Java Full Stack Development

Technologies

1. Java
2. SQL & JDBC
3. HTML, CSS & Javascript
4. Spring
5. React.js

Pre-requisites

* Basic computer knowledge

Software’s requirement

1. JDK - 11 or later
2. STS (Spring Tool Suite)/Eclipse
3. Derby/MySQL database
4. Node.js

Java

Java is a platform independent programming & Object oriented programming language

Datatypes

These are keywords used to represent the types for the data like int, float and so on.

Types of datatypes

1. Primitive datatypes - byte, short, int, long, float, double, char, boolean
2. Non-Primitive datatypes - Classes, Arrays

|  |  |  |
| --- | --- | --- |
| Primitive Datatypes | Size in bytes | Range |
| byte | 1 | -128 to +127 |
| short | 2 | -32768 to +32767 |
| int | 4 |  |
| long | 8 |  |
| float | 4 |  |
| double | 8 |  |
| char | 2 |  |
| boolean | 1 |  |

Creating variables with different types

int intValue = 25;   
char charValue = ‘B’;  
double doubleValue = 25.5;

Operators:

+, -, \*, /, %, =, <, >, <=, >=, ++, --, ==, !=

int a = 2 % 3; // a will be 2

int a = 3 % 2; // a will be 1

int b = 15;

int c = b++; // c will be 15, b will be 16

int d = 15;

int e = ++d; // e will be 16, d will be 16

== checks for equality, and returns true/false

int a = 10;

int b = 20;

boolean c = (a == b); // c will be false

Conditional Statements

1. if
2. if else
3. if else if else if …. else
4. switch

int a = 5;  
int b = 6;  
if( a <= b) {   
 // print hello  
}  
else {  
 // print bye  
}

int marks = 55;  
if(marks >= 80) {  
 System.out.println(“A+”);  
}  
else if (marks < 80 && marks >= 70) {  
 System.out.println(“A”);  
}   
else if(marks < 70 && marks >= 60) {  
 System.out.println(“B”);  
}  
else if(marks <60 && marks >= 50) {  
 System.out.println(“B-“);  
}  
else {  
 System.out.println(“C”);  
}

int choice = 2;  
switch(choice) {  
 case 1 : System.out.println(“Case 1”); break;  
 case 2 : System.out.println(“Case 2”); break;  
 case 3 : System.out.println(“Case 3”); break;  
 default : System.out.println(“Default”);   
}

Looping constructs

1. for loop
2. while loop
3. do while loop

While loop:

while(condition) {  
 // statements  
}

Do While loop:

do {  
 // statements  
} while(condition);

Naming Conventions

1. Don’t use underscores for anything except constants
2. Class names & File names must begin in capital letter, if there are more than one word use camel case i.e., HelloWorld
3. Variable names & Method names must begin in small letter, if there are more than one word use camel case
4. Package names must follow same convention as variable & method names, but you can use . operator to create sub-folder

String methods

String is an inbuilt class in Java, it has various methods which you can access on a string variable

ex:   
String username = “Alex”;  
username.toUpperCase(); // ALEX

username.toLowerCase(); // alex  
username.length(); // 4  
username.concat(“ander”); // Alexander  
username.charAt(2); // e

Class & Objects

classes can have variables, methods & constructors

objects are the instances of the classes which are created from the constructors

A.java  
class A {   
 int x = 10;  
 void display() { … }  
}  
A a1 = new A();  
a1.display();  
a1.x = 50;

Day 2 agenda

1. classes & objects
2. constructors
3. static members
4. this keyword
5. overloading
6. passing primitives & objects to the methods
7. array of objects
8. OOPS concepts
   1. Encapsulation - setters & getters
   2. Inheritance
9. Java 11 new methods in String class

Best practices:

classes - start with uppercase and use camel case

methods & variables - start with lowercase and use camel case

package names - same like methods & variables

indentation - keep block of code in the same level

ex:  
// without indentation  
class A {   
int x = 10;   
int y = 20;  
}  
// with indentation

class A {   
 int x = 10;  
 int y = 20;  
}

Constructor overloading:

more than constructors in the same class

class A {   
 A() { }   
 A(int a, int b) { … }  
 A(int a, int b, int c, byte d) { .. }  
}

A a1 = new A(20, 50);   
A a2 = new A(40, 50, 60, 70); // compilation error

this keyword:

It refers to the current object, you can use this keyword to differentiate between instance variable and parameter variable  
class A {   
 int x;  
 int y;  
 A(int x, int y) {   
 this.x = x;  
 this.y = y;  
 }  
}

Benefits of using this keyword

1. to differentiate between instance variables & parameter variables
2. to call the constructor from another constructor of the same class to reuse the code

Static members

These can be accessed without creating the object

static variables: These are the properties which are shared among all the objects i.e., single copy of static variables are shared among all the objects

static methods: These are the methods which you can access directly by the class names, if any methods doesn’t want to access instance members can be made static

ex: public static void main(String[] args) : main method is static because JVM needs to call main without creating the object of the main() method class

Java Memory management

There will be two types of memory

1. stack - keeps all the classes, local variables, non-static methods, static methods & variables
2. heap - keeps all the objects

Working: Firstly class will be loaded along with that all of its static members will be loaded, hence you can access static members from the class name, when object is created all of its non-static members will be loaded, hence through non-static methods you can access static members also, however you can’t access non-static members in the static methods directly.

Note: You can’t use this & super keyword also in the static methods, but you can use them in non-static methods

OOPs principles

1. Encapsulation
2. Inheritance
3. Abstraction
4. Polymorphism

Encapsulation: Wrapping up of data and methods in a single unit(class), and controlling the data using private and accessing them only through public methods (setters & getters)

Setter methods: they must accept arguments and return void

Getter methods: they don’t accept arguments but return the data

Note: variables of the class must be private

Note: setters & getters must be public

Creating array

type[] variable = new type[size]   
or  
type[] variable = { value, value, value }

The type can be any datatype like int, float, String, Employee, Student

String[] strings = { “hello”, “bye”, “thankyou” } // string array

Student[] students = { new Student(…), new Student(…), new Student(…) } // student array

Student[] students = new Student[5]; // 5 blocks are allocated for student objects  
In the above statement each block will have null by default you can assign value to each block using  
students[0] = new Student(…..);  
students[1] = new Student(….);

Scanner class:

It is used to take input from the keyboard, it provides various methods like nextInt(), nextFloat(), next(), nextLine(), nextShort() and so on

import java.util.Scanner;

Note: java.lang classes are automatically imported ex: System, String, Object, Thread and so on

New methods that are added in String class in JDK 11

JDK - Java Development Kit - It gives JRE also  
JRE - Java Runtime Environment, it will have JVM, JIT, Class loader, Byte code verifier  
JVM - Java Virtual Machine  
JIT - Just In Time compiler

New methods of String class: isBlank(), stripLeading(), stripTrailing(), strip(), lines()

String s = “ “; // s.length() returns 2 because it has two white spaces  
s.isBlank(); // returns true if string is empty or has only white spaces  
String s2 = “ Hello World “;  
s2.stripLeading(); // removes extra white spaces coming at the end  
s2.stripTrailing(); // removes extra white spaces in the beginning  
s2.strip(); // it’s the combination of stripLeading() & stripTrailing()  
  
lines(): It creates a stream of strings separated by new lines (we can discuss this in Java Streams)

Method overloading

Same method names with different signature like different parameters, different types of parameters

Inheritance: Acquiring properties and behaviors from parent class to child class

What are inherited to the child class

* parent class public & protected members will be inherited

Whare are not inherited to the child class

* parent class private members will not be inherited
* constructors of parent classes are not inherited

Important points:

* Every subclass constructors by default calls default constructor of their parent class
* If a class is not extending any class then by default it extends Object (root class for all)
* Subclass constructors can use super(args) when it has to call the Super class argument constructor
* If super class doesn’t have the default constructor then sub-class might give compilation error has its constructor will be calling the default constructor of the super class, to resolve the error either you create default constructor in the super class or use super(args) in the subclass constructor to call the super class constructor which are present

Method Overriding: Re-implementing the inherited methods in the sub-class to have sub-class related logics

Another use of overriding

* You can achieve runtime polymorphism through method overriding

Note: Method overloading is for compile time polymorphism

Polymorphism: A method with many forms, you can use a single method and perform multiple actions

How to achieve runtime polymorphism

In order to make a single method to run all the overridden methods we need to follow two guidelines

* Methods must be overridden
* You must use super class reference variable to accept all the subclass objects including super class object & call the overridden methods from the super class reference

ex:   
Person p;  
p = new Person(…); p.display();  
p = new Employee(…); p.display();  
p = new Student(…); p.display();  
[ or ]  
void testPoly(Person p) {   
 p.display();  
}

You can’t achieve runtime polymorphism using subclass references

Employee e;  
e = new Employee(….) // ok  
e = new Person(…) // error  
e = new Student(…) // error

long x = 25;  
int y = x; // error  
  
int z = 55;  
long z2 = z; // ok

Object class: It is the root class hence it can accept all the objects

Object obj = new Employee(…) // ok

All the methods of Object class are inherited to every class, some of the methods are

toString()  
equals()  
hashCode()

toString(): It is automatically called when you print the object, it returns the representation of the object in a string format

* by default toString() returns package name and the class name: ex: return com.hsbc.Employee
* You can override toString() in the form you want: ex:   
  return “Id = “+id+”, Name = “+getName()+”, Salary = “+salary

toString() method is overridden in many inbuilt classes almost like 80 to 90% of the classes have overridden toString() to represent their object in an user understandable string format.

Ex:

LocalDate which is a replacement for old Date class represents the date  
String returns the content of the string object  
ArrayList, LinkedList, HashSet and many more classes have overridden toString to represent their object

Create a main class with the name TestToString and create below objects

LocalDate date1 = LocalDate.now(); // import from java.time  
LocalTime time1 = LocalTime.now(); // import from java.time  
LocalDate date2 = LocalDate.of(1970, 10, 25); // yyyy, MM, dd   
System.out.println(date1);  
System.out.println(time1);  
System.out.println(date2);

Note: toString() is overridden in LocalDate to represent the date in the format yyyy-MM-dd, LocalTime toString() is in the format hh:mm:ss.MS

Final keyword: It is used when something you don’t want to change, you can use it on class, methods, instance variables, static variables, local variables

final int x = 10; // constant

x = 25; // error

Using final on class: You can’t extend the final class, that is the last class in the hierarchy, use final class when you want no one to extend id  
ex:   
final class A { } // now no one can extend class A

Using final on methods: You can’t override the final methods  
class B {   
 final void demo() { … }   
}

Using final on variables: You can’t change the final variable value

Note: final variables must be initialized at the time of declaration or in the constructor if its instance variable, once its initialized then you can’t change the value

class X {   
 final int i = 10;  
 final int j; // this must be initialized either here or in the constructor  
 X(int j) { this.j = j }  
}  
X x1 = new X(55); x1 = {i: 10, j: 55}  
X x2 = new X(66); x2 = {i: 10, j:66}

Abstraction: Hiding the complexity of the object and showing the necessary details, so that user can use the object easily

In abstraction you will

* hide method implementations
* show method signature

End users should be using only method signature to call, they don’t have to know how it is implemented

Abstraction is to understand what method does, instead of understanding how it is implemented

How to achieve abstraction

There are two ways to achieve abstraction

1. interface: 100% abstraction - it provides only method declaration & no implementation
2. abstract class: 0 to 100% abstraction - it provides both method declarations & implementations

Interface:

* It is a kind of class which can have only abstract methods (method declarations or methods without body)
* It acts like a contract between two code/two programs
* It can have constants along with the abstract methods

Ex: consider Car Driver using Accelerator, Break, Clutch which interacts with the Vehicle engine, Car Driver can be considered as one program, car engine is another program, Accelerator, Break, Clutch as interfaces, every driver knows how these interface work accordingly every engine has to be designed

Note: The real use of an interface is to provide abstraction so that any changes made at one end doesn’t force to change at the other end.

How to create interface

interface X {  
 // by default methods are public & abstract  
 void add(int x, int y); // public abstract void add(int x, int y);  
 void sub(int x, int y); // public abstract void sub(int x, int y);  
}  
abstract: it is a keyword which you can use to declare methods & create abstract class  
Interfaces must be implemented by the class, a class can implement one or more interfaces, if it does then it must provide body for all the methods mandatorily.

Suppose you have interface Y as below  
interface Y {   
 void div(int x, int y);  
 void mul(int x, int y);  
}

Now you can create a class to implement X or Y or both

class Calculator implements X, Y {   
 // should implement 4 methods : add, sub, div, mul  
}  
class Calculator1 implements X {   
 // should implement 2 methods: add, sub  
}  
class Calculator2 implements Y {   
 // should implement 2 methods: div, mul  
}

Here you can consider

* Calculator, Calculator1, Calculator2 is like a mechanical engineer who implements the contract like an engine implementation - add(), sub(), mul() & div() implementation
* X & Y are the interfaces which has the contract methods - add(), sub(), mul() & div() method declarations
* Driver can be considered as some code that would access the interface X & Y methods like add(), sub(), mul() & div()

Interfaces can also be used to for access restriction on some methods based on the role

IRCTC -> regular user -> bookTicket( )  
 -> agent user -> bookTicket( )  
 modifyTicket( )

interface RegularUser { bookTicket(); }  
interface AgentUser extends RegularUser{ modifyTicket(); }   
  
RegularUserUI >> RegularUser >> bookTicket();  
AgentUserUI >> AgentUser >> bookTicket() & modifyTicket()

Important points on interfaces

* Interfaces will not have constructors
* You cannot create object on interface
* You can always declare a reference variable of an interface and pass its implementation object to it
* An interface can have one ore more implementations in real-time
* The methods of interface will be polymorphic based on the implementations

interface Movement { void accelerator(); }  
class Brezza implements Movement { public void accelerator() { … } }  
class Alto implements Movement { public void accelerator() { … } }   
class Swift implements Movement { public void accelerator() { … } }  
  
Movement move; // a reference variable not an object  
move = new Brezza(); move.accelerator();  
move = new Alto move.accelerator();  
move = new Swift(); move.accelerator();

abstract class:

It is used when you want to have both method declarations and implementations

abstract class Car {   
 void printWheels() { System.out.println(“4 wheels”); }  
 abstract void mileage();  
}  
class Alto extends Car {   
 void mileage() { // 25kmpl }  
}  
class Alcazar extends Car {  
 void mileage() { // 8kmpl }  
}

Real-time example:

Assume you want a data-structure to store & display the data, you need to store in sorted order, random order, insertion order and display the stored data, in such case you can create an abstract class that implements display() however store() will be abstract as it will be specified by the subclasses like SortedStoring, RandomStoring, SequenceStoring

