Web Full Stack

Java

Java is a platform independent programming language & object oriented programming language

Platform Independent: It can be run on any platform

Object Oriented Language: it helps to create real-world entities in the application where be these real world entities are called as objects which will have 2 main things

1. properties: What object has
2. behaviours: What object does

Main building blocks of any object oriented language is

1. class: Blueprint of an object that describes it
2. object: It is a real-world entity which is in stance created from the class, it will have the initialized values for their properties

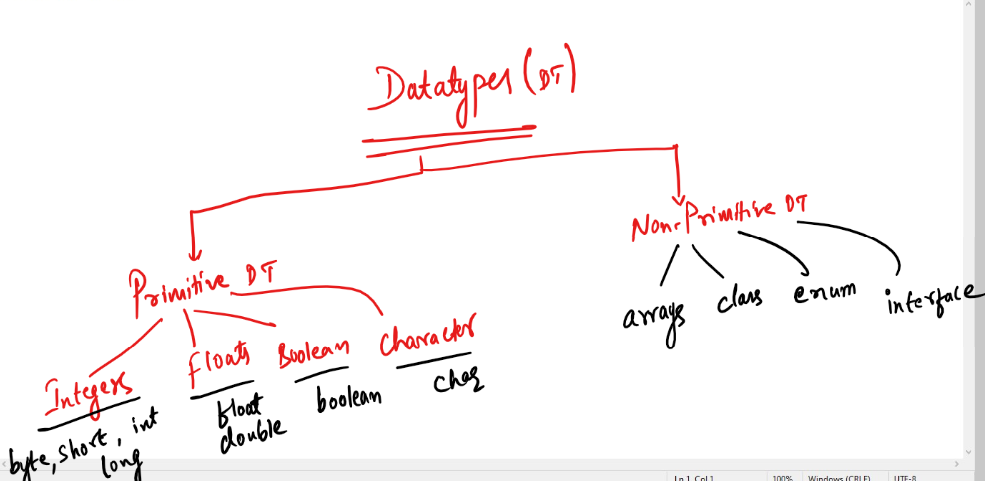
Fundamentals of Java

1. Operators: +, -, \*, /, =, %, ++, --, <, >, ==, <=, >=, !=
2. Datatypes: int, short, long, byte, double, float, boolean, char
3. Conditional Statements: if, else, switch
4. Loops: for, while, do-while
5. Arrays
6. Class & Objects

Java main entry point

public static void main(String[] args) {   
   
}

Datatypes



Operators: These help to operate on data

Conditional Statements

When you want some statements that needs to be executed based on the conditions you can use conditional constructs like

* if
* if else
* if else if else if … [or] if else if else if ….. else
* switch

Syntax:

if (condition) {   
 statements  
}

else {  
  
}

Multiple conditional statements  
if (condition) {….} else if (condition) {……} else { …. }

switch (choices) { case choiceValue: statement; break; }

Loops

for loop: It is used when you know how many iterations you want to do

while loop: It is used when you want to execute statements repeatedly until condition is false

do while loop: It is like while, but executes the statements at-least and then checks the condition

Arrays: It is container that can store multiple values of same datatypes

String class: It is used to work with collection of characters, it provides methods like length(), concat(), toUpperCase(), toLowerCase(), charAt(), split() and so on.

Day 2 Agenda

* Methods
* Classes & Objects
* Constructors
* Keyword this
* OOPS Principles

Understanding method arguments & return types

public static void add(int x, int y) {   
 int result = x + y;  
 System.out.println(result);  
}  
public static String sayHi(String name) {   
 return “Hi “+name;  
}  
public static void main(String[] args) {  
 add(20, 30);  
 String result = sayHi(“Sachin”);  
}

Classes & Objects

Class is a template / blueprint of an object

Object is a real world entity which is an instance created from the class, it will have initialized properties & methods

How to call a constructor from another constructor

There are two ways depending on the scenario

1. When you want to call constructor of same class use ‘this(args)’ keyword
2. When you want to call super-class constructor then use ‘super(args)’ keyword

Note: call to this() or super() should always be the first line of the constructor

class A() {  
 A() {   
 ….  
 }  
 A(String a, int b) {   
 this(a); // A(String)  
 ……  
 }  
 A(String a) {   
 this(); // A()  
 ….  
 }  
}

Enhanced for loop

It simplifies iterating arrays and collections by providing a simple syntax

for(type t : collection) {   
  
}

int arr = {…}

for(int x : arr) {   
 instead of arr[i] you can use ‘x’  
}

User[] users = ….

for(User u : users) {   
 instead of users[i] you can use ‘u’  
}

Day 3 Agenda

* Static keyword
* Inheritance - extends, super
* Polymorphism - overloading & overriding
* final keyword
* Abstraction - abstract class & interface

Static keyword:

Static members are accessible without creating objects, you can use class-name to access static members.

