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

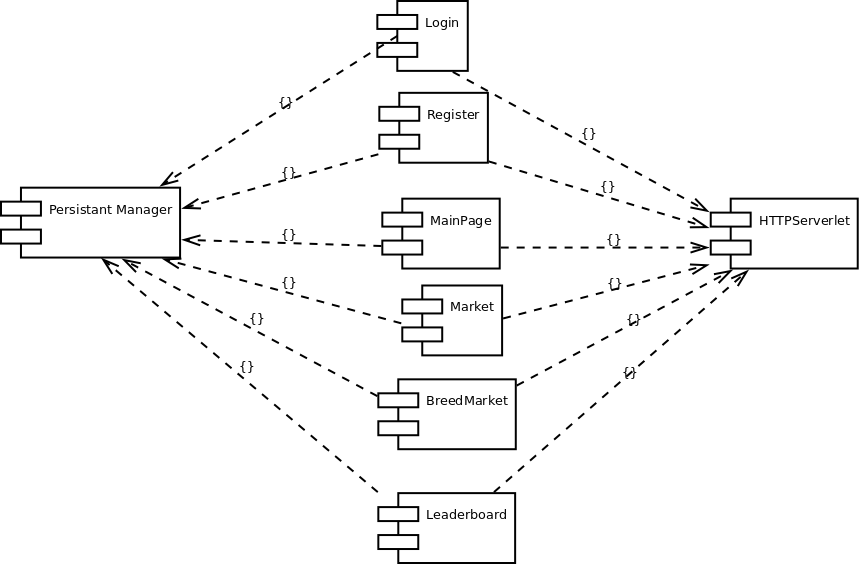
|  |  |
| --- | --- |
| Functional Requirement | Classes Providing Requirement |
| FR1 | Register, Player, Login, PersistenceManager |
| FR2 | Player, PersistenceManager |
| FR3 | Monster, Player, PersistenceManager |
| FR4 | Monster, Player, FightRequest, PersistenceManager |
| FR5 |  |
| FR6 | Register, 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

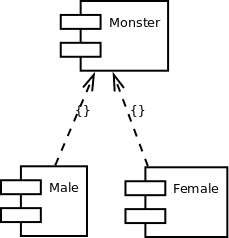
### Client

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].

### BreedingOffer

**import** Player**;**

**import** Male**;**

public class BreedingOffer

**{**

/\*\* Attributes \*/

private int id**;**

private Player player**;**

private Male monster**;**

private int moneyCost**;**

private Date offerStartTime**;**

**}**

### Female

**import** Male**;**

**import** Monster**;**

public class Female **extends** Monster

**{**

/\*\* Attributes \*/

private final int MAX\_CHILDREN**;**

/\*\*

\* Operation

\*

\* **@param** monster

\* **@return** Monster[]

