Okay, let's explore how to implement data validation within your Hexagonal Architecture using custom validation classes in the "Domain" layer.

**Key Concepts:**

* **Domain Layer:** This layer should contain all the core business logic, including data validation rules. By keeping validation logic within the domain, you ensure it's independent of external concerns like file formats or specific data sources.
* **Custom Validation Classes:** Create dedicated classes to encapsulate specific validation rules. This promotes code reusability and makes your validation logic more organized and maintainable.

**Example Structure:**

domain/  
├── entities/   
│ └── <YourEntityClass>.py   
│ class <YourEntityClass>:  
│ def \_\_init\_\_(self, ...):   
│ # ...  
│ def is\_valid(self) -> bool:   
│ # Implement validation logic here  
│ # Check data types   
│ # Perform range checks  
│ # Validate formats  
│ # ...  
│ return True/False   
├── value\_objects/  
│ └── <YourValueObject>.py  
│ class <YourValueObject>:  
│ def \_\_init\_\_(self, value):  
│ self.value = self.\_validate(value)   
│ def \_validate(self, value):  
│ # Implement validation logic for specific value   
│ return value

**Example Implementation:**

Python

# domain/value\_objects/name.py  
class Name:  
 def \_\_init\_\_(self, value):  
 self.value = self.\_validate(value)  
  
 def \_validate(self, value):  
 if not isinstance(value, str):  
 raise ValueError("Name must be a string.")  
 if not value.isalnum():  
 raise ValueError("Name must contain only alphanumeric characters.")  
 return value

**Benefits:**

* **Encapsulation:** Validation logic is encapsulated within the domain layer, improving code organization and maintainability.
* **Reusability:** Validation classes can be reused across different parts of your application.
* **Testability:** Validation logic can be easily tested in isolation.
* **Flexibility:** You can easily add or modify validation rules without affecting other parts of your application.

By creating custom validation classes within your "Domain" layer, you can establish a robust and maintainable data validation strategy within your Hexagonal Architecture.

**Next Steps:**

In the next step, we can discuss how to integrate these validation classes into your file processing pipeline and how to handle validation errors effectively.

I hope this helps! Feel free to ask if you have any further questions.

**Sources**

1. <https://github.com/Filip-Povidernyi/HW11>