Important points on the abstract class

* You can have constructors
* But you cannot create object of an abstract class
* abstract keyword is mandatory to declare methods & classes, unlike in interface which is automatically considered abstract

|  |  |
| --- | --- |
| Interface | Abstract class |
| Methods are by default abstract & public | We must explicitly use abstract keyword for abstract methods |
| Variables are by default constants, they are by default public, static & final | We must use final, static if we need |
| No constructors | Will have constructors |
| Can’t create object of an interface | Can’t create object of an abstract class |
| Used as a reference in the client code as it doesn’t change & also code wouldn’t exposed | Must not be used as a reference in the client code as it might expose the logics to the clients |

Note: abstract class constructors will be useful to initialize the properties of it, so that sub-class can reuse those constructors using super(arguments)

Problem:

Create everything in the separate files

1. Create an abstract class Car with an abstract method mileage() and a non-abstract method wheels() which prints 4 wheels
2. Create a sub-class Alto that extends Car and implements mileage() to print 25kmpl
3. Create another sub-class Alcazar that extends Car and implements milage to print 8kmpl
4. Create a main class with Car as a reference, assign Alto() object & call mileage() and wheels(), then assign Alcazar() object & call milage() and wheels() from the same Car reference (check the polymorphic behavior of mileage() and wheels)

Day 4 agenda

* More about the interfaces
* Layered architecture
* Access specifiers - private, protected, public
* Factory pattern
* equals & hashCode methods
* Exception Handling (if possible)

Access Modifiers or Specifiers

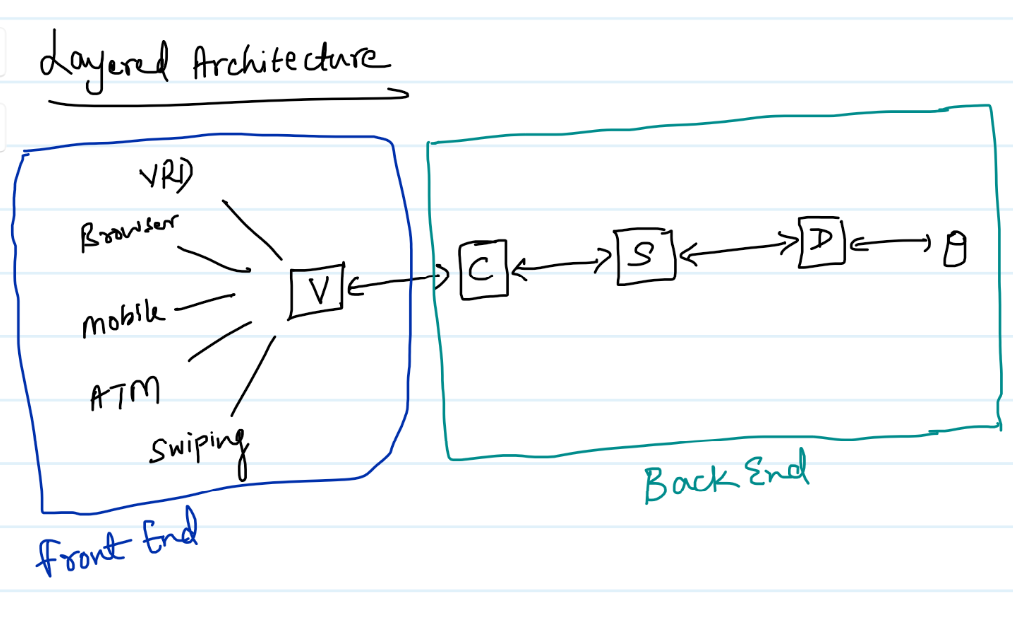
These are set of keywords that specify the visibility of classes & their members, there 4 access specifiers

1. private: visible only within the class
2. no keyword: it is called as default scope, visible only within the package
3. protected: visible within the package & outside the package only to the sub-class
4. public: visible to all

Layered architecture

It is used to categorize the code in different layers so that the application will be loosely coupled for the changes, there will be multiple layers where each layers does specific job

1. View: It will have the logics of User Interface like taking input from the user, showing the data to the user on browser, mobile, atm monitor, swiping device and so on
2. Controller: It will have the logics to accept the request from the UI and generate the response to the UI, it interacts with the service layer i.e., it calls the service layer methods
3. Service: It will have the business logics which interacts with DAO layer by access their methods
4. DAO (Data Access Object): it will have database logics to perform CRUD operations (Create, Retrieve, Update, Delete)



Note:

1. Since we don’t have database at present we can use only Service layer, once we learn SQL & JDBC we can use DAO layer, till that time Service layer itself will maintain the data
2. Since we can’t use any other views like Browser, Mobiles, ATM, Swiping machine, we can consider main method itself as View and Controller, hence using Scanner and System.out.print() must be in main itself and also main acts like controller to access the service layer methods

How to create a loosely coupled application so that one layer can interact with another layer and also guarantee that if any one layer changes other layers doesn’t need to change

We need to abstract every layer that interacts with another layer, it is possible with the help of

1. interface: a contract between the layers
2. factory pattern: takes care of creating implementation object i.e., interface implementation

Controller code must use a factory pattern to get the object of Service layer

Service code must use a factory pattern to get the object of DAO layer

Note: View layer doesn’t use any interface or factory pattern because it will be implemented with client & server approach, so that it acts like a client who uses URL to send the request.

Since we are considering main method like a View & Controller we will be using factory pattern to access the service layer

Creating a layered architecture to maintain the employee objects in the array

1. interface in the service layer
2. implementation of the service layer interface that maintains the employee objects in the array
3. factory pattern that creates the object of the service layer implementation
4. main method that acts as view and controller uses factory pattern to get the service layer object
5. since we are maintaining employee objects we need Employee class which will have properties, setters & getters - these classes are also called as Java Beans

Java Beans: They are the reusable classes with private properties, setters & getters, toString

Steps:

1. Create Employee class
2. Create EmployeeService interface
3. Create EmloyeeServiceArrayImpl that implements EmployeeService
4. Create ObjectFactory that creates object
5. Create Main class which acts like View & Controller

Note: We need to use Scanner & System.out.println() only in main method

Object class method

* equals
* hashCode

These two methods also you can override which part of Object class

equals method: it is to compare two objects

hashCode: It returns object hash code which is a memory address

Note: These two methods plays an important role to identify the duplicates of objects, which you can understand in Collection Framework

Activity:

Create a Service Implementation for EmployeeService that stores unique employee objects, the store method must add employee objects in the array only if the same object is not present in the array, if present it must not add that object

Create a separate class name EmployeeServiceUniqueImpl the implements EmployeeService, store() method should have logic of storing employee object by checking if that object exists, if no it stores else return -1 without storing

getAll() method returns all the employees

In ObjectFactory replace the EmployeeServiceeArrayImpl object with EmployeeServiceUniqueImp

Note: No need to change the main method, it must access store() & getAll() on the 2nd implementation

Day 5 Agenda

Exception Handling

Exceptions are runtime errors which needs to be handled else program abnormally terminates, there are 5 keywords which are used in the exception handling mechanism

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

try: It will have a block of code which might cause exception, ex: file read/write, database operations, accessing remote service and so on

catch: it will come after try block to handle the exception caused in the try block, you can have more than one catch blocks after the try block

finally: It is also a block which is optional to keep, it is executed definitely either exception handled or not i.e., in all scenarios it will be executed

* scenario1: no exception at all
* scenario2: exception occurred and handled
* secnario3: exception occurred but not handled

Note: finally can appear after try or after catch

Note: you can have following combinations

1. try - finally
2. try - catch - finally
3. try - catch - catch … finally
4. try - catch - catch …
5. try - catch

throws: It is used to propagate the exceptions from one code to another code when a code doesn’t know how to handle the exceptions, it is written beside the methods signature

i.e.,   
Service layer  
public boolean debit( double amount ) throws Exception {  
 // check for the balance if amount > balance generate exception  
 // check the number of transactions done in a day  
 // debit the amount from the account  
}  
Controller layer  
performDebit(double amount) {  
 try {   
 debit(amount);   
 return “Transaction Success”;   
 }   
 catch(Exception e) {   
 return “Transaction Failed”   
 }  
}

throw: It is a keyword used to manually generate the exception

ex: throw new ExceptionName(arguments);

Basically you will generate exceptions on certain conditions like

if(amount > balance) throw new InsufficientBalanceException(“some error message”);

Note: InsufficientBalanceException is one of the exception class we must create

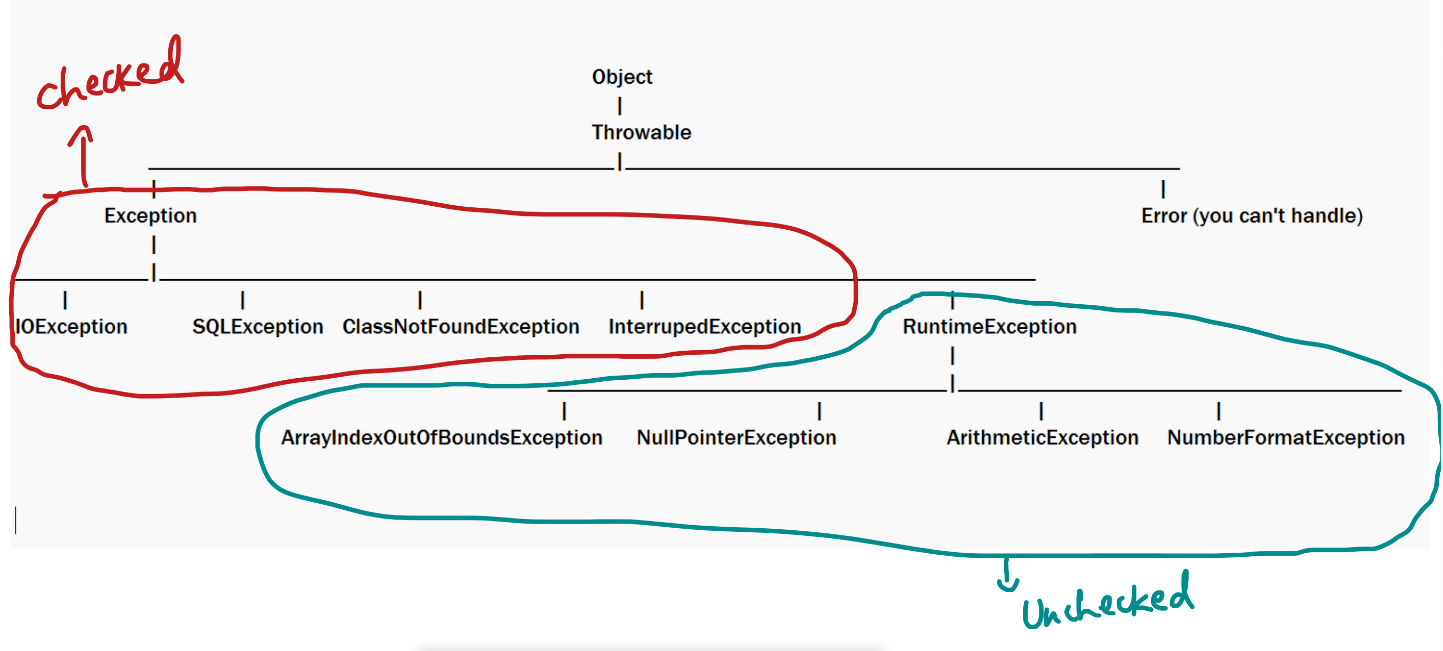
if(!found) throw new EmployeeNotFoundException(“Sorry employee with an id not found”);

Exception types

There are two types of exceptions

1. Checked exceptions: Which must be handled at the time of compilation,
2. Unchecked exceptions: They are ignored by the compilers

Exception hierarchy



Checked Exceptions need to be handled because they are not under developers control to avoid (ex: file not having write permission, database is down)

Unchecked exceptions are ignored by compilers however if they occur program will still terminate abnormally, but you can avoid these exceptions (ex: using for-each to iterate array to avoid array index out of bounds exception, using reference!=null to avoid null pointer exception)

Custom exceptions

Whenever we want to create our own exception classes for our project requirement we can create custom exceptions either by extending (preferred are Exception or RuntimException) or any other exception classes  
  
// checked exception - need to be handled  
class EmployeeNotFoundException extends Exception { }

// unchecked exception - ignored by the compiler  
class EmployeeNotFoundException extends RuntimeException { }

Writing multiple catches

When you write catch you must handle specific exceptions and then give chance to generic exceptions, i.e., handle all sub-class exceptions first and then use catch with super-class exception

try { … }  
catch(RuntimeException e) { }   
catch(SQLException e) { }   
catch(Exception e) { }   
Note: Above code will work fine

try { }   
catch(RuntimeException e) { }  
catch(ArithmeticException e) { }  
Note: Above code will give compilation error as ArithmeticException is already handled by RuntimeException

Rules of exception handling while overriding

1. If super class method doesn't throw any checked exception then sub-class must not throw any checked exceptions

2. If super class method throws any checked exception then sub-class can throw the same checked exception or sub-class of that checked exception or can ignore the throws, but it must not throw the super-class checked exception(i.e., if super class method throws SQLException, then subclass method must not use throws Exception)

Adding custom exceptions in our case-study

public class EmployeeNotFoundException extends Exception {   
 public EmployeeNotFoundException(String err) { super(err) }   
 public EmployeeNotFoundException() { }   
}   
  
Employee getEmployee(int id) throws EmployeeNotFoundException {   
 ….  
 if(!found) throw new EmployeeNotFoundException(id+” not found”);  
 else return employeeObject;  
}  
  
Caller

try {   
 Employee e = getEmployee(200);  
 System.out.println( e ); // success response  
} catch(EmployeeNotFoundException e) {   
 System.err.println(e.getMessage() ); // error response  
}

Steps

1. create EmployeeNotFoundException that extends Exception in com.hsbc.exception package
2. create getEmployee(int id) method in EmployeeService interface that returns Employee and throws EmployeeNotFoundException
3. implement the getEmployee(int id) in the implementation class
4. create a switch case to get the employee based on id in the ViewController

Exception re-throwing

It is to handle an exception in a catch and throwing a different exception to the caller

try {   
 // some exception occurs  
} catch(SomeException e) {   
 // rethrowing a different exception  
 throw new DifferentException(arguments);  
}

ex:

public int createAccount(…) throws AccountCreationFailedException {   
try {  
 // accessing the db layer  
} catch(Exception e) {   
 throw new AccountCreationFailedException(“some error message”);  
}  
}

Activity:

1. Using the same case-study create another implementation for the EmployeeService which stores only unique employee objects, if the employee id is duplicate throw EmployeeCreationException
2. We will be having now 2 versions of EmployeeService implementation
   1. EmployeeServiceArrayImpl: stores duplicates also
   2. EmployeeServiceUniqueImpl: stores only unique employee objects
3. From the factory pattern create both the EmployeeService implementation objects and return each object based on the parameter, like for input 1 return 1st version & for input 2 return 2nd version object
4. From the main method take user input for ObjectFactory to get the EmployeeService implementation object, then perform the operation

Day 6 agenda

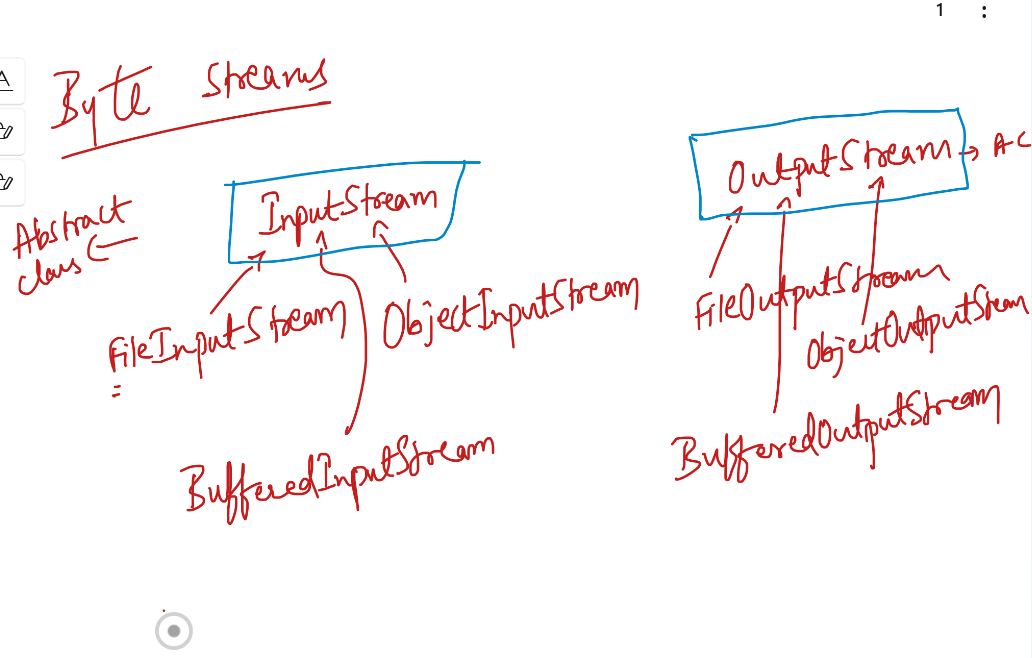
IO Streams  
Serialization  
Multithreading

IO Streams: Deals with reading & writing of data into various source & destination like file, buffer, console, browser and so on.

Streams are nothing but flow of data from source to destination, there are two types of streams

1. Byte Streams - binary data(.png, .img, .mp4 and so on)
2. Character Streams - text data(.txt, .java, pdf, docx)

Byte Streams hierarchy

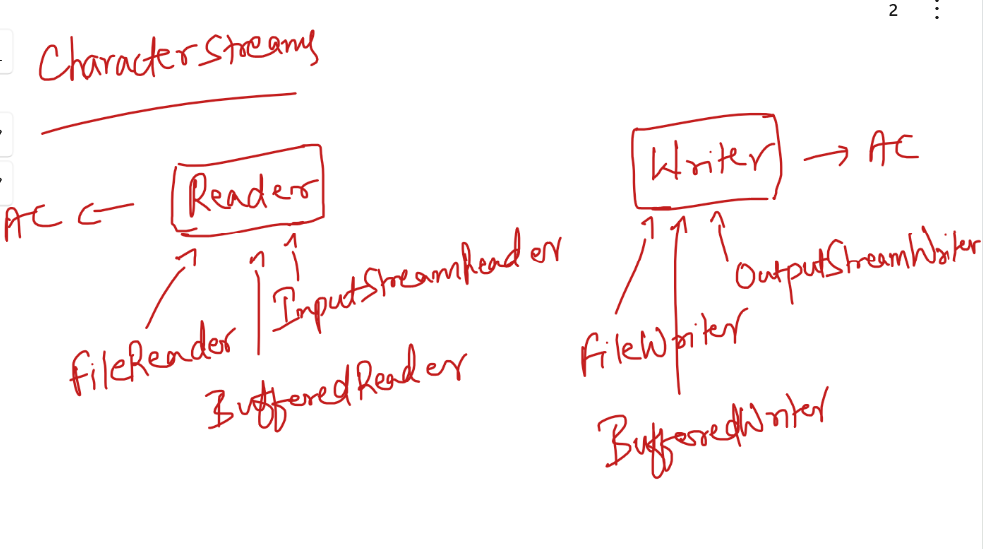


FileInputStream & FileOutputStream: These classes can read & write data from/to the file

BufferedInputStream & BufferedOutputStream: These classes can read & write data form/to the memory or buffer

ObjectInputStream & ObjectOutputStream: These classes are not source/destination, they are used to read/write complex data either from file/buffer

Character Streams hierarchy



Character Streams class deal with text data

FileReader & FileWriter: Reads/Writes text data from/to file

BufferedReader & BufferedWriter: Reads/Writes text data from/to buffer

InputStreamReader: It acts like a bridge between byte stream & character stream, it converts binary text to character text i.e., 65 to A

Ex: System.in reads your keyboard characters in ascii format & Scanner class uses InputStreamReader to convert the ascii format to character format, i.e., if you hit A then System.in reads as 65 & Scanner converts that 65 to A using InputStreamReader

OutputStreamWriter: It converts character stream to byte stream, you can use this when you want to convert character text to ascii, like A to 65

Note: All these classes are present in java.io package

Read & Writing text data

FileWriter fileWriter = new FileWriter(“demo.txt”);  
String data = scan.next();  
fileWriter.writeString(data) or fileWriter.write(data) // if demo.txt doesn’t exist it creates it.

FileReader fileReader = new FileReader(“demo.txt”);   
fileReader.readString(); or fileWriter.read() // if demo.txt doesn’t exist it throws exception

Note: Many of the IO class methods & constructors throw checked exceptions ex: read() & write() throw IOException, FileWriter & FileReader constructors throw FileNotFoundException

Note: FileNotFoundException is a sub-class of IOException & IOException is a sub-class of Exception

try with resource closing and flushing syntax

This feature automatically closes & flushes so that developer need not to worry about the resource closing, it works only with the classes which are implementing some interfaces like Closeable, Flushable, AutoCloseable

ex: FileInputStream, FileOutputStream, FileReader, FileWriter are all implementing such interfaces hence you can use try-with resource closing syntax

Earlier:

try {   
 FileReader reader = new FileReader(“….”);  
 …  
 reader.close();   
}

Now:

try ( FileReader reader = new FileReader; Scanner scan = new Scanner(System.in); …) {  
 // no need to call reader.close() & scan.close();  
}

Object Serialization & Deserialization:

Serialization: Storing the object in a file

Deserialization: Reading the object from a file

Any object you want to serialize must implement a marker interface “Serializable”, this makes the object a Serializable type

Marker interface: These interfaces doesn’t have any methods, but they add one extra type to the object, you don’t need to implement any methods at all

class A implements Serializable { } // You can store the object of A into the File

class A { } // You can’t store the object of A into the File

There are many marker interfaces in Java which marks objects for different operations

1. Serializable: It is to store the objects
2. Remote: It is to create remote objects which are accessible over the internet
3. Cloneable: It is to clone the objects

How to Serialize the objects

Step1: implement Serializable  
public class Employee implements Serializable { }   
Step2: Create Object of Employee & pass it to the ObjectOutputStream.writeObject(obj) method that writes the object to the file  
Step3: Use ObjectInputStream.readObject() to read the object from the file

Note:

1. The object you want to serialize will serialize only the object properties
2. If any instance properties you don’t want to serialize you can make it transient   
   i.e., transient int age; // wouldn’t be serialized
3. static properties are not serialized as they are not part of the Object

Serialization Program

class Employee implements Serializable { … }  
  
FileOutputStream fos = new FileOutputStream(“emp.ser”);  
ObjectOutputStream oos = new ObjectOutputStream(fos);  
Employee employee = new Employee(……);  
oos.writeObject(employee); // writeObject converts Employee to Object while writing

Deserialization Program

FileInputStream fis = new FileInputStream(“emp.ser”);  
ObjectInputStream ois = new ObjectInputStream(fis);  
Employee employee = (Employee) ois.readObject();

// returns Object as it was written as Object, we need to convert to Employee to access Employee class members

Activity:

In the same employee case-study create a new implementation for EmployeeService that can perform 3 operations as explained:-

1. store(Employee employee): store the object in the array & serialize the array into the file
2. getAll(): read the array from the file and return it
3. getEmployee(int id): call getAll() and find the employee based on id and return the employee object or throw the EmployeeNotFoundException

In the factory pattern create the object of this new implementation on a particular input

From the main method pass the input to get the object of EmployeeService implementation and perform the CRUD operations

Day 7 Agenda

Files.writeString() & Files.readString()

Multithreading

Collection Framework

New class “Files” in java.nio

Java has introduced Files class in a new IO package java.nio, it has methods to easily write & read Strings

Firstly new need specify the Path, which is done using an inbuilt class Paths and its static methods get

Files.writeString(path, content, options): It is used to write the string content

Files.readString(path): it is used to read the entire string content from the file.

Note: It works only if you have Java 11 or higher version

Files.writeString( Paths.get(“filename.txt”), stringContent, StandarOptions.APPEND) is used to write the string content

Parameters are:

Paths.get(“….”) is to locate the file  
stringContent is the string that is written to the file  
StandardOptions.APPEND is to append the content with the previous content

Reading the file  
Files.readString(Paths.get(“filename.txt”)) is used to read the string content, it reads the entire file content

These are useful while writing / reading the log files.

Multithreading

It allows multiple methods to run simultaneously, each method is run by OS threads when the CPU time is given to the threads

In Java you can create programs to run simultaneously by using 2 API’s

1. Runnable interface: It has an abstract methods run()
2. Thread class

Runnable interface:

It provides a run() method which is an entry point for the threads, Once the thread gets registered it would look for the run() method to begin its execution.

Note: All the operations user wants to do must be called in run() method, as the run() can be executed by multiple threads by sharing the CPU time

run() should be implemented by some classes

Thread class:

It creates thread object and can register the OS threads to run the java application in a multithreaded environment

Thread object must know the entry point to begin, hence while creating the thread object you need to pass the object of Runnable implementation, so that thread will know which run method to execute when it gets the CPU time

Synchronization:

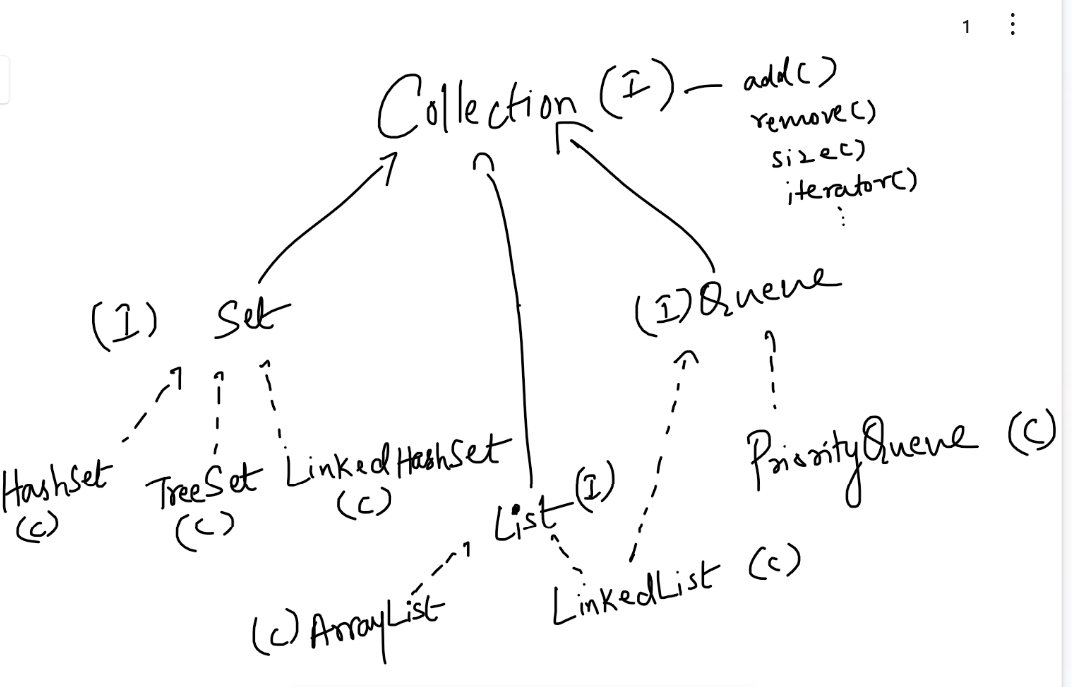
it is a process of making only one thread to execute the method & blocking other threads if a thread has already entered inside the method, other threads would get a chance to execute the method only when there are no threads inside the method, even if there are multiple threads only one thread can execute a method which is called as synchronized method.

