1. Introduction

In this phase of our project our main goal is to define our software objects and model both visually and textually how they collaborate to fulfill the requirements that were mentioned during our analysis. In this phase we will construct a conceptual solution to our game's requirements, so that this report will play a key role in the implementation phase of our project.

1.1 Purpose of the system

System is a text based game and designed for entertaintment. In the game player has a character and by using command prompt player commands his character to perform actions like moving around, killing enemies, collecting items, inspecting surroundings etc. Aim of this game is exploring the world while keeping his character alive.

1.2 Design Goals

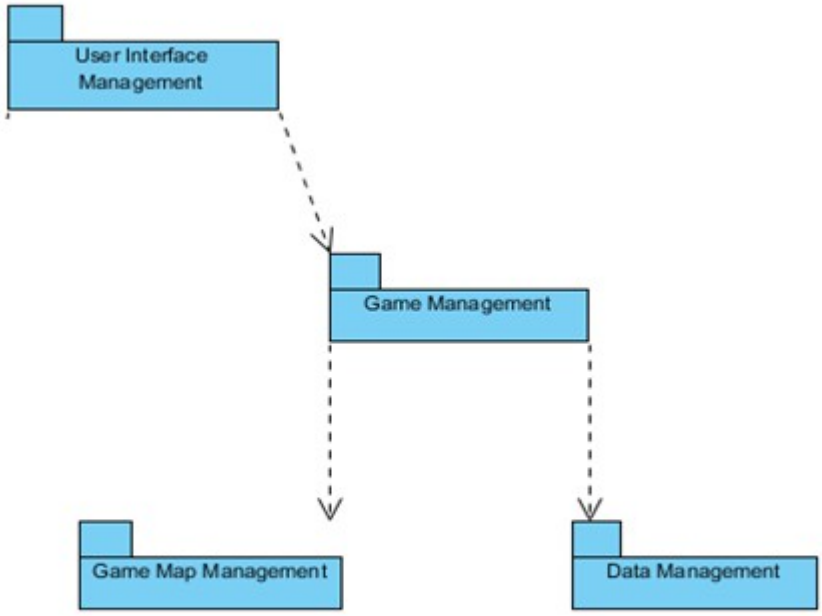
- Allow user to complete their task without being distracted by software or losing train of thought
- After user types commands system responses in less than one second
- Allow user to run the system on all types of operation systems
- Give users access to information they need to complete their task (i.e. information on other pages, etc.).
- Add clear tutorial about game to the help menu to make system easily understandable
- System will not interrupt the game play because of the small typos of the player
- System can be modified without changing entire structure of it

2. Software Architecture

We design our system in Model- View- Controller Architectural Style. It is the best convenient way for us.

2.1 Subsystem Decomposition

We decided to handle our games in 5 subsystems which connect themselves with the interactions shown in the figure. Details about subsystems and their classes are given in the subsystem services part.



2.2 Hardware / Software Mapping

In our project Java will be used as a programming language. Users who has enough tool to execute Java projects will be able to execute our program. Only software requirement for pur system is that.

When issue comes to hardware requirements, a regular keyboard is enough. Keyboard will be used to initialize the game, enter login information and commands to direct the game.

As a result, our project is not required an extra tool than that already exist in all computers.

2.3 Persistent Data Management

User information and save files will be saved as .DAT files to the harddrive. Interior of these files will be decided during implementation.

2.4 Access Control and Security

Upon creation of an account, login credentials of that account will be stored to the harddrive. When an user wants to play the game players must type in their login ID's and passwords. If login is successfull their game data will be loaded and game will start. Otherwise they will get an error message explaining the situation. Users can access other players via a proxy and they only will be able to see their game condition and scores.

2.5 Boundary Conditions

Initializaton

Since Zoork does not have regular .exe or such extension, it will not require any installation. Game will be initialized via a .jar file.

Termination

Because of the neccessity of saving, game must be terminated with exit command otherwise system will not be able to save and store game data and will lose the process achieved since last save.

Error

In the case of the corruption of the data system will not be able to recover the data and all processes will be lost.

3.1. User Interface Management Subsystem

User Interface Management Subsystem includes a ConsoleView Class

ConsoleView Class

ConsoleView class is used to view all menus that can be reached by the main menu of the game. This class is the first class that is initialized when user opens the game. The UI that is created by this class includes a text based interface with following options “Play Game”, “Instructions”, “High Scores”, “Settings” and “Exit Game”. When “Play Game” option is choosed, further commands are asked from the used by providing a set of choices in a text based format. If “Instructions”, “High Scores” or “Settings” option is choosed, new text based interface is created by the class in order to show the information that is wanted by the user. ConsoleView class gets required information from GameSystem class. Last choice on the menu, “Exit Game”, if any progress made it saves it and finally exits the game.

3.3. Game Management Subsystem

Game Management Subsystem includes following classes.

GameSystem Class

GameSystem manages all aspects of the game. GameSystem supplies information to UI-related classes with the information it gets from the classes of the Attribute Management Subsystem and Data Management Subsystem, and controls the progress of the game and updates the game during game-play. Decision of whether the game is over or not is also made by this class.

Game Class

After user starts to play the game, game class becomes the messanger of the system. It helps to the objects to reach t othe classes in other subystems. For example object of the Reader class can call methods of the player object with the instance of Game object which is a property of Reader class(game.getPlayer().go(Direction.d) will be called from Reader according to Input)

Character Class

Character class is an abstract class which is expanded by Player class and NonPlayerCharacter class

NonPlayerCharacter class

NonPlayerCharacter class is an expanded by HostileCharacter class. It exist to make it possible to make friendly NPC's later on

HostileCharacter Class

Instances of HostileCharacter class tends to attack to player

Player Class

Player class is where all player stats and action methods exist.

3.4. Game Map Management Subsystem

Game Map Management Subsystem includes 3 classes which are Map, Location and LocationFactory class.

Map Class

Map class keeps a 2 dimensional Location array and if player hits the bounds it will extend it from every side. It has a LocationFactory which will generate locations the player moves.

Location Class

Location class keeps a list of Things as a property, which will be randomly generated by LocationFactory.

LocationFactory Class

LocationFactory class keeps a queue of Locations, and it will feed the map with those when player moves to an unvisited location. It will also generate new maps when idle to keep the queue non-empty.

3.6. User Management Subsystem

User Management Subsystem includes following classes.

UserManager Class

It handles User Login and data save operations

User Class

Object of this class is initialized when user login, it will carry information about user and this data will be saved before exitting

3.7. Map Objects Subsystem

Map Objects Subsystem includes following classes.

Direction Enumeration

It carries direction information. Used in many direction based methods across the system.

Thing Interface

An Interface that implemented by Item and Furniture Classes

Furniture Class

Furnitures that exists in locations, player cant take those but may interact with it

Item Class

Items that exist in locations, and inventories of characters. Can be taken and used accordingly by player. This includes weaponry and foods.

3.7. IO Subsystem

Reader Class

Reads and analyzes the typed user input, and calls player methods accordingly.

4.1 Object Design Trade-Offs

This section defines the different tradeoffs in object design and the need to make decisions about them. In the making of a text-based computer game, aspects such as memory usage, availability, durability and maintainability are important. Compromises need to be made from these in order to come up with the optimal design.

- Memory space vs. response time

Text-based games often need a good share of memory even though no graphics are involved. As more memory is required to store all the user's data, response time may deteriorate; this, however, should not happen. The management classes involved in sending and receiving data, in turn, take care of this problem by limiting data transfer to only when it is required.

- Buy vs. build

Usually, a build policy may be sought because of the simple design of most objects. The parser component, however, may be supplied from elsewhere, which will in fact save time. The compromises needed to be made here is between cost and time, and one component may be bought if the cost is not too high, and if it will save time.

