Innova Solutions:

1. Singleton
2. What happens when we serialize singleton Ans: need to override **readResolve**()
3. How objects will be created for class A, Class B

```
@Singleton
Class A{
        @autowired
        B b;
}
@protoType
Class B{
        A a = new A();
        A aa=new A();
}
```

Ans: 1 for A, 1 for B

4. How many objects will be created here.

```
String x = "abc";

String y = new String("ABC");

String z = "ABC";
```

Ans: x in String pool, Y in Heap memory, Z in String pool

5. How to achieve this relationship in hibernate in single entity. Eg: It's a single table but the has separated like this.

```
Class A {
}
Class B extends A{
}
Class C extends A{
}
```

Ans:@ Discriminator, @embeddable

1. What is polymorphism
2. Can we override static method, What happends when we override static method

```
Class A {
```

Static M(){}

}

Class B extends A{

Static M(){}

}

Ans: Both act as individual method but looks like overriding

3. Diff between Spring and Boot
4. What are the way that we can create spring application **Ans**: Maven starter, Initlizr, CLI
5. How to create application from scratch with rest api and spring boot
   Ans: download with Spring starter, add @EnableAutoConfig uration @SpringBootApplication @RestController
6. What is initializer
7. How the auto configuration works in Spring **Ans**: AutoConfiguator
8. Can we use Spring in command line **Ans**: Spring CLI
9. What is @EnableautoConfiguration
   @SpringBootApplicatio@EnableAutoConfiguration annotation auto-configures the beans that are present in the classpath. This simplifies the developers work by guessing the required beans from the classpath and configure it to run the application. This annotation is part of the spring boot project.

10. How two servlets communicate each other **Ans**: **Request Dispatcher**'s **Forward** or **Include** methods (available with request object) or **Send Redirect** method
11. How Dispatcher Servlet work

The DispatcherServlet is one of the important components of the Spring MVC web framework and acts as a Front Controller. ... The DispatcherServlet is a front controller like it provides a single entry point for a client request to Spring MVC web application and forwards request to Spring MVC controllers for processing.

12. How to create threat Ans: Runnable & thread class
13. How two threads communicate each other Ans: wait, notify and notifyAll
14. What is wait, notify & notifyAll
15. Diff betweem Notify & NotifyAll

. First and main difference between notify() and notifyAll() method is that, if multiple threads is waiting on any lock in Java, notify method send notification to only one of waiting thread while notifyAll informs all threads waiting on that lock.

16. Why wait notify notifyAll in Object class

    Locks are made available on per Object basis, which is another reason wait and notify is declared in Object class rather then Thread class.

17. Why Sleep and yield was static

    The code would only execute when someXThread was executing, in which case telling someYThread to yield would be pointless. So since the only thread worth calling yield on is the current thread, they make the method static so you won't waste time trying to call yield on some other thread.

18. Thread pool

    Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times.

    In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

19. Java Shutdown Hook

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

20. When does the JVM shut down?

The JVM shuts down when:

- user presses ctrl+c on the command prompt
- System.exit(int) method is invoked
- user logoff
- user shutdown etc.

21. What is atomicity, atomic variables.

Atomic variables come to the rescue in these types of situations. Atomic variables allow us to perform atomic operations on the variables.

22. ArrayList Vs LinkedList, when for what
23. Can we change the value while traverse ArrayList

Throws Concurrent Modification Exception.

- Use CopyOnWriteArrayList
- With Java 8 stream we can achive
- ArrayList.remove()

It is not recommended to use ArrayList.remove() when iterating over elements. This may lead to ConcurrentModificationException (Refer this for a sample program with this exception). When iterating over elements, it is recommended to use Iterator.remove() method .

24. Is ArrayList thread safe. Is LinkedList a thread safe
25. What other option for arrayList thread safe.

- Collections.synchronizedList() method – It returns synchronized list backed by the specified list.
- CopyOnWriteArrayList class – It is a thread-safe variant of ArrayList.

26. Dialect

Ans: The SQL dialect makes Hibernate generate better SQL for the chosen database

27. get VS load in hibernate

load()	get()
Only use the `load()` method if you are sure that the object exists.	If you are not sure that the object exists, then use one of the `get()` methods.
`load()` method will throw an exception if the unique id is not found in the database.	`get()` method will return null if the unique id is not found in the database.
`load()` just returns a proxy by default and database wonï¿½t be hit until the proxy is first invoked.	`get()` will hit the database immediately.

28. How to configure JDBC template
29. Is session thread safe
30. how to make session thread safe

    Why to make session object thread safe if we already have a SessionFactory(immutable) object.

Ideally you should have different session object for each thread.
//use ThreadLocal to achive same
private static ThreadLocal<Session> threadLocal = new ThreadLocal<Session>();

31. how to update a row in hibernate
    https://www.dineshonjava.com/difference-between-merge-and-update-methods-in-hibernate/

Ramco (14/11/19):

32. React VS Angular

Technology	React	AngularJS	Angular 2
Author	Facebook community	Google	Google
Type	Open source JS library	Fully-featured MVC framework	Fully-featured MVC framework
Toolchain	High	Low	High
Language	JSX	JavaScript, HTML	TypeScript
Learning Curve	Low	High	Medium
Packaging	Strong	Weak	Medium
Rendering	Server Side	Client Side	Server Side
App Architecture	None, combined with Flux	MVC	Component-based
Data Binding	Uni-directional	Bi-directional	Bi-directional
DOM	Virtual DOM	Regular DOM	Regular DOM
Latest Version	15.4.0 (November 2016)	1.6.0	2.2.0 (November 2016)

