Workshop #2

Technical Report

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tito Burbano

Software Modeling Foundations

2024

Introduction:

The objective of this report is to explain how the workshop solution was reached, taking into account what was explained in class in order to comply with the user stories provided.

Methods:

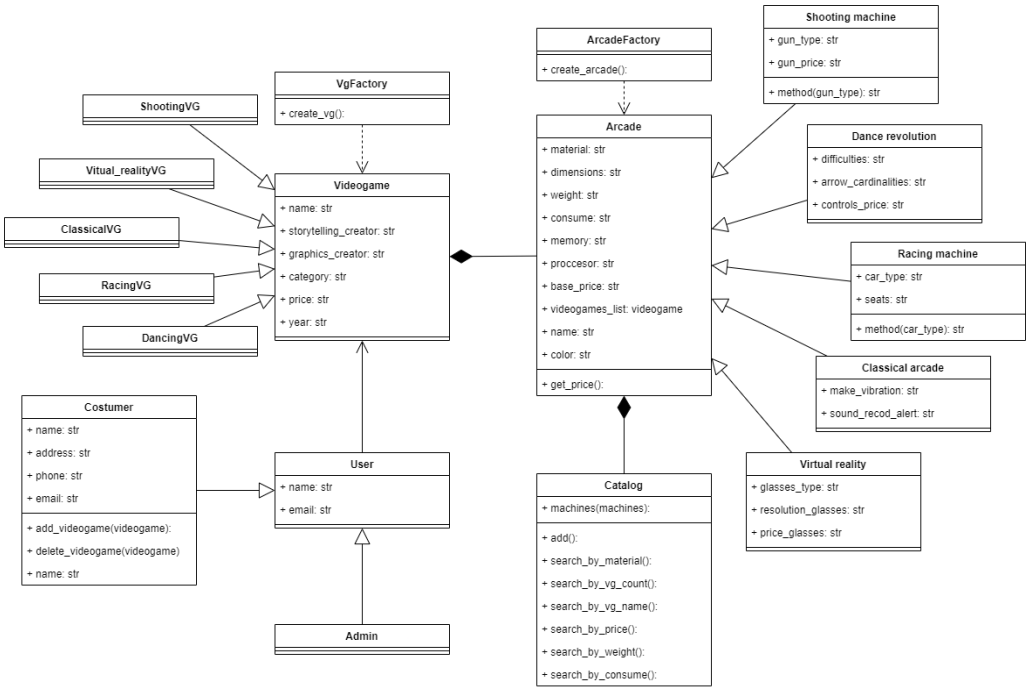
Taking into account the requested requirements, in this project I choose to use object-oriented programming (OOP) as the main development paradigm. This choice is based on the inherent advantages that OOP offers for structuring complex systems. Through this approach, a "modular division" of the system is achieved, facilitating its scalability and long-term maintenance.

Object-oriented programming allows you to organize code into classes and objects, which represent entities with specific attributes and behaviors. This modular structure provides a flexible environment where each module (class or set of classes) can be developed, maintained and expanded independently, without directly affecting other modules in the system. For example, if at some point it is necessary to add new functionality or modify existing behavior, this update can be implemented in a specific module without the need to make global changes that could compromise the stability of the entire system.

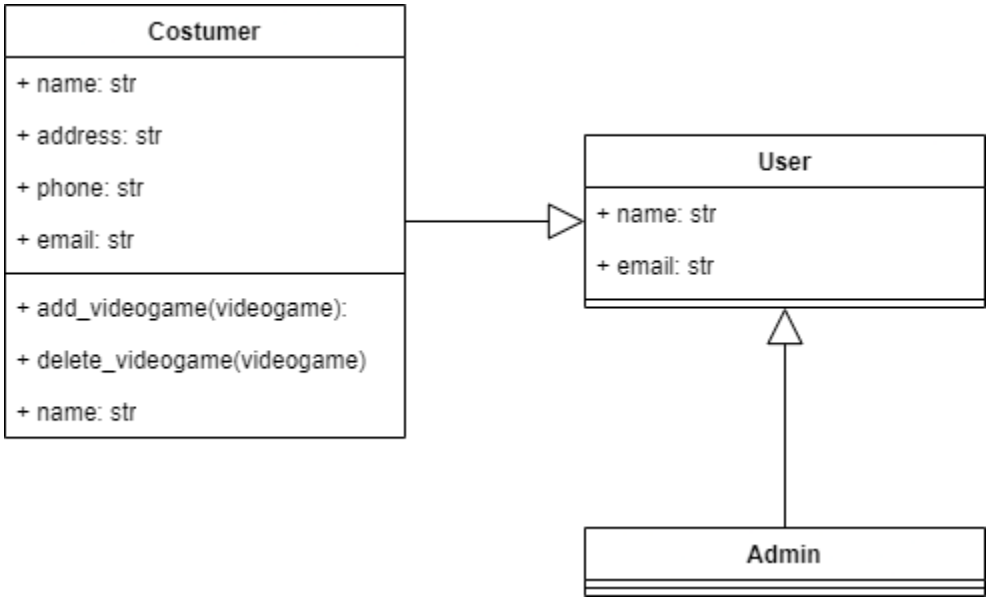
This level of modularity not only reduces potential errors by minimizing unwanted interactions between components, but also improves the system's ability to accommodate growth. As the system evolves, new modules can be integrated incrementally without having to redo substantial parts of the code base. In terms of scalability, OOP allows you to create robust systems prepared to handle a greater volume of data or interactions without compromising performance.