\*/

public Monster**[]** breeding **(** Male monster **)**

**{**

**}**

**}**

### FightRequest

**import** Player**;**

**import** Male**;**

public class FightRequest

**{**

/\*\* Attributes \*/

private int id**;**

private Player sender**;**

private Player reciver**;**

private int moneyOffer**;**

private Male monster**;**

private Male opponentMonster**;**

private Date offerSentTime**;**

private String figthKey**;**

**}**

### FriendRequest

**import** Player**;**

public class FriendRequest

**{**

/\*\* Attributes \*/

private int id**;**

private Player sender**;**

private Player reciver**;**

private Date offerSentTime**;**

private String localKey**;**

private String remoteKey**;**

**}**

### Male

**import** Male**;**

**import** Monster**;**

**import** BreedingOffer**;**

**import** FightRequest**;**

public class Male **extends** Monster

**{**

/\*\* Attributes \*/

private final int MAX\_RANGE**;**

private boolean injured**;**

/\*\*

\* Operation

\*

\* **@param** opponent

\* **@return** boolean

\*/

public boolean fight **(** Male opponent **)**

**{**

**}**

**}**

### MarketOffer

**import** Player**;**

**import** Monster**;**

public class MarketOffer

**{**

/\*\* Attributes \*/

private Player seller**;**

private Monster monster**;**

private Date offerSentTime**;**

private int id**;**

private int money**;**

**}**

### Monster

**import** MarketOffer**;**

public abstract class Monster

**{**

/\*\* Attributes \*/

protected int id**;**

protected String name**;**

protected Date dob**;**

protected float genetic\_strength**;**

protected float speed**;**

protected float accuracy**;**

protected float endurance**;**

protected float armor**;**

protected float dodge**;**

protected float age\_rate**;**

protected float fertility**;**

protected float health**;**

protected float strength**;**

**}**

### Notification

**import** Player**;**

public class Notification

**{**

/\*\* Attributes \*/

private int id**;**

private String text**;**

private Player player**;**

private Date timeSent**;**

/\*\* Associations \*/

private Player unnamed**;**

**}**

### Player

**import** Player**;**

**import** Monster**;**

**import** FriendRequest**;**

**import** FightRequest**;**

**import** MarketOffer**;**

**import** BreedingOffer**;**

public class Player

**{**

/\*\* Attributes \*/

private int id**;**

private String password**;**

private String email**;**

private ArrayList**<**Player**>** friends**;**

private int money**;**

/\*\* Associations \*/

private Monster unnamed**;**

/\*\*

\* Operation

\*

\* **@return**

\*/

public createInitialMonster **(** **)**

**{**

//

**}**

/\*\*

\* Operation

\*

\* **@param** friend

\* **@return**

\*/

public addFriend **(** Player friend **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** friend

\* **@return**

\*/

public removeFriend **(** Player friend **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** monster

\* **@return**

\*/

public addMonster **(** Monster monster **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** monster

\* **@return**

\*/

public removeMonster **(** Monster monster **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** notefication

\* **@return**

\*/

public addNotification **(** Notefication notefication **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@return**

\*/

public updateMonsters **(** **)**

**{**

**}**

**}**

### BreedingMarket

**import** HttpServlet**;**

public class BreedingMarket **extends** HttpServlet

**{**

/\*\*

\* Operation

\*

\* **@param** request

\* **@param** response

\* **@return**

\*/

public processBreedRequest **(** HttpServletRequest request**,** HttpServletResponse response **)**

**{**

**}**

**}**

### Leaderboard

**import** HttpServlet**;**

public class Leadboard **extends** HttpServlet

**{**

/\*\*

\* Operation

\*

\* **@param** request

\* **@param** response

\* **@return**

\*/

public processRequest **(** HttpServletRequest request**,** HttpServletResponse response **)**

**{**

**}**

**}**

### Login

**mport** HttpServlet**;**

public class Login **extends** HttpServlet

**{**

/\*\*

\* Operation

\*

\* **@param** request

\* **@param** response

\* **@return**

\*/

public processLoginRequest **(** HttpServletRequest request**,** HttpServletResponse response **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** request

\* **@param** response

\* **@return**

\*/

public processLogoutRequest **(** HttpServletRequest request**,** HttpServletResponse response **)**

**{**

**}**

**}**

### Mainpage

iimport HttpServlet**;**

public class MainPage **extends** HttpServlet

**{**

/\*\*

\* Operation

\*

\* **@param** request

\* **@param** response

\* **@return**

\*/

public processFriendRequest **(** HttpServletRequest request**,** HttpServletResponse response **)**

**{**

**}**

**}**

### Market

**import** HttpServlet**;**

public class Market **extends** HttpServlet

**{**

/\*\*

\* Operation

\*

\* **@param** request

\* **@param** response

\* **@return**

\*/

public processSellRequest **(** HttpServletRequest request**,** HttpServletResponse response **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@return**

\*/

public processBuyRequest **(** **)**

**{**

**}**

**}**

### Persistencemanager

**import** Register**;**

**import** Login**;**

**import** Market**;**

**import** Leadboard**;**

**import** BreedingMarket**;**

**import** MainPage**;**

public class PersistenceManager

**{**

/\*\* Attributes \*/

private final String dbName**;**

private final String dbHost**;**

private final String dbPort**;**

private final String dbUsername**;**

private final String dbPassword**;**

private Connection connection**;**

private String error**;**

/\*\*

\* Operation

\*

\* **@param** query

\* **@return** boolean

\*/

public boolean query **(** String query **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** query

\* **@return** String[]

\*/

public String**[]** select **(** String query **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@return** String

\*/

public String getError **(** **)**

**{**

**}**

**}**

### Register

**import** Player**;**

**import** HttpServlet**;**

public class Register **extends** HttpServlet

**{**

/\*\*

\* Operation

\*

\* **@param** username

\* **@param** email

\* **@param** password

\* **@return** boolean

\*/

public boolean createPlayer **(** String username**,** String email**,** String password **)**