33. WebSocket
    WebSockets provide a persistent connection between a client and server that both parties can use to start sending data at any time. The client establishes a **WebSocket** connection through a process known as the **WebSocket** handshake. This process starts with the client sending a regular HTTP request to the server

    https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9

34. PWA

Progressive Web App (**PWA**) is a new class of Web Application that provide native app experiences using a browser. PWAs function even when the device is offline. PWAs make

full use of modern web features including push notification, cache, and secure connections to provide rich Web based experience

35. Microservices
36. NodeJs as middleware
37. Framework in NodeJS'

    1. AdonisJs [GitHub Stars: 5,053]
    2. Express.js [GitHub Stars: 41,036]
    3. Meteor.js [GitHub Stars: 40,490]
    4. Nest.js [GitHub Stars: 10,128]
    5. Sails.js [GitHub Stars: 19,887]
    6. Koa.js [GitHub Stars: 23,902]
    7. LoopBack.js [GitHub Stars: 11,985]
    8. Hapi.js [GitHub Stars: 10,371]
    9. Derby.js [4,350]
    10. Total.js [Github stars: 3,853]

HCL (15/11/19):

38. What is Distributed transaction
39. When we have reservation, in situation where the payment is not success, how to revert all transaction. Earlier transaction was committed
    2Phase commit
    Saga Pattern
    CQRS, with event sourcing
    https://dzone.com/articles/3-most-important-patterns-for-building-microservic

40. In Spring boot, how to connect a database repository eg:mysql.

    It is done in the following steps.
    Step 1 -
    The first step to connect the database like Oracle or MySql is adding the dependency for your database connector to pom.xml.
    Step 2 -
    The next step is the elimination of H2 Dependency from pom.xml
    Step 3 -
    Step 3 includes the schema and table to establish your database.
    Step 4 -

The next step is configuring of the database by using Configure application.properties to connect to your database.

Step 5-

And the last step is to restart your device and your connection is ready to use.

Caterpillar 11/08/2019:

41. What is the backend of the UI for your last project? Oracle
42. Personal Experience that data modelling with sql server.
43. How to data model DB for micro services?
44. Performance tuning with DB w.r.t Rest API?]

https://medium.com/rpdstartup/rest-api-performance-tuning-getting-started-7a6efefa9e20

Network Latency

No matter how good you design the REST layer, the speed of the network matters a lot. Network latency can thus indirectly impact the responsiveness of the REST server.

Needless Data

Sometimes the design of REST layer could be done in haste leading to unwanted data being sent back and forth from REST server to client.

Business and Database Layer Performance

Even if REST API is designed properly, the performance can be impacted due to poor design of Business Logic or the DAO layers.

Security

Since RESTful Web Services work on top of HTTP, it is important to ensure proper web service security is implemented to avoid unauthorized access, snooping and DDoS attacks.

Server Side Load

Having too many client requests in place could make the server load high and if the server doesn't have enough capacity it can result in poor performance at the REST server end.

45. How to do performance testing in rest API? Apache JMeter
46. How to identify which dependency is the root cause of performance in REST API. Ex: HTTP Dependencies?
    1. Sluggish client
    2. Slow server
    3. Diminutive databases

4. Chatty conversations
5. Slow network services

47. How to do functional testing with micro services?
    1. Unit Testing
    2. Contract Testing
    3. Integration testing
    4. End to End testing
    5. UI/Functional testing

48. Integration testing in micro service and rest API?
49. AWS Api gate & Lamda pros & cons?

Lamda Advantages	Disadvantages
Faster Development: Easier Operational Management: Scaling benefits of FaaS beyond costs: Reduction in Operational Costs:	State No Control Over Environment DoS (Denial of Service) Execution Duration Startup Latency Testing
ApiGateway Advantages	Disadvantages
-Microservices can focus on business logic - Clients can get all the data in a single hit - Authentication, logging, and monitoring - It gives flexibility to use completely independent protocols in which clients and microservice can talk - It can give tailor-made results, as per the clients needs - It can handle partial failure	Performance degradation

50. What is best API gateway
51. What if microservices goes slow?
    Scenrio: Micor service can go down
    Solution: Have multiple service instance

Scenrio: Instance is slow
Solution: 1. Time out
2. Circuit breaker pattern – Hystrix
3. BulkHead pattern

52. Spring micro service integration

Virtusa

53. Write Factory pattern?
54. What is the new feature in Java8?
55. What is new feature w.r.t inter thread communication? Ans: Blocking Queue
56. Denial of Service attack
    A **Denial-of-Service (DoS) attack** is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash. In both instances, the DoS attack deprives legitimate users (i.e. employees, members, or account holders) of the service or resource they expected.
57.

