Design pattern and Architecture pattern training

Every Sat and Sunday (1 hour)

www.questpond.com

Guru Mantra of Architects	• How do human (devs) understand complex system? • Philosophy of decoupling, classification -- Abstraction. • Definition of good architecture: - Decoupling changes.
OOP , SOLID ,DI ,IOC concepts	• Why OOP? • Identify Entities (things) and Classify them. • Abstraction (Planning) thinking 30,000 feet. • Writing implementation for Abstraction. • Following Abstraction using Encapsulation. • Classification: - Polymorphism and Inheritance by product of Abstraction. • Classification, Families, Polymorphism. • Inheritance, creating families and abstract classes • Interface enforcing classification, abstract classes common code in inheritance. • Wrong classification, Wrong abstraction. interfaces and contract, LISKOV • Aggregation, Composition and Association. • SOLID principle understanding. • Dependency Injection and IOC. • Full decoupling Simple Factory pattern.
Domain Driven Development	• What is DDD ? • Importance of ubiquitous language. • Entity, Value objects and Services. • Every is bounded with a context.
Design Patterns	• What is Design pattern, who is GOF ? • Design pattern vs Architectural Pattern vs Style. • Creational, Structural and Behavioral. • GOF and NON-GOF patterns. • Aggregate root pattern. • Iterator pattern. • Extreme Complex Object creation:- Factory and Abstract Factory. • Builder Pattern • Prototype Pattern • Singleton Pattern • Adapter Pattern • Bridge Pattern • Composite Pattern • Decorator Pattern

	• Facade Pattetrn • Flyweight Pattern • Proxy Pattern • Mediator Pattern • Memento Pattern • Interpreter Pattern • Iterator Pattern • COR Pattern • Command Pattern • State Pattern • Strategy Pattern • Observer Pattern • Template Pattern • Visitor Pattern • Explain IOC(Inversion of Control)? • Dependency Injection • Fluent interface and method chaining • Lazy Loading • Explain RIP(Replace IF with Polymorphism) Pattern? • Immutable object design pattern • Explain Null Design Pattern and its usage. • CQRS • Repository

We need to create a customer management system for an Indian company which will help to main customer data, products bought by customer, discounts and delivery mechanism.

- System should be able to maintain customer data which includes name, age, geography and products bought.
- System should record for data for all types of Customer:-
    - Paid Customer who have bought products.
    - Enquiry Customer who have not bought products but done enquiry.
- System can have different discount calculation as per customer: -
    - Normal customer will not have any discounts.
    - If your age is above 60 , you get 10% senior discount.
    - If you buy on weekends, you get 2% discount.
    - Customers from Mumbai and Pune will get 1% discount.
- There are different delivery mechanisms of products to customer:-
    - Home delivery through Courier.
    - Pickup from shop.

