**Principles/Best Practices:**

1. Single responsibility – Each class/method should do one thing
2. Open/closed – Classes should be open for extension and closed for modification
3. Liskov substitution principle – Derived types should be substitutable with their base type.
4. Interface segregation
5. Dependency inversion - Depend on abstractions rather than concretions.
6. Tell don't ask - Tell an object what you want to know rather than asking for properties and determining that yourself
7. Hollywood principle – Don’t call us, we’ll call you
8. Reused abstraction principle – Only create abstractions when reuse is needed. Super classes have more than one subclass. Interfaces are implemented more than once. Abstract methods are overridden multiple times.
9. Fail Fast – Stop processing on failure.
10. Return early – Return from methods ASAP
11. Dry (once and only once) – Don’t repeat yourself
12. Loose coupling - Code to an interface, not an implementation
13. Abstract what changes – Don’t hard code configuration, data
14. Encapsulation - information hiding
15. Be consistent (naming, opposites)
16. Separation of concerns – UI, BL, DAL, SOA
17. Programming style as documentation – Comments and outside docs are a last resort
18. Turn implicit concepts into explicit (DDD)
19. Guard clause – Check up front before starting business logic
20. Law of Simplicity - The ease of maintenance of any piece of software is proportional to the simplicity of its individual pieces.
21. Outline rule – Code should read like an outline. Layers of knowledge.
22. Mayfly variables - Minimize variable lifetime
23. Maximize signal to noise – Remove anything superfluous. Format consistently.
24. Optimize horizontal and vertical whitespace
25. Favor composition over inheritance
26. Use ternary for simple conditionals
27. Compare and assign booleans implicitly
28. YAGNI – You ain’t gonna need it
29. Do the simplest thing that could possibly work

**Design Patterns:**

1. Domain event
2. Composite
3. Façade -
4. Flyweight - an object that minimizes memory use by sharing as much data as possible with other similar objects
5. Ravioli Code – Many small classes that do one thing well.
6. Strategy - Interchangeable algorithms
7. Decorator – Extend a class at runtime. Flexible alternative to sub classing
8. Singleton – Assure only a single instance is created
9. Null Object pattern – An object that represents a empty state for a given entity
10. Factory – Centralized point for object instantiation
11. Repository – Abstract away database and centralize data access
12. State (closely related to strategy) - objects for states
13. Virtual proxy
14. Parameter objects (fowler) – Replace long parameter list by passing an object.
15. Object pooling
16. Circuit breaker
17. Humble object – Lean object that delegates to another class
18. N-teir – Split application into physical layers
19. Active record – Object per DB row
20. Query by example – Instantiate an object with desired properties for filtering
21. Data Transfer Object (DTO) – Dumb data structure for passing around – no behavior.
22. Arrange act assert (unit testing) – Pattern for writing unit tests
23. Design by contract – Define what you expect/require up front
24. Aspect oriented