HCL:

58. HLD, LLD what kind of document, diagram, and how you create the HLD and LLD.
59. UML diagram
60. Microservice architecture pattern, CQRS
61. Two phase commit
62. How to implement 2pc through hibernate
63. We have two services; we have db operation on both services. How to ensure the acid properties esp atomicity here.
    One service DB transaction is successful and other was failed. How to rollback.
64. Pros and cons of 2pc
65. How two micro service communicate – WebClient or RestTemplate
66. Api gate way vs service discovery
67. Async communication in microservices
68. Data structure for Car parking
69. Data structure when the data is not used will be removed and newly added data were added in to it.
70. How to create LinkedList
71. What api you have used for rest call. Eg; Jackson

72. What is stateless in rest api. Pros and cons

Cognizant

73. Advantages of Spring boot
74. How will you communicate between micro services?
75. How to implement eureka hystrix
76. Micro services pattern
77. Apigateway, what it does
78. Rest api commincation between rest api?
79. How to find the bean details instantation in spring boot not with actuator?
80. Lifecycle of bean in springs
81. Use qualifier in spring
82. Operational
83. Extend and super w.r.t genrics
84. Class<?>
85. Twoway binding in angular vs interpollation
86. Filter in angular
87. @input @output
88. Sonar vs Fortify ANs: Both were static code analyser, where sonar mainly for Security and Vulnerability
89. Write java8 code for filter and map
90. Write code for try resources
91. What happens when we give wrong hierarchy for in catch block
92. How Transaction happens in micro service
93. Design patterns for micro service

FreshWorks

94. How to estimate the new development
95. How to migrate old application into new building new application
96. On building new application with micro services, How to select technology
97. Testing strategy
98. Write code Filter, map in angular
99. Write Rest api Code for Entity, controller, Sevice

First data

100. SOAP vs rest

	SOAP	REST
Meaning	Simple Object Access Protocol	Representational State Transfer
Design	**Standardized protocol with pre-defined rules to follow.**	**Architectural style with loose guidelines and recommendations.**
Approach	Function-driven (data available as services, e.g.: “getUser”)	Data-driven (data available as resources, e.g. “user”).
Statefulness	**Stateless by default, but it’s possible to make a SOAP API stateful.**	**Stateless (no server-side sessions).**
Caching	API calls cannot be cached.	API calls can be cached.
Security	**WS-Security with SSL support. Built-in ACID compliance.**	**Supports HTTPS and SSL.**
Performance	Requires more bandwidth and computing power.	Requires fewer resources.
Message format	**Only XML.**	**Plain text, HTML, XML, JSON, YAML, and others.**
Transfer protocol(s)	HTTP, SMTP, UDP, and others.	Only HTTP
Recommended for	**Enterprise apps, high-security apps, distributed environment, financial services, payment gateways, telecommunication services.**	**Public APIs for web services, mobile services, social networks.**
Advantages	High security, standardized, extensibility.	Scalability, better performance, browser-friendliness, flexibility.
Disadvantages	**Poorer performance, more complexity, less flexibility.**	**Less security, not suitable for distributed environments.**

101. Left join
102. Micro service pros

Luxsoft

103. 12 Factor Principle
104. Lamda
105. Streams
106. Collection interface
107. Array List Linked List, equals
108. Contract of HashMap key

109. Why linked list is fast
110. Normalisation/denormalization

Normalization	De-Normalization
Normalization is the process of dividing the data into multiple tables, so that data redundancy and data integrities are achieved.	De-Normalization is the opposite process of normalization where the data from multiple tables are combined into one table, so that data retrieval will be faster.
It removes data redundancy i.e.; it eliminates any duplicate data from the same table and puts into a separate new table.	It creates data redundancy i.e.; duplicate data may be found in the same table.
It maintains data integrity i.e.; any addition or deletion of data from the table will not create any mismatch in the relationship of the tables.	It may not retain the data integrity.
It increases the number of tables in the database and hence the joins to get the result.	It reduces the number of tables and hence reduces the number of joins. Hence the performance of the query is faster here compared to normalized tables.
Even though it creates multiple tables, inserts, updates and deletes are more efficient in this case. If we have to insert/update/delete any data, we have to perform the transaction in that particular table. Hence there is no fear of data loss or data integrity.	In this case all the duplicate data are at single table and care should be taken to insert/delete/update all the related data in that table. Failing to do so will create data integrity issues.
Use normalized tables where more number of insert/update/delete operations are performed and joins of those tables are not expensive.	Use de-normalization where joins are expensive and frequent query is executed on the tables.

111. Bean invocation
112. Saga pattern
113. Functional interface
114. More than 2 abstrct method and mark as @functionalInterface, will it compile:
    NO
115. Apigateway pattern
116. CQRS
117. Decorator pattern

118. Proxy pattern
119. IOC
120. What is Concurreny
121. Code First Approach
122. Solid principle

S.O.L.I.D. Class Design Principles

Principle Name	What it says? howtodoinjava.com
Single Responsibility Principle	One class should have one and only one reasonability
Open Closed Principle	Software components should be open for extension, but closed for modification
Liskov's Substitution Principle	Derived types must be completely substitutable for their base types
Interface Segregation Principle	Clients should not be forced to implement unnecessary methods which they will not use
Dependency Inversion Principle	Depend on abstractions, not on concretions

123. Haetos rest api

Hypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate

Instead, if you return the URLs, it could use for the actions, then it is loosely coupled. There is no tight dependency on the URI structure, as it is specified and used from the response.

HAL — Hypertext Application Language

When you design a RESTful service, there is a need to specify how to return data and links corresponding to a request. HAL is a simple format that gives an easy, consistent way to hyperlink between resources in your REST API.

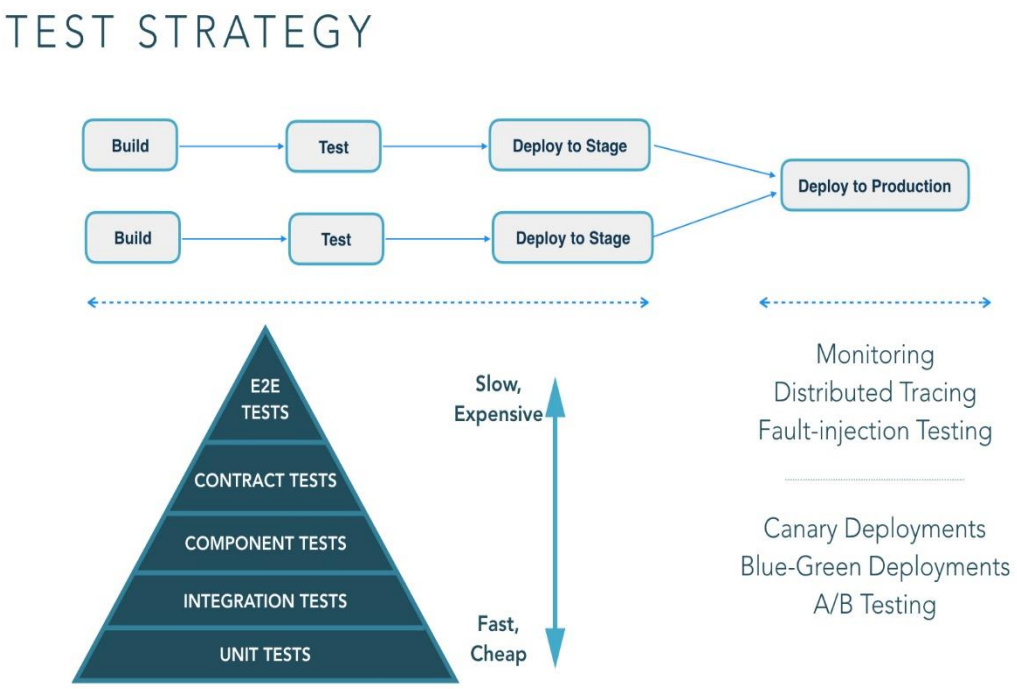
124. Circular dependency Spring – Ans: @Lazy on Constructor or Setter Injection
125. Self invocation @Transactional

that means a method within the target object calling another method of the same target object will not create a transaction even if the method called has @Transactional set.

In this case you can use AspectJ with Spring Transactional to get this working.

126. What are the test strategy in micro services
    - UnitTesting

- Integration Testing
- Contract Testing
- End-End Testing



127. Event sourcing in MS, what tool you use

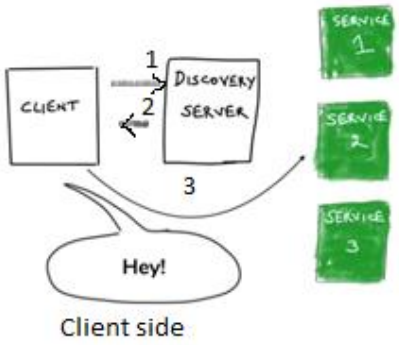
A service command typically needs to update the database **and** send messages/events. For example, a service that participates in a saga needs to atomically update the database and sends messages/events. Similarly, a service that publishes a domain event must atomically update an aggregate and publish an event. The database update and sending of the message must be atomic in order to avoid data inconsistencies and bugs.
https://microservices.io/patterns/data/event-sourcing.html

128. Service discovery

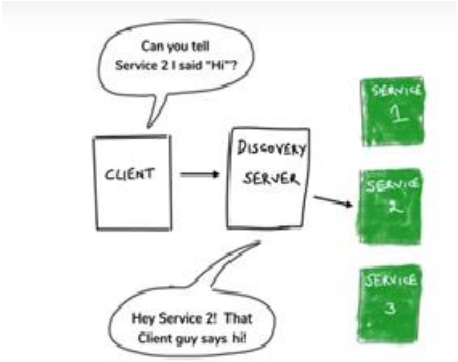
- Maintaining dynamic URL, cloud has dynamic urls genrated
- Load Balanced
- Multiple environment maintennce

Eg: Eureka
@EnableEurekaServer
@EnableEurekaClient
Service Discovery

/ \
Client Side Server Side



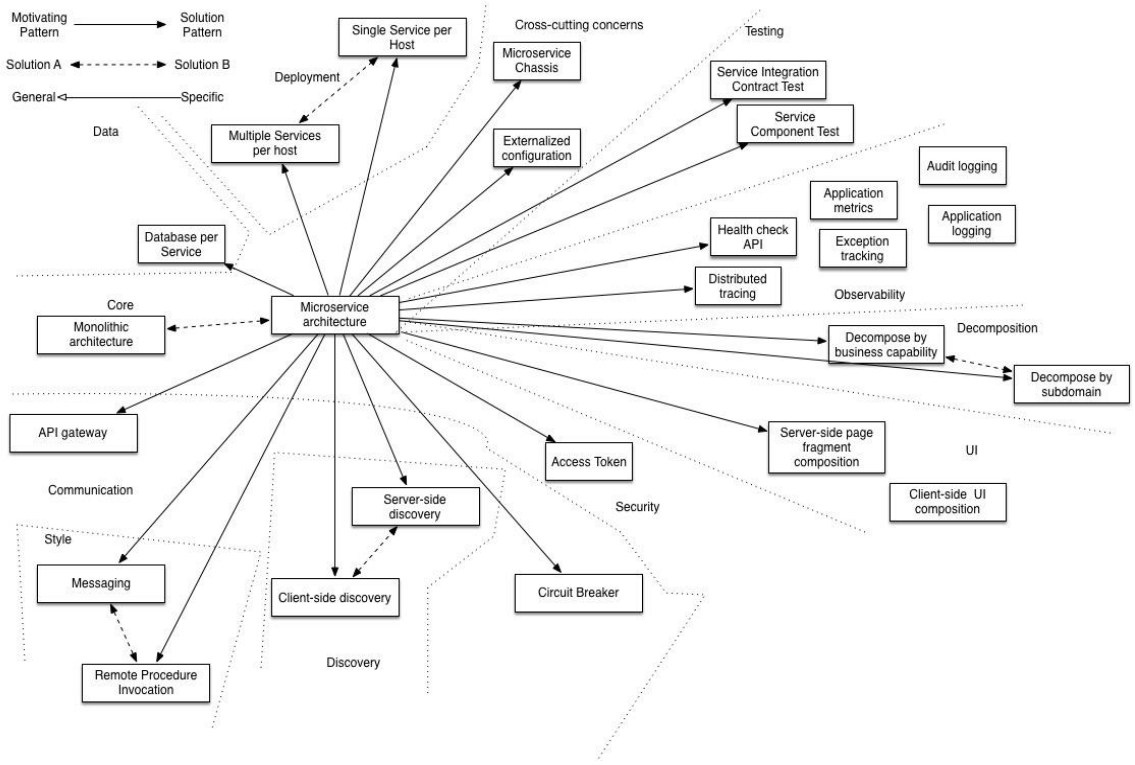
Client side



Server side

129. How you segregate the micro service
Based on DDD – Domain driven design
130. Microservice patterns
https://dzone.com/articles/design-patterns-for-microservices

131. How HaspMap works
132. How hassh map works
133. load factor of hashmap .75
134. OCncurrent hashmap
135. load factor
136. concurrency level in con.hashmap
137. Serialziation
138. what serializable object
139. what if non serialaziable extends serilizable
140. what immutable
141. non immutable objects in mutable
142. microservices vs monolithic
143. @SpringbootApplication
144. cloud config server
145. Acutator
146. service discoverey
147. Apigateway
148. Actuator
149. Admin server – Spring boot



Application architecture patterns

Decomposition

- Decompose by business capability
- Decompose by subdomain
- Self-contained Service
- Service per team

Data management

- Database per Service
- Shared database
- Saga
- API Composition
- CQRS
- Domain event
- Event sourcing

Transactional messaging

- Transactional outbox
- Transaction log tailing

Deployment patterns

- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment
- Service deployment platform

Cross cutting concerns

- Microservice chassis
- Externalized configuration

Communication style

- Remote Procedure Invocation
- Messaging
- Domain-specific protocol

External API

• Polling publisher **Testing** • Service Component Test • Consumer-driven contract test • Consumer-side contract test **Observability** • Log aggregation • Application metrics • Audit logging • Distributed tracing • Exception tracking • Health check API • Log deployments and changes **Security** • Access Token	• API gateway • Backend for front-end **Service discovery** • Client-side discovery • Server-side discovery • Service registry • Self registration • 3rd party registration **Reliability** • Circuit Breaker **UI patterns** • Server-side page fragment composition • Client-side UI composition

Hexaware

150. Hibernate Translation propagation

Propagation

Defines how transactions relate to each other. Common options:

- Required: Code will always run in a transaction. Creates a new transaction or reuses one if available.
- Requires_new: Code will always run in a new transaction. Suspends the current transaction if one exists.

Isolation

Defines the data contract between transactions.

- Read Uncommitted: Allows dirty reads.
- Read Committed: Does not allow dirty reads.
- Repeatable Read: If a row is read twice in the same transaction, the result will always be the same.
- Serializable: Performs all transactions in a sequence.

Msys

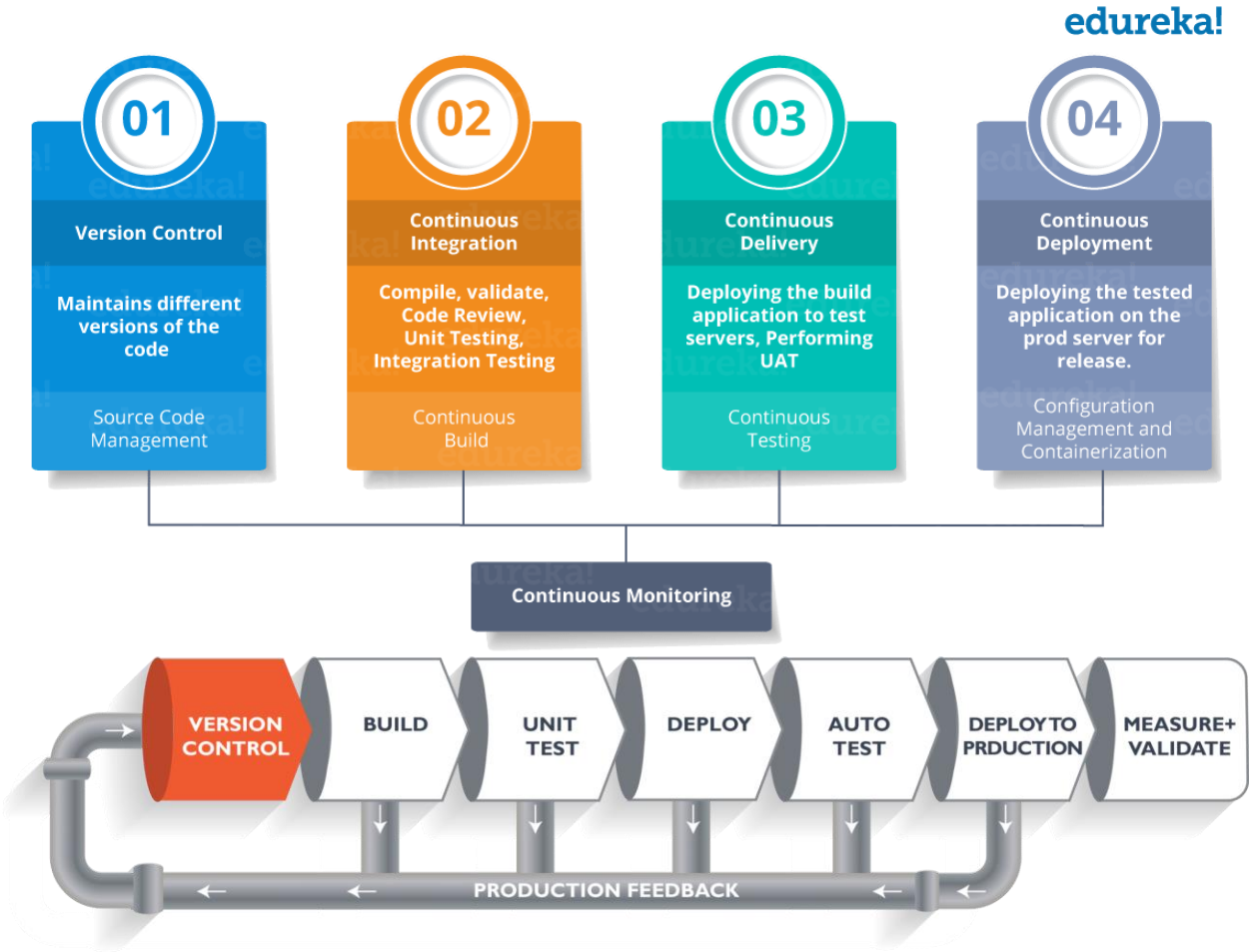
151. Time Out in Rest Template

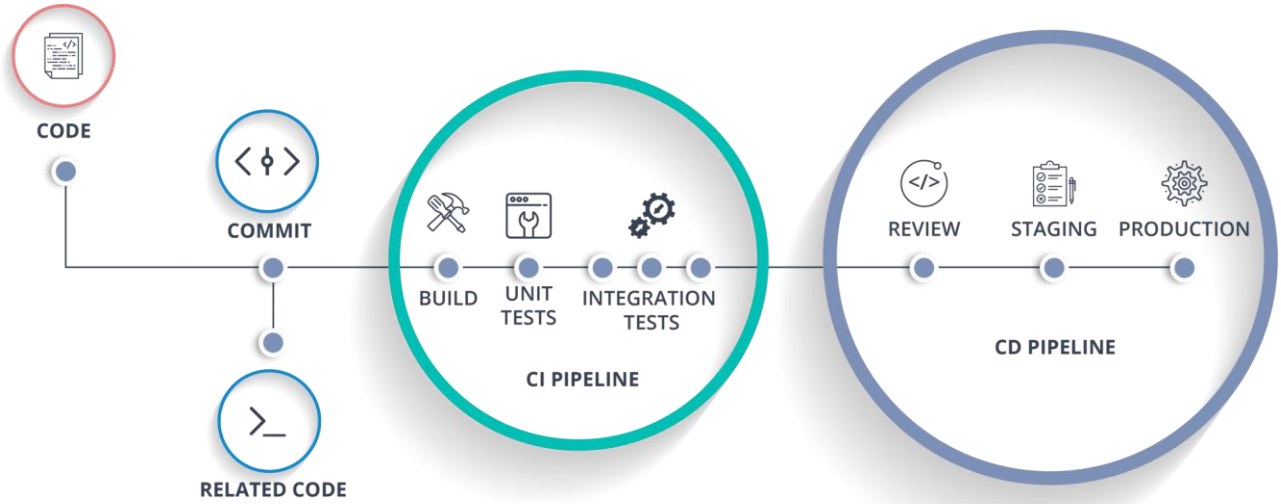
ConnectionRequestTimeout. This is timeout in millis for getting connection from connectionManager
ConnectionTimeout. This is timeout in millis for establishing connection between source and destination
ReadTimeout. This is timeout in millis which expects the response/result should be returned from the destination endpoint.

152. How to implement swagger

Springfox-swagger2 dependency, @EnableSwaggerClient

153. CI/CD – how to impelement





https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch

Ideas2IT

154. Two interfaces with same methods having same signature but same return type

Yes it works. If the return type is different it throws error.