**{**

**}**

/\*\*

\* Operation

\*

\* **@param** player

\* **@return**

\*/

public unregister **(** Player player **)**

**{**

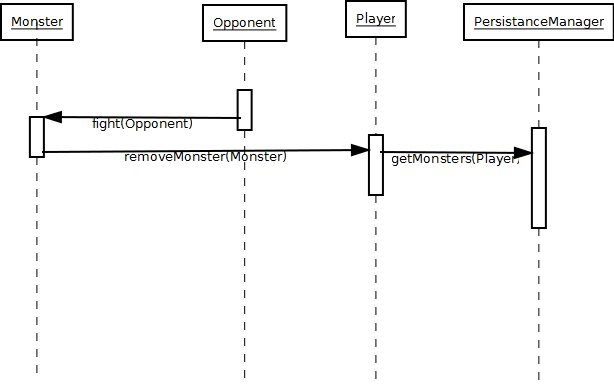
**}**

# Detailed Designs

## Sequence Diagrams

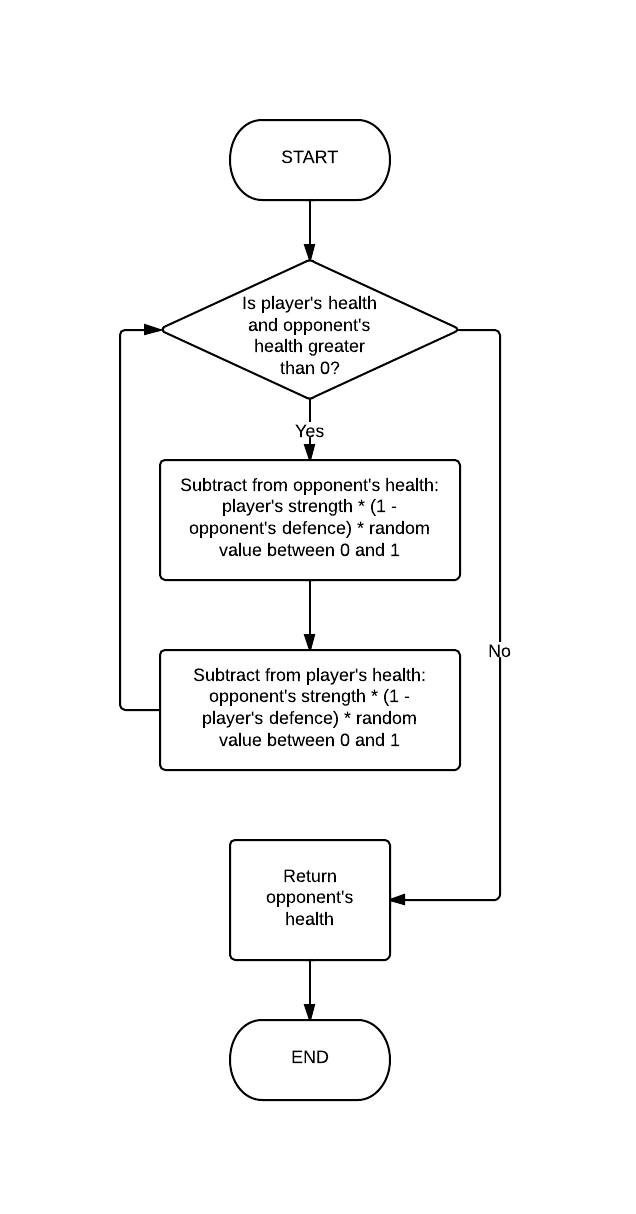
### Fighting

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**;**

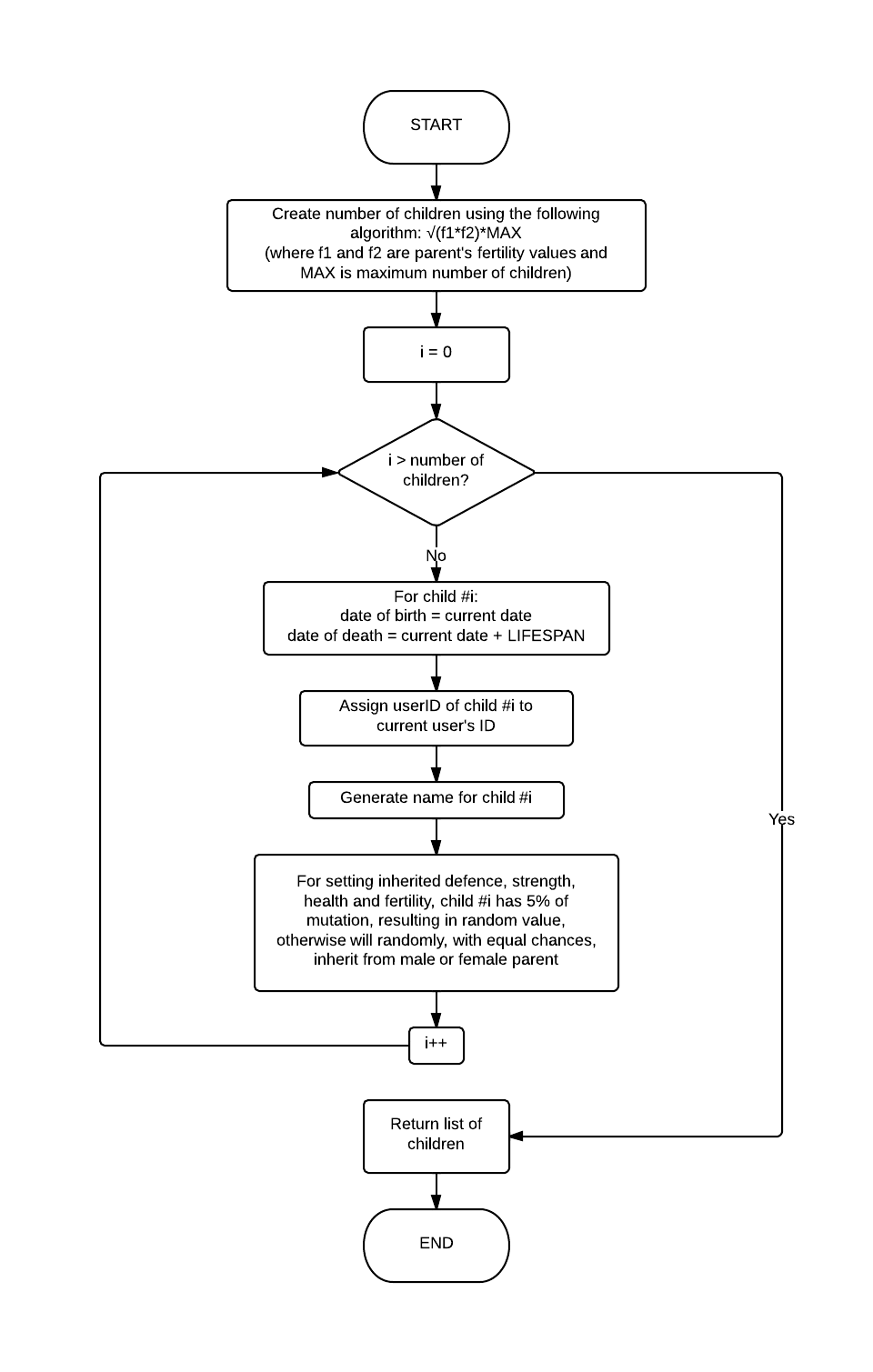
**this.**currentHealth **-=** opponent**.**currentStrength **\*** **(**1**-**

