1. JAVA 8 FEATURES

forEach() method in Iterable interface.
default and static methods in Interfaces.
Functional Interfaces and Lambda Expressions.
Java Stream API for Bulk Data Operations on Collections.
Java Time API.
Collection API improvements.
Concurrency API improvements.
Java IO improvements

2.OOPS

Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

Inheritance: Inheritance is a process where one class acquires the properties of another.
Encapsulation: Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.
Abstraction: Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.
Polymorphism: Polymorphism is the ability of a variable, function or object to take multiple forms.

3. POLYMORPHISM

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.---- run time polymorphism

Method overloading
in method overloading we have same method with different type of arguments or different number of argruments. the other name of method over loading is compile time polymorphism.

4. INHERITANCE

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).
Inheritance in Java is a concept that acquires the properties from one class to other classes; for example, the relationship between father and son.

In Java, a class can inherit attributes and methods from another class. The class that inherits the properties is known as the sub-class or the child class.

5. ENCAPSULATION

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

encapsulation in java
We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java
By providing only a setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.

It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.

6. ABSTRACTION AND INTERFACE

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

interface doesn't have any method having body
all the methods defined inside interface are abstract method.
interface is going to be a contract.
the sub class that is going to implements interface should implement all the method shown in interface.

using interface we are going to get 100% abstraction in java.

7. FUNCTIONAL INTERFACE

A functional interface is an interface annotated with @FunctionalInterface annotation and contains only one abstract method, but the interface can have multiple default methods

As the name suggests, a functional interface is an interface that represents a function. Technically, an interface with just one abstract method is called a functional interface.
Eg.runnable, callable, comparable(compareTo()), comparator

8. LAMDA EXPRESSION

A lambda expression can implement a functional interface by defining an anonymous function that can be passed as an argument to some method.

9. EXPLAIN FUNCTION INTERFACE PREDICATE, CONSUMER, SUPPLIER

Supplier<T> represents a supplier of results.

Consumer is functional interface that takes in one argument and returns nothing.

Represents a boolean-valued function of one argument.

10. MARKER INTERFACE

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Cloneable and Remote interface.

11. COMPARABLE VS COMPARATOR INTERFACE

COMPARABLE
A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface to compare its instances.
Consider a Book class that has members like, rating, name, year. Suppose we wish to sort a list of Books based on year of release. We can implement the Comparable interface with the Book class, and we override the method compareTo() of Comparable interface.

COMPARATOR
Comparator is used when we want to sort a collection of objects which can be compared with each other. This comparison can be done using Comparable interface as well, but it restrict you compare these objects in a single particular way only. If you want to sort this collection, based on multiple criterias/fields, then you have to use Comparator only.

12. ABSTRACT CLASS VS INTERFACE

Interface
What is Interface in Java?
An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. A Java interface contains static constants and abstract methods. A class can implement multiple interfaces. In Java, interfaces are declared using the interface keyword. All methods in the interface are implicitly public and abstract.

Abstraction in Java
Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for

example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

13. MEMORY- HEAP, STACK, STATIC

The code section contains your bytecode.
The Stack section of memory contains methods, local variables, and reference variables.
The Heap section contains Objects (may also contain reference variables).
The Static section contains Static data/methods.

14. STREAM IN JAVA

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

15. EQUALS AND HASHCODE IN JAVA

equals() method
Java equals()
The java equals() is a method of lang.Object class, and it is used to compare two objects. To compare two objects that whether they are the same, it compares the values of both the object's attributes.
By default, two objects will be the same only if stored in the same memory location

In java equals() method is used to compare equality of two Objects.

hashCode() method
It returns the hashcode value as an Integer. Hashcode value is mostly used in hashing based collections like HashMap, HashSet, HashTable….etc. This method must be overridden in every class which overrides equals() method.

16.WHY STRING IS IMMUTABLE

Immutable class means that once an object is created, we cannot change its content. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well.
https://www.geeksforgeeks.org/create-immutable-class-java/
The class must be declared as final (So that child classes can't be created)
Data members in the class must be declared as private (So that direct access is not allowed)
Data members in the class must be declared as final (So that we can't change the value of it after object creation)

The String is immutable in Java because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability

and to not allow others to extend it.

The String objects are cached in the String pool, and it makes the String immutable

17. STRING BUFFER AND STRING BUILDER

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable

```
//Java Program to demonstrate the use of StringBuffer class.
public class BufferTest{
public static void main(String[] args){
StringBuffer buffer=new StringBuffer("hello");
buffer.append("java");
System.out.println(buffer);
}
}
```

hellojava

18. WHAT IS MICROSERVICE

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

19. DEEP COPY, SHALLOW COPY

shallow copy
Whenever we use default implementation of clone method we get shallow copy of object means it creates new instance and copies all the field of object to that new instance and returns it as object type, we need to explicitly cast it back to our original object. This is shallow copy of the object.