Note: To achieve synchronization all the threads must share common object, because thread locks the method once it enters inside the synchronized method and releases the lock once exits the synchronized method, one object will have one lock, to run synchronized method thread need to acquire the lock

Collection Framework

It provides set of classes & interfaces to maintain the elements

* It is dynamic in nature, the size would increase/decrease when add/remove elements
* It can store heterogenous types of elements
* It provides inbuilt methods to meet all the requirements to maintain the elements ex:- sorted elements, unique elements, duplicate elements, search elements, remove elements, iterating the elements and so on



Collection is an interface with lot of abstract methods like

* add
* remove
* size
* clear
* isEmpty
* iterator

Collection has 3 sub-categorizes based on the way we want to maintain the data

1. List: Allow duplicates, it provides index for each element, hence you can add/remove/retrieve using the index, it has 2 implementations
   1. ArrayList: Maintains the elements in contiguous memory address, add/remove of the elements shifts the elements to different address, here adding/removing will be slower, however retrieval will be faster.
   2. LinkedList: Maintains the elements in non-contiguous memory address, add/remove will be faster, however retrieval will be slower

Creating different types of List to store elements

1. List to store integer numbers

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

or

List<Integer> list = new LinkedList<Integer>();

1. List to store employee objects

List<Employee> employees = new ArrayList<Employee>();  
or  
List<Employee> employees = new LinkedList<Employee>();

list.add(element); // it adds the element

list.remove(3); // it removes the element at index 3

list.remove( Integer.valueOf(3) ); // it removes the element whose value is 3

list.add(employee); // it adds employee object

list.remove(employee); // it removes an employee object

Note: All the collection api’s are present in java.util package

Using collection framework in our case-study

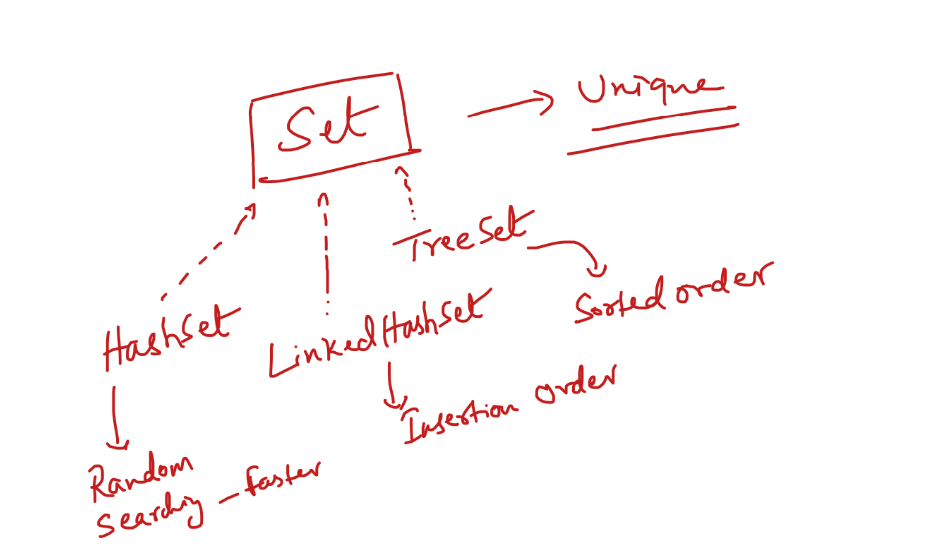
1. Create an implementation EmployeeServiceListImpl for EmployeeService that maintains employee objects in the ArrayList<Employee>, implement below methods as
   1. store(Employee employee): store the elements into the ArrayList
   2. getAll(): return all the employee objects in the ArrayList as Employee[ ]
   3. getEmployee(int id): return the employee based on id by searching in the ArrayList or throw the exception
2. Update the ObjectFactory that can return this new implementation
3. From the main method get this new implementation object and perform the CRUD operations

Important points in the layered architecture

1. Follow the coding standards
   1. package names must be like com.hsbc.beans, com.hsbc.service, com.hsbc.dao, com.hsbc.exceptions and so on
   2. Comments on variables, classes, methods
   3. Proper Indentation
2. Interface as a reference in the caller code and getting its implementation object using factory pattern
3. Not to write print statements in any layer except the view layer, even exception you catch must be rethrown to the view layer
4. Propagating the exceptions to different layers using throws
5. Using throws in the interface methods wherever necessary

Day 8 Agenda

* Collection Api’s
* Set & Queue
* Sorting
* Map



Set: It maintains only unique elements, it uses hashCode & equals to maintain the uniqueness

hashCode: It can determine the location of the element, i.e., the bucket numbers to store the elements

equals: it compares two elements to find the equality whenever there’s an object collision

Note: Every object will have hashCode & equals

equals & hashCode contract

1. two objects having same hashCode may not be always equal
2. two object which are equal will have always same hashCode

Note:

By default Object class hashCode returns the memory address of the object.

By default Object class equals compare two objects memory address

Example: When hashCode & equals are not overridden  
Employee e1 = new Employee(10, “Alex”, 5000); // hashCode = 783939  
Employee e2 = new Employee(20, “Bruce”, 6000); //hashCode = 89999  
Employee e3 = new Employee(10, “Alex”, 5000); // hashCode = 98888  
  
e1.hashCode() == e3.hashCode() // false  
  
Example: When hashCode & equals are overridden on some property (id)

Employee e1 = new Employee(10, “Alex”, 5000); // hashCode = 10  
Employee e2 = new Employee(20, “Bruce”, 6000); // hashCode = 20  
Employee e3 = new Employee(10, “Alex”, 5000); // hashCode = 10

TreeSet: It maintains the elements in sorted order

TreeSet can automatically sort primitives & objects which are implementing Comparable interface

Comparable interface: It has a method that will have the sorting logic, it has only one abstract method called compareTo(Object)

TreeSet can easily sort numbers, strings because they are single values, however it can’t sort complex objects as they will not be having single property, hence TreeSet expects Comparable interface to be implemented, which would mention on which property sorting must apply

Implementing Comparable: compareTo method must be implemented to return an int value, TreeSet can sort based on these int values, if 0 then 2 elements are equal, if -ve then keeps towards the left side of the sorted element, if +ve then keeps towards the right side of the sorted element  
  
class Student implements Comparable<Student> {   
 // rollNo, name, dob, …  
 public int compareTo(Student s) { return rollNo - s.rollNo; }  
}

Set<Student> set = new TreeSet<Student>();  
set.add(new Student(5, … )); // compareTo >> 5 - 5  
set.add(new Student(7, …)); // compareTo >> 7 - 5 = +ve, so 7 goes towards right side of 5  
set.add(new Student(4, …)); // compareTo >> 4 - 5 = -ve, so 4 goes towards left side of 5

Note: You can compare either by finding the difference or by using some inbuilt compare methods in Wrapper classes & String classes

ex: Integer.compare(x, y): returns 0 if x & y are same, else -1 if x < y and +1 if x > y

ex: Double.compare(x, y): same as above

ex: string.compareTo(string): there’s a compareTo method in String class which you can call on any string object

How to use these methods in Comparable is

public int compareTo(Student s) {  
 return Integer.compare(rollNo, s.rollNo); // to sort based on rollNo  
 or  
 return name.compareTo(s.name); // to sort based on name  
}

Note: With Comparable you can sort the objects on any one property not on all, it means you can have only one sorting logic on any one property

Comparable can be used only for default sorting, if you need different types of sorting use Comparator

Comparator:

It is also an interface which has only one abstract methods compare(Object, Object)

* It is implemented separately not within the same class unlike Comparable
* It can have multiple implementations when you want multiple ways of sorting

ex:

* Sort rollNo in descending order
* Sort name in ascending order
* Sort name in descending order and so on

You can create TreeSet object by passing the implementation of Comparator so that it can use the sorting logic of Comparator instead of Comparable

class SortByName implements Comparator<Student>{   
 public int compare(Student s1, Student s2) {   
 return s1.getName().compareTo(s2.getName) ;  
 }   
}   
class SortByRollNoDesc implements Comparator<Student> {   
public int compare(Student s1, Student s2) {   
 return Integer.compare(s2.getRollNo(), s1.getRollNo());   
 }  
}

Now you can create TreeSet by passing object of any Comparator implementation

Set<Student> set = new TreeSet( ); // it uses Comparable   
// below uses Comparator of SortByRollNoDesc  
Set<Student> set = new TreeSet( new SortByRollNoDesc () );   
// below uses Comparator of SortByName  
Set<Student> set = new TreeSet( new SortByName() );

Note: You can implement Comparator in 3 different ways

1. An independent class which is not recommended
2. An inner class of the class to which sorting must be applied, which is recommended

class Student implements Comparable {   
 class SortByName implements Comparator<Student> { } // inner class  
}

Set set = new TreeSet( new Student.SortByName() );

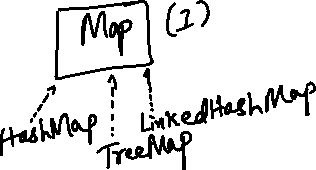
1. A local inner class/anonymous class which is created at the place where you create TreeSet which is more widely used

// local inner class  
Comparator<Student> sortRollNo = new Comparator<Student>() {   
 // compare implementation  
}

Above code implements & creates object of comparator implementation with a reference variable sortRollNo  
Set s = new TreeSet( sortRollNo );

Map:

It maintains the elements in key value pairs, it is not part of collection framework, but has inbuilt classes & interfaces to maintain the data in key & value pairs



Map<String, String> map = new HashMap<>();

map.put(“username”, “system”);  
map.put(“password”, “root”);  
map.put(“url”, “http://abc.com”);

System.out.println(map.get(“url”)); // prints http://abc.com

System.out.println(map);

Immutable Collections:

We can create collections that can’t be modified, there’s a static method present in List & Set, which accepts 0 or more arguments, the method name is of(arguments…);

List<Integer> integersList = List.of(2, 1, 6, 7, 5);   
// creates a list of values passed in of, you can’t modify it  
  
Set<String> stringSet = Set.of(“Mobile”, “Laptop”, “Books”, “Bags”);

Set<Student> studentSet = Set.of( s1, s2, s3, s4);

s1, s2, s3 and s4 are student objects

Personal laptop configuration

1. Install MYSQL for tomorrow’s session - you need to install mysql server and mysql shell (which are available from a single MySQL software)
2. Download MySQL connector jar file of same MySQL software version

ex: If MySQL database is v8.0 then download the MySQL connector jar of v8.0

Hint: mysql connector jar file name will be mysql-connector-j-8.x.x

Using layered architecture try to manipulate student data which must able to add, remove, getAllStudents(), getStudent(), then Student must have rollNo, name, marks of 3 subjects, average, dob, maintain only unique students in the DAO layer,

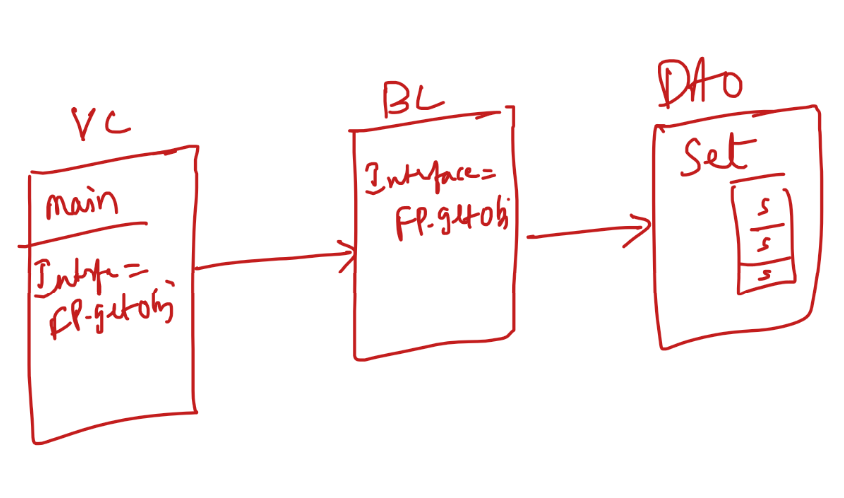
From the service layer access the DAO layer and call add, remove, getAllStudents, getStudent from the marks, calculate the average in the service layer and initialize the average of each Student you want to get.

If duplicate student throw StudentAlreadyExists exception, if student is not found throw StudentNotFoundException required in remove() & getStudent()

Design the classes by using all the best practices, and with the help of main method create a menu with 4 options

* 1. Add student
  2. Remove student based on rollNo
  3. Display a student based on rollNo, but dob must be displayed in dd/MM/yyyy and print the average marks
  4. Display All Students
  5. Exit

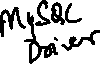
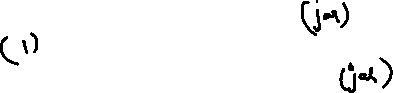
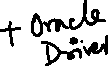
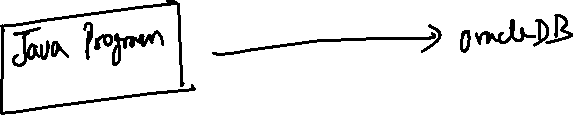
Use all the coding standards, make the layers loosely coupled



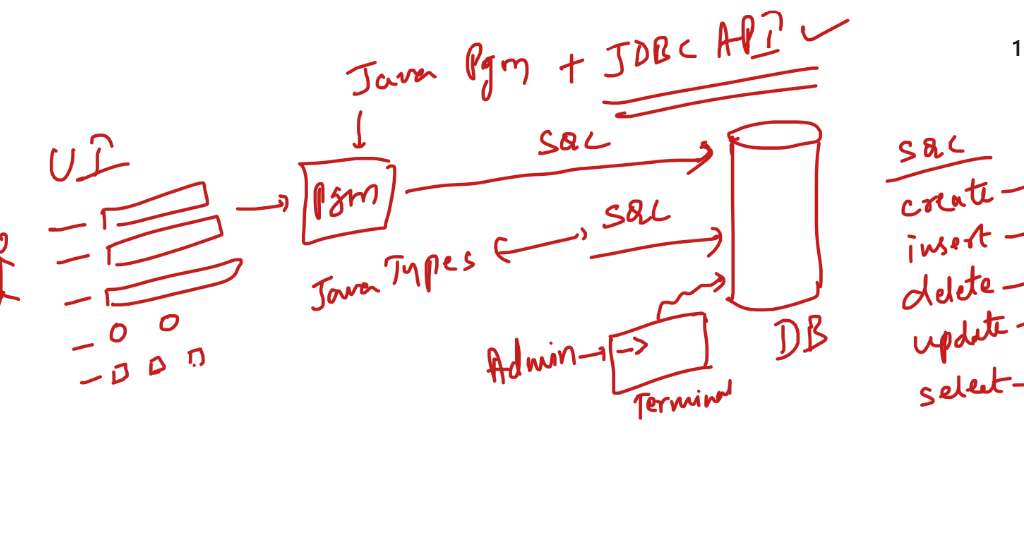
JDBC

Java Database Connectivity: These are API’s which can interact with any databases, every database provides database drivers in the form of Jar files, you need to use these jar files to interact with their database

Database Drivers: These are the classes which implement JDBC API, every database has their own implementations for JDBC API



Note: Every database gives jar files which will have the database drivers  
ex: MySQL database provides mysql-connector-j.jar  
ex: Oracle database provides ojdbc.jar



JDBC helps to pass SQL queries using java programs, which converts Java objects to SQL and vice versa

Steps to interact with the database

1. Loading the driver

Class.forName(“com.mysql.cj.jdbc.Driver“); // MySQL driver  
Class.forName(“oracle.jdbc.OracleDriver”); // Oracle driver

1. Connecting to the database

Connection connection = DriverManager.getConnection(url, username, password);  
url = “jdbc:mysql://localhost:3306/database\_name”  
username = root  
password=….

1. Create Statement

PreparedStatement statement = connection.prepareStatement( sqlQuery );

sqlQuery = “insert into employee values(?, ?, ?)”;  
sqlQuery = “select \* from employee”;  
sqlQuery = “update employee set name = ? where id = ?”;  
sqlQuery = “delete employee where id = ?”;  
Note: ? will accept values at runtime, you can replace ? with the values using the setter methods present in the PreparedStatement

ex: statement.setInt(?position, value); // converts Java int to SQL number  
ex: statement.setDouble(?position, value); // converts Java double to SQL number  
ex: statement.setString(?position, value); // converts Java String to SQL varchar

Note: ? position starts from ‘1’

1. Execute the Statement
   1. int updatedRows = statement.executeUpdate(); // insert, update & delete

Above code is only for DML it returns an int that says how many rows are affected

* 1. ResultSet result = statement.executeQuery(); // select

Above code is for DRL it returns 0 or more records which are navigated using ResultSet

1. Closing the resources

result.close();  
statement.close();  
connection.close();

Note: Most of the methods throw checked exceptions, like SQLException, ClassNotFoundException, you need to handle them using try-catch  
Note: As most of the resources need to be closed, all these resources are auto-closeable hence you can use try-with resource syntax

ResultSetMetaData:

It helps you to get more details on the ResultSet like all the column names, column datatypes, number of rows and etc.