Methods:

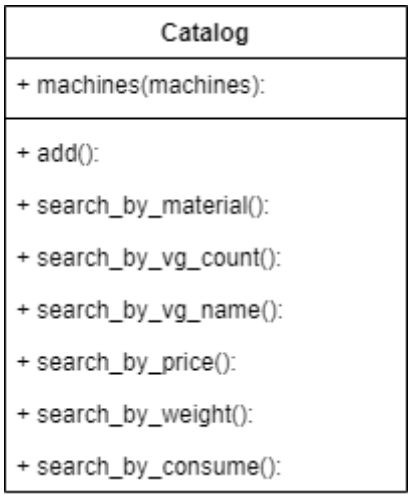
First, the class diagram was made, which allowed a clear abstraction of the problem and its possible solution. During this process, the methods and attributes that each class should contain were identified, according to their responsibility within the system, below is the class diagram:



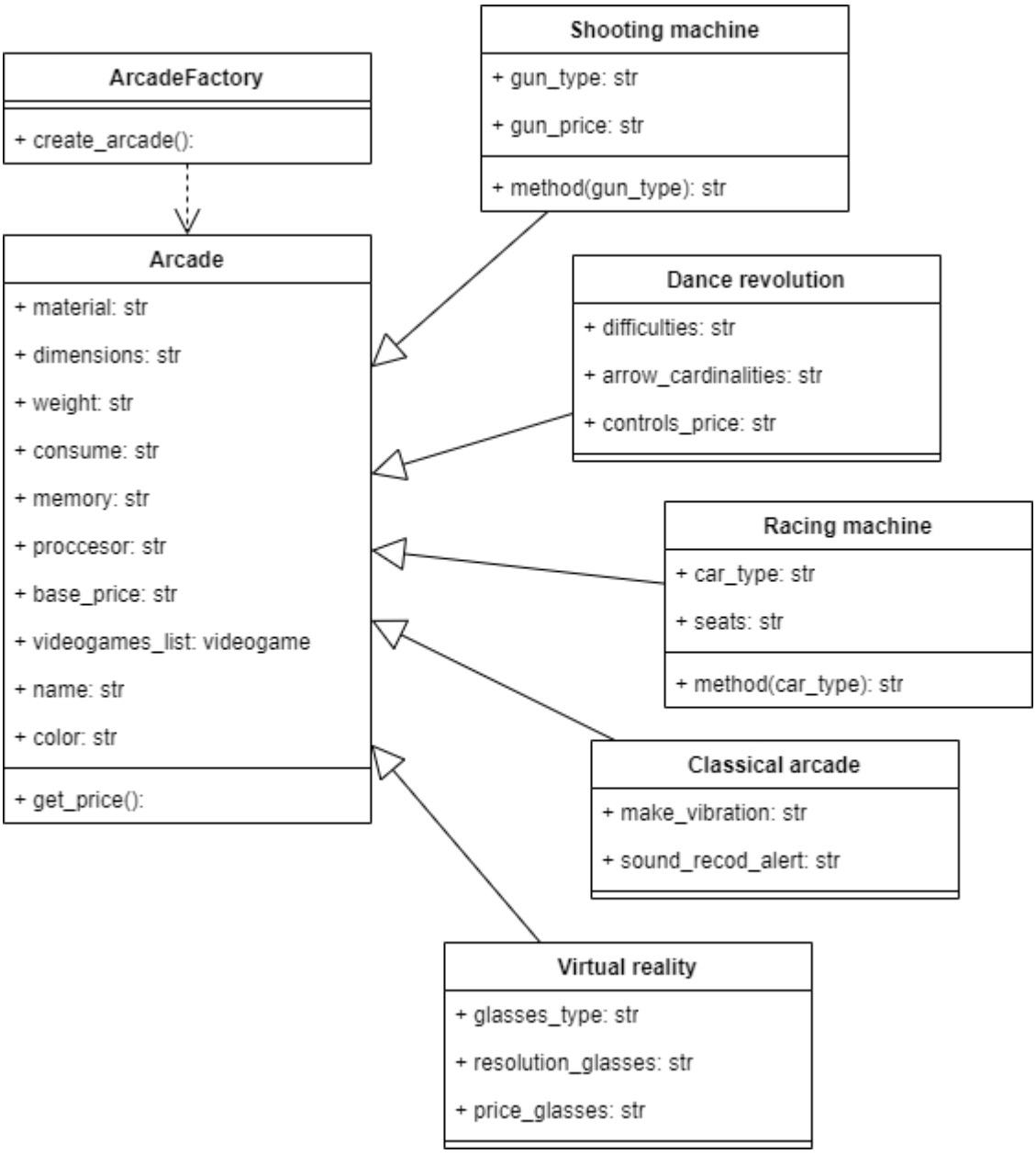
By having a modular division, we have the following modules:
User module:
He is in charge of everything related to buyers and administrators. His main functions include removing and adding video games to the machines, as well as purchasing them.



