**Design Patterns**

**Problem # 1 –** Testing the chain of responsibility design pattern

You have to design a cash dispensing machine:

1. Let’s say there are $**50**, $**20** and $**10** notes
2. The user enters the amount to be dispensed and the machine dispenses amount in terms of defined currency bills such as $50, $20, $10 etc.

For example - If user wants to dispense 50$, one 50$ note should be dispensed and NOT two 20$ and one 10$.

1. If the user enters an amount that is not multiples of 10, it throws error.

Answer: You can use chain of responsibility design pattern.

-------------------------------------------------------------- ---------------------------------------------------------------------------

**Problem# 2 –** Testing the Strategy design pattern

Strategy pattern is used when we have multiple algorithms for a specific task and client decides implementation to be used at runtime.

Let’s say, you have to design a shopping cart for an online portal. Online portal supports multiple payment methods (a) Credit Card (b) Debit Card (c) Paypal (d) Net Banking etc

Client, during the checkout time will decide which payment method to use. How would you design such a shopping cart ?

CreditCardStrategy  
DebitCardStrategy  
PaypalStrategy  
NetbankingStrategy

Item item1 = new Item("1234",10);  
Item item2 = new Item("5678",40);

ShoppingCart cart = new ShoppingCart();  
  
cart.addItem(item1);  
cart.addItem(item2);

//pay by paypal  
cart.pay(new PaypalStrategy("myemail@example.com", "mypwd"));

Question 2.1 – What kind of deign pattern java’s

Arrays.sort(arryObj, sortStrategyAlgo) uses ????

Strategy Design Pattern  
------------------------------------------------------------------------------------------------------------------------------------------

Problem # 3 – What is Flyweight Design Pattern. Can you tell me a scenario where you would use it ?

In my usecase, I have to create plenty of objects

Problem# 3.1 – What is the underlying design pattern being used in the String class implementation. String objects are pooled. For example, String str = “MyString”

------------------------------------------------------------------------------------------------------------------------------------------

Calendar is an abstract class. The getInstance() method tries to instantiate GregorianCalendar()   
i.e. Parent class is instantiating a drived class. This looks like a non-Object oriented design ??? Can you explain why it is so???   
Calendar class is an abstract class. This is true. However the point is that the getInstance() method returns the Calendar using the default timezone and locale. In this case, it is GregorianCalendar which IS A Calendar. This follows the “IS-A” relationship which is perfect object oriented design.So what you get from getInstance() is a class that does some specialized work based on default locale.

------------------------------------------------------------------------------------------------------------------------------------------

**Adapter Pattern**

What is the intent of the Adapter pattern?  
The intent of the Adapter is to match an existing object that is beyond your control to a particular interface

What are the two variations of the Adapter pattern?  
Object Adapter: relies on one object to contain the other object.  
Class Adapter: uses multiple inheritance to provide the interface.

------------------------------------------------------------------------------------------------------------------------------------------

**Abstract Factory**

What are the three key strategies in the Abstract Factory?

Find what varies and encapsulate it  
Favor aggregation over inheritance  
Design to interfaces, not to implementations  
------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**Decorator Design Pattern**

Give me an example of Decorator design pattern used in Java?

Stream IO  
------------------------------------------------------------------------------------------------------------------------------------------

**Observer Design Pattern**

What is the intent of the Observer pattern?  
Intent of the Observer pattern is to "define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically

Can you give me some real life example of Observer Pattern ?  
Radio station: It broadcasts its signal; anyone who is interested can tune in and listen when they want to

------------------------------------------------------------------------------------------------------------------------------------------

**OOAD & Design Principles**

Define "coupling" and "cohesion". What is "tight" coupling?

Favor composition over inheritance

------------------------------------------------------------------------------------------------------------------------------------------

**SOA**

Problem # : What is SOA ?

Common pitfalls for SOA ?

What is SOA governance? What are its functions?  
Service-Oriented Architecture (SOA) governance is a concept used for activities related to exercising control over services in an SOA Some key activities that are often mentioned as being part of SOA governance are:

Managing the portfolio of services: This includes planning development of new services and updating current services.

Managing the service lifecycle: This is meant to ensure that updates of services do not disturb current services to the consumers. Using policies to restrict behavior: Consistency of services can be ensured by having the rules applied to all the created services. Monitoring performance of services: The consequences of service downtime or underperformance can be severe because of service composition. Therefore action can be taken instantly when a problem occurs by monitoring service performance and availability.

What is service end point virtualization?

