Java Basics

1. What is Java?

Java is platform independent oop language.

2. What makes Java platform independent?

The java source code (.java) when gets compiled using javac compiler generates bytecode. Because of this bytecode, the Java language becomes platform indepenedent. Every machine has the jvm which will be able to understand the bytecode and transform it into machine code.

The **JVM** is platform **DEPENDENT** but the **bytecode** is platform **INDEPENDENT**.

The **jdk or jre** which we install on your machine is platform **dependent.**

3. What is the difference between JDK and JRE?

JRE is lighter than JDK.

JDK is used to create and run a Java Application.

JRE is used to run/execute a Java .class file(bytecode).

JDK=JRE+APIs

JDK folder also has a JRE folder in it.

JRE/bin we get to see the java application launcher.

JDK/bin we also get to see the javac compiler to compile Java scource code.

4. What is JVM?

It is a virtual machine environment provided by JRE.

Let us write our **first ever Java program**.

```
//prog.java**
class Demo{}
class Sample{
	static public void main(String[] ab){
		System.out.println("Hello friend");
	}
}
```

Points:

1. When we compile a Java source code that many number of .class

files will be created as class declarations are present.

2. If we donot provide any constructor, the javac compiler will

provide a default no argument constructor for us.

C:\Users\HP\Desktop\BatchesNow\ADM21JF036\MyJava>javap Demo

Compiled from "Prog.java"

class Demo {

  Demo();

}

3. The main() method is the entry point into the program.

C:\Users\HP\Desktop\BatchesNow\ADM21JF036\MyJava>java Demo

Error: Main method not found in class Demo, please define the main method as:

   public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

4. **

static modifier implies that this method will be available even if there is no instance of the class.

An **instance of the class** means an object which is the real time implementation of the class.

class Dog (pedigree, height, weight, furcolor,...)

GoldenRetriever, Labrador,Pug | each one is the real time example

of the Dog class or entity.

class Car (manufacturer, mileage, comfort,...)

Kia, i20| real time implementations of the class Car

public is the access modifier which tells that this method will be acessible to everyone.

void means that this method will not return any value.

main() is just a method name.

String[] is an array of String elements; command line arguement.

System.out.println("Hello friend"); is used to print something. Here **java.lang.System** is a class **out** is a variable and **println()** is a method

The **java.lang** is a package which is present by default.

A **package** is a collection of classes and interfaces which provides some functionality.

- **import java.util.Date;**

To print current Date I need an instance of Date class.Hence to include Date class which is part of java.util package we will add the above line.

String[] is a String array. String is a universal data type.

for(int i=0;i<arr.length;i++)

System.out.println(arr[i]);//ith position element in the array "arr".Here, arr.length shares the number of elements present in the array.

How to take user input?

java.util.Scanner class to accept user input.

```
import java.util.Scanner;
class Sample{
	//java application launcher calls this one
	static public void main(String[] ab){
		System.out.println("Hello friend tell me your name");
		Scanner scanner=new Scanner(System.in);
		String name=scanner.nextLine();
		System.out.println("Hello " +name);
	}
}
/* java.util.* will include all the classes hence heavy
java.util.Scanner implies I only want the Scanner class */
```

We created an instance of the class Scanner

```
	Scanner scanner=new Scanner(System.in);
```

Class is a template which shares the common behaviour and attributes. An instance is the real time implementation of the class.

Scanner scanner;

Here scanner is a reference variable(envelope) which is of type

Scanner but has no address written on it currently.

The new keyword is responsible for memory allocation in the JVM

heap memory.

scanner=new Scanner(System.in);

We did it here to associate the Scanner to standard input device

(keyboard).

System.in is a reference variable which is mapped to java.io.PrintStream class.

We can also use Scanner to read a File.

The Scanner class provides different methods to accept inputs of different types.

The java primitive data types (8)

byte(8), short(16), int(32) and long(64)

float(32), double(64)

char(16) -->Unicode charset

boolean --> true/false

String is a class which is part of java.lang package.

