB.

\* **@param** playerID id of player

\* **@return** list of fight requests

\*/

public ArrayList**<**FightRequest**>** getFightRequests**(**String playerID**){**

**}**

/\*\*

\* Gets all notifications from DB ordered by date.

\* **@param** playerID id of player

\* **@return** list of notifications

\*/

public ArrayList**<**Notification**>** getNotificationList**(**String playerID**){**

**}**

/\*\*

\* Gets a list of monsters owned by this player

\* **@param** playerID id of player

\* **@return** list of monsters owned

\*/

public ArrayList**<**Monster**>** getMonsterList**(**String playerID**){**

**}**

/\*\*

\* Gets player from DB with all monsters, notifications, friends.

\* **@param** id id of player

\* **@return** object of Player class, null when player doesn't exist

\*/

public Player getPlayer**(**String userID**){**

**}**

/\*\*

\* Returns player id at index 0 and player server id at index 1

\* **@param** userID player's userID

\* **@return** player name (index 0) player server id (index 1)

\*/

public int getPlayerServerID**(**String userID**){**

**}**

/\*\*

\* Checks if friend request has been already sent.

\* **@param** playerOne userID of first player

\* **@param** playerTwo userID of second player

\* **@return** true when such a request was sent

\*/

public boolean isFriendRequestSent**(**String playerOne**,** String playerTwo**){**

**}**

/\*\*

\* Adds new record to "Friendship" table and if user is on different server, sends JSON object.

\* **@param** senderID userID of sender (always our server)

\* **@param** receiverID userID of reciver

\* **@param** receiverServerID server ID of receiver

\*/

public void sendFriendRequest**(**String senderID**,** String receiverID**,** int receiverServerID**){**

**}**

/\*\*

\* Confirms friendship between players (senderID and receiverID)

\* **@param** senderID id of player who sent request

\* **@param** senderServer address of sender's server

\* **@param** receiverID id of player who accepted request

\* **@param** receiverServer address of receivers's server

\*/

public void confirmFriendship**(**String senderID**,** int senderServer**,** String receiverID**,** int receiverServer**){**

**}**

/\*\*

\* Cancel friendship request between players (senderID and receiverID)

\* **@param** senderID id of player who sent request

\* **@param** senderServer address of sender's server

\* **@param** receiverID id of player who accepted request

\* **@param** receiverServer address of receivers's server

\*/

public void rejectFriendship**(**String senderID**,** int senderServer**,** String receiverID**,** int receiverServer**){**

**}**

/\*\*

\* Remove friendship request between players (playerOne and playerTwo)

\* **@param** playerOne user id of first player

\* **@param** playerTwo user id of second player

\*/

public void removeFriendship**(**String playerOne**,** String playerTwo**){**

**}**

/\*\*

\* Get highscores for logged player, ordered by amount of money

\* **@param** playerID id of logged player

\* **@return** ArrayList of HTML table rows

\*/

public ArrayList**<**String**>** getHighscores**(**String playerID**){**

**}**

/\*\*

\* Gets player username by player id and server id.

\* **@param** playerID id of selected player

\* **@param** serverID id of player's sever

\* **@return** player's username

\*/

public String getPlayerUsername**(**String playerID**,** int serverID**){**

**}**

/\*\*

\* Gets all friend requests from DB.

\* **@param** playerID id of player

\* **@return** list of friend requests

\*/

public ArrayList**<**String**>** getFriendRequestList**(**String userID**){**

**}**

/\*\*

\* Accepts a firend request

\* **@param** requestID id of the request

\* **@param** receiverID id of the receiver

\*/

public void acceptFriendRequest**(**String requestID**,** String receiverID**){**

**}**

/\*\*

\* Cancels a friend request

\* **@param** requestID id of the request

\* **@param** receiverID id of the receiver

\*/

public void cancelFriendRequest**(**String requestID**,** String receiverID**){**

**}**

/\*\*

\* Gets monsters for sale from the database.

\* **@param** playerID id of the player

\* **@return** list of monsters for sale

\*/

public ArrayList**<**Monster**>** getMonstersForSale**(**String playerID**){**

**}**

/\*\*

\* Gets monsters for breeding from the database.

\* **@param** playerID id of the player

\* **@return** list of monsters for breeding

\*/

public ArrayList**<**Monster**>** getMonstersForBreeding**(**String playerID**){**

**}**

/\*\*

\* Creates new market offer

\* **@param** userID id of the user

\* **@param** monsterID id of the monster

\* **@param** offerAmount amount of money paid for monster

\* **@return** whether the monster has been offered for sale

\*/

public boolean makeNewMarketOffer**(**String userID**,** String monsterID**,** int offerAmount**){**

**}**

/\*\*

\* Creates new breeding offer

\* **@param** userID id of the user

\* **@param** monsterID id of the monster

\* **@param** offerAmount amount of money paid for breeding with the monster

\* **@return** whether the monster has been offered for breeding

\*/

public boolean makeNewBreedOffer**(**String userID**,** String monsterID**,** int offerAmount**){**

**}**

/\*\*

\* Cancels monster sale offer

\* **@param** userID id of the user

\* **@param** monsterID id of the monster

\* **@return** whether the monster offer has been canceled

\*/

public boolean cancelMonsterOffer**(**String userID**,** String monsterID**){**

**}**

/\*\*

\* Cancels monster breeding offer

\* **@param** userID id of the user

\* **@param** monsterID id of the monster

\* **@return** whether the monster offer has been canceled

\*/

public boolean cancelBreedingOffer**(**String userID**,** String monsterID**){**

**}**

/\*\*

\* Gets monster's name

\* **@param** monsterID id of the monster

\* **@return** monster's name

\*/

public String getMonsterName**(**String monsterID**){**

**}**

/\*\*

\* Buys a monster from the market.

\* **@param** userID id of the user

\* **@param** monsterID id of the monster

\* **@param** serverID id of the server

\* **@return** new id of the bought monster

\*/

public String buyMonster**(**String userID**,** String monsterID**,** int serverID**){**

**}**

/\*\*

\* Gets a monster

\* **@param** monsterID id of the monster

\* **@param** serverID id of the server

\* **@return** Monster object

\*/

public Monster getMonster**(**String monsterID**,** int serverID**){**

**}**

/\*\*

\* Updates money of given player

\* **@param** player player object

\*/

public void updateMoney**(**Player player**)** **{**

**}**

/\*\*

\* Checks if any monster dies

\*/

public void checkIfAnyMonsterDies**(){**

**}**

/\*\*

\* Executes given query

\* **@param** query query to execute

\*/

public boolean insert**(**String query**){**

**}**

/\*\*

\* Counts number of records for a given query

\* **@param** query

\* **@return** number of records

\*/

public int count**(**String query**){**

**}**

/\*\*

\* Gets error message

\* **@return** error message

\*/

public String getErrorMessage**(){**

**}**

**}**

### Player

package data**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**Random**;**

/\*\*

\* The Player class is the representation of the Player's i our database.

\* **@author** $Author sis13 $

\*/

public class Player **{**

/\*\* Attributes \*/

private String userID**;**

private String password**;**

private String username**;**

private int money**;**

private int serverID**;**

private ArrayList**<**Player**>** friends**;**

private ArrayList**<**Notification**>** notifications**;**

private ArrayList**<**Monster**>** monsters**;**

/\*\*

\* Creates object of a new "friend".

\* **@param** email username (email address)

\* **@param** password encrypted password

\* **@param** money default amount of money

\* **@param** initialMonsterName name of random initial monster

\*/

public Player**(**String userID**,** String username**,** int serverID**){**

**}**

/\*\*

\* Creates object of a new player.

\* **@param** email username (email address)

\* **@param** password encrypted password

\* **@param** money default amount of money

\* **@param** initialMonsterName name of random initial monster

\*/

public Player**(**String userID**,** String username**,** String password**,** int money**,** String initialMonsterName**){**

**}**

/\*\*

\* Creates object of a player selected from DB

\* **@param** id player's id from DB

\* **@param** email player's email address

\* **@param** password encrypted password

\* **@param** money current amount of money

\* **@param** friends list of player's friends

\* **@param** notifications list of player's notifications

\* **@param** monsters list of player's monsters

\*/

public Player**(**String userID**,** String username**,** String password**,** int money**,** ArrayList**<**Player**>** friends**,** ArrayList**<**Notification**>** notifications**,** ArrayList**<**Monster**>** monsters**,** int serverID**){**

**}**

public Player**()** **{**

**}**

/\*\*

\* Sorts a ArrayList of players by the amount of money. The Player with

\* the most money should appear in the front of the list.

\* **@param** players List of Players you want to sort.

\* **@return** Returns a list of sorted Players.

\*/

public ArrayList**<**Player**>** sortByMoney**(**ArrayList**<**Player**>** players**)** **{**

**}**

/\*\*

\* Adds a friend

\* **@param** friend

\*/

public void addFriend**(**Player friend**){**

**}**

/\*\*

\* Removes given friend

\* **@param** friend

\*/

public void removeFriend**(**Friend friend**){**

**}**

/\*\*

\* Adds a monster

\* **@param** monster

\*/

public void addMonster **(** Monster monster **){**

**}**

/\*\*

\* Removes given monster

\* **@param** monster monster to remove

\*/

public void removeMonster **(** Monster monster **){**

**}**

/\*\*

\* Adds a notification

\* **@param** notefication

\*/

public void addNotification **(** Notification notification **){**

**}**

**}**

### RemoteTalker

package ServerCom**;**

**import** com**.**sun**.**jersey**.**api**.**client**.**Client**;**

**import** com**.**sun**.**jersey**.**api**.**client**.**WebResource**;**

**import** com**.**sun**.**jersey**.**api**.**client**.**config**.**ClientConfig**;**

**import** com**.**sun**.**jersey**.**api**.**client**.**config**.**DefaultClientConfig**;**

**import** com**.**sun**.**jersey**.**core**.**util**.**MultivaluedMapImpl**;**

**import** data**.\*;**

**import** database**.**OtherPersistenceManager**;**

**import** database**.**PersistenceManager**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**Random**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**ws**.**rs**.**core**.**MultivaluedMap**;**

**import** org**.**json**.**JSONArray**;**

**import** org**.**json**.**JSONException**;**

**import** org**.**json**.**JSONObject**;**

/\*\*

\* RemoteTalker holds all the methods needed for the client part of the

\* server to server communication. It has methods to access all of the

\* request specified by the standards which the groups agreed on.

\*

\* **@author** $Author sis13 $

\*/

public class RemoteTalker **{**

private Client client**;**

private WebResource resource**;**

public RemoteTalker**()** **{**

ClientConfig config **=** **new** DefaultClientConfig**();**

client **=** Client**.**create**(**config**);**

client**.**setConnectTimeout**(**2000**);**

client**.**setReadTimeout**(**5000**);**

**}**

/\*\*

\* Get a Player from a remote server.

\* **@param** userID userID of the player you want to request.

\* **@param** remoteAddress The address to the remote server.

\*

\* **@return** A Player object.

\*

\* **@see** data.Player

\*/

public Player getRemotePlayer**(**String userID**,** String remoteAddress**)** **{**

**}**

/\*\*

\* Gets a Monster from a remote server.

\*

\* **@param** monsterID Id of the monster you want.

\* **@param** remoteAddress The address of the server you want to look up the

\* monster at.

\* **@return** Monster object if found, returns null elswise.

\* **@throws** JSONException

\*

\* **@see** data.Monster

\*/

public Monster getRemoteMonster**(**String monsterID**,** String remoteAddress**)** **throws** JSONException **{**

**}**

/\*\*

\* Get all the monsters of a user from a remote server.

\*

\* **@param** userID The userID of the user you want monsters off.

\* **@param** remoteAddress Address of the server.

\*

\* **@return** returns a list if successful and null if something goes wrong.

\* **@throws** JSONException

\*

\* **@see** data.Monster

\*/

public ArrayList**<**Monster**>** getRemoteUsersMonsters**(**String userID**,** String remoteAddress**)** **throws** JSONException **{**

**}**

/\*\*

\* Convert a server number to a remote address.

\* **@param** serverNumber

\* **@return** The remote address.

\*/

public String getRemoteAddress**(**int serverNumber**)** **{**

**}**

/\*\*

\* Send a friend request to a user on a remote server.

\*

\* **@param** localUser The user you want to add as friend.

\* **@param** remoteUserID Your local userID.

\* **@param** serverNumber Your server number

\*

\* **@return** returns true if the function is successful, false otherwise.

\*

\* **@see** data.Player

\*/

public Boolean remoteFriendRequest**(**Player localUser**,** String remoteUserID**,** int serverNumber**)** **{**

**}**

/\*\*

\* Accept a friend request gotten from another server.

\*

\* **@param** friend Friend ship data bout the friend ship.

\*

\* **@return** Returns true if successful, false otherwise.

\*/

public Boolean acceptRemoteFriendRequest**(**Friend friend**)** **{**

**}**

/\*\*

\* Reject a remote friend request.

\* **@param** friend Friendship object holding data bout the friendship.

\* **@return** true if successful false otherwise.

\*/

public Boolean rejectRemoteFriendRequest**(**Friend friend**)** **{**

**}**

/\*\*

\* Send a remote fight request to a remote server given by the server number.

\*

\* **@param** fightRequest Fight request object with the data about the fight.

\* **@param** serverNumber Server number the request is going too.

\*

\* **@return** true if successful false otherwise.

\*

\* **@see** data.FightRequest

\*/

public Boolean remoteFightRequest**(**FightRequest fightRequest**,** int serverNumber**)** **{**

**}**

/\*\*

\* Send a request that a fight has been won.

\* **@param** fightRequest

\* **@param** serverNumber

\* **@param** monster

\* **@return**

\*/

public Boolean wonRemoteFight**(**FightRequest fightRequest**,** int serverNumber**,** Monster monster**)** **{**

**}**

/\*\*

\* Sends a request to a remote server indicating our local user lost a fight

\* versus one of their's users.

\* **@param** fightRequest Fight request object of the lost fight.

\* **@param** serverNumber The server number of the remote user.

\* **@return** Returns true if the request was sent successfully.

\*/

public Boolean lostRemoteFight**(**FightRequest fightRequest**,** int serverNumber**)** **{**

**}**

/\*\*

\* Reject a fight sent from a remote server.

\* **@param** fightRequest The fight request object for this fight.

\* **@param** serverNumber The server number of the remote server.

\*

\* **@return** Returns true if the request was sent successfully.

\*/

public Boolean rejectRemoteFight**(**FightRequest fightRequest**,** int serverNumber**)** **{**

**}**

/\*\*

\* Method to send a breed request to a remote server.

\*

\* **@param** monsterID The ID of the monster you want to breed with.

\* **@param** serverNumber Remote server ID.

\* **@return**

\*/

public Boolean sendBreedRequest**(**String monsterID**,** int serverNumber**)** **{**

**}**

/\*\*

\* Sends a buy request for a monster on a remote server.

\* **@param** monsterID The ID of the monster you want to buy.

\* **@param** serverNumber The ID of the remote server.

\*

\* **@return** Returns true if the request was sent successfully.

\*/

public Boolean sendBuyRequest**(**String monsterID**,** int serverNumber**)** **{**

**}**

/\*\*

\* Finds a user. The method will first search locally for the user, and if

\* the user is not found the method will search for the user on a remote

\* server.

\*

\* **@param** userID The ID of the user you want to find.

\*

\* **@return** Returns true if the request was successful.

\*/

public Player findUser**(**String userID**)** **{**

**}**

/\*\*

\* Will generate the highscore list for a player. It will fetch any friends

\* that are on remote server as well.

\*

\* **@param** player Player object of the player you want to generate a high score list for.

\* **@return** Returns a HTML list of the Players in sorted order.

\*/

public ArrayList**<**String**>** getHighScores**(**Player player**)** **{**

**}**

**}**

### Unregister

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** javax**.**servlet**.**http**.**HttpSession**;**

/\*\*

\* Allows user to unregister his account, deleting his monsters, etc.

\*/

public class Unregister **extends** HttpServlet **{**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

**}**

**}**

### User

package ServerCom**;**

**import** data**.**Player**;**

**import** database**.**OtherPersistenceManager**;**

**import** database**.**PersistenceManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** javax**.**servlet**.**ServletException**;**

**import** javax**.**servlet**.**annotation**.**WebServlet**;**

**import** javax**.**servlet**.**http**.**HttpServlet**;**

**import** javax**.**servlet**.**http**.**HttpServletRequest**;**

**import** javax**.**servlet**.**http**.**HttpServletResponse**;**

**import** org**.**owasp**.**esapi**.**Encoder**;**

**import** org**.**owasp**.**esapi**.**codecs**.**OracleCodec**;**

**import** org**.**owasp**.**esapi**.**reference**.**DefaultEncoder**;**

/\*\*

\* Servlet implementing the API used to get users from this server.

\* If the url /users is requested from our server server the servlet will return

\* a json array containing all the users registerd on the server. If the client

\* wants a specific user only it can be requested by sending a request to

\* /users?userID=xxx where xxx is the ID of the user.

\*

\* **@author** $Author sis13 $

\*

\* **@see** data.Player

\*/

@WebServlet**(**name **=** "users"**,** urlPatterns **=** **{**"/users"**})**

public class User **extends** HttpServlet **{**

/\*\* encoder \*\*/

Encoder encoder **=** **new** DefaultEncoder**();**

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* **@param** request servlet request

\* **@param** response servlet response

\* **@throws** ServletException if a servlet-specific error occurs

\* **@throws** IOException if an I/O error occurs

\*/

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

**}**

/\*\*

\* Returns a short description of the servlet.

\*

\* **@return** a String containing servlet description

\*/

@Override

public String getServletInfo**()** **{**

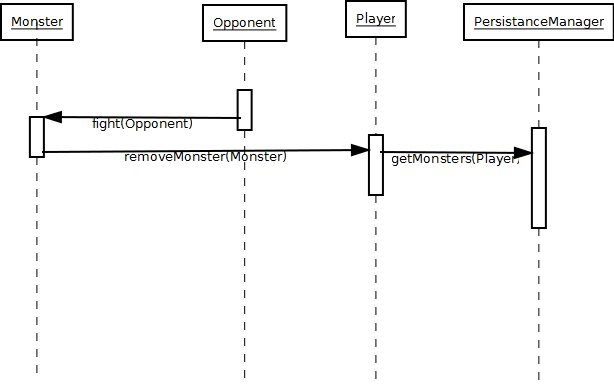
**}  
}**

# Detailed Designs

## Sequence Diagrams

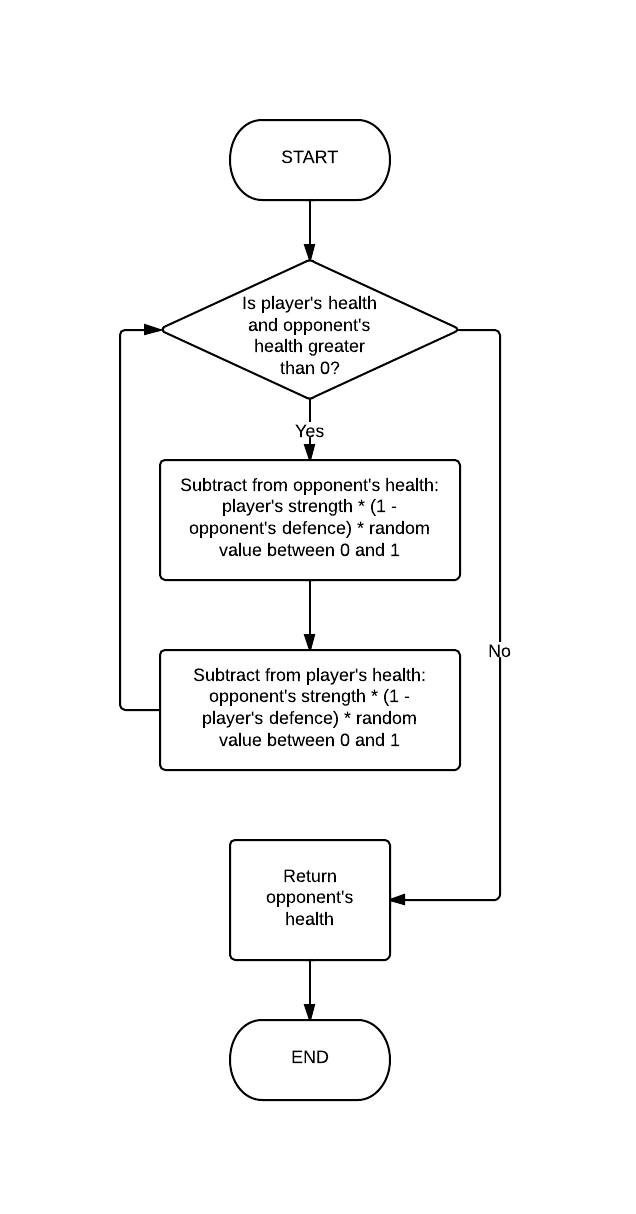
### Fighting

This shows how the fight would take place at a slightly higher level than the actual algorithm. The dead monster would be passed back to the player (owner) who would send it off to get removed from the server.



## Significant Algorithms

### Battling



public double fight**(**Monster opponent**)**

**{**

Random randomGenerator **=** **new** Random**();**

double random **=** randomGenerator**.**nextDouble**();**

**while(this.**currentHealth **>** 0 **&&** opponent**.**currentHealth **>** 0**)** **{**

opponent**.**currentHealth **-=** **this.**currentStrength **\*** **(**1**-**

opponent**.**currentDefence**)** **\*** random**;**

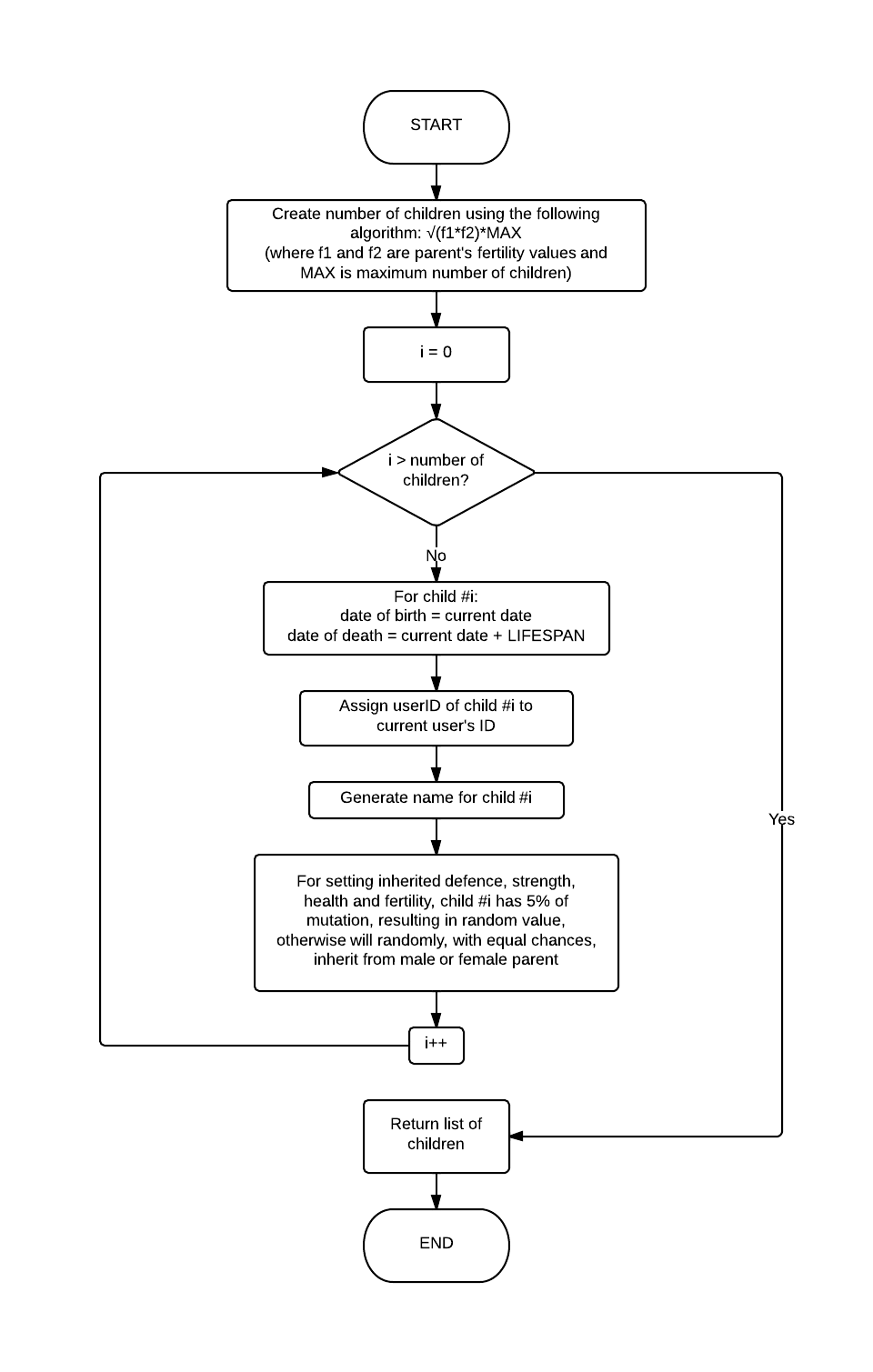
**this.**currentHealth **-=** opponent**.**currentStrength **\*** **(**1**-**

**this.**currentDefence**)** **\*** random**;**

**}**

**return** opponent**.**currentHealth**;**

### Breeding



public Monster**[]** breeding**(**Monster other**)** **{**

Random r **=** **new** Random**();**

int numberofchildren **=** **(**int**)** **(**Math**.**sqrt**(**fertility **\*** other**.**fertility**)** **\*** MAX\_CHILDREN**);**

Monster**[]** children **=** **new** Monster**[**numberofchildren **+** 1**];**

**for** **(**int i **=** 0**;** i**<=** numberofchildren**;** i**++){**

children**[**i**]=new** Monster**();**

children**[**i**].**id **=** "0"**;**

children**[**i**].**dob**=new** Date**();**

children**[**i**].**dod **=** **new** Date**(**children**[**i**].**dob**.**getTime**()+**LIFESPAN**);**

children**[**i**].**userID **=** **this.**userID**;**

children**[**i**].**name **=** NameGenerator**.**getName**();**

//generating inherited defense

**if(**r**.**nextInt**(**100**)<**5**){**

children**[**i**].**baseDefence**=**r**.**nextDouble**();**

**}** **else** **if(**r**.**nextInt**(**100**)<**50**){**

children**[**i**].**baseDefence**=**baseDefence**;**

**}** **else** **{**

children**[**i**].**baseDefence**=**other**.**baseDefence**;**

**}**

children**[**i**].**currentDefence **=** children**[**i**].**baseDefence**;**

//generating inherited strength

**if(**r**.**nextInt**(**100**)<**5**){**

children**[**i**].**baseStrength**=**r**.**nextDouble**();**

**}**

**else** **if(**r**.**nextInt**(**100**)<**50**){**

children**[**i**].**baseStrength**=**baseStrength**;**

**}** **else** **{**

children**[**i**].**baseStrength**=**other**.**baseStrength**;**

**}**

children**[**i**].**currentStrength **=** children**[**i**].**baseStrength**;**

//generating inherited health

**if(**r**.**nextInt**(**100**)<**5**){**

children**[**i**].**baseHealth**=**r**.**nextDouble**();**

**}else** **if(**r**.**nextInt**(**100**)<**50**){**

children**[**i**].**baseHealth**=**baseHealth**;**

**}** **else** **{**

children**[**i**].**baseHealth**=**other**.**baseHealth**;**

**}**

children**[**i**].**currentHealth **=** children**[**i**].**baseHealth**;**

//generating inherited fertility

**if(**r**.**nextInt**(**100**)<**5**){**

children**[**i**].**fertility**=**r**.**nextFloat**();**

**}** **else** **if(**r**.**nextInt**(**100**)<**50**){**

children**[**i**].**fertility**=**fertility**;**

**}** **else** **{**

children**[**i**].**fertility**=**other**.**fertility**;**

**}**

**}**

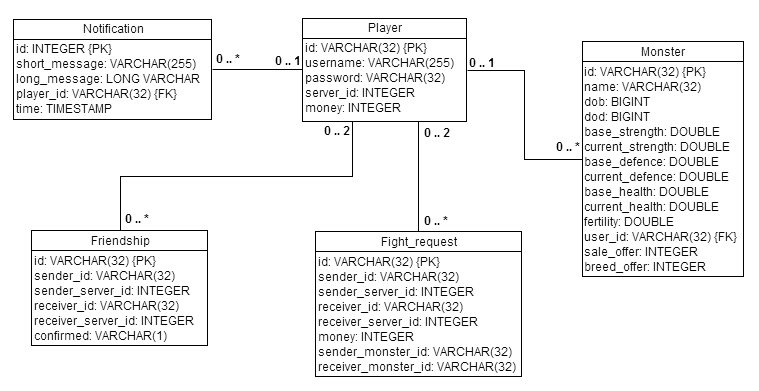
**return** children**;**

**}**

## Significant Data structures

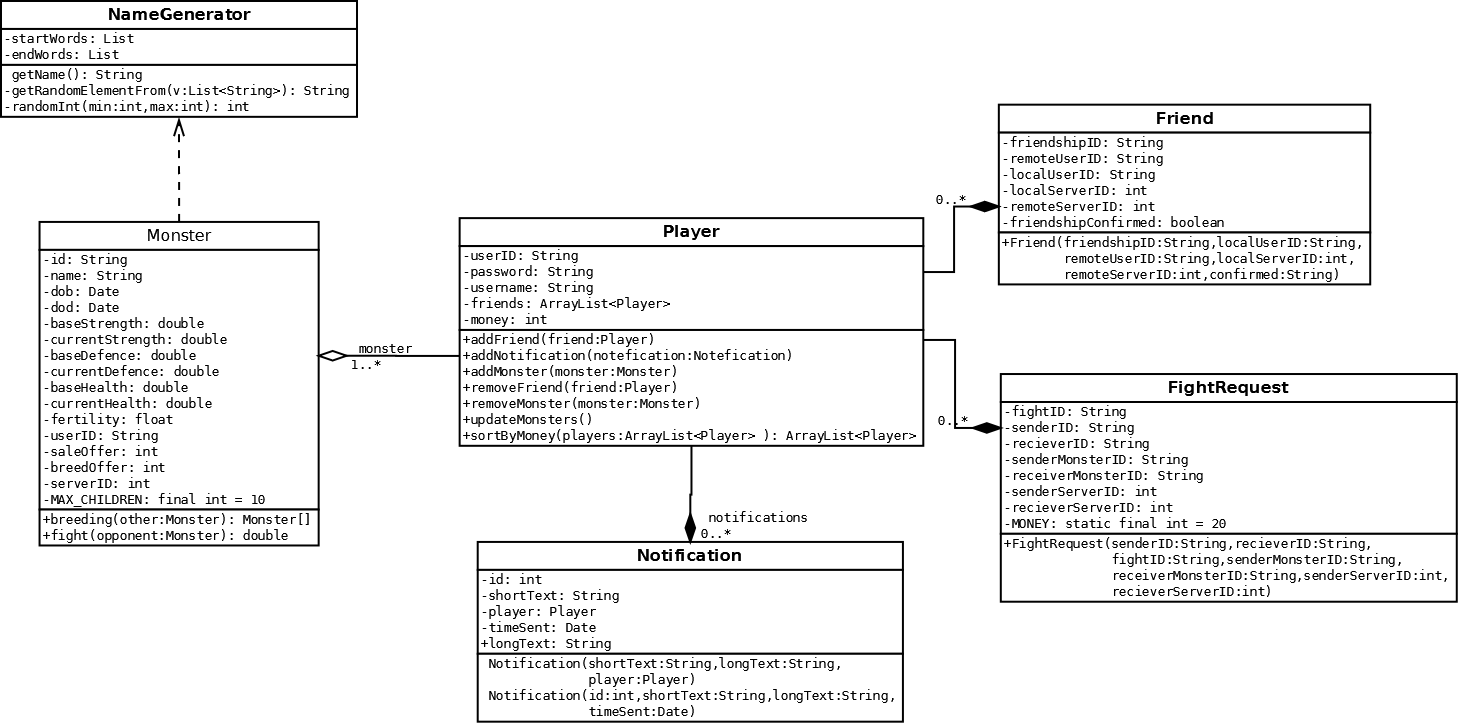
### Database Description

The database is used to store all data, so that data is not lost when the server is down. The instance variables of the java objects are saved so that they can re-instantiated. In the diagram below the *Fight\_request* and *Friendship* tables do not have ForeignKeys for monster and player because not all players and monsters are stored in our database.

