Project documentation

Destiny

LUT University

CT60A2412 Object-Oriented Programming

Table of Contents

[1. Team 2](#_Toc196071571)

[2. Project description 2](#_Toc196071572)

[3. Features 2](#_Toc196071573)

[4. Class Diagram 3](#_Toc196071574)

[4.1. Data 3](#_Toc196071575)

[4.1.1. Adventurer 4](#_Toc196071576)

[4.1.2. Enemy 5](#_Toc196071577)

[4.1.3. Special 6](#_Toc196071578)

[4.2. Domain 7](#_Toc196071579)

[4.2.1. Area 7](#_Toc196071580)

[4.2.2. Battle 8](#_Toc196071581)

## Team

* Boris Simeonov, 002297714, [Boris.Simeonov@student.lut.fi](mailto:Boris.Simeonov@student.lut.fi) – developed back-end and helped with front-end.
* Tereza Opanska, 002285551, [Tereza.Opanska@student.lut.fi](mailto:Tereza.Opanska@student.lut.fi) – developed front-end, helped with back-end and made all in-game pixel art sprites/icons.

## Project description

Destiny is a Pokémon style turn based android game where players can hire adventurers to use against battles with monsters and progress through increasingly more difficult fights. Each adventurer belongs to a class which gets picked by the player. Currently there are 6 classes – Knight, Mage, Paladin, Berserker, Assassin and Warlock. Every class has its own unique stat distributions and special ability which make them effective in certain situations.

Combat is further enriched through attack and resistance types. Both adventurers and enemies can have either physical or magical attack types depending on their class, while also possessing the corresponding resistances. This for example means that a mage who uses magical attacks might struggle against an enemy with high magical resistance. This encourages the player to use different types of adventurers.

Players can also engage in battles that scale in difficulty: Easy, Medium and Hard. Each increased difficulty adds an additional enemy to the battle, making it tougher to win. Each defeated enemy earns an adventurer one experience point. These points can later be traded in the training area to boost any one combat stat of an adventurer by one point.

Adventurers’ performance gets tracked and can be viewed in the guild. Players can see the total battles fought by an adventurer along with their total victories.

## Features

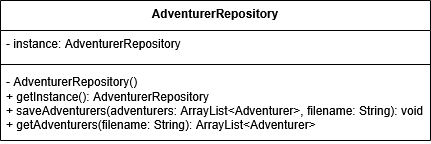
* Hiring adventurers.
* Adventurers have unique sprite art.
* Picking an adventurer to use in combat.
* Picking combat difficulty.
* Turn based combat.
* During combat a player can either use an adventurer’s basic or special ability.
* Battle log providing information during combat.
* The adventurer is returned to the guild on defeat.
* Combat includes randomness implemented with critical strikes.
* The application is divided into three main fragments.
* The adventurers are stored in and loaded from internal storage.
* Checking adventurers’ combat stats and past records.
* Training adventurers by spending their experience points in return for extra combat stats.

## Class Diagram

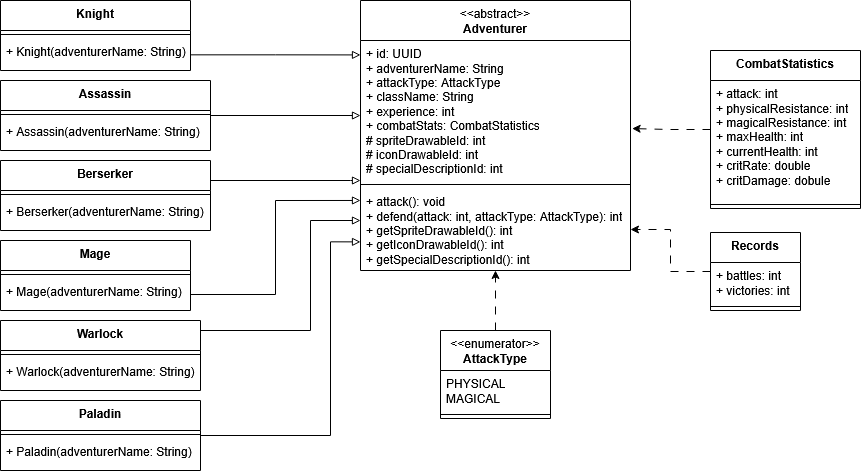
The class diagrams and accompanying explanations do not include the UI layer – app. The document is structured first according to layers and then subpackages within those layers.