**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**;**

**}**

**}**

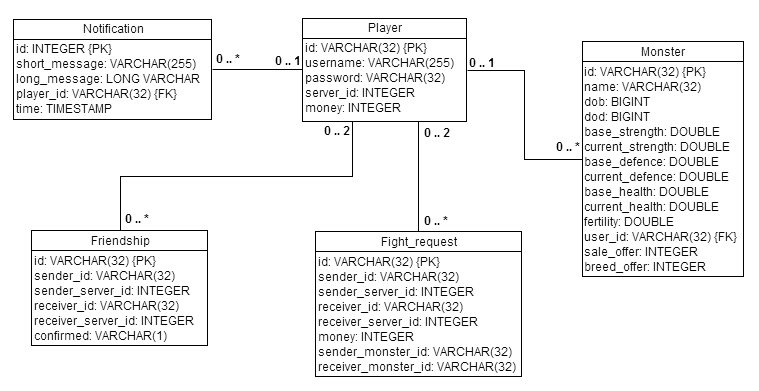
**return** children**;**

**}**

## Significant Data structures

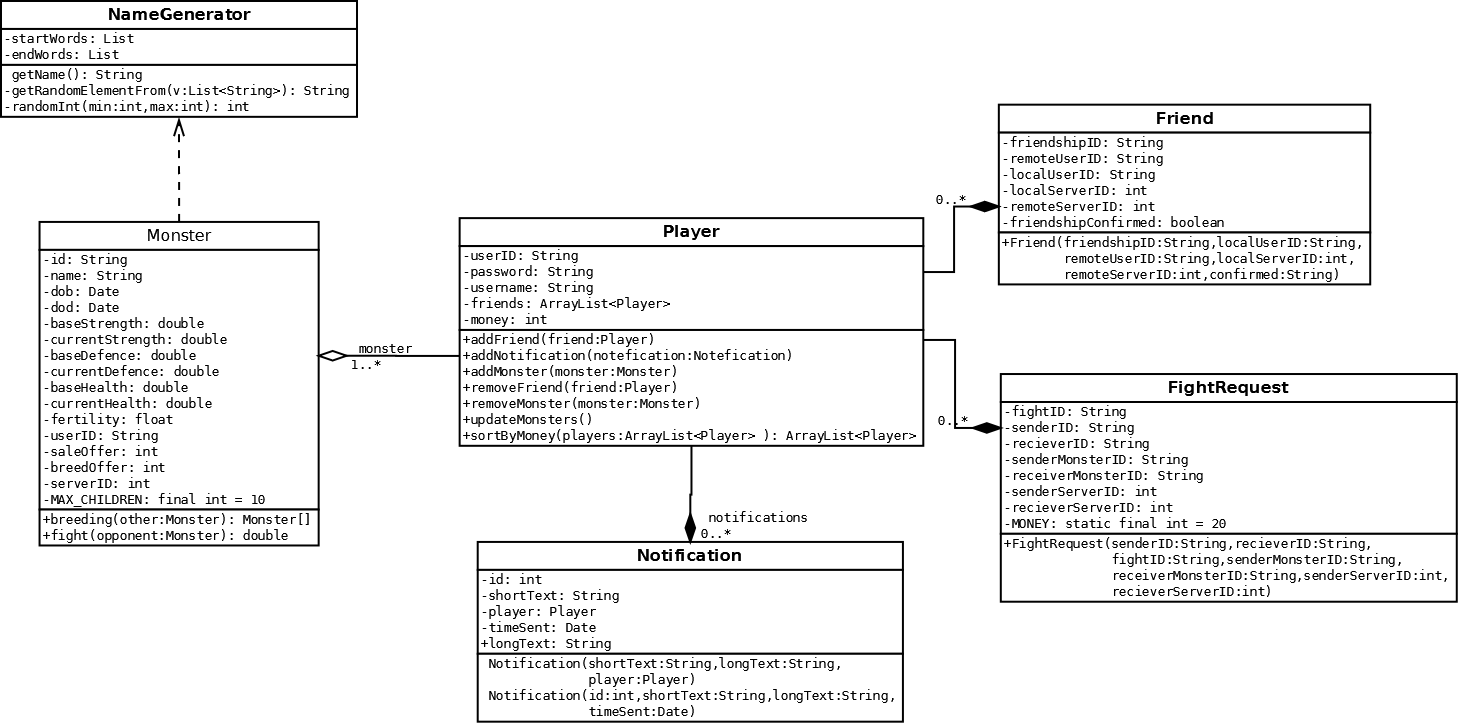
### Database Description

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.