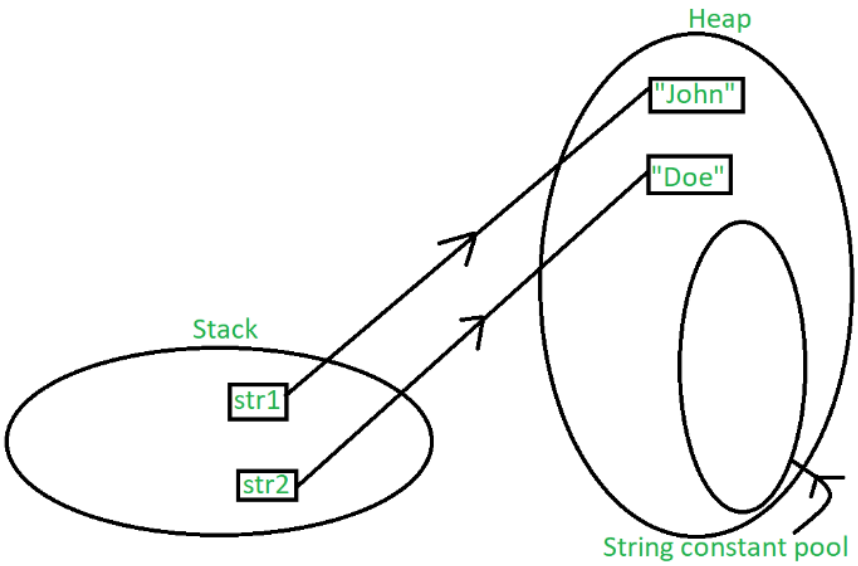
deep copy
A deep copy means actually creating a new array and copying over the values.

Lazy Copy
A lazy copy can be defined as a combination of both shallow copy and deep copy. The mechanism follows a simple approach – at the initial state, shallow copy approach is used. A counter is also used to keep a track on how many objects share the data. When the program wants to modify the original object, it checks whether the object is shared or not. If the object is shared, then the deep copy mechanism is initiated.

20. STRING CONSTANT POOL

```
String str1 = new String("John");
String str2 = new String("Doe");
```

The following illustration explains the memory allocation for the above declaration:



21. MULTI THREADING

Threads are light-weight processes within a process.
Java creates threads by using a "Thread Class"
Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

Extending the Thread class
Implementing the Runnable Interface
Multithreading in Java is a process of executing two or more threads simultaneously to maximum utilization of CPU. Multithreaded applications execute two or more threads run concurrently. Hence, it is also known as Concurrency in Java. Each thread runs parallel to each other. Mulitple threads don't allocate separate memory area, hence they save memory. Also, context switching between threads takes less time.

22. CALLABLE RUNNABLE

The Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return

a result and cannot throw a checked exception.

Callable Interface

In a callable interface that basically throws a checked exception and returns some results. This is one of the major differences between the upcoming Runnable interface where no value is being returned. In this interface, it simply computes a result else throws an exception if unable to do so.

23. DOCKER

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

24. STATIC METHOD

When a method is declared with static keyword, it is known as static method. The most common example of a static method is main( ) method.As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object.Methods declared as static have several restrictions:

They can only directly call other static methods.
They can only directly access static data.
They cannot refer to this or super in any way.

25. CLASS AND OBJECT

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

State: It is represented by attributes of an object. It also reflects the properties of an object.
Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.
Identity: It gives a unique name to an object and enables one object to interact with other objects.

26. TERNARY OPERATOR

max = (n1 > n2) ? n1 : n2;
Java ternary operator is the only conditional operator that takes three operands. It’s a one-liner replacement for if-then-else statement and used a lot in Java programming. We can

use the ternary operator in place of if-else conditions or even switch conditions using nested ternary operators. Although it follows the same algorithm as of if-else statement, the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.
variable = Expression1 ? Expression2: Expression3

27. REFLECTION

Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

28. FAIL SAFE FAIL FAST

Iterators in java are used to iterate over the Collection objects.

Fail-Fast iterators immediately throw ConcurrentModificationException if there is structural modification of the collection. Structural modification means adding, removing any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes are some examples of fail-fast Iterator.

Fail-Safe iterators don't throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that's why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes are examples of fail-safe Iterator.

29. SERIALIZAITON IN JAVA

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

30. SERVLET & JSP

servlet are java enabled web server to handle request obtained from web server
jsp is a java server page which is server side language for creating web application

31. HOW TO INSTANTIATE AN OBJECT IN JAVA

Java provides the new keyword to instantiate a class. We can also instantiate the above class as follows if we defining a reference variable. We observe that when we use the new keyword followed by the class name, it creates an instance or object of that class.

32. STATUS CODE, RESTFUL API

200 (OK)
201 (Created)
400 (Bad Request)

401 (Unauthorized)
403 (Forbidden)
404 (Not Found)
500 (Internal Server Error)

HTTP Response broadly has 3 main components:
Status Line
Headers
Body (Optional)

33. DEPENDECNY INJECTION

Dependency Injection (DI) is a programming technique that makes a class independent of its dependencies. “In software engineering, dependency injection is a technique whereby one object supplies the dependencies of another object. A 'dependency' is an object that can be used, for example as a service.

Spring IoC is achieved through Dependency Injection. Dependency Injection is the method of providing the dependencies and Inversion of Control is the end result of Dependency Injection. IoC is a design principle where the control flow of the program is inverted.

34. HASH MAP AND ITS INTERNAL WORKING

hashMap is a data structure that uses a hash function to map identifying values, known as keys, to their associated values. It contains “key-value” pairs and allows retrieving value by key.
HashMap allows to store one null key and many null values i.e. many keys can have null value in java.
Implementation to illustrate above methods

// Java program illustrating use of HashMap methods -
//**put**(), **get**(), **isEmpty**() **and size**()

35. ARRAY VS ARRAYLIST

An array is basic functionality provided by Java. ArrayList is part of collection framework in Java. Therefore array members are accessed using [], while ArrayList has a set of methods to access elements and modify them.
Array: Simple fixed sized arrays that we create in Java, like below
int arr[] = new int[10]
ArrayList : Dynamic sized arrays in Java that implement List interface.
ArrayList<Type> arrL = new ArrayList<Type>();
Here Type is the type of elements in ArrayList to
be created

Primitive data types cannot be stored in ArrayList but can be in Array.
ArrayList is a kind of List and List implements Collection interface.

An Array (System.Array) is fixed in size once it is allocated. You can't add items to it or remove items from it.

36. LIST VS SET

1. The List is an ordered sequence. 1. The Set is an unordered sequence.
2. List allows duplicate elements 2. Set doesn't allow duplicate elements.
3. Elements by their position can be accessed. 3. Position access to elements is not allowed.
4. Multiple null elements can be stored. 4. Null element can store only once.
5. List implementations are ArrayList, LinkedList, Vector, Stack 5. Set implementations are HashSet, LinkedHashSet.

37. HASH MAP VS CONCURRENT HASH MAP VS SYNCHRONISED HASH MAP

1. ConcurrentHashMap: ConcurrentHashMap is a class which implements the ConcurrentMap interface. It uses Hashtable, underlined data structure. As we know, while dealing with thread in our application HashMap is not a good choice because of the performance issue. To resolve this issue, we use ConcurrentHashMap in our application. ConcurrentHashMap is thread-safe therefore multiple threads can operate on a single object without any problem.
2. Synchronized HashMap: Java HashMap is a non-synchronized collection class. If we need to perform thread-safe operations on it then we must need to synchronize it explicitly. The synchronizedMap() method of java.util.Collections class is used to synchronize it. It returns a synchronized (thread-safe) map backed by the specified map.

38. SPRING VS SPRING BOOT

SPRING
spring framework which supports dependency injection used to develop loosely coupled application.
there are various features Spring Security,Spring AOP, Spring ORM, Spring DAO, Spring WebMVC
SPRING BOOT
spring boot is combination of spring framework and embedded servers.

It is developed on Spring framework, with Spring boot we can make production ready and deployable web application with only few configuration change.
Spring Boot is known for Auto Configuration
@EnableAutoConfiguration - that means it automatically maintains the configuration part of application

39. @OVERRIDE ANNOTATION

@override annotation is used when a method override another method. It indicates a child class method is overriding its base class method.

The purpose of Method Overriding is that if the derived class wants to give its own implementation it can give by overriding the method of the parent class. When we call this overridden method, it will execute the method of the child class, not the parent class.

40. SPECIFIC ANNOTATION TO START SPRING BOOT APP AND WHAT IS THAT COMBINED OF

@SpringBootApplication: It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

41. SPRING BOOT ACTUATOR

Spring Boot includes a number of additional features to help you monitor and manage your application when you push it to production. You can choose to manage and monitor your application by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to your application.

ENDPOINTS USAGE
/metrics To view the application metrics such as memory used, memory free, threads, classes, system uptime etc.
/env To view the list of Environment variables used in the application.
/beans To view the Spring beans and its types, scopes and dependency.
/health To view the application health -up (localhost:8090/actuator/health)
/info To view the information about the Spring Boot application.
/trace To view the list of Traces of your Rest endpoints.

42. DIFFERENT ANNOTATION USED IN SPRING BOOT

1. @Bean
   The @Bean annotations are used at the method level and indicate that a method produces a bean that is to be managed by Spring container. It is an alternative to th\e XML<bean> tag.
   2. @Service
   It is used at the class level. It shows that the annotated class is a service class, such as business basic logic, and call external APIs.
   3. @Repository
   It is a Data Access Object (DAO) that accesses the database directly. It indicates that the annotated class is a repository.
   4. @Configuration
   It is used as a source of bean definitions. It is a class-level annotation.
   5. @Controller
   The annotation is used to indicate that the class is a web request handler. It is often used to present web pages. It is most commonly used with @RequestMapping annotation.
   6. @RequestMapping
   RequestMapping is used to map the HTTP request. It is used with the class as well as the method. It has many other optional elements like consumes, name, method, request, path, etc.
   7. @Autowired
   @ Autowied annotation is used for dependency injections we can automatically fetch the object from any bean.

