DataBase: Oracle

1. **Get a List of all Tables in Oracle SQL**
   The easiest way to see all tables in the database is to query all_tables view: SELECT owner, table_name FROM all_tables.

2. **How to schedule any task in Java?**
   Scheduling a Task in Java: The scheduler is used to schedule a thread or task that executes at a certain period or periodically at a fixed interval.

```
@Scheduled(initialDelay = 70000, fixedDelay = 300000)
@SchedulerLock(name = "scheduledforSomething", lockAtMostFor =
"240s", lockAtLeastFor = "240s")
```

   Two rules for writing schedulers:
   1. Should have void return types.
   2. Should not have method parameters.

   If 2 or more instances there is a chance to pick the same record or event to happen, it will result in duplicate operation, to avoid this Pessimistic Locking (it will lock the entire row in the DB for the mentioned time)

```
@Lock(LockModeType.PESSIMISTIC_READ) @QueryHints({@QueryHint(name =
"javax.persistence.lock.timeout", value = "3000")})
public Optional<Customer> findById(Long customerId);
```

3. **Why are we using JPA (Java Persistent (Any entity we create) API) only?**
   JPA supports all ORM related databases which has implemented JPA interfaces.

4. **Types of queries supported in JPA**

   - *Query*, written in Java Persistence Query Language (JPQL) syntax
     - ✓ *TypedQuery*

```
@Query("select * from EmployeeDetails where id=:pid and name=pname")
public Optional<EmployeeDetails> findByIdAndName(@Param("id") int pid,
@Param("name") String pname);
```

✓ *NamedQuery*

```
@Table(name = "users")
@Entity
@NamedQuery(name = "UserEntity.findByUserId", query = "SELECT u FROM
UserEntity u WHERE u.id=:userId")
public class UserEntity {
    @Id
   private Long id;
   private String name;
   //Standard constructor, getters and setters.

}
```

- *NativeQuery*, written in plain SQL syntax
- *Criteria API Query*, constructed programmatically via different methods

a. Method Queries
b. Query Methods (Native query and JPQL)
c. NamedQueries (Native and JPQL)

a. Method Queries

b. Dynamic queries/ User defined queries

```
@Query("select * from EmployeeDetails where id=:pid and name=pname")
public Optional<EmployeeDetails> findByIdAndName(@Param("id") int pid,
@Param("name") String pname);
```

c. JPQL example: (Entity name and entity properties will be used to define a query)

```
@Query("select * from EmployeeDetails")
public List<EmployeeDetails> findDetails();
```

d. Named Queries:

For named queries we got to create Entity manager first then using entity manager we must create Named query as shown below.

```
@NamedQueries(value={@NamedNativeQuery( })

TypedQuery<Student> createNamedQuery =
entityManager.createNamedQuery("find_all_students", Student.class);
        return createNamedQuery.getResultList();
```

Later we must write the query on top of the entity for which we want to perform job.

```
@NamedQuery(name="find_all_students", query="select s from Student s")
```

5. **What are Joins and types of joins?**
   a. **Inner join or Join:** Data matched in both the tables.
   b. **Left Join:** All data in left table and matched data in right table.
   c. **Right join:** All data in right table and matched data in left table
   d. **Outer Join:** All the date from both the tables.

6. **What is INDEX.**

   Oracle index is one of the effective tools for boost the **query performance.** However, to use it effectively, you must understand it correctly.

   Create Index:

   CREATE INDEX index_name ON table_name (column1[, column2,])

   Drop Index:

   DROP INDEX [schema_name.] index_name

7. **What is view in oracle?**

   View is a virtual table that does not physically exist. It is stored in Oracle data dictionary and do not store any data.
   We can create view from multiple tables.
   Materialized view: It will have all data updated and stored so that we can fetch and refresh.

```
CREATE VIEW view_name AS SELECT
columns FROM tables
WHERE conditions.
```

8. **What is the d/f b/w Primary Keya and Foreign Key?**

9. **Primary key and Unique Key.**
   Primary key: its cannot be null and always unique.

Unique Key: A unique key is a set of one or more fields/columns of a table that uniquely identify a record in a database table. You can say that it is little like primary key, but it can accept only one null value and it cannot have duplicate values.

10. **Composite Primary Key:**
When we create a primary key with more than one column.

11. **D/F between Delete and truncate.**
    - **a.** **Delete:** It is a DML query While **Truncate** is DDL Query.
    - **b.** Delete uses where clause while Truncate is not.
    - **c.** Delete is not an Auto Commit but Truncate is by default Auto Commit.
    - **d.** Delete is used to delete only row but truncate is delete entire table.
    - **e.** Delete is slower than Truncate.

12. **DDL/DML/DCL (Data Control Language)**
    - **a.** **DCL:** All permission related queries like Grant and Revoke
    - b. **DDL**: Alter, create, Drop, truncate, and rename.
    - c. **DML**: select, update, delete, merge call and lock table.
    - d. **TCL**: Commit, rollback, save point.

13. **Query to find Nth Highest salary.**
To find nth highest salary using Sub-Query
SELECT TOP 1 SALARY FROM (SELECT DISTINCT TOP N SALARY FROM EMPLOYEES ORDER BY SALARY DESC) RESULT ORDER BY SALARY

select * from (SELECT * FROM employee ORDER BY salary DESC LIMIT 5)
AS table1 ORDER BY salary ASC LIMIT 1;

In Oracle:
select salary from employee order by salary DESC offset 4 rows FETCH NEXT 1 ROWS ONLY.

14. **What is the difference between Procedure and Function?**
Procedure have return value but not Functions.
Procedure and Functions can be used to write some business logic.

15. **Char and varchar2**
Char:
- by default, length is 1 byte and if we specify the length and store 1 char it will occupy all the length with empty spaces.
- 1 to 2000

Varchar:
- Store per length of the string
- 1 to 4000

16. **What are constraints:**
Primary key, foreign key, Not Null, Check.

17. **What are ACID properties in Database?**
A: Atomicity
C: Consistency
I: Isolation
D: Durability

18. **Truncate and Drop** (Both are Auto Commit)
Truncate: Will delete entire data in the table.
Drop: To remove the entire table structure.

19. **Group Functions:**
Avg, Sum, Max, Min, etc.

Core Java:

Questions:

1. **10 Java8 Features**

   1. **Lambda Expressions**: is an Anonymous Functions, with m=or without parameters, return types and possible exceptions.
   2. **Funcationa**l Interfaces
   3. Date, Time, and Date Time API
   4. **Java Stream** API for Bulk Data Operations on Collections
   5. **Optional Class:** it is used to avoid the null pointer Exception.
   6. Default Method and Static Method in Interface: backward compatibility
   7. ForEach () method in Iterable interface
   8. Collection API improvements

9. Concurrency API improvements
10. Java IO improvements

2. **Why do we have default and static methods in java8?**
   - ➔ Default methods helps us to retain the class untouched in there is a need of interface change.
   - ➔ We can override the default method in implementation class, but static method cannot override.

1. **OOPS concepts (what is Date hiding, encapsulation, polymorphism, Inheritance)**

- **Date hiding:** The process of hiding the data from external resources or objects by allowing to access directly is called data hiding.

This can be achieved by using private key word.

```
class Account
{
        private double balance.
        .....................;
        .....................;
}
```

- **Abstraction**: Process of hiding the internal implementations or logic from the outside world and showing only the services.

Example: ATM, CAR, Mobile Phone Etc**.**

- **Encapsulation:**

Encapsulation = Data Hiding + Abstraction
Process of encapsulating both **data member or variables** and **methods** together into a single module is called Encapsulation**.**

  - ▪ In encapsulated class we must maintain getter and setter methods for every data member.

- **Tightly encapsulated class:**

  A class is said to be tightly encapsulated **if and only if every variable** of that class declared as **private** whether the variable has getter and setter methods are not and whether these methods declared as public or not, not required to check.

Example:

```
class Account
{
        private double balance.
        public double getBalance ()
        {
                return balance.
        }
}
```

- **Inheritance** *or* **IS – A Relationship:** is a process for obtaining the behavior of parent class. Can be achieved by using the keyword **extends.**

  The main advantage of inheritance is reusability.

  ```
  class Parent
  {
          public void methodOne()
          {}
  }
  class Child extends Parent
  {
          public void methodTwo()
          {}
  }
  ```

  Real time examples:
  - For all java classes the most required functionality is define inside **object class** hence **object class** acts as a root for all java classes.
  - For all java exceptions and errors, the most common required functionality defines inside **Throwable** class hence Throwable class acts as a root for exception hierarchy.

➔ **Multiple Inheritance** is not possible in java with **classes**, because we cannot extend more than one class at a time. But we can achieve the same by using the **Interfaces**.

➔ **Multiple Inheritance** is possible in java by using the interfaces.
Because an Interface can extend N number of Interfaces.

Java 8: Default and Static methods: - prior java 8 we have only declaration but in java8 …..

Java 9: we can write Private Method in java 9, we can call this method by using the interface public method or default or static.

Association, Aggregation and Composition:

Association: 1 to 1, 1 to many, many to 1 and vice versa.
Aggregation: We have students and belongs to at least any one department
Composition: Composition is tightly coupled.

Can we have more than one Main method:
For executing the standalone application, we need only one main method.
Yes, because Java supports method overloading so one class can have n number of main methods.

Scopes of Access Modifiers:

a. **Public:** Anywhere and **Private:** Same Class only
b. **Default:** Same class, Same package and same package sub class
c. **Protected and Public:** both are same except protected cannot be accessed in different package non sub class.

Why we have called java is not fully Object-oriented language:

- **Why multiple inheritance is possible in java by using Interfaces.**
  Because in interfaces we do not have implementations for the methods we only have declarations. Implementations for these declarations will be specific to implementing classes, hence there is no ambiguity.

- **Has-A Relationship:**
  Can also be called has composition or Aggregation.

Can be achieved by using the **new** key word.
Aggregation: Without any one department many professors can exist.
Composition: University object must exist to have department objects

- **Polymorphism:** Means performing the same action in different forms.
  We can achieve this using 2 ways:
  a. Method overloading or Compile time polymorphism.
  b. Method overriding or Runtime polymorphism.

1. **String (Difference between String, StringBuffer, StringBuilder)**

String	StringBuffer	StringBuilder
String are Immutable	String Buffer are mutable	String Builder are mutable
	String Buffer is synchronized	String Buffer is not Synchronized
	String Buffer is thread safe; means we cannot call a method of string buffer simultaneously	Not thread safe, 2 or more threads can call methods of string builder simultaneously.
	Less Efficiency	More Efficiency
String is a final class; we cannot extend		

2. **What is String Pool? How is String stored in memory? Are strings immutable in java? Why?**
   String pool: A specially designed memory area for the String literals.

3. **What is the meaning of immutability and mutability?**
   Immutability: For example, after creating a string object we cannot change the string again, if we try to do so it will lead in creating new object without any reference.

NOTE: All **Wrapper** and **String classes** are Immutable in JAVA

Mutability: After creating String buffer or String builder object we can edit the object by using append method, thus we called as mutability.

4. **Explain immutability and mutability with an example?**

```
                {
        //Example for immutability
                        String s=new String("Sreeni");
                        System.out.println("S is:"+s);
                        s.concat(" BV");
                        System.out.println("S is:"+s);

         //Example for mutability
                        StringBuffer sb=new StringBuffer("Sreeni");
                        System.out.println("Sb is:"+sb);
                        sb.append(" BV");
                        System.out.println("Sb is:"+sb);
                }
```

5. **What is SCP?**
    - A specially designed memory area for the String literals.

6. **What is the advantage of SCP?**
    - Instead of creating a separate object for every requirement we can create only one object and we can reuse same object for every requirement. This approach improves performance and memory utilization.

7. **What is the disadvantage of SCP?**
    - In SCP as several references pointing to the same object by using one reference if we are performing any changes the remaining references will be inflected. To prevent this compulsory String objects should be immutable. That is immutability is the disadvantage of SCP.

8. **Why SCP like concept available only for the String but not for the StringBuffer?**
    - As String object is the most used object sun people provided a specially designed memory area like SCP to improve memory utilization and performance.
    - But StringBuffer object is not commonly used object hence specially designed memory area is not at all required.

9. **Why String objects are immutable where as StringBuffer objects are mutable?**
    - In the case of String as several references pointing to the same object, by using one reference if we are allowed perform the change the remaining references will be impacted. To prevent this once we created a String object, we can't perform any change in the existing object that is immutability is only due to SCP.
    - But in the case of StringBuffer for every requirement we are creating a separate object by using one reverence if we are performing any change in the object the remaining references won't be impacted hence immutability concept is not require for the StringBuffer.

10. **Like String objects any other objects are immutable in java?**
    - In addition to String all wrapper objects are immutable in java.

11. **Is it possible to create our own Immutable class?**
    Yes.

12. **Explain the process of creating our own immutable class with an example?**

    To create an immutable class, you should follow the below steps:

    1. Make your class ***final,*** so that no other classes can extend it.

2. Make all your fields ***final,*** so that they are initialized only once inside the constructor and never modified afterward.
3. Do not expose setter methods.
4. When exposing methods which modify the state of the class, you must always return a new instance of the class.
5. If the class holds a mutable object:
    - Inside the constructor, make sure to use a clone copy of the passed argument and never set your mutable field to the real instance passed through constructor, this is to prevent the clients who pass the object from modifying it afterwards.
    - Make sure to always return a clone copy of the field and never return the real object instance.

13. What is the difference between final and immutability?

1. final means that you cannot change the object's reference to point to another reference or another object, but you can still mutate its state (using setter methods e.g.). Whereas immutable means that the object's actual value cannot be changed, but you can change its reference to another one.
2. final modifier is applicable for variable but not for objects, Whereas immutability applicable for an object but not for variables.
3. By declaring a reference variable as final, we will not get any immutability nature, even though reference variable is final. We can perform any type of change in the corresponding Object. But we cannot perform reassignment for that variable.
4. final ensures that the address of the object remains the same whereas the Immutable suggests that we cannot change the state of the object once created.

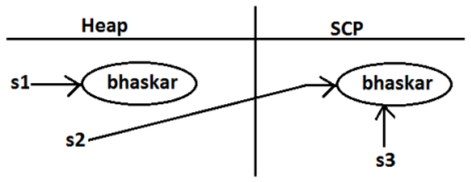
14. What is interning of String objects?

By using heap object reference, if we want, corresponding SCP object reference we should go for intern () method.

Example 1:

```
class StringInternDemo
{
        public static void main(String[] args)
        {
                String s1=new String("bhaskar");
                String s2=s1.intern();
                String s3="bhaskar";
                System.out.println(s2==s3);//true

        }
}
```

<u>Diagram:</u>



If the corresponding object is not there in SCP then intern () method itself will create that object and returns it.

15. How to create an Immutable class in Java?

By using the final key word along with class, we can create Immutable classes.

16. What is Singleton, Factory design pattern? Write an example. Or (GOF) Gang of Four patterns.

There are 3 types of design patterns available:

a. Creational Design pattern
b. Structural Design pattern
c. Behavioral Design pattern

There are 5 design patterns.

a. **Singleton** – Ensures that at most only one instance of an object exists throughout application.

   Example: Printer

b. **Prototype** – Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern.

    a. We create set of objects and store them in HashMap and when required we can get the same by using its ID, instead of creating a new one always.

c. **Factory Method Design pattern** – Creates objects of several related classes without specifying the exact object to be created.
d. **Abstract Factory** – Creates families of related dependent objects.
e. **Builder** – Constructs complex objects using step-by-step approach.

a. Singleton: Ensures that at any given time a Singleton java class have only one instance created.

```
public class SingletonExample {

        private SingletonExample() {}

        private static SingletonExample singletonExample=new
SingletonExample() ;

        /*
         * public static SingletonExample getSIngletonObject() {
         * if(singletonExample==null)
         *      {
         *              singletonExample=new SingletonExample();
         *      }
         *      return singletonExample;
         * }
         */
        public static SingletonExample getSIngletonObject() {

                return singletonExample;
        }
}
```

b. Factory Method design pattern or Factory design pattern:
It is one of the mostly used DP in JAVA.
As the name indicates we use Factory method to produce objects from the factory class which results in low coupling among the classes.

For Example: Give example of Order DAO with 20 fields and 3 API depending on that.

17. Final and Static:

18. Data structures. What is a HashMap? Explain internal working of HashMap.

➔ Answered.

19. Explain Linked list, TreeMap (Comparator & Comparable difference).

➔ Answered.

20. Streams

a. **Intermediate Operations**
b. **Terminal Operations**

Intermediate		
Method	**Purpose**	**Return Value**
Filter	Return stream containing elements matching a Predicate	Stream<T>
Map	Applies an operation against each element in the stream, possibly transforming it	Stream<U>
flatMap	Flattens out a stream	Stream<R>
Sorted	Sorts the elements of a stream	Stream<T>
Distinct	Returns a stream which contains no duplicate elements	Stream<T>
Limit	Restricts the number of elements in the returned stream	Stream<T>
Skip	Skips over a specific number of elements in a stream	Stream<T>

Terminal		
Method	**Purpose**	**Return Value**
forEach	Performs some operation on each element of the stream and terminates the stream	Void
Reduce	Combines stream elements together using a `BinaryOperator`	T
Collect	Puts the stream's elements into a container such as a `Collection`	R
Min	Finds the smallest element as determined by a C`omparator`	Optional<T>
Max	Finds the largest element as determined by a `Comparator`	Optional<T>

Terminal		
Method	**Purpose**	**Return Value**
Count	Determines the number of elements in a stream	Long
anyMatch	Determines whether at least one element of the stream matches a `Predicate`	boolean
allMatch	Determines whether every element of the stream matches. a `Predicate`	boolean
noneMatch	Determines whether there are no elements of the stream matches a `Predicate`	boolean
findFirst	Returns the first element of a stream	T
findAny	Returns any element of a stream	T
Iterator	Returns an `Iterator` for the stream	Iterator
toArray	Creates an array based on the elements of the stream	A[]

21. Stream API & Lambda expressions. Write code and explain. What interface do they implement?

```
List<String> companyList = Arrays.asList("Googlee", "Yahoo", "Microsoft", "Tesla", "",
"TikTok");
        System.out.println(companyList.stream().filter(s->s.isEmpty()).count());
        System.out.println("..............");
        companyList.stream().filter(x->x.chars().count()>6).forEach(System.out::println);
        System.out.println("..............");
        companyList.stream().filter(x->x.startsWith("T")).forEach(System.out::println);
        System.out.println("..............");
        List<String> collect = companyList.stream()
.filter(x->!x.isEmpty()).collect(Collectors.toList());
                System.out.println("..............");
                System.out.println(collect);
                companyList.stream().map(x-
>x.toUpperCase()).forEach(System.out::println);
                System.out.println("..............");
                List<String> collect2 = companyList.stream().filter(x-
>x.chars().count()>6).collect(Collectors.toList());
                System.out.println(collect2);
                List<String> collect3 = companyList.stream().filter(x-
>x.chars().count()>6).collect(Collectors.toList());
                System.out.println(collect3);
```

NOTE: By making use of Functional interface.

22. What is the difference between Application server and Web Server?

Webserver:

- ➔ It provides support for running web related application.
- ➔ It Supports Servlets, JSP's, HTML etc.

Example for webserver:

- Apache Tomcat
- Resin

Application Server: (J2EE tech compatible server)

- ➔ It provides support and environment for running Enterprise application.
- ➔ It Supports Servlets, JSP's, HTML + EJB, JMS etc.

Inside every application server there will be an inbuilt webserver to support for web applications.

Examples of Application Server:

- WebLogic
- JBoss
- WebSphere

S.NO	WEB SERVER	APPLICATION SERVER
1.	Web server encompasses web container only.	While application server encompasses Web container as well as EJB container.
2.	Web server is useful or fitted for static content.	Whereas application server is fitted for dynamic content.

3.	Web server consumes or utilizes less resources.	While application server utilizes more resources.
4.	Web servers arrange the run environment for web applications.	While application servers arrange the run environment for enterprises applications.
5.	In web servers, multithreading is not supported.	While in application server, multithreading is supported.
6.	Web server's capacity is lower than application server.	While application server's capacity is higher than web server.
7.	In web server, HTML and HTTP protocols are used.	While in this, GUI as well as HTTP and RPC/RMI protocols are used.

23. What is the difference between WAR and JAR Files?

- ➔ Jar files are the combination of java class files associated with Metadata and -Resources along with class files.
- ➔ Using command line, we can run Jar files but for WAR we need a webserver.
- ➔ Jar files act as a supporting class for another program.
- ➔ We can Jar using jar command or tools like maven.

- ➔ WAR files are used only to deploy web applications in servlet or JSP container.
- ➔ WAR file contains *WEB-INF* public directory with all the static web resources, including HTML pages, images, and JS files. Moreover, it contains the *web.xml* file, servlet classes, and libraries.

24. Multi-threading?

Executing several tasks simultaneously where each task is a separate independent part of the same program.

We can create a Threads using 2 ways.

a. By Extending **Thread** class
b. By Implementing Runnable Interface.

The main important application areas of multithreading are:

1) To implement multimedia graphics.
2) To develop animations.
3) To develop video games etc.

25. Difference between t.start() and t.run() methods.

In the case of t.start() a new Thread will be created which is responsible for the execution of run() method. But in the case of t.run() no new Thread will be created and run() method will be executed just like a normal method by the main Thread.

26. After starting a Thread, we are not allowed to restart the same thread once again otherwise we will get runtime exception saying "IllegalThreadStateException".

<u>Example:</u>

```
MyThread t=new MyThread ();
t.start();//valid
;;;;;;;;
t.start();//we will get R.E saying: IllegalThreadStateException
```

27. What are executor services?

In simple Java applications, we do not face much challenge while working with a small number of threads. If you have to develop a program that runs a lot of concurrent tasks, this approach will present many disadvantages such as lots of boiler plate code (create and manage threads), executing thread manually and keeping track of thread execution results.p[

Executor framework (since Java 1.5) solved this problem. The framework consist of three main interfaces (and lots of child interfaces) i.e. **Executor**, **ExecutorService** and **ThreadPoolExecutor**.

28. Synchronization

- Synchronized is the keyword applicable for methods and blocks but not for classes and variables.
- If a method or block declared as the synchronized, then at a time only one Thread can execute that method or block on the given object.
- The main advantage of synchronized keyword is we can resolve data inconsistency problems.
- But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.
- Hence if there is no specific requirement then never recommended to use synchronized keyword.
- Internally synchronization concept is implemented by using lock concept.
- Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into the picture.
- If a Thread wants to execute any synchronized method on the given object 1st it must get the lock of that object. Once a Thread got the lock of that object then it's allowed to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock.
- While a Thread executing any synchronized method, the remaining Threads are not allowed execute any synchronized method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [lock concept is implemented based on object but not based on method].