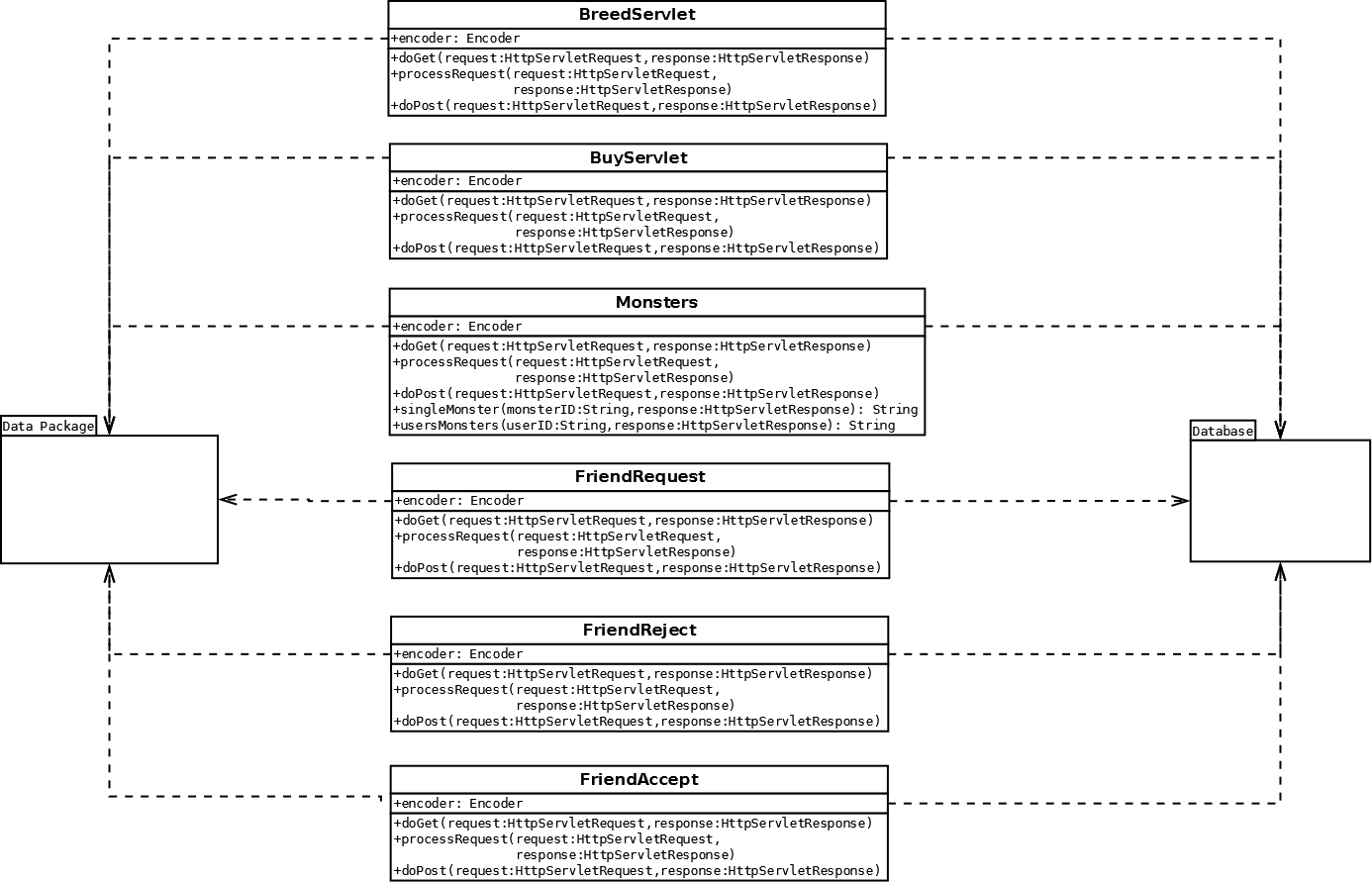


## Class Diagrams

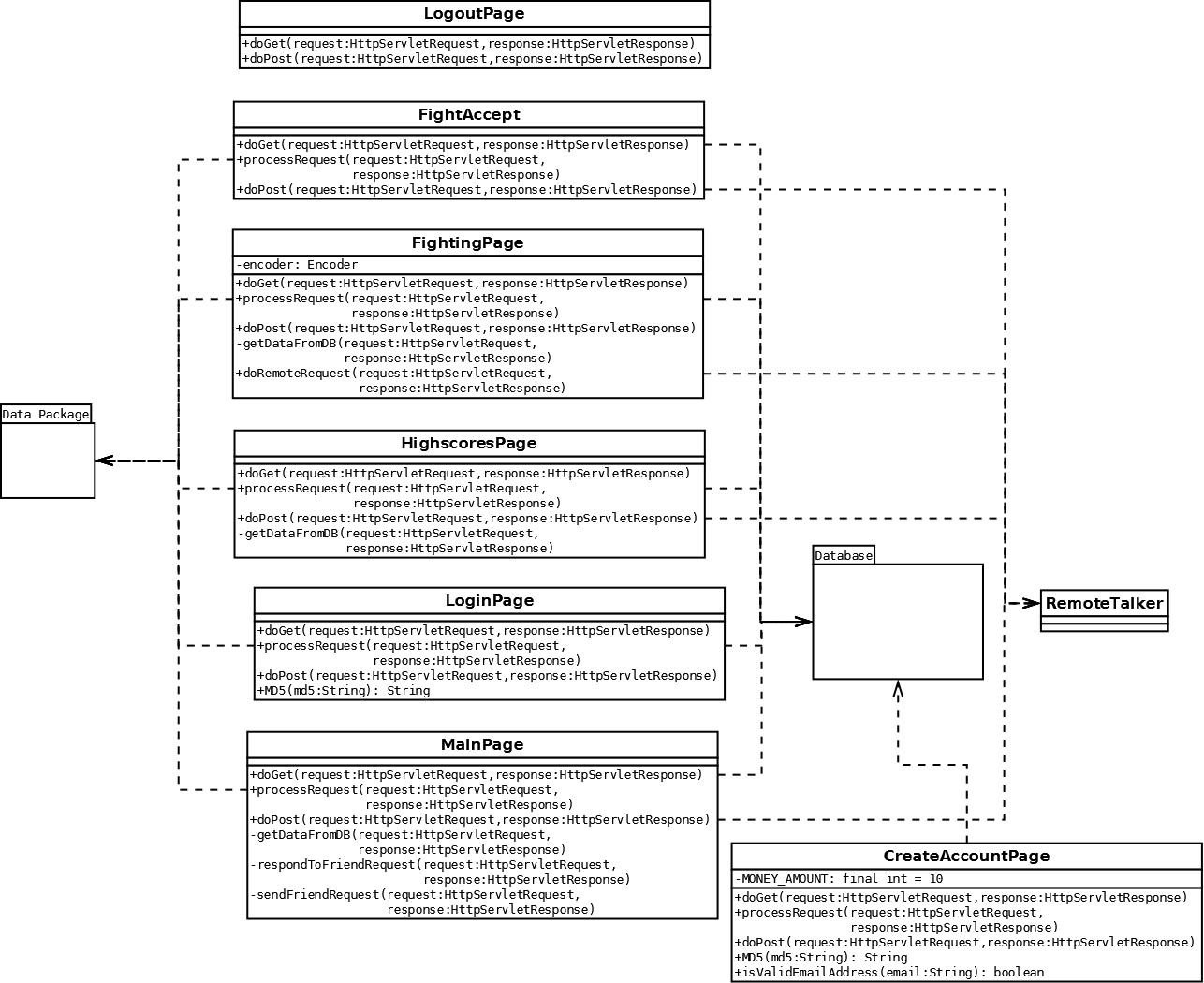
### Data Classes Package