   This annotation is used to auto-wire spring bean on setter methods, constructor and instance variable. It injects object dependency implicitly. When we use this annotation, the spring container auto-wires the bean by its matching data type.
   8. @Component

It is a class-level annotation that turns the class into Spring bean at the auto-scan time.

9. @SpringBootApplication

It consists of @Configuration, @ComponentScan, and @EnabeAutoConfiguration. The class annotated with @SpringBootApplication is kept in the base package. This annotation does the component scan. However, only the sub-packages are scanned.

10. @EnableAutoConfiguration

It is placed on the main application class. Based on classpath settings, other beans, and various property settings, this annotation instructs SpringBoot to start adding beans.

11. @ComponetScan

It is used to scan a package of beans. It is used with the annotation @Configuration to allow Spring to know the packages to be scanned for annotated components. This annotation is also used to specify base packages.

12. @Required

This annotation is applied to bean setter methods. It indicates that the required property must be filled at the configuration time in the affected bean, or else it throws an exception: BeanInitializationException.

13. @Qualifier

It is used along with @Autowired annotation. It is used when more control is required over the dependency injection process. Individual constructor arguments or method parameters can be specified by using this annotation. Confusion arises when more than one bean of the same type is created, and only one of them is to be wired with a property, @Qualifier is used to get rid of the confusion.

14. @CookieValue

It is used at the method parameter level as an argument of the request mapping method. For a given cookie name, the HTTP cookie is bound to a @CookieValue parameter.

15. @Lazy

It is used in the component class. At startup, all auto-wired dependencies are created and configured. But a @Lazy annotation can be created if a bean is to be initialized lazily. This means that only if it is requested for a bean will be created. It can also be used on @Configuartion classes. It’s an indication that all @Bean methods within that @Configuration should be lazily initialized.

43. @TRANSACTIONAL

@Transactional annotation is used when you want the certain method/class(=all methods inside) to be executed in a transaction.

Let's assume user A wants to transfer 100$ to user B. What happens is:

We decrease A's account by 100$
We add 100$ to B's account
Let's assume the exception is thrown after succeeding 1) and before executing 2). Now we would have some kind of inconsistency because A lost 100$ while B got nothing. Transactions means all or nothing. If there is an exception thrown

somewhere in the method, changes are not persisted in the database. Something called rollback happens.

If you don't specify @Transactional, each DB call will be in a different transaction. Generally the @Transactional annotation is written at the service level.

It is used to combine more than one writes on a database as a single atomic operation.

When somebody call the method annotated with @Transactional all or none of the writes on the database is executed.

By using @Transactional, many important aspects such as transaction propagation are handled automatically. In this case if another transactional method is called by businessLogic(), that method will have the option of joining the ongoing transaction.

44. @PATH VARIABLE VS @REQUESTPARAM

1) The @RequestParam is used to extract query parameters while @PathVariable is used to extract data right from the URI.{id}.

   2) @RequestParam is more useful on a traditional web application where data is mostly passed in the query abatements while @PathVariable is more suitable for RESTful web services where URL contains values, like http://localhost:8080/book/9783827319333, here data, which is ISBN number is part of URI.

   @RequestMapping(value="/order/{orderId}/receipts", method = RequestMethod.GET) public List listUsersInvoices(@PathVariable("orderId") int order, @RequestParam(value = "date", required = false) Date dateOrNull) { ... }

45. SPRING BOOT FEATURES

Web Development
SpringApplication
Application events and listeners
Admin features
Externalized Configuration
Properties Files
YAML Support
Type-safe Configuration
Logging
Security

46. SPRING MVC

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application

(input logic, business logic, and UI logic), while providing a loose coupling between these elements.

The Model encapsulates the application data and in general they will consist of POJO.

The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.

The Controller is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

The DispatcherServlet
The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC DispatcherServlet is

47. DIFFERENCE CONTROLLER & REST CONTROLLER

Difference between @RestController and @Controller in Spring. ... The @Controller is a common annotation which is used to mark a class as Spring MVC Controller while the @RestController is a special controller used in RESTFul web services and the equivalent of @Controller + @ResponseBody .

48. @PROFILE

Spring 3.1 introduced the annotation @Profile. Profile annotation is a logical grouping that can be activated programmatically. It can be used on type-level annotation on any class or it can be used as a meta-annotation for composing custom stereotype annotations or as a method-level annotation on any @Bean method. Essence is, @Profile is used to conditionally activate/register.

49. APPLICATION PROPERTIES

What is the use of application properties file?

properties is a key-value repository resource that is widely used for configuration properties and runtime variables. Those are the most common uses of the application. properties file. However, as a key-value storage resource, it can also function similarly to how a dictionary does in Python or JavaScript.

