Technical report - Workshop 2

Julian David Pulido Carreño – code: 20231020064

Considerations:

In this project, the Builder pattern was implemented to manage the creation of different types of arcade machines, such as Dance Revolution machines, racing machines, classic arcade machines, virtual reality machines, and slot machines. This pattern was chosen due to the significant variety in attributes and functionalities among the machines.

Why use the Builder pattern?:

Arcade machines differ greatly in terms of structure (color, dimensions, power consumption, memory, processors, etc.), which would have made it very complex to handle their creation using a standard constructor or a pattern like Factory. The **Builder** pattern provides a modular and flexible way to construct complex objects step by step, allowing customization of only the relevant aspects without the need to create multiple constructors for each variation.

Advantages of the Builder pattern in this context:

Flexibility: The pattern allows for flexible creation of different types of arcade machines, as each machine can be built according to its specific characteristics without modifying the core code.

Separation of concerns: It facilitates the separation between the logic of machine creation and its final representation. Each builder specializes in a specific type of machine, keeping the code more organized and easier to maintain.

Readable and maintainable code: With a clear structure for step-by-step machine creation, the code becomes more understandable and easier to modify in case of future changes or the addition of new types of arcade machines.

Code reuse: Instead of duplicating code for different types of machines, much of the construction process can be shared, leveraging common interfaces or abstract classes.

Application of the Builder pattern in the class diagram:

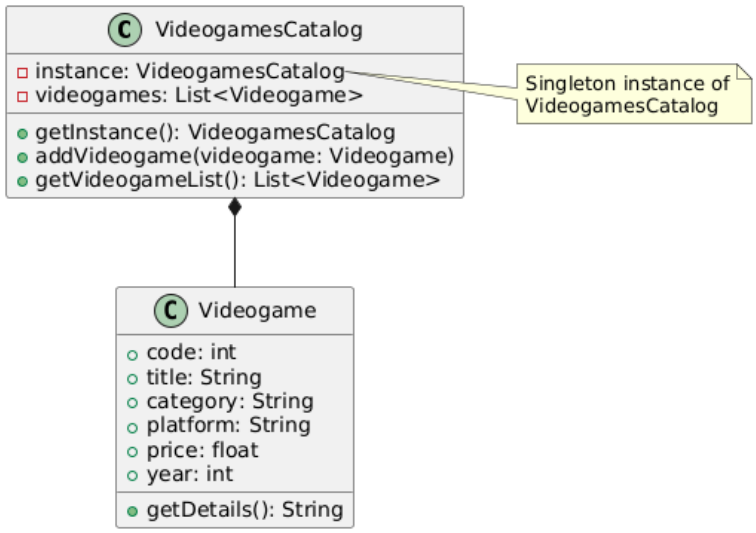
In the class diagram, each arcade machine type has its own concrete builder like ConcreteDanceRevolutionBuilder or ConcreteRacingMachineBuilder, which allows for the creation of specific machine instances from a common interface (`Builder`). Additionally, a `Director` object is implemented to orchestrate the building process of the machines, ensuring a clear and centralized logic for creating different machine variations.

Relationship with other patterns:

