# Pokemon Arena


**Example for professional Software-Engineering**

# Table of Contents

# 1 Requirement Engineering

At the beginning of every Software-Project stands the challenge to figure out all the requirements. It's the time to sit together with your team and the customer to discuss and brainstorm functionalities and requirements of the problem.

**Concrete questions:**

- What is the problem that the software must solve?
- What functionalities needs the software to cover the problem?

In the end of each Requirement Engineering document the **decided decisions**.

**Problem:**
Every Pokemon-Trainer should receive a "randomly" assigned Starter-Pokemon. The Pokemon-Trainer should be able to fight with his assigned Starter-Pokemon to catch wild Pokemons.

**Requirements:**

- The App must provide an assign Starter-Pokemon mechanism.
- The App must provide a catch Wild-Pokemon mechanism.
- The App must provide a list of catched Pokemons per trainer.
- The App must provide an active Pokemon.
- The App must provide a fight mechanism for Pokemons.

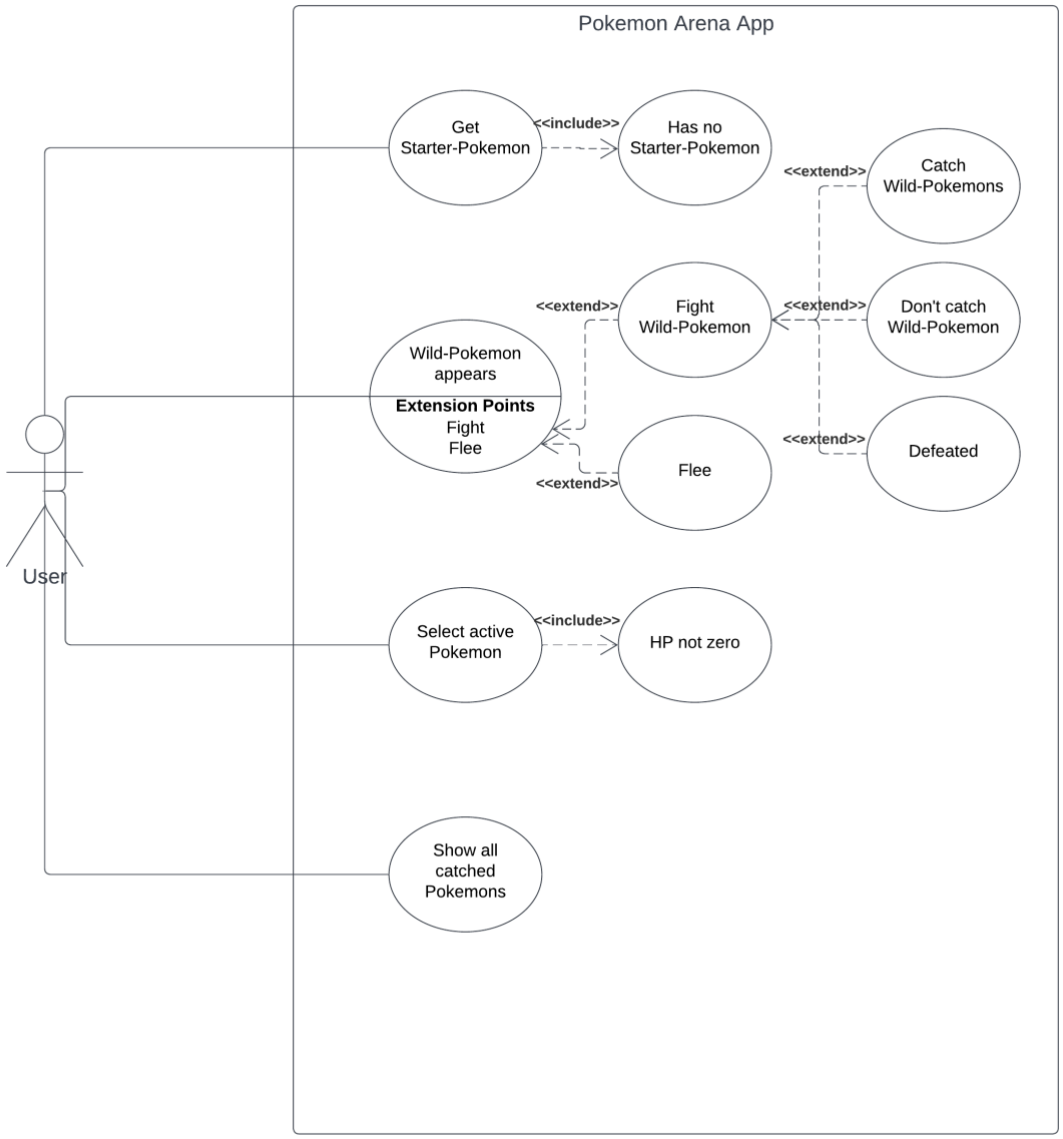
# 2 Application description

Now that we have the requirements for the Pokemon Arena application, it's time to describe the application from the End-user perspective. In this phase it is important to show the customer a non-technical vision of the App. To do so we're formulating so called User-Stories, which will later use to derive Use-Cases.