The nextXXX() method will be there for each pdt

nextBoolean() boolean

nextInt() int

nextFloat() float

* except for char

Looping styles in Java

```
for(int i=0;i<10;i++){
}
do{
i++;
}while(i<5);
while(i<5){
	i++;
}
```

Example:

```
import java.util.Scanner;
class Sample{
	static public void main(String[] ab){
		int[] arr={1,5,2,3,6};
		//advanced for loop
		for(int k:arr){
			System.out.println(k);
		}
	}
}
```

Switch case example in Java

Also supports String data type

```
import java.util.Scanner;
class Sample{
	static public void main(String[] ab){
		String something="tone";
		switch(something){
```

```
                default:
                        System.out.println("Invalid");
                        break;
                case "one":
                        System.out.println("One");
                        break;
                case "three":
                        System.out.println("Three");
                        break;
                case "two":
                        System.out.println("Two");
                        break;
        }
    }
}
```

We can **declare an array** in Java by writing any of the following

a) int[] arr=new int[size];

b) int[] arr={4,1,2,3,4};//array initializer block

In Java, an array can be of any primitive data type, wrapper type,String type as well as any user defined data type.

If you create an array using (a), it will automatically have

elements with default value based on that data type.

5. What is the use of wrapper type?

int has a wrapper type called Integer.

I cannot convert String to int and vice versa.

String -->Integer -->int

The **wrapper class** makes the pdt more powerful with its methods

When the value type is converted to reference type it is known as boxing. When the reference type is converted to value type, it is known as unboxing.

char -->Character

boolean -->Boolean

float -->Float

int -->Integer

Example:

```
import java.util.Scanner;
class Sample{
	static public void main(String[] ab){
		String s="5";
		//int num=s;//compile time error
		int num=Integer.parseInt(s);
		int x=21;
		//String t=x;//error
		String t=Integer.toString(x);
	}
}
```

We also have a java.util.Arrays class with a bag of utility methods.

Example:

```
import java.util.Arrays;
class Sample{
	static public void main(String[] ab){
	String[] arr={"abcd","ac","abbc","acbc","ad","aad","adac"};

		Arrays.sort(arr);//dual pivot quick sort algo
		for(String s:arr)
```

```
                System.out.println(s);


                Arrays.fill(arr,2,5,"xyz");
                System.out.println(Arrays.toString(arr));
        }
}
```

We can define as

a) static or class variable

b) instance variable

c) local variable

A change in the static variable value will be affecting all the instances. A change in the instance variable value is specific to an instance.

Example:

```
class Demo{
        String abc="one";//instance
        static String xyz="two";//static
        void someMethod(){
                String pqr="three";//local
        }
}
class Sample{
        static public void main(String[] ab){
                Demo de=new Demo();//instance
                Demo de1=new Demo();//instance
                de.abc="JF036 abc";
                de.xyz="JF036 xyz";

                System.out.println(Demo.xyz);//static
```

```
        System.out.println(de.xyz);//JF036 xyz
        System.out.println(de.abc);//JF036 abc

        System.out.println(de1.xyz);//JF036 xyz
        System.out.println(de1.abc);//one

        //System.out.println(de.pqr);//not visible
    }
}
```

6. What is the constructor?

A special method which has the same name as that of the class. It is used to initialize the instance variables to some non default values. If we do not provide a constructor, the compiler adds a default no arg constructor.If we add a constructor, the compiler does not any longer add any constructor.

A constructor does not have any return type defined in it.

It is recommended to add a default no arg constructor whenever you add a constructor with parameters.

Example:

```
class Person{
    String name;
    int age;
    Person(String name,int age){
        this.name=name;
        this.age=age;
    }
    Person(){
        this.name="Guest";
        this.age=5;
    }
```

```
    void print(){
    System.out.println(name+ " is "+age+" years old.");
    }
}

class Sample{
    static public void main(String[] ab){
        Person p1=new Person("Aman",20);
        Person p2=new Person();
        p1.print();
        p2.print();
    }
}
```

Eclipse IDE

IDE (Integrated Development Environment) used to create applications (console based, web based, web service and more)

A project contain multiple classes which makes a single application.

7. **How to create a Java Project?**

   File > New > Java Project

   You get to see the project under the "Package Explorer" view.

   Expand the project to find the src folder.

   Right click the src folder to add a new class.

   The class name is recommended to start with upper case letter.

   Syso[Ctrl+space] System.out.println("Hello");

   It will show the output on the Console Editor.If not visible, Window > ShowView > Console.

Java Naming Conventions

---

a) The class name should start with upper case letter.If it contains multiple words, each word's first letter should be in caps. Ex: ShoppingCart, Employee, MobileStore

b) The class name MUST BE A NOUN.

c) The method name MUST BE A VERB; some action register(), compute(), process(), signIn()

d) The method name should start with lower case letter.If it contains multiple words, second word's first letter should be in caps. Ex: addToCart(), saveEmployeeRecord(), editProfile()

e) The variable name should follow similar convention as method.

Ex: firstName,joiningDate,salary,paySlip,leaveBalance

f) The package name must be entirely in lower case.

Ex: java.util, java.lang, com.cognizant.model,com.cognizant.service

PMD (Programming Mistake Detector), CPD (Copy Paste Detector)

Points:

1. If a class is defined without public access modifier, we can

write it in a file with different name.

2. If a class is defined with public access modifier, we have to keep the class in the file with same name.

class Sample is public, should be declared in a file named Sample.java

3. In a source file, we can declare a package by writing

```
package com.project;
```

If present this line should be the first line in a file.

4. A package declaration can be followed by zero or more import

statements; to include other packages.

5. A class which is defined public can be accessed from any other class.

6. We declare a package to properly arrange multiple classes and interfaces related to common objective.

7. Other than public Java has 3 more access modifiers

private | Any resource which is private cannot be accessed outside the class.

package private aka default | Any resource which is package private can be only accessed from any class in the same package

but not outside the package.

protected | Any resource which is protected can be accessed from all the classes in the same package plus derived classes in a different package. Derived means having inheritance relationship(IS A)

Example:

```
package com.project;

public class Main {
	protected int secretCode=485;
}
```

```
package com.task;

import com.project.Main;

public class Demo extends Main{
	public static void main(String[] args) {
		Demo m=new Demo();
		System.out.println(m.secretCode);
	}
}
```

Concept of getters and setters

8. **Why??**

The getter and setter ae the two public methods which create a way to access the private data member defined in a class.

```
private int age;

//used to fetch the value

public int getAge(){
	return age;
}

//used to store the value

public void setAge(int age){
	if(age>0)
		this.age=age;
	else
		this.age=18;
}
```

```
private long minTemp;
public long getMinTemp(){}
public void setMinTemp(long minTemp){}
```

Add getter and setter without writing code manually in Eclipse.

Press [Alt+Shft+S] to generate getter and setter as well as constructors.

Example:

```
class Boiler {
	private long minTemp;
	public long getMinTemp() {
		return minTemp;
	}
	public void setMinTemp(long minTemp) {
		if (minTemp > 500)
			this.minTemp = minTemp;
		else
			this.minTemp = 500;
	}
}
```

Use of "this" keyword in "this.minTemp = minTemp;"

The this keyword implies current instance.

It is being used to resolve the ambiguity between local variable(parameter) and the instance variable when both of them have the same name.

this.minTemp =>instance variable

minTemp => parameter

String class and its methods

We can create a String instance either in one of the two ways:

a) String s=new String("java");

String ss=new String("java");

The memory will be allocated from the JVM heap and for each above line a new memory allocation will be done.

b) String t="java";

String tt="java";

There is a special area within the jvm heap known as String pool.

The JVM will first search the pool for an object with same content.

If not found, the jvm will create a object in the pool and assign

the address into the variable.

If found, the jvm will not create a object in the pool rather assign the address of the existing object into the variable.

MEMORY GETS SAVED.

9. **Why is String object IMMUTABLE?**

   What if I change the value of tt varible to 'oracle' then what will happen. will jvm assign new memory to tt varible?

   String tt="oracle";

   The original content will not be modified, rather the JVM will search the pool for the same content. If not found, will create an object with the content and assign back to the variable tt. So t will now refer to "java" and tt will now refer to "oracle".