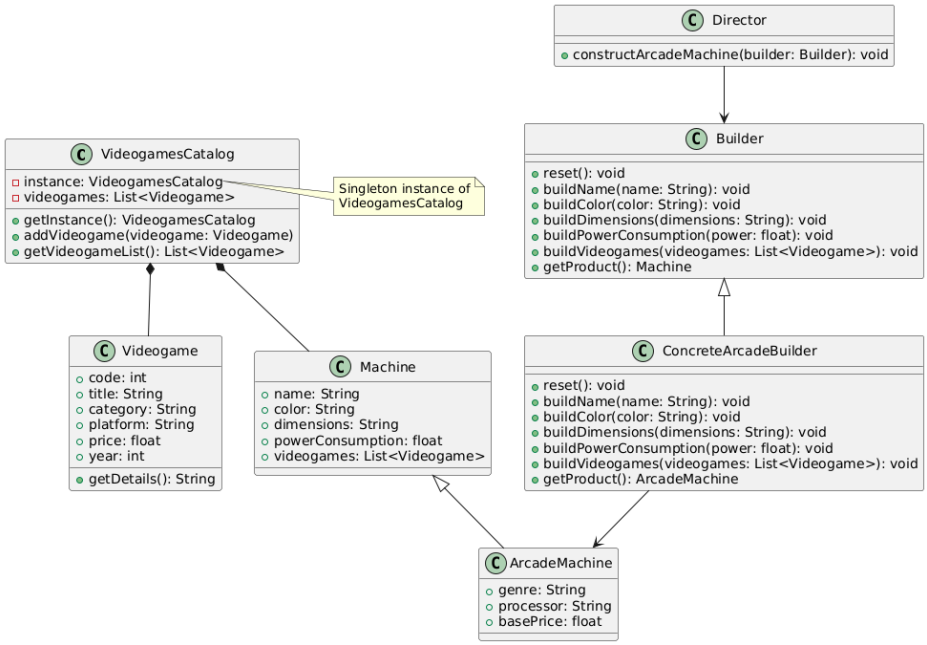
the **Singleton** pattern was also used in the `VideogamesCatalog` class to ensure that only one instance of this catalog exists throughout the system, providing controlled access and avoiding inconsistencies in the list of available video games.

This modular and design-pattern-oriented approach creates a scalable and maintainable system, aligned with SOLID principles, particularly the Single Responsibility Principle (SRP) and the Dependency Inversion Principle (DIP).