Notes…

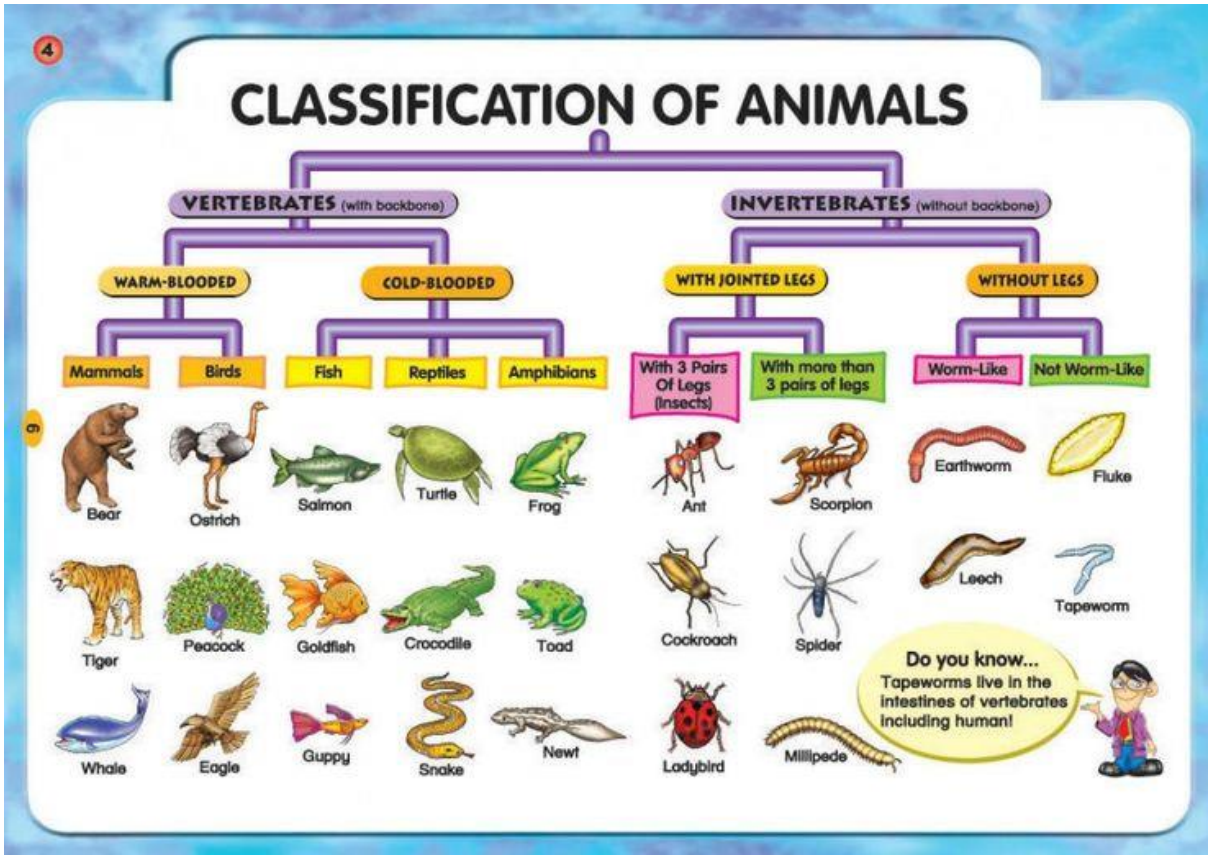
Philosophy: - Humans learn complex system by Entity Family classification.

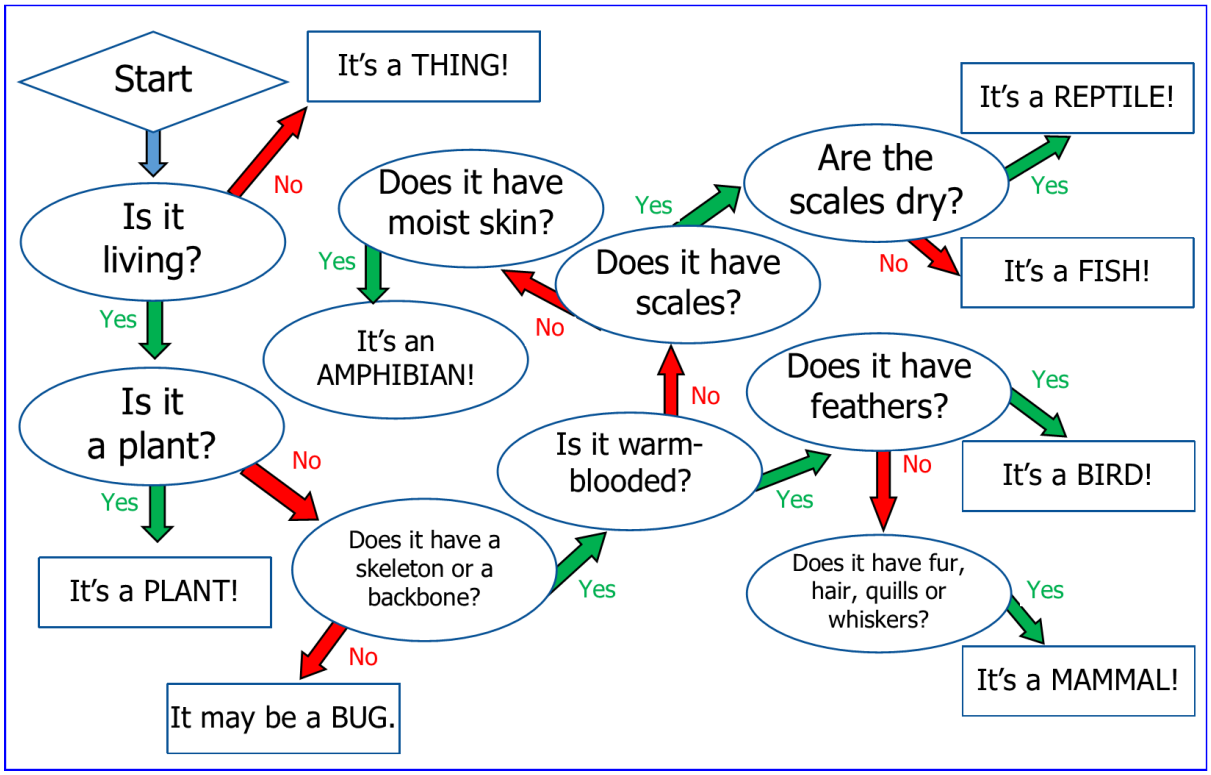
Achieve:- Change at one place you not need to make changes in lot of places. Less impact of changes.