29. Semaphores

A semaphore controls access to a shared resource using a counter. If the counter is greater than zero, then access is allowed. If it is zero, then access is denied. What the counter is counting are permits that allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore.

Working of semaphore: In general, to use a semaphore, the thread that wants access to the shared resource tries to acquire a permit.

30. **how to communicate between threads?**
    wait () -> notify () -> notifyAll ()

31. **How do you sort the elements in HashSet and difference between Hash set and ArrayList?**
    The HashSet class implements the Set interface, backed by a hash table which is a HashMap instance. No guarantee is made as to the iteration order of the set which means that the class does not guarantee the constant order of elements over time.

    It means that HashSet does not maintains the order of its elements. Hence sorting of HashSet is not possible.

    However, the elements of the HashSet can be sorted indirectly by converting into List or TreeSet, but this will keep the elements in the target type instead of HashSet type.

32. **Can we Override 2 static methods. if we do what error we get, compile time, or Run time error.**

    Overloading:
    We can overload or we can have 2 or more static methods with same name but different parameters.

    Overriding:
    Overriding is not supported in Java for static methods so it will always call the Parent class static method.
    Why? Because Overriding will always happen dynamically and during run time. But static methods always bind during compile time, hence static method overriding is not possible.

33. how can you create custom exceptions?

In spring we have one class called `ResponseEntityExceptionHandler`
By extending the class we can our own user defined exception according to our Customer specific requirement.

34. possible to write only try block?

Not possible to write only try without catch or Finally block.

35. Java 7 features.

Underscores in numeric literals:

```
private static final Integer ONE_MILLION = 1_000_000;
```

Strings in switch:

```
switch (requestType) {
   case "displayBalance":
    printBalance(accountId);
    break;
   case "lastActivityDate" :
    printLastActivityDate(accountId);
    break;
   case "amIRich" :
    amIRich(accountId);
    break;
   case "lastTransactions" :
    printLastTransactions(accountId, Integer.parseInt(args[2]));
    break;
   case "averageDailyBalance" :
    printAverageDailyBalance(accountId);
    break;
   default: break;
  }
```

Multi-catch:

```
catch (AccountFrozenException | ComplianceViolationException |
AccountClosedException e)
```

Type inference for typed object creation:

```
List<Transaction> transactions = new ArrayList<Transaction>();
```

we can simply do

```
List<Transaction> transactions = new ArrayList<>();
```

try with resources and suppressed exceptions:

```
try (
 PreparedStatement s = _conn.prepareStatement(sql);
 ResultSet rs = s.executeQuery();
 ) {
```

Collections and Data structures

1. **Collection**
2. **List**:
    1. Array List
    2. Linked List
    3. Stack
    4. vector
3. **SET**
    1. Hash SET
    2. Linked HashSet
4. **Sorted SET**
5. **Navigable SET**
    1. Tree SET
6. **QUEUE**
    1. Priority Queue
    2. Blocking Queue.
7. **Map**
8. Hash Map
9. **Sorted map**
9. **Navigable map**
    1.Tree Map

Collections in Java

Collection: A Collection is a group of individual objects represented as a single unit.
Collection Framework: It's a group of Classes and Interfaces.
Collections in Java reside in the package: import java.util.*;
(**java.util.Collection, java.util.Map**)
Java provides (Collection Framework) several classes and interfaces to represent a group of objects as a single unit.

Before Collection framework (or JDK1.2) comes into existence we have 3 standard ways or methods to represent grouping of the java Objects.

1. Arrays
2. Vectors
3. Hash tables

Drawbacks of Arrays, Vector, Hash tables and Need for Collection Framework.

Arrays:

1. Array will not allow multiple Data types to store. I.e. Only Homogeneous.
2. Array dimensions cannot be resized and manipulating elements of the Array is difficult.
3. Underlying Data Structures are not available, so readymade methods are not available.
4. Memory management wise not recommended.

```
int arr[] = new int[] {10, 20, 30, 40};
// Array instance creation requires [], while Vector and hash table
require ()
// Accessing first element of array: System.out.println(arr[0]);
```

Vectors and Hash Tables:

```
Vector<Integer> v = new Vector();
Hash table <Integer, String> h = new Hash table ();
v.addElement(10);
v.addElement(20);
h.put(1,"Java2Cloud");
h.put(2," Java2Cloud");

// Vector element insertion requires addElement(), but
// Hash table  element insertion requires put()
// Accessing first element of vector and Hash table
//System.out.println(v.elementAt(0));
//System.out.println(h.get(1));
```

1. ArrayList VS LinkedList and When to use and Why.

List: List as the name indicates, elements are stored in ordered sequence. (Duplicates are allowed). But not like in **SET**, where elements are not stored in ordered sequence. (Duplicates are not allowed).

Their main d/f is implementation which causes performance issues based on the operation:

ArrayList:

1. Array List is implemented on data structure called ***Resizable Array or Growable Array.***
2. Its performance on adding and removing elements is little slow compared to LinkedList. But get and set methods are faster compared to LinkedList.
3. Array size will automatically increase based on the addition of elements in to the ArrayList.
4. Array List Elements can be accessible by using Get and Set methods. **Marker interface** used by List implementations to indicate that they support fast **(generally constant time) random access**.

LinkedList:
1. LinkedList is implemented on data structure called Double LinkedList.
2. Its performance on adding and removing elements is better compared to ArrayList. But get and set methods are slower compared to ArrayList.

Vector:
1. Vector is like ArrayList, but it is synchronized.

Note:
1. Vector each time doubles its array size, while ArrayList grow 50% of its size each time. The default initial capacity of an ArrayList is very small so it's a good habit to construct the ArrayList with a higher initial capacity. This can avoid the resizing cost.
2. LinkedList, however, also **implements Queue interface** which adds more methods than ArrayList and Vector, such as offer (), peek(), poll(), etc.

Interview Prospective:

1. Insertions are easy and fast in LinkedList as compared to ArrayList because there is no risk of resizing array and copying content to new array if array gets full which makes adding into ArrayList of O(n) in worst case, while adding is O(1) operation in LinkedList in Java. ArrayList also needs to be update its index if you insert something anywhere except at the end of array.
2. Removal also better in LinkedList than ArrayList due to same reasons as insertion.
3. LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node.
4. Both LinkedList and ArrayList require O(n) time to find if an element is present or not. However, we can do Binary Search on ArrayList if it is sorted and therefore can search in O(Log n) time.

2. How Array List increase its Size:

1. ArrayList works on multiplying factor 1.5 and int newCapacity = oldCapacity + (oldCapacity >> 1).
2. >> is right shift operator which reduces a number to its half. Thus,
   int newCapacity = oldCapacity + (oldCapacity >> 1);
   => int newCapacity = oldCapacity + 0.5*oldCapacity;
   => int newCapacity = 1.5*old Capacity.

3. From Sun's JDK6: ArrayList grows to 15 elements by using the grow () method code in the JDK. By default, the ArrayList size would be 10.

4. newCapacity = 10 + (10 >> 1) = 15.

3. Vector:

1. Vector is Synchronized, means at a given time only one thread can access Vector elements. (Thread Safe)
2. ArrayList is not synchronized, means at a given time many threads can access ArrayList. (Not Thread Safe).
3. Vector and ArrayList increase size in the same way.

4. Enumeration and Iteration can be used to traverse in Vector but only Iterator can be used in ArrayList.

4. Which are the Synchronized collections?

Thread safe Collections -

1. **ConcurrentHashMap:** Thread safe without having to synchronize the whole map Very fast reads while write is done with a lock No locking at the object level Uses multitude of locks.
2. **Synchronized HashMap**: Object level synchronization Both read and writes acquire a lock Locking the collection has a performance drawback May cause contention.
3. Vector
4. Hash table
5. CopyOnWriteArrayList
6. CopyOnWriteArraySet
7. Stack

Rest all are not thread safe.

5. List VS Set VS Map and When to use.

List	Set	Map
List allows duplicates to be added and order is not preserved	Duplicates are not allowed.	Key cannot be duplicate only value can be duplicated
Insertion order is preserved	Insertion Order is not preserved	NA
Concrete classes are ArrayList, LinkedList, Vector	Concrete classes are. HashSet, LinkedHashSet, TreeSet	HashMap, LinkedHashMap, Hash table, TreeMap
Null values are allowed	Allows only once	Allows only values but only one null Key.

When to use:

ArrayList:

1. To retrieve objects from the List. It uses Random Access Interface.
2. Store elements and main the order of insertion.

LinkedList:

For adding and removing elements from the List.

Set:
1. All elements stored in SET, are Unique means no duplicates are allowed.
2. Along with above, SET is having Tree Set which stores elements in sorted order.
3. Comparable and Comparator are used in TreeSet to Sort elements.
4. LinkedHashSet maintains insertion order.

MAP: if we want to store data in the form of Key value pair then we choose MAP.

a. HashMap
b. Hash table
c. Linked HashMap
d. TreeMap

6. **Comparable VS Comparator**

Comparable is a default sorting method used in TreeSet.
It consists of compareTo (), default method.
Comparator consists of methods Compare and Equal, which we must use, to define user defined sorting method.

7. **HashMap VS Hash Table**

- ✓ HashMap is not Synchronized where has Hash table is synchronized.
- ✓ HashMap allows multiple null values and one null key to be inserted but Hash table will not allow either value or key to be inserted.
- ✓ HashMap is preferable over Hash table if thread synchronization is not needed.

8. **Why Hash table does not allow null and HashMap does?**

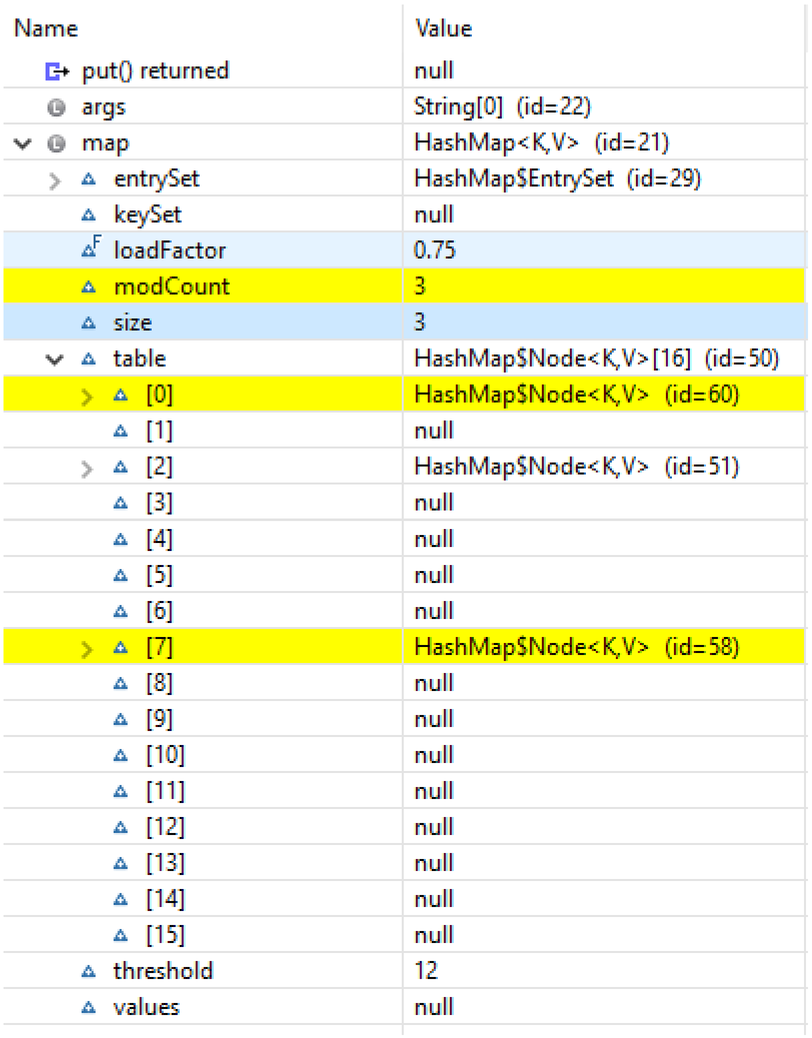
To successfully store and retrieve objects from a Hash table, the objects used as keys must implement the hash Code method and the equals method. Since null is not an object, it can't implement these methods. HashMap is an advanced version and improvement on the Hash table. HashMap was created later.

9. **How HashMap Works internally forget () and put () methods**

As soon as we create HashMap:
Map<String, Integer> map=new HashMap<> ().
JVM will create an array of Nodes, each contains a linked list with space for hash code, Key, Value and Next linked list address.
The maximum size it creates is 16 Nodes, later when reaches a load factor of .75%, it just doubles its size.

Capacity=number of buckets * load factor

To calculate hash code index.
Index = hashcode(key) &(n-1);



Name	Value
put() returned	null
args	String[0] (id=22)
map	HashMap<K,V> (id=21)
entrySet	HashMap$EntrySet (id=29)
keySet	null
loadFactor	0.75
modCount	3
size	3
table	HashMap$Node<K,V>[16] (id=50)
[0]	HashMap$Node<K,V> (id=60)
[1]	null
[2]	HashMap$Node<K,V> (id=51)
[3]	null
[4]	null
[5]	null
[6]	null
[7]	HashMap$Node<K,V> (id=58)
[8]	null
[9]	null
[10]	null
[11]	null
[12]	null
[13]	null
[14]	null
[15]	null
threshold	12
values	null

```
map.put("Sreeni", 12345);
map.put("Sre", 12345);
map.put(null, 12345);
```

In that case, check via hashCode() and equals() method that if both the keys are same.
If keys are same, replace the value with current value.
Otherwise connect this node object to the previous node object via linked list and both are stored at index 6.

For Get():

1. Calculate hash code of Key {"sreeni"}. It will be generated as hashcode.
2. Calculate index by using index method.
3. Go to index of array and compare first element's key with given key. If both are equals then return the value, otherwise check for next element if it exists.
4. In our case it is not found as first element and next of node object is not null.
5. If next of node is null then return null.
6. If next of node is not null traverse to the second element and repeat the process 3 until key is not found or next is not null.

10. What are ConcurrentHashMap and how it works and its internal implementation?

ConcurrentHashMap: This map allows concurrent access to the Map. I.e. A part of the map or Segment (Internal data structure) is only locked and allowing other threads to access the HashMap.

How it Works:
According to ConcurrentHashMap Oracle docs, The constructor of ConcurrentHashMap looks like this:

```
public ConcurrentHashMap (int initialCapacity, float loadFactor, int concurrencyLevel)
```

So the above line creates a new, empty map with the specified initial capacity, load factor and concurrency level.

initialCapacity: Initial capacity of the concurrent hash Map.

concurrencyLevel: the estimated number of concurrently updating threads.

The default values in concurrent hash map is:

```
static final int DEFAULT_INITIAL_CAPACITY = 16;
static final int DEFAULT_CONCURRENCY_LEVEL = 16;
```

Thus, instead of a map wide lock, ConcurrentHashMap maintains a list of 16 locks by default (number of locks equal to the initial capacity , which is by default 16) each of which is used to lock on a single bucket of the Map. This indicates that 16 threads (number of threads equal to the concurrency level, which is by default 16) can modify the collection at the same time, each thread works on different bucket. So unlike Hash table , we perform any sort of operation (update ,delete ,read ,create) without locking on entire map in ConcurrentHashMap.

Retrieval operations (including get) generally do not block, so may overlap with update operations (including put and remove). Retrievals reflect the results of the most recently completed update operations holding upon their onset.

11. Can two threads update the ConcurrentHashMap simultaneously?

Yes, it is possible that two threads can simultaneously write on the ConcurrentHashMap.

12. Can multiple threads read from the Hash table concurrently?

No, multiple threads cannot read simultaneously from Hash table. Reason, the get () method of Hash table is synchronized. As a result, at a time only one thread can access the get() method.

13. What is Concurrent Modification?

When one or more thread is iterating over the collection, in between, one thread changes the structure of the collection (either adding the element to the collection or by deleting the element in the collection or by updating the value at particular position in the collection) is known as Concurrent Modification

14. **Difference between Fail Fast iterator and Fail-Safe iterator**

Fail fast Iterator
Fail fast iterator while iterating through the collection instantly throws Concurrent Modification Exception if there is structural modification of the collection. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Fail Safe Iterator:
Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure. Any structural modification done to the iterator affects the copied data structure. So, original data structure remains structurally unchanged. Hence, no ConcurrentModificationException throws by the failsafe iterator.

15. **What are equals and hash code contract and when to override these methods?**

All Wrapper (Int, Double, …) and String classes implements Equals method.
But if you add any user defined objects in to MAP we have to implement Equals methods by using generate Equals and Hashcode in eclipse.

16. Why we need ConcurrentHashMap when we already had Hash table?

Hash table provides concurrent access to the Map and are synchronized, but the problem here is, it will lock entire Map to perform any sort of operation like update, delete, read, create.

If use it in any web application then to perform one operation other threads must wait, this may lead in performance problems.

17. How TreeMap Works what is internal Data structure it uses.

a. What is tree map?
Tree map is just like Hash map the only d/f is it sorts values in ascending.

TreeMap is a Red-Black tree based Navigable Map implementation. In other words, it sorts the TreeMap object keys using Red-Black tree algorithm.

b. **How TreeMap works in java**?
TreeMap is a Red-Black tree based Navigable Map implementation. In other words, it sorts the TreeMap object keys using Red-Black tree algorithm.

c. **Why and when we use TreeMap?**
We need TreeMap to get the sorted list of keys in ascending order.

18. What is Hash Code?
Hash Code is an integer value and is used to identify an Object. Each object will be having a method called hashcode(), which in turn returns an integer value that represents an Object.
If any 2 or more objects can have same Hash Code, then object will be identified using other attributes of the object like value.

19. **Why do we need hash Code of an Object? What is the use of Hash Code?**
Even though we have many applications of hash Code, one of the important uses is, Hash Map will use Hash Code as its base for storing and retrieving the object.
Any object can be retrieved in O(1) time complexity.

20. How to sort Employee Objects with two fields
We can use **collections.sort()** to achieve this but we need to implement the Comparable Interface to **Employee class**.

```
/Compare by first name and then last name
        Comparator<Employee> compareByName = Comparator
                                   .comparing(Employee::getFirstName)
                                 .thenComparing(Employee::getLastName);

        List<Employee> sortedEmployees = employees.stream()
                                         .sorted(compareByName)
                                         .collect(Collectors.toList());
```

21. How HashMap increase its size when it reaches threshold.

We can increase the size of the HM based on the number of items in the HM. That is based on the threshold or Load Factor.
The decision to increase the size is based on the Load Factor on the Hash Map.
Initial capacity of an HashMap is 16 and when Hash Map size reaches 12 and after inserting the 13th value, Hash Map increses its size by 0.75.
I.e. initial capacity of hashmap * Load factor of hashmap = 16 * 0.75 = 12.

22. What is Rehashing?

Re-hashing is a process of re-calculating the Hash Code for already stored key-value pairs in the hash map.

Why do we need Rehashing?

Re-Hashing will be required once the hash map reaches its threshold that is, when 12 objects or Items are inserted then to keep the time complexity of O(1), hash map creates new Hash Map by doubling the size of the hash Map, then all objects in the old hash will be rehashed in to new hash map.

23. How time complexity of HashMap get() and put() operation is O(1)?

Step1. Computing hashcode of key.
Step2. Calculating array index/bucket from hashcode and then,
Step3. With the help of index calculated, directly jump to that index/bucket.
Step4. Now, each and every element in the bucket is scanned sequentially to see, is there any
key-value pair present, which has the same key we are trying to put.

If key-value pair is found, which has same key then instead of storing new entry / key-value pair, it simply replace value stored against key.
If no matching key is found then it will go till end of the list and create a new key-value pair at the end.

We can say that item lookup inside bucket take expected O(1) time only under the assumptions that good hashcode() function is implemented, Above line means to say that, If hashcode() function is written good then, hashcode generated will distribute the items across all the buckets and doesn't end up putting all item in one bucket.

Say if we have 5 items and 5 bucket then good hashcode method will distribute each item in each bucket, this gives best complexity of O(1) as each bucket have only one item.
So look up required is only once.

Now, if we have 10 items and 5 bucket then good hashcode method will distribute 2 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 2 look up.

Now, if we have 20 items and 5 buckets then good hashcode method will distribute 4 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 4 look up. So far, so good.

Now, if we have 40 items and 5 bucket then good hashcode method will distribute 8 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 8 look up.
Now, performance is little bit degraded.

Now, if we have 80 items and 5 bucket then good hashcode method will distribute 16 items in each bucket, In this case, number of items is doubled, still for searching any element it will require only 16 look up.
Now, performance is little bit degraded.

But, if you observe, as the number of items are doubled, elements that need to be searched within bucket, that is the number of look ups are not increasing very high and remain almost constant compared to number of items increased.
That is why it is called that hashmap's get and put operation takes O(1) time.

“Remember, hashmap's get and put operation takes O(1) time only in case of good hashcode implementation which distributes items across buckets.”

24. How Hashmap data structure works internally? How hashcode and equals method work in hashmap? Explain with real time example.

To understand HashMap working, first we need to understand how Hashcode() and Equals() Methods works.

2. Each hashmap Contains 16 Nodes (0 to 15).
3. Each node is containing null values before inserting in to hashmap.
4. Each node contains a linked List to store key, value.
5. When put operation is called:
    a. Hashmap.put("hello","world");
       Key:hello and value:world

Here hashmap generates the hashcode for the key using hashcode() method.

Say for example we go an hashcode of 12234 for key: hello

But we have only index from 0 to 15 in the hashmap, so this hashcode will be converted to get an index using:

indexFor(hash, table.length)

After getting the index, key value will be stored in that index.

Same will be repeated for **get(key)** method also.

Note: If we derive the same hashcode for different key then separate node will be created with key value.

While retrieving the using the **GET(key)** method**,** hashcode will generated for the key using the **hashcode()** method and then indexFor() method for the index and then **Equals()** method() will be used to compare all the nodes key value in that index.

HashMap uses Array and Linked list data structure internally for storing key-value pair.

25. HashMap VS ConcurrentHashMap.

- HashMap is non-Synchronized in nature i.e. HashMap is not Thread-safe whereas ConcurrentHashMap is Thread-safe in nature.

- HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously. But ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.
- While one thread is Iterating the HashMap object, if other thread try to add/modify the contents of Object then we will get Run-time exception saying ConcurrentModificationException.Whereas In ConcurrentHashMap we won't get any exception while performing any modification at the time of Iteration.

26. What is "this" Key word?

Its reference variable that reference to the current object.

27. What is the difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

28. What is Static Import?

Usually if we want to use static variables, we use Class Name followed by Static variable to access the data in that static variable.
But by using static import we can use directly.
Ex.
Import static java.lang.System.*;
So that we can directly use out.println("hello");

Exception

29. Exceptions:

1. Exception (Checked): Checked during compilation
2. Unchecked Exception: Checked during run time.
3. Error: happen due to system errors.

a. Multiple inheritance:

Multiple Inheritance is not possible with Class in java, same can be achieved via Interfaces.
i.e. **Class extends A, B** → is compile time error

Why multiple inheritance is possible via Interfaces
Implementation is provided by the implementation class so there will be no ambiguity.

Marker Interfaces:

1. Serializable Interface
2. Cloneable Interface
3. Random Access Interface
4. SingleThreadModel

While using TreeSet(I), JVM will automatically call the Comparable(I).
Obj1.compareTo(Obj2):
    -1 will be returned if Obj1 comes first than Obj2
    +1 will be returned if Obj2 comes first than Obj1
0 if both are same.
A. compareTo (B): -1 is returned

Comparable is default sorting for string and not or string buffer
Methods are:
CompareTo ().

Comparator Interfaces: is used for customized sorting
Methods are:
Compare () and Equal ()

Name some of the Immutable classes:
➔ Int, Short, Boolean… All wrapper classes and Sting class are Immutable.

Transient, Strict FP, Volatile:
➔ Transient will be used to avoid serialization and to not send the object via network.
➔ Strict FP: To keep the value for a variable same among all the platform we must use Strict FP.
➔ Volatile: To make a variable or method available to all the threads by storing the value in main memory we must use volatile key word.

30. Solid principals.

- ❖ **S: Single responsibility Principle**: One class should perform only one task, like reading a file or connecting to database or writing to a file. A class should do only one job to be called as Single responsibility principle.

- ❖ **O: Open – Close principle**: Open for Extension and Closed for Modifications. Example is Browser

- ❖ **L: Liskov Substitution principle:** Every sub class or derived class would be a substitute for the base class.

- ❖ **I: Interface Segregation principle:** Like no class should ever force to implement an interface even though the methods in the interface is not really sed by that class.

- ❖ **D: Dependency Inversion Principle:** According to this the Entities should always depend on the Abstractions but not on the Concretions.

31. For spring boot which is the minimum version of java should be.

- ❖ Java 1.8

32. which version of java recently launched.

- ❖ Java 15 in September

33. what features available in java 1.8?

- ❖ Lambda Expressions
- ❖ Stream API
- ❖ Date and Time API
- ❖ Functional Interface
- ❖ Interface Default and Static method

34. String Builder works on which design patterns?

Builder Design pattern.

35. Having a stream: " My name is Sreenivas”. i want to search how many times A came; B came and so on … and how can we do that.

```
public class howManyTimesAAndBComes {
	public static void main(String[] args) {
		String S="my name is sreenivas";
		char[] chars=S.toCharArray();
		HashMap<Character, Integer> hm=new HashMap<>();
		for (int i = 0; i < chars.length; i++) {
			int count=1;
			if(hm.containsKey(chars[i])) {
				count=hm.get(chars[i]);
				hm.put(chars[i], ++count);
			}
			else
				hm.put(chars[i], count);
		}
		System.out.println(hm.toString());
		}
}
```

36. **Threads:**

1. What are the **2 ways or Methods** for creating Threads in java?
   => Extend the Thread class.
   => implement the thread class.
2. **volatile**: is used to make a variable available to all the threads, it will declare this variable in the main memory so that all threads will access that variable.
3.**caching variables in threads:** if 2 or more threads tries to alter the same variable, some times that variable would be cached so it will not tale the altered value instead it will refer to cached memory and thread starts work on that to avoid we can use volatile.
4. **Synchronized**: Can be applied along with the methods. So whenever a thread tries to access the method which is of synchronized method, first the calling thread will acquire a Lock called intrinsic lock, so that if any other thread tries to access the same method it has to wait until first thread get complete its task.

37. What is the difference between maven and Gradle?
In gradle we can write own scripts, and we can create our own tasks like build or clean. We can use this to perform core processing tasks.

It’s completely like a groovy script, we can perform all the tasks like how we do it in Groovy.

38. **String Joiners in java**

It is a Class in a **Sting.Util** package, which is used to join strings with delimiter and with optional supplied suffix and prefix (braces).
We have 2 types of constructor available for String Joiner.

1. StringJoiner (CharSequence delimiter)
   Example: StringJoiner sj1=new StringJoiner (“,”) ;

2. StringJoiner (CharSequence delimiter, CharSequence prefix, CharSequence suffix)
   Example: StringJoiner sj2=new StringJoiner (“,”,” [“, ”]”);

   Methods
   1. add(string)
   2. toString ()
   3. merge ()
   4. length ()
   5. setEmpltyValue ()

39. String tokenizer class

Used to tokenize the string submitted using the delimiter, the default delimiter is space, next line, or tab. We have 2 constructors in string tokenizer.

40. Why Default Methods are Introduced.

- Existing code does not have to be updated to implement a new method. This allows for interfaces to grow or be extended.
- To provide additional flexibility in the design of an inheritance hierarchy such as:
  - Providing additional options in the placement of functionality within an interface hierarchy
  - Supporting interfaces that mimic adapter class type functionality
- To support lambda expressions

41.
42. Dfsdf
43. Sdfs

44. Sdfsd
45.

Micro Services and Spring Boot related Questions:

1. **What is a Web Service?**
   It is a service over a web among group of applications with interoperable and over a network. (Request and response among the application irrespective of application they are developed).

2. **Ways in which date exchange takes place among applications.**
   XML and Json format

3. **What is service definition?**
   It is will have all the details like request/ response format, structure of the request, structure of the response and end Point (URL)

4. **How transportation happen for this webservices?**
   By 2 ways
   a. HTTP for URL based request (Web)
   b. MQ for communication over a Queue

5. **What are the uses of Micro Services or What do you mean by micro services?**
   - Micro services or Microservices Architecture is a collection of small autonomous applications, each work individually with separate DB.
   - Here each application or service is individual but related to each other.

6. **What is Rest API? (**Roy fielding**): Rest is** Representational state transfer Protocol.

Communication via Internet using the HTTP and its methods (get, put, post, and delete) and status codes with below.

- It is not environment specific.
- It is not language Specific.
- Should allow communication over the internet. (HTTP).
- Which should support all the XML and Json etc.

7. **Technologies you are using in your project.**
    - ❖ Java 1.8, Spring boot 2.1.4
    - ❖ Maven and Hibernate.
    - ❖ **Spring Cloud**
        - Eureka: Naming server
        - Zuul: API Gateway Service
        - Feign Rest Client
        - Ribbon: load balancing
        - Spring Cloud Sleuth

8. **How many micro services are there in your project?**
    - 20+ services:
    - Order, shipment, Manifest service, cancel shipment., print service, label service.

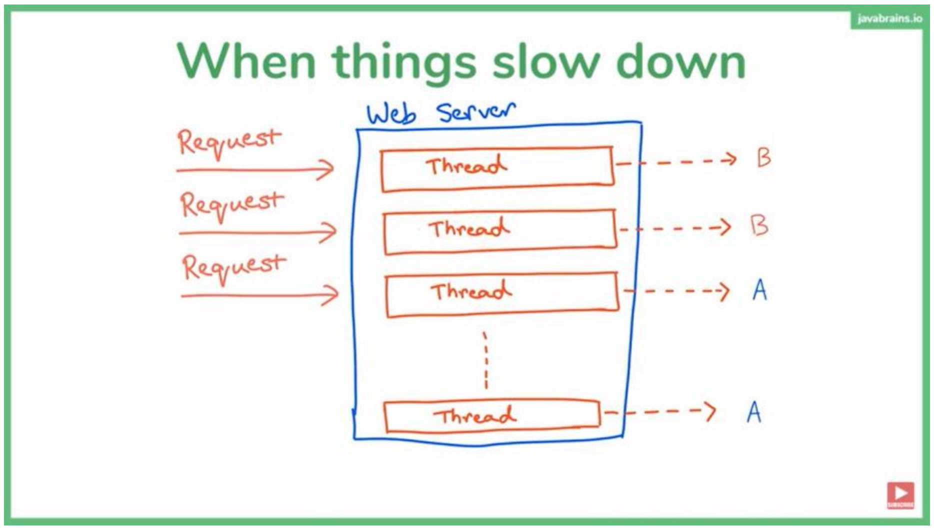
9. **So, each micro service is having its Own DB or different microservices?**

    Per Microservice architecture we should have separate DB but in out project we are using same DB, in **micro services world we will be having each DB for each micro service.** Yes, that would be standard way of developing.

10. **Issues with Microservices.**

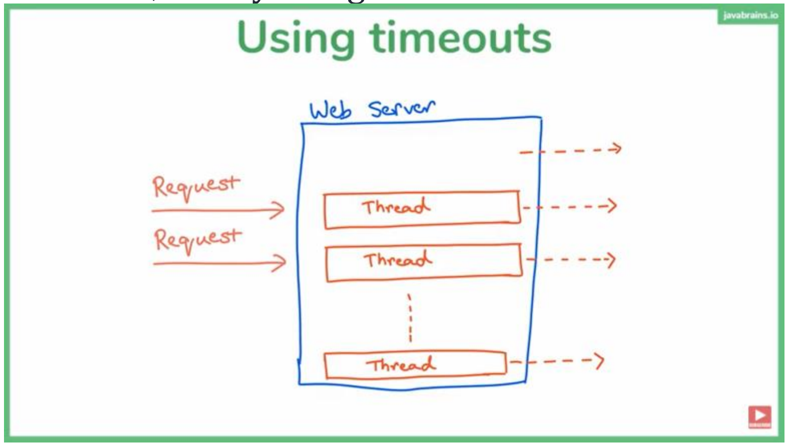
    a. If Microservice Service instance is going down.
    b. If Microservice Service instance is Slow.
    c. Configuration
    d. pack of cards (if any brake down happens between services)
    e. Increase and decrease instances based on load.
    f. Tracking and Monitoring the services

In below scenario as long as we have thread count not reached the threshold limit we are good for all other services if not, other services also will become slow.

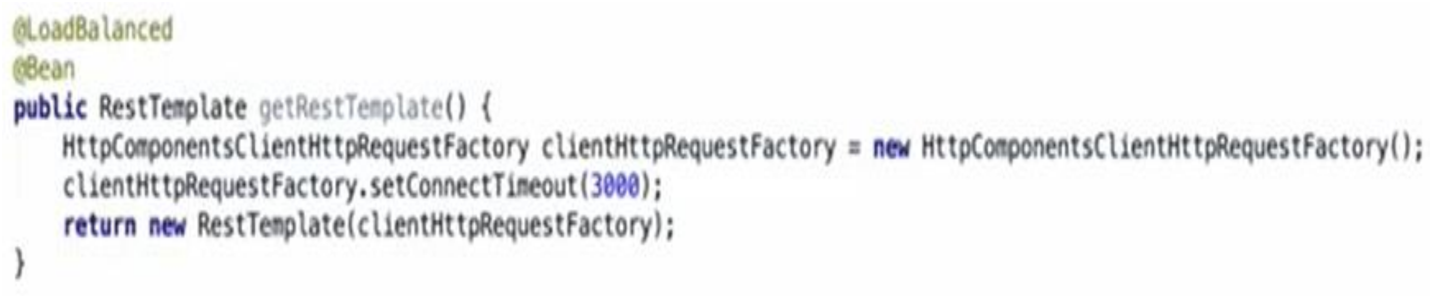


11. How to solve the above problem.

Time Out, Yes by using the Time out



But how to enable or set Time Out in `RestTemplate restTemplate=new RestTemplate();`.

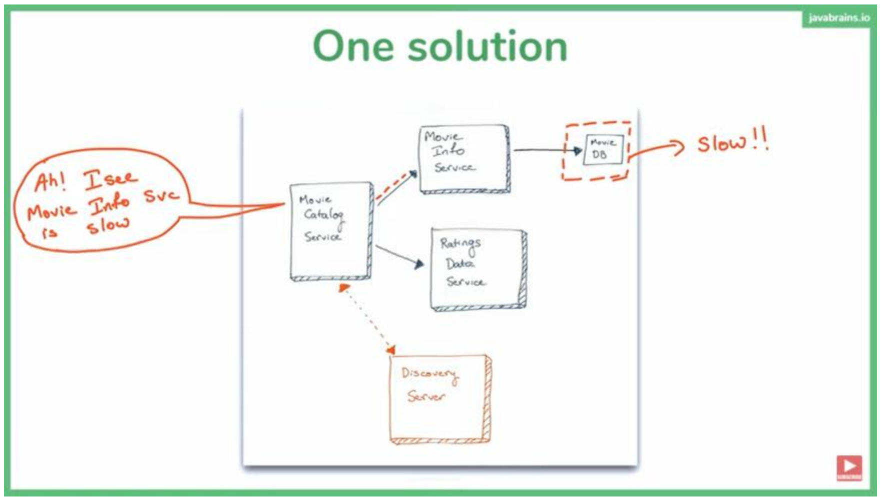


```java
@LoadBalanced
@Bean
public RestTemplate getRestTemplate() {
    HttpComponentsClientHttpRequestFactory clientHttpRequestFactory = new HttpComponentsClientHttpRequestFactory();
    clientHttpRequestFactory.setConnectTimeout(3000);
    return new RestTemplate(clientHttpRequestFactory);
}
```

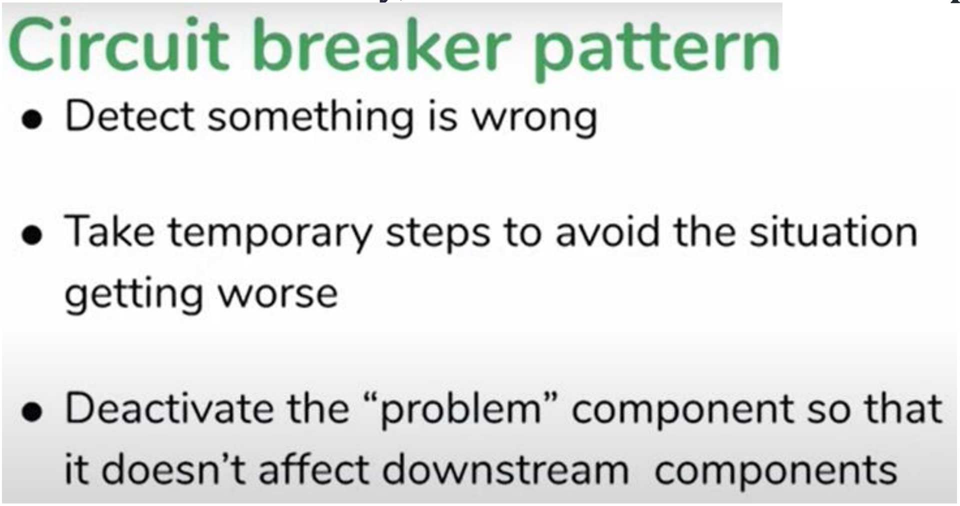
Ok , SO does this solves our problem ?

But still it does solve the issues fully, it solves only little like to say only for 3 secs, if we get multiple requests in the same time, its again the same problem.

So how to solve this problem.

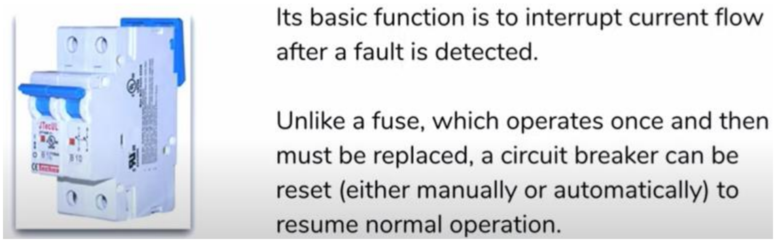


To solve this issue fully, we have followed the below steps:



Circuit breaker pattern

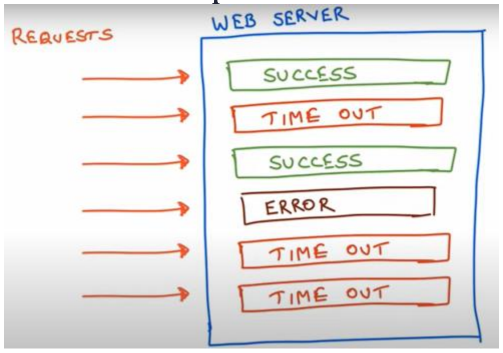
- Detect something is wrong
- Take temporary steps to avoid the situation getting worse
- Deactivate the "problem" component so that it doesn't affect downstream components



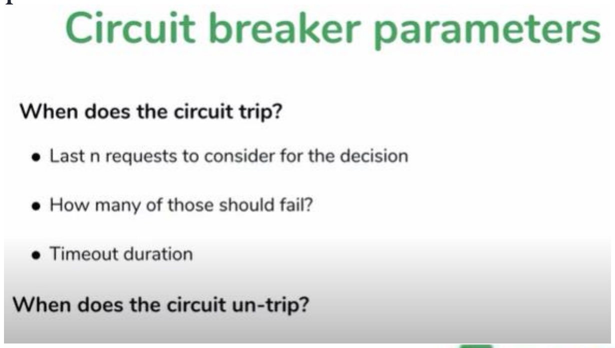
Its basic function is to interrupt current flow after a fault is detected.

Unlike a fuse, which operates once and then must be replaced, a circuit breaker can be reset (either manually or automatically) to resume normal operation.

So how to implement the above circuit breaker in Microservices:



If we have a time out or Failure then check for some of the last requests, if last N requests are failed or time out, then Circuit breaker will come in to the picture.

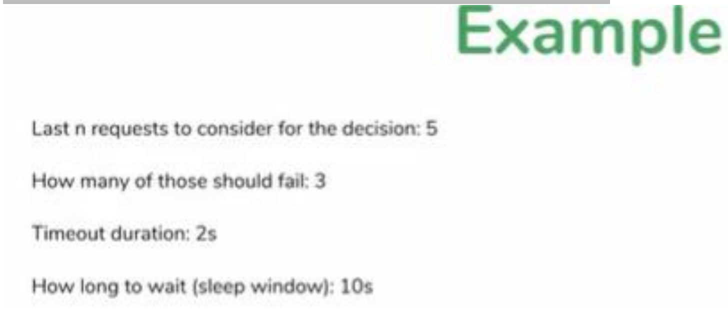


Circuit breaker parameters

When does the circuit trip?

- Last n requests to consider for the decision
- How many of those should fail?
- Timeout duration

When does the circuit un-trip?



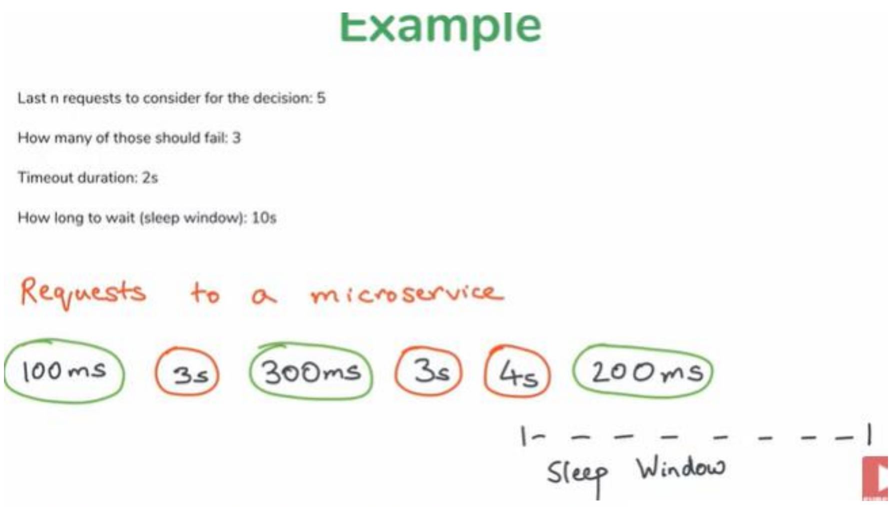
Example

Last n requests to consider for the decision: 5

How many of those should fail: 3

Timeout duration: 2s

How long to wait (sleep window): 10s



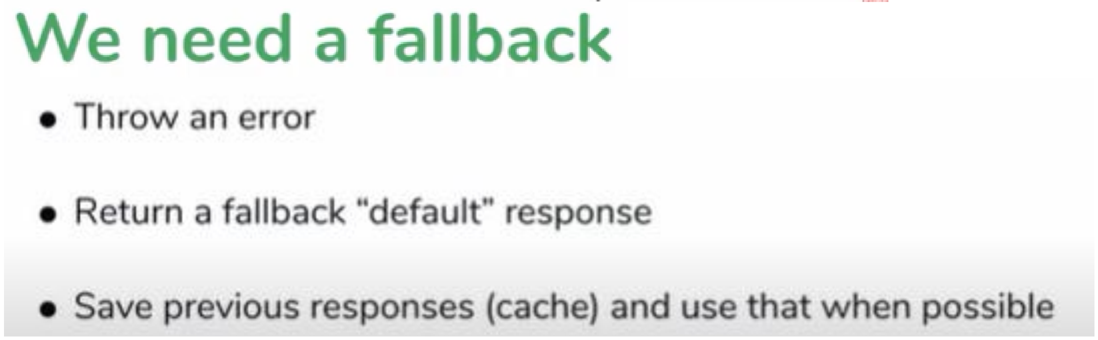
Example

Last n requests to consider for the decision: 5

How many of those should fail: 3

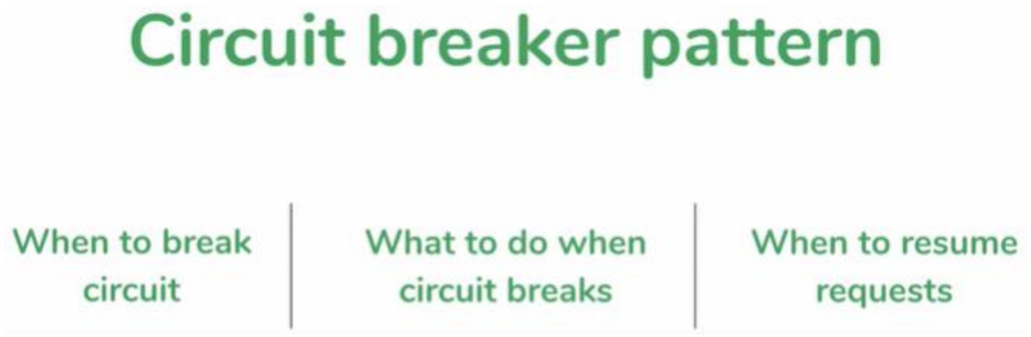
Timeout duration: 2s

How long to wait (sleep window): 10s



We need a fallback

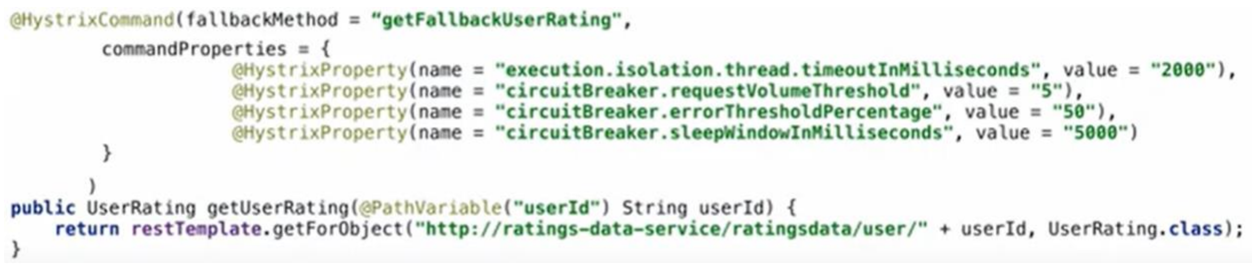
- Throw an error
- Return a fallback "default" response
- Save previous responses (cache) and use that when possible



Circuit breaker pattern

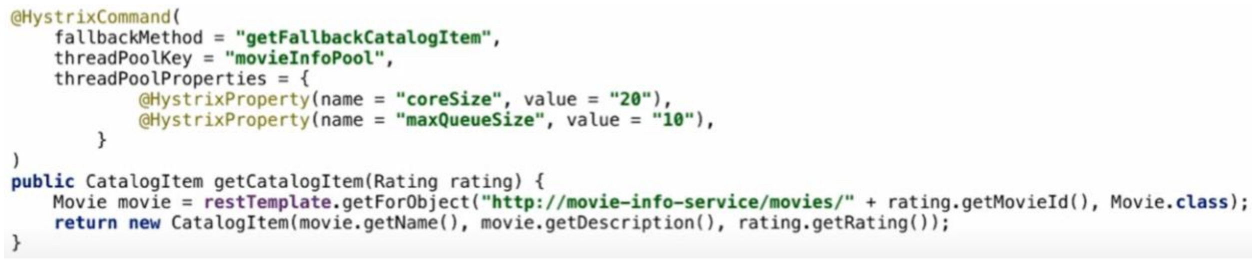
When to break circuit	What to do when circuit breaks	When to resume requests

Configuring Hystrix properties.



```
@HystrixCommand(fallbackMethod = "getFallbackUserRating",
        commandProperties = {
                @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000"),
                @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "5"),
                @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "50"),
                @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "5000")
        }
       )
public UserRating getUserRating(@PathVariable("userId") String userId) {
    return restTemplate.getForObject("http://ratings-data-service/ratingsdata/user/" + userId, UserRating.class);
}
```

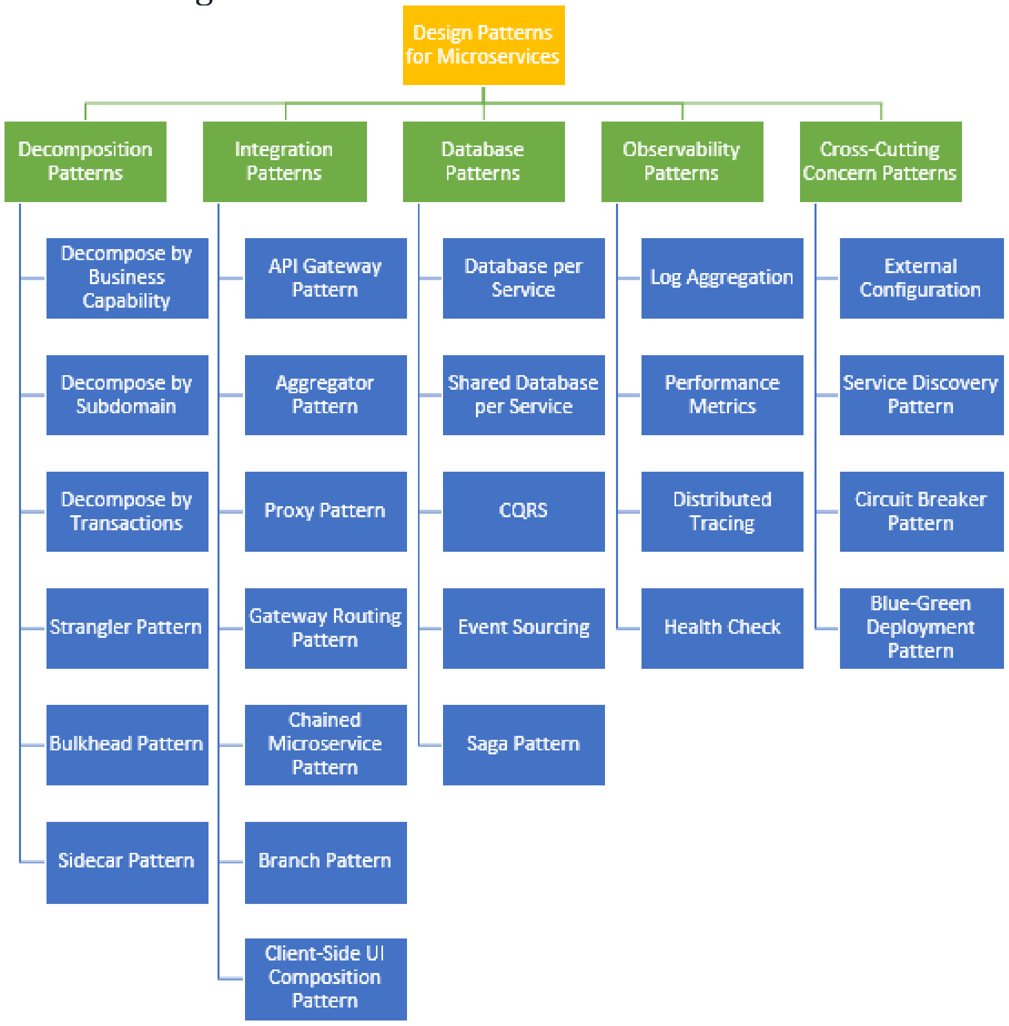
Configuring Bulkhead pattern.



```
@HystrixCommand(
    fallbackMethod = "getFallbackCatalogItem",
    threadPoolKey = "movieInfoPool",
    threadPoolProperties = {
            @HystrixProperty(name = "coreSize", value = "20"),
            @HystrixProperty(name = "maxQueueSize", value = "10"),
        }
)
public CatalogItem getCatalogItem(Rating rating) {
    Movie movie = restTemplate.getForObject("http://movie-info-service/movies/" + rating.getMovieId(), Movie.class);
    return new CatalogItem(movie.getName(), movie.getDescription(), rating.getRating());
}
```

12. **Micro services Design patterns.**

- Decomposition Pattern
- Integration patter n
- Database pattern
- Observability pattern
- Cross-Cutting Pattern



13. What is the use of Zuul API Gate Way?

- Zuul acts as an API gateway or Edge service. It receives all the requests coming from the UI and then delegates the requests to internal microservices. So, we must create a brand new microservice which is Zuul-enabled, and this service sits on top of all other microservices. It acts as an Edge service or client-facing service. Its Service API should be exposed to the client/UI. The client calls this service as a proxy for an internal microservice, then this service delegates the request to the appropriate service.
- Main advantage of this type of service is authentication, and security can be put into a centralized service.

14. What is the use of Eureka Server?

- Eureka Server is an application or a service in MS architecture that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.
- **In Eureka server:** @EnableEurekaServer
- **In Client or other service:** @EnableDiscoveryClient

15. What is the use of Ribbon?

- For load balancing among the services, we use.

16. What is the use of Circuit breaker and Hystrix?

- Hystrix is the implementation of Circuit Breaker pattern, which gives a control over latency and failure between distributed services. The main idea is to stop cascading failures by failing fast and recover as soon as possible — Important aspects of fault-tolerant systems that self-heal.

17. What is the use of SLUETH (Used for login or tracking purpose)?

- One of the problems developers encounter as their microservice apps grow is tracing requests that propagate from one microservice to the next. It can quite be daunting to try and figure out how a requests travels through the app, especially when you may not have any insight into the implementation of the microservice you are calling.

- ❖ Spring Cloud Sleuth is meant to help with this exact problem. It introduces unique IDs to your logging which are consistent between microservice calls which makes it possible to find how a single request travels from one microservice to the next.

18. What is actuator.

- ❖ Actuator provides Production-ready Features choose to manage and monitor your application by using HTTP endpoints.

---

19. In Spring How bean life cycle works.

When we define @Component, @Service, @Repository, @Controller, In Spring boot the life cycle of a bean will be managed by the Spring IOC (Inversion of Controller)

20. In Spring what is the use of Qualifier?

@Qualifier annotation is used to resolve the ambiguity when resolving a bean at run time.

Ex.

```
Public Interface SortingAlgo {
        Public int [] sort (int [] numbers)
}

Public BubbleSortAlgorith implements SortAlgo {
}
@Qualifier
Public QuickSortAlgorith implements SortAlgo {
}
```

Now it will resolve QuicksortAlgo at run time.

Same can be achieved by sing **@Primary** and

@Autowired
Private SortingAlgo BubbleSortAlgorith.

21. what is the difference between @Controller and @RestController?

The @RestController annotation in Spring MVC is nothing but a combination of the @Controller and the @ResponseBody annotation.

It was added into Spring 4.0 to make the development of RESTful Web Services in Spring framework easier. If you are familiar with the REST web services you know that the fundamental difference between a web application and a REST API is that the response from a web application is a generally view of HTML + CSS + JavaScript while REST API just return data in form of JSON or XML.

This difference is also obvious in the @Controller and @RestController annotation. The job of the @Controller is to **create a Map of model object** and find a view but the @RestController simply returns the object and object data is directly written into HTTP response as JSON or XML.

22. What is the user of @EnableAutoConfiguration?

Many Spring Boot developers like their apps to use auto-configuration, component scan and be able to define extra configuration on their "application class". A single `@SpringBootApplication` annotation can be used to enable those three features, that is:

- `@EnableAutoConfiguration`: enable Spring Boot's auto-configuration mechanism
- `@ComponentScan`: enable `@Component` scan on the package where the application is located (see the best practices)
- `@Configuration`: allow to register extra beans in the context or import additional configuration classes

The `@SpringBootApplication` annotation is equivalent to using `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` with their default attributes,

23. Which Build tool you are using in your project?

Maven

24. When we are defining the spring boot starter project, many dependencies will be coming into the project how to remove the one we do not want.

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>sample.ProjectA</groupId>
      <artifactId>Project-A</artifactId>
      <version>1.0</version>
      <scope>compile</scope>
      <exclusions>
            <exclusion>  <!-- declare the exclusion here -->
                  <groupId>sample.ProjectB</groupId>
                  <artifactId>Project-B</artifactId>
            </exclusion>
      </exclusions>
    </dependency>
  </dependencies>
</project>
```

25. I have a DB, and table Student, so in this student table i have inserted duplicate records, how can I remove the duplicate records.

By using the **Common table Expression**, we can create another column, and, on that column, we can work to delete the duplicate records from the table.

```
WITH studentsCTE AS
(
```

select *, ROW_NUMBER () OVER (PARTITION BY ID ORDER BY ID) AS ROWNUMBER FROM students
)
delete from studentsCTE where ROWNUMBER >1.

26. Select Nth Highest Salary from an Employee Table.
To find nth highest salary using Sub-Query

SELECT TOP 1 SALARY FROM (SELECT DISTINCT TOP N SALARY FROM EMPLOYEES ORDER BY SALARY DESC) RESULT ORDER BY SALARY

27. I Have an Array and having a size of over 1 Million and inserted values from 1 to 10 repeatedly and I want to sort that array, how to sort it.

28. If your using Rest Web services How do you create new End Point?
By using any of the rest methods like GetMapping (), Put, Delete, Post, Patch Mapping ("///")

29. What is the architecture of your current application?
Not Sure

30. How did you map your DB or what kind of data access layer used for mapping to DB.?
Using and DAO and JPA.

31. What about Load Balancing or Client-Side Load Balancing? Or why do we need load balancing in your application?

We generally create a service discovery like Eureka or Consul, where each service instance registers when bootstrapped. Eureka server maintains a service registry; it maintains all the instances of the service as a key/value map, where the {service id} of your microservice serves as the key and instances serve as the value. Now, if one microservice wants to communicate with another microservice, it generally looks up the service registry using DiscoveryClient and Eureka server returns all the instances of the calling microservice to the caller service. Then it was a caller service headache which instance it calls. Here, client-side load balancing stepped in. Client-side load balancing maintains an algorithm like round robin or zone specific, by which it can invoke instances of calling services. The advantage is s service registry always updates itself; if one instance goes down, it removes it from its registry, so when the client-side load balancer talks to the Eureka server, it always updates itself, so there is no manual intervention- unlike server side load balancing- to remove an instance.

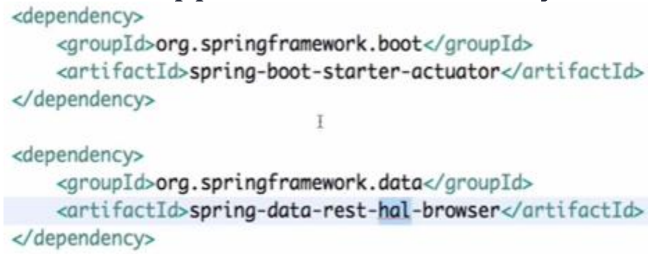
Another advantage is, as the load balancer is in the client side, you can control its load balancing algorithm programmatically. Ribbon provides this facility, so we will use Ribbon for client-side load balancing.

32. What is an Actuator?

An Actuator is a Monitoring tool to monitor all the activities that is happening in our service using the default end points provided by the actuator.

Some of the End points are:

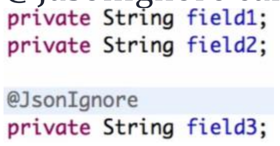
Logs, health, info, how many times service got invoked, how many its service stopped or failed and many other metrics of the service.



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-browser</artifactId>
</dependency>
```

33. Filtering the content from backend to frontend

@JasonIgnore can be used to ignore this field in response.



```
private String field1;
private String field2;

@JsonIgnore
private String field3;
```

34. I have list of employees and I want to convert them in to map?

```
List<Employee> empList=Arrays.asList(
                    new Employee (100,"sree",new Date()),
                    new Employee (101,"bv",new Date()),
                    new Employee (103,"bv",new Date()));

        HashMap<Integer, Employee> empHashmap=new HashMap<>();
        empList.stream().forEach(item -> {
            empHashmap.put(item.getId(), item);
});
```

35. What is the concurrent modification Exception and when it will occur?

For example, if there is a collection and some thread is already reading the data from the collection and in the same time if another thread tried to modify the same collection, then we get concurrent modification exception and so called Fail Fast

36. Why is list an Asynchronous.

37. Maven Life Cycle

- `validate` - validate the project is correct and all necessary information is available
- `compile` - compile the source code of the project
- `test` - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- `package` - take the compiled code and package it in its distributable format, such as a JAR.
- `verify` - run any checks on results of integration tests to ensure quality criteria are met
- `install` - install the package into the local repository, for use as a dependency in other projects locally
- `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

38. Where Jar file will be created

Build Path

39. Junit testing

40. Difference between Interface and Abstract class.

1. **Type of methods:** Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
2. **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
3. **Type of variables:** Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.

4. **Implementation:** Abstract class can provide the implementation of interface. Interface can't provide the implementation of abstract class.
5. **Inheritance vs Abstraction:** A Java interface can be implemented using keyword "implements" and abstract class can be extended using keyword "extends".
6. **Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
7. **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

41. Real time scenario on Interface and Abstract class

42. can we have 2 primary keys in a single table?

No But can have multiple fields combined to form a key.

43. what is Unique key

Unique Key in SQL. A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table. You can say that it is little like primary key, but it can accept only one null value and it cannot have duplicate values

44. Transient

Answered

45. **Static key word: can we access non static method from static block and vice versa.**

We cannot call non static methods from static block and if you want to call one, use object of same class to call the method from static block. But we can call static methods from Non-Static method.

46. Exception handling in Spring boot.

In 2 ways:

a. Extend ResponseEntityExceptionHandler and override one of the methods based on the requirement.
b. Create an own exception response class with attributes.
c. Use HTTP Status or Project specific codes to send error codes

Or

Just extend an exception class and define the variables required for you and throw that exception class when even you want to throw.

@Controller Advice is the annotation that helps to apply the exception method to all the controllers in the project.

47. **What is Internationalization:** Configuring the messages to locations
(US, Canada, France Etc.)

48. Content Negotiation:
All the content negotiation will happen from Json to Objects and vice versa due to **Jackson**, if we want to have the response in XML conversions then add the below dependency.
<dependency> Jackson-data-format-XML</dependency>

49. How to enable swagger:
Add the below dependencies to pom.xml
```
 compile("io.springfox:springfox-swagger2:2.9.2")
compile("io.springfox:springfox-swagger-ui:2.9.2")
```

Annotations used in Spring boot.

@SpringBootApplication
    @EnableAutoConfiguration
    @ComponentScan
    @@Configuration
@RestController
    @Service
    @Controller
    @Repository
@RequestMapping
@ResponseBody
@GetMapping
@PostMapping
@PutMapping
@PatchMapping
@Advice

```
        @ExceptionHandler
        @RestControllerAdvice
        @ControllerAdvice
        @Entity
        @Id
        @OneToMany (Fetch=FetchType.LAZY, cascade=cascade. ALL, mapped
        by=”class Object”
        @ManyToOne
        @joinColumn (name=” Primary Key”, nullable=false)
        @Column
```

```
class MdfeEventMaster
{
        @OneToMany (fetch=FetchType.LAZY, cascade=cascadeType.ALL,
        Mapped By="MdfeEventMaster")
        private List<MdfeEventDetails> mdfeEventMaster.
}

class MdfeEventDetails
{
        @ManyToOne
        @joinColumn (name="primary Key", nullable=false)
        private MdfeEventMaster mdfeEventMaster
}
```

Questions about Layers (Module, View, Controller Layers)

- MVC is design pattern and in this the entre application been divided in to 3 layers called above.

- **Model**: DTO, DAO (These are also called Design pattern), Repo
  i.e. Model is associated with Database
- **View**: Front End (All responses and I/p): presentation layer where we can see output.
- **Controller:** this layer will be handled by **Dispatcher Servlet** and is also called as Front Controller

Questions about Hibernate

Questions about Primary Key usage in Hibernate

Scenarios given to explain the usage of Primary key and Foreign key

Questions about combination of Index and Primary key:

Questions about Log-in in a web page.

- We must POST, and details will go via Method Body.
- Request return 200 means Valid User and Password, if it 404 means User is not Found.

Example – how u write or process behind below scenario

§ After entering username and password in the log in page, we click submit button and wait for response. If response delayed user will click submit button twice or thrice. What is the process for disabling a submit button until response comes for the earlier click? if the submit button must be disabled after pressing once until a response comes.
Once user click Login Button for the first time we can disable until we get response.

Process behind once we click submit after username and password
Answered

Password stored procedures in database.
Data is stored DB is of encrypted format, whenever we have login, first get password for that user from DB and we decrypt it and match with user Given password. The encrypted format is BASE64 (API)

Example:

```
        // Encode into Base64 format
```

```
 String BasicBase64format
= Base64.getEncoder()
      .encodeToString(sample.getBytes());

// decode into String from encoded format
byte[] actualByte = Base64.getDecoder()
                    .decode(encoded);
```

Process behind storage a password
By Decoding by using the BASE64 API for Encryption and decryption

Questions about API

- **Difference between Https and API webpages (HTTP)**

  The only **difference between** the two protocols **is** that **HTTPS** uses TLS (**SSL: Secure Socket layer**) to encrypt normal HTTP requests and responses. As a result, **HTTPS is** far more secure than HTTP. A **website** that uses HTTP has http:// in its URL, while a **website** that uses **HTTPS** has **https**://.

How you write query for a specific scenario in Java
If our requirement is not meet by JPA provided repository then we will go for JPQL, where we can define our own queries.
Or
We can use Javax Persistence Entity Manager.

Example - How you fetch data from table
FindById
FindAll

why Java is called platform independent?
It produces Byte cod after compilation hence we can use that byte code in any of the platform.

What is Encryption?
Base64

What is session management?
It's a mechanism used by the web container to store the session information for a user.

Session management can be achieved in one of the following ways-

- Cookies
- Hidden form field
- URL Rewriting
- HttpSession

we will be using the **Spring Session module**

Spring Session consists of the following modules:

- **Spring Session Core -** provides core Spring Session functionalities and APIs
- **Spring Session Data Redis -** provides SessionRepository and ReactiveSessionRepository implementation backed by Redis and configuration support
- **Spring Session JDBC -** provides SessionRepository implementation backed by a relational database and configuration support
- **Spring Session Hazelcast -** provides SessionRepository implementation backed by Hazelcast and configuration support

We will use Spring session JDBC and Tomcat container (Web Container) by default uses HttpSession Objects in memory.

Spring Boot and Rest API

POM : Project Object Model XML file
STS : Spring Too, Suite

what is spring Bean?
-> Elements/tag defined in the Spring bean Configuration File. OR
-> These are Objects that are managed by our Spring application container.

Why do we use Beans?
-> We use beans to **create instances of the classes** through our spring application.

How bean Mapping happens in spring.
-> By mapping (wiring up the beans in Config file) the application classes with unique identifiers that
spring can use.
* Unique Id
* Reference to a class
*Class should have default constructor

What is Spring bean Scope?
-> **Singleton**: Only one Bean or instance is created for that Spring container.
-> **prototype**: Multiple instances per IOC Container
Means Spring creates multiple beans and handed over to user and spring will no longer manage them.
General rule is we will user Singleton for stateless bean and prototype for stateful bean when we create applications.
-> **requestScope**: One instance per user request.
-> **SessionScope**: One instance is created for that session.
-> **Global Session**: One instance per global HTTP session (portlet-based web applications).

Setting Bean Properties:

```
<bean id="Myvegetable" class="org.my.maven.project.Vegetable">
        <property name="myVeg" value="Brocalli"/>
</bean>
```

To set bean properties via context, the property name="myVeg" should have the same name as that of the member variable of that class, witha setter adn getter methods.
Another way to set properties: P-name:"value" in <bean P-name:"value"/>

what is Auto Wiring in Spring?
-> Spring attempts automatically figure out like how beans are connected.
-> In Normal if we want to connect beans user has to take to connect the bean based on the requirements but using Auto-wiring spring will automatically try to connect
the beans by reducing the manual configuration.

Types of Auto Wiring
->By Type: It will work when only one bean is present of that type.
->By Name:
->By Constructor:
->No
->default

Understanding Tight Coupling:

- ➔ If any class depends directly using its object, then we call it as tightly coupling.
- ➔ **How to achieve tight Coupling**: By making use of Interfaces.

Note: 2 things to understand here is Dependency Injection and Loose Coupling.

Using Spring framework to manage Dependencies:

To manage spring on behalf of us we must tell spring 3 thing.

1. What are the beans?
   - ➔ To tell spring this is like a bean, we must use annotation called **@Component**
2. What are the dependencies of a bean?
   - ➔ Dependencies can be set by using the Annotation **@Autowired**
3. Where to search for beans?

➔ Spring boot scans the current package and its sub packages as long we are in same package no need to set explicitly.

Note: All the beans we cerate will be in **Spring Application Context**

@Component: In 3rd step, spring boot scans for all the packages and sub packages with for annotations like @Component and others and creates beans for the classes, while doing so if spring finds any @Autowired annotation then Spring tries to auto wire them.

Auto wiring in Depth: when we have multiple candidates for Autowiring**.**

Autowiring can be done 3 ways by **using @Primary, @Qualifier** Autowiring by name (`private SortAlgorithm bubbleSortAlgorithm;).`

Usage:

a. @Primary
b. @Qualifier("Quick")

Scope of a Bean:

a. **Singleton**: When we request Application Context for bean, we always get same bean, because in spring by default its singleton.

b. **Prototype:** By default, its Singleton so make it Prototype we have done like below.
   `@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)`

c. **Request:** one bean is created by one HTTP request and response
d. **Session:** One bean created for one session

Complex Scope Scenarios:

What will happen if Class is of singleton and its dependent class is of prototype.

```
@Component
@Scope(value=ConfigurableBeanFactory.SCOPE_PROTOTYPE,           proxyMode           =
ScopedProxyMode.TARGET_CLASS)
public class JDBCConnection {

	public JDBCConnection () {
		System.out.println("JDBCConnection. ");
	}
}
```

We can resolve the same by suing the proxy.

Scope based on GOF and Spring:
Singleton in GOF: Will create only object for entire JVM
Singleton in Spring: Will create separate object for each Application Context.

@ComponentScan:
Can be used when we want scan packages which are not the part of current package or it sub packages, then we need to explicitly mention the `@ComponentScan("com.in28minutes.spring.basics.componentscan")` annotation to scan the package.

Bean Life Cycle methods:
When we use @Component, then the entire life cycle of this bean is managed by Spring IOC (Inversion of Control).

Let's say if want to manage some jobs before and after bean creation?
@PostCreation
@PreDestroy

CDI: Context and Dependency Injection

@Inject=@Autowired
@Named=@Component & @Qualifier
@Singleton (Define scope of singleton).

Group id and Artifact id: **javax.inject**

Read values from External properties file:
```
@PropertySource("classpath:app.properties")
```

```
@Value("${external.service.url}")
      private String URL;
```

Unit and Junit testing:
Unit testing: Writing test case for specific method.

Junit: Writing test case for specific method ☹These are automated testcases. Its falls under continuous integration

@Mock is used to mock the resources in the test class.
@InjectMocks is used to inject dependencies
@RunWith(MockitoJUnitRunner.class): Will helps in configuring the above annotations before running the test cases.

Spring Boot Full details and why spring boot is required.

What are the goals, What's Spring Boot is not and what are the features?

Today world moving toward Micro Services Instead of developing large monolithic applications. We are building a lot of micro services. So instead of building one big application, we are building 20-25 maybe 50 smaller micro services.
One of the important things with these micro services is you would want to be able to build them quickly. -> *That's where Spring Boot comes in*

How did Spring boot achieve this and What are the most important goals of Spring Boot?

The most important goal of Spring Boot is to *enable building production ready applications quickly.* The other important goal is to provide all the common non-functional features like -> embedded *servers, metrics health checks and externalized configuration.*

These are the two important goals of Spring Boot –

A) To enable you to quickly build applications and
B) Provide common non-functional features.

Before we go more in depth into Spring Boot you also need to understand what Spring Boot is not.

a) What Spring Boot is not is there is no code generation at all. Some of the people call Spring Boot as a code generation framework. Spring Boot does zero code generation and that is what makes this concept great.
b) The second thing, Spring Boot is neither an application not a web. Spring Boot provides great integration with embedded servers like Tomcat, Jetty but by itself Spring Boot is not a web server or It is not an application server.

These are two things that you would need to remember.
There is no code generation in Spring Boot and Spring Boot is neither an application server nor a web server.

Now that we understood the goals and understood what Spring Boot is not.
Let us look at how Spring Boot achieved these things.

The most important part of Spring Boot is this concept called Starter projects.
Consider example of developing a web application.
If let us say I want to develop a web application
I would need Spring MVC.
I would need Spring core.
I would need some validation framework.
I would need some logging framework. In addition to that,
I would need to configure all this stuff that is needed.

For example, if I'm using Spring MVC,
I would need to configure dispatcher servlet.
I would need to configure view resolveser and a lot of things like that.
However, with ***Spring Boot starter project*** it becomes very easy. All that you need to do is to *add a starter called Spring boot starter web* into your project and that's it.
You'll get to Spring MVC for free.
You get Spring core for free.
You get a validation framework for free and
logging framework for free.

Similarly, for JPA there is a starter called **Spring Boot starter JPA.**
Once we use this starter. You would not only get JPA but also a default implementation of *JPA with Hibernate and auto configuration of that*.
So you would not need to worry about the framework part and you can directly start creating your entities.

Another important feature we already talked about is embedded servers. Let's say I am developing a web application I would want to deploy it on to a Linux box.

In the olden days the way it used to work is first I would need to install the Linux box. Then I would install java on it and then I would need to install a web server. So I would need to install either Tomcat, WebLogic or WebSphere and then I would take my application war and deploy it.
This is the usual way we use to deploy stuff. With Spring Boot, comes a concept called embedded server. What you can do is you can package your server. So, you can package Tomcat along with your application jar. So, I can include the Tomcat server in my application jar. So, I don't need to install it on the Linux box and all that I need to do, on the Linux box is if I have Java installed that's enough.
I can go ahead and run my application. I don't need any other server installed on the Linux box. In the world of Micro services, this makes a huge difference.

Spring Boot provides several production ready features.

1. Spring Boot provides monitoring for your application through something called Spring Boot actuator.

   For example, you can find out how many times a service is called. You can find out how many times a service failed, and you can check whether the application is up and running or not. All these features come built in.

2. Another important feature that Spring Boot provide is externalized configuration. The configuration of applications varies between different environments. Your configuration from dev different from your configuration in production. Spring Boot provides these features built in.

You can simply create property files matching a simple naming convention and that's it. You're ready with externalized configuration. Spring Boot also provides support for different profiles.

How spring projects used to develop before the Spring boot:

One of the best things to do to understand the beauty of Spring Boot is to understand how things were done before Spring Boot. In this step, we'll look at one

of the projects which we developed in our Spring MVC course. Before we start with any project, we need to decide what frameworks and dependencies to use.
This was a web project, so we would want to use Spring MVC and things like that.
So, we needed to make decisions on what frameworks and what version of them to use.
We needed to decide ->
I would want to use Spring MVC and Spring security web, Spring security config.
I would want to use Jackson data bind because I want to do some binding and validation.
I would want to use JSTL,
I would want to use Hibernate validator and for logging I would want to add in log4j.
We had to decide what dependencies to add into our projects.
Not only that, we needed to decide the versions of them as well.

Sometimes this 5.0.2. FINAL version might not be compatible with some other version of Spring MVC. So, you needed to decide what are the comparable versions and start using them. Choosing the frameworks to use and which version to use is a major decision that we needed to make when we were not using Spring Boot. We needed to implement default exception handling. You can see that here; we are just implementing simple default exception handling and we needed to create a complete Spring configuration file.
We needed to define the component scan and then we would need to configure a view resolver to redirect the views to a JSP. To implement internationalization, we needed to implement a message source and a locale resolver and in addition to that we needed to configure our web.xml as well. We needed to configure the dispatcher servlet in web.xml so that it can handle all the requests and act as a front controller.
We needed to configure the context configuration location in here and we also needed to configure the Spring security. We needed to configure the filter for it and make sure that it intercepts all the request. There is a lot of work that you would need to do to get a simple web application up and running. We configured dependencies, dependency version, Spring configuration, configuration for internationalization, logging and a lot of other stuff. All these stuffs which we looked at in this specific step is the kind of stuff which you don't need to do with Spring Boot. Spring Boot would automatically provide all that stuff for you, so that you can really concentrate on developing your business logic.

All the infrastructure would come free for you. The idea behind this step was to give you an idea of the world before Spring Boot. If we directly jump into spring boot, You, might not really appreciate the value that Spring Boot provides. All the stuff which we looked at in this specific example would be replaced by a simple starter project called Spring Boot startup web. In combination with Spring Boot starter security. Just the combination of these two starters would eliminate the need for a lot of configuration that.

What is Auto Configuration?

@SpringBootApplication ->
@EnableAutoConfiguration, @Configuration, @ComponentScan

In the previous step, we were able to create a REST service very easily. In the step, we will understand what auto configuration is?

How did all the things that are needed for the REST service to be up and running get configured automatically.

we have the **@SpringBootApplication annotation**. The annotation **@SpringBootApplication** indicates that this is a Spring context file, That's number one.

Number two, it enables something called **auto configuration**.

Number three, it enables something called **component scan**.

@Component scan is one of the important features of Spring where it would start automatically scanning these classes in this package and its sub package for any beans.

For Example, if we have added in an annotation **@Restcontroller**. **@Component scan** is the annotations which is scanned for it So, this would be registered as a component. So, the book Controller would be registered as a bean and it would be managed by the Spring framework.

There are three things that are essentially done by **@SpringBootApplication.**

1. *It indicates that this is a Spring context.*
2. *The second thing is the fact that this enables auto configuration.*
3. *The third thing is that it enables automatic scan of this specific package.*

Now if we want to look at auto configuration in dept. **SpringApplication.run ()** method is used to run a Spring context.

So, I'm giving a Spring context as an input to it and it would be able to run that. The run method also returns something. So, it returns an application context. So, I'll get the application context back. So, I'm getting the application context back. This returns the application context.

I’m taking it and I would say application context. I would want to loop around it. So, for each, control one. Control space.

I would want to loop around the **Applicationcontext.getBeans().** I want to get the names of all the beans which are configured, and I would call this name and I would

want to log them. I’ll for now do a **sysout**.

I would go ahead and print all the beans which are present in here. Let's see what would happen. Run us Java application.

So, there are a host of things that are getting configured for us automatically. How are these being configured?

That's basically the feature called auto configuration. What Spring Boot does is as part of the Spring Boot framework, there is something called **Spring Boot autoconfigure**.

So, if I look at, one of the dependencies is Spring Boot auto configure. And this Spring Boot auto configure has a lot of logic built into it. So, you can see that there is a lot of classes that are written.

Inside Jar’s: So, if you look at the auto configuration Jar, it's a very big jar and if we go to the Web, We will find our configurations for all the things that we are looking at right now. I'm going into the auto configure web servlet and you would see something called dispatcher servlet auto configuration.

These auto configuration classes are what are creating these beans. What would happen at startup is the Spring Boot framework would trigger the auto configuration jar and it would loop through classes which are on the classpath. So, it would see that, OK, do I have a specific class on the classpath? It would see that I have a Spring web MVC framework on the classpath.

What does this Spring Boot auto configure do?

It says, “Oh! There is Spring web MVC on the classpath, then I would need to configure a dispatcher servlet.” It would say, “Oh! there is Spring MVC on the classpath, so I would need to configure a view resolver.” So, all those kinds of things happen, and you'd be able to see all the beans which were created in here. How does the configuration work?

Spring Boot basically looks at two things. So, Spring Boot looks at

A) the frameworks which are available on the classpath.

B) It looks at the existing beans which are configured for the application and based on that, it provides the configuration needed.

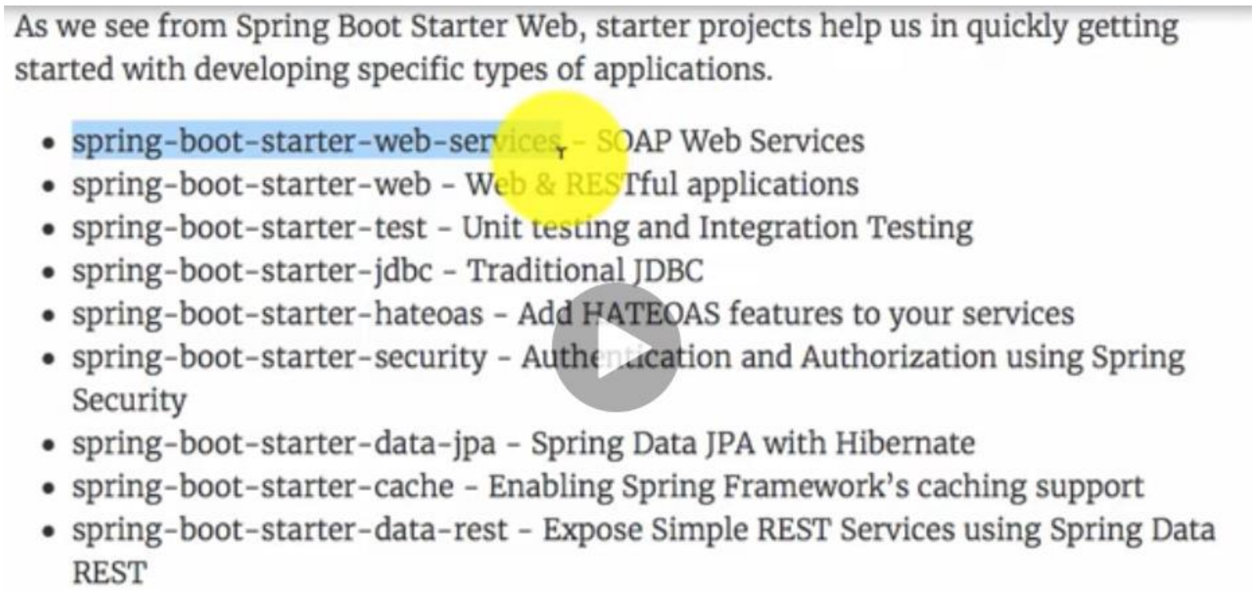
So, for example, if you are using a web starter, the basic things that you would need would be a dispatcher servlet, you would need a, probably, some internationalization, you need some messaging. All that kind of stuff would get autoconfigured for you. Now, we are looking at one of the auto configurations beans which is data source auto configuration.

If you see the code of data source auto configuration which is inside the Spring boot auto configure jar. You would see that it's conditional on class data source dot class and an embedded database type dot class to be present on the classpath. But so, what happens is only these classes are on the classpath, then this specific bean would be created. The data source bean would be only created when these are on the classpath.

But you can see the code in here. So, data base auto configuration is conditional on a specific class being available on the classpath. But you can also see another example in here, So embedded data source database configuration.

dispatcher servlet auto configuration has matched because I found something of this kind in the classpath. So, it’s saying, I found a servlet called dispatcher servlet on the classpath. So I've configured dispatcher servlet automatically. It’s saying I've configured an error MVC auto configuration. I've already configured default error controller. I've configured a cache, generic cache configuration. I've configured Jackson for doing binding and I’ve auto configured validation as well. You can also see that there are things like view resolver and everything that were auto configured Not only that, the auto configuration report also shows what are the things that were not auto configured So it would show why the rest of the auto configuration things did not match and why they were not executed.

Spring Boot Started Dependencies:


As we see from Spring Boot Starter Web, starter projects help us in quickly getting started with developing specific types of applications.

- spring-boot-starter-web-services - SOAP Web Services
- spring-boot-starter-web - Web & RESTful applications
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST

Actuators and their Need:

So, I'm copping the start of a verb and changing the starter name to after that it implies that I didn't start to fracture it. So, what does the actuator bring in. It brings in a lot of monitoring around your application So, in actuator you'd be able to read a lot of data about the application.

Let's say you

1. You want to see what are the beans that are configured.
2. You want to see how Auto configuration has worked.
3. You want to see how many times the specific services called.
4. you want to see how many times a specific service has failed.

All that kind of stuff you can check in your actuator and all that I need to do to enable that is in something called an **actuator**.

In springboard that the actuator exposes a lot of the rest services and these services are compliant with the standard called **HAL standard Browser**.

Some of the standard end points of actuators:

1. beans
2. health
3. Info
4. Loggers
5. Audit Events
6. Self
7. Env
8. HttpTrace

Etc..

To enable more endpoints in the HAL browser we have to update properties file with:

*management.endpoints.web.exposure.include=**

Spring Boot Dev Tools:

Will load only the classes and its components but not all the jars available in the class path Hence it is little faster than normal restart.

Spring AOP: Aspect Oriented programming

```
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
</dependency>
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aspects</artifactId>
</dependency>
```

AOP is best suitable for cross cutting concerns: Intercept the calls among the classes and do some functionality we use AOP.

What is command Line Runner?

command line runner: whatever we have defined in command line runner, would be invoked as soon as the application is launched by running the implemented method called run ().

Explicit autowirning will always override any auto wiring.

Common AspectJ annotations:

1. @**Before** – Run before the method execution
2. **@After** – Run after the method returned a result
3. **@AfterReturning** – Run after the method returned a result, intercept the returned result as well.
4. **@AfterThrowing** – Run after the method throws an exception
5. **@Around** – Run around the method execution, combine all three advices above.
6. **@Aspect** declares the class as aspect.
7. **@Pointcut** declares the pointcut expression.

Content Negotiation: Implementing support for XML

All our services that we created until now only work with the JSON input and the JSON output. So, we're using the JSON representation of the resource. If you want to add XML format to one of the representations that is supported,

how do we do that?

All that we need to do is to make a simple jar available in our pom dot xml. Let's do that right now. So, let's go to the pom dot xml. All the binding from JSON to objects and objects to JSON is happening through something called Jackson. Jackson does this binding. There's another simple dependency which we can add in here which can help us with handling the XML conversion as well.

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

Awesome! Now you're getting the data back in XML format as well.

Configuring Auto generation of Swagger Documentation:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

By adding the above 2 dependencies and below code we can generate Swagger Documentation for any application.

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2);
    }
}
```

http://localhost:8080/swagger-ui.html
http://localhost:8080/v2/api-docs

How we can alter the Document based on our requirements:

```
@Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(DEFAULT_API_INFO)
                .produces(DEFAULT_PRODUCES_AND_CONSUMES)
                .consumes(DEFAULT_PRODUCES_AND_CONSUMES);
```

Monitoring API's with Spring Boot Actuator:

Let's look at a few of these things that the actuator is exposing.

1. The **audit events** is related to security. Audit events would show which users were properly validated, how many people failed the authentication, and all that kind of stuff.
2. **Beans** is another interesting thing. This would show all the Spring Beans that are configured right. Spring is a dependency management framework and it manages all the beans it creates, all the components for you, and you can look at all the beans that Spring has created for you.
3. **Health** of the application. It's up.
4. **conditions** are another interesting thing. As we have discussed until now, Spring Boot is all about auto configuration, there is a lot of stuff that Spring Boot auto-configures. Conditions service exposes the positive matches which are all the conditions that matched as well as if you search for negative matches, the conditions which did not match. so, it would show a list of beans which are configured, which are not configured and all that kind of stuff.
5. **configuration properties**, the environment loggers which are configured.
6. Heap dump,
7. Thread dump
8. metrics as well. So, if you go to the metrics, I can see a list of metrics which are valid.
   - So, let's say, I would want to find out how much memory is used.
   - So, I can say, I can copy this, jvm.memory.used and add it in here.
   - So, jvm.memory.used and say Go!, you would see the value of the amount of memory that is being used.
   - CPU usage and all other kind of stuff. Now,
9. Scheduled tasks.
10. **http trace**. It shows all the requests that were executed, like requests and responses.
11. **Mappings** shows all the different things that are mapped to URIs, right. So, whenever we are creating a web application or a web service, we are mapping a lot of URLs. Whenever we are adding the dependency on actuator, we add a few URLs as well.

Implementing static Filtering for Rest Full services.

The idea of filtering is like, I'm sending a GET request to our users' resource and I'm getting this response back, right? *Id, name, birthdate. Id, name, birthdate.* Let's say, I don't want birthday to come in my response. How do I do that? ***That concept is called filtering.***

The filtering can be done in 2 ways

1. **Static**

a. we must use **Json Ignore** on the variables or fields.

```
private String field1;
private String field2;
@JsonIgnore
private String field3;
```

b. we can use @JsonIgnoreProperties(value=”field1”);

```
@JsonIgnoreProperties(value = "field1")
public class SomeBean {
}
```

2. Dynamic Filtering

I would want to configure dynamic filtering; how do I do that? There is a class called **Mapping Jackson value**. We can create an instance of the mapping value class.

And if you look at the code for the mapping value class, you'd see that there is something called filters that you can configure.

```
@GetMapping("/filtering")
public MappingJacksonValue retrieveSomeBean() {
       SomeBean someBean = new SomeBean("value1","value2","value3");

       MappingJacksonValue mapping=new MappingJacksonValue(someBean);
             SimpleBeanPropertyFilter
filter=SimpleBeanPropertyFilter.filterOutAllExcept("field2");
      FilterProvider filters=new
SimpleFilterProvider().addFilter("SomeBeanFiltering", filter);
      mapping.setFilters(filters);


       return mapping;
}
```

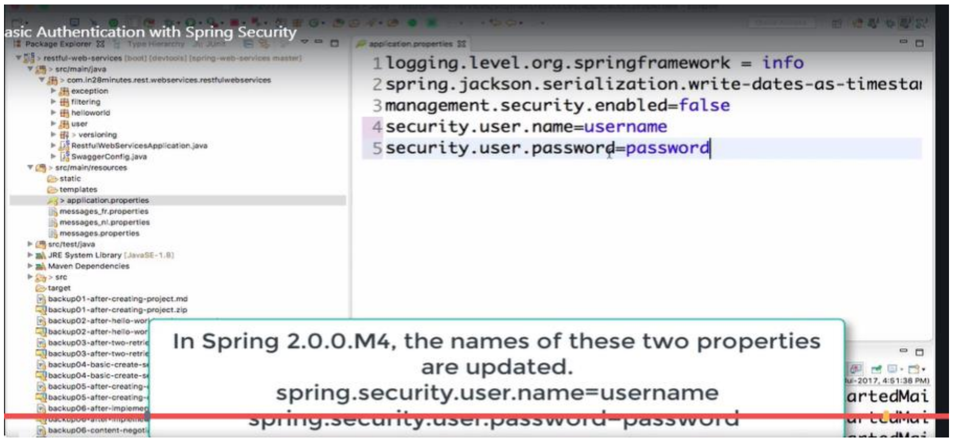
Versioning the RESTFUL Services with basic Approach with URI

One way is do manually by creating 2 versions of the same based on the requirements.

Versioning the RESTFUL Services with Header and Content Negotiation Approach

Spring Security:

Implementing basic Authentication with Spring Security



```
logging.level.org.springframework = info
spring.jackson.serialization.write-dates-as-timesta
management.security.enabled=false
security.user.name=username
security.user.password=password
```

In Spring 2.0.0.M4, the names of these two properties are updated.
spring.security.user.name=username
spring.security.user.password=password

5 things associated security:

a. Authentication
b. Authorization
c. Principal
d. Granted Authority
e. Roles

Authentication:

Knowledge based authentication: Answering who are you to the application / Site is called Authentication like by providing user Id and Password and this type of authentication is called **Knowledge based authentication.**
Possession based Authentication: like text message will be sent to you to authenticate.
Multi Factor Authentication: User Id and Password + text Message.

Authorization: For authorization we need Authentication in most of the cases, to decide who can access what.

Principal: Currently logged in user in the context of application.

Granted Authority: Vs **Role: (Who can do what)**

How to add Spring security to a Spring Boot Application:

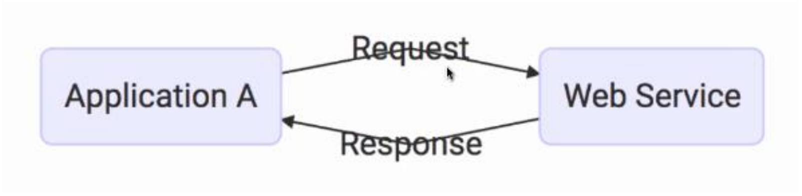
➔ **Its Default behavior:**

a. Adds mandatory authentication for URLs
b. Adds login form
c. Login error handling messages
d. Creates users and sets default password.

Mastering MS with Spring Boot and Spring Cloud

1. **What is Web Service?**
    - It says it's a software system designed to support interoperable machine to machine interaction over a network.
      i.e.
      a. interoperable: Not a platform dependent.
      b. machine to machine (Application to Application): Should be able to communicate with other applications.
      c. interaction over a network: Interaction should happen over a N/W.

2. **How does data exchange between applications take place?**



Here request and response need not to be in same format.

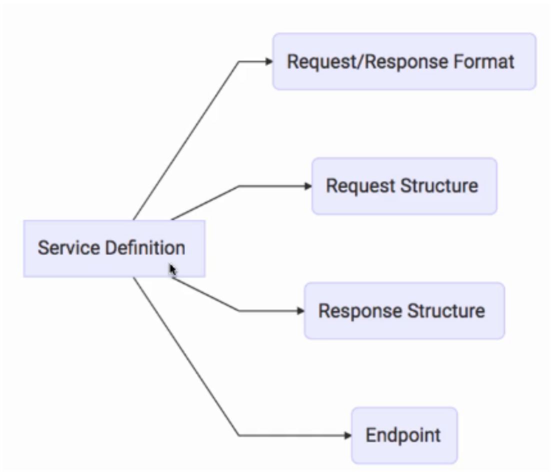
There are 2 formats for Request and Response in Web services.

a. XML

b. JSON : **JavaScript Object Notation**

How does it know what request to send, where to send it, and what is the format of the response?

- The solution to that is service definition.

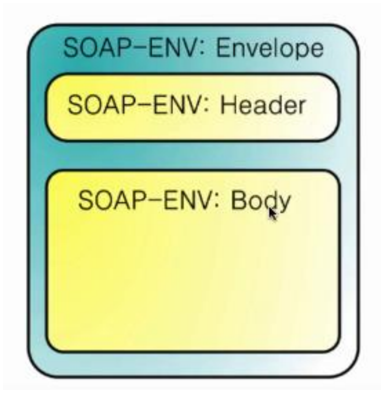


3. **Key terminologies in Web-Services:**
    a. service provider
    b. service consumer.
    c. service definition.
    d. what is transport.
    e. Request and response
    f. Message Exchange Format
    g. Transport
        a. HTTP and MQ

4. **Web services types / Groups / Kinds**
    a. SOAP: Simple Object Access Protocol
    b. REST: Representational State Transfer

a. SOAP : Simple Object Access Protocol.

SOAP is just a term. SOAP defines a specific way of building web services. In SOAP we use XML as the request exchange format.

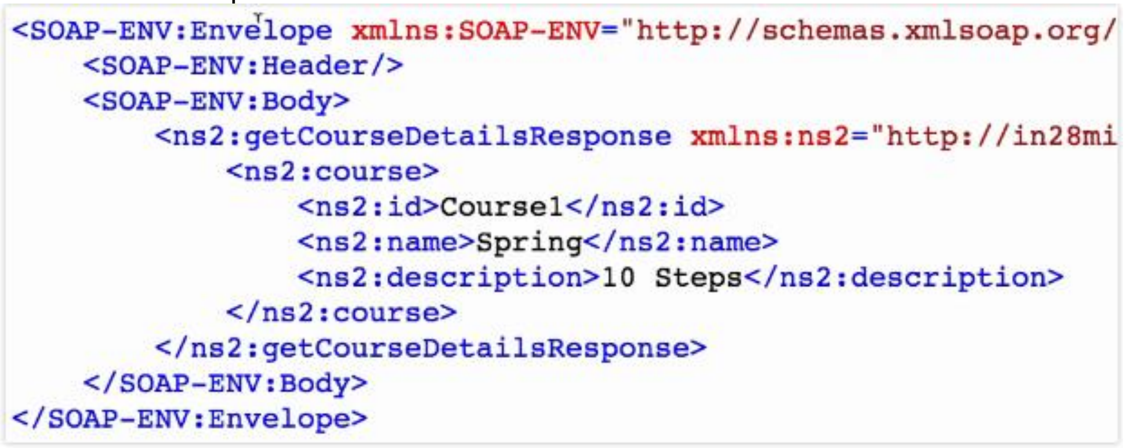


Note : SOAP Header is optional.

Important thing is SOAP defines a specific XML request and response structure.

If you're using SOAP, then you must use this structure. so, you have to create a SOAP and envelope which contains a SOAP header and a SOAP body.

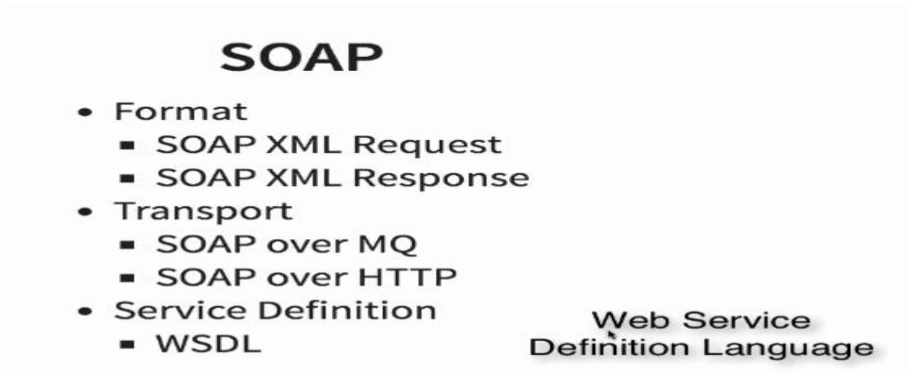
XML requests and a SOAP XML response and example response is shown in here. so, you have a SOAP envelope.



```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCourseDetailsResponse xmlns:ns2="http://in28mi
            <ns2:course>
                <ns2:id>Course1</ns2:id>
                <ns2:name>Spring</ns2:name>
                <ns2:description>10 Steps</ns2:description>
            </ns2:course>
        </ns2:getCourseDetailsResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In summary, SOAP is all about adhering to the services XML structure. Adhering to the envelop header and the body.

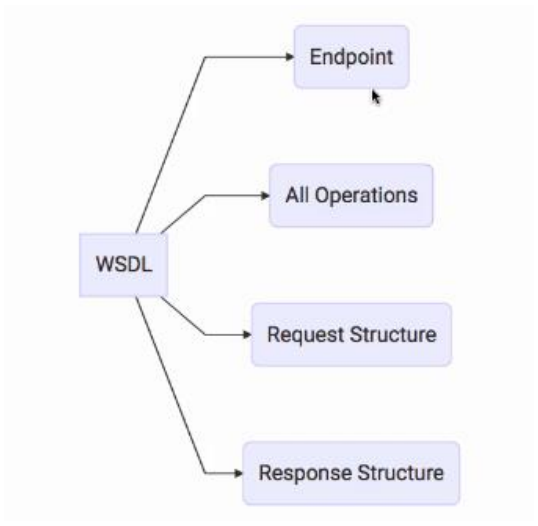
Once you have your request and response in that structure then you can say you are developing SOAP Web services.



SOAP

- Format
  - SOAP XML Request
  - SOAP XML Response
- Transport
  - SOAP over MQ
  - SOAP over HTTP
- Service Definition
  - WSDL

Web Service Definition Language

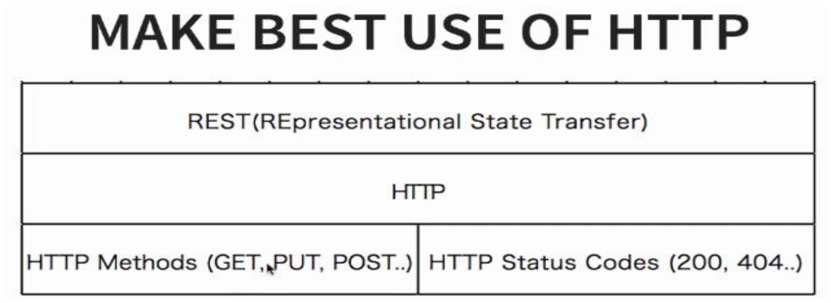


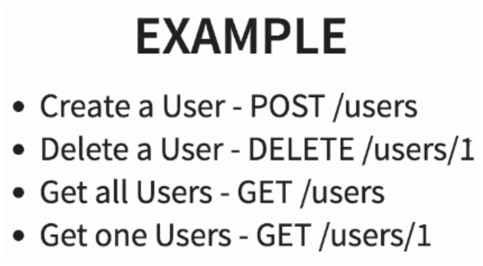
b. REST Web Services:

REST stands for **representational state transfer**.

It's a term which is Coined by **Roy Fielding (Also developed Http protocol)**.

Rest uses HTTP: Hypertext Transfer Protocol, to its best.


MAKE BEST USE OF HTTP
REST(REpresentational State Transfer)
HTTP
HTTP Methods (GET, PUT, POST..)
HTTP Status Codes (200, 404..)

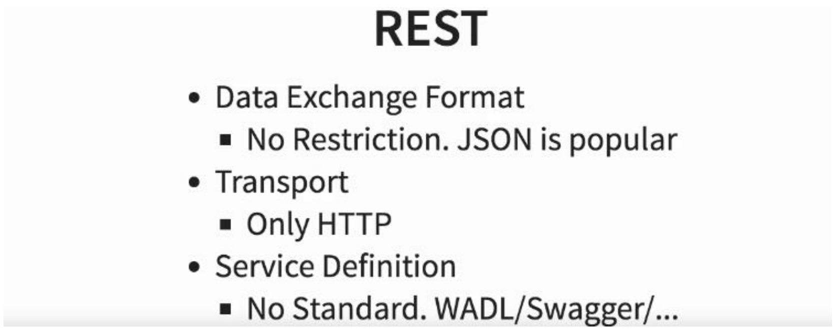


EXAMPLE

- Create a User - POST /users
- Delete a User - DELETE /users/1
- Get all Users - GET /users
- Get one Users - GET /users/1

POST :is used to **create**.
PUT :is used to **create or update**.

REST is completely built on HTTP



REST

- Data Exchange Format
  - No Restriction. JSON is popular
- Transport
  - Only HTTP
- Service Definition
  - No Standard. WADL/Swagger/...

REST focuses on your resources and how do you perform actions on them making best use of HTTP.

5. **Differences between REST and SOAP**

Let's talk about the data exchange format.

In SOAP the data exchange format is always XML that too specifically the SOAP XML with SOAP envelope header and body. Both your request and response should adhere to the SOAP Structure.

In REST there is no strict date exchange format. You can exchange XML, a JSON or any other format you would want use. However, JSON is the most popular data exchange format in REST. As far as the service definitions are concerned SOAP uses WSDL. Web service definition language.

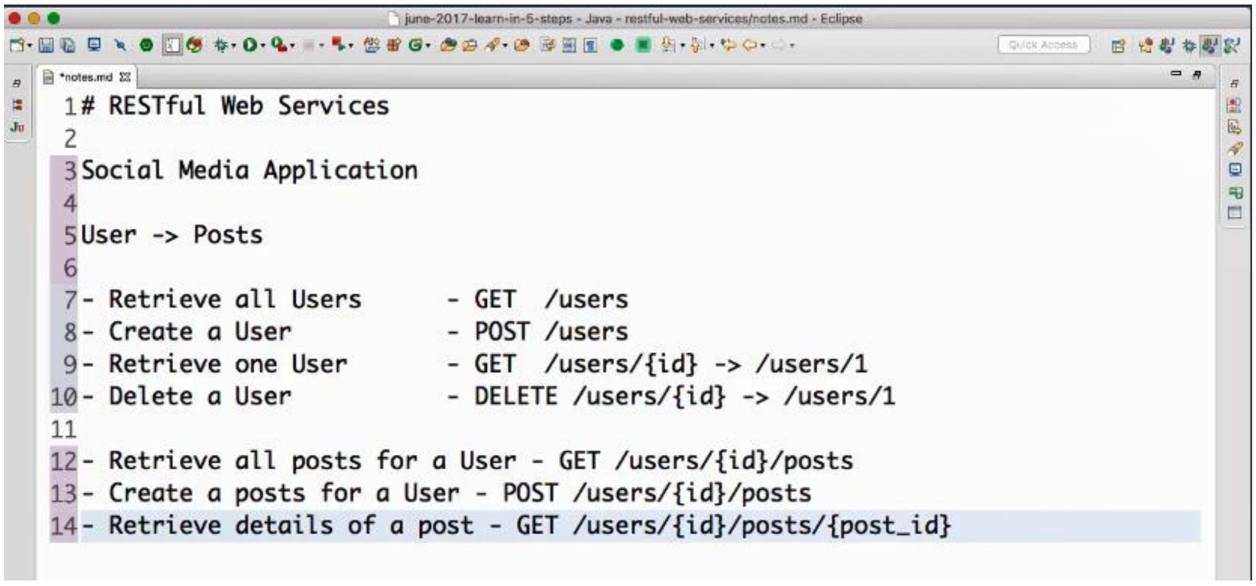
REST does not have a standard definition language. While WADL is one of the standards. Web application definition language. However, it’s not popularly Used. In this course, we are going to use one of the popular definitions for RESTful web services - Swagger.

Service Definition:

As far as the transport protocol is concerned SOAP does not pose any restrictions at all You can use web that is HTTP, or you can use MQ. REST is very specific about making the best use of HTTP protocol. RESTful services are typically easier to implement than SOAP.

RESTful services are typically based on JSON which an easy format is to pass and do things with it and with RESTful services. we don't really need to mandate really define a service definition. But with SOAP you must define WSDL and there are a lot of complexities associated with parsing your XMLs as well.

6. **Under Standing the Restful Web Services.**



```
# RESTful Web Services

Social Media Application

User -> Posts

- Retrieve all Users      - GET  /users
- Create a User           - POST /users
- Retrieve one User       - GET  /users/{id} -> /users/1
- Delete a User           - DELETE /users/{id} -> /users/1

- Retrieve all posts for a User - GET /users/{id}/posts
- Create a posts for a User - POST /users/{id}/posts
- Retrieve details of a post - GET /users/{id}/posts/{post_id}
```

7. **Annotations and its explanations:**

   a. @RestController :
      To tell the Spring MVC that this class is going to handle REST requests, or it would be controller, then we need to use annotation **@RestController.**

   b. @RequestMapping()
   c. @GetMapping()

      @RequestMapping(method=RequestMethod.GET, path="/helloworld"):
      - ♦ @ request mapping is used to tell spring the method (Get, Put, Delete, POST) we using to send the Request to server.

      @GetMapping(path="/helloworld")
      @PostMapping(path="/helloworld")

      @PutMapping() or @Patch() to update only particularparameter
      - ♦ It is shortcut method for @RequestMapping to tell spring.

we want to use is GET and the URI but, how do we map the URI? That's by using the path. Path is equal to slash hello world.

Note:

If You see that an unexpected error, **no converter found for return value of type Hello world bean**. This is one of the important things that you need to understand. It is not really an easy error to debug. And that's the reason why I illustrated this first step. so, this hello world bean does not have a getter. So, if we don't create a getter then the automatic conversion will not work. So, let's go out and create the getter.

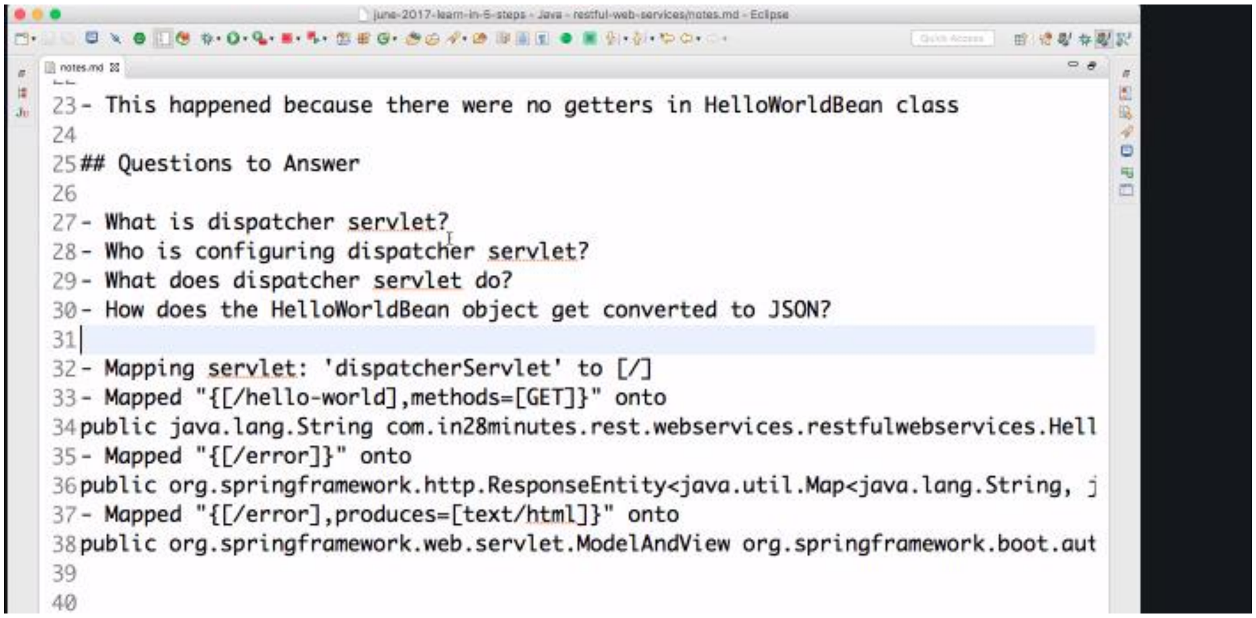
```
        public String getMessage() {
                return Message;
        }

        public void setMessage(String message) {
                Message = message;
        }

}
```

White label Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jun 05 12:16:41 IST 2019
There was an unexpected error (type=Internal Server Error, status=500).
No converter found for return value of type: class
com.rest.webservices.restfulwebservices.HelloWorldBean

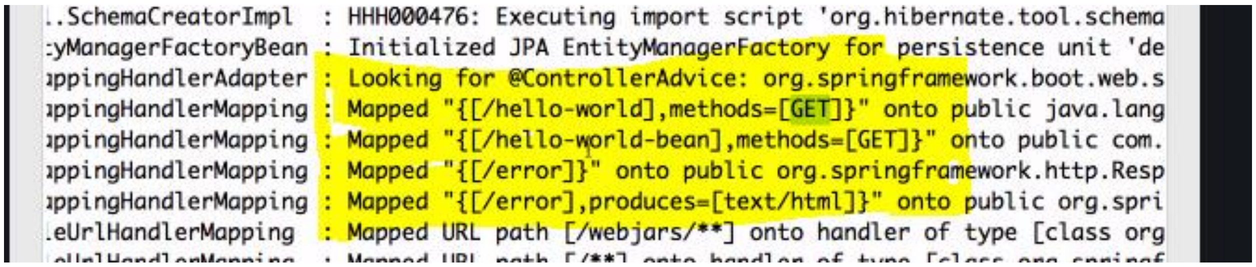
8. **Step 05 - Quick Review of Spring Boot Auto configuration and Dispatcher Servlet**



```
- This happened because there were no getters in HelloWorldBean class

## Questions to Answer

- What is dispatcher servlet?
- Who is configuring dispatcher servlet?
- What does dispatcher servlet do?
- How does the HelloWorldBean object get converted to JSON?

- Mapping servlet: 'dispatcherServlet' to [/]
- Mapped "{[/hello-world],methods=[GET]}" onto
public java.lang.String com.in28minutes.rest.webservices.restfulwebservices.Hell
- Mapped "{[/error]}" onto
public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, j
- Mapped "{[/error],produces=[text/html]}" onto
public org.springframework.web.servlet.ModelAndView org.springframework.boot.aut
```

what's happening in the background. You'd see the mappings that we have created. so we created a mapping from Hello world get, Hello World bean get but there are additional mappings also present in here for example, /error.

When we look at the log:



```
..SchemaCreatorImpl   : HHH000476: Executing import script 'org.hibernate.tool.schema
:yManagerFactoryBean  : Initialized JPA EntityManagerFactory for persistence unit 'de
appingHandlerAdapter  : Looking for @ControllerAdvice: org.springframework.boot.web.s
appingHandlerMapping  : Mapped "{[/hello-world],methods=[GET]}" onto public java.lang
appingHandlerMapping  : Mapped "{[/hello-world-bean],methods=[GET]}" onto public com.
appingHandlerMapping  : Mapped "{[/error]}" onto public org.springframework.http.Resp
appingHandlerMapping  : Mapped "{[/error],produces=[text/html]}" onto public org.spri
.eUrlHandlerMapping   : Mapped URL path [/webjars/**] onto handler of type [class org
```

All the other configurations are happening just because of spring boot auto configuration, which configures all the things which are available in the class path.

For Example, some of the below:

1. **What is dispatcher servlet?**

   We have not been heard anything about dispatcher servlet so who is configuring dispatcher servlet?

   Spring boot looks at all the class path and based on the jars which are available on the classpath, it tries to auto configure different things like:

   1. Dispatcher servlet: Is configured by **SpringBootAutoConfiguration**

      What does Dispatcher servlet do?

      Whenever we give the URL, the dispatcher servlet will take that, and the request would go to Dispatcher servlet. Dispatcher servlet would be the one which would be handling that request first.

      This is following a pattern called **front controller**. Dispatcher servlet is the front controller for spring web MVC framework. So, the request goes to dispatcher servlet. Dispatcher servlet knows all the different mappings which are present in the application. So, it knows hello world get method is mapped to this method. Dispatcher servlet knows that hello world bean get request is mapped to this specific method.

2. **HTTP message converters:**

   Jackson to object mappers: Are responsible for converting the Bean in to JSON (JavaScript Object Notation).

3. **How does the hello world bean object get converted to JSON?**

   - Jackson to object mappers: Are responsible for converting the Bean in to JSON (JavaScript Object Notation).

4. **who is configuring the error mapping?**

   - SpringBootAutoConfiguration : It creates a Default error page for us.

So, there are a lot of questions that you'd have based on what we have done in the previous two steps and whatever was in the log. In this step, we will slowly look at all these questions.

I would want to start running this application in something called debug mode and the way:

```
logging.level.org.springframework = debug;
```

So, we had the bean automatically converted to JSON, right? So, these HTTP message converters were responsible for doing that. So, what is happening is something called Jackson to object mapper. So, this does the conversion from JSON to beans and beans to JSON.

Who is configuring dispatcher servlet?

- Spring boot auto configuration. How does the hello world bean get converted to JSON? It’s also because of Spring Boot auto configuration because the message convertors and the Jackson beans are getting initialized. Who's configuring the error mapping? Again, spring boot auto configuration. It creates a default error page for us. Now the last thing we would want to understand is the dispatcher servlet.

If you look at the log you would see something like this - mapping servlet dispatches servlet to slash. what is happening here is dispatcher servlet is handling all the requests. So, this is the root of the web application, right? So, it’s handling all the requests. Whenever you type a URL into the browser, whenever I type hello world bean into the browser, what would happen is the request would go to Dispatcher servlet. Dispatcher servlet would be the one which would be handling that request first. This is following a pattern called **front controller**. Dispatcher servlet is the front controller for spring web MVC framework.

So, the request goes to dispatcher servlet. Dispatcher servlet knows all the different mappings which are present in the application. So, it knows hello world get method is mapped to this method. Dispatcher servlet knows that hello world bean get request is mapped to this specific method. So, once it gets to request, it determines which is the right controller to execute that request. So, it looks at the URI and the request method. When we type a URL in the browser, we’re sending a get request. The dispatcher servlet says, “Ok! there's a get request to Hello World bean. Which is the right controller that would be able to execute it for me?” It will find this specific method. It would see that hello world controller dot hello world bean is the right one to execute. So, it would execute that, so it would make sure that this method is executed, and this bean is returned. Once the bean is returned then dispatches servlet looks at how do I send the response back. I have bean, now how do I send the message back?

We have a **@RestController** annotation in here. Part of the **@RestController** annotation, if you look at it is something called response body:

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {

        /**
         * The value may indicate a suggestion for a logical component name,
         * to be turned into a Spring bean in case of an autodetected component.
         * @return the suggested component name, if any
         * @since 4.0.1
         */
        String value() default "";
}
```

annotation. One of the important annotations which is present in **@RestController** is response body. What would happen when I put response body on a controller, is the response from that would be mapped by a message convertor into some other format. So here the message convertor which is going to be used is **Jackson**. So, dispatcher servlet would say, “OK! do the Jackson conversion. Do the conversion to JSON.” It would convert Hello World bean; whatever content is there in there.

Process of converting request it into JSON and sends the response back.

---> The hello world bean request goes to a dispatcher servlet **--->** dispatcher servlet finds the hello world controller, the specific method hello world bean **--->** It calls it, gets the bean, invoke the conversion on it **--->** converts it into JSON and returns the response back.

Whether you're working with web applications or with spring rest services, dispatcher servlet plays a key role. We will further discuss about dispatcher servlet when we talk about other things like security. In this step, we tried to take a quick look at the world behind this hello world controller. We looked at dispatcher servlet and how it controls the entire flow.

9. **Step 06 - Enhancing the Hello World Service with a Path Variable**

```
@GetMapping(path="/hello-World-Bean/path-variable/{name}")
        public HelloWorldBean helloWorldvariable(@PathVariable String name)
        {
                return new HelloWorldBean(String.format("Hello World, %s", name));
        }
```

@pathVariable:

So whatever value we are passing in URIs, that value is being picked up by the controller and it is being sent back in the response.

Path variables are a critical part of creating REST resources.

//hello-World-Bean/path-variable/{name}

E.g. //hello-World-Bean/path-variable/Sreeni

//hello-World-Bean/path-variable/Ram

- @PathVariable is to obtain some placeholder from the URI (Spring call it an URI Template) — see Spring Reference Chapter 16.3.2.2 URI Template Patterns
- @RequestParam is to obtain an parameter from the URI as well — see Spring Reference Chapter 16.3.3.3 Binding request parameters to method. parameters with @RequestParam

If URL http://localhost:8080/MyApp/user/1234/invoices?date=12-05-2013 gets the invoices for user 1234 on December 5th, 2013, the controller method would look like:

```
@RequestMapping(value="/user/{userId}/invoices", method = RequestMethod.GET)
public List<Invoice> listUsersInvoices(
        @PathVariable("userId") int user,
        @RequestParam(value = "date", required = false) Date dateOrNull) {
  ...
}
```

10. **Implementing Exception Handling - 404 Resource Not Found**

To get the response or status from browser Details About
@ResponseStatus(HTTP.NOT_FOUND);
Error Status:

- 404 File Not Found
- 500 Internal Server Error

11. **Implementing Generic Exception Handling for all Resources**

To Return the exception in some specific format, and provide a customized exception handling, we can extend below
ResponseEntityExceptionHandler : A convenient base class for
{
    @link ControllerAdvice @ControllerAdvice
}
Classes that wish to provide centralized exception handling across all
{
    @code @RequestMapping
}
methods through methods
{
    @code @ExceptionHandler
}.

It is typically used to define @ExceptionHandler, @InitBinder, and @ModelAttribute methods that apply to all @RequestMapping methods.

Exception handling Explanation by Ranga:
package com.rest.webservices.restfulwebservices.exception;
```
import java.util.Date;
import javax.xml.ws.http.HTTPException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@RestController
@ControllerAdvice
public class CustomizedResponseEntityExceptionHandler extends
ResponseEntityExceptionHandler{
        @ExceptionHandler(Exception.class)
```

```
	public final ResponseEntity<Object> handleExceptions(Exception ex, WebRequest
request) {
		ExceptionResponse exceptionresponse=new ExceptionResponse(new Date(),
ex.getMessage(), request.getDescription(false));
		return new ResponseEntity(exceptionresponse,
HttpStatus.INTERNAL_SERVER_ERROR);
	}

	@ExceptionHandler(UserNotFoundException.class)
	public final ResponseEntity<Object>
handleUserNotFoundExceptions(UserNotFoundException ex, WebRequest request) {
		ExceptionResponse exceptionresponse=new ExceptionResponse(new Date(),
ex.getMessage(), request.getDescription(false));
		return new ResponseEntity(exceptionresponse, HttpStatus.NOT_FOUND);
	}
}
```
In the previous step, we saw that when we executed the request to a non-existing user, we were getting a 404, not found but the specific structure which is defined by default by spring MVC. As we discussed earlier, in an organization you would to define a standard structure.
So let's say you have some standard structure designed in your organization.

And how do we adhere to that. How do we create responses in that specific structure. That's basically what we would look at in this specific step. Customizing the exception handling to a structure that is defined by us. Go ahead and create a simple structure.So I would want to have a new class and I'll call this exception. I can call this exception details, exception response or whatever I would want to. So I'll just call this exception response.Actually, I would want to create it in the package dot exception and move it to that package. In the exception response, we can create any custom structure. At the basic level, I mean at the most basic level the important things that you would need to have are - first would be a timestamp indicating where…when the exception happened then probably you would want to have some exception message and some detail. So let's create simple elements for them. Private date time stamp, private string message and I would create private string details. I'll import java dot util dot date. Ok! That's a very basic structure. And let's quickly add in the basic constructor. So right click source generate constructor using fields. That's cool! I have a constructor and I'll also generate getters and setter. I'll say select getters. Setters aren't really needed. I just create the getters. So what we have created is a simple exception response bean. We have a data string and string time stamped message and details. There's a simple constructor and a few getters also in there. And now what do we want to do, when an exception happens, I would want to return the exception in this format. So instead of the message we were getting earlier, timestamp status error message path. But I would want to create a message of this structure. Timestamp, message and details. At a later point in time, you can actually enhance this to meet the standards of your organization. So name of these variables might be different. So, instead of calling it a timestamp, maybe you'd call it something else. Instead of calling this message or details, you might be calling it something else. But you can easily customize it to whatever your needs are. The most important thing is this exception response should be something that should be standard across your organization. Not just your project, I would even think a level above and make it a standard across almost all services in your organization so that everybody can use this structure. The structure is the most important part and the structure has to be language independent because you are creating a java service.

You are creating a java resource. There might be other resources which are exposed from let's say Nodejs or dot Net or whatever it is. Once the structure is defined you can provide the Java implementation for it. What we want to do is whenever an exception happens we would want to return a response back in this specific format.

How can we do that? One of the important classes which is present in spring is response entity exception handler. This is basically an abstract class which can be extended to provide centralized exception handling across all the different exception handler methods. So this one I can use as the base class to provide some default exception handling for all our requests. What we'll do is we'll extend this class. So we'll extend this and provide a customized exception handling functionality. So let's create a new class. Again I'll go to the package exception and create a new class. I would want to call this customized response entity exception handler. Actually, you can put your organization name or whatever you'd want as the name here. That would make it even more specific. But for now I'll just keep it simple. I'd want to extend the default exception handling which is provided by spring in response entity exception handler. The other thing is this exception handling I would want it to apply to all controller. There might be a number of controllers that are defined, right? So there's a hello world controller, there is a user resource. One of the things you need to understand is we are calling this user resource because we're exposing it as a resource. Some projects, some people might even call this user controller. So instead of user resource, this might be to a user controller as well. So a controller or resource, I'm trying to be more specific to say this is a resource which is being exposed but you can still call it a user controller. Controller is something which we follow from the MVC Model View Controller pattern. So, this is exactly doing something similar. This is specifically exposing resources. So I call it user resource, you can also call it user controller. It doesn't really matter. The most important thing is to remain consistent. So, once you call this user resource, I'm going to call everything else as a resource as well. First thing we need to do is we need to call this a rest controller because this is providing a response back. In case of exceptions, that's what it does, at rest controller and other thing is I would want this to be applicable across all the other controllers. So, how can we do that? It can be done by using another spring annotation called Spring controller advice. I am importing controller advice here. So if you look at the controller advice, it says specialization of the component for classes that declare methods to be shared across multiple controller classes. So when we have multiple controller classes and we would want to share things across them, then we

would use this thing called controller advice. Using a controller advice, the typical things which are defined are exception handling. That's what we are defining right now. The other things which can be defined is how do you handle dates. That's defined by using init binder for example. And also the other things which can be defined are common modal attributes that you would want to use across controllers. In all these scenarios. we would use a controller advice. We have defined the controller advice annotation we have made it a rest controller. Now I would want to go ahead. So response entity exception handler. If you go further down here there is a specific syntax that you would need to provide to over ride things in here. So there's a default thing that is provided by response entity exception handler. We want to over ride that for specific exceptions. So what I'll do I'll copy this method from here. So I'll say public final response entity of object. Handle exception, exception comma web request. Now I want to over ride this for a specific exception. So, whenever some exception happens, I would want to say at exception handler. What kind of exception does this handle? I would want, for now, to handle all the exceptions. So exception dot class. I can import the exception handler in. And I'll rename the method. I'll call this all exceptions. Right now we are not returning anything back. What we want to do is when a exception happens we want to create a new instance of our specific bean. So what we want to create is…we created a bean called exceptional response. Right? So we want to return our specific thing back. We want to return a exception response back. And to create an exception response, first thing I'd need to have is the date. So I'll for now just say new date. I'll import java util date. Java util date, that's what I want. The next thing I would need to send is the message. So I'll get that from the exception. I'll say exception dot get message. I can … I need to add in a few more details. Any details that you would want to add in for that specific exception you can add. You have to be sensitive about security when we're sending any response back and in the detail you need to think about what you would want to send. For now, I would send something called request dot get description. And I'd include the client info as well. This has to be evaluated on case by case basis to see if you'd want to really include this description as part of your response. But for now let's just create this. I would actually take this into a variable. So exception response exception response is equal to new exception response. That's cool. It's perfect. Format this a little bit. Now we have the exception response. Now I would want to create a response entity object. So how do I create a new response entity object? Response entity object. I would pass in the exception response. That's one of the parameters that's input to it. And the next thing you would need to send in is the HTTP status. HTTP status dot…I'll give a status of internal server error for now. So for all exceptions, this is the status that is being returned back right now. We will quickly override this specific exceptions. That's cool, right? So now we have defined a default exception handling for all the things which would return an internal error back and we're creating an exception response object in our structure. Let's wait for the server to pick this change up. And now I would execute the request. So if you see that when I get ….execute the request for 500 it says, internal server error, time stamp is this, essage is this details is this. So we are getting the message back in our format but we're going back to the earlier thing of 500 internal server error. For all exceptions we would want to return internal server error but we would want to customize it for specific exceptions to return a different status back. For exception class, it's OK to return internal server error.

But let's say for user not found exception I can customize it. So I can say handle user not found exception. I'm just copying the earlier method, renaming a few things. Exception and this instead of exception I'll call this user not found exception. And now I can go ahead and customize the exception status that's returned. dot not found. And now when I execute the request, you would see that I'm getting a 404 not found and with a specific structure that we're using for our specific thing. At a high level, the most important things that you would want to understand is the fact that having a customized message structure across the organization is very important. You need to have some class of this kind trying to define what is the status that you would want to return back to for different kinds of exceptions. Probably you can create a simple jar with the exception responses that you'd want to give for the common ones. What we have looked at is different options that are present in spring MVC to customize how you return the exception object back. Actually you might not even need to create this entity exception handler if the default structure which is provided by spring MVC is good enough for you. But if you want to customize it then you can create a customized response entity exception handler where you can actually go ahead and customize how you would want your exception responses to look like and what kind of HTTP response status you'd want to return back. Until the next step, bye-bye.

12. **Step 14 - Implementing DELETE Method to delete a User Resource**

```
@DeleteMapping("/Users/{id}")
	public User DeleteUser(@PathVariable int id) {
		User user = service.delete(id);
		if(user==null) {
```

```
                throw new UserNotFoundException("id-"+ id);
            }
            return user;
        }
```

13. **Step 15 - Implementing Validations for RESTful Services**

So what I would need to do is add in something called at **@valid**. This is an annotation which is present in **JavaX validation API**. This is defined by the Java validation API. We already have the Java validation in API and its default implementation.

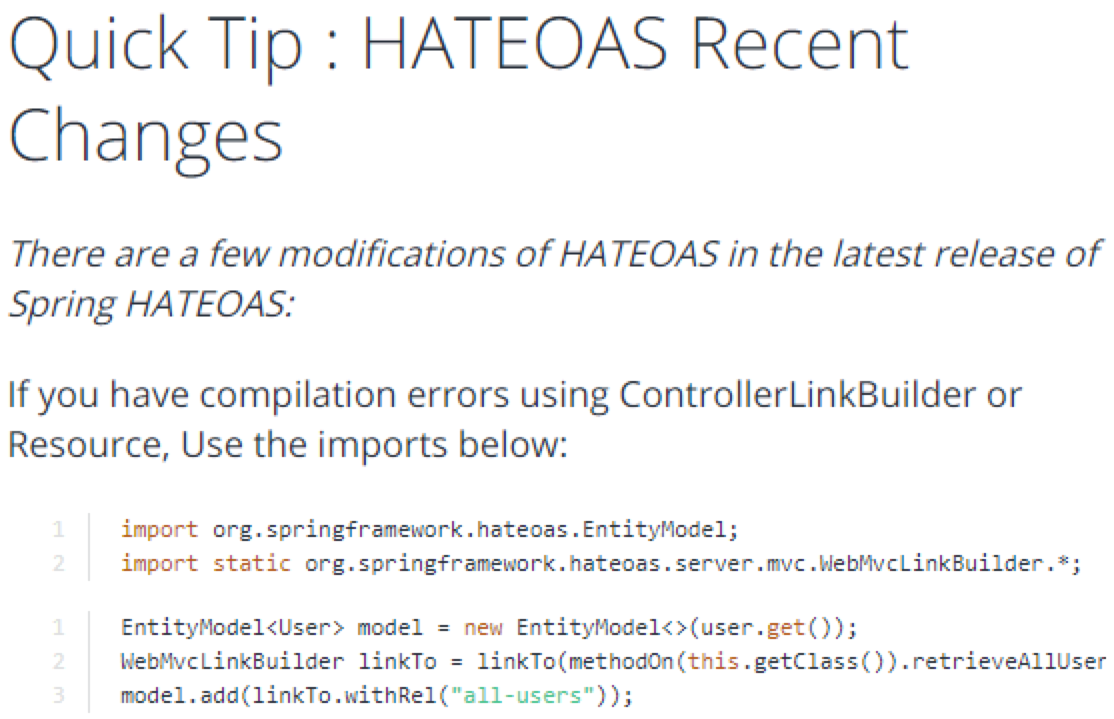
@valid: we just need to put this Annotation in the request body and wwe have to specify the conditions or validation in Pojo class.

@Size(Min=2) : to define size

@Past: Date should be past not future.

For more validation we have jars called : validation-api.jar and hibernate-validator.

14. **Quick Tip : HATEOAS Recent Changes**



Quick Tip : HATEOAS Recent Changes

There are a few modifications of HATEOAS in the latest release of Spring HATEOAS:

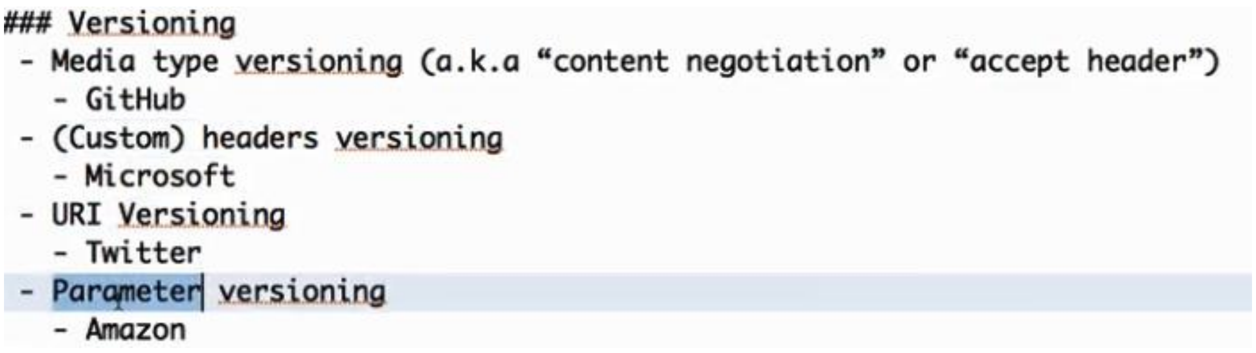
If you have compilation errors using ControllerLinkBuilder or Resource, Use the imports below:

```java
import org.springframework.hateoas.EntityModel;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;
```

```java
EntityModel<User> model = new EntityModel<>(user.get());
WebMvcLinkBuilder linkTo = linkTo(methodOn(this.getClass()).retrieveAllUser
model.add(linkTo.withRel("all-users"));
```

15. **Step 26 - Versioning RESTful Services - Basic Approach with URIs**

The versioning options available are :



```
### Versioning
 - Media type versioning (a.k.a “content negotiation” or “accept header”)
   - GitHub
 - (Custom) headers versioning
   - Microsoft
 - URI Versioning
   - Twitter
 - Parameter versioning
   - Amazon
```

The thing is, there are quite popular organizations where each one of these is used. Media type versioning is used in GitHub, custom headers versioning is used by Microsoft. URI versioning is used by Twitter and request paramter versioning is used by Amazon. The sage here is quite varied. Typically the factors that you’d need to consider when you talk about URI versioning are basically, think about the URI pollution. If you look at the URI versioning and the request paramter versioning, we are actually polluting the

URI space. So we are actually creating new URIs or creating new request parameters as part of the URIs. Whereas in these two we are actually not polluting the URI space at all. These two do, what is called misuse of HTTP headers.

HTTP header were never intended for versioning. And the other important factor is cashing. Because these two approaches use headers to differentiate
between the requests, the caching can never come into picture because I cannot cache requests because they have the same URI. I would also need to look at the headers and do complicated things like that. So caching becomes very difficult here. But in these two options URI versioning and request paramter

16. **Step 28 - Implementing Basic Authentication with Spring Security**

Spring boot Security can be achieved by using the starter :

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

After this the basic security will be turned , with password provided in the logs. **Or** we can set our own password and username like below in properties files.
Security.user.name=**user name**
Secutiry.user.password=**password**

17. **Step 29 - Overview of Connecting RESTful Service to JPA**

The other thing I would want to do is to go to **applications.properties** we would want to enable something called, which to control so that we can see that the table is created. we can see what data is inside the table. we would want to enable logging.

so, we would want to see all these queries that are being executed in the log where we can do that is: **spring.jpa.show-sql=true**

This would start logging the sql in log whenever a statement is executed and the other one is **spring.h2.console.enabled=true**.

After writing the above 2 in properties files we can see this in logs:

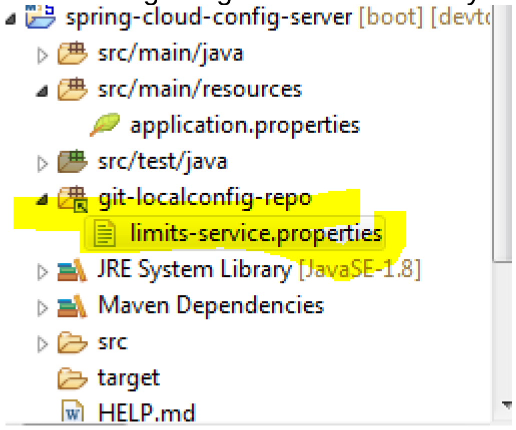
Hibernate: drop table user if exists

Hibernate: ***create table user (id integer generated by default as identity, birth_date timestamp, name varchar(255), primary key (id))***

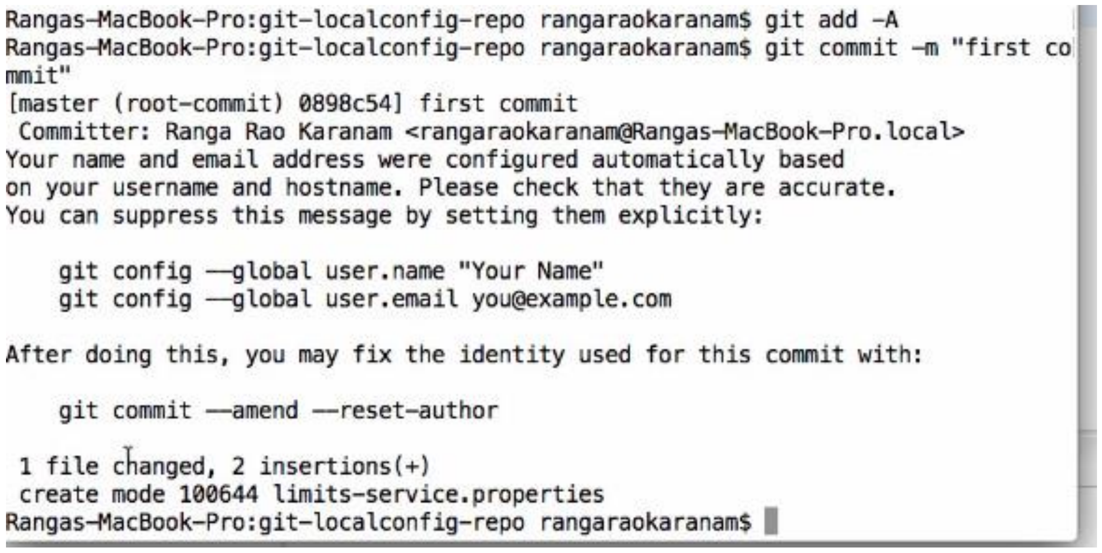
2019-06-14 17:45:14.738 INFO 9692 --- [restartedMain] org.hibernate.tool.hbm2ddl.SchemaExport : HHH000230: Schema export complete

Git Basics

1. **git init**: For initializing the git
2. After linking the git folder in to the by using the link source.



3. **Git add -A** : to add file name to local repository.
4. Git commit -m “first commit”



```
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git add -A
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git commit -m "first co
mmit"
[master (root-commit) 0898c54] first commit
 Committer: Ranga Rao Karanam <rangaraokaranam@Rangas-MacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 2 insertions(+)
 create mode 100644 limits-service.properties
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$
```

⇨ Every time we must restart the application or service after changing the Git or Configuration files to avoid that we can use spring cloud bus, it will make all the changes to reflect automatically reflected.

Micro Services Concepts

What are Micro services

A small autonomous service that work together.
Or
Anything that's consists of Rest, Small chosen deployable units and cloud enabled.

Advantages:

1. Dynamic scaleup and scale Down
2. Quick updates that helps in getting new features
3. New technology: We can use any technology

Disadvantages:

2. Configuration
3. Pack of cards
4. Bound lessness: With great knowledge we should know what the end of microservices is.
5. Dynamic scale up and scale down
6. Visibility: There is no Centralized or Monitoring is easily possible

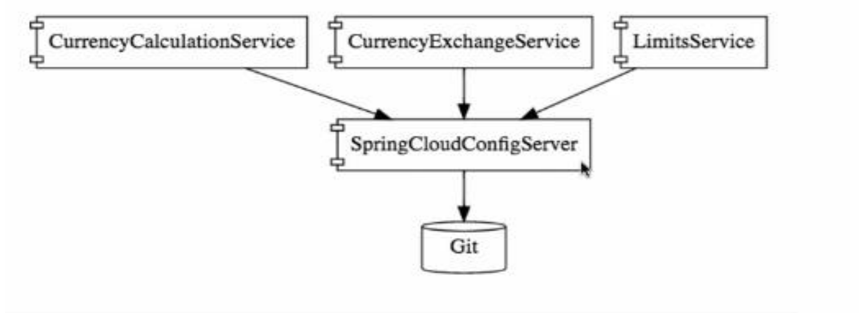
Spring Cloud:

We will be using **Finchley M2** version of Spring cloud would be using.
Under spring cloud, we will be having many frameworks as named below:

Below are the challenges we are facing using the micro services and we will see how we can overcome by using the spring cloud.

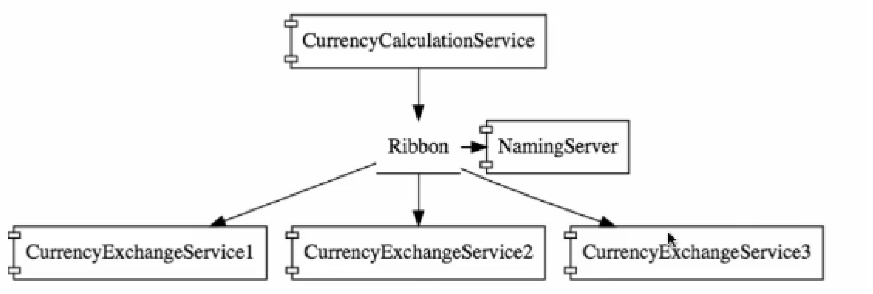
1. **Configuration:** The problem with the configuration is we will be having N number of services and managing configurations of all the services will be Nightmare, so we need a solution for this problem, here comes the service called **SpringCloudConfigServer**.

We can store all the configurations of all the micro services in one place by using **SpringCloudConfigServer** by using the centralized repository or Location called Git.



Spring Cloud Config Server

2. **Dynamic scale up and scale down:** This can be achieved by using the below microservices called :
    a. Naming Server (Eureka) : All Micro services should register with Eureka naming server and Service discovery.
    b. Ribbon : Client-Side Load Balancing, means it will distribute load among the exiting instances among the services we get from naming server.
    c. Feign (Easier REST Clients)



Ribbon Load Balancing

3. **Visibility and Monitoring:** ZipKin Distributed Tracing and Netflix API Gateway(For tracing requests from multiple services)
4. **Bound lessness:** With great knowledge we should know what the end of microservices is.
5. **Pack of cards: Hystrix :-** We will also implement fault tolerance using Hystrix if a service is down.

Annotations :
@Autowired : Marks a constructor, field, setter method or config method as to be auto-wired by Spring's dependency injection facilities.

@component : when we use @component mean that we are telling the spring to take care of the bean creation and its life cycle management.

Best ways to read the properties from the configuration files:
One of the best things about spring boot is the approach called *configuration properties.* Let's use that to read the values from the property file.

To achieve this, we have to use @Configurationproperties Annotations in the class with the prefix by service name.

Example:

```
@Component
@ConfigurationProperties("limits-service")
public class Configuration {

……….

}
```

So, when we are using the @ConfigurationProperties its must to use the Setter method along with the getter method.

How to make the spring cloud config server up and running: After creating the spring cloud config server, we must tell the spring that this is a cloud config server by defining the Annotation @@EnableConfigServer. As we read above using cloud config server, we can use different environments like Prod, L2, QA and UA.

How to fetch the properties from the Spring cloud config server: If you would want to pick up the configuration from a spring cloud config server, the property file must be renamed. It should not be **application.properties**, but I would start calling this **bootstrap.properties**.

What to use for what functionality?

Web : To create a rest service and web application.
We would want to create a service or a rest service so I will use web that's the one we have use to create a rest full web.

DevTools : Services I would want it to be able to automatically pick up changes', so we are using dev tools.

Actuator : I would also want to be able to monitor the application actuator.
config client : We would want to connect the application to a spring cloud config server.
So, we would need config client so config client.

Annotations :
@GetMapping: Annotation for mapping HTTP GET requests onto specific handler methods. Specifically, *@GetMapping* is a composed annotation that acts as a shortcut for *@RequestMapping(method = RequestMethod.GET).*

@PathVariable: Annotation which indicates that a method parameter should be bound to a URI template variable. Supported for RequestMapping annotated handler methods.

To get the current port value:

```
exchangeValue.setPort(Integer.parseInt(environment.getProperty("local.server.port")));
```

To fetch the values from the in-memory data base (H2), first we need to create a repository then we need to access the data in DB using the repository interface and I'll call this exchange value repository.

Entity name followed by the repository.

Thing is repository should be an interface so a public interface should extend what should it extend. If you have remembered the interaction JPA stuff it should extend JPA repository that to things that you have to pass in the first one is what is the type of the entity that it manages its exchange value and what it the type of the ID field let's organize the imports.

```
public interface ExchangeValueRepository extends JpaRepository<ExchangeValue, Long>{
        ExchangeValue findByFromAndTo(String from, String to);
                }
```

ExchangeValue : is the type of entity it is managing and **Long** is the type of ID.

What can be done using Repository ?

There are lot of methods that the repository provides. It will allow us to find different things by id, rows, find all and lot of find methods which are present in there.

But what we wanted to do is to search based on the **from and to** if you want to do such kind of things with spring data JPA we must provide something called the **query method.**

How do we search rows in the table by **from and to** we provide the **query method?**

Currency Exchange Service:

$$\begin{tikzcd} \text{CurrencyCalculationService} \arrow[r] & \text{CurrencyExchangeService} \arrow[r] \arrow[d, dash] & \text{LimitService} \arrow[d, dash] \\ & \text{DataBase} & \text{Configuration} \end{tikzcd}$$

Here we want **CurrencyCalculationService** to be calling the **CurrencyExchangeService,** to calculate the value.

How to retrieve which instance is responding to us?

```
        @Autowired
        private Environment environment;

        @GetMapping("/currency-exchange/from/{from}/to/{to}")
        public ExchangeValue retriveExchangeValue(@PathVariable String from,
@PathVariable String to) {
                ExchangeValue exchangeValue=new ExchangeValue(1000L, from, to,
BigDecimal.valueOf(65));
        exchangeValue.setPort(Integer.parseInt(environment.getProperty("local.server.port")));
        return exchangeValue;
```

Rest template

Now you would want to invoke the *currency exchange service* from the *currency calculation service* how do we do that ?

The thing which you typically use to invoke an external service which is exposed using entity *is the rest template.*

```
public CurrencyConversionBean convertCurrency(@PathVariable String from,
@PathVariable String to, @PathVariable BigDecimal quantity) {
            Map<String, String> uriVariables =new HashMap<>();
            uriVariables.put("from",from);
            uriVariables.put("to",to);
            ResponseEntity<CurrencyConversionBean> responseEntity = new
RestTemplate().
      getForEntity("http://localhost:8000/currency-
exchange/from/{from}/to/{to}",

            CurrencyConversionBean.class,

            uriVariables );

            CurrencyConversionBean responseBody = responseEntity.getBody();
            return new  CurrencyConversionBean(responseBody.getId(),from,to,
                        responseBody.getConversionMultiple(),
                        quantity,

      quantity.multiply(responseBody.getConversionMultiple()),
                        responseBody.getPort());
      }
```

Why Feign is used ?

Feign is a rest service client.

1. Feign makes it very easy to call other Micro services or restful services.
2. The other additional thing that feign put away it is it provides integration with something called Ribbon which is a client-side load balancing framework.

Main problem it solves is : Invokes other services.

Note: `<artifactId>spring-cloud-starter-openfeign</artifactId> : Open-Feign need to be used if version is greater > than 2.0.0`

How to enable Feign ?

➔ Need to add the annotation `@EnableFeignClients("com.java2cloud.springmicroservices.currencyconversionservice")`

➔ `Need to create a proxy`

Feign is rest service client, It makes us easy to call the rest full Web services. While using the rest template to call the service, It was really complex we have to write a lot of code. To avoid this all we need to do is to define a simple proxy. we defined a currency exchange to use proxy and we defined a simple method on it. Feign helps us to simplify the client code to talk to or ask for that service.

Why Ribbon is used ?

1. For load balancing among the services we use ribbon.

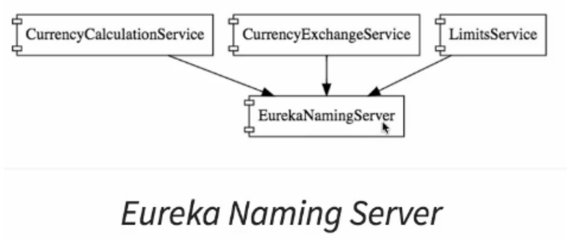
How to use ribbon ?

- ➔ We must use `@RibbonClient(name = "currency-exchange-service")` on proxy where while using Feign proxy instead of using default address like url = "localhost:8000")
- ➔ In the properties file we must configure the below:

```
currency-exchange-
service.ribbon.listOfServers=http://localhost:8000,http://localhost:800
1,http://localhost:8002
```

Ok Now load is distributing equally distributed among all the services irrespective of traffic, so how make use of ribbon only if it is required to distribute the load among services.

- ➔ This can be achieved by using the Eureka naming service :

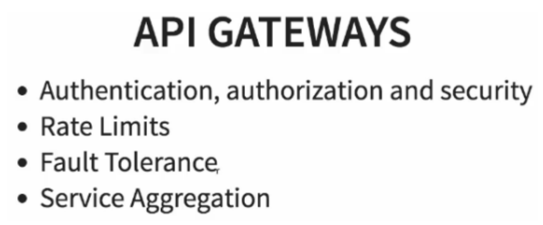


Whenever an instance of a micro service comes up it would register itself with Eureka naming service this is called Service registration and whenever a service wants to talk to another service let's say the currency calculation Service wants to talk with currency exchange service what would it do. It would talk to the name server and it would ask the name server what are the instances of the currency exchange service that are currently running. This is called Service discovery. so, the currency calculation Service is asking for the location of the currency exchange service the instances of currency exchange service that is called service discovery.

To Enable eureka server: @EnableEurekaServer

```
spring.application.name=netflix-eureka-naming-server
server.port=8781
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Why API Gate Ways ?



API GATEWAYS

- Authentication, authorization and security
- Rate Limits
- Fault Tolerance
- Service Aggregation

We will implement all the above all by using the Zuul.

So, what we'll do is, if any request that comes through the gateway. What we'll log it. Let's implement that during this step. I'm opening eclipse and I'm in the Netflix Zuul API gateway server project and I would want to create a new class.

Distributed Tracing using Sleuth

Let's say there is a small defect. This service is not really working fine and you would want o debug it. How do you do that. We'll look at the look at the currency calculation service do we look at the currency exchange service or We'll look at the API gateway. How do you find out where the defect is and how do I know what is happening in total with that specific request. One of the most important things with implementing a micro services architecture is we would need to have something called distributed tracing. I would want one place where I can go and see what happened with a specific request. I would want to have one single centralized location where I can see the complete chain of what happened with a specific request. This is especially important because there are variety of components that are involved. When we talk about the typical micro service architecture until now we looked at that different. micro services we talked about API gateways we talked about naming server we talked about configuration server. I want n number of components are involved. If you want to solve a problem and if you want to debug a problem through this you would need a centralized kind of information. And that's where this distributed tracing comes into picture. There are a variety of options that are present for distributed tracing. What we'll do in this section is to use something called Spring cloud sleuth with Zipkin.

Spring Cloud Bus:

Hystrix

If some service is having problem, It would prevent the entire chain from going down.

Hystrix framework helps us to build fault tolerant micro services. How do we do that.

.

Maven:

- **What maven Update will do:** Maven update essentially looks at your pom.xml and checks whether new dependencies have been added or existing ones have been changed (like, dep. version upgrade). If so, it fetches the dependencies libraries to your maven local repo to use for your code.
- **Maven clean:** removes the target folder - it deletes all class files, the java docs, the jars, reports and so on. If you don't **clean** , then **maven** will only "**do** what has to be done", like it won't compile classes when the corresponding source files haven't changed (in brief)

- **Maven-source:** The "maven-source" plugin is used to pack your source code and deploy along with your project. This is extremely useful, for developers who use your deployed project and want to attach your source code for debugging.
-

Dependencies with explanation:

1. **Spring Boot actuator:** Supports built or custom end points that let you monitor the application – Such as application health, metrics, sessions etc.
2. **Config –** Client : Client that connects to the spring cloud config server that lets you fetch the config details.
3. **Spring web :** Builds web including RESTful for application using spring MVC and uses apache tomcat as default embedded container.
4. **Spring boot DevTools:** Provides fast application restarts, live reloads and configuration for enhanced development experience.
5. **H2 Data base:** Provides a fast in memory data base that supports JDBC API and R2DBC access with small 2MB footprint. Supports embedded, server modes as well as browser-based console applications.
6.

To Set application name and port for a service:

```
spring.application.name=currency-exchange-service
server.port=8000
```

@RestContoller: To be able to send json request and receive response in json format out we use @RestContoller. A convenience annotation that is itself annotated with @Controller and @ResponseBody.

@GetMapping: Annotation for mapping HTTP {@code GET} requests onto specific handler methods. Specifically, @GetMapping} is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

@PathVariable: Annotation which indicates that a method parameter should be bound to a URI template variable.
Supported for RequestMapping annotated handler methods. If the method parameter is java.util.Map Map String, then the map is populated with all path variable names and values.

Creating an Empty or Default constructor will make JPA happy or easy to create Object in DB.

How to get local port in the current Environment:

```
@Autowired
private Environment environment;

int port=Integer.parseInt(environment.getProperty("local.server.port"));
```

@Entity: Specifies that the class is an entity. This annotation is applied to the
 * entity class.

@ID: Specifies the primary key of an entity. The field or property to which the **Id** annotation is applied should be one of the following types:
any Java primitive type or any primitive wrapper type.

@Column: Specifies the mapped column for a persistent property or field. If no Column annotation is specified, the default values apply.

```
Map<String, String> uriVariables = new HashMap<>();

             uriVariables.put("from", from);
             uriVariables.put("to", to);
             ResponseEntity<CurrencyConversionBean> responseEntity = new
RestTemplate().getForEntity(
                          "http://localhost:8000/currency-
exchange/from/{from}/to/{to}", CurrencyConversionBean.class,
                          uriVariables);

             CurrencyConversionBean body = responseEntity.getBody();
return new CurrencyConversionBean(body.getId(), from, to,
body.getConversionMultiple(), quantity,
quantity.multiply(body.getConversionMultiple()), port);
```

Feign:

The above is to implement a Rest Template to communicate with Other Micro service in the architecture. But having a greater number of services and writing the above code for all of them is little complex and time consuming, to avoid that we have a solution called **Feign.**

Uses of Feign :

1. Feign makes it very easy to invoke other Micro services
2. It provides integration called Ribbon which is client-side load balancing.

Step1:

```
      <dependency>
             <groupId>org.springframework.cloud</groupId>
             <artifactId>spring-cloud-starter-openfeign</artifactId>
      </dependency>
```

Step 2:

That’s the dependency to add Feign into the MS.
Ok After that we must enable the Feign using the below annotation with parameter as current package.

```
@EnableFeignClients("Org.SpringBoot.Rangan.currencyConversionservice")
```

Step 3:

Just like the way we created a Repository proxy to speak with JPA we must create a proxy to speak with other Micro services.

```
@GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")
	public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from,
@PathVariable String to, @PathVariable BigDecimal quantity) {

		int port =
Integer.parseInt(environment.getProperty("local.server.port"));
		CurrencyConversionBean body = proxy.retriveExchnagevalue(from, to);

		return new CurrencyConversionBean(body.getId(), from, to,
body.getConversionMultiple(), quantity,
				quantity.multiply(body.getConversionMultiple()), port);
		}
```

Ribbon:

Currently we are manually configuring the port for the currency exchange service to communicate with currency conversion service in Feign Proxy as shown below.

```
@FeignClient(name="currency-exchange-service", url="localhost:8000")
```

To avoid this manual configuration and to automatically distribute the load we have service called **Ribbon.**

We are using Feign proxy to call the Currency Exchange Service, so Ribbon can make use of the Feign configuration that we have already done and help us distribute the calls between different instances of the Currency Exchange Service.

So, what we'll do in this step is, enable Ribbon on the Currency Calculation Service and we would see that once we enable Ribbon, we would distribute the load between the different instances of the Currency Exchange Service.

Dependency for Ribbon:

```
	<dependency>
		<groupId>org.springframework.cloud</groupId>
		<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
	</dependency>
```

```
//@FeignClient(name="currency-exchange-service", url="localhost:8000")
@FeignClient(name="currency-exchange-service")
```

```
@RibbonClient(name="currency-exchange-service")
public interface CurrencyExchnageServiceProxy {
}
```

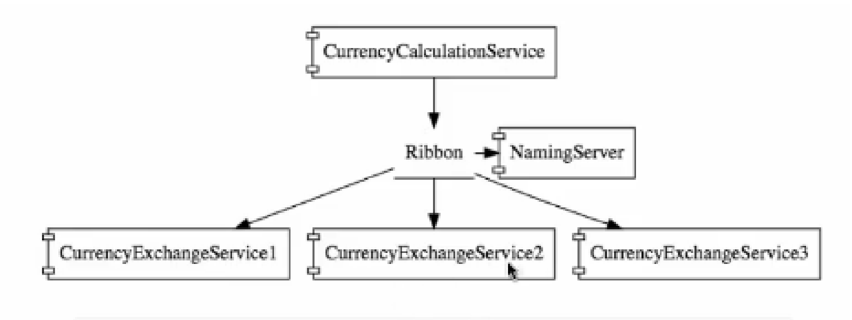
After adding the dependency and adding the annotation in Feign Proxy we have add below into the property file of the service which want to speak with other multiple instances(in our case currency exchange service).

Need add the ports in properties file of currency conversion service like below:

```
currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001
```

Now Ribbon will distribute load or calls from currency-conversion-service equally among both the ports.

Naming Server / Eureka Naming Server



Ribbon Load Balancing

Let's assume that I would start up another instance of the **currency exchange service** let's call it currency exchange service 3 and launch it up on **8002,** will ribbon be able to distribute the load to it with things as they stand right now. Think about it.

Thing is if I would want ribbon to distribute the load to the new server, I would need to add it to the configuration like below. so I would need to change my application configuration whenever a new server is created.

```
currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001,http://localhost:8002
```

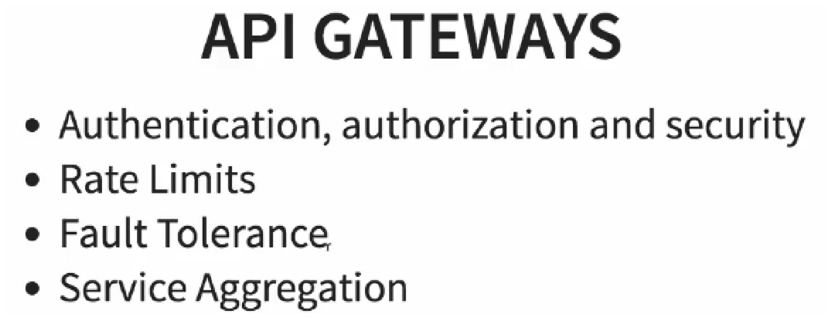
But if want to increase or decrease the bandwidth, we have to manually add or delete servers from here… that's nit a good thing so to handle the same we have service called Discovery client and Discovery server.

Netflix Eureka sever and Discovery client :

To enable we must add in the below in the service intended to access multiple instances of other services:

```
eureka.client.service-url.default-zone=http://http://localhost:8761/eureka
```

API gateways:Zuul



API GATEWAYS

- Authentication, authorization and security
- Rate Limits
- Fault Tolerance
- Service Aggregation