50. CHANGE DEFAULT SERVER IN SPRING BOOT

You will need to update pom.xml, add the dependency for spring-boot-starter-jetty. Also, you will need to exclude default added spring-boot-starter-tomcat dependency.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<exclusions>
<exclusion>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
```

</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jetty</artifactId>
</dependency>

51. SCOPE OF BEANS AND DEFAULT SCOPE

Bean scope decides which type of bean instance should be returned to the caller from the Spring container. Scopes are defined using @Scope annotation.

There are six types of bean scopes in the latest version of the Spring framework:

singleton
prototype
request
session
application
WebSocket
The scopes request, session, application, and WebSocket are only available in a web-aware application.

52. DIFFERENT DESIGN USED

Dependency injection or inversion of control (IOC): ...
Factory Design Pattern: ...
Proxy Design Pattern: ...
Singleton Design Pattern: ...
Model View Controller (MVC): ...
Front Controller Design Pattern: ...
View Helper: ...
Template method:

53. SINGLETON DESIGN PATTERN, SINGLETON CLASS

Single design pattern
The singleton pattern is one of the simplest design patterns. Sometimes we need to have only one instance of our class for example a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly.

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time.
After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class

through any instance, it affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

54. GET, POST, PUT , DELETE

Spring @GetMapping Example

```
@RestController
public class UserController {

@Autowired
UserService userService;

@GetMapping("users")
public ResponseEntity<List<User>> getAll() {
return new ResponseEntity<>(userService.getAll(), HttpStatus.OK);
}

@GetMapping("users/{id}")
public ResponseEntity<User> getById(@PathVariable long id) {
Optional<User> user = userService.getById(id);
if (user.isPresent()) {
return new ResponseEntity<>(user.get(), HttpStatus.OK);
} else {
throw new RecordNotFoundException();
}
}
}
```

Spring @PostMapping Example

```
@PostMapping(path = "users",
consumes = MediaType.APPLICATION_JSON_VALUE,
produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<User> create(@RequestBody User newUser) {
User user = userService.save(newUser);
if (user == null) {
throw new ServerException();
} else {
return new ResponseEntity<>(user, HttpStatus.CREATED);
}
}
```

55. REST API

A REST API - or Representational State Transfer is a way for two computer systems to communicate over HTTP in a similar way to web browsers and servers. Sharing data between two or more systems has always been a fundamental requirement of software development.

56. JOINS

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

57. BREAK

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition.

58. HOW TO AVOID DUPLICATES IN SQL

Adding the Distinct Keyword to a Query to Eliminate Duplicates. ...
The SELECT DISTINCT is used to get distinct data from tables using a query. The below SQL query selects only the DISTINCT values from the “Country” column in the “Customers” table: SELECT DISTINCT Country FROM Customers;
Using SQL WHERE NOT IN to Remove Duplicate Values. ...
Using INSERT INTO WHERE NOT IN SQL Operator. ...
Using SQL INSERT INTO IF NOT EXIST. ...
Using COUNT(*) = 0 Without Duplicates.

59. WHY HIBERNATE PREFERRED OVER JDBC

DBC
jdbc stands for java database connectivity . it is a free open source application programming interface for java that enables applications to access databases. it enables developers to create queries and update data to a relational database using the structured query language (sql).
Hibernate
hibernate is a free, open source object-relational mapping library for java designed to map objects to an rdbms and to implement the object-oriented programming concepts in a relational database.

unlike jdbc, hibernate connects with the database itself and uses hql (hibernate query language) to execute the queries, then maps the results to java objects.
the results are mapped to objects based on the properties given in the hibernate configuration xml file.

What Is Java Persistence API?
The Java Persistence API provides a specification for persisting, reading, and managing

data from your Java object to relational tables in the database.

What is JPA?
A JPA (Java Persistence API) is a specification of Java which is used to access, manage, and persist data between Java object and relational database. It is considered as a standard approach for Object Relational Mapping.

JPA can be seen as a bridge between object-oriented domain models and relational database systems. Being a specification, JPA doesn't perform any operation by itself. Thus, it requires implementation. So, ORM tools like Hibernate, TopLink, and iBatis implements JPA specifications for data persistence.

What is Hibernate?
A Hibernate is a Java framework which is used to store the Java objects in the relational database system. It is an open-source, lightweight, ORM (Object Relational Mapping) tool.

Hibernate is an implementation of JPA. So, it follows the common standards provided by the JPA.

60. WANT ONLY 10 RECORDS OF 100 WRITE SQL QUERY

How can I get only 10 records from a table where there are more than 1000 records. I have a test table with rowid, name, cost.

SELECT name, cost FROM test LIMIT 10

61. INDEXING A DATABASE

Select * from Student where Rollno = 1;
if data is non-ordered
I/O cost is reduced by indexing.
Indexing is the way to get an unordered table into an order that will maximize the query's efficiency while searching.

62. TABLE, ENTITY

```
ALTER DATABASE `MATERIAL` CHARACTER SET utf8 COLLATE utf8_general_ci;

USE `MATERIAL`;

CREATE TABLE IF NOT EXISTS `ADMIN_USERS` (
`ID` BIGINT NOT NULL AUTO_INCREMENT,
FULL_NAME VARCHAR(200) NOT NULL,
EMAIL VARCHAR(200) NOT NULL,
MOBILE VARCHAR(20) NOT NULL,
USERNAME VARCHAR(200) NOT NULL,
PASSWORD VARCHAR(200) NOT NULL,
ROLE VARCHAR(1) NOT NULL,
```

```
CREATED_BY VARCHAR(200) NOT NULL,
CREATED_DATE DATETIME NOT NULL,
UPDATED_BY VARCHAR(200) NOT NULL,
UPDATED_DATE DATETIME NOT NULL,
ACTIVE VARCHAR(1) NOT NULL,
DELETED VARCHAR(1) NOT NULL,
PRIMARY KEY (`ID`)
) ENGINE=INNODB DEFAULT CHARSET=UTF8;
CREATE INDEX ON
```

Entity

```
import java.util.Date;
import javax.persistence.*;


@Entity
@Table(name = "ADMIN_USERS")
public class AdminUser {

@Id
@Column(name = "ID")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(name = "FULL_NAME")
private String fullName;

@Column(name = "EMAIL")
private String email;

@Column(name = "MOBILE")
private String mobile;

@Column(name = "USERNAME")
private String username;

@Column(name = "PASSWORD")
private String password;
```

63. PRIMARY KEY VS FOREIGN KEY

Primary key:
it is known as canditate key with no null values as it uniquely identify record of a table.
Foreign key:
reference of Primary key into another table. It is the primary key of another table(reference table). DepId is the foreign key of table Employee.

64. HIBERNATE OVERVIEW

Hibernate is an Object-Relational Mapping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.

65. CONNECTING DATABASE AND JAVA

pom.xml
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<scope>runtime</scope>
</dependency>

application.properties
spring.datasource.url=jdbc:mysql://localhost:3306/bookshop
spring.datasource.username=root
spring.datasource.password=password
https://www.codejava.net/frameworks/spring-boot/connect-to-mysql-database-examples

connecting spring boot application to database jdbc
Spring Boot starter projects provide the required libraries to connect the application with JDBC. So, for example, if you just have to create an application and connect it with MySQL database, you can follow the below steps:

Step 1: Create a database in MySQL

```
1
CREATE DATABASE example;
```

Step 2: Then you have to create a table inside this database.

```
1
CREATE TABLE customers(customerid INT PRIMARY KEY NOT NULL AUTO_INCREMENT, customername VARCHAR(255));
```

Step 3: Now, create a Spring Boot project and provide the required details

Step 4: Add the JDBC, MySQL and web dependencies.

Step 5: Once the project is created, you have to configure the database into application properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/example
spring.datasource.username=root
spring.datasource.password=edureka
spring.jpa.hibernate.ddl-auto=create-drop
```

Step 6: The main application.java class should have the following code:

```
package com.edureka;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SampleApplication {
public static void main(String[] args) {
SpringApplication.run(SampleApplication.class, args);
}
}
```

Next, you have to create a controller to handle the HTTP requests, by mentioning the following code:

```
package com.edureka;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class JdbcController {
@Autowired
JdbcTemplate jdbc;
@RequestMapping("/insert")
public String index(){
jdbc.execute("insert into customers(name)values('Aryya')");
return "Data Entry Successful";
}
}
```

Step 8: Finally, execute this project as a Java application.
Step 9: Next, open the URL (localhost:8080/insert), and you will see the output as Data Entry Successful. You can also go forward and check if the data is entered into the table.

66. EAGER AND LAZY LOADING IN HIBERNATE

Eager Loading is a design pattern in which data initialization occurs on the spot
Lazy Loading is a design pattern which is used to defer initialization of an object as long as it's possible

67. JPA VS JDBC, SPRING DATA JPA

Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data

access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically

JPA vs JDBC
https://www.baeldung.com/jpa-vs-jdbc#1-database-interactions
https://spring.io/projects/spring-data-jpa
https://spring.io/projects/spring-data-jdbc

1. Overview
In this tutorial, we're going to look at the differences between the Java Database Connectivity (JDBC) API and the Java Persistence API (JPA).

2. What Is JDBC
JDBC is a programming-level interface for Java applications that communicate with a database. An application uses this API to communicate with a JDBC manager. It's the common API that our application code uses to communicate with the database. Beyond the API is the vendor-supplied, JDBC-compliant driver for the database we're using.

3. What Is JPA
JPA is a Java standard that allows us to bind Java objects to records in a relational database. It's one possible approach to Object Relationship Mapping(ORM), allowing the developer to retrieve, store, update, and delete data in a relational database using Java objects. Several implementations are available for the JPA specification.

4. JPA vs JDBC
When it comes to deciding how to communicate with back-end database systems, software architects face a significant technological challenge. The debate between JPA and JDBC is often the deciding factor, as the two database technologies take very different approaches to work with persistent data. Let's analyze the key differences between them.

4.1. Database Interactions
JDBC allows us to write SQL commands to read data from and update data to a relational database. JPA, unlike JDBC, allows developers to construct database-driven Java programs utilizing object-oriented semantics. The JPA annotations describe how a given Java class and its variables map to a given table and its columns in a database.

Let's see how we can map an Employee class to an employee database table:

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {
```

```
@Column(name = "employee_name")
private String employeeName;
}
```
The JPA framework then handles all the time-consuming, error-prone coding required to convert between object-oriented Java code and the back-end database.

4.2. Managing Associations
When associating database tables in a query with JDBC, we need to write out the full SQL query, while with JPA, we simply use annotations to create one-to-one, one-to-many, many-to-one, and many-to-many associations.

Let's say our employee table has a one-to-many relationship with the communication table:

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {

@OneToMany(mappedBy = "employee", fetch = FetchType.EAGER)
@OrderBy("firstName asc")
private Set communications;
}
```
The owner of this relationship is Communication, so we're using the mappedBy attribute in Employee to make it a bi-directional relationship.

4.3. Database Dependency
JDBC is database-dependent, which means that different scripts must be written for different databases. On the other side, JPA is database-agnostic, meaning that the same code can be used in a variety of databases with few (or no) modifications.

4.4. Exception Handling
Because JDBC throws checked exceptions, such as SQLException, we must write it in a try-catch block. On the other hand, the JPA framework uses only unchecked exceptions, like Hibernate. Hence, we don't need to catch or declare them at every place we're using them.

4.5. Performance
The difference between JPA and JDBC is essentially who does the coding: the JPA framework or a local developer. Either way, we'll have to deal with the object-relation impedance mismatch.

To be fair, when we write SQL queries incorrectly, JDBC performance can be abysmally sluggish. When deciding between the two technologies, performance shouldn't be a point of dispute. Professional developers are more than capable of producing Java applications that run equally well regardless of the technology utilized.

4.6. JDBC Dependency
JPA-based applications still use JDBC under the hood. Therefore, when we utilize JPA, our code is actually using the JDBC APIs for all database interactions. In other words, JPA serves as a layer of abstraction that hides the low-level JDBC calls from the developer,

making database programming considerably easier.

4.7. Transaction Management
In JDBC, transaction management is handled explicitly by using commit and rollback. On the other hand, transaction management is implicitly provided in JPA.

5. Pros and Cons
The most obvious benefit of JDBC over JPA is that it's simpler to understand. On the other side, if a developer doesn't grasp the internal workings of the JPA framework or database design, they will be unable to write good code.

Also, JPA is thought to be better suited for more sophisticated applications by many developers. But, JDBC is considered the preferable alternative if an application will use a simple database and we don't plan to migrate it to a different database vendor.

The main advantage of JPA over JDBC for developers is that they can code their Java applications using object-oriented principles and best practices without having to worry about database semantics. As a result, development can be completed more quickly, especially when software developers lack a solid understanding of SQL and relational databases.

Also, because a well-tested and robust framework is handling the interaction between the database and the Java application, we should see a decrease in errors from the database mapping layer when using JPA.

HIBERNATE
What Is Hibernate In Java?
Hibernate is a framework in Java which comes with an abstraction layer and handles the implementations internally. The implementations include tasks like writing a query for CRUD operations or establishing a connection with the databases, etc.

A framework is basically a software that provides abstraction on multiple technologies like JDBC, servlet, etc.

Hibernate develops persistence logic, which stores and processes the data for longer use. It is lightweight and an ORM tool, and most importantly open-source which gives it an edge over other frameworks.

What Is An ORM Tool?
It is a technique that maps the object stored in the database. An ORM tool simplifies data creation, manipulation, and access. It internally uses the Java API to interact with the databases.

Scenario based programs

1. SINGLETON CLASS JAVA PROGRAM

```
public class Singleton {

private static Singleton singleton = new Singleton( );

/* A private Constructor prevents any other
* class from instantiating.
*/
private Singleton() { }

/* Static 'instance' method */
public static Singleton getInstance( ) {
return singleton;
}

/* Other methods protected by singleton-ness */
protected static void demoMethod( ) {
System.out.println("demoMethod for singleton");
}
}
```

Here is the main program file where we will create a singleton object −

```
// File Name: SingletonDemo.java
public class SingletonDemo {

public static void main(String[] args) {
Singleton tmp = Singleton.getInstance( );
tmp.demoMethod( );
}
}
```

This will produce the following result −

Output

```
demoMethod for singleton
```

2. AVOID DUPLICATES IN HASH MAP

to avoid duplicate user defined objects as a key from HashMap. You can achieve this by implementing equals and hashcode methods at the user defined objects.

3. LIST TO SET

```
public class Example {
public static void main(String[] args) {
List l = new ArrayList();
l.add("Good");
l.add("Morning");
l.add("Morning");
```

```
Set s = new HashSet(l);
System.out.println(s);
}
}


public class Demo {
public static void main(String[] args) {
List<Integer> list = new ArrayList<Integer>();
list.add(10);
list.add(20);
list.add(30);
list.add(40);
list.add(50);
list.add(20);
list.add(40);
list.add(50);
Set<Integer> set = new HashSet<Integer>(list);
System.out.println("Set = ");
for (Object ob : set)
System.out.println(ob);
}
}
```

4. LIST TO MAP

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class TestListMap {

public static void main(String[] args) {

List<Hosting> list = new ArrayList<>();
list.add(new Hosting(1, "liquidweb.com", 80000));
list.add(new Hosting(2, "linode.com", 90000));
list.add(new Hosting(3, "digitalocean.com", 120000));
list.add(new Hosting(4, "aws.amazon.com", 200000));
list.add(new Hosting(5, "mkyong.com", 1));

// key = id, value - websites
Map<Integer, String> result1 = list.stream().collect(
Collectors.toMap(Hosting::getId, Hosting::getName));

System.out.println("Result 1 : " + result1);
```

```
// key = name, value - websites
Map<String, Long> result2 = list.stream().collect(
Collectors.toMap(Hosting::getName, Hosting::getWebsites));

System.out.println("Result 2 : " + result2);

// Same with result1, just different syntax
// key = id, value = name
Map<Integer, String> result3 = list.stream().collect(
Collectors.toMap(x -> x.getId(), x -> x.getName()));

System.out.println("Result 3 : " + result3);
}
}
```

5. STREAM MAP SCENARIO BASED

```
import java.util.stream.Stream;

public class Main
{
public static void main(String[] args)
{
Stream<Integer> numStream = Stream.of(1,3,5,4,2);

numStream.sorted()
.forEach(System.out::println);
}
}
```

6. VARIOUS WAYS OF SORTING PROGRAM SCENARIO BASED ANY ONE SORTING METHOD OF ARRAY

1. using Stream

Stream class provides a sorted method that can be used to sort a Stream, once you sort the stream you can collect the result in a List which will be sorted.

```
List<Integer> unsorted = Arrays.asList(11, 2, 5, 3, 2, 55, 32, 34);
List<Integer> sorted = unsorted.stream()
                               .sorted()
                               .collect(Collectors.toList());
Output
unsorted list : [11, 2, 5, 3, 2, 55, 32, 34]
sorted list using Java 8 Stream: [2, 2, 3, 5, 11, 32, 34, 55]
```

2. Using Collections.sort()

This is the oldest and classic way to sort a List in Java. This method is present from JDK 1.0 and it will work on all Java versions. It's a static method so you can directly call them using class names like `Collections.sort()` and it will sort in the increasing order of elements or default order whatever is defined by the Comparable method.

```
List<Integer> unsorted = Arrays.asList(11, 2, 5, 3, 2, 55, 32, 34);

Collections.sort(unsorted);
Output
sorted list using Collections.sort(): [2, 2, 3, 5, 11, 32, 34, 55]
```

3. Using List.sort()

This is the new way to sort a List in Java and it works from Java 8 and higher version. This was possible because Java 8 allows you to add static and default methods on an interface and the JDK team took advantage of that to define useful utility methods on the popular interfaces like Collection, List, and Map.

This is now preferred way to sort List in Java. It optionally take a comparator which you can provide to change the sorting order, if you want to sort in default order just provide null.

```
List<Integer> unsorted = Arrays.asList(11, 2, 5, 3, 2, 55, 32, 34);
unsorted.sort(null);

Ouptut:
sorted list using Java 8 List.sort():
[12, 15, 20, 23, 34, 101, 155, 312]
```

As I said, the `List.sort()` method expects a Comparator, but you can pass a null value as well. The null means list will be sorted in the natural order of elements.

Read more: https://www.java67.com/2021/09/3-ways-to-sort-list-in-java-8-and-11.html#ixzz7fnAKRnF8

7. EQUALS.TO VS == --> SCENARIO BASED

String s1 = "abcde";
String s2 = new String("abcde");
String s3 = "abcde";
s1 == s2 // is false
s1 == s3 // is true
s1.equals(s2) // is true

and if you do

String s1 = new String("abc");

String s2 = new String("abc");

System.out.println(s1==s2)

op:

false

8. STRING , NEW STRING DIFFERENCE --> SCENARIO BASED

When you use a string literal the string can be interned, but when you use new String("...") you get a new string object.

In this example both string literals refer the same object:
String a = "abc";
String b = "abc";
System.out.println(a == b); // true
Here, 2 different objects are created and they have different references:
String c = new String("abc");
String d = new String("abc");
System.out.println(c == d); // false
In general, you should use the string literal notation when possible. It is easier to read and it gives the compiler a chance to optimize your code.

9. WHEN WE USE LAMBDA EXPRESSION --> SCENARIO BASED

```
public class LambdaSortedLang {
public static void main(String[] args) {
ArrayList<String> allLang = new ArrayList<>();
allLang.add("tamil");
allLang.add("arab");
allLang.add("marathi");
allLang.add("malay");
allLang.add("hindi");
allLang.add("malayalam");

System.out.println(allLang);
List<String> sortedLang = allLang.stream().sorted().filter(s ->
s.startsWith("m")).collect(Collectors.toList());
System.out.println(sortedLang);
allLang.replaceAll(a-> a.toUpperCase());
System.out.println("Updated Lang :" +allLang);
}
}
```

10. WRITE SQL QUERY FOR SECOND LARGEST SALARY

SELECT MAX(SALARY) FROM Employee WHERE SALARY < (SELECT MAX(SALARY) FROM Employee);