Example:

```
public class Main {
        public static void main(String[] args) {
                String t="java";
                String tt="java";
                System.out.println(t==tt);//memory address
                System.out.println(t.equals(tt));//content
                tt="oracle";
                System.out.println(t==tt);//memory address
                System.out.println(t.equals(tt));//content
          System.out.println(t+" | "+tt);
```

```
	}
}
```

String class methods

Example:

```
public class Main {
	public static void main(String[] args) {
		String t="java program";
	System.out.println(t.charAt(0));
	System.out.println(t.endsWith("ram"));//startsWith(),contains()
	System.out.println(t.indexOf("a"));//first occurrence
	System.out.println(t.lastIndexOf("a"));//last or -1
	System.out.println(t.length());
	System.out.println(t.equals("Java"));
	System.out.println(t.equalsIgnoreCase("Java PRogram"));
System.out.println(t.trim());//removes spaces from beg or end
	}
}
```

Example 2:

```
public class Main {
	public static void main(String[] args) {
		String t="AB1DEF234K";
		//pan card
		/*
		 * first 5 alphabets,next 4 digits,last 1 alphabet
		 */
		String pattern="[A-Z]{5}[0-9]{4}[A-Z]{1}";
		System.out.println(t.matches(pattern));
		String mobPattern="[6-9]{1}[0-9]{9}";
		String colorPattern="[#]{1}[0-9A-F]{6}";
```

```
	}
}
```

Example 3:

```
public class Main {
	public static void main(String[] args) {
		String fdbck="orange-color|orange-fruit";
		String words[]=fdbck.split("\\|");
		System.out.println(words.length);
	}
}
```

Composition relationship (HAS A) Handle user defined array of objects

Trainee HAS A Laptop

Car HAS A Engine

Author HAS A Book

Example: Team HAS MUTIPLE Players.

Please refer to .java files attached

Interface

*Collection Framework|Exception Handling

Points:

a) Java supports single inheritance hence we can only inherit a single class but it also supports multi level inheritance.

It means a class inherits a base class which in turn inherits

another base class.

b) The base class contains the common attributes & behaviour(methods) to be shared/modified by the derived classes.

c) Only the non private data members are shared with the derived classes.

d) The derived class inherits only the instance methods and variables(not the static or final methods or variables).

e) A class when declared final cannot be inherited.

Ex: java.lang.String class is such an example

A method when declared final cannot be overridden.

A variable declared final cannot be modified.

f) The constructors are never inherited or overridden

g) Using super() we call the default no arg cons defined in the

immediate super class.

h) The java.lang.Object class is the root class in the Java hierarchy.

10. What is the use of super in base class?

Either directly or indirectly every class in Java inherits the java.lang.Object class. The super() call is used to invoke the base class default(no arg) constructor.

11. Why we need to override an inherited method?

We override an inherited method to modify or improvise on its working. We donot change the definition of the method but change the logic within the body of the method.

12. **Is there any other use of super keyword?**
    Yes, it can also be used to access the overridden method from the base class or to access a variable inherited from base class.

Example:

class Account also has the withdraw() method. In SavingsAccount class, you have overridden the withdraw() method. During that time, you have also tried to access the withdraw() method from base class (Account).

```
//overriding
public double withdraw(double amount) {
        if(balance-amount<=1000) {
                System.out.println("Insufficient Funds,Transaction Aborted");
        }else {
                super.withdraw(amount);
        }
        return balance;
}
```

The base class (Account)method is overridden

The **derived class** (Savings Account)method is overriding

13. What are the rules to be followed during method overriding?

a) The overriden and overriding method must have same name and same signature.

b) The access modifier of the overridden method should be same or lesser than the access modifier of the overriding method.

c) The return type of the overridden method should be same or

super class of the return type of the overriding method.

String to primitive data type is not possible.

String = PDT we take the help of wrapper class.

int num=Integer.parseInt("56");

String s=Integer.toString(48);

int has a Wrapper class for it called Integer.

Runtime Polymorphism

Polymorphism= Poly(multiple)+Morphos(Forms)

We have static polymorphism (method overloading) and runtime polymorphism (the method invocation which depends on the object's type)

We use the reference variable of base type (because every sub class IS A super type as well)

Every circle IS A shape

Every Dog IS A Mammal

displayArea(sq);// where sq is an object of type Square

The JVM will invoke the area() method of Square class.

displayArea(cr);// where cr is an object of type Circle

The JVM will invoke the area() method of Circle class.