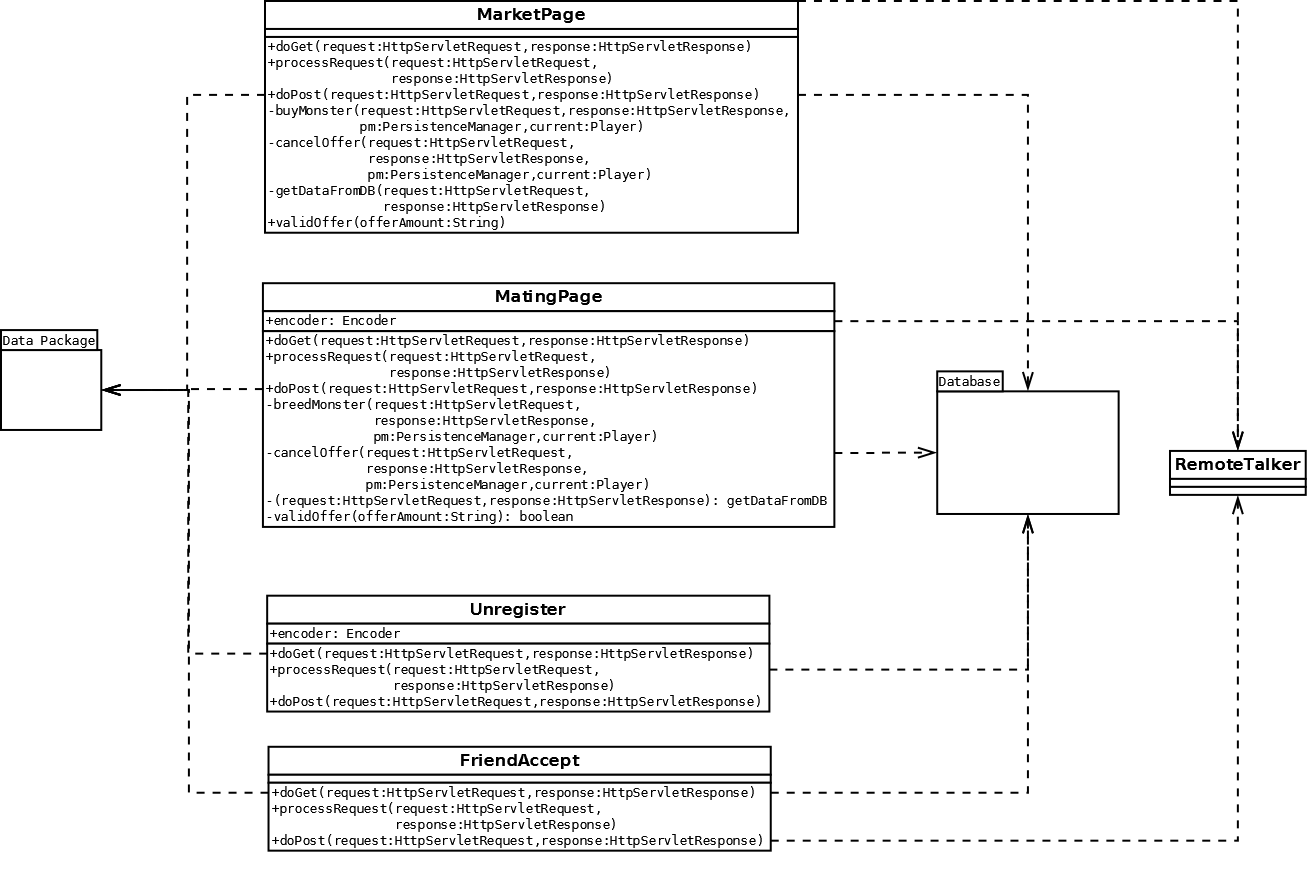
### Servlet Class Packages



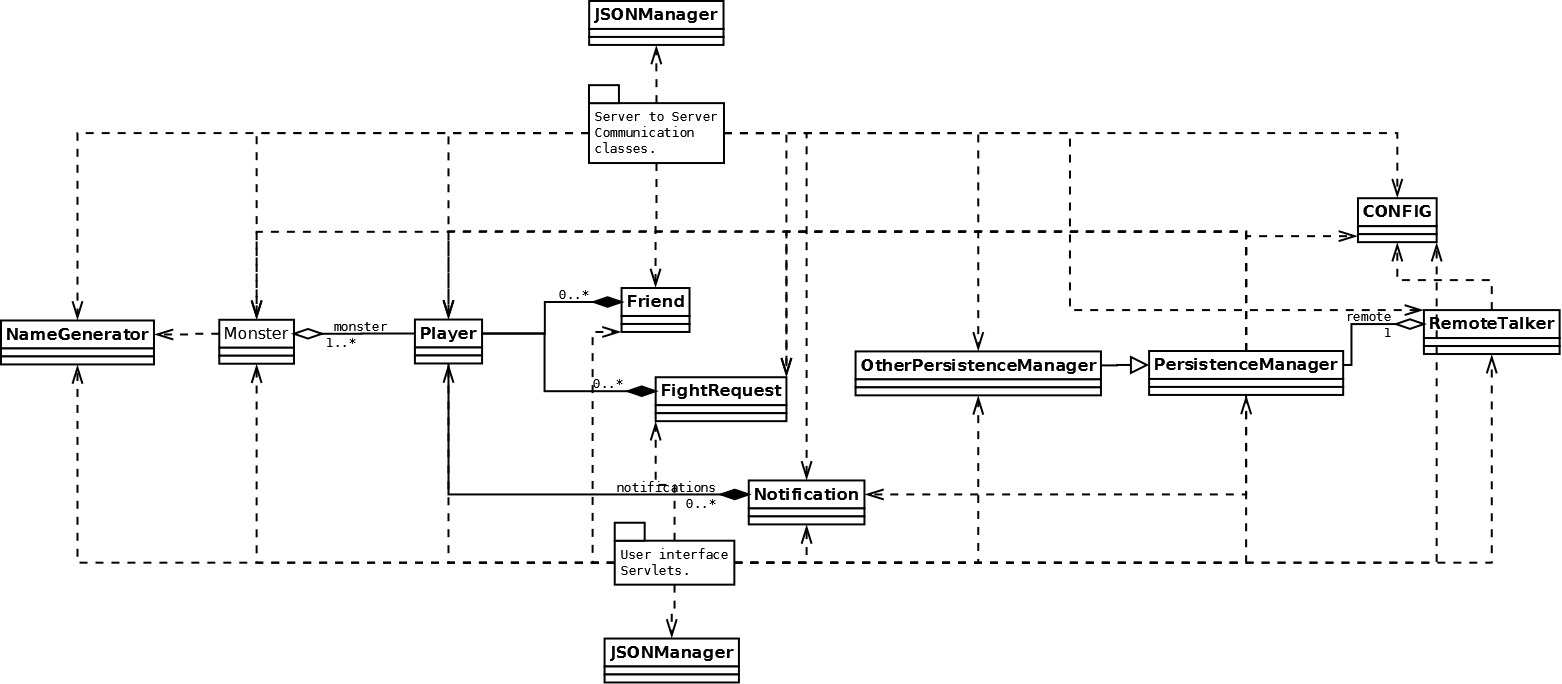
### Server Communications 1



### 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 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |