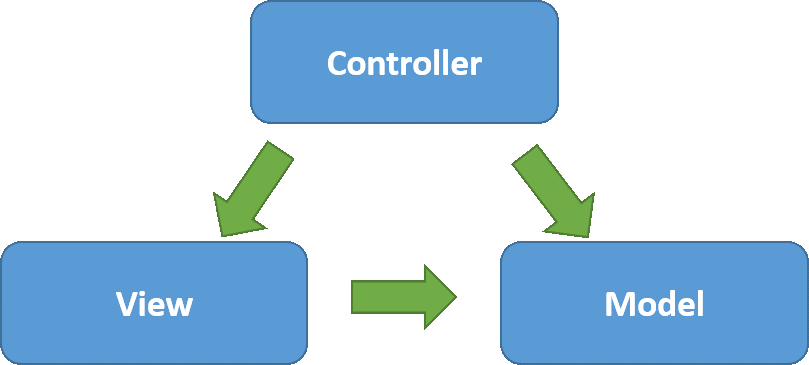
1. Demonstrate Model View Controller(MVC) architecture in JavaScript. Use AngularJs

framework to explain those.

**Model view controller**

In the MVC pattern, the different aspects of the application are broken into components to separate responsibilities. The Model contains the data and logic, the View contains the visual layout and presentation, while the Controller connects the two.



In AngularJS the MVC pattern is implemented in JavaScript and HTML. The view is defined in HTML, while the model and controller are implemented in JavaScript

1. Give real world examples of when to use an ArrayList and when to use LinkedList.(5marks). Demonstrate those using a java program.

Accessing elements are faster with ArrayList, because it is index based.

But accessing item is difficult with LinkedList. It is slow access. This is

to access any element, you need to navigate through the elements one by

one. But insertion and deletion is much faster with LinkedList, because

if you know the node, just change the pointers before or after nodes.

Insertion and deletion is slow with ArrayList, this is because, during

these operations ArrayList need to adjust the indexes according to

deletion or insetion if you are performing on middle indexes. Means,

an ArrayList having 10 elements, if you are inserting at index 5, then

you need to shift the indexes above 5 to one more.

1. Highlight the benefits of using spring and what subprojects do you know?

Benefits of spring include:

* **Lightweight:** there is a slight overhead of using the framework in development
* **Inversion of Control (IoC):** Spring container takes care of wiring dependencies of various objects, instead of creating or looking for dependent objects
* **Aspect Oriented Programming (AOP):** Spring supports AOP to separate business logic from system services
* **IoC container:** it manages Spring Bean life cycle and project specific configurations
* **MVC framework:** that is used to create web applications or RESTful web services, capable of returning XML/JSON responses
* **Transaction management:** reduces the amount of boiler-plate code in JDBC operations, file uploading, etc., either by using Java annotations or by Spring Bean XML configuration file
* **Exception Handling:** Spring provides a convenient API for translating technology-specific exceptions into unchecked exceptions

Spring subprojects include:

* **Core** – a key module that provides fundamental parts of the framework, like IoC or DI
* **JDBC** – this module enables a JDBC-abstraction layer that removes the need to do JDBC coding for specific vendor databases
* **ORM integration** – provides integration layers for popular object-relational mapping APIs, such as JPA, JDO, and Hibernate
* **Web** – a web-oriented integration module, providing multipart file upload, Servlet listeners, and web-oriented application context functionalities
* **MVC framework** – a web module implementing the Model View Controller design pattern
* **AOP module** – aspect-oriented programming implementation allowing the definition of clean method-interceptors and pointcuts

1. IntelliSoft Kenya Limited urgently needs a system to manage their employees. The

system should collect names, date of birth, unique identifier, address, contract

information(when signed, expiry date and type). You are required to model this

system using UML, OOP principles, flow diagrams. Provide a practical solution through java code, demonstrate industrial coding accepted styles

1. What is Dependency Injection?

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept stating that you do not create your objects manually but instead describe how they should be created. An IoC container will instantiate required classes if needed.

And how can we inject beans in Spring and how is it related to Inversion of Control(IoC)? Explain clearly using code examples

A few different options exist for injecting beans in Spring:

* Setter Injection
* Constructor Injection
* Field Injection

The configuration can be done using XML files or annotations.