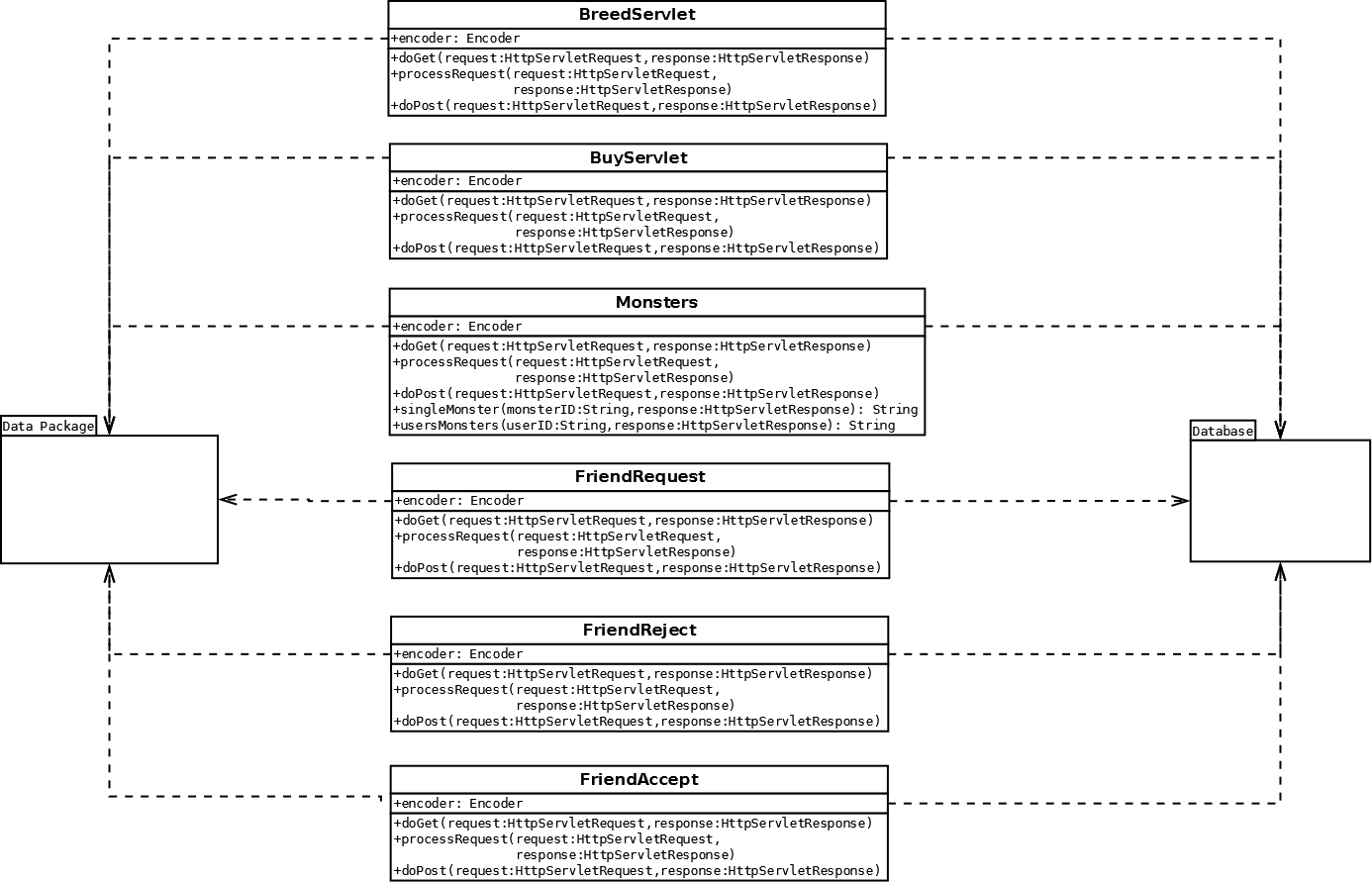


## Class Diagrams

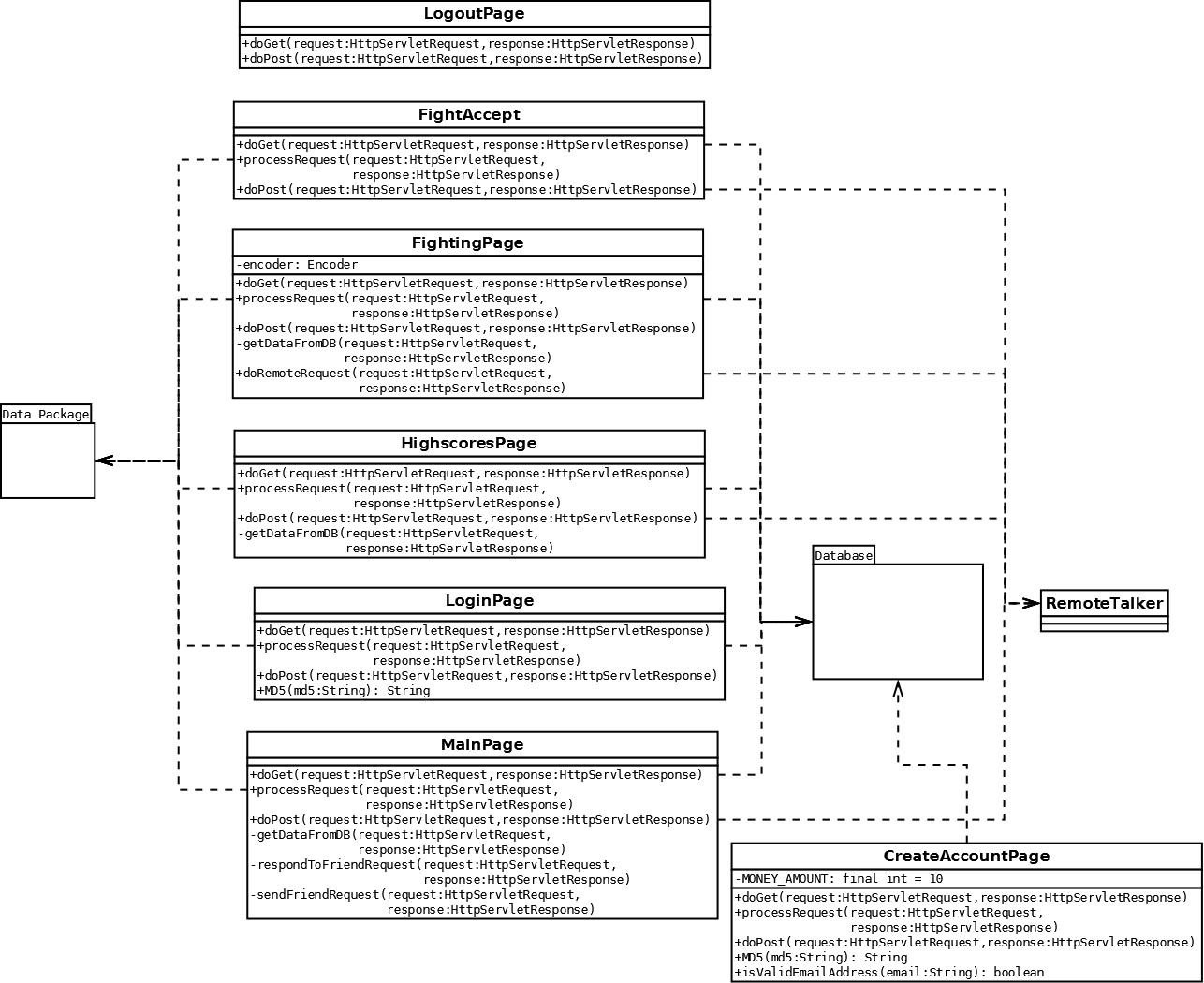
### Data Classes Package



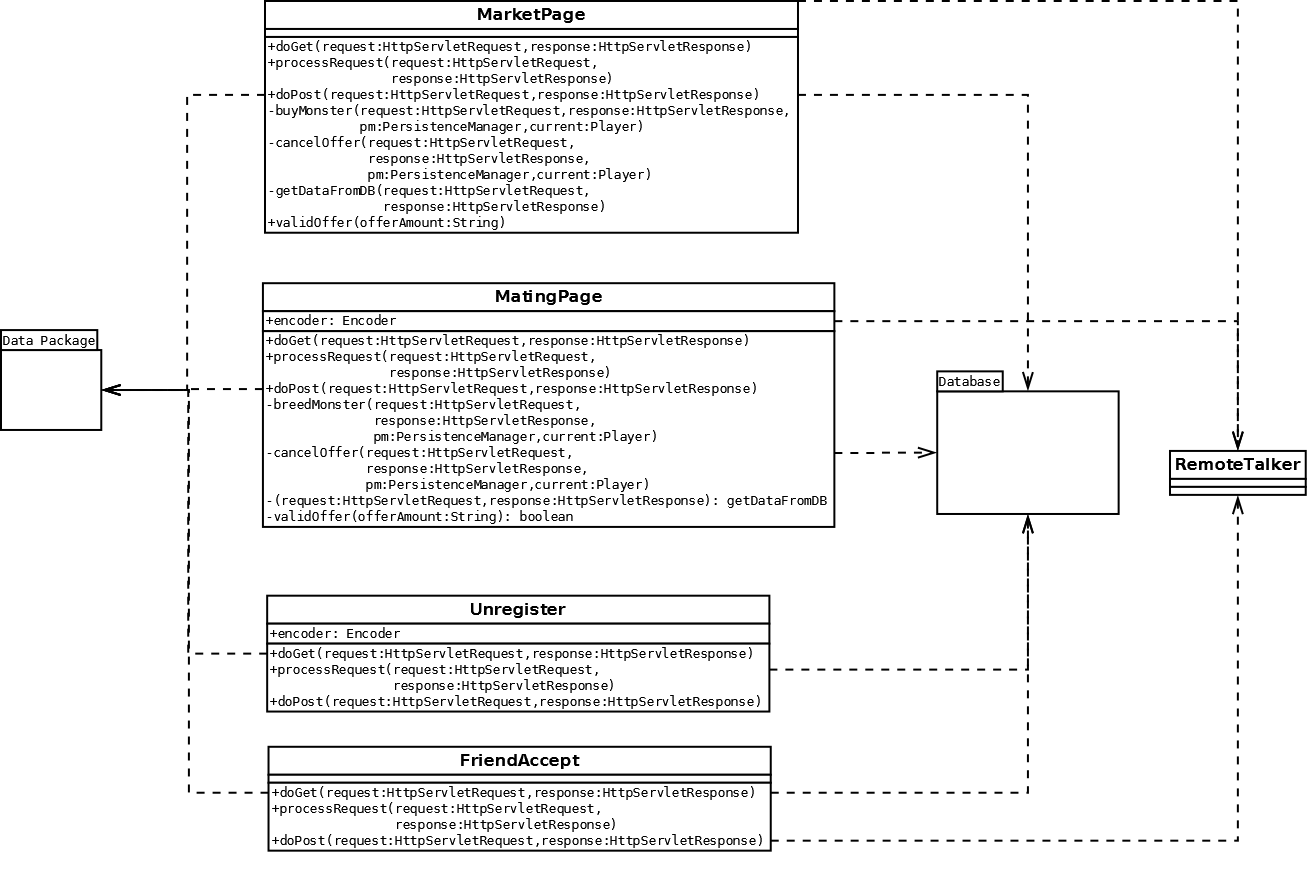
### Servlet Class Packages



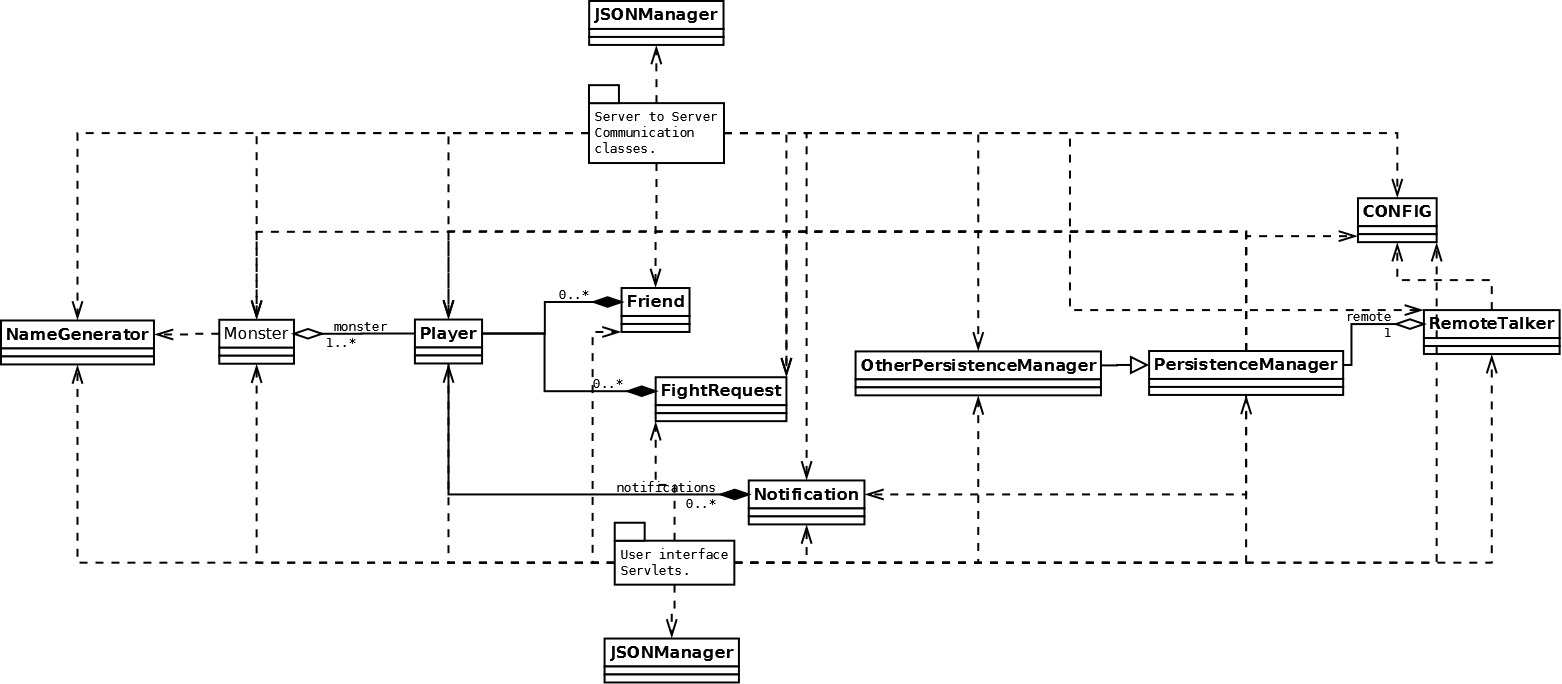
### Server Communications 1



### Server Communications 2



### Overall Class Diagram



REFERENCES

[1] **Software Engineering Group Projects Monster Mash Game Requirements Specification**

Config Ref: SE.CS.RS

[2] **Software Engineering Group Projects Design Specification Standards**

Config Ref: SE.QA.05A

[3] **Software Engineering Group Projects Java Coding Standards**

Config Ref**:** SE.QA.09

DOCUMENT HISTORY

| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 1.0 | N/A | 05/12/12 | Creation of document, all major elements added | G12 |
| 1.1 | N/A | 06/12/12 | Overhaul of many documentation and format elements | jau1 |
| 1.2 | N/A | 15/02/13 | Refactoring of many elements so they conform to the changes made to the design | G12 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |