**Slay The Password  
Design Document**

**Week 2**

**CIS 3296 Section 03**

**Fall 2024**

**Team Members:**

- Jonathan Cherry

- Liye Guo

- Illia Matviienko-Hart

- Chiku Okechukwu

- Ryan M Tenerelli

- Stella Wang

**Repository URL:**

https://github.com/cis3296f24/project-03-slaythepassword

**Project Board URL:**

https://trello.com/b/O9q6tCuB/slay-the-password

Table of Contents

Table of Contents

[Document Overview 3](#_Toc182438457)

[Architecture 3](#_Toc182438458)

[Detail Design API 4](#_Toc182438459)

[Game Class 5](#_Toc182438460)

[Password Class 5](#_Toc182438461)

[PasswordImpl Class 6](#_Toc182438462)

[User Interface 6](#_Toc182438463)

[UserImpl Class 7](#_Toc182438464)

## Document Overview

This Design Document describes the software architecture and how the requirements are mapped into the design. This document will be a combination of diagrams and text that is describing what the diagrams are showing. The Design Document also specify the complete design of the software implementation using Javadoc.

## 

## Architecture

This section describes the different components and their interfaces using UML. For example: client, server, database. For each component provide class diagrams showing the classes to be developed (or used) and their relationship.

A screen shot of a computer screen

Description automatically generated

PLACEHOLDER Figure 1 UML Class Diagram for SlayThePassword

This UML diagram represents the core structure and interactions of the "Slay the Password" game. The Game class orchestrates the gameplay, utilizing the User interface (implemented by UserImpl) to manage player health and the Password class (potentially implemented by PasswordImp) to generate and validate password challenges. The Game class controls the game flow, initializing the user's health, presenting password challenges, and validating user input. The Password class is responsible for generating passwords based on specific conditions, providing hints, and checking user-submitted passwords against the correct answer. The User interface defines methods for managing the player's health, which are implemented in the UserImpl class. Additionally, the Conditions interface and its potential implementation ConditionsImpl suggest a mechanism for defining and managing various password conditions or challenges within the game.

## Detail Design API

For each class define the data fields, methods.

* The purpose of the class.
* The purpose of each data field.
* The purpose of each method
* Pre-conditions if any.
* Post-conditions if any.
* Parameters and data types
* Return value and output variables
* Exceptions thrown\*.

This information should be in structured comments (e.g. Javadoc) in the source files. A documentation generation tool (e.g. Javadoc) may be used to generate the document as a draft.

### Game Class

**Purpose:** The Game class contains the main game loop and controls the overall flow of the "Slay the Password" game.

**Data Fields:**

None explicitly defined in the provided code.

**Methods:**

***main(String[] args) :***

**Purpose:** Starts and runs the game.

**Parameters:** id (int) - the account ID, initialBalance (int) - the initial balance.

**Parameters: args (String[])** - Command line arguments (not used).

**Return Value:** None.

**Exceptions Thrown:** InterruptedException - If the thread is interrupted during

### Password Class

**Purpose:** The Password class represents a password challenge in the game. It handles password generation, providing hints, and validating user input.

**Data Fields:**

folder (String) - Stores the name of the folder containing password conditions.

randCondition (Conditions) - Holds the randomly selected password condition for the current challenge.

conditions (List&lt;Conditions>) - A list of available password conditions.

**Methods:**

***Password()***

* **Purpose:** Constructor for the Password class. Loads conditions and selects a random one.
* **Parameters:** None.
* **Return Value:** None.

***loads()***

* **Purpose:** Loads conditions from the specified folder.
* **Parameters**: None.
* **Return Value:** List&lt;Conditions> - A list of loaded conditions.

**v*alidate(String userInput)***

* **Purpose:** Validates the user's password input against the condition.
* **Parameters:** userInput (String) - The password entered by the user.
* **Return Value:** None.

***displayConditions()***

* **Purpose:** Displays the hint for the current password condition.
* **Parameters:** None.
* **Return Value:** None.

### PasswordImpl Class

**Purpose:** The PasswordImp class represents an implementation of a password with specific conditions. It is not used in the main game.

**Data Fields:**

difficulty (int) - The difficulty level of the password.

conditions (Conditions[]) - An array of conditions for the password.

**Methods:**

***PasswordImp(String conditions[])***

* **Purpose:** Constructor for the PasswordImp class.
* **Parameters:** conditions (String[]) - An array of conditions for the password.
* **Return Value:** None.

### User Interface

**Purpose:** The User interface defines methods for managing the user's health in the game.

**Data Fields:**

None

**Methods:**

***losehp(Integer value)***

* **Purpose:** Decreases the user's health by a specified value.
* **Parameters:** value (Integer) - The amount by which to decrease the health.
* **Return Value:** None.

***restorehp(Integer value)***

* **Purpose:** Restores the user's health by a specified value.
* **Parameters:** value (Integer) - The amount by which to restore the health.
* **Return Value:** None.

***getHealth()***

* **Purpose:** Returns the user's current health.
* **Parameters:** None.
* **Return Value:** int - The user's health.

### UserImpl Class

**Purpose:** The UserImpl class is an implementation of the User interface. It manages the user's health.

**Data Fields:**

health (int) - The user's current health.

maxHealth (int) - The user's maximum health.

**Methods:**

***UserImpl()***

* **Purpose:** Constructor for the UserImpl class. Initializes the user's health.
* **Parameters:** None.
* **Return Value:** None.

***losehp(Integer value)***

* **Purpose:** Decreases the user's health by a specified value.
* **Parameters:** value (Integer) - The amount by which to decrease the health.
* **Return Value:** None.

***restorehp(Integer value)***

* **Purpose:** Restores the user's health by a specified value.
* **Parameters:** value (Integer) - The amount by which to restore the health.
* **Return Value:** None.

***getHealth()***

* **Purpose:** Returns the user's current health.
* **Parameters:** None.
* **Return Value:** int - The user's health.