Example:

```
class Shape{
	double dim;
	double area() {
		return 0;
	}
}
class Circle extends Shape{
	double area() {
		return Math.PI*Math.pow(dim,2);
	}
}
class Square extends Shape{
	double area() {
		return dim*dim;
	}
}
public class Demo {
	static void displayArea(Shape c) {
		System.out.println(String.format("%.2f",c.area()));
	}
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("[S]quare,[C]ircle,[T]rianlge");
		Shape shape=null;
		String opt=sc.nextLine();

		if(opt.equalsIgnoreCase("S")) {
			shape=new Square();
			System.out.println("Dimension");
			shape.dim=sc.nextDouble();
		}else if(opt.equalsIgnoreCase("C")) {
```

```
                shape=new Circle();
                System.out.println("Dimension");
                shape.dim=sc.nextDouble();
            }else if(opt.equalsIgnoreCase("t")) {
                shape=new Triangle();
                System.out.println("Dimension");
                shape.dim=sc.nextDouble();
            }

            //System.out.print("Area of Circle: ");
            //System.out.print("Area of Square: ");
            displayArea(shape);

            sc.close();
        }
}
```

We have used reference variable of super type which is assigned object of sub type based on some concept at runtime.

Use of instanceof operator to find if the reference variable is an instance of some derived class or the other.

```
public class Demo {
    static void displayArea(Shape c) {
        System.out.println(String.format("%.2f",c.area()));
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("[S]quare,[C]ircle,[T]rianlge,[R]ectangle");
        Shape shape=null;
        String opt=sc.nextLine();
```

```
                if(opt.equalsIgnoreCase("S")) {
                        shape=new Square();
                }else if(opt.equalsIgnoreCase("C")) {
                        shape=new Circle();
                }else if(opt.equalsIgnoreCase("t")) {
                        shape=new Triangle();
                }else if(opt.equalsIgnoreCase("r")) {
                        shape=new Rectangle();
                }
                System.out.println("Dimension");
                shape.dim=sc.nextDouble();

                if(shape instanceof Triangle) {
                        System.out.println("Base");
                        //downcasting
                        Triangle t=(Triangle)shape;
                        t.base=sc.nextDouble();
                }else if(shape instanceof Rectangle) {
                        System.out.println("Breadth");
                        Rectangle r=(Rectangle)shape;
                        r.breadth=sc.nextDouble();
                }

                displayArea(shape);
                sc.close();
        }
}
```

Explore **the java.lang.Object** class.

The Object class provides certain instance methods which every

other class does inherit and can override. The getClass() cannot be overridden because the method is declared final.

We can ovveride the

a) **toString()** || displays the class name along with @ symbol and hexadecimal represenation of the hashcode.We can override as per our concept.

b) **equals()** || matches the memory address allocated in the JVM heap and return boolean response. We can override as per our concept.

The String class and all the wrapper classes hve already overriden the equals() method.

Example:

```
class Person{
	String name;
	String city;
	public String toString() {
		return name+" says hello";
	}
	public Person(String name, String city) {
		super();
		this.name = name;
		this.city = city;
	}
	public Person() {
		super();
		// TODO Auto-generated constructor stub
	}
	public boolean equals(Object obj) {
		Person p=(Person)obj;
		if(p.name.equals(this.name) &&
				p.city.equals(this.city)){
			return true;
		}
		return false;
```

```
	}
}
public class Demo2 {
	public static void main(String[] args) {
		Person person1=new Person("Suresh","Kota");
		Person person2=new Person("Suresh","Kochi");
		System.out.println(person1.getClass());
		//Class@HexCode
		System.out.println(person1.toString());
		System.out.println(person1.equals(person2));//boolean
	}
}
```

Use of abstract keyword

a) When added to a method, we cannot complete the method in that

class. Any derived class which inherits the base class having such

a method MUST OVERRIDE the abstract method.

b) When added to a class, we can not create an object of abstract class.

- An abstract method cannot be final and a final method cannot be abstract.
- An abstract class can have non abstract methods, constructors defined in them.