- Platform dependence vs. flexibility

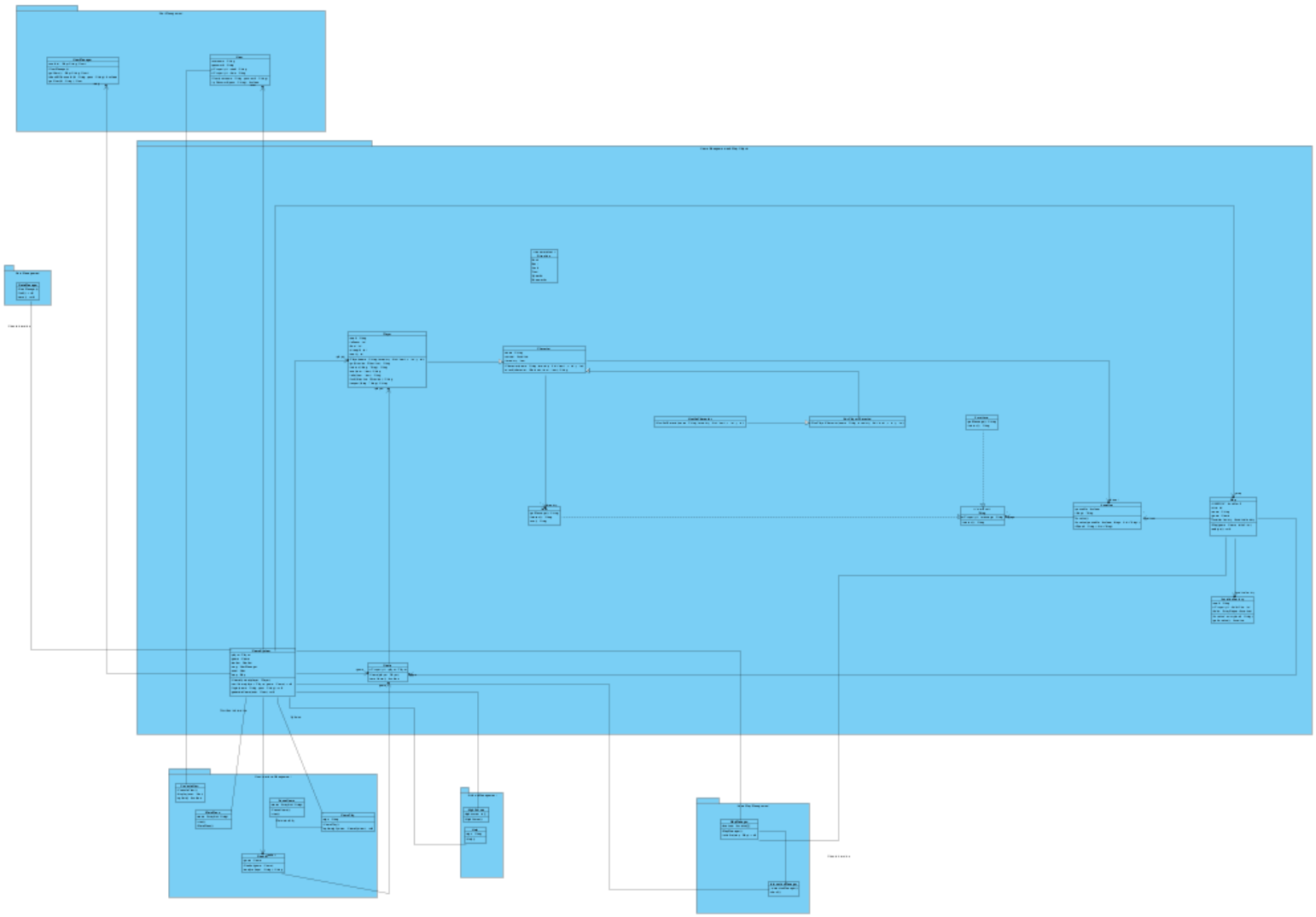
The game needs an optimal platform, such as the operating system and even architecture. This optimal platform will allow perfect programmability for the game, but it also needs to be flexible, such that only subtle changes to the design will grant it new platform dependency properties. Most games in the market are optimized for one platform, however since this is a text-based game, the switch between platforms is as simple as an integrated pattern.

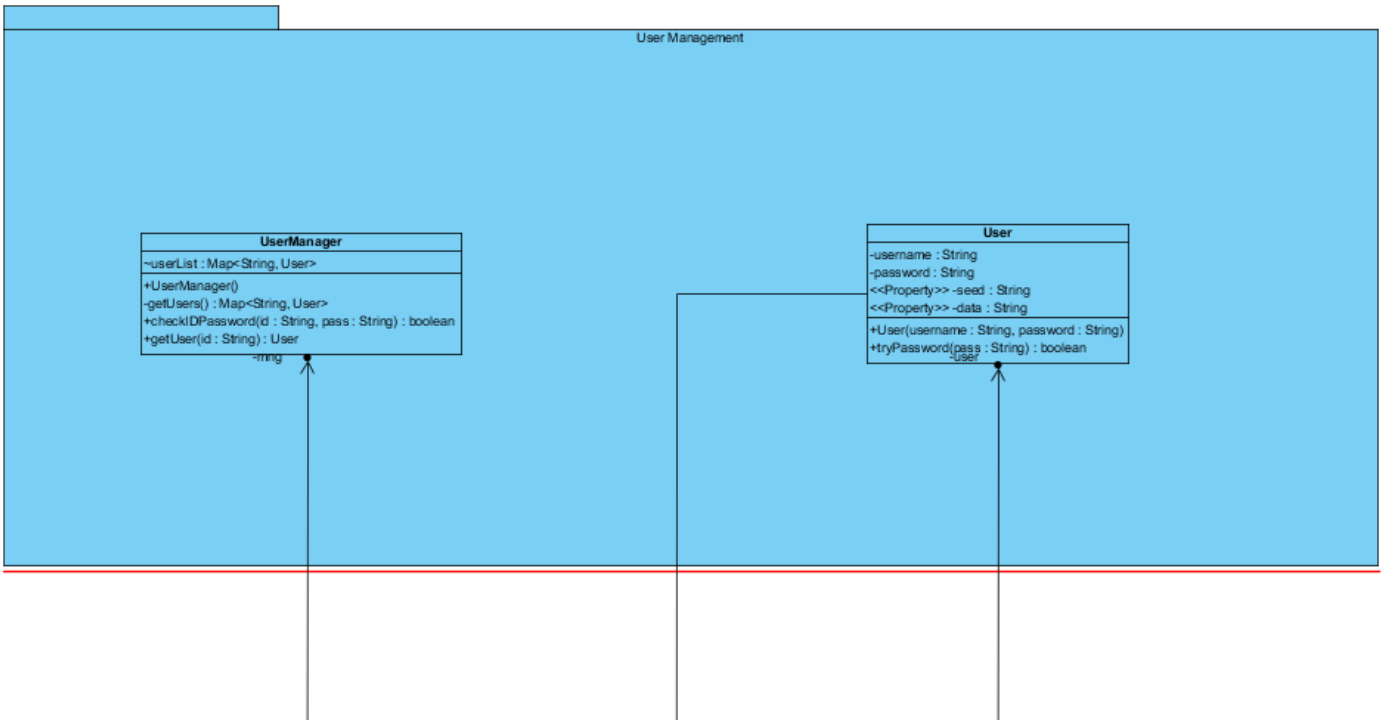
- Design patterns

Several design patterns have also been compared in terms of tradeoffs. The Bridge pattern was seen to provide more flexibility in terms of modifying existing code. Examples of this pattern can be seen in the figure provided in the following page. Since a general algorithm was not required for the game, a strategy pattern was not considered. In fact, several design patterns were used in different subsystems. These subsystems and the design patterns will be discussed in the upcoming section.

4.2 Final Object Design

As the final object design is too large to make every component legible, the document will display the object design as composed of its subsystems.




User Management
UserManager
~userList : Map<String, User>
+UserManager()
-getUsers() : Map<String, User>
+checkIDPassword(id : String, pass : String) : boolean
+getUser(id : String) : User
-ring
User
-username : String
-password : String
<<Property>> -seed : String
<<Property>> -data : String
+User(username : String, password : String)
+tryPassword(pass : String) : boolean
-user

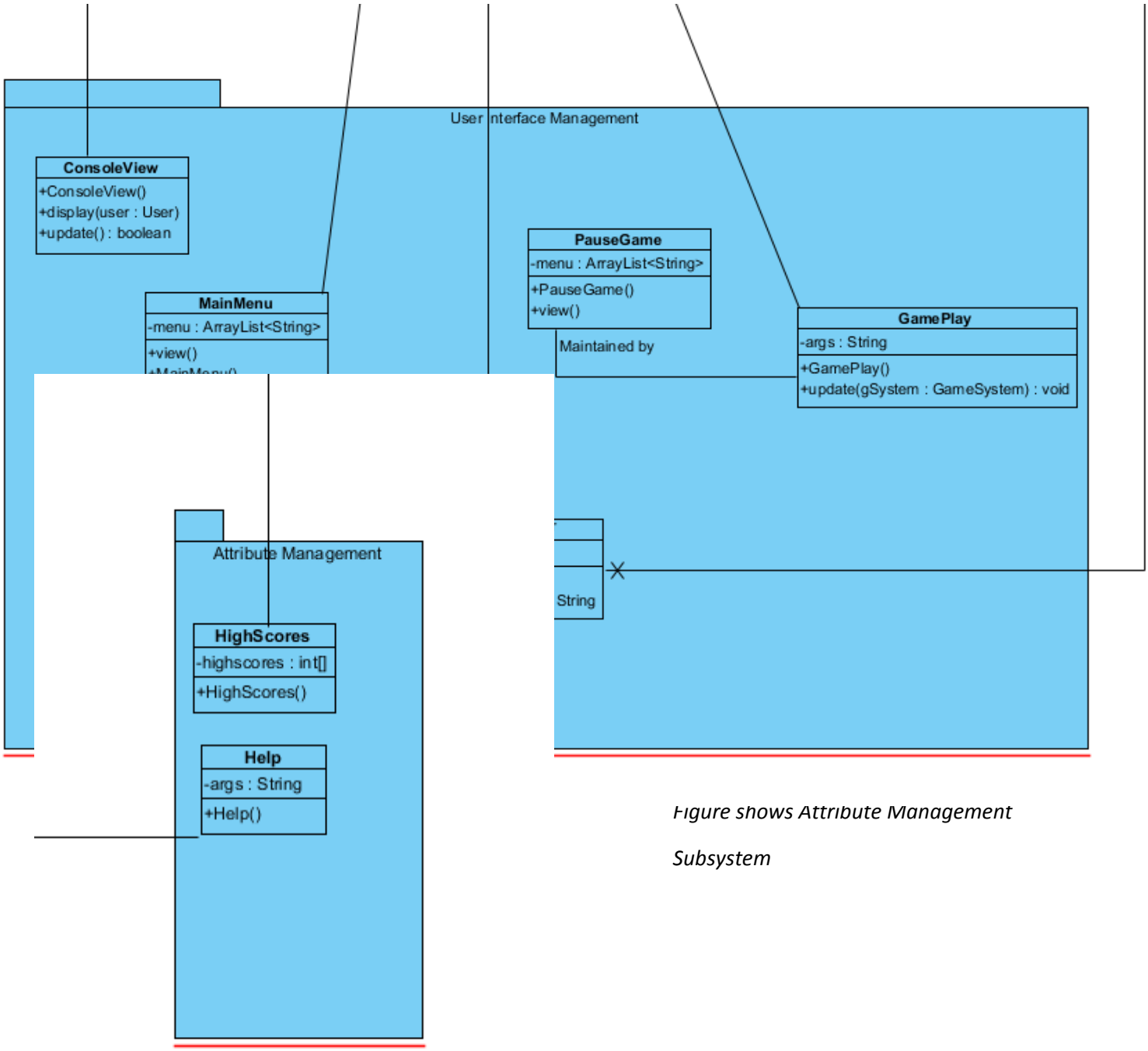


Figure shows Attribute Management Subsystem

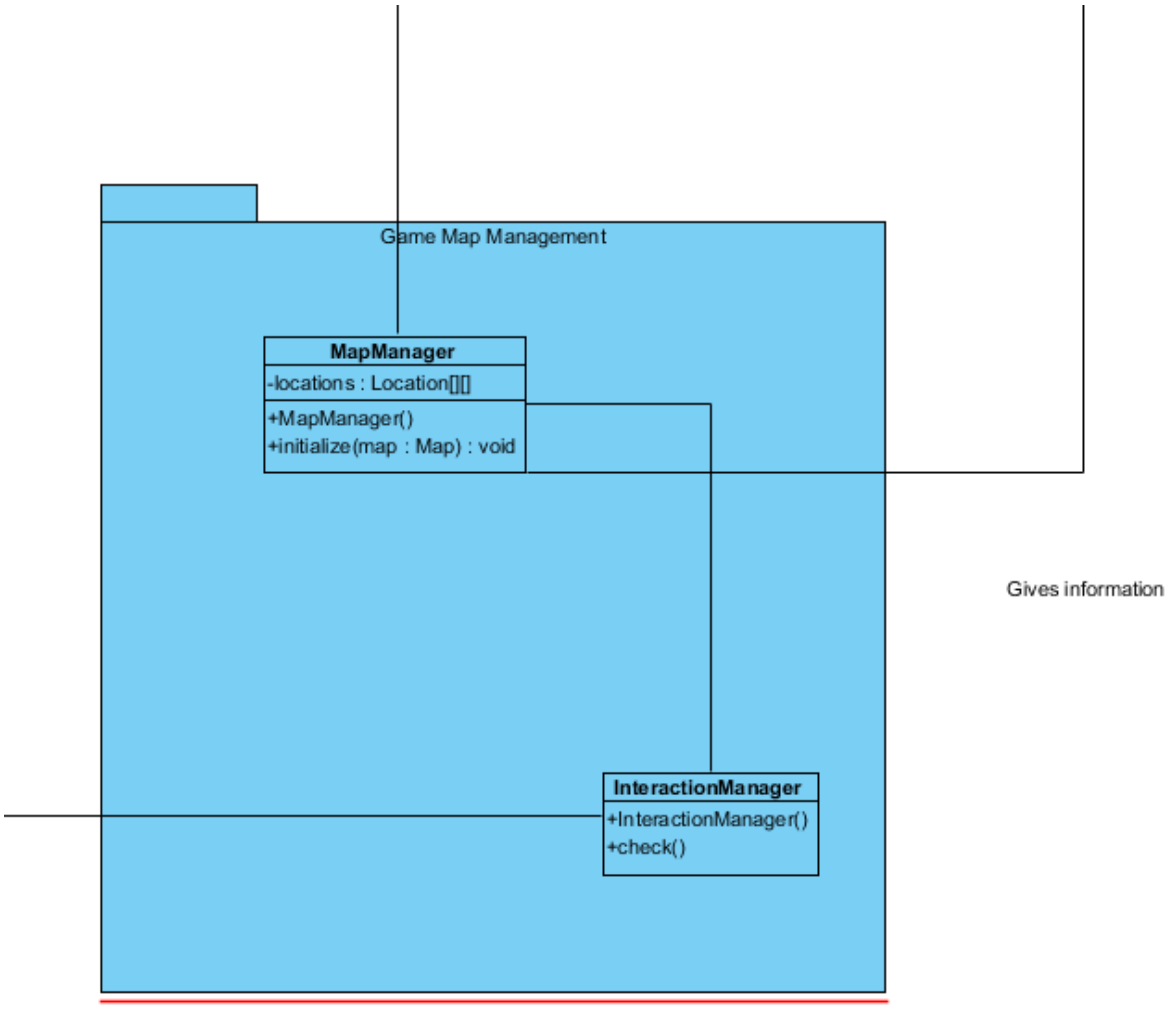
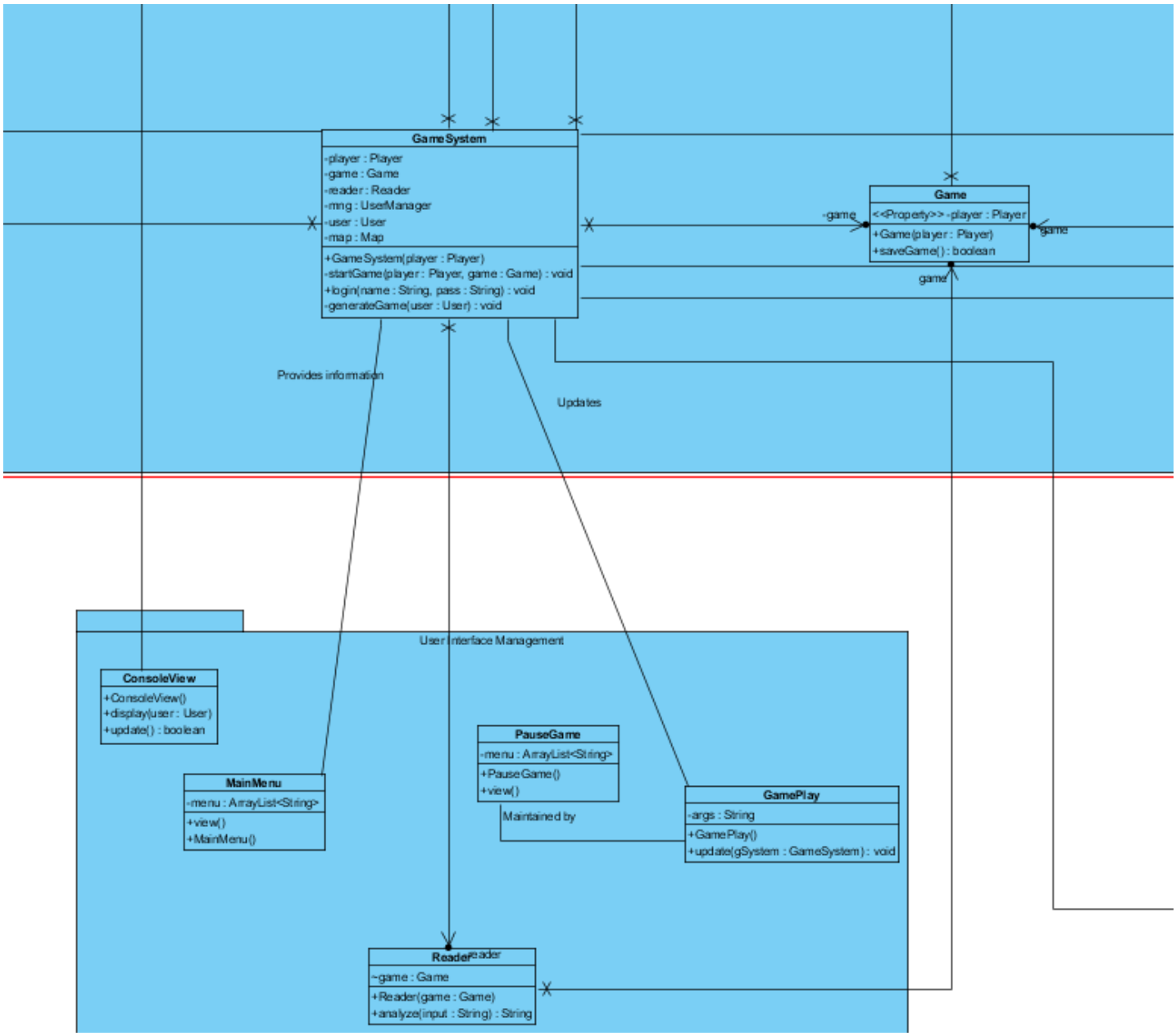


Figure shows Game Map Management Subsystem


GameSystem
-player : Player
-game : Game
-reader : Reader
-mng : UserManager
-user : User
-map : Map
+GameSystem(player : Player)
-startGame(player : Player, game : Game) : void
+login(name : String, pass : String) : void
-generateGame(user : User) : void
Game
<<Property>> -player : Player
+Game(player : Player)
+saveGame() : boolean
-game
game
game
Provides information
Updates
User Interface Management
ConsoleView
+ConsoleView()
+display(user : User)
+update() : boolean
MainMenu
-menu : ArrayList<String>
+view()
+MainMenu()
PauseGame
-menu : ArrayList<String>
+PauseGame()
+view()
Maintained by
GamePlay
-args : String
+GamePlay()
+update(gSystem : GameSystem) : void
Reader
Reader
~game : Game
+Reader(game : Game)
+analyze(input : String) : String

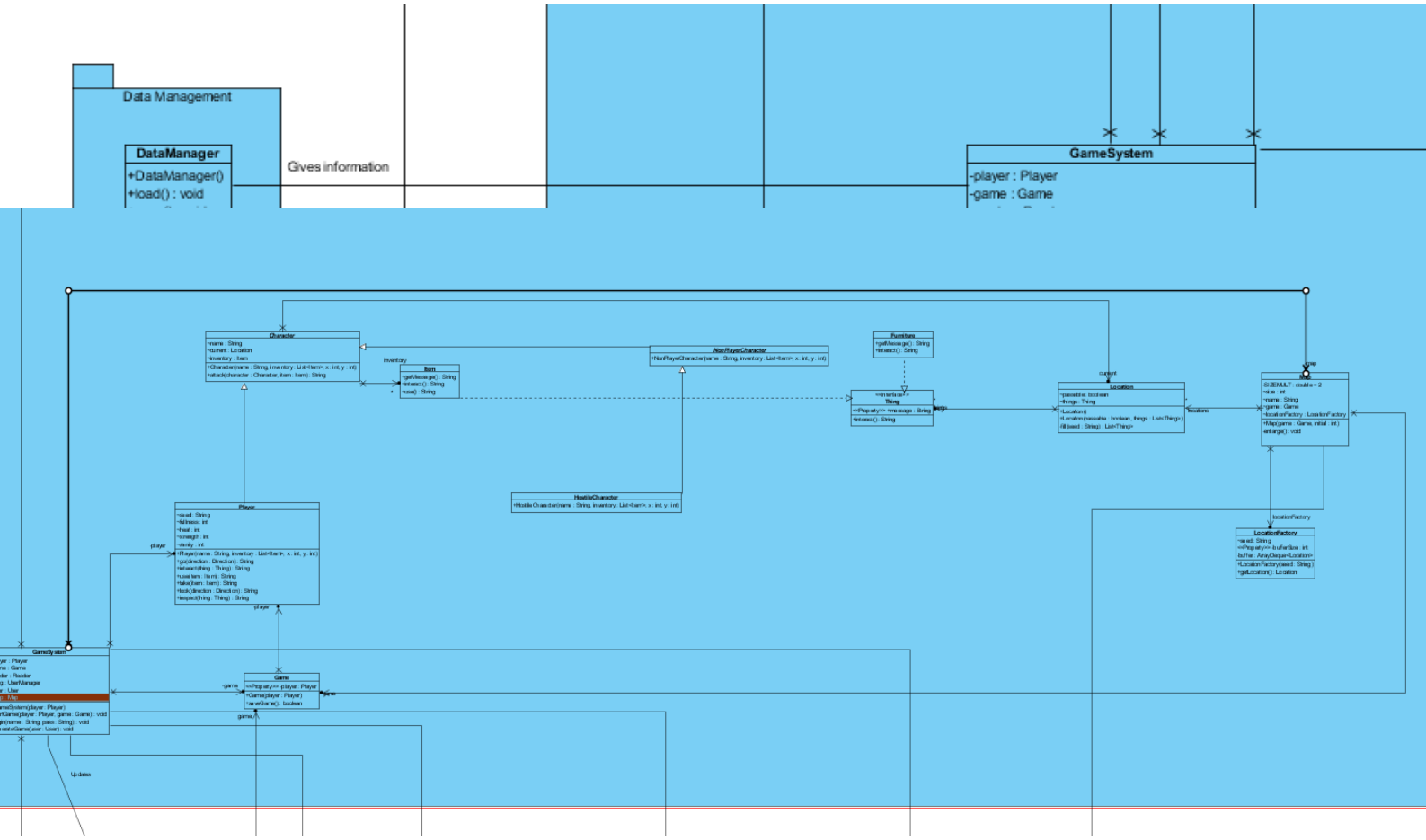


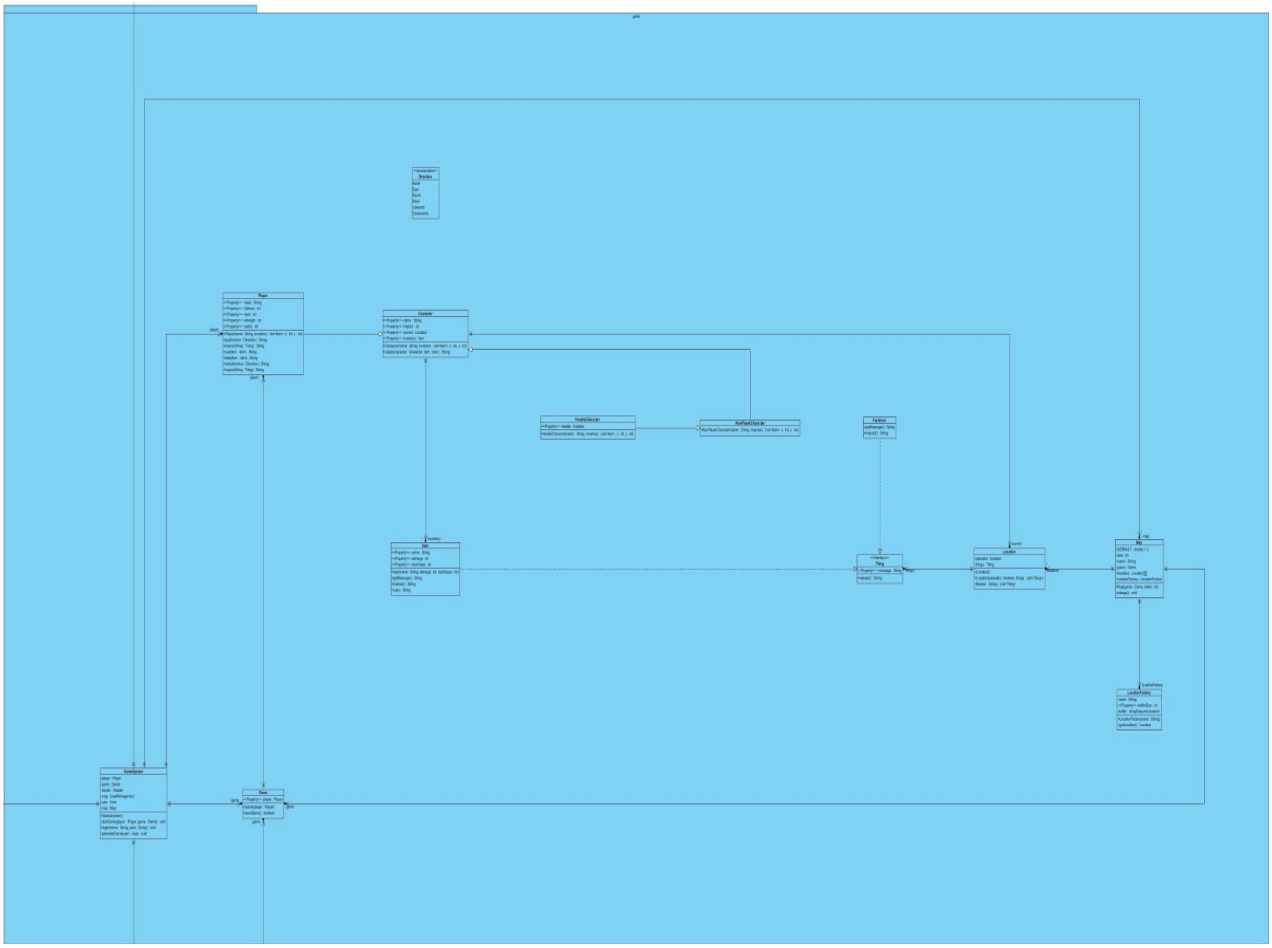