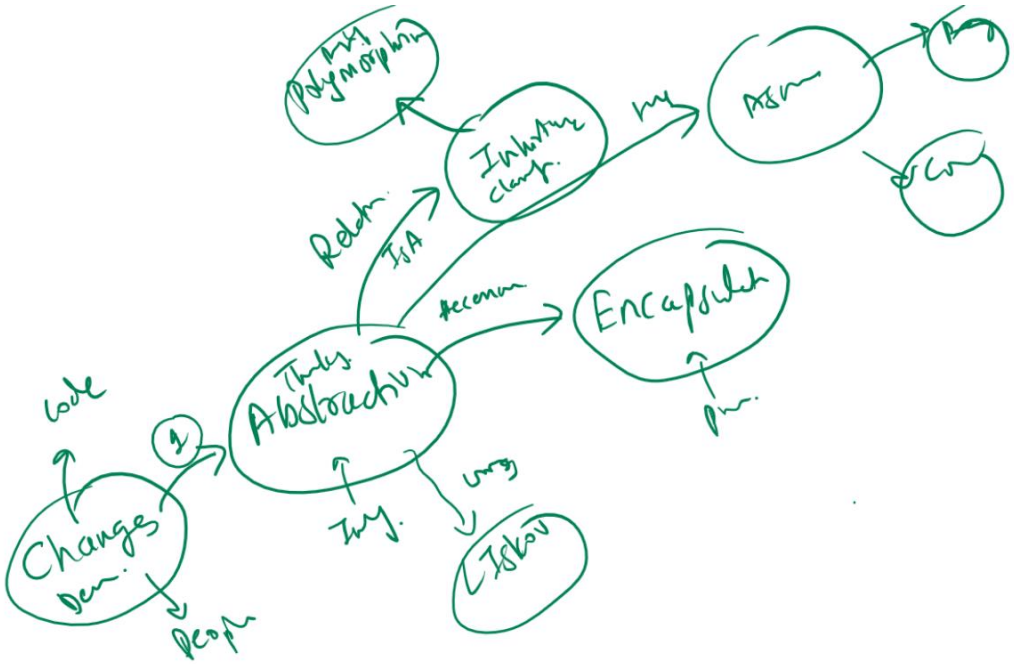
1. Identify Entity and do family classification (Abstraction).
2. Identify IS A and HAS A relationship between Entity abstraction.
3. Write implementation, respect abstraction by using encapsulation.
4. Use inheritance and Abstract classes with same family to share common logic.
5. Use proper Composition, Aggregation ,Association to implement HAS A.
6. Consume in client by using Object Polymorphism:- Decoupling.
7. Any wrong Abstraction identified during implementation Rethink Step 1 and 2.
8. SOLID :- SRP : Every class should have only intention…Overloading.
9. SOLID : - OCP : Open close principle…Open for extension close for modification.
10. SOLID : - LISKOV :- Child class should be able to substitute parent object with out any side effects.
11. LISKOV :- You cannot remove method from inheritance
12. SOLID :- ISP :- Interface Segregation principle. Do not force client to use method code which they do not want it.
13. SOLID : DIP :- Dependency Inversion :- Caller class should not call the callee concrete via interfaces. Higher level module and lower-level module should be talking via abstraction.
14. Interfaces is very helpful when you have complex classification of families..Uniformity…Interface
15. Centralize your object creation (Simple Factory pattern) and refer to abstraction in your client.
16. IOC : Inversion of control says uncessary task move from that class to another..
17. DI :- is process , its mechanism where depedent objects are injected through constructor or through propert and main class point to a abstraction.
18. DDD says follow the domain , follow UBL
19. Service , Entity and Value.
20. Bounded context
21. Aggregate root – Everything goes via the parent object…maintains integrity.
22. For iterating through the aggregate root you will need iterator pattern. Clone for maintaining integrity.
23. Iterator allows to just to enumerate and iternally we do not have to know the types.
24. Aggregate root overloads the root object….So be careful.
25. Bridge pattern decouples the abstraction from the implementation.
26. Repository pattern helps to decouple the model from the data access implementation.
27. Final call of CQRS goes to repository pattern.
28. Adapter is a wrapper on the top of classes which can not modified…. And they have incompatible. Incompatible interfaces..
29. Adpater object adapter and class adapter…Most used is the object adapter..
30. Sticking Reusability toooooo much can lead to complicated classes , classes which do SRP.
31. A model and command class are not same.
32. CQRS : Separate Command and Query
33. CQRS gives more structure as compared to reporting , DTO , POCO
34. Command Query , CommandHandler , QueryHandler , Repository , Aggregate Root , Event Sourcing , Projected Data …
35. Aggregate root is not a compulsion. Many times a normal proper design domain objects.
36. Event sourcing has collections of events what happened when the command fired.