Example:

```
abstract class Vehicle{
	abstract public void accelerate() ;
	Vehicle(){
		System.out.println("I am a vehicle");
	}
}
class Car extends Vehicle{
	@Override
	public void accelerate() {
```

```
            System.out.println("The car accelerates");
        }
}
public class Main3 {
        public static void main(String[] args) {
            Vehicle v=new Car();
            v.accelerate();
        }
}
```

Interface :

Interface is an aggrement which states the services which will be made available but not how they will be made available.

MenuCard exposes the food items you will get from the outlet but does not tell how will they be...

A class in Java can implement multiple interfaces; a way of having multiple inheritance.

a) Till JDK 1.7, an interface could only have public abstract methods.

b) From Java 8, an interface can also have non abstract (default)

as well as static methods.

c) The static method can only be accessed directly using the Interface name.

```
interface IFace{
        default void meth1() {
            System.out.println("non abstract meth");
        }
        void meth2();
        static void meth3() {
            System.out.println("static method");
        }
}
```

d) An interface without any methods declared is known as a MARKER interface. Ex:
java.io.Serializable interface

It is an empty interface (no field or methods)

e) An interface when has one and only one abstract method,it is

known as a FunctionalInterface. Ex: java.lang.Runnable which has only one abstract method public void run();

Such interfaces become eligible candidates for Lambda Expression.

f) A class can implement multiple interfaces but can extend only one class. A class first extends and then implements.

g) An interface can extend other interfaces.When a class implements an interface which again extends other interfaces,the class has to override all the abstract methods present in the interface.

```
interface IFace{
	void meth1();
}
interface IFirst extends IFace{
	void meth2();
}
class SomeClass implements IFirst {
	@Override
	public void meth1() {
		// TODO Auto-generated method stub
	}
	@Override
	public void meth2() {
		// TODO Auto-generated method stub
	}
}
```

Abstraction :

Abstraction is one of the pillars of OOP. We will expose the relevant details,hiding the irrelevant details.

Through an interface, exposing the services and hiding the implementation details.

Encapsulation :

By encapsulation, we hide the behavior/attributes from other classes.

14. Can't we use an abstract class here in place of an interface?

We can but still interface is a step better.

Collection Framework

An array is a collection of elements but is of fixed size.

We cannot add more elements that its size. Similarly, we cannot shrink the array all of a sudden.

In Java, we have the **java.util.Collection** interface which exposes the services:

a) add an element

b) remove an element

c) search for an element

d) get an element

e) find the size of the collection

f) check if the collection is empty or not

g) display the collection

h) clear the collection

java.util.Collection has two main sub interfaces : **java.util.List and java.util.Set**.

15. The basic differences between List and Set are

a) List allows duplicate elements whereas Set doesnot allow duplicate elements

b) List allows index based search and removal whereas Set does not

allow any index based operation.

- The List interface is implemented by the classes

  a) ArrayList b) LinkedList c) Vector

- ArrayList is based on resizable array concept.
- ArrayList instance, the JVM creates an array implicitly of size
- The Set interface is implemented by the classes

a) HashSet (has all the elements stored in random order) b) TreeSet (has all the elements stored in sorted order)

TreeSet does not allow any NULL value.

c) LinkedHashSet (has all the elements stored in insertion based order)

LinkedHashSet and HashSet: One NULL value is allowed.

Example

```
import java.util.*;
public class Main {
	public static void main(String[] args) {
		List<String> coll=new ArrayList<>();
	System.out.println(coll.add("abcd"));//boolean [true]
		coll.add("abbc");
		coll.add("babc");
		System.out.println(coll.add("abcd"));//true
		System.out.println(coll);//toString()
		System.out.println(coll.get(0));
		System.out.println(coll.remove(1));//"abbc"
		System.out.println(coll.remove("bcbc"));//boolean
		System.out.println(coll.contains("abbc"));
		System.out.println(coll.size());//int
		System.out.println(coll.isEmpty());//boolean
		coll.clear();//void
		System.out.println(coll.size());//0
	}
}
```

Example:

```
import java.util.*;
public class Main {
	public static void main(String[] args) {
```

```
                List<String> coll=new ArrayList<>();
                coll.add("abcd");
                coll.add("acbc");
                coll.add("abcc");
                coll.add("babc");
                //share all the elements to a Set
                Set<String> set=new HashSet<>();
                set.add("abcc");
                set.add("acbb");

                //set.addAll(coll);//union
                //set.retainAll(coll);//intersect
                set.removeAll(coll);//minus [b-a]
                // you can also perform the same
                //on a list
                System.out.println(set);
        }
}
```

java.util.Collections class which provides several utility methods like sort(), frequency()

Traversing a List in different ways...