Static members are loaded before object is created which means at the time class loading

Inheritance:

Process of acquiring properties & behaviours of a class from another class, you use extends to achieve inheritance, then use super() to call the constructor of parent class from sub-class constructor, you can also use super(args) when you want to call argument constructor of the parent class

Note: super() must always be in the first line of the constructor

Note: this() can also be in the first line, but when you use this() or this(args) it calls constructor of the same class, however you can’t keep both this() or super() in the first line so you can keep either of the one, when you use this()/this(args) then super() will be ignored by compiler

Polymorphism

A method with many forms

* Overloading
* Overriding

Accessing subclass members from superclass reference

Person p1;  
p1 = new Person(…);  
p1 = new Employee(….);  
p1 = new Student(….)

If you have super-class reference then you can only access members present in superclass but not the members present in sub-class

p1.setName(..); // valid  
p1.setGender(..); // valid  
p1.display(); // valid  
p1.setId(..); // CTE  
p1.setSalary(..); // CTE  
p1.setGrade(..); // CTE  
p1.setRollNo(..); // CTE

Abstraction:

Hiding the complexity and showing only the necessary details to the end user so that the user could use the object easily

Abstraction divides the roles & responsibility of the end users so that they can concentrate on their work without thinking of others work, this makes code flexible to the changes so that when a code is modified at any one’s end then others code need not to be modified.

Note: Abstraction is achieved with method declarations or abstract methods or methods without body

// abstract method

abstract void m1();

// non-abstract method

void m2() {   
  
}

Abstraction is achieved in two ways

1. interface - 100% abstraction - all the methods are abstract by default
2. abstract class - partial abstraction - not all the methods are abstract some methods have body and some don’t.

Day 4 Agenda

* Interfaces
* Abstract classes & Methods
* Access Modifiers
* Layered Architecture

Layered architecture:

It is used to separate presentation logics, navigation logics & model logics in the application so that it can be easily maintained in the large scale applications, each layers are completely abstract with other layers so that any changes done in one layer doesn’t force to change other layers.

Layered architectures makes your code more reusable especially Model layer and also you must have a loosely coupled code, if you make changes in one place then the dependent codes need not to modify

ex: Controller depends on Model Layer object

There 3 different scenarios that controller can use to use the model layer object

Scenario1

Using model layer class as a reference & creating the object, it makes code tightly couple because if the model layer class changes then controller code should also be changed, if the controller is doing the same task then it doesn’t need to change its code if the implementations for such tasks are changing.

Controller Code

OracleDb db = new OralceDb();  
db.save(); db.find(); db.modify()

If the new implementation to perform same tasks needs to be used then you need to change the controller code and also the new implementation is not forced to implement same methods of OracleDb, hence we need to have an interface

Scenario2

interface DB { save(); find(); modify(); }

This interface can be implemented by any number of classes

class OracleDB implements DB { /\* implement all methods \*/ }

class MySqlDB implements DB { /\* implement all methods \*/ }

Controller Code

DB db = new OracleDB();   
db.save(); db.find(); db.modify()

In this scenario controller code is still modified if the new implementation object needs to be used i..e, DB db = new MySqlDB();

Scenario3

Abstract object creation of these dependencies using some design pattern like Factory Design pattern.

Factory Design pattern: This helps to returns the object by abstracting the object creation at the caller side

Implementing layered architecture with User, UserDao, ObjectFactory, ViewController, UserDaoArrayImpl

We need to create following list of Java files

1. User.java >> com.hsbc.model.beans
2. UserDao.java >> com.hsbc.model.dao
3. UserDaoArrayImpl.java >> com.hsbc.model.dao
4. ObjectFactory.java >> com.hsbc.utility
5. ViewController.java >> com.hsbc.ui

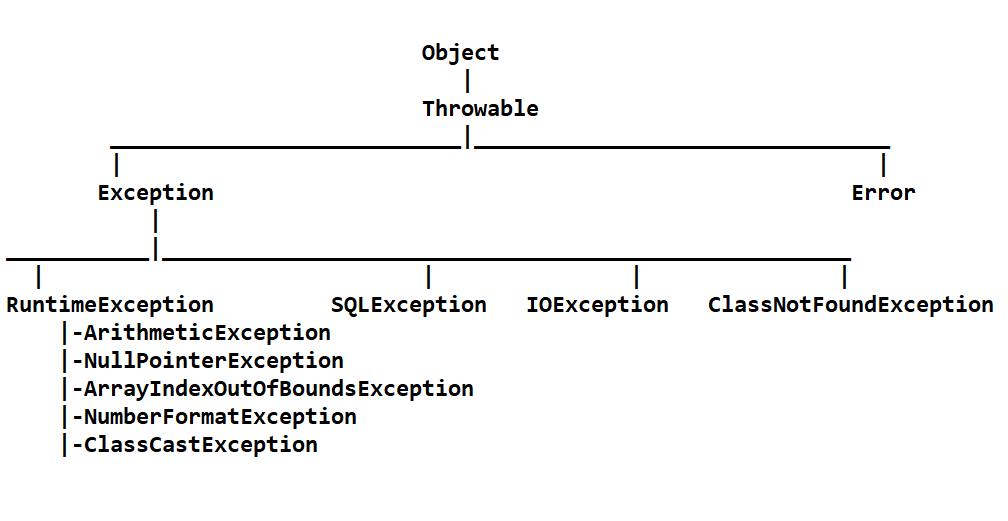
Exception Handling

Exceptions are runtime errors which abruptly terminate the application if in case not handled

