Object Oriented Design Interview

Analyzing and Designing a System :

1. Identifying the objects in a system.
2. Defining relationships between objects.
3. Establishing the interface of each objects; and
4. Making a design, which can be converted to executables using OO language.

Benefits of using UML (Unified Modeling Language) :

1. Helps develop a quick understanding of a software system.
2. UML modeling helps in breaking a complex system into discrete pieces that can be easily understood.
3. To communicate design decisions.
4. Easier to abstract out concepts.
5. Easier to hand over the system to new teams.

Structural UML diagrams :

1. Class diagram -> Used to define structure and behavior in the use cases, a conceptual model of the system in terms of entities and their relationships.
2. Object diagram
3. Package diagram
4. Component diagram
5. Composite Structure diagram
6. Deployment diagram
7. Profile diagram

Behavioral UML diagrams :

1. Use case diagram -> Used to describe a set of user scenarios, illustrates the functionality provided by the system.
2. Activity diagram -> Used to model the functional flow-of-control between two or more class objects.
3. Sequence diagram -> Used to describe interactions among classes in terms of an exchange of messages over time.
4. State diagram
5. Communication diagram
6. Interaction overview diagram
7. Timing diagram.

ClassDiagram : Backbone of object oriented modeling. (Refer to ClassDiagram.pmg under Images)

How different entities (people, things, data) relate to each other.

1. Analyse and Design of the STATIC view of an application
2. Describe responsibilities of a system
3. Base for Component and Deployment diagrams
4. Forward and Reverse engineering.

ClassName | Class Properties | Class Methods.

Relationships :

- Association : Either bi-directional or uni-directional. Link between classes.
- Multiplicity : Indicates how many instances of a class participate in the relationship.
- Aggregation : The lifecycle of a PART class is independent of the WHOLE class's lifecycle. A relationship where the child can exist independently of the parent.
- Composition : Child class's instance lifecycle is dependent on the parent class's instance lifecycle. Child can not exist independently of the parent.
- Generalization : Think of it as interface in java world.
- Abstract Class : Same as in java.

Conventions : (Refer to UMLConventions.pmg under images)

<<Interface>> NAME | methods
Class Name | properties(variables) | methods

A -------|> B : Generalization : A implements B.
A _______|> B : Inheritance : A inherits from B.
A ---------- B : Use Interface : A uses interface B.
A __________ B : Association : A and B call each other.
A __________> B : Uni-directional Association : A can call B but not vice versa.
A <>_______ B : Aggregation : A "has-an" instance of B. B can exist with out A.

A <*>_____ B : Composition : A "has-an" instance of B. B can not exist with out A.

Encapsulation --> Binding data together and hiding it from outside world.
Abstraction --> Hiding internal implementation and only revealing operations that are relevant to other objects.
Inheritance --> Subclass, Interfaces.
Polymorphism --> Overloading and Overriding methods.

Use Case Diagram : (Refer to diagram UseCaseDiagram.png under images folder)

Actors and set of actions that they perform.

Describe high-level functional behavior of the system.
What system does from the user point of view.
What system does and what it does not do.

Visualize functional requirements of the system..

Components of UseCaseDiagram :

1. System Boundary
2. Actors
3. Use Case -> Every business functionality is a potential use case.
4. Include -> Include relationship (invocation of one use case by another use case.)
5. Extend -> Extending one use case from other user case. (E.g: SearchProductByName & ByCategory extends SearchProduct.

Sequence Diagram :

Interaction among classes in terms of an exchange of messages over time.
To explore logic of complex operations, functions or procedures.

Identify each class instance by putting each class instance inside a box.
If class instance sends a message to another class, draw a line with open arrowhead (=>) pointing to the receiving class instance and place name of the message above the line.
A dotted line with an arrowhead pointing back to represent returned value (<==)

Activity Diagram :

Illustrate flow of control in a system.
Emphasizes condition of the flow and the sequence in which it happens.
Also to refer to the steps involved in the execution of a use case.

Model flow of control from activity to activity.
Model workflow or business process and internal operations.

User performing online shopping diagram is under ActivityDiagram.png under images.

SOLID - Design by Sandi Metz :

Issues with code with out design :

1. Rigid --> Every change causes a cascade of related changes.
2. Fragile --> Distant and unrelated code breaks after every change.
3. Immobile --> Can not reuse as code is hopelessly entangled.
4. Viscosity --> Behaving bad and not adhering to the original design.

SOLID :

1. **S**olid Responsibility -- There should never be more than one reason for a class to change.
2. **O**pen/Closed -- A module must be open for extension and closed for modification.
3. **L**iskov Substitution -- Subclasses should be substitutable for their base classes. (avoid isInstanceOf)
4. **I**nterface Segregation -- Many client specific interfaces are better than one general purpose Interface.
5. **D**ependency Inversion -- Depend up on abstractions and not concretions.

DESIGN IS ALL ABOUT DEPENDENCIES.

To avoid dependencies code must be :

1. Loosely coupled --> Dependency Injection.
2. Highly Cohesive --> A class should all be about the same thing. (Single Responsibility)
3. Easily Composable -->
4. Context Independent

```
DownloadParseUpdateFile class:
    run() {
        download_file()
        parse_file()
        update_file()
    }
    download_file() {
        File temp = new File("filename")
        FTP.open('localhost',
'user','pwd')
        FTP.getFile('remote file path',
temp)
    }
    parse_file() {
        read(temp)
    }
    update_file() {
        createcontent(file)
    }
END CLASS
```

Problems in the Left side ??

This code do not tolerate change.
What is ftp host/login/password change ?
What if I need to create another job like this ?
What if I dont want to ftp a file in every test ?