**Clean architecture**

Clean architecture is a design philosophy that aims to create systems that are maintainable, scalable, and easy to test. It organizes code into layers, each with a specific responsibility, to achieve separation of concerns and minimize dependencies. The main layers in clean architecture are:

**1. Domain Layer**

* **Purpose:** Represents the core business logic and rules. It is independent of any external dependencies or frameworks.
* **Components:** Entities (objects with unique identities) and Value Objects (immutable objects with no identity).
* **Example:** In a banking system, the Domain Layer would include entities like Account and Transaction and business rules like transfer limits or account balance calculations.

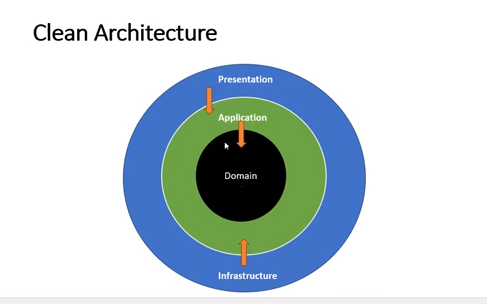
**2. Application Layer**

* **Purpose:** Coordinates the flow of data between the domain layer and other layers. It defines the operations (use cases) that the application can perform.
* **Components:** Use Cases (or Interactors) which execute specific business processes, and Services which handle application logic that doesn’t fit into the domain.
* **Example:** In a banking system, the Application Layer would handle use cases like CreateAccount, DepositMoney, and TransferFunds.

**3. Infrastructure Layer**

* **Purpose:** Contains implementation details such as data storage, external service communication, and other infrastructure concerns. It depends on both the application and domain layers.
* **Components:** Repositories (data access), external APIs, and frameworks or libraries for persistence, messaging, etc.
* **Example:** In a banking system, the Infrastructure Layer would include database implementations for storing account data, APIs for communicating with other banking services, and libraries for logging and authentication.

**Visualizing the Layers**

****

**Interaction Between Layers**

* **Domain Layer**: The most independent, containing only business logic.
* **Application Layer**: Mediates between the domain and infrastructure, orchestrating the use cases.
* **Infrastructure Layer**: Implements the technical details, such as databases and external services.

**Key Principles**

1. **Independence:** Each layer is independent and can be modified without affecting the others.
2. **Dependency Rule:** Higher-level layers (like the domain) should not depend on lower-level layers (like the infrastructure).
3. **Testability:** The separation of concerns makes the system easier to test.

By following clean architecture, you can build systems that are robust, adaptable to changes, and easier to maintain over time.