There are 5 keywords in Exception Handling mechanism

1. try
2. catch
3. finally
4. throw
5. throws

Java has provided some predefined exceptions



Object class methods

1. toString(): Returns the object properties, but by default returns memory address
2. equals(Object): Returns boolean while comparing two objects, by default compare two objects’ address
3. hashCode(): Returns the hashCode of the object, by default returns object address in int

How you can override them

1. toString(): override it to return properties so that when any code prints the object it calls toString() automatically
2. equals(): Override it to compare two objects properties to identify duplicates ex: two employees id you can compare or two customers id you can compare
3. hashCode(): Override it to return an unique id for each object, so that it can be searched using that unique id

Note: Both equals() & hashCode() must be overridden together, if you want to represent uniqueness of any object when you store it in the Set or Map (Will be discussed in Collection Framework).

Multithreading

A program that can perform more than one task simultaneously .

In Java you can make more than methods to run simultaneously

Day 7

Collection Framework API’s

List

Set

Map

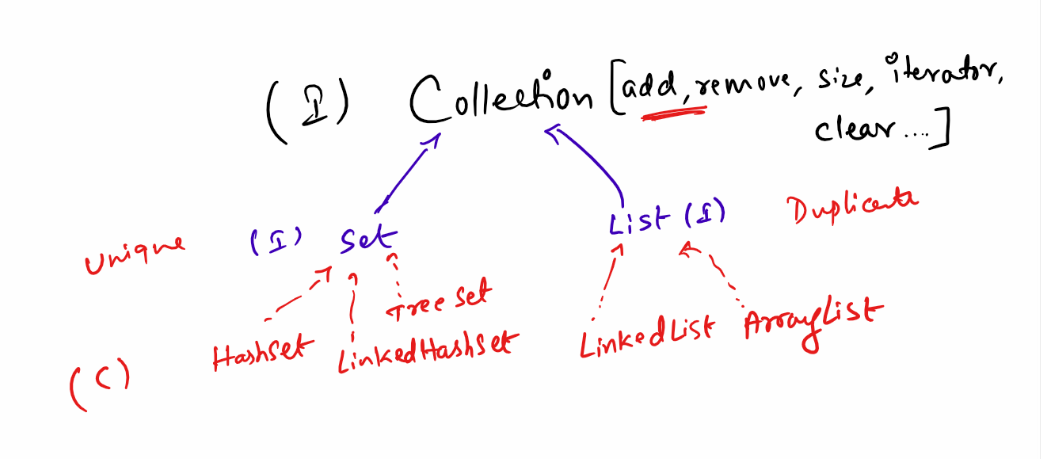
Comparable

Comparator

SQL

Collection Framework

Collection is a container which can store multiple objects and it is dynamic in nature, it provides many inbuilt methods to main the objects, it provides various implementations where you can have only unique elements, duplicate elements, sorted elements, elements in sequential order, elements in random order and so on



Collection is a root interface with methods like add(), remove(), size(), iterator(), clear() and so on, it has 2 sub-interfaces

1. List: Supports duplicates
2. Set: Supports uniqueness

List: It will have an implementations that maintains elements in sequential order, how you store the same order the elements are maintained, it supports duplicates

List has 2 implementations:

1. LinkedList: Stores elements in non-contiguous memory address, adding/removing the elements are faster compare to iteration, because it uses link between one element to another and each elements might be stored in non-contiguous memory address
2. ArrayList: Stores elements in contiguous memory address, adding/removing will shift the elements to different memory address to maintain contiguous memory address, retrieval is faster in array list compare to adding/removing elements

Set: It will have implementations that will support only unique elements, it has 3 implementations

1. HashSet: Retrieval is faster
2. LinkedHashSet: Maintains elements in insertion order
3. TreeSet: Maintains elements in sorted order

How Set identify duplicates

Set internally uses equals() & hashCode() method to identify the duplicates, these two methods are present in Object class.

equals(Object ob): It compares two objects

hashCode(): It returns the hashCode of the object that must be unique for every object, it is useful at the time of adding/removing/searching

Note: Primitive types equals() and hashCode() are internally used by its value, however for complex types we must override equals & hashCode

All the collection API’s use generics to mention the type of data it wants maintain

ex: If you want to maintain String then you can use Set<String>, if you want to maintain only int, then you must use Set<Integer> not Set<int> this is wrong, Integer is a class in Java that represents int type, if you want to maintain double then use List<Double> not List<double>, because Double is the class that represents double type, similarly we have corresponding wrappers classes for every primitives

Set<Customer> it maintains customer objects

List<Employee> it maintains employee objects

Comparable<T>: It is used by inbuilt classes who wants to sort the objects, it has a method called compareTo() which returns int value, if the value is +ve then it means the object is bigger, if the value is -ve then object is smaller, if value is 0 it means its neutral.