Figure shows the remaining objects and their interactions with GameSystem.

This part of the design was done choosing Abstract Factory design, hence the multiple inheritance relationships. The GameSystem class provides methods to classes like Map and Player. This design was superior to others in the sense that it allows compatibility; there is a single solution for implementing the methods of these classes. It also allows quick modifications.

4.3.Packages

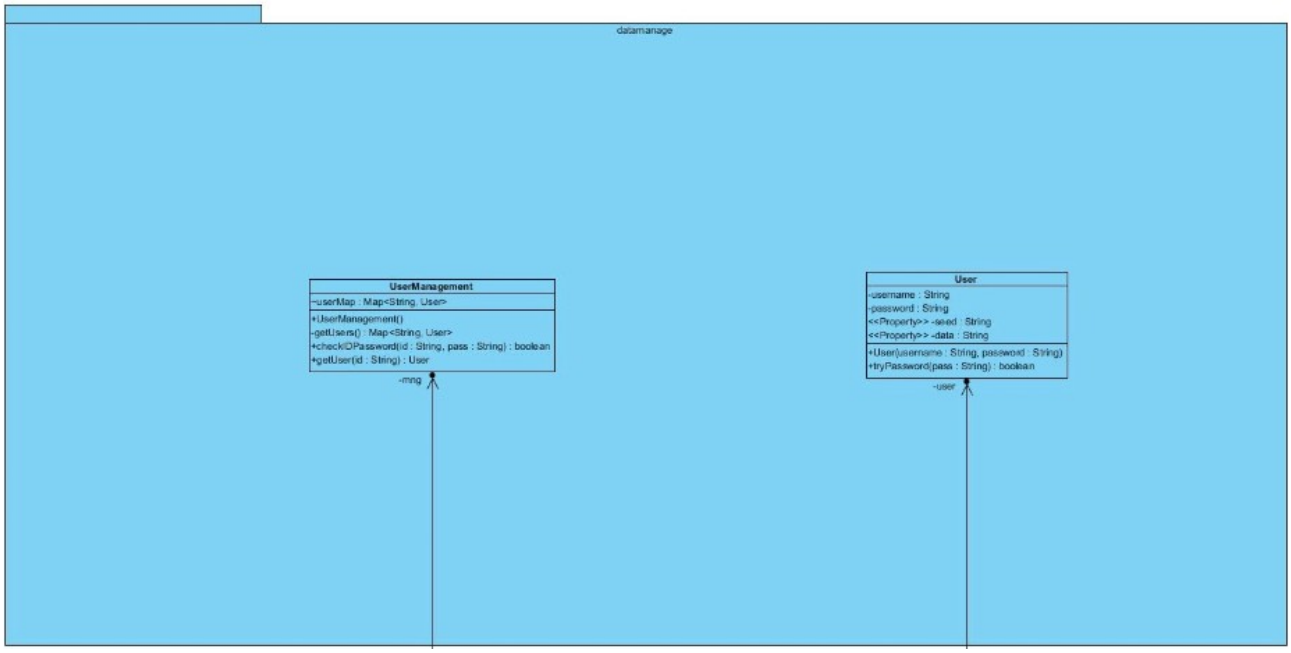
We divided the project into packages according to its subsystems

a)game Package



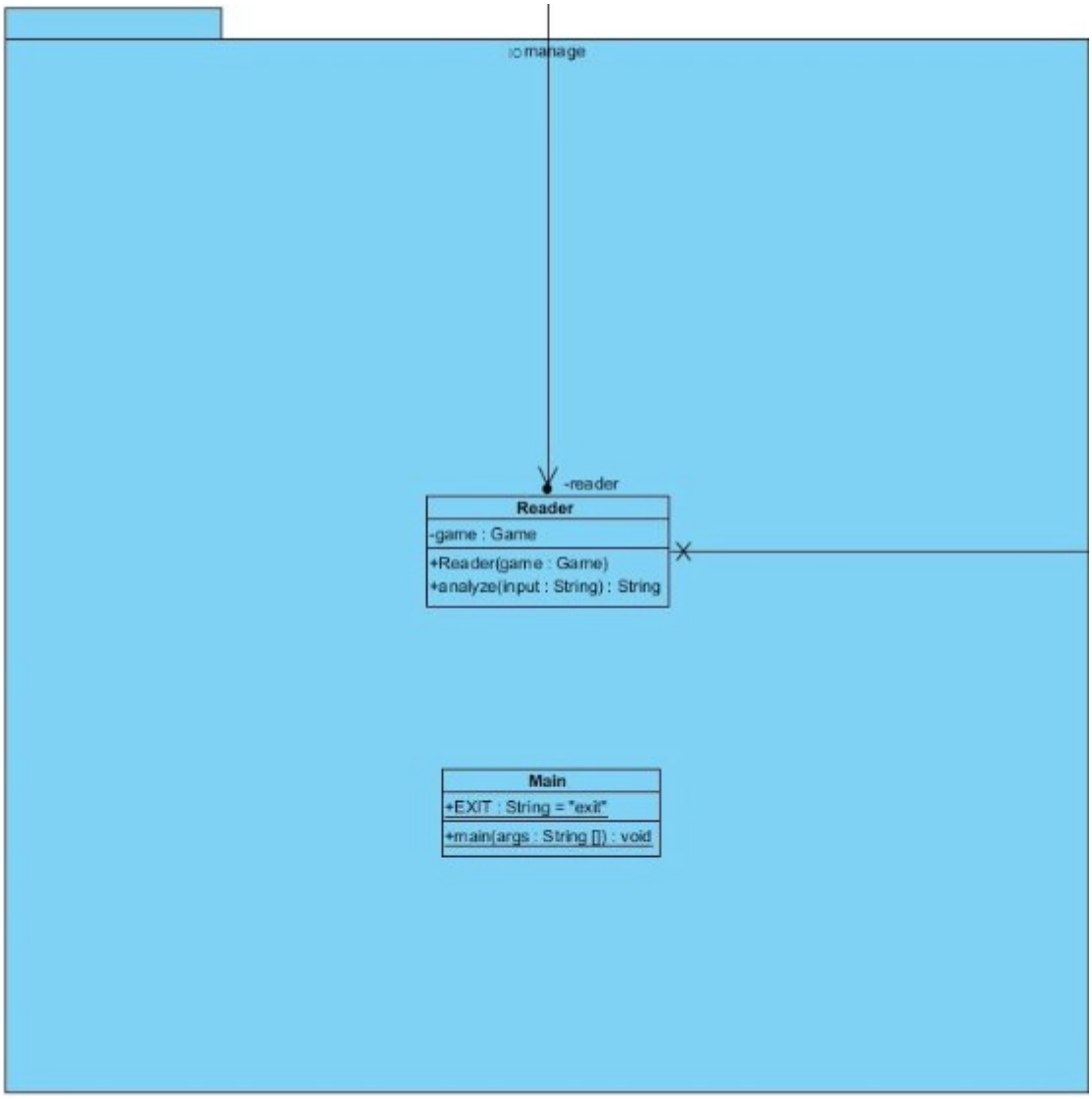
This package consists of game elements such as player,item, Map. This package itself does not have any connections to outside of the program. It just handles tasks like, map generation, actions and it defines objects like player, hostile characters, items.

b)datamanage Package



This package handles tasks like reading files, user authentication, saving and loading games.

c)IOmanage Package

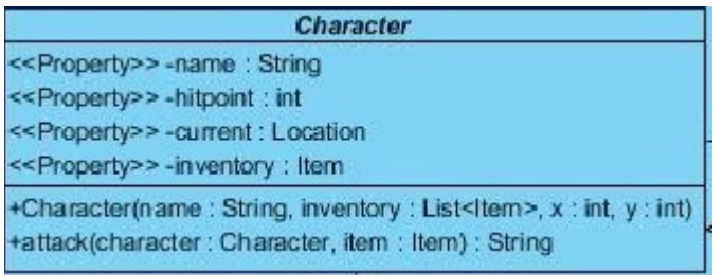


This package manages reading and understanding of the user inputs and calls Methods from game Package according to this inputs

4.4.Class Interfaces

a)game Package

1.**public** abstract class Character



Properties

private String name:

This is name of the Character

This property is **private** because it is only intended to access by setter and getter Methods provided

private Location current:

This is the current location of the Character

This property is **private** because it is only intended to access by setter and getter Methods provided

private List<Item> items:

This is the list of items that character has

This property is **private** because it is only intended to access by setter and getter Methods provided

private int hitpoint:

Remaining hit-points of the Character

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public Character(String name,List<Item> inventory,int x,int y);

@param name String name to assign to character

@param inventory List of Items Character has

@param x Location of the Character

@param y Location of the Character

name and inventory will be initialized. current Location of the Character will be set according to x and y

Methods

public String attack(Character target,Item item);

Player attacks given target with the given item if it is possible, result given with an appropriate **String** message Damage will be calculated according to users stats, and equipment This method designed to be called from Player object inside Game object inside Parser object game.player.attack(character,item);

@param target to Attack

@param Item to use while attacking

public String getName();

Getter method for the property name

@return name as a String

public Location getLocation();

Getter method for the property Location

@return current Location of the Character

public List<Item> getItems()

Getter method for the property items

@return items as a List<Item> object

public int getHitPoints()

Getter method for the property hitpoint

@return hitpoint Integer

2.**public** abstract class NonPlayerCharacter extends Character

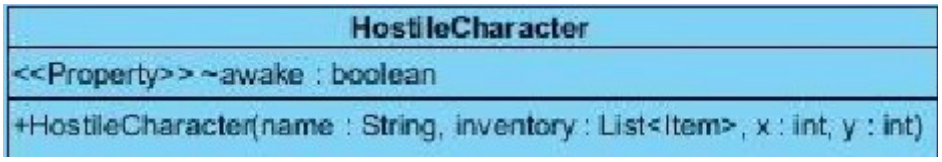
Constructors

public NonPlayerCharacter(String name,List<Item> inventory,int x,int y);

@param name String name to assign to character
@param inventory List of Items Character has
@param x Location of the Character
@param y Location of the Character

This will call the super Constructor with the given parameters

3.**public** class HostileCharacter extends NonPlayerCharacter



Properties

private boolean awake:

this boolean shows that hostile character is aware of the presence of the player, and it will attack to player with the each action player does.

This property is **private** because it is only intended to access by setter and getter

Methods provided

Constructors

public NonPlayerCharacter(String name,List<Item> inventory,int x,int y);

@param name String name to assign to character
@param inventory List of Items Character has
@param x Location of the Character
@param y Location of the Character

This will call the super Constructor with the given parametersMethods

Methods

public boolean getAwake()

Getter method for the property awake

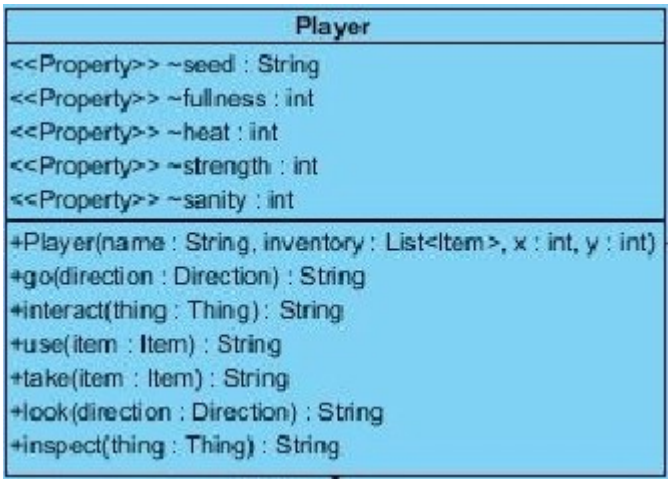
@return awake as a boolean

public void setAwake(boolean newAwake)

Setter method for the property awake

sets the awake boolean accordingly

4.**public** class Player extends Character



Properties

private String seed:

This is the seed of the Player, all randomness will be according to this value, this way players will be able to challenge other players on same generation, also they will be able to play to play on the same world again

This property is **private** because it is only intended to access by setter and getter Methods provided

private int fullness:

This is the hunger level of the Player, it is designed as fullness to make it parallel with other stats, this number will decrease with every action player does, when it hits zero player will starve

This property is **private** because it is only intended to access by setter and getter Methods provided

private int heat:

This is the heat level of the player, this will change according to surrounding temperature. Player is forced to find some heat source in order to survive nights

This property is **private** because it is only intended to access by setter and getter Methods provided

private int strength:

This is the strength stat of the player, this number will act as a multiplier when calculating the damage output of his/her atacks. Player's strength may increase according to tasks done, or decrease with hunger, etc.

This property is **private** because it is only intended to access by setter and getter Methods provided

private int sanity:

This is the sanity level of the player, It will decrease in dark (night,dungeons,etc.) and will increase if player has no hunger, heat problem during daytime. If this number goes too low, game will be over

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public Player(String name, List<Item> inventory, int x, int y);

@param name String name to assign to Player
@param inventory List of Items Player has
@param x Location of the Player
@param y Location of the Player

This will call the super Constructor with the given parameters

Methods

public String go(Direction direction);

@param direction to move

@return resulting String message

Player moves according to given direction if it is possible

public String interact(Thing thing);

@param thing to interact

@return resulting String message

Player interacts with given Thing if it is possible

public String use(Item item);

@param item to use

@return resulting String message

Player uses given Item if it is possible

public String take(Item item);

@param item to take

@return resulting String message

Player takes given Item if it is possible, if successful Item will be added to players inventory.

public String look(Direction direction);

@param direction to look

@return resulting String message

Player looks to given direction if it is possible

public String inspect(Thing thing);

@param thing to inspect

@return resulting String message

Player inspects given thing if it is possible,Reader will understand look Thing, as inspect Thing

public String getSeed();
@return seed as String

Getter method for the property seed
public void setFullness(int fullness);
@param fullness to set
Setter method for the property fullness
public int getFullness();
@return fullness as int

Getter method for the property fullness

public void setHeat(int heat);
@param heat to set
Setter method for the property heat
public int getHeat();
@return heat as int
Getter method for the property heat

public void setStrength(int strength);

@param strength to set

Setter method for the property strength

public int getStrength();

@return strength as int

Getter method for the property strength

public void setSanity(int sanity);

@param sanity to set

Setter method for the property sanity

public int getSanity();

@return sanity as int

Getter method for the property sanity

5.public interface Thing

Methods

public String getMessage();

@return String message to be displayed

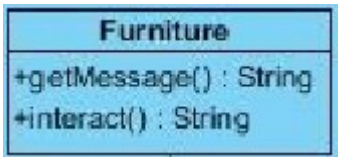
This Methods returns the message to be send when interacted with the object itself

public String interact();

@return String message to be displayed

A method to interact with the Thing

6.public class Furniture implements Thing



Methods

public String getMessage();

@return String message to be displayed

This Methods returns the message to be send when interacted with the object itself

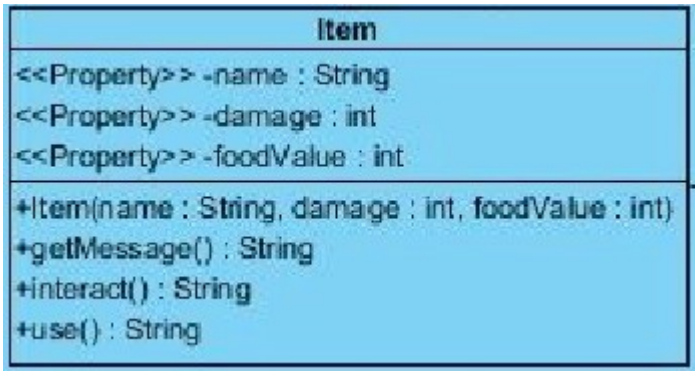
Methods

public String interact();

@return String message to be displayed

A method to interact with the Furniture

7.**public** class Item implements Thing



Properties

private String name:

This is name of the Item

This property is **private** because it is only intended to access by setter and getter Methods provided

private int damage:

If Item is a weapon this will get a positive value and it will be used to calculate output damage of the Character attacks. If it is zero this means it is not a weapon

This property is **private** because it is only intended to access by setter and getter Methods provided

private int foodValue:

If Item is a food this will get a positive value and it will be edible and this food will increase the fullness of the Player by the value of it

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public Item(String name,int damage,int foodValue);

Methods

public String getMessage();

@return String message to be displayed

This Methods returns the message to be send when interacted with the object itself

Methods

public String interact();

@return String message to be displayed

A method to interact with the Furniture

public String use()

@return String message to be displayed

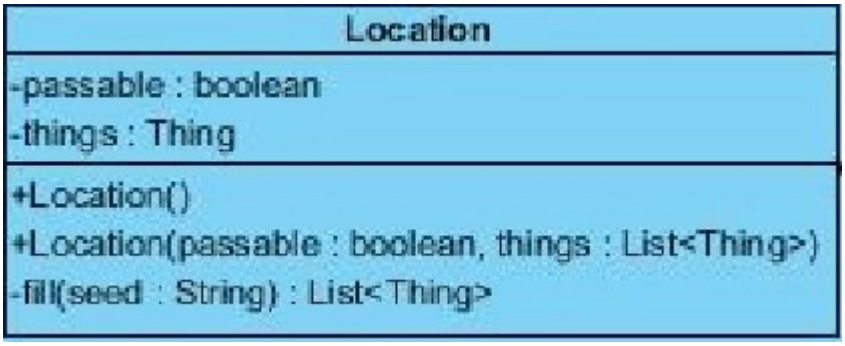
This method is called when Item is used

public String getName();
@return name as String
Getter method for the property name

public int getDamage();
@return damage as int
Getter method for the property damage

public int getFoodValue();
@return foodValue as int
Getter method for the property foodValue

7.**public** class Location



Parameters

private boolean passable;

Indicates if Location is passable, can be changed as Boolean[3] passable,

to indicate directions from this location depending of the implementation

This property is **private** because it is only intended to access by setter and getter Methods provided

private List<Thing> things;

List of Thing's inside this location, will be randomly generated if location is new, while loading it will be generated according to given List

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public Location();

This is the default Constructor which is used to generate new locations randomly

public Location(boolean passable,List<Thing> things);

@param passable boolean that designates if location is passable

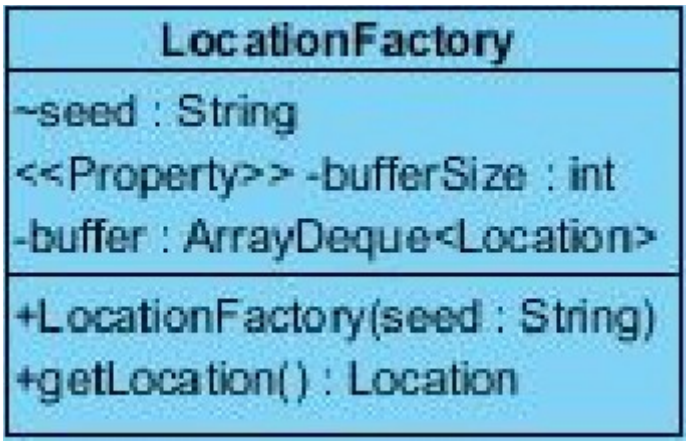
@param things is a list of Thing objects which exist on that location object
This Constructor is used when loading a saved game from a save file

Methods

private List<Thing> fill(String seed);
@param seed a string that manages the randomness
@return list of Thing's that exists on that location

This method randomly fills a location according to a seed. This **private** method called when object is constructed with default Constructor.

8.**public** class LocationFactory



private String seed;

A copy of the seed of the player to manage randomness

This property is **private** because it is only intended to access by setter and getter Methods provided

private int bufferSize;

A designated buffer size of buffer to manage the number of Locations to keep in the ArrayDeque

This property is **private** because it is only intended to access by setter and getter Methods provided

private ArrayDeque<Location> buffer;

An ArrayDeque that keeps generated Locations inside. It is used as a queue in this program

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public LocationFactory(String seed);

@param seed as a String to manage randomness

This will initialize the property seed with parameter seed

public void setBufferSize(int bufferSize);

@param bufferSize as a String to change buffer size according to needs

public Location getLocation();

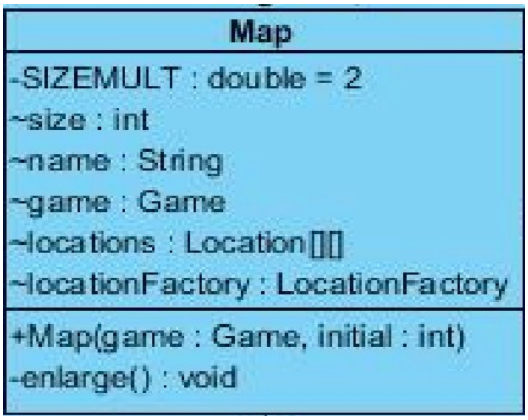
@return newLocation

This method poll a Location from arraydeque and return it.

public void addLocation();

This method will be called when system is idle and it will fill the arraydeque with newly generated Locations

8.public class Map



Parameters

private final double SIZEMULT = 2;

SIZEMULT is a **private** constant double that designates the size increase of the map when border is reached

private int size;

size is an **int** that shows the current size of the square map

private String name;

This property is **private** because it is only intended to access by setter and getter Methods provided

Game game;

An instance of the game object to reach other objects in the program

private Location[][] locations;

A 2 dimensional array of Locations to represent the Map.

This property is **private** because it is only intended to access by setter and getter Methods provided

private LocationFactory locationFactory;

A LocationFactory to generate Locations when player move

Constructors

public Map(Game game,int initial);

@param game, game object passed from the caller to make it possible to reach other objects In the program

@param initial an integer value that shows the initial size of the square 2 dimensional Location array

Constructor sets the Properties with the parameters and initializes locations and LocationFactory

Methods

private void enlarge();

This method is called when player hits the borders of the existing Map, it will create a new Location[][] with the size = (size *SIZEMULT) and pass the existing locations to the middle of the new Location[][]

9.**public** enum Direction

Direction is enumeration that shows directions with the following constants in it

North,

East,

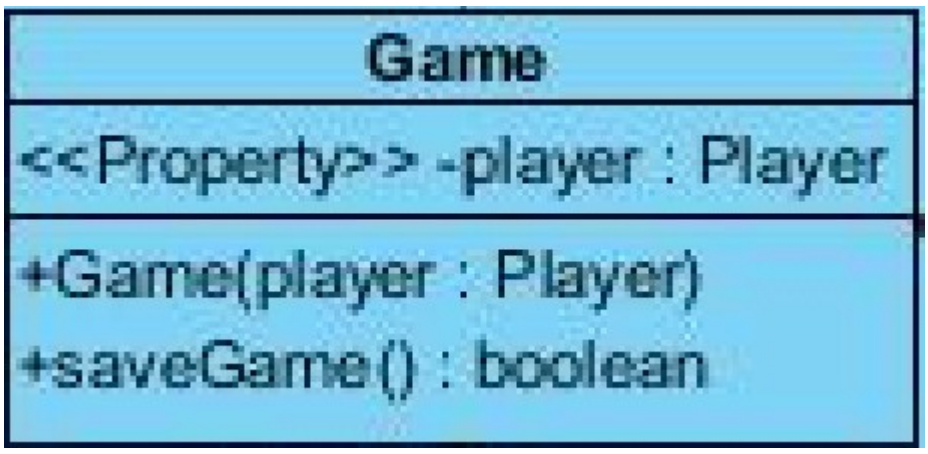
South,

West,

Upwards,

Downwards

10.**public** class Game



Parameters

private Player player;

An Instance of the player objects to be called from other classes that have the instance of the game object

for example game.getPlayer().go(Direction.North)

This property is **private** because it is only intended to access by setter and getter

Methods provided

Constructors

public Game(Player player);

@param player is an instance of player from caller object

Constructor sets property player with parameter player

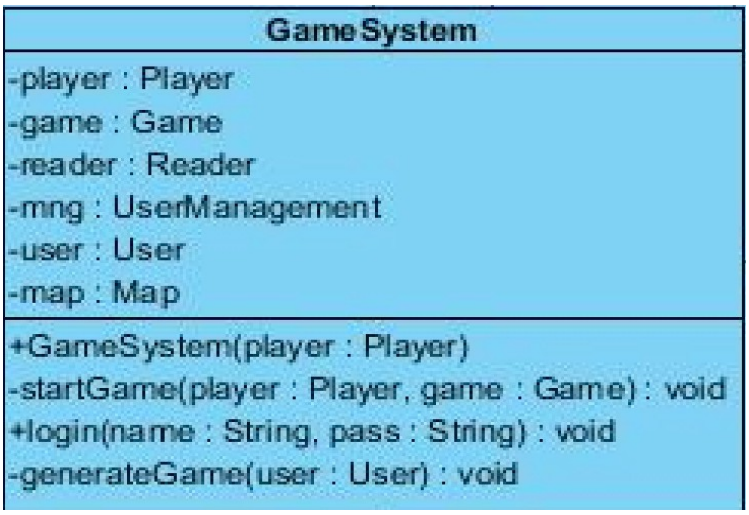
Methods

public Player getPlayer();

@return Player

returns the player object

11.**public** class GameSystem



Properties

private Player player;

A player object that will be generated according to User logged in

This property is **private** because it is only intended to access by setter and getter Methods provided

private Game game;

A Game object that will be generated according to User logged in

This property is **private** because it is only intended to access by setter and getter Methods provided

private Reader reader;

A Reader object which will analyze the inputs of the User and call the Methods of the player object

This property is **private** because it is only intended to access by setter and getter Methods provided

private UserManagement mng;

An Object that manages the login authentications

This property is **private** because it is only intended to access by setter and getter Methods provided

private User user;

An object that represents the user logged in and keeps its data at runtime and saves it during logout

This property is **private** because it is only intended to access by setter and getter Methods provided

private Map map;

A Map object which player will be wandering during playtime

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public GameSystem();

A Constructor with no parameters will initialize the UserManagement object

Methods

private void startGame(Player player,Game game);

@param player a Player object that generated according to user data

@param game, a Game object that generated according to user data

A **private** method that will be called when user logs in and starts the game

public void login(String name, String pass);

@param name is a String that shows the login name of the user

@param pass is a String that shows the login password of the user

This method will take username and password and call the authentication method of the UserManagement object. If authentication is successful it will call the generateGame method.

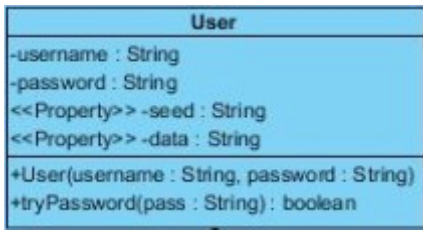
private void generateGame(User user);

@param user the user object that authenticated

This method will load the game according to User data, if data is empty it will randomly assign a seed and generate according to that value. After loading it will start the game.

b)datamanage Package

1.**public** class User



Properties

private String username;

A **String** that indicates the login username of the User

This property is **private** because it is only intended to access by setter and getter Methods provided

private String password;

A **String** that indicates the login password of the User. This can be encrypted in order to increase security

This property is **private** because it is only intended to access by setter and getter Methods provided

private String seed;

A String, seed value that will guide randomness

This property is **private** because it is only intended to access by setter and getter Methods provided

private String data;

A String, data which will be loaded at the startup, and will be saved during exit

This property is **private** because it is only intended to access by setter and getter Methods provided

Constructors

public User(String username, String password);

@param username,that indicates the login username of the User

@param password,that indicates the login password of the User

This Constructor sets the parameter values to property values

Methods

public String getSeed();

@return seed as a String

This is the getter Method for Property seed

public String getData();

@ return data as a String

This is getter Method for Property data

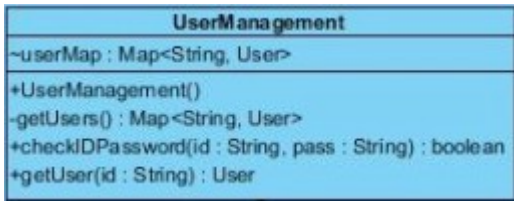
public boolean tryPassword(String pass)

@param pass a String that will be compared to password of the user

@return boolean success of the trial

This method checks if password user entered is correct

2.**public** class UserManagement



private Map<String,User> userMap;

public UserManagement();

This Constructor takes no parameters and it fills userMap with the method getUsers()

private Map<String,User> getUsers();

@return Map<String,User> as a map of Users with names as their keys

This method reads a certain data file and fills the userMap accordingly

public boolean checkIDPassword(String id,String pass);

@param id a String that shows the id that will be tried

@param pass a String that shows the password that will be tried

@return boolean success

This method finds the user with given id and checks if pass is correct, and returns a boolean that shows the success of the authentication

public User getUser(String id);

@param id a String as the id of the User

@return User that found,null if User does not exists

This method looks for given id in the userMap and returns that user if it exists

c)IOManage package

1.**public** class Reader

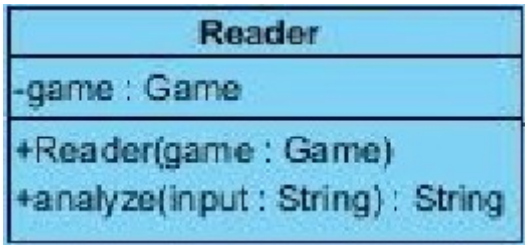
Properties

private Game game;

A game object that will help Reader to reach other objects in the program

Constructors

public Reader(Game game)



@param game a simple Constructor to set property game as parameter game

Methods

public String analyze(String input)

@param input a string input that typed by the user

@return resulting String message to print on the screen

This Method takes a **String** input with format action receiver tool or action direction, etc. and execute the regarding Methods. After execution returns the resulting **String** to the caller

5.1.Glossary

zOOrk : Text based game, that implemented by Java an an OO project.

5.2.References

http://en.wikipedia.org/wiki/Zork

fatih.ug.bilkent.edu.tr – Project API documentation