ResultSetMetaData metaData = result.getMetaData();

metaData.getColumnCount() // returns the number of columns   
metaData.getColumnTypeName(columnIndex) // returns the column data type name like varchar, int …  
metaData.getColumnLabel(columnIndex) // returns the column name of a particular index

These helps you to use the right getter methods of ResultSet like

result.getString(“name”);

Database Transactions:

It deals with set of dependent task either all should be saved or none

In JDBC transaction is disabled because it auto-commits each changes you make on the database it could be insert, update or delete statements, you need to enable the transaction by setting auto-commit to false

How to set auto commit false

connection.setAutoCommit(false);

Now any changes you make in the database will not be saved until you use connection.commit(), to undo the changes you can use connection.rollback();

Account table with accountNumber, balance columns

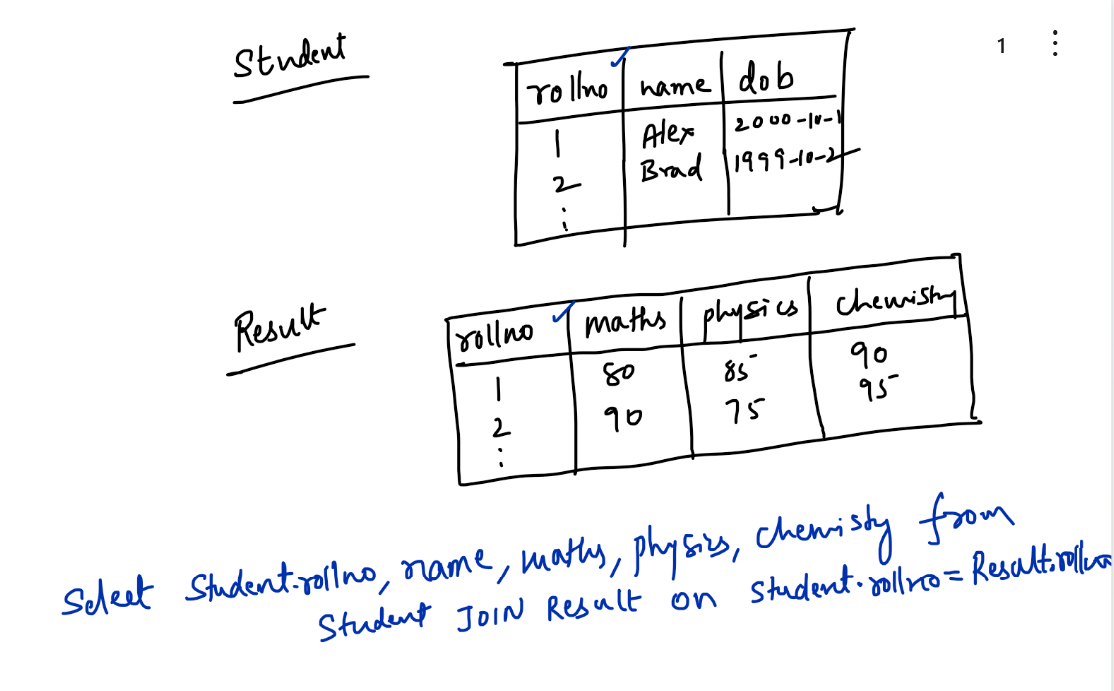
transfer(long source, long destination, double amount) {   
 connection.setAutoCommit(false);  
 // update query on source account  
 // update query on destination account  
 connection.commit();  
 // if any exception occur or destination account is not available then you can have   
 connection.rollback();  
}

SQL Joins:

Joining multiple tables to get a result based on some condition.

Syntax:

select column1, column2, …. from table1 join table 2 on condition;



employee-case-study: perform CRUD operations on the database

interface EmployeeDAO {   
 int save(Employee employee) throws EmployeeCreationException;  
 Employee findById(int id) throws EmployeeNotFoundException;  
 List<Employee> findAll();  
 void deleteById(int id) throws EmployeeNotFoundException;  
}

* Implement the above interface in EmployeeDAOJdbcImpl, use factory pattern to return this object
* Create an interface in the service layer to perform CRUD operation

interface EmployeeService {   
 int store(Employee employee) throws EmployeeCreationException;  
 Employee findEmployee(int id) throws EmployeeNotFoundException;  
 List<Employee> findEmployees();  
 void deleteEmployee(int id)throws EmployeeNotFoundException;  
 int store(Employee[] employees)throws EmployeeCreationException;  
}

* Implement the above interface, use factory pattern to create its object, the implementation must use EmployeeDAO reference and perform CRUD operations
* EmployeeService.store(Employee[] employees) must able to call EmployeeDAO.store(Employee) by iterating the array
* Create a main class and call all the methods of Service using some menu option
* Design the classes & interfaces as per the need by following best practices

Linux:

* It is a multi-user OS
* Used mainly to deploy real-time applications
* It is a terminal based OS

Multi User OS: Multiple user can work remotely at the same time in the same OS

You can connect to Linux OS using any SSH clients like Git bash, Putty and so on.

Linux will come in 2 forms

1. Terminal based - used to run servers, databases, applications and many other real-time work
2. GUI based - used by the end users to create applications or do some other work similar to Mac & Windows machine

Ex: If Windows/Mac takes 5 minutes to install any software, then the terminal based OS takes 10 to 20 seconds

Some of the useful Linux commands

# creating a single file

touch test.txt

# creating multiple files with the name test, demo, hello

touch test.text demo.txt hello.txt

# creating a folder

mkdir bin

# Navigating to the directories

Going one folder backward: cd ../  
Going inside a sub-bolder named bin: cd bin  
Going 2 steps backward from the current folder: cd ../../foo

# View files & folders

Lists files & folders: ls   
Lists files & folders with permissions: ls -l  
Lists files & folders including hidden files & folders: ls -la

# Deleting files and folders

rm test.txt: deletes the file test.txt

rm -rf demo: deletes demo & its files and sub-folders

# View content of the file

cat test.txt

cat test.txt hello.txt demo.txt

# Writing the content

Appending: cat >> demo.txt

Rewrite: cat > demo.txt

# Renaming files & folders

Rename 1 to 11: mv 1.txt 11.txt

# Copying the files

Copy 1.txt content to 2.txt: cp 1.txt 2.txt

# Installing some software’s

Installing Git: sudo yum install git

Note: sudo means super user do, it works only if you have a real Linux terminal

# Searching the contents: grep (Global Regular Expression Print)

To search hello in 1.txt: grep hello 1.txt

To search hello in multiple files: grep hello 1.txt 2.txt 3.txt

To count hello in 1.txt: grep -c hello 1.txt

To count hello in multiple files: grep -c hello 1.txt 2.txt 3.txt

HTML, CSS & Javascript

HTML: Hyper Text Markup Language

CSS: Cascading Style Sheet

Browser can interpret all the three technologies, they are mainly used to create front-end UI’s

Front end will have   
HTML: used to display the UI like text, buttons, input box, radio button, check box, forms, images, tables

CSS: used to style the HTML elements like coloring the footer, header, buttons, text and etc

Javascript: It is a programming language for the front-end which can perform various actions based on the interactions like changing the content, basically it can modify HTML & CSS at runtime

public or WebContent / \*.html, \*.css, \*.js

<form action = “url-pattern” method = “post”> <!-- by default method = “get” -->  
  
</form>

Types of CSS

1. Inline CSS - Adding CSS to the particular tag using style attribute

ex: <h1 style = “color:red; font-family:arial”>….</h1>

1. Internal CSS - Adding CSS to the entire HTML document using <style> tag

<style>  
 h1 { color : blue; font-family: arial }  
 div { margin: 5px; }  
</style>

1. External CSS - Adding CSS to the multiple HTML document using .css file

h1 { property: value; property: value}   
h2 { property: value; property: value}

h1, h2, div are all treated as selectors in CSS, in CSS you can also use other types of selectors other than tag names

1. class selector: multiple html tags can use same class
2. id selector: only one tag can use one id (to uniquely identify the tag we use id)

class selector:

.c1 { property: value; property: value }

id selector:  
#i1 { property:value; property:value }

<div class = “c1”>…</div>  
<h2 class = “c1”>…</div>

<h3 id = “i1”>..</h3>  
<h3 id = “i2”>..</h3>

Javascript is used to access HTML & CSS to make changes at runtime based on the user interaction like placing mouse over, clicking an element, focusing on an element, submitting the form, entering input and etc.

Javascript is a programming language which are case-sensitive and doesn’t allow errors, browser shows the errors in the console if there are any syntax/typo errors.

Javascript supports many programming constructs like other programming languages like:-

1. variables: let x = 10; var y = 10; const z = 55;
2. operators: +, -, /, %, ++, --, ==, <, >
3. loops: for, do-while, while
4. conditional statements - if, if - else, if else if else if… else, switch
5. Functions: function test() { … }
6. Objects: employee = { id: 100, name: “Alex”, salary: 40000 }
7. Arrays: [20, 30, 10, 50, 40]

Printing output in Javascript

console.log(“print some strings”); // prints in browser developer tools console

alert(“print some strings”); // shows an alert dialog box

document.write(“print some strings”); // print string in the web page

How to call Javascript functions

They are normally called based on the HTML events / DOM events (Document Object Model) like onclick, onmousover, onmouseout, oninput, onblur, onfocus, onsubmit, …

<button onclick = “greet(args)”>Button1</button>

<p onclick = “greet(args)”>Some text</p>

How to include javascript in HTML

<script type = “text/javascript” src = “main.js”></script>

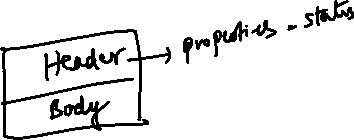
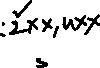
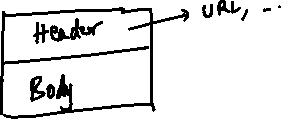
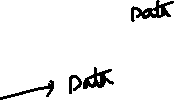
AJAX: Stands for Asynchrnous Javascript and XML, it enables javascript front-end applications to access the backend resources so that front-end can send the data to the backend also get the data from the backend.

Note: There are other front-end technologies which uses their own way to access the backend resources

ex:

1. React.js uses axios library to access the backend resource
2. Angular Framework uses HttpClient to access the backend resource
3. Java based front-end technologies like Android apps can use HttpClient

Javascript uses an API called XMLHttpRequest to make AJAX calls, it can send Http Request to the backend and get the Http Response from the backend.



let xhr = new XMLHttpRequest();

xhr.open(“method”, url); // to create request

xhr.send(); // sends the request

xhr.onreadystatechange = function() {   
 if(xhr.readyState == 4) {   
 let data = xhr.responseText; // JSON format  
 let jsData = JSON.parse(data); // converts JSON to Javascript object  
 }  
}

Java Servlets

Servlets: These are java programs which are run by servers to handle request & generate response.

How to create Servlets in Java

You must implement an interface provided by Java which is a “Servlet”, those implementations would follow the rules on handling the request & generating the response

Servlet interface has 5 methods which specifies its role

1. void init()
2. void service(ServletRequest, ServletResposne)
3. void destroy()
4. String getServletInfo()
5. ServletConfig getServletConfig()

Out of the above 5 methods, only service(..) is useful, because it is called when the request comes to the servlet & generates the response

init(), service() & destroy are called as life cycle methods

init(): it is executed when the servlet object is created

service(): it is executed when the request comes to the servlet

destroy(): it is executed when the servlet object is destroyed/removed from the memory

Servlet Architecture

1. Servlet objects are created on the first request by default, however you can have a load on startup configuration on any servlet which needs to be instantiated before the first request
2. Servlets are multi-threaded by default, so only one servlet can handle one or more requests or clients.
3. Every servlet will have an URL mapped, so that client sends the URL to the server, then the server maps that URL to the servlet, then the service method can handle the request & generate the response.
4. init() is called when the servlet object is created i.e., on first request by default
5. destroy() is called when the object is removed from the memory, it is removed only if the servlet is idle for long time as per the server configuration.
6. Servlet by default generates the response content in HTML format, however we can change it to the format the client expects like JSON, XML and so on
7. In Servlet multiple threads are created for each request instead of multiple servlet objects.

Note: Servlet acts like a controller in the MVC pattern application

New Features of Java

* Functional Programming: Passing functions as parameters (From Java 8)
* Lambda Expression: Enhancement to the anonymous class that you write for the interface having single abstract method (From Java 8)
* Static & Default methods to the interface (From Java 8)
* Local variable syntax in Lambda expression (From Java 11)

Functional Programming: Passing functions as an argument instead of passing object as arguments, it is applied only on the interface which has one abstract method which is also called as Functional interface

interface A {   
 void test( );   
}

void demo(A a) {   
 a.test();  
}

// older approach

class Impl implements A {   
 public void test() { … }  
}

demo(new Impl());

// another approach is using an anonymous class

A a1 = new A() {   
 public void test() { … }  
};

demo(a1);

// new way is you can pass the test() implementation directly

// you must use lambda expression for this

A a1 = () -> { … } // this is the implementation of interface A which is using lambda expression

demo(a1);

or

demo(() -> { } );

interface B {   
 int add(int x, int y);  
}

// older approach

class ImplB implements B {   
 public int add(int x, int y) { return x + y; }   
}

void demo(B b) { b.add(20, 30); }

demo(new ImplB() );

// lambda expression

B b1 = (x, y) -> x + y; // this is converted to

B b1 = new B() {   
 public int add(int x, int y) { return x + y; }   
}

demo( (x, y) -> x + y );

Different ways of writing lambda expressions

1. If its only one line implementation no need of return keyword & { }
2. If more than one line implementation you need to use { } and return keyword if method returns any value

(a, b) -> a \* b; // returns a \* b

() -> 10; // returns 10

a -> a \* 10; // returns a \* 10, where a is the parameter

a -> { // some statements and then return a \* 10; } when 1 or more lines of implementation

Anonymous class implementation for Comparator

Comparator<Integer> comparator = new Comparator<Integer>() {   
 public int compare(Integer x, Integer y) {   
 return Integer.compare(x, y);  
 }  
}

Lambda expression for the above Comparator implementation is

Comparator<Integer> comparator = (x, y) -> Integer.compare(x, y); // now you can pass this function to the tree set directly instead of the comparator reference

i.e.,

Set<Integer> set = new TreeSet<Integer>( (x, y) -> Integer.compare(x, y) )

Some of the inbuilt functional interfaces are:

1. Comparator
2. Runnable
3. Function
4. Predicate
5. Consumer

Java 11 Local variable syntax for lambda expression

You can use var keyword to the parameters of the lambda expression

i.e.,

(var a, var b) -> a \* b;

Using type in the parameter enables you to attach some annotations that will give some extra features to the parameters without writing the code

ex:

(@NotNull var a, @NotNull var b) -> a \* b;

(@NotNull @NotEmpty var a, @NotNull @NotEmpty var b) -> a \* b;

String s = null; // s is null

String s = “”; // s is empty

Method Reference:

These are used on the functional interface whose abstract method will not be implemented instead you would refer to the existing implementation which will have the same signature of the functional interface abstract method.

interface X {   
 int demo();  
}

class Abc {   
 public int m1() { … }  
 public int m2() { … }  
 public int m3() { … }  
}  
  
Abc a = new Abc();  
// method reference  
X x == a :: m1;  
x.demo(); >> it calls m1() method  
// with method reference  
X x2 = ( ) -> { …. } // your own implementation for demo  
x2.demo(); // it calls your own implementation done in lambda  
// method reference  
X x3 = a :: m2;  
x3.demo(); >> it calls m2() method

Note: You can reference static methods also to the function interface

interface A {   
 int demo(int x, int y);  
}  
A a = Integer :: compare; // Integer is the class & compare is a static method  
int r = a.demo(2, 3); // returns -1

interface B {   
 LocalDate test(String s);  
}  
B b = LocalDate :: parse;

Java 8 Streams

Java 8 Streams are API’s released in Java 8, they are used to manipulate the collection elements without modifying the existing collection/data-structure like sorting, filtering, iterating, transforming, aggregating and so on.