Iterator interface works with a List or Set.

It is used to traverse a List or Set in forward direction.

It can also remove an element from the collection while traversing.

Example:

```
import java.util.*;
public class Main {
        public static void main(String[] args) {
                Integer[] arr= {5,1,2,3,6,4,1,2,3,7};
```

```
		List<Integer> list=Arrays.asList(arr);

		//Collections - utility class
		Collections.sort(list);//asc order
		System.out.println(list);

		Collections.sort(list,Collections.reverseOrder());//desc order
		System.out.println(list);

System.out.println(Collections.frequency(list, 2));
//returns no of times 2 is present.

		list=new ArrayList<>();
		list.add(5);list.add(2);
		list.add(3);list.add(7);
		list.add(6);list.add(2);

		//Iterator
		Iterator<Integer> it=list.iterator();
		while(it.hasNext()) {
			int num=it.next();
			System.out.println(num);
			if(num%2!=0)
				it.remove();
		}
		System.out.println(list);
	}
}
```

List Iterator :

- The **List Iterator** interface extends the Iterator interface.
- We can use it only on a List not a Set.
- It is used to traverse a List in both directions.
- It can add as well as remove an element from the collection while traversing.

Example:

```
import java.util.*;


public class Main {
	public static void main(String[] args) {
		List<Integer> list = new ArrayList<>();
		list.add(5);
		list.add(2);
		list.add(3);
		list.add(7);
		list.add(6);
		list.add(2);
		boolean flag = true;

		//add or remove during iteration
		//not possible
		/*for(int k:list) {
			System.out.println(k);
			if (flag) {
				//list.add(16);
				list.remove(2);
				flag = false;
			}
		}*/

		for (int i = 0; i < list.size(); i++) {
```

```
                System.out.println(list.get(i));
                if (flag) {
                        list.add(16);
                        list.remove(2);
                        list.remove(new Integer(2));
                        flag = false;
                }
        }

        /*ListIterator<Integer> it = list.listIterator(list.size());
        while (it.hasPrevious()) {
                int num = it.previous();
                System.out.println(num);
                if (num % 2 != 0) {
                        it.remove();
                }
                if (flag) {
                        it.add(16);
                        flag = false;
                }
        }
        */
        System.out.println(list);
    }
}
```

Map Interface :

java.util.Map is an interface which holds elements as key-value pairs.The keys can never be duplicate but values can be

duplicate.

There are **3 implementations** of the Map interface.

a) TreeMap which has the keys in sorted order

It cannot contain a null key.

b) LinkedHashMap which has the keys stored in insertion based order

c) HashMap which has the keys present in random order.

It can contain a null key as well as null as value.

map.put(k,v);

If we try to add the same key, the old value gets replaced with the new value.

Example:

```
public class Main {
	public static void main(String[] args) {
		Map<Integer, String> map = new LinkedHashMap<>();
		System.out.println(map.put(1, "apple"));//null
		System.out.println(map.put(1, "orange"));//apple
		System.out.println(map.putIfAbsent(1,"lemon"));//orange
		System.out.println(map);
		System.out.println(map.putIfAbsent(2,"lemon"));//null
		System.out.println(map);
		System.out.println(map.isEmpty());
		System.out.println(map.size());
		System.out.println(map.containsKey(2));
		System.out.println(map.containsValue("banana"));
	//	System.out.println(map.remove(1));
		System.out.println(map.remove(1,"grapes"));//false
		System.out.println(map);
		map.clear();
```

```
		System.out.println(map.isEmpty());
	}
}
```