155. Can we have Final in hashMap?

No. we cant put new objects

156. Mutable vs immutable

157. Struture of String class

public final class **String**
extends Object
implements Serializable, Comparable<String>, CharSequence

158. Immutable Class.

```
// An immutable class
public final class Student
{
    final String name;
    final int regNo;
```

```
    public Student(String name, int regNo)
    {
        this.name = name;
        this.regNo = regNo;
    }
    public String getName()
    {
        return name;
    }
    public int getRegNo()
    {
        return regNo;
    }
}
```

159. Annotation how spring identifies annotation
160. Security in spring
161. HQL conversion with native query
162. First level cache
163. Filter VS Interceptor

Filter:

A filter dynamically intercepts requests and responses to transform or use the information contained in the requests or responses. Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.

Filters can perform many different types of functions. We'll discuss examples of the italicized items in this paper:

- Authentication-Blocking requests based on user identity.
- Logging and auditing-Tracking users of a web application.
- Image conversion-Scaling maps, and so on.
- Data compression-Making downloads smaller.
- Localization-Targeting the request and response to a particular locale.

Request Filters can:

- perform security checks
- reformat request headers or bodies
- audit or log requests

Response Filters can:

- Compress the response stream
- append or alter the response stream
- create a different response altogether

Examples that have been identified for this design are

- Authentication Filters
- Logging and Auditing Filters
- Image conversion Filters
- Data compression Filters
- Encryption Filters
- Tokenizing Filters
- Filters that trigger resource access events
- XSL/T filters
- Mime-type chain Filter

Interceptors

Interceptors are used in conjunction with Java EE managed classes to allow developers to invoke interceptor methods in conjunction with method invocations or lifecycle events on an associated target class. Common uses of interceptors are logging, auditing, or profiling.

- Cookie Interceptor
- Checkbox Interceptor
- FileUpload Interceptor

Difference:

A Servlet Filter is used in the web layer only, you can't use it outside of a web context. Interceptors can be used anywhere. That's the main difference.

for authentication of web pages you would use a servlet filter. For security stuff in your business layer or logging/bugtracing (a.k.a. independent of the web layer) you would use an Interceptor.

Apart from the fact that both Interceptors and filters are based on intercepting filter,there are few differences when it comes to Struts2.

Filters: (1)Based on Servlet Specification (2)Executes on the pattern matches on the request.(3) Not configurable method calls
Interceptors: (1)Based on Struts2. (2)Executes for all the request qualifies for a front controller(A Servlet filter).And can be configured to execute additional interceptor for a particular action execution.(3)Methods in the Interceptors can be configured whether to execute or not by means of excludemethods or includeMethods

Hexaware:

- Count vowels, alphabets, numbers and special characters in the sentence
- Count duplicate words in a sentence
- Url for rest Api get,post,put methods

```
@RestController
@RequestMapping("api/v1/")
public class ShipwreckController {
  @RequestMapping(value = "shipwrecks", method = RequestMethod.GET)
  public List<Shipwreck> list(){
    return ShipwreckStub.list();
  }
  @RequestMapping(value = "shipwrecks", method = RequestMethod.POST)
  public Shipwreck create(@RequestBody Shipwreck shipwreck){
    return ShipwreckStub.create(shipwreck);
  }
  @RequestMapping(value = "shipwrecks/{id}", method = RequestMethod.GET)
  public Shipwreck get(@PathVariable Long id){
    return ShipwreckStub.get(id);
  }
  @RequestMapping(value = "shipwrecks/{id}", method = RequestMethod.PUT)
  public Shipwreck update(@PathVariable Long id, @RequestBody Shipwreck shipwre
ck){
    return ShipwreckStub.update(id, shipwreck);
  }
  @RequestMapping(value = "shipwrecks/{id}", method = RequestMethod.DELETE)
  public Shipwreck delete(@PathVariable Long id){
    return ShipwreckStub.delete(id);
  }
}
```

Code review
Database connection properties in server

- **Connection Timeout**

  Specify the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown. This action can occur when the pool is at its maximum (Maximum Connections) and all of the connections are in use by other applications for the duration of the wait.

  For example, if Connection Timeout is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is not available within this time, the Pool Manager throws a ConnectionWaitTimeoutException.

  Tip: If Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection is allocated.

- **Maximum Connections**

  Specify the maximum number of physical connections that can be created in this pool.

  These connections are the physical connections to the back-end database. After this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use is returned to the pool, or a ConnectionWaitTimeoutException is thrown.

  For example, if Maximum Connections is set to 5, and there are five physical connections in use, the Pool Manager waits for the amount of time specified in Connection Timeout for a physical connection to become free. If, after that time, there are still no free connections, the Pool Manager throws a ConnectionWaitTimeoutException to the application.

- **Minimum Connections**

  Specify the minimum number of physical connections to be maintained. Until this number is reached, the pool maintenance thread does not discard any physical connections. However, no attempt is made to bring the number of connections up to this number.

For example, if Minimum Connections is set to 3, and one physical connection is created, that connection is not discarded by the Unused Timeout thread. By the same token, the thread does not automatically create two additional physical connections to reach the Minimum Connections setting.

- **Reap Time**

  Specify the interval, in seconds, between runs of the pool maintenance thread.

  For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval you set, the greater the accuracy. When the pool maintenance thread runs, it discards any connections that have been unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Minimum Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

  Tip: If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout.

  The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

- **Unused Timeout**

  Specify the interval in seconds after which an unused or idle connection is discarded.

Caterpillar:
AWS lambda api gate
Comparator with lamda
Entity mapping with hibernate
How you estimate
How to lead team

- Communicate, communicate, communicate.
- Give them the freedom to use their talents.
- Be accessible and available.