37. Aggregate root , collection of objects needed during that command and the contained objects should be modified through the single root.
38. Event sourcing, AR not compulsory but if you doing MS yes they are very much needed
39. Projection is nothing it's a process of converting a model in to another structure.
40. Façade Pattern :- Simplifies a subsystem.
41. Simple Factory :- centralizes object creation (Container)
42. Replace If with polymorphism :- create collection like dictionary and do look up.
43. Factory pattern :- inheritance to complex obect creation , permutation and combination
44. Abstract factory : Stands on the top of factory
45. Template pattern :- Fixed steps , inherit and change particular step in the child class.
46. Bridge pattern :-
47. Decorator pattern :-
48. Singleton , Composite, Template,Decorator,Flyweight and Prototype


AQUATIC ANIMALS
SQUID
SEAL
OCTOPUS
TURTLE
CUTTLEFISH
STARFISH
CLOWNFISH
SNAIL
JELLYFISH
ANGELFISH


CLASSIFICATION OF ANIMALS
VERTEBRATES (with backbone)
INVERTEBRATES (without backbone)
WARM-BLOODED
COLD-BLOODED
WITH JOINTED LEGS
WITHOUT LEGS
Mammals
Birds
Fish
Reptiles
Amphibians
With 3 Pairs Of Legs (Insects)
With more than 3 pairs of legs
Worm-Like
Not Worm-Like
Bear
Ostrich
Salmon
Turtle
Frog
Ant
Scorpion
Earthworm
Fluke
Tiger
Peacock
Goldfish
Crocodile
Toad
Cockroach
Spider
Leech
Tapeworm
Whale
Eagle
Guppy
Snake
Newt
Ladybird
Millipede
Do you know...
Tapeworms live in the intestines of vertebrates including human!





1. Most of their Species live in water and some of them live on the land

2. They have paired and unpaired fins which help them to swim.

3. They have either webbed limbs or limbs are modified to paddles for swimming.

4. Their body shape is streamlined and their bones are light and spongy.

5. The skull undergoes modification to form a slender snout.

6.The neck is reduced and external ears are disappeared.