**IOC** (**Inversion of control**) is a general parent term while DI (Dependency injection) is a subset of **IOC**. **IOC** is a concept where the flow of application is inverted. DI provides objects that an object needs. So rather than the dependencies construct themselves they are **injected** by some external means

Code url <https://www.vogella.com/tutorials/SpringDependencyInjection/article.html>

1. Project X has so many methods(around 1000) that commit and write information to

the database, all methods have been prefixed with the word save. The requirement

is before any of these methods are called, a check must be performed to see if the

parameters passed are valid. By use of an aspect, demonstrate through code on how that could be realized with fewer lines of code and in the most efficient way

1. What are the important benefits of using Hibernate Framework? Using examples,

show how Hibernate and Java persistence API can be used together with spring

framework

## ORM

Hibernate ORM easily solves the data mismatch found between the object oriented classes of an application and relational database. ORM connects these two with ease through the use of the XML mapping file. It enables to gain complete control over the application as well as the database design. This feature makes Hibernate flexible and powerful.

## Transparent Persistence

Hibernate’s transparent persistence ensures the automatic connection between the application’s objects with the database tables. This feature prevents developers from writing lines of connection code. Transparent persistence enables hibernate to reduce development time and maintenance cost.

## Database independent

Hibernate is database independent. It can be used to connect with any database like Oracle, MySQL, Sybase and DB2 to name a few. This cross database portability of Hibernate is easily achieved by changing a parameter ‘database dialect’ in the configuration file. Database independency is considered as one of the major advantages of Hibernate.

## HQL

Hibernate supports a powerful query language called HQL (Hibernate Query Language). This query language is more powerful than SQL and is completely object oriented. HQL’s advanced features like pagination and dynamic profiling are not present in SQL.  
HQL can be used to implement some of the prominent object oriented concepts like inheritance, polymorphism and association.

## Dual-layer Caching

Hibernate supports both first level and second level caching mechanisms. The first level caching is associated with Session object which is used by default. The second level caching is related to the Session Factory object.  
Through caching concept, Hibernate retains the objects in cache to reduce repeated hits to the database. This feature makes Hibernate highly scalable and optimizes the application’s performance.

## Version Property

Hibernate supports optimistic locking through its version property feature. This functionality supports multiple transactions without affecting one another.  
For example, when two or more users try to alter a database entity at the same time, the version field avoids the conflict and gives preference to the user who commits the changes first. The other user will be prompted with an error message and will be asked to restart the process.

## Scalability

Hibernate is highly scalable. It adapts itself in any environment. It may be an intranet application with few hundreds of users or large critical application with thousands of users. Hibernate supports both the applications equally.

## Lazy-Loading

The lazy-loading concept fetches only the necessary object that is required for the execution of an application.

For example, if there is one parent class and n number of child classes, during execution, there is no need to load all the child classes. Instead, only the class that is required for the query or join need to be loaded. This concept of lazy-loading prevents the unnecessary loading of objects. It enhances the performance of the application.

1. Demonstrate encapsulation, polymorphism and show how multiple inheritance can be achieved and supported in Java

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations.

public class BasicCoffeeMachine {

// ...

public Coffee brewCoffee(CoffeeSelection selection) throws CoffeeException {

switch (selection) {

case FILTER\_COFFEE:

return brewFilterCoffee();

default:

throw new CoffeeException(

"CoffeeSelection ["+selection+"] not supported!");

}

}

public List brewCoffee(CoffeeSelection selection, int number) throws CoffeeException {

List coffees = new ArrayList(number);

for (int i=0; i<number; i++) {

coffees.add(brewCoffee(selection));

}

return coffees;

}

// ...

}

Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Example

// First Parent class

class Parent1

{

    void fun()

    {

        System.out.println("Parent1");

    }

}

// Second Parent Class

class Parent2

{

    void fun()

    {

        System.out.println("Parent2");

    }

}

// Error : Test is inheriting from multiple

// classes

class Test extends Parent1, Parent2

{

   public static void main(String args[])

   {

       Test t = new Test();

       t.fun();

   }

}

1. A general Java Enterprise Web Application is broken down into layers; Database

Implementation, DAO, Domain Model, Service Layer and Web Layer. What is the

benefit of this layered approach? Demonstrate that layered architecture using Spring

framework

A Layered Architecture, as I understand it, is the organization of the project structure into four main categories: **presentation, application, domain, and infrastructure**. Each of the layers contains objects related to the particular concern it represents.