There are two types of streams

1. sequential stream: all the stream operations are done by single thread
2. parallel stream: steam operations are done by multiple threads

Java 8 has provided two methods to apply streams on the collection framework API’s

* stream(): sequential stream
* parallelStream(): parallel stream

Java 8 stream API’s provide methods that accepts functional interface as a parameter to perform the operation, it uses functional interface to enable developers to use lambda expression to simplify the operations like filter, sort, transform, aggregate, iterate and so on.

Below are the functional interface Java 8 Stream methods use based on the type of operation

|  |  |  |
| --- | --- | --- |
| Functional Interface | Abstract method | Description |
| Predicate<T> | public boolean test(T t) | applies condition and returns true/false |
| Consumer<T> | public void accept(T t) | accepts value and does some task on it |
| Function<T> | public R apply(T t) | accepts a value and returns some other value  T is a type of one value  R is a type of another value |
| Comparator<T> | public int compare(T t1, T t2) |  |
| Supplier<T> | public T get() | returns some value |

How the lambda expressions look for the above functional interface

Predicate<T> predicate = (t) -> t == value;

Consumer<T> consumer = (t) -> statement;

Function<T> function = (t) -> t.value;

All these functional interface are parameters of some or other methods present in the stream API’s

ex: below are the stream API methods

filter(Predicate): takes some elements and filters on some condition to return some elements

forEach(Consumer): takes some elements that needs to be iterated

map(Function): takes x elements and transforms to another form ex: converting object to another object

mapToInt(Function): converts x element to int type elements

mapToDouble(Function): converts x elements to double type elements

mapToLong(Function): converts x elements to long type elements

sort(Comparator): takes x elements and sorts them

Filtering a List<Employee> using stream

Stream methods return Stream, hence you can chain the stream methods

ex: list.stream().filter().forEach()  
ex: list.stream().filter().sort().forEach()  
ex: list.stream().filter().filter().filter().sort().forEach()  
ex: list.stream().filter().map().forEach()

List<Employee> employees = new ArrayList<>();  
….  
employees.stream()  
 .filter( employee -> employee.getSalary() > 50000 )  
 .forEach(employee -> System.out.println(employee));

Filtering the FoodItems, based on the food-type, ratings, menu-type, price  
List<FoodItems> items = new ArrayList<>();

items.stream().filter(veg).filter(ratings).filter(starters).filter(price).forEach(item)

Types of stream operations

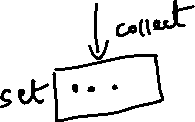
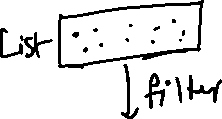
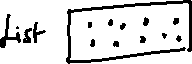
1. Intermediate: This gives another stream
2. Terminal: This ends the stream operations

Intermediate operations are: filter(), sort(), map()

Terminal operations are: forEach(), collect(), sum()

collect(): it creates a new data-structure by collecting the streams, it accepts a parameter to represent stream in some data-structure

sum(): It takes all the element of stream and returns the sum of all the elements



Aggregating methods in streams:

In streams you can perform operations by aggregating the elements to get some result like adding, counting, min, max, below are the methods

sum(): aggregates all the elements and returns their sum  
count(): aggregates all the elements and returns their total occurrence  
max(Comparator): sorts all the elements and returns the Optional type   
min(Comparator): sorts all the elements and returns the Optional type

Optional<T>: It is an instance that may or may not contain value, it gives methods like get() to get the value, isEmpty() if the value is not present,

Java 9 modular system

Modules are collection of packages and classes, unlike jars, modules can expose only selected number of packages whereas jar exposes all the packages

Parallel Streams:

It make use of multiple threads to perform the stream operations, most of the OS supports Fork Join threads which can split the work into multiple threads and join them later.

collection.parallelStream(): This creates a stream that can be worked by Fork Join Threads & main threads

collection.stream(): This creates a stream that can be worked by single thread (main thread)

How to identify the threads used in the parallel stream

collection.parallelStream().forEach(item -> System.out.println(Thread.currentThread));

The above code prints thread names which are performing the iteration, usually you get multiple thread names.

collection.stream().forEach(item -> System.out.println(Thread.currentThread));

The above code prints only a single thread name that performs the iteration

Note: Each time you call intermediate methods you get a new stream

List<Integer> list = List.of(3, 1, 4, 6, 5, 2);  
Stream<Integer> stream = list.stream();  
Stream<Integer> mapStream = stream.map(…);  
Stream<Integer> sortedStream = mapStream().sorted(….);

Stream is the API that has forEach(), map(), sorted(), max(), min(), count(), sum() and so on

Note: Each time you call terminal methods from the stream you either get different type other than Stream like void, long, int, double, List, Set, Map and so on

forEach(..): void  
sum(): T  
count(): T  
collect(..): Collection<T>  
max(..): Optional<T>  
min(..): Optional<T>

Note: Don’t perform sorting using parallelStream()

Thread1 -> 1, 2, 3  
Thread2-> 7, 8, 9  
Thread3-> 4, 5, 6

When its joined by the Fork-Join it can be joined in any order hence you don’t get the desired result

Note: You can use parallelStream() on sum, filter, map, collect, count and so on.

Changes done to the interfaces from Java 8

1. default methods can be added in interface, so that it still doesn’t break any contract, if you override then the overridden method will be called else default method will be called
2. static methods can be added in interface

Note: Java has done some changes in their predefined interfaces like adding default & static methods to many Collection interfaces

Ex: List.of(…) is a static method which is added in List interface, Set.of(..) is another static method part of Set

Ex: Collection stream(), Collection parallelStream() are default methods present in Collection

Static method vs Default method

1. Default methods you can override
2. Static methods you can’t override

Java 8 features

1. Default & Static methods in interface
2. Functional Interface
3. Lambda expression
4. Method reference
5. Stream API
6. LocalDate, LocalTime, LocalDateTime

Java 9 features

1. Module system

Java 11 features

1. String isBlank(), stripLeading(), stripTrailing()
2. local variable syntax in lambda expression, i.e., var in Lambda expression
3. static method of in List, Set, Queue interfaces

Maven:

It is a build tool to configure jar files in your project, it is one the most widely used build tool to configure jars, it takes care of downloading all the dependent jars based on three information’s

1. groupId
2. artifactId
3. version

How to configure jar in our project

We need to mention <dependency> tag along with <groupId>, <artifactId> and <version> of each jar which are usually copied from the internet.

<project>  
 ….  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.0.0-RELEASE</version>  
 <dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>5.0.0-RELEASE</version>  
 <dependency>  
 ……..  
 </dependencies>  
</project>

Spring Framework

Framework is like a semi-implemented application which takes care of all the common task of the application, so that developers can only implement new task required for the application

* It reduces repeated work in every application
* It almost does 60% of the work for your application

Framework takes care of lot of repeated work like

* Type conversion (String to int to String, String to LocalDate, SQL to Java to SQL)
* Handling all the checked exceptions which are defined in java like SQLException, ClassNotFoundException
* Establishing database connection and closing the database resources
* Design pattern - Singleton pattern, Factory pattern and many more
* Object creation for reusable layers like DAO, Service, Controller
* Initializing the dependency objects - Dependency Injection
* Application configurations will be separated from the code

ex: database credentials will be written in the configuration file instead of writing in the code

* It uses industry best practices like configurations and code will be separated

Spring Framework: It helps you to create applications for various platforms like desktop, web, enterprise, mobile, cloud and so on

* It uses many design patterns to make your code loosely coupled
* Uses dependency injection to supply an object to another object
* It provides many modules to perform different set of work

ex: jdbc-module: for database interaction  
ex: web-module: for web application  
ex: cloud-module: for cloud application

* Separates the application configuration from the code

ex: XML file will be used to provide application configurations like database username, password, url

Spring Framework takes care of creating objects in the spring container and supplying them to their dependencies.

Dependency Injection: Process in which an object is supplied to another object

Spring Container: It is a place where objects are created based on the bean configurations done in XML file, it is also called as Inversion of Control (IoC) as the objects are controlled in the container

Older approach:   
I i1 = new Impl(); // object is created in the application and supplied to the runtime environment

Spring approach

I i1 = context.getBean(“a”); // object is created in the container (runtime environment) and supplied to the application, this is why spring container is called as IoC.

How spring container is initialized by reading the xml file

ApplicationContext context = new ClassPathXmlApplicationContext(“file.xml”);

// file.xml name can be anything like beans.xml, applicationContext.xml, spring.xml

Types of Dependency Injection

1. Setter Injection: Spring container initializes the property using setter method, its done by using <property> tag
2. Constructor Injection: Spring container initializes the property using constructor parameter, its done by using <constructor-arg> tag

Note: Either you use setter or constructor injection the class must have suitable setter or constructors with parameters

Generate default & argument constructor for Datasource

Configure xml to have 2 bean for the same class

<bean id = “data” class = “com.hsbc.dao.util.Datasource”>  
 …   
</bean>

<bean id = “data1” class = “com.hsbc.data.util.Datasource”>   
 <constructor-arg name = “username” value = “hsbcadmin” />  
 <constructor-arg name = “url” value = “jdbc:mysql://localhost…” />  
 <constructor-arg name = “password” value = “hello@1234” />  
</bean>

In Main class get the bean id data1 & print their properties

Day 16 agenda

* Spring Framework
* Dependency Injection
* Autowiring
* Annotation based configuration
* Spring Boot

Spring Framework: It helps you to create various types of applications, it takes care of following things in the application

* Design patterns - Singleton, factory, proxy, prototype, adapter, builder and so on
* Object creation - initializing the object properties
* Dependency Injection - dependent object is supplied to another object

ex: dao object is supplied to service, service object is supplied to controller

* Exception Handling - handling all the checked exception except user-defined exception
* Type conversion - string to other types, java types to sql and vice versa
* Database configuration - establishing connection & closing the connections

Spring framework uses spring container which is also called as IoC to perform all the above features

Dependency Injection: supplying an object to another object or initializing the object property  
There are 2 types

* Setter Injection : <property> tag calls setter method
* Constructor Injection: <constructor-arg> tag calls constructor with argument.

Note: By default spring initializes the object using default constructor, if you have not used <constructor-arg>   
Ex:  
<bean id = “i1” class = “com.hsbc.A”></bean>  
Spring container creates an object of A using default constructor  
<bean id = “i2” class = “com.hsbc.B”>  
 <property name = “x” value = “20” />  
</bean>  
Spring container creates an object of B using default constructor & calls setX(20) to initialize x  
<bean id = “i3” class = “com.hsbc.C”>  
 <constructor-arg name = “y” value = “30” />   
</bean>  
Spring container creates an object of C using one argument constructor & calls C(30) to initialize y, however the class must have this constructor else you get an exception

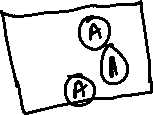
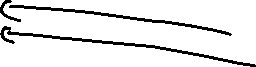
Bean scopes

1. Singleton: On each <bean> one object will be created & you can use that object from getBean(id) - by default <bean> is singleton
2. Prototype: Multiple object can be created for the same <bean>, but it must use one attribute called scope=”prototype”

i.e., <bean id = “i2” class = “com.A” scope = “prototype” />

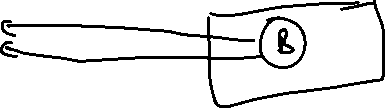
Spring container can create new instance of the class A on each getBean(“i2”) call

A a1 = context.getBean(“i2”);  
A a2 = context.getBean(“i2”);  
A a3 = context.getBean(“i2”);



However if the scope is not present then <bean> will use scope=”singleton”

i.e., <bean id = “i3” class = “com.B” />  
B b1 = context.getBean(“i3”);  
B b2 = context.getBean(“i3”);



Dependency Injection on complex type

spring container can supply an object to another object when <bean> configuration uses “ref” to refer another <bean>

ex: controller depends on service, hence controller <bean> can refer to service <bean>

ex: service depends on dao, hence service <bean> can refer to dao <bean>

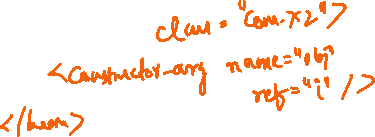
When these configures are done spring container supplies an object to another object

Spring container can supply the object to another object using

1. setter injection or
2. constructor injection

Hence you must either have setter method or constructor with an argument for that complex type

interface I { }   
class I1 implements I { }   
class I2 implements I { }  
  
class X1 {   
 I obj;  
 void setObj(I obj) { this.obj = obj; }   
}  
class X2 {   
 I obj;  
 X2(I obj) { this.obj = obj; }  
}



New Project

1. Maven Project
2. Modify pom.xml
3. Copy applicationContext.xml to the src/main/resources and rename to spring-beans.xml
4. Create EmployeeDao, EmployeeDaoImplV1, EmployeeService, EmployeeServiceImpl, TestViewController
5. Modify beans.xml to configure EmployeeDaoImplV1 & EmployeeServiceImpl

Auto-wire

It helps spring container to supply the object without using setter or constructor injection which means you don’t need to use <property> or <constructor-arg> tag for the dependencies, but you need the setter method & constructors in the class.