### Data

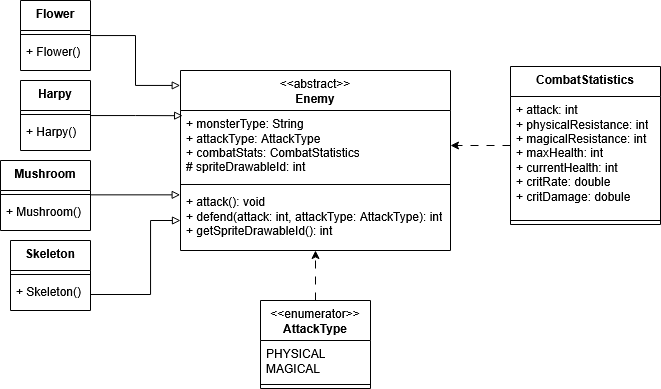
This layer contains models and the repository which are used by the rest of the application to store, retrieve and use data. There exists only one repository class which stores a list of adventurers to a given internal file, as the application’s scope does not require more complex repositories.



### Adventurer

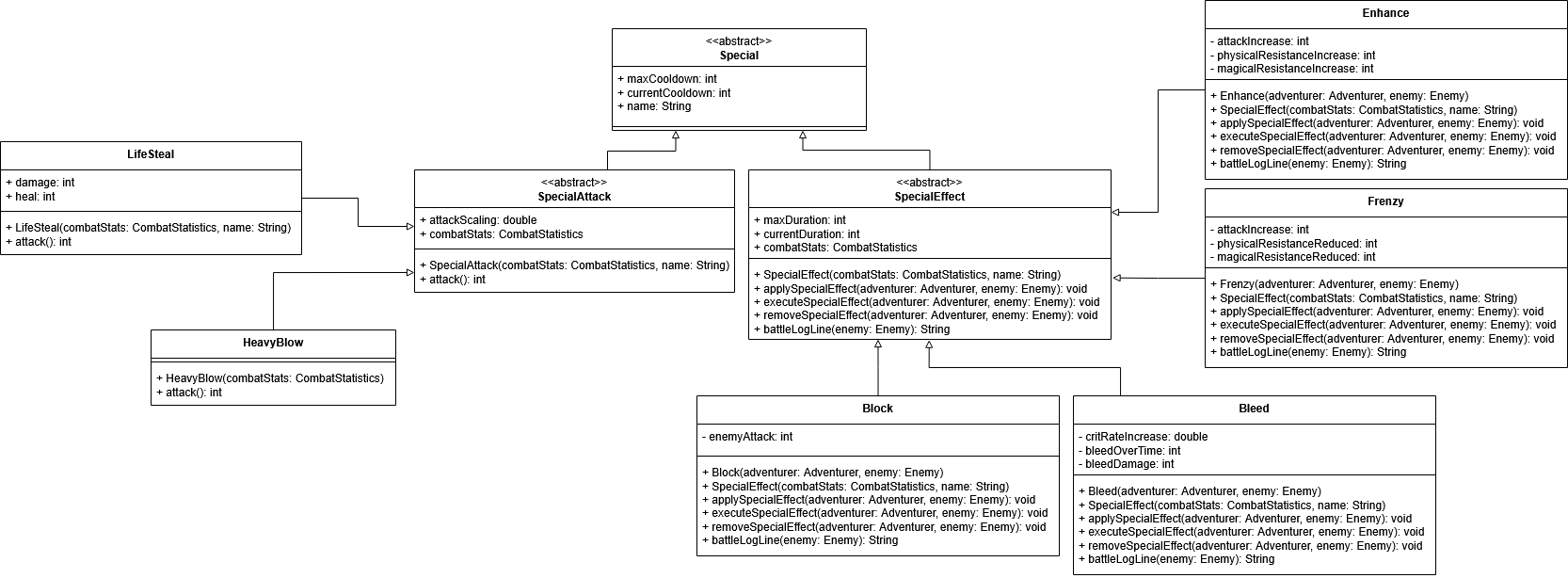
This package contains the abstract class Adventurer and its specific implementations. Additionally, it includes enumerations and classes, which help describe the Adventurer. The Adventurer class is the only class that is stored to file.