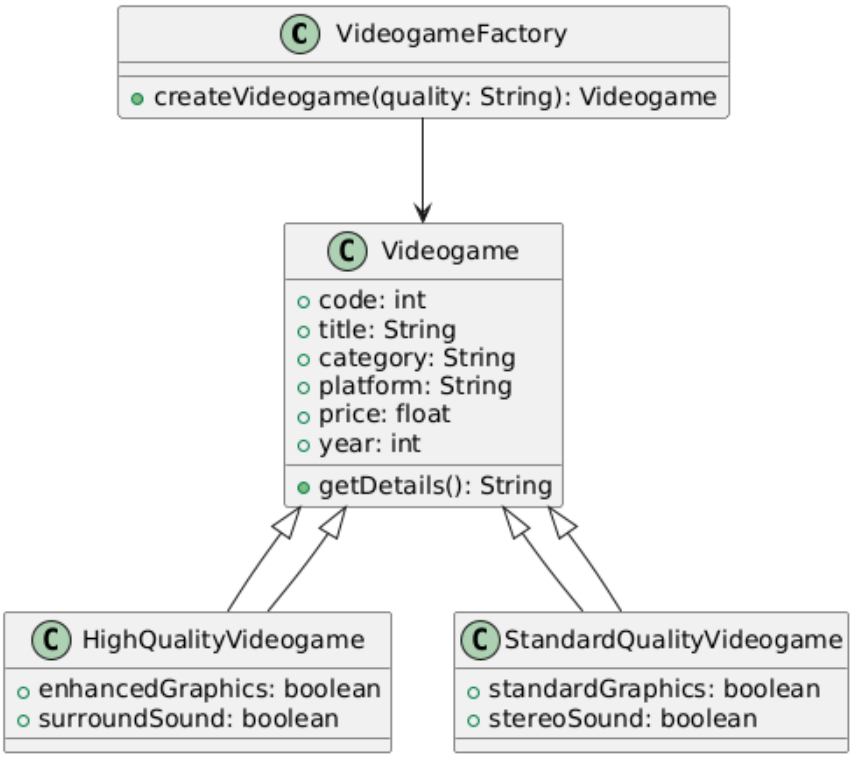
Singleton pattern videogames component

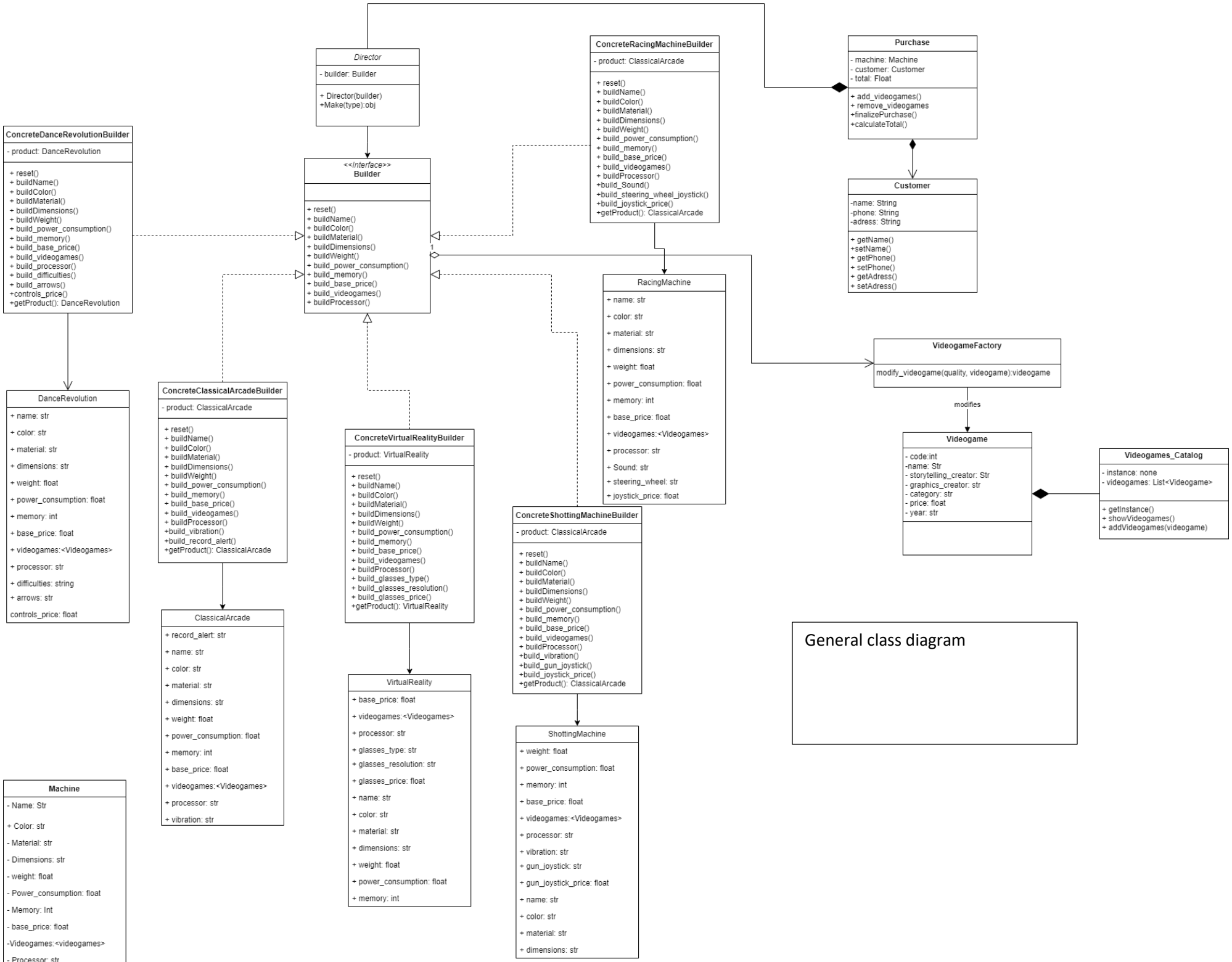


Resume of a builder pattern



Use of a factory pattern




Director
- builder: Builder
+ Director(builder)
+Make(type):obj
ConcreteRacingMachineBuilder
- product: ClassicalArcade
+ reset()
+ buildName()
+ buildColor()
+ buildMaterial()
+ buildDimensions()
+ buildWeight()
+ build_power_consumption()
+ build_memory()
+ build_base_price()
+ build_videogames()
+ buildProcessor()
+build_Sound()
+build_steering_wheel_joystick()
+build_joystick_price()
+getProduct(): ClassicalArcade
Purchase
- machine: Machine
- customer: Customer
- total: Float
+ add_videogames()
+ remove_videogames
+finalizePurchase()
+calculateTotal()
Customer
-name: String
-phone: String
-adress: String
+ getName()
+setName()
+ getPhone()
+ setPhone()
+ getAdress()
+ setAdress()
ConcreteDanceRevolutionBuilder
- product: DanceRevolution
+ reset()
+ buildName()
+ buildColor()
+ buildMaterial()
+ buildDimensions()
+ buildWeight()
+ build_power_consumption()
+ build_memory()
+ build_base_price()
+ build_videogames()
+ build_processor()
+ build_difficulties()
+ build_arrows()
+controls_price()
+getProduct(): DanceRevolution
<<Interface>>
Builder
+ reset()
+ buildName()
+ buildColor()
+ buildMaterial()
+ buildDimensions()
+ buildWeight()
+ build_power_consumption()
+ build_memory()
+ build_base_price()
+ build_videogames()
+ buildProcessor()
1
RacingMachine
+ name: str
+ color: str
+ material: str
+ dimensions: str
+ weight: float
+ power_consumption: float
+ memory: int
+ base_price: float
+ videogames:<Videogames>
+ processor: str
+ Sound: str
+ steering_wheel: str
+ joystick_price: float
VideogameFactory
modify_videogame(quality, videogame):videogame
modifies
Videogame
- code:int
-name: Str
- storytelling_creator: Str
- graphics_creator: str
- category: str
- price: float
- year: str
Videogames_Catalog
- instance: none
- videogames: List<Videogame>
+ getInstance()
+ showVideogames()
+ addVideogames(videogame)
DanceRevolution
+ name: str
+ color: str
+ material: str
+ dimensions: str
+ weight: float
+ power_consumption: float
+ memory: int