It avoids writing multiple <property> or <constructor-arg> tag

<bean id = “dao” class = “com.dao.EmployeeDaoImpl”></bean>  
<bean id = “dao2” class = “com.dao.EmployeeDaoImpl”></bean>  
  
<bean id = “s1” class = “com.service.EmployeeServiceImpl” Autowire=”byName”></bean>

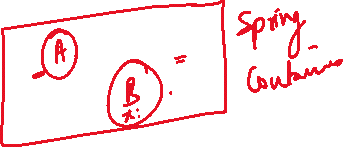
Note: Here EmployeeServiceImpl must have a variable name “dao” or “dao2”

Note: If there was only one object of EmployeeDaoImpl then we could use byType

Bean configurations using annotations

Spring 2.5 onwards you can configure the beans using annotations, these annotations help spring container to scan and perform some operations like creating object, supplying object

@Repository   
class A { }   
  
@Service  
class B {   
 @Autowired  
 A x;  
}

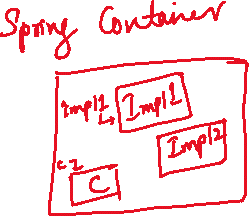


Spring has provided annotations for different layers,   
DAO : @Repository  
Service: @Service  
Controller: @Controller  
Rest Web-Service: @RestController  
Utility classes: @Component

@Autowired: It does the dependency injection job by supplying the object based on datatype, however if you multiple object matching to the datatype you need to use @Qualifier to specify which object qualifies

interface I { }

@Repository  
class Impl1 implements I { }  
@Repository(name = “x”)   
class Impl2 implements I { }   
@Service  
class C {   
 @Qualifier(“impl1”)  
 @Autowired  
 I obj;  
}



Note: Spring creates bean id using the class name starting with lowercase, but you can also give the id using @Repository(name = “x”), @Service(name = “y”)

@Qualifier: It is used when there’s an unsatisfied dependency, i.e., when spring container requires one bean to inject but finds more than one, in that case @Qualifier would be helpful.

Note: All the above annotations work only if you mention a scanning configuration in the XML file

<context: component-scan base-package = “com” />

@Primary: It gives higher priority while supplying the object, it can be used when there are multiple objects matching but spring container wants to supply the higher priority object to avoid the unsatisfied dependency exception

@Repository  
class A implements I { }   
  
@Repository  
@Primary  
class B implements I { }

@Autowired  
I obj; // now the class having @Primary will be qualified

Note: @Autowired doesn’t need setter method or constructor argument to supply the dependency, which was mandatory when you autowire in the XML

ApplicationContext:

It is a reference to the container, it is extended from an interface called BeanFactory

i.e., interface ApplicationContext extends BeanFactory { …. }

which means below code also works

BeanFactory factory = new ClassPathXmlApplicationContext(“…”);

Day 17 Agenda

* Spring Boot
* Spring MVC
* Spring JDBC
* Spring AOP

Spring MVC

It is one of the module helps to create web application, it takes care of initializing spring container with the help of a Front-Controller (DispatcherServlet)

Spring MVC architecture

It will have

1. View
2. Front Controller (Dispatcher Servlet)
3. Controller
4. Model (Service & DAO)

View: is a layer that will have html & JSP

Front Controller:

1. it accepts all the incoming request & maps the request to appropriate controller
2. it takes care of initializing the spring container i.e., ApplicationContext context = new ClassPathXmlApplicationContext(xml-file-name), it finds the xml-file-name by looking at the web.xml <name> tag
3. it also takes care of rendering/calling the view using the response given by the controller

Controller: It process the request and calls the model and returns the response to the front-controller with the view-name

Model: it will have service & Dao layer

Note: The above architecture is used only in a traditional web-application where view also part of the application and it can be accessed only by the browser

DispatcherServlet: It is a servlet which is programmed to handle all the incoming request to the application, it is called as front-controller

Spring REST

It uses Spring MVC architecture in an enhanced form, where any type of application can exchange the data and view will be separated from the application.

Ex:

Google pay application is sending the request to the REST webservice and getting the data from it, these data will be in JSON format so that applications can convert them to the structure it can understand.

Note: Either we use Spring MVC or Spring REST both will use Front-Controller which is provided by Spring framework, whose name is DispatcherServlet which you will get from spring-mvc.jar file

Server: It is a runtime environment that can run your application, there are various servers

1. Apache Tomcat: Apache company
2. JBoss: Redhat company
3. Weblogic: Oracle company
4. Jetty: Eclipse
5. Glasfish: Sun Microsystem / Oracle

Note: If you want to develop Spring Web MVC or Spring REST based applications you need to use server, which means you need to configure your project to use any one of the server

Spring Boot:

It helps you to quickly create spring applications for any environment, it automatically configures your applications for the environment you need

* Web/REST application - provides embedded tomcat server for you, provides dispatcher servlet as well, no need of web.xml or any other xml files
* Component scanning - automatically detects all the annotations
* Database connection - provides database connection for you based on the configurations done in application.properties (default configuration file for spring boot)
* It uses all the spring core configurations like design patterns, exception handling and other common features
* It uses build tools like Maven/Gradle

How can spring boot perform these auto-configurations

Spring boot provides some starter jar files which are called starter projects, which would give the feature you need when you have these jars in your project

1. Spring Boot Starter Web: If this jar is added in your project then you get below configurations automatically
   1. Embedded server (Tomcat)
   2. Component Scanning
   3. Dispatcher Servlet
2. Spring Boot Starter Data JDBC: If this jar is added in your project, it reads application.properties automatically for database credentials
   1. Establishes database connection
   2. Handles all the checked exceptions like SQLException, ClassNotFoundException
   3. Provides an object “JdbcTemplate” that can perform CRUD operations in a simple-way
   4. Also closes all the database resources
3. Spring Cloud: It this jar is added in your project, then it provides all the features to work in any cloud environment (AWS, Azure, GCP, Digital Ocean)

Note: All these features are automatically configured by running the spring boot application - to enable this automation spring boot provides an annotation called @SpringBootApplication - this annotation will take care of configuring the application for the particular environment

How to use this @SpringBootApplication

@SpringBootApplication  
public class MyApp {   
 public static void main(String[] args) }   
 SpringApplication.run(MyApp.class, args);  
 }  
}

SpringApplication is an inbuilt class to load the spring boot application class

RESTful webservices:

It helps heterogenous applications to exchange the data, RESTful uses 2 main things to make clients to communicate

1. URL
2. HTTP method

HTTP method

1. GET: fetch operation
2. POST: store operation or create new resource
3. PUT: update operation
4. DELETE: delete operation

Spring provides annotations for each HTTP methods

1. GET: @GetMapping
2. POST: @PostMapping
3. PUT: @PutMapping
4. DELETE: @DeleteMapping

Spring JDBC

It helps spring applications to interact with the database

Advantage

* It takes care of establishing the database connection
* It takes care of closing all the resources
* It takes care of handling all the checked exceptions like SQLException, ClassNotFoundException
* It provides inbuilt methods where you need to pass SQL queries

Note: When you use Spring JDBC you will not create Connection, PreparedStatement, ResultSet, Class.forName(..)

JdbcTemplate: It is an object spring container creates to perform the CRUD operations, it has methods like

1. update(sqlQuery, arg1, arg2, arg3,….): it is used to update the table using insert, update & delete queries, returns int (rows updated)
2. queryForObject(sqlQuery, RowMapper<T>, arg1, arg2,…): it is used to select the record based on some arguments.
   1. sqlQuery: select query with where condition
   2. RowMapper<T>: it is a functional interface that generates the ResultSet based on the arguments and creates a java object out of that ResultSet
   3. arg1, arg2,..: These are the values for where condition

ex: “select \* from employee where id = ?” for this query only one argument, then RowMapper might get a ResultSet with 0 or one record

Note: queryForObject returns the object the RowMapper<T> returns

Note: RowMapper<Employee> returns employee  
Note: RowMapper<Student> returns student

Note: RowMapper<T> has one abstract method i.e., map(ResultSet, rowNum), map method will convert ResultSet to an object

1. query(sqlQuery, RowMapper<T>): It is used for selecting multiple records, it accepts RowMapper<T> to convert the ResultSet to List<T>

JdbcTemplate: It is an object spring container creates by reading application.properties, the property file must have datasource username, password, url & driver class

@Autowired  
JdbcTemplate template; // Spring container injects the JdbcTemplate present in the container, you will auto-wire it in the DAO layer

RowMapper<T>: It takes care of converting ResultSet data to Java object, it is a functional interface & it has a method map(ResultSet, rowNum) that returns T

Implementation of RowMapper for Employee class: id, name, salary

RowMapper<Employee> mapper = (rs, row) -> new Employee(rs.getInt(1), rs.getString(2), rs.getDouble(3));

Now you need to pass this mapper to queryForObject and query

Employee emp = template.queryForObject(“select \* from employee where id = ?”, mapper, someID);  
  
List<Employee> list = template.query(“select \* from employee”, mapper);

Note: query & queryForObject method generates the ResultSet & passes to the RowMapper so that it can give an object or List

We need to use a library called

spring data jdbc: This library provides JdbcTemplate, RowMapper & helps to connect / disconnect from the database, register the JdbcTemplate in the spring container

Day 18 Agenda

* Spring REST
* Spring Microservices
* AOP

RESTful webservices

1. URL of the webservice
2. HTTP methods : get, post, put, delete
3. Data-structure: JSON, XML, CSV, TEXT, HTML

In Spring Framework you have annotations to map the HTTP methods and also media-type to specify the data-structure it can produce or consume

ex:

@GetMapping(path = “/fetch”, produces = MediaType.APPLICATION\_JSON\_VALUE)  
public Account getAccount() // returns account object in JSON format

i.e., { “id”: 12345, “balance”: 25000, “type”: “savings” }

Other HTTP mappings in spring rest

@PostMapping(path = “…”, consumes = …., produces = …. ): It is used to create a new resource like storing the new resource, it maps to HTTP POST

@PutMapping(path = “…”, consumes = …., produces = …): It is used to update the existing resource, it maps to HTTP PUT

@DeleteMapping(path = “…”, consumes = …., produces = …): It is used to delete the existing resource, it maps to HTTP DELETE

@GetMapping(path = “…”, produces = …): It is used to fetch the existing resource, it maps to HTTP GET.

Out of 4 methods, GET will not expect client to send any data in the request body, only the data coming from the request body can be consumed.

Note: When client sends GET request, it may not be able to send data through request body, but it can send the data from the URL

ResponseEntity<T>:

It is used to generate the response with header & body, T stands for the datatype of the response content (body)

How to create ResponseEntity<T>

ResponseEntity.status(200).body(data): This creates a ResponseEntity of status code 200 & body will have some data

ResponseEntity.status(404).body(data): This creates a ResponseEntity of status code 404 & body will have some data

Dev tools:

It is one of the spring boot library it auto-reloads the server when you make changes in the application.

Microservice

It is an independent web-service from other web-service which you can develop & deploy independently

Benefits

1. If one service goes down other services won’t be affected
2. You can scale only a particular service required
3. You can use any programming language to implement each service within the application
4. Testing all the services are not required if any modification is done to a particular service
5. You can make one service to exchange the data to another service as they are REST services

Design patterns & tools to implement microservice

1. Service Discovery: It is a program which can register the microservices(instance-id & physical address), Spring uses Eureka Server as a service discovery, K8S uses config map as a service discovery
2. Client side load balancer: It is a program in a client microservice which can resolve the physical address by searching the instance-id in the service discovery, it also takes care of distributing the requests to multiple instances
3. Distributed configuration: It is a centralized common configuration for multiple microservices
4. Security: It is to ensure only authenticated & authorized user can send the request using some security like OAuth2, JWT(JSON Web Token)
5. Circuit breaker: It breaks the flow of request when remote service is down to save other client microservice from going down.

and so on.

Spring Microservices:

It uses two spring project starters

1. spring cloud: provides all the required design patterns to create microservice
2. spring boot: auto-configures the application based on the annotations you use from spring cloud

Spring Cloud Library for the microservice design pattern

|  |  |  |
| --- | --- | --- |
| Design Pattern | Library Dependency | Annotation |
| Service Discovery | Eureka Server | @EnableEurekaServer |
| Microservice Program | Eureka Client | Optional: @EnableEurekaClient |
| Client Side Load Balancer | Eureka Client | @LoadBalanced |
|  |  |  |

Default behavior of every Eureka Client i.e., Microservice

1. Automatically registering to the Eureka Server
2. Searching Eureka Server in 8761 port by default
3. Fetching the registry or pinging the registry every 30s to notify its alive or sends heart beats every 30s

Programs we are going to create

1. Service Discovery: Needs eureka server
2. 2 microservice programs: Needs eureka client

Eureka Server: It gives a default dashboard for the users, who can see all the registered microservices using the http://ip:port of the eureka server i.e., http://localhost:8761/

How to create a microservice

Steps

1. Create a spring boot project
2. Add eureka client & web library
3. provide an Instance-Id (this will be registered in the service discovery) in application.properties

ex: spring.application.name = value

1. Create webservices using @RestController, @RequestMapping, and other annotations

Activity

create a new spring boot project add eureka client & web library, give an instance-id (second-ms) and register this in the service discovery as a microservice

* Create a webservice using the @RestController that return a simple message

Note: You will see 2 microservices in the eureka-dashboard

Things to summarize in spring annotations

* @Configuration
* @Bean
* @ComponentScan
* @Aspect
* @Component, @Repository, @Service, @Controller, @RestController
* @Autowired
* @SpringBootApplication
* @EnableEurekaServer
* @EnableEurekaClient

Note: AOP needs to be discussed

Day 19

1. Communication between the microservices using RestTemplate
2. AOP
3. HttpClient in Java
4. Node.js & ES6 features

RestTemplate: It is an object that is used to call the remote service, it provides inbuilt methods to make request using different HTTP methods, these inbuilt methods can convert Java to JSON & JSON to Java as well.

To create RestTemplate object

RestTemplate template = RestTemplateBuilderObject.build();

Sending a GET request to a remote web service

T t = template.getForObject(remoteURL, T.class);

if getForObject gets a JSON then T.class should have the properties matching to the JSON, so that getForObject converts the JSON to the object of T

i.e., if JSON is {“id”:100, “name”:”Alex”} then T.class must have 2 properties id & name so that getForObject creates an object T by initializing id & name to 100 & Alex.