### Enemy

The structure of this package is very similar to Adventurer, as they have a lot in common. The principle is the same – a main abstract class Enemy and specific implementations for each enemy type. In addition, similar helper classes are used to describe the enemy.

### Special

This package contains the models for the special moves that each adventurer has. They are separated into either Special Attack or Special Effect, which themselves have specific implementations. The class diagram may look cluttered, but this is to show explicitly that each non-abstract class overrides the abstract methods given by its parent class.

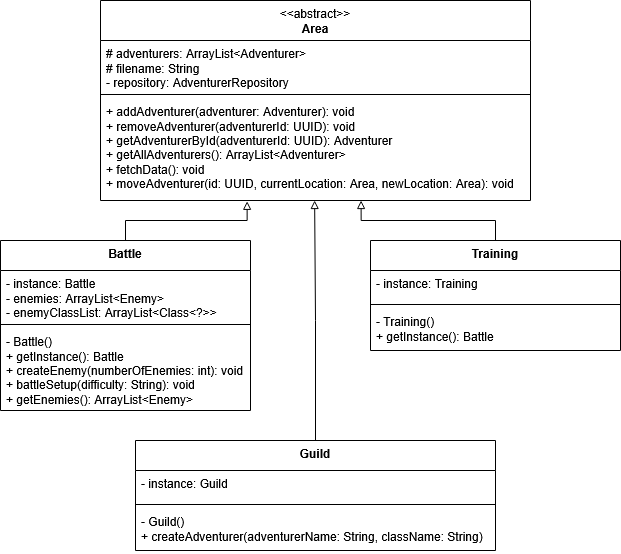


### Domain

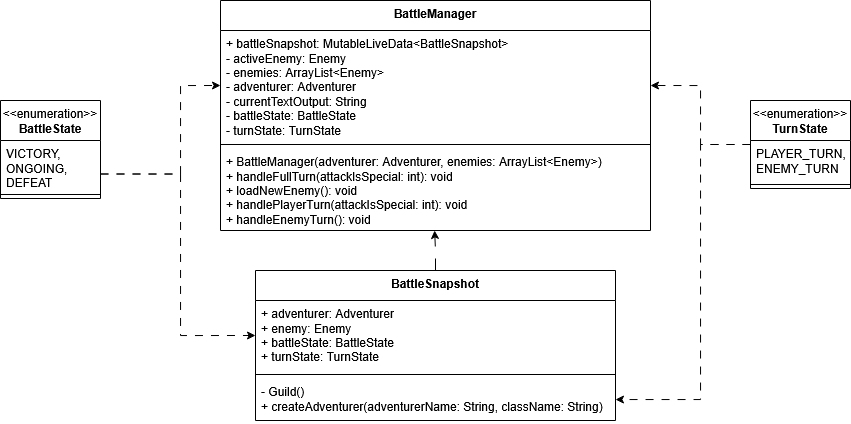
The domain layer contains all the business logic – the code that makes the application what it is. In this case it contains the three areas that the application is divided into plus the battle manager.

### Area

The area package contains an abstract class definition of Area and three implementations, each representing one fragment the user can access. Throughout the lifecycle of the application the adventurers can be moved between areas so specific actions can be executed on them. This implementation additionally allows for further expansion of the scope of the project, as this means that every area can have deeper functionality.

The guild area handles hiring new adventurers, the battle area handles backend preparations for battle and the training are handles stat upgrading.

### Battle

This package contains the core of the application – the battle manager. It accepts an adventurer and enemies and handles turns, taking into account player action and battle logging. The most important part of the battle manager is the way it communicates with the UI layer. As the handling of a full turn (a player and an enemy turn) is done, the UI needs to stay responsive and update everything constantly. The solution is to create a BattleSnapshot class which is used to transfer a snapshot of the state of the battle from the domain layer to the UI layer. The way that the transfer is done is with an observer pattern, where the UI listens for changes in the snapshot. The implementation uses MutableLiveData.