## 2.1 User-Stories:

a) As a user I want to get a Starter-Pokemon to get started for the adventure.
b) As a user I want to fight against Wild-Pokemons to catch them and extend my collection.
c) As a user I want to select a catches Pokemon to be my active one.
d) As a user I want to get an overview of my catches Pokemons.

## 2.2 Use-Cases:




# 3 Object Oriented Design

Now that we have gained a clear image of the application, it's time to get more technical and identify the needed classes and put them into relation. The common tool for this is UML and the class diagram.

A class always aims to represent an object in the real world, that is not always easy because a lot of things in the real world just happen and we don't have to think them through, that is different for computer programs. Even though the technology evolved very fast in the past few years, a computer still has not been able to independently solve complex, for computer

impropriety structured, problems. Hence, we should target to think all aspects of the program through it's not always easy to think of everything in advance.

It's better to neglect details than to not moving forward. A lot of programmers tend to get lost into details and get into what's called **"Analysis Paralysis"**.

## 3.1 Class Diagram

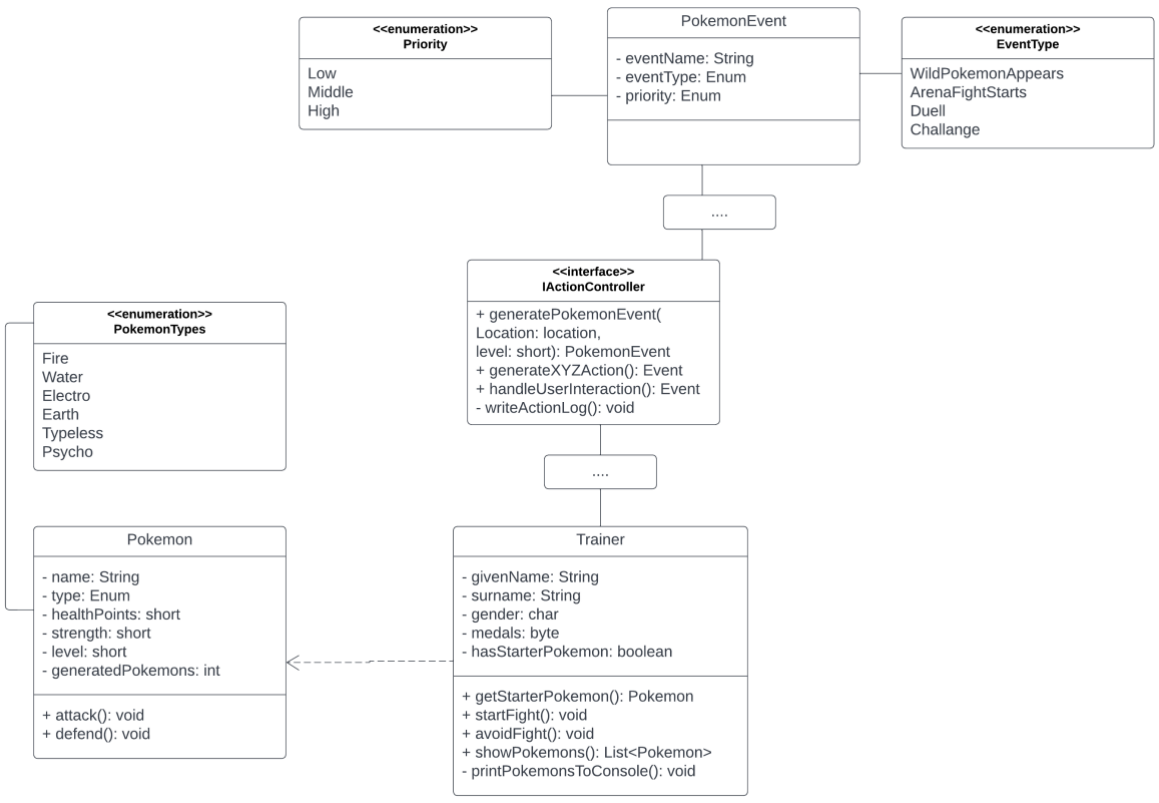
To create a Class diagram, the first step is to identify the Entities.

**Entities:**

- Pokemon
- Trainer

The next step is to design the class diagram.

**Class Diagram:**




The class diagram is not at all finished, but it already points out some of the complexity assigned to the program.

## 3.2 Sequence Diagram of Pokemon

A Sequence Diagram describes the flow of a program in a specific scenario. In contrast to class diagrams and use cases, this one is dynamic and not static. This diagram is used to show clippings of the dynamic interactions between objects.

In a sequence diagram we're working with objects and not with classes, that is the reason why the name changes from Pokemon to aPokemon or from Trainer to aTrainer.