Other methods of RestTemplate are

public T postForObject(url, javaObject, T.class): url is to access the webservice, the 2nd parameter is the java data the post method sends which will automatically be converted to JSON, then the response data will be converted T

HTTP PUT in RestTemplate

public T putForObject(url, javaObject, T.class)

HTTP DELETE in RestTemplate

public T deleteForObject(url, javaObject, T.class)

RestTemplate with Client Side Load Balancer

Since the client microservice uses the instance-id of the remote microservice, the RestTemplate must be marked with the client side load balancer which will resolve the physical address of the instance-id

@LoadBalanced  
RestTemplate template = RestTemplateBuilderObject.build();

Note: the above code doesn’t create RestTemplate, we need to use a different method to create RestTemplate and register in the spring container

What is the correct method to make spring container to create RestTemplate

@Bean  
@LoadBalanced  
public RestTemplate createRestTemplate(RestTemplateBuilder builder) {   
 return builder.build();   
}

@Bean: it registers the object in the spring-container

You can use @Autowired on RestTemplate in any place.

Activity:

1. In First Microservice we will return a json {“balance”: 25000}
2. In Second Microservice we will use @LoadBalanced RestTemplate and consume the JSON given by First Microservice, then will return another JSON to the user

Solution in First Microservice

Map<String, Double> map = new HashMap<>();  
map.put(“balance”, 25000);   
It will be converted to JSON

Solution in Second Microservice

* Create wallet class with balance & name property
* use the instance-id of the first microservice and send request using RestTemplate that converts the JSON of first microservice to wallet

Create a @GetMapping in the first microservice as below

// write below code in the @RestController which is already present  
@GetMapping(path = “/balance”, produces = MediaType.APPLICATION\_JSON\_VALUE)  
public ResponseEntity<Object> fetchBalance() {   
 Map<String, Double> map = new HashMap<>();  
 map.put(“balance”, 35000);  
 return ResponseEntity.status(200).body(map);  
}

Spring AOP

AOP stands for Aspect Oriented Programming, it is mainly used to add cross-cutting concerns in the code without writing it in the actual business logics / without calling them in the actual business logics

AOP terminologies

Aspect: The class having multiple advices

Advice: a method that will have cross cutting concerns (logics like logging, tracing, security)

Join Point: actual business logic method

Point cut: is an expression on the advice which must run on a particular join point like Before join point, After join point, Around join point(combination of before & after)

HttpClient:

Java 11 has released one API called HttpClient, it takes care of calling remote webservice

RestTemplate vs HttpClient

RestTemplate is released by Spring Framework and the methods will convert java to json and vice versa

HttpClient is released by Java and the methods will not automatically convert java to json and vice-versa

Creating HttpClient object

HttpClient client = HttpClient.newBuilder();  
HttpRequest request = HttpRequest.newBuilder();

// to send the request from the HttpClient use below code  
client.send(url, request).GET() // HTTP GET

client.send(url, request).POST(data) // HTTP POST

How to create a webservice in spring to handle POST request with some JSON data  
Assume the json coming in request-body: { “id” : 100, “name” : “Raj”, “salary”: 50000 }

@PostMapping(path = “/save”, consumes = MediaType.APPLICATION\_JSON\_VALUE)  
public ResponseEntity<Object> store**(@RequestBody** Employee employee) {   
 // @RequestBody creates employee object maps json property to employee object, so that   
 //you get a fully initialized employee object which you can pass to the service layer  
}

Node.js

It is a runtime environment to run javascript for the backend applications, node.js helps javascript to access backend resources like OS, Files, Database, Servers and many more.

Earlier Javascript was used only for the front-end, we had browser to run the Javascript, but now you can use Javascript for the back-end and node.js can run it.

Node.js has modules which will have reusable classes & functions

Note: You can create your own module and import it in any other javascript.

a.js

console.log(….);

To run use: node a.js

Day 20 agenda

* New Features of Javascript - ES6 New features
* React.js
* JSX
* Virtual DOM
* Components
* State & Props
* Event Handling

ECMA Script is a standard specification for Javascript, Node.js and many other scripting languages, it has released new features in the year 2015 with the name ES2015 or ES6 to simplify the syntax of the Javascript.

* let, const, class, constructor, extends, super
* template strings (back-tick)
* arrow functions - simplifies anonymous functions
* rest & spread operator.
* object & array de-structuring.

Arrow function: It is a replacement for the anonymous function/callback function, it simplifies the anonymous / callback function by reducing some lines

Old approach

let xhr = new XMLHttpRequest();

xhr.onreadystatechange = function() {   
 xhr.readyState == 4 // update the DOM  
}

Arrow function approach

xhr.onreadystatechange = () => {   
 xhr.readyState == 4 // update the DOM  
}

There are many inbuilt javascript functions / react.js functions which accepts anonymous function as a parameter, to them you can pass arrow function also.

example: forEach(callback), map(callback), these are the functions present in the array

forEach(callback) do the iteration

map(callback) do the iteration and also transforms each iterated item to another value and stores them in the array.

Example of arrow functions

(x, y) => x + y; // returns x + y  
(x, y) => { return (x + y) }; // same as above  
() => “hello”; // returns string hello

() => console.log(“….”); // doesn’t return but prints the console.log

React.js

Javascript library to develop UI’s for different types of applications like web, mobile

* It helps you to develop single page application
* It helps you to independently create reusable components

Components: These are reusable templates / reusable UI’s that can be used in other components

ex: In Facebook: Post component uses Profile, Comment component uses Profile

npx create-react-app app-name: It creates a react application

npm start: This is to launch the application

Creating components

function App() {   
 return (<div> some content </div>);  
}

<App />

React.js uses JSX to write HTML in Javascript

JSX: Javascript XML or Extension, it simplifies writing HTML in Javascript in a declarative way

React.js uses a transpiler called Babel to convert JSX to Javascript

React.js uses Virtual DOM to render the content which is faster in rendering

Note: Most of the technologies renders the content directly to the Real DOM which slows down the rendering time, because it rewrites the DOM tree behind the scene but in the front-end it looks like only certain content of the element is modified

How to share the data from one component to another component

Using props you can share the data from one component to another component

function Profile(props) {   
 <div>  
 <p> {props.name} </p>   
 <img src = {props.url} … />  
 </div>  
}  
function App() {   
 let text = “some comments”;  
 let username = “Alex”;  
 let imageUrl = “…..”;  
 <Profile name = {username} url = {imageUrl} />  
}

How to create the functions/classes outside the javascript and import them

components/posts.js

export function Profile(props) {  
  
}

export function Comment(props) {  
  
}

App.js

// to import Profile or Comment

import { Profile, Comment } from ‘.components/posts.js’;

Creating components using class

export class User extends React.Component {   
 constructor(props) { super(props); }  
 render() {   
 let username = this.props.firstname;  
 return content;  
 }  
}   
<User firstname = “Alex”/>

Conditional rendering:

Rendering the content based on the conditions

if(…) return content;

else return content;

How to add CSS

In React you can create .css file and import the css classes, but to use the class names you must use className attribute instead of class attribute because React uses HTML code inside the javascript

HTML:

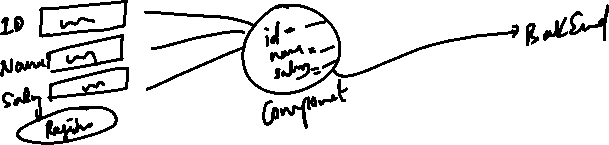
<h2 class = “success”>…….</h2>

React.js Components

<h2 className = “success”>…</h2>

Component states

These are data that can be updated in the component, there’s one more data in the component which are props that can’t be updated, props are read-only data.



Difference between normal variable and state variable

normal variables can be updated, but react doesn’t re-render the component, so user may not able to see the updates, however when states are updated, react re-renders the component, which means user can see the updates.

How to create state: We have a useState() function in React using which we can create the state and the function that can update the state

let [ name, setName ] = useState();

here name is the state/variable to store some value, setName is a function to modify the name

let [name, setName] = useState(“”); // default value of name will be “”

let [age, setAge] = useState(); // default value of age will be undefined

let [items, setItems] = useState([]); // default value of items is [ ];

What is the difference between

let [name, setName] = useState(“”);

&  
let age = 0;

Day 21 agenda

Flow of program execution : npm start

States

Event Handling

Forms

npm start:

npm start >> package.json >> scripts : react-scripts start.js >> config/paths.js >> load public/index.html

States:

These are the data components can store like User Input from the form, data coming from the backend service and etc.

States are updated using setter methods and its declared with the help of an inbuilt method useState

i.e.,

let [name, setName] = useState(“”); // name is a state, setName updates the name  
let [age, setAge] = useState(“”); // age is a state, setAge updates the age

let phone = 0; // phone is just a variable but it can’t be considered as state

How to update the state

To update the state you need to call their setter methods, however these methods are not automatically called, you need to either handle events to update them or access backend service to update them.

How to handle the event:

React has special kind of event called Synthetic events which are similar to HTML/DOM event names, but they follow camel case and start the first letter in lowercase

|  |  |
| --- | --- |
| HTML/DOM Events | React Event |
| onclick (or) ONCLICK (or) onClick | onClick |
| onmouseover or ONMOUSEOVER | onMouseOver |
| onchange | onChange |
| onsubmit | onSubmit |
| onblur | onBlur |

In react.js you need to attach an event handler which is a callback function that is executed when a particular event occurs

ex: onClick = ( ) => { …. }  
ex: onClick = (event) => { … }  
ex: onChange = ( ) => { … }  
ex: onChange = (event) => { … }

All the above events has a callback which are executed on those events, the parameter “event” is an Event Object generated, which is useful to know about the element that generated the event, it can have any name like “event”, “e”, “eve”, “x” and so on.

You can write callbacks in two ways

1st way  
onClick = { (event) => { … } }

2nd way

onClick = { handleClick }

let handleClick = (event) => { … }

Form Handling

You can store the form input in the component-state and pass it to the backend service, for the time being we will not submit the form data to the backend

<form onSubmit = { (event) => event.preventDefault() } >  
 <input type = “text” onChange = { event => setValue(event.target.value) }   
</form>

preventDefault(): prevents the browser to refresh on submitting the form, this needs to be done because React.js is used to create single page application and it must not refresh all the components in the page

You can also prevent the default behavior and access the backend using a named callback

let handleSubmit = (event) => {   
 event.preventDefault();  
 // pass the component state to the backend   
}

<form onSubmit = { handleSubmit }>  
  
</form>

Accessing the backend webservice/microservice

In React you must use axios library to access the webservice

Note: This library is not inbuilt in React, we must download it using npm command.

How to install

npm install axios

React can use axios to make HTTP calls using axios methods like  
axios.get(URL)  
axios.post(URL, data)  
axios.put(URL, data)  
axios.delete(URL, data)

All the 4 methods sends request but don’t know when it can get the response, there 2 methods you will call on top of those 4 methods to handle the response  
.then(callback) & .catch(callback)  
i.e., axios.get(URL).then(callback).catch(callback);

.then(callback): It is used if the response is successful, the callback in the then() is executed when response arrives i.e., status code 2xx  
.catch(callback): It is used if the response has errors, the callback in the catch() is executed when the error response arrives i.e., status code 4XX

CORS error: by default different domain can’t exchange the data, hence it blocks the request, which is called as Cross Origin Resource Sharing error, so we need to enable the CORS in the backend.

@CrossOrigin( origins = { “\*” }): This enables request to arrive from any origin

@CrossOrigin( origins = { “http://ip:port” } ): This enables only the particular origin to send the request

Implementing the webservice to maintain the Employee object

1. Employee - id, name, salary, 2 constructors, equals & hashCode, setters & getters, toString
2. Adding @CrossOrigin to enable the CORS in the @RestController
3. Implementing @PostMapping to accept JSON data
4. Implementing @GetMapping to returns all the employees in JSON format

Implementing the components to store and fetch the employee data from the backend

1. Install axios library - npm install axios
2. EmployeeForm must send HTTP POST request
3. EmployeeList must send HTTP GET request

axios.post(URL, data), axios.get(URL), axios.put(URL, data), axios.delete(URL, data): All these methods return a Javascript Promise object

Promise: It is to perform an action which could be resolved (success) or rejected(failed), it’s result can be identified using 2 functions then or catch.

then & catch accepts a callback function as an argument which is executed when the response arrives



axios.post(“api/employees”, { “id”:100, “name”:35000})  
.then( (response) => { … })  
.catch ( (error) => { … } )



Note: the arguments like (response) in then or (error) in catch can have any name

Creating a component that displays all the employee data in a table format

We must create ListEmployees components which sends

axios.get(url).then( callback )

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Salary | X |
| 100 | Alex | 35000 | Delete 100 |
| 200 | Bruce | 45000 | Delete 200 |
|  |  |  |  |

Day 22 Agenda

* Perform delete operation
* Calling the microservice from the React
* React Router
* React Hooks
* Summary on some important topics

Create webservice which can delete the employee based on id

Client >> sends id >> webservice takes that id and deletes the id

@DeleteMapping(path = “/employee/{ id }”)  
public ResponseEntity<Object> delete(@PathVariable(“id”) int id) {   
 // use id and delete the employee  
}

client sends the id as /employee/200, /employee/300 and so on

React Router

It makes your react application to render the components independently based on the link / button / navigation, so that you will not see all the components in a single page, instead you will navigation section based on the link you click your application can render the component.

npm i react-router-dom

This above command downloads the router library and it gives lot of components which you can use like <BrowserRouter>, <Link>, <Routes>, <Route> and so on.

BrowserRouter: Your entire application must be inside the browser router so that it can update only part of the browser instead of refreshing the entire browser

i.e.,  
<BrowserRouter>  
 <App />  
</BrowserRouter>

Link: It helps to create hyper links with url’s, these URL’s are for the components.

i.e.,

<Link to = “/store”>Registration</Link>  
<Link to = “/fetchAll”>Display Employee</Link>

Route: It is the component that will have the mapping of the URL & the component information, so that when Link generates the URL, Route is the one that will be load the component for the URL

i.e.,  
<Route path = “/store” element = { <EmployeeForm /> } />  
<Route path = “/fetchAll” element = { <EmployeeList /> } />

Note: For a given path you can load only one component, which means Route can load only one component

Routes: It is the container of Route, it is the one which selects the <Route> for the particular URL generated by <Link>, it is like a single container to load a particular component based on the Link

i.e.,

<Routes>  
 <Route path = “/store” element = { <EmployeeForm /> } />  
 <Route path = “/fetchAll” element = { <EmployeeList /> } />  
 ….  
</Routes>

Steps:

1. Add root component to the <BrowserRouter>
2. In Root component create <Link>, <Routes> & <Route>

Note: You can create the <Link>, <Routes> & <Route> in other components also

React hooks

These are some features which you can add to your components, these are inbuilt functions provided by React

ex:

useState(initialValue): to create states

useEffect(callback): This takes a callback which is executed when any state property is modified, you can use this useEffect() to perform validations, call the backend to fetch some information while state is updated

useEffect( () => {   
 if(id <= 0) {   
 // disable the submit button  
 }  
 // if id is already present display the error saying id already exists.  
});