- Guide them to work together toward a compelling vision.
- Give them permission to make mistakes.
- Show up as the leader and develop leaders within.
- Give them what they need to be successful.
- Create an environment of fun and enjoyment
- Model accountability and teach responsibility.
- Be decisive and purposeful.
- Don't underestimate the power of EQ
- Give them something to learn and grow on.
- Show them fearlessness and encourage them to be brave.
- Earn their respect and give them yours.
- Admit that you don't have all the answers.
- Create win-win situations.
- Be agile and flexible.
- Be honest and encourage candor.
- Consistently praise them and always appreciate them.

Micro services design pattern

Design Car parking

Cap Gemini:
Explain DBLink

Logger in Java

1. We can print log messages on the console only. So, when the console is closed, we will lose all of those logs.
2. We can't store log messages in any permanent place. These messages will print one by one on the console because it is a single-threaded environment.

Log4j Components

Log4j has three main components, which are the following:

1. Logger
2. Appender
3. Layout

Logger

Syntax to get Logger objects:

```
static Logger logger = Logger.getLogger(CurrentClass.class.getName()).
```

We have five methods in the Logger class

1. info()
2. debug()
3. warn()
4. fatal()
5. error()

debug < info < warn < error < fatal

Appender

Appender is used to write messages into a file or DB or SMTP.

Layout

This is used to define the formatting in which logs will print in a repository.
We have different types of layouts:

1. PatternLayout
2. SimpleLayout
3. XMLLayout
4. HTMLLayout

log4j.properties

```
# Root logger option
log4j.rootLogger=INFO, file, stdout
# configuration to print into file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=D:\\log\\logging.log
log4j.appender.file.MaxFileSize=12MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%
L - %m%n
# configuration to print on console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{
1}:%L - %m%n
```

Virtusa:

1. How you estimate a development if you know solution

A **Function Point** (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user. FPs measure software size. They are widely accepted as an industry standard for functional sizing.

A **Use-Case** is a series of related interactions between a user and a system that enables the user to achieve a goal.

Use-Cases are a way to capture functional requirements of a system. The user of the system is referred to as an 'Actor'. Use-Cases are fundamentally in text form.

Delphi Method is a structured communication technique, originally developed as a systematic, interactive forecasting method which relies on a panel of experts. The experts answer questionnaires in two or more rounds. After each round, a facilitator provides an anonymous summary of the experts' forecasts from the previous round with the reasons for their judgments. Experts are then encouraged to revise their earlier answers in light of the replies of other members of the panel.

Analogous Estimation uses a similar past project information to estimate the duration or cost of your current project, hence the word, "analogy". You can use analogous estimation when there is limited information regarding your current project

Work Breakdown Structure (WBS), in Project Management and Systems Engineering, is a deliverable-oriented decomposition of a project into smaller components. WBS is a key project deliverable that organizes the team's work into manageable sections. The Project Management Body of Knowledge (PMBOK) defines WBS as a "deliverable oriented hierarchical decomposition of the work to be executed by the project team."

Planning Poker Estimation Planning Poker is a consensus-based technique for estimating, mostly used to estimate effort or relative size of user stories in Scrum.

Planning Poker combines three estimation techniques – Wideband Delphi Technique, Analogous Estimation, and Estimation using WBS.

2. How you estimate a development that is very new to you
3. Last sprint has too many defects, it eradicates in next sprint:
   root cause analysis: why 5, fish bone,
4. How to ensure you are on track in sprint development
5. How you lead the team

Olam International:

1. Project Architecture
2. How zuul configures end points
3. How Eureka server configure end points
4. How eureka routes the service
5. Spring boot works async
6. How to fetch billion of records from micro service
7.

Verizon:

1. How to track the micro service, what re the steps.
2. Log format for micro service
3. How to set time outs for micro service
4. How to set time outs hirerechyly ?
5. How to handle if micro service is not responding
6. Two communication
7. Micro service - Service config
8. How to handle if some common code used across all micro service
9. How to build common utility
10. Is two-way communication good for micro service Ans: No. hierarchy is best
11. Exception handling across micro service
12. Software engineering principles:
    a) Measure twice Cut once
    b) Divide and conquer
    c) DRY – Don't repeat yourself
    d) KISS – Keep it simple, stupid
    e) Yagni – You aren't gonna need it
    f) DRITW (Don't Re-Invent The Wheel)
    g) Write code that does one thing well
    h) Kaizen (leave it better than when you found it)
    i) Make it WORK first, then work RIGHT, THEN look pretty
    j) SOLID
        1. Single responsibility
        2. Open closed
        3. Liskow principle - That every child/derived class should be substitutable for their parent/base class without altering the correctness of the program. In other words, the objects of your subclass should behave in the same way as the objects of your superclass.
        4. Interface Segregation

5. Dependency inversion

Others:

1. Java streams reduce
2. Write sum of collection using streams, lamda
3. How to tune performance in java application? What were the tools you used?
4. What is index in DB. What re the drawbacks
5. What is AWS-Route 53
6. What is stream api? Where it is used?
7. Total salary of employee collection using stream api
8. Different kind of hash map
9. Identity hash map?
10. Collection that Read while write on iterating?
11. Write Many-Many implementations in hibernate between two tables in hibernate
12. Many-many implementation in DB
13. Write crud operation in rest service
14. How the url be designed in rest api
15. Write get,I nsert, update apis in rest service
16. Pagination in hibernate
17. Queue and topic in ESB