But Comparable<T> can’t sort multiple properties it can only sort one property at a time, if you want to have multiple sorting logics for different properties then you must use Comparator<T> it has a method compare(Object, Object).

Comparator<T> is an interface with one method compare(Object, Object), this needs to be implemented separately and use the object of the Comparator instance in the sorting class i.e., TreeSet.

TreeSet automatically sorts if the object implements Comparable, however you can use Comparator and make treeset to use different sorting technique by passing comparatorImplemenation as a argument.

How to sort List<T>

You can use a class called Collections that has a static method ‘sort(List, Comparator)’ Collections.sort(List<T>, comparatorImplementation)

Example

List<Employee> list = new ArrayList<Employee>();

list.add(…….); // assume you have some employee objects then you can sort it using

Collections.sort(list, comparatorInstance);

Map:

It helps you to maintain data in key value pairs, it is not part of the collection but works similar to collection.

Key must be unique, but value can have duplicates, Map has methods like

put(key, value): To store  
get(key): To retrieve   
remove(key): To remove  
keyset(): Returns all the keys in Set format

Map is an interface with 3 implementations

1. HashMap: Random order
2. TreeMap: Sorted order
3. LinkedHashMap: Insertion order

Database: It is used to store the data, it understand SQL (Structured Query Language).

SQL provides many keywords to maintain the data

It has

* DDL: create, alter, drop, truncate
* DML: insert, update, delete
* DQL/DRL: select

1. Create: It is used to create the table with columns

ex: create table emp (id integer primary key, name varchar(20), salary double);

1. Alter: It is used to modify the table structure after creation, it can be used to add/delete columns, add/remove primary key, change table name

ex: alter table emp add column desig varchar(15);

1. Truncate: It is used to delete all the records of the table, however the table wouldn’t be deleted

ex: truncate table emp;

1. Drop: It is used to drop the tables

ex: drop table emp;

1. insert: It is used to insert record to the table

ex: insert into emp values(1, ‘Raj’, 3300);

ex: insert into emp(id, name) values (2, ‘Vijay’);

1. update: It is used to update the records

ex: update emp set salary = 30000 where id = 2; # update the record with id = 2

ex: update emp set salary = 20000; # updates all the records

1. delete: It is used to delete the records

ex: delete from emp where id = 2; # delete the record with id = 2

ex: delete from emp # deletes all the records

1. select: It is used to fetch records

ex: select id, name from emp; # selects id & name of all the records

ex: select id, name from emp where id = 2; # selects id, name of emp id = 2

ex: select \* from emp; # selects all the columns of all the records

ex: select \* from emp where id = 2; # selects all the columns from id = 2

Generating sequence

*create sequence emp\_seq as int start with 201 increment by 1;*

insert into employee values(next value for emp\_seq, 'Dexter', '2002-05-20');

Transaction: It helps you to control when to commit or rollback, we need to do this by using connection.setAutocommit(false);

connection.rollback()

connection.commit()

JOINS:

When you want multiple tables records to be joined and show a single result then you can use SQL JOIN.

Syntax:

select column(s) from table1 join table2 on common\_column\_condition;

[or]

select columns(s) from table1 join table2 on common\_column\_condition where condition

Day 13

* Java 8 Features
* Strings in switch
* Improved exception handling
* Functional programming
* Lambda expressions
* Default methods
* Method reference
* Java 8 Streams

Java 8 Features

1. Functional Interface for functional programming
2. Lambda Expressions
3. Default & Static methods in Interface
4. LocalDate, LocalDateTime, LocalTime API’s
5. Stream API’s

Functional Programming:

It enables you to pass functions as an argument to the method instead of passing objects, it is achieved through functional interface & lambda expression.

Functional Interface: It is an interface with only one abstract method no matter how many static or default methods the interface has it is not counted

Lambda Expression: It is a simplified form of implementing the interface having only one abstract method, without writing class body, without writing method name, without having argument datatype and also no { } for method body if its one line implementation.

Changes in the interface from Java 8 onwards

1. Default methods in the interface which will have default implementations so that classes are not forced to implement, but it can be overridden
2. Static methods in the interface which is inherited to all the classes but it can’t be overridden, because its called using name of the interface/class

Note: Java has modified its old interfaces like Collection, Comparator, List and so on to have default & static methods

Changes in switch statement

Earlier switch used to take only int, char, enum, but Java 7 onwards it can also accept strings

String option = “price”;  
switch(option) {   
 case “id”: sort based on id; break;  
 case “price”: sort based on price; break;  
 default: sort based on name  
}

Changes in Exception Handling

* Auto-closing & flushing the resources like files, databases, you don’t have to use close() method at all, it is called as try-with resource closing feature

try (FileInputStream fis = new FileInputStream(); FileOutputStream fos = new FileOutputStream()) {  
 // no need of close() method invoking on fis & fos  
 // no need to call flush() on fos  
 }

* Multi-catch blocks to handle one or more exceptions

try { }catch(IOException | SQLException e) { }

Java 8 Streams

It is used to process the collection of data in a declarative way through lambda’s

like sorting, filtering, iterating can be done in very less lines of code.

You have two types of streams that helps to process collection

1. sequential stream: Uses only one thread
2. parallel stream: Uses multiple threads

Collection framework provides methods like

stream(): it returns a stream that is sequential

parallelStream(): it returns a stream that can be processed parallelly

forEach(Consumer<T>):

forEach method is a default method added in Collection, it accepts a functional interface called Consumer which has an abstract method void accept(T t);

Consumer<T> >> void accept(T t)

Corresponding Lambda for consumer will be (t) -> System.out.println(t);

list.forEach( ( t ) -> System.out.println(t));

stream(): It makes your collection items to be maintained in a stream i.e., like a memory where changes you make in the stream is not reflected in the original collection framework

stream has methods like filter(), sort(), map(), collect(), count() and so on

Most of the methods return a new stream like filter(), sort(), map(): they return new streams

filter(Predicate): It is used to iterate the items and filter out the items that matches to the Predicate.

Predicate<T>: is a functional interface with a method an abstract method boolean test(T t)

Lambda expression for Predicate<T>

(item) -> booleanExpression;

filter( ( item ) -> booleanExpression) // returns only the items that has true in Predicate

integers  
 .stream()  
 .filter(item -> item % 2 == 0)  
 .forEach(item -> System.out.println(item));

map(Function): It is used to take item from the stream and return a result which is a transformation of the same item.

Function is the interface that has a method apply(T t) that returns T

Create a Laptop class with properties like

* name : String
* price: double
* ramSize: int
* os: String
* 2 constructors, toString(), setters & getters

Create a Main class that will have a main method that will have a List<Laptop> with atleast

12 laptops of different name, price, ramSize & os

ex:   
- acer, dell, lenovo are laptops name,  
- 40000, 50000, 60000 are laptops price  
- 16, 8, 4, 32 are laptops ramSize  
- Windows, Mac, Linus are laptops os

Use streams on the List<Laptop> you have created and

* Create a List<Laptop> list1 that has only ramSize 8 & print it
* Create a List<Laptop> list2 that sorts all the laptops based on ramSize & print it
* Create a List<String> list3 that will only have laptop names & print it

Method Reference

It is used to reuse already implemented method as the implementation to the abstract method present in functional interface, the existing method signature must be same as the abstract method signature of the functional interface.

interface A {   
 int size();  
}  
String s = “hello”;

A a1 = s :: length;

a1.size();

Spring Framework

Framework: It is like a semi-implemented application which gives you all the common features every application needs, so that you can concentrate on the new requirements, some of the common features framework implements are:-

1. Type conversion - SQL to Java types, Java types to SQL, String to Numbers or LocalDate etc.
2. Design patterns like Factory pattern, singleton pattern, proxy pattern, prototype pattern, here framework provides these design patterns so that you don’t have to write them
3. Exception Handling - Framework handles all the resource level exceptions like SQLException, IOException and so on, so developers can concentrate on handling only custom exceptions
4. Provides you the structure where it separates infrastructure code or configuration code from the application code like datasource configurations you can separate it from application code

Spring Framework

Spring is a Java Framework which helps you to create various types of applications

1. Web applications
2. Cloud based applications
3. Enterprise applications

Spring provides many modules to develop applications

1. Spring Core or Spring context
2. Spring ReST
3. Spring Data Jpa
4. Spring Boot
5. Spring Cloud

Spring Core or Spring Context

It takes care of implementing all the design patterns like factory, singleton, prototype etc and helps to initialize all the dependencies required by the objects

Dependency Injection: It is a process where the dependencies are supplied to an object, these dependencies are created by spring container which is called as spring context also knows as IoC (Inversion of Control).

Types of dependency injection

1. setter injection: Initializes properties using setters
2. constructor injection: Initializes properties using constructors

setter injection is done using <property name = “propertyName” value = “..”>

constructor injection is done using

<constructor-arg index = “0” value =”…”>

Bean Scope

There are 4 scopes for a bean

* singleton: By default bean is singleton
* prototype
* session
* request

<bean> will by default allow spring container to create singleton instance, it means only one object of that class is configured with a <bean>.

<bean scope = “prototype”> it creates instance of the class for each getBean() method, it means multiple instances are created for the <bean>

<bean> scope session & request are used in web application, when scope is session it will create the instance of <bean scope = “session”> for a particular user/session,

<bean scope = “request”> here the bean is created for each request from the user

Note: If you have multiple <bean> configured for the same class then for each <bean> the instance is singleton

Annotation based configuration

Spring provides annotations to configure the beans & supply the dependencies, so that without using <bean> tag spring can create the object in the container, for that the class must have these annotations like

@Component  
@RestController  
@Controller  
@Service  
@Repository

These annotations must be written on top of the class  
@Component  
public class A { …. }

@RestController  
public class B { … }

Note: Though spring provides annotations to configure the beans we may still need xml configurations, i.e., when a user get a class from another user which don’t have any spring annotation configuration but still that class must be instantiated by spring container, then the user must configure it using XML.

Spring Jdbc

Spring provides one of the module called spring-jdbc which helps you to autoconfigure datasource informations in the XML, it provides few classes that helps you to perform database operations with spring API’s

1. DriverManagerDataSource: It has 4 properties like username, password, url, driverClassName - we need to configure this using <bean>
2. JdbcTemplate: It has a property called datasource of DriverManagerDataSource, it means it uses this property to auto-connect to DB & auto-close from the DB

Note: JdbcTemplate is the object which helps you to perform CRUD operations like Insert, Retrieve, Update & Delete

Libraries required for spring jdbc

1. derby-client
2. spring jdbc

Note: You need to mention them in the pom.xml

Spring Boot

application.properties

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

spring.datasource.username = admin

spring.datasource.password = 12345

spring.datasource.url = jdbc:derby://localhost:1527/..

.....

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

@SpringBootApplication

- Takes care of checking the libraries in the classpath & does

auto-configuration like component scanning, dependency injection,

initializing the spring container (no need to create

ApplicationContext object)

- spring-boot-starter-web

- configures web related environment

- embedded server

- component-scanning

- mvc configurations

- spring-boot-starter-data

- configures all the dependencies for database

- reads datasource properties in application.properties

- like DriverManagerDataSource will be created

then JdbcTemplate will be created & DriverManagerDataSource will be

injected, it means you don't have to write <bean> configurations

## no need to use this in spring-boot

ApplicationContext ctx = new ClasspathXml("xml", "xml", ...)

ctx.getBean("");

Through spring-boot we can get the application context using a functional

interface called CommandLineRunner, this implementation bean must be

registered in the spring container

CommandLineRunner -> void run(String... args);

...

@Bean // it registers the object that is retured by the method in the

// spring container

public CommandLineRunner load(ApplicationContext ctx) {

ctx.getBean("...")

}

This is similar to creating a reference to ApplicationContext

@Bean

public CommandLineRunner command(ApplicationContext ctx) {

return (args) -> {

ctx.getBean("someId");

}

}

[ OR ]

@Component

class Xyz implements CommandLineRunner {

@Autowired

ApplicationContext ctx;

public void run(String... args) {

ctx.getBean("someId");

}

}

Note: You don't need the ApplicationContext in Web MVC applications

application.properties: it is the default configuration file in spring boot, you can configure application related information’s like server properties, database properties, log properties, microservice properties and so on.

spring boot starter dev tools: It is the library used to auto-reload the application on changes

Things we avoid in spring boot

1. XML Configuration not required
2. Container Initialization not required
3. All the generic setups to develop spring application like configuring server, configuring set up for web development are avoided

Spring Rest

ReST stands for Representational State Transfer, it helps heterogenous applications to exchange the data in a common format like JSON, XML, CSV, Text, HTML, these formats are converted to the types the consumer understand.

Note: RESTful webservices is a architecture style, it means you can develop it in any technologies like C#, C++, Python, Javascript and so on.

@RestController  
@RequestMapping(“/users”)  
public class UsersAPI {   
 @GetMapping(“/test”)  
 public String greet() { return “…. “}  
 @GetMapping  
 public List<User> getUsers() { ….. return list; })  
 @PostMapping  
 public User store(….) { …. return user; }  
 @PostMapping(“/test”)  
 public User store2(….) { …. return user; }  
}

users/test >> POST calls store2()  
users/test >> GET calls greet()  
users >> POST calls store()  
users >> GET calls getUsers()

Spring AOP

AOP stands for Aspect Oriented Programming, it is used to add cross cutting logics across different methods without explicitly writing them in the code

Ex:  
// store() is a business logic which is called as join-point in AOP  
int store(User user) {   
 dao.store(user);   
}  
 // trackTime is the advice  
void trackTime() {   
 // tracking time each time store() is executed  
}  
AOP Terminologies

Aspects: It is a class that will have one or more advice  
Advice: A method that will have cross cutting concerns  
Joinpoint: A method that will have a business logic of the application [or] A method on which the advice is applied

Spring AOP provides some annotations to apply cross cutting logics

@Aspect // to configure the class to behave as Aspect  
@Component // registers this class as a bean in the container  
class Tracker {   
 @Before(“exection(public int com.hsbc.model.UserServiceImpl.save(..))”)  
 void m1() { …. }  
 @After(“execution(public void com.hsbc.model.UserServiceImpl.login(..))”)  
 void m2() { … }  
}

@Configuration & @Bean

It is used to configure the beans and register them in the spring container when the beans are instantiated by us.  
@Component, @Service, @Controller, @RestController, @Repository all these configures the class and registers as a bean in the spring container, but it will be instantiated by spring container  
@Bean is used on a method that returns the object that spring container needs to maintain, the method having this is called automatically, @Bean are written in a class that is configured as @Configuration, you can have any number of @Bean in a @Configuration class.  
class A { }   
class B { }  
  
@Configuration  
class Xyz {   
 @Bean  
 public A createA() { return new A(); } // object of A is maintained by spring container  
 @Bean  
 public B create() { return new B(); }// object of B is maintained by spring container  
}

What are microservices

Microservices are small independent services which can be developed and deployed independently.

A single application can be decomposed into multiple small services and can be deployed separately so that they all will be loosely coupled and doesn’t need to re-test, re-build & re-deploy all the services when any one service is changed

Advantages of following microservice architecture

* Technology independent, it means services of same application can be implemented in heterogenous technologies like Java, C#, Javascript, Python, PHP etc.
* Scaling the particular service on demand doesn’t force to scale other services
* Loosely coupled services hence each service can be developed & deployed in separate server, so that if any one service go down other services will still be available
* Releasing new feature to the application takes very less time, we can use agile methodology
* Microservices can communicate with other microservices through REST calls, because all the microservices are Rest Webservices itself

Disadvantages of microservices

* Costlier, because you need more resources
* People must have DevOps knowledge as well
* Return on Investment

Rules to follow while developing applications with microservice architecture

* Every microservice must be registered in a Service Discovery so that microservices can locate other microservices
* Failovers should be taken care so that it doesn’t affect other microservices which are communicating

How to develop microservice in spring

1. Firstly we need to create a service discovery which is possible from the library called Eureka Server, this should be a separate project
2. Secondly we need to create microservice and register it in the service discovery, to create microservice we have a library called Eureka Client, this library will automatically register to Eureka Server

Spring Microservice is very easy to implement because spring provides annotations which will do most of the jobs for microservices, some of the annotations are:-

@EnableEurkeaServer: Creates a service discovery & provides a dashboard so that users can see what all the services registered in it, better run Eureka Server in 8761 port only, because all the microservice by default looks for Eureka Server in 8761 to register

@EnableEurekaClient: This creates microservice & it automatically registers in Eureka Server looking it in port 8761 by default

RestTemplate: It is used to make REST calls to another webservice or microservice, since webservices/microservices will have URL & HTTP methods RestTemplate provides http methods that accept the URL of these services, some of the methods are:-

1. getForObject(url)
2. postForObject(url, data)
3. putForObject(url, data)
4. deleteForObject(url, data)

Note: RestTemplate we must create & register in spring container, so that it can be supplied to the dependencies

@Bean must be used on a method that returns RestTemplate object  
Create RestTemplate & registering to spring

@Bean  
@LoadBalanced // it resolves the location of microservice  
public RestTemplate createRestTemplate() {   
 return new RestTemplate();  
}

@Controller vs @RestController

|  |  |
| --- | --- |
| @Controller | @RestController |
| It is used only to develop MVC app, where your view is part of the project | It is used develop MVC based webservices, it accepts & sends data from all types of applications, view is not known |
| return type of the method must be ModelAndView, which will have model & view name | return type of the method must be ResponseEntity<T>, which will have status & data both |
| @Controller  @RequestMapping(“/v1”) public class SimpleController{..} | @RestController @RequestMapping(“/v2”) public class SimpleController{..} |

Node.js:

It is a runtime environment for Javascript, it can run javascript programs to perform backend operations

Earlier we had only browsers to run javascript programs so that we could only create front-end application from javascript,

Now we can run javascript at the backend using node.js so that we can create back-end applications from javascript

>> node filename.js

The above command runs the javascript file

Note: node.js provides javascript libraries available over the internet which helps to develop both front-end(React.js, Angular, Vue.js) & back-end applications(webservices, web apps, db-operations..)

The command used to download these libraries is ‘npm’ stands for Node Package Manager.

Angular Framework

Angular is framework used to develop single page applications, where you can see everything in one page

Angular it uses Typescript & HTML to develop the application, Angular team has provided a toolkit called ‘@angular/cli’ that helps you to create a ready to deploy angular application, this toolkit provides the environment that can

* compile your code i.e., typescript code to javascript
* run your code on an embedded server i.e., lite server provided by angular/cli
* modularity to store ts, html, css code separately
* commands to simplify the development process

Angular internally compiles typescript to javascript, it uses a command to do that internally which is ‘tsc’

>> tsc filename.ts

Note: You don’t have to type tsc, angular does this internally.

Steps

1. Installing @angular/cli : This is one time job
2. Create angular project using ‘ng’ command that you get after installing @angular/cli

Command to install angular/cli

npm install @angular/cli

[OR]

npm i @angular/cli

Once this installation is done, you can create a new angular project using

*ng new project-name*

Component: It is the visible part in the page, it is created using a decorator called @Component({….})

@Component: It accepts a javascript object that will have properties like selector, templateUrl, styleUrls,

@Component({selector: value, templateUrl: value, styleUrls: value})

AppModule: It represents all the features you create in your application like services, components, pipes, including your root component, this AppModule is loaded by angular when you use ng serve

Interpolation: It is used to display the property of a component to the user, it uses two flower braces i.e., {{ property }}

username = “Alex”;   
age = 35;  
In HTML you can use {{username}} and {{age}}

You can also access object properties

ex: obj = {name : “Bruce”, age : 33}

In HTML you can use {{obj.name}} and {{obj.age}}

Pipes:

It is to transform the data into another form like showing content in uppercase, lowercase, date in different formats, currency formats, json formats and so on

We have inbuilt pipes like:-

uppercase, lowercase, date, json, currency

How to use it  
{{ property | pipeName }} [ OR ] {{ property | pipeName: ‘args’ }}

{{ salary | currency }}, {{ salary | currency : ‘INR’ }}

{{ dob | date : ‘dd-MM-yyyy’ }}

NgFor: It is one of angular attribute which is used to iterate over the arrays, it is used as \*ngFor in the HTML

items = [2, 3, 1, 4, 5];

<div \*ngFor = “let i of items”> {{ i }} </div>

<tr \*ngFor = “let i of items”>   
 <td> {{ i }}</td>  
</tr>

Data Binding

This is to share the data from view template (HTML) to component class (TS) or vice versa.

1. Interpolation {{..}}: It shares data from Component to View
2. Event Binding (eventName): It shares data from View to Component
3. Property Binding [property]: It shares data from Component to View, it is used to apply value on DOM properties like hidden, disabled, ..

Note: In Interpolation you will only display the content, in Property Binding you will manipulate the HTML elements using its DOM properties

1. Two way data-binding [(ngModel)]: It shares data in both the directions i.e., V -> C & C -> V

Forms Module

It provides attributes that can store all the form data in an object called NgForm so that you don’t have to track each form input using template reference

<form #user = “ngForm” (ngSubmit) = “register(user.value)”>  
 Name <input type = “text” name = “un” ngModel>  
 Age <input type = “number” name = “age” ngModel>  
 Phone <input type = “number” name = “phone” ngModel>  
 <input type = “submit” value = “Submit” >  
</form>

user = {un = value, age = value, phone = value}

user.value instead of using tf1.value, tf2.value, tf3.value

You need to add FormsModule in the AppModule imports

@NgModule({   
 imports : [FormsModule,..]  
})

How to share data between the components i.e., from parent to child & child to parent component

There are two decorators we have in angular to communicate between the components which are:

1. @Input(): It is used to share data from parent to child component
2. @Output(): It is used to share data from child to parent component

Angular Services

Services are classes which will have reusable logics, these classes will have @Injectable() decorator, when a component needs the instance of Service, then angular will supply the instance to the component via constructor-injection.

Creating a Service  
@Injectable()  
export class UserService { ….. }

Using Service in a Component  
@Component ( { …., providers : [UserService], …})  
export class UserComponent {   
 constructor(private \_service : UserService) { … }  
}

Agenda

1. Calling back-endservices using HttpClient from Angular
2. Communicating with webservices / microservices to perform all the HTTP operations like post, get, put, delete from Angular using HttpClient

HttpClient & HttpClientModule:-

====================================================================

Angular application uses HttpClient to send request to webserivces written

in various languages like C#, Java, Python, Javascript, once request

is sent webservices sends response, these response are handled throug

Observable which is a container to main the response data.

HttpClient has methods like post, put, delete & get to call Rest Webservices

Webservices (C#)

Angular Application >>>>> Webservices (Java)

Webservices (Python)

Webservices (Javascript)

HttpClient.

post(url, data)

get(url) >> JSON >> Webservices

put(url, data)

delete(url, data)

All the methods of HttpClient returns Observable<any>

Observable<any>: it is a container which can store the response data from

the webservice once you send the request to the webservice

post(url, data): Observable<any>

get(url): Observable<any>

put(url, data): Observable<any>

delete(url, data): Observable<any>

---------- you need to use subscribe() method from Observable to get the

data------

httpClient.post(url).subscribe(callback);

callback is executed by passing the data from the observable

Ex: The angular service will have http client calls

public store(data): Observable<any> {

return httpClient.post(url, data);

}

Ex: Components would call these methods present in service

handleSubmit() {

this.\_service.store(data).subscribe(...);

}

Once you call subscribe the response data will be supplied in the argument

of subscribe()

HttpClientModule: It is a module that provides the object of HttpClient so that Angular can supply the HttpClient object, you need to add this module in the @NgModule, so that angular can supply HttpClient

@NgModule({ …  
 imports : [.. HttpClientModule, …]  
…})

If any one need HttpClient then they must use HttpClient as an argument in the constructor like below:-

constructor(private \_http : HttpClient) { }

Above constructor is written in Angular Service

Ensure your webservice is ready with atleast 3 or 4 Url’s

Create an angular service with a name user-db

>> ng g s user-db

Steps to call the webservice from the angular

* Add HttpClientModule to your AppModule
* Inject HttpClient to your Angular Service
* Create methods that return Observable<T>
* Call methods of HttpClient like post(), put(), delete(), get() and return them in the angular service methods
* From the components call the angular service methods & subscribe to the Observable

HttpClientModule & HttpClient are part of @angular/common/http

Observable is part of ‘rxjs’

Implementing storing the data

@PostMapping(consumes = MediaType.JSON\_VALUE)

ResponseEntity<T> store(@RequestBody User user) {   
 // calls save(user) method of Service Layer  
}

The above code accepts JSON from the Request and converts to Java object