What is a XA data source? How it differs from a non-XA data source?  
 An XA transaction involves a coordinating transaction manager, with one or more databases (or other resources, like JMS) all involved in a single global transaction. Non-XA transactions have no transaction coordinator, and a single resource is doing all its transaction work itself (this is sometimes called local transactions).

Tell me some of the important characteristics of ESB ?  
**Invocation**: – It support for synchronous and asynchronous transport protocols, service mapping (locating and binding).

**Routing**: – Routing means addressability, static/deterministic routing, content-based routing, rules-based routing, and policy-based routing.

**Mediation**: – In metadata and data warehouse, a data transformation convert’s data from a source data format into destination data. Data transformation can be divided into two steps:

* Data mapping maps data elements from the source to the destination and captures any transformation that must occur.
* Code generation that creates the actual transformation program.

The functions of mediation are adapters, protocol transformation and service mapping.

**Process choreography**: – It is the implementation of complex business processes.

**Service orchestration**: – It is the coordination of multiple implementation services exposed as a single, aggregate service.  
  
**Complex event processing**: – The functions are event-interpretation, correlation and pattern-matching.

**Other quality services**:- It ensure security (encryption and signing), reliable delivery,transaction management

**Management**: – Services provide the management categories are monitoring, audit, logging, metering, admin console, BAM (BAM is not a management capability in other words the ESB doesn’t react to a specific threshold. It is a business service capability surfaced to end users. ).

**Agnosticism**: – In general agnosticism to operating-systems and programming-languages; for example, it should enable interoperability between Java and .NET applications.

**Protocol Conversion/Bridging**: - comprehensive support for topical communication protocols service standards.

**Message Exchange Patterns**:- It support for various MEPs (Message Exchange Patterns) (for example: synchronous request/response, asynchronous request/response, send-and-forget, publish/subscribe)

**Adapters**:- Adapters helps for supporting integration with legacy systems, possibly based on standards such as JCA

**Security**: – It is a standardized security-model to authorize, authenticate and audit use of the ESB.

**Transformation**:-It is the facilitation of the transformation of data formats and values, including transformation services (often via XSLT or XQuery) between the formats of the sending application and the receiving application

**Validation**:-Validation against the schemas for sending and receiving messages.

**Governance**: – It is the ability to apply business rules uniformly.

**Enrichment**:- It enriching messages from other sources

**Split and Merge**: – The splitting and combining of multiple messages and the handling of exceptions.

**Abstraction**: – The provision of a unified abstraction across multiple layers.

**Routing and Transformation**: – Routing or transforming messages conditionally, based on a non-centralized policy (without the need for a central rules-engine).

**Queuing and staging**: – Queuing, holding messages if applications temporarily become unavailable or work at different speeds.

**Commodity Services**: – Provisioning of commonly used functionality as shared services depending on context.

-------------------------------------------------------------------------------------------------------------------------------  
**UML**

What are the basic elements of a sequence diagram?  
The basic elements are:

* Boxes. These are shown at the top and represent the objects that are interacting.  
  Name in the form objectname**:**classname. The object name is optional.
* Dashed vertical lines, also known as swim lanes, one for each object, to indicate time.
* Arrows, may be horizontal or vertical, showing the interaction between objects. Each arrow is labeled to describe the interaction
* Notes. This is optional

-------------------------------------------------------------------------------------------------------------------------------  
**Core Java**

How would you design a HashMap?

-------------------------------------------------------------------------------------------------------------------------------  
**Web Service**

What is RESTful service  
REST defines a set of architectural principles by which you can design Web services that focus on a system’s resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages

REST principles  
Use HTTP methods explicitly.  
Be stateless.  
Expose directory structure-like URIs.  
Transfer XML, JavaScript Object Notation (JSON), or both.

Advantages of REST  
Lightweight not a lot of extra xml markup  
Human Readable Results  
Easy to build no tool-kits required  
  
When should use RESTFUL webservice?  
(a) The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.  
  
(b) If the data that the web service returns is not dynamically generated and can be cached  
  
(c) The service producer and service consumer have a mutual understanding of the context and content being passed along. Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it meaningfully. In the real world, most commercial applications that expose services as RESTful implementations also distribute so-called value-added toolkits that describe the interfaces to developers in popular programming languages

-------------------------------------------------------------------------------------------------------------------------------  
**Spring**

-------------------------------------------------------------------------------------------------------------------------------  
**Hibernate**

-------------------------------------------------------------------------------------------------------------------------------  
**SQL & Databases**