* **The presentation layer** contains all of the classes responsible for presenting the UI to the end-user or sending the response back to the client (in case we’re operating deep in the back-end).
* **The application layer** contains all the logic that is required by the application to meet its functional requirements and, at the same time, is not a part of the domain rules. In most systems that I've worked with, the application layer consisted of services orchestrating the domain objects to fulfill a use case scenario.
* **The domain layer**represents the underlying domain, mostly consisting of domain entities and, in some cases, services. Business rules, like invariants and algorithms, should all stay in this layer.
* **The infrastructure layer (also known as the persistence layer)**contains all the classes responsible for doing the technical stuff, like persisting the data in the database, like DAOs, repositories, or whatever else you’re using.

## Benefits of a Layered Architecture

Although some of you might still not believe it, Layered Architecture has some benefits, including:

* **Simplicity** – the concept is very easy to learn and visible in the project at first grasp.
* **Consistent across different projects**– the layers and so the overall code organization is pretty much the same in every layered project.
* **Guaranteed separation of concerns**– just the concerns that have a layer and to the point that you stick to the rules of Layered Architecture, but it’s very easy with the code organization it implies.
* **Browsability from a technical perspective**– when you want to change something in some/all objects of a given kind, they’re very easy to find and they’re kept all together.

1. Describe why transaction management is important, give an example

* **Isolation**: The degree to which this transaction is isolated from the work of other transactions. For example, can this transaction see uncommitted writes from other transactions?
* **Propagation**: Typically, all code executed within a transaction scope will run in that transaction. However, you have the option of specifying the behavior in the event that a transactional method is executed when a transaction context already exists. For example, code can continue running in the existing transaction (the common case); or the existing transaction can be suspended and a new transaction created. *Spring offers all of the transaction propagation options familiar from EJB CMT*.
* **Timeout**: How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.
* **Read-only status**: A read-only transaction can be used when your code reads but does not modify data. Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.

1. What is an index? What performance benefits do indexes give? If it improves performance then why not add it to all Database tables? How does it affect the various CRUD operations? Show using example using any RDMS

A **database index** is a data structure that improves the speed of data retrieval operations on a **database** table at the cost of additional writes and storage space to maintain the **index** data structure

What performance benefits do indexes give?

* Their use in queries usually results in much better performance.
* They make it possible to quickly retrieve (fetch) data.
* They can be used for sorting. A post-fetch-sort operation can be eliminated.
* Unique indexes guarantee uniquely identifiable records in the database.

Adding indexes to a table affects the performance of writes 23

Also each index must be updated any time a row is updated, inserted, or deleted. So the more indexes, the slower performance you'll have for write operations.

1. What are some differences between SQL and NoSQL databases?

**SQL databases:**

* store related data in tables
* require a schema which defines tables prior to use
* encourage normalization to reduce data redundancy
* support table JOINs to retrieve related data from multiple tables in a single command
* implement data integrity rules
* provide transactions to guarantee two or more updates succeed or fail as an atomic unit
* can be scaled (with some effort)
* use a powerful declarative language for querying
* offer plenty of support, expertise and tools.

**NoSQL databases:**

* store related data in JSON-like, name-value documents
* can store data without specifying a schema
* must usually be denormalized so information about an item is contained in a single document
* should not require JOINs (presuming denormalized documents are used)
* permit any data to be saved anywhere at anytime without verification
* guarantee updates to a single document — but not multiple documents
* provide excellent performance and scalability
* use JSON data objects for querying
* are a newer, exciting technology.

How do you decide which is best for you application?

* **SQL is digital.** It works best for clearly defined, discrete items with exact specifications. Typical use cases are online stores and banking systems.
* **NoSQL is analog.** It works best for organic data with fluid requirements. Typical use cases are social networks, customer management and web analytics systems.