16. How to traverse a Map instance?

Example:

```
public class Main {
	public static void main(String[] args) {
		Map<Integer, String> map = new TreeMap<>();
		map.put(1, "abcd");
		map.put(4, "acbd");		map.put(3, "cabd");
		map.put(5, "bcad");		map.put(2, "bbca");
		//System.out.println(map);//toString()

		//use keySet()
		Set<Integer> set=map.keySet();
		for(int k:set) {
		System.out.println(k+" | "+map.get(k));
		}

		//use entrySet()
		Set<Map.Entry<Integer,String>> set1=map.entrySet();
		for(Map.Entry<Integer, String> en: set1) {
	System.out.println(en.getKey()+" --> "+en.getValue());
		}
	}
}
```

Work with a collection of user defined objects

Class Book with the private data members [title, price]

Collection of such books...

Collections.sort(books);//error

Book is an udt;jvm is not aware about which attribute it should use to sort the books (title, price, ...).

Solution: To inform the JVM the criteria to be used for sorting. We will therefore implement the java.lang.Comparable interface in the Book class and override the compareTo(Object obj){} in the Book class. In this method, we will mention the logic for sorting.

```
@Override
public int compareTo(Book bk) {
        // TODO Auto-generated method stub
        return bk.price.compareTo(this.price);
}
```

All the Wrapper classes and String class has already overridden the compareTo() method.

bk.price.compareTo(this.price); //desc

this.price.compareTo(bk.price);//asc

When you call the sort() method on the Collections class the JVM checks if all elements in the list has implemented the Comparable interface.

- **Exploring the Date API (Java 7 & Java 8)**

-- Java 7 --

java.util.Date

java.util.Calendar

java.text.SimpleDateFormat

getTime() of Date class Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.

Example1:

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class Main3 {

	public static void main(String[] args) {
		// today's date
		Date today=new Date();
		System.out.println(today);

		//String input
		String joining="2022-01-27";
		//convert into Date object
		SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
		try {
		//parse() is used to convert String to Date reference
	Date joinDate=sdf.parse(joining);
	System.out.println(joinDate);

			//joinDate before or after today??
	System.out.println(today.after(joinDate));//before

			//diff btwn two dates
	long milis=today.getTime()-joinDate.getTime();
	System.out.println(milis/(1000*60*60*24)+" days");

		}catch(Exception ex) {
			System.out.println(ex.getMessage());
		}
```

```
	}
}
```

Calendar is an abstract class which has methods like add() to add or substract days/months/years from a Date reference.

Example 2:

```
import java.util.Calendar;
import java.util.Date;
public class Main3 {
	public static void main(String[] args) {
		// today's date
		Date today=new Date();
		System.out.println(today);

		Calendar cal=Calendar.getInstance();
		//instance of GregorianCalendar class
		cal.setTime(today);
		cal.add(Calendar.DATE, -6);
		Date revisedDate=cal.getTime();
		System.out.println(revisedDate);
	}
}
```

Java 8|

java.time.LocalDate

java.time.LocalTime

java.time.LocalDateTime

java.time.Period, java.time.Duration

....

Example:

```
import java.time.*;
import java.time.format.DateTimeFormatter;

public class Main4 {
	public static void main(String[] args) {
		LocalDate today=LocalDate.now();
		System.out.println(today);
		LocalTime current=LocalTime.now();
		System.out.println(current);

		String joining="2020-06-27";
		//String to LocalDate conversion
		DateTimeFormatter dtf=
				DateTimeFormatter.ofPattern("yyyy-MM-dd");
		LocalDate joinDate=LocalDate.parse(joining,dtf);
		System.out.println(joinDate);

		System.out.println(joinDate.isAfter(today));
		System.out.println(joinDate.isLeapYear());
		System.out.println(joinDate.plusDays(6));
		System.out.println(joinDate.minusMonths(5));

		//diff btwn to dates
		Period period=Period.between(joinDate,today);
		System.out.println(period);
		System.out.println(period.toTotalMonths()+" months & "+period.getDays()+" days");

		//LocalTime
		System.out.println(current.plusHours(3));
	} }
```

- **Exception Handling :**

What is an exception?

An exception is an abnormal situation which disrupts the flow of the program.

We are not going to try to stop exception