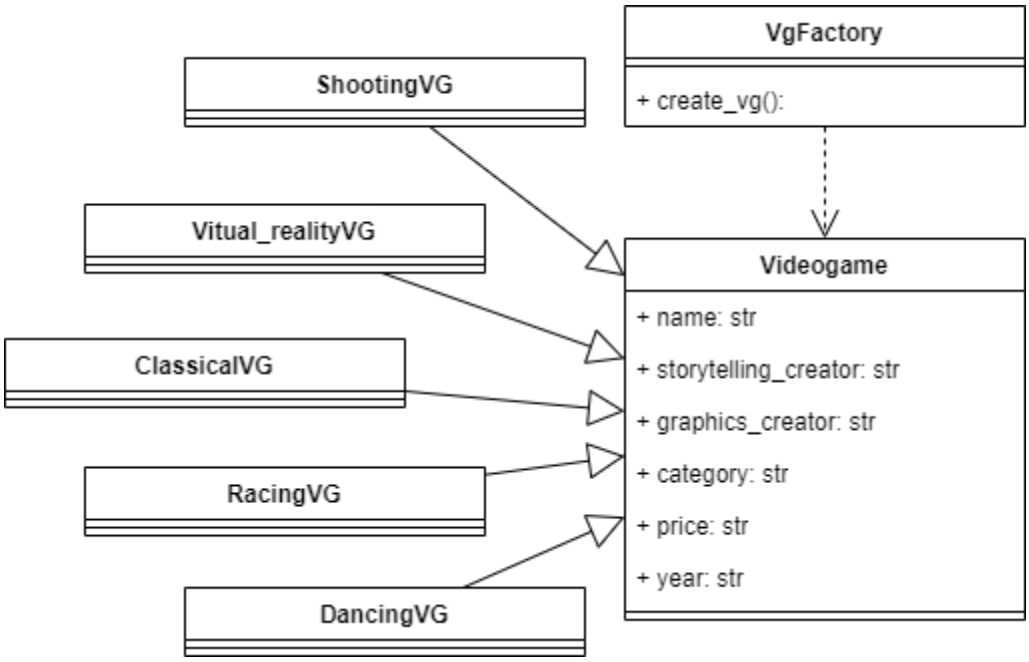
Catalog module:
It is responsible for containing all the information about existing machines, and also about video games.



Machines module:
He is in charge of managing the machines, as well as their creation, here are all the different types of machines that a user can buy.



Videogame module:
It is in charge of managing the video games, as well as their creation, here are all the different types of video games that a user can add to their arcade machine.



In terms of pattern design, the Factory creational pattern was chosen due to the modular and flexible structure of the system, in which we have two main abstract classes: Machines and Videogames. Each of these abstract classes has multiple subclasses that represent different specific types of machines and videogames. This creates the need for an efficient and decoupled way to create instances of these subclasses without the client code having to explicitly know which subclass needs to be instantiated.
The arcade and video game system are based on abstraction. The abstract class Machine can have several subclasses like Dance Revolution, Classic Machine, Racing Machine, etc. Each subclass has its own specific attributes and behaviors, but they all share a common structure defined in the abstract class. The same goes for Video Games, which can have different categories like racing games, shooters, or dance games, each with its own logic and features.