+ base_price: float
+ videogames:<Videogames>
+ processor: str
+ difficulties: string
+ arrows: str
controls_price: float
ConcreteClassicalArcadeBuilder
- product: ClassicalArcade
+ reset()
+ buildName()
+ buildColor()
+ buildMaterial()
+ buildDimensions()
+ buildWeight()
+ build_power_consumption()
+ build_memory()
+ build_base_price()
+ build_videogames()
+ buildProcessor()
+build_vibration()
+build_record_alert()
+getProduct(): ClassicalArcade
ConcreteVirtualRealityBuilder
- product: VirtualReality
+ reset()
+ buildName()
+ buildColor()
+ buildMaterial()
+ buildDimensions()
+ buildWeight()
+ build_power_consumption()
+ build_memory()
+ build_base_price()
+ build_videogames()
+ buildProcessor()
+ build_glasses_type()
+ build_glasses_resolution()
+ build_glasses_price()
+getProduct(): VirtualReality
ConcreteShottingMachineBuilder
- product: ClassicalArcade
+ reset()
+ buildName()
+ buildColor()
+ buildMaterial()
+ buildDimensions()
+ buildWeight()
+ build_power_consumption()
+ build_memory()
+ build_base_price()
+ build_videogames()
+ buildProcessor()
+build_vibration()
+build_gun_joystick()
+build_joystick_price()
+getProduct(): ClassicalArcade
ClassicalArcade
+ record_alert: str
+ name: str
+ color: str
+ material: str
+ dimensions: str
+ weight: float
+ power_consumption: float
+ memory: int
+ base_price: float
+ videogames:<Videogames>
+ processor: str
+ vibration: str
VirtualReality
+ base_price: float
+ videogames:<Videogames>
+ processor: str
+ glasses_type: str
+ glasses_resolution: str
+ glasses_price: float
+ name: str
+ color: str
+ material: str
+ dimensions: str
+ weight: float
+ power_consumption: float
+ memory: int
ShottingMachine
+ weight: float
+ power_consumption: float
+ memory: int
+ base_price: float
+ videogames:<Videogames>
+ processor: str
+ vibration: str
+ gun_joystick: str
+ gun_joystick_price: float
+ name: str
+ color: str
+ material: str
+ dimensions: str
Machine
- Name: Str
+ Color: str
- Material: str
- Dimensions: str
- weight: float
- Power_consumption: float
- Memory: Int
- base_price: float
-Videogames:<videogames>
- Processor: str
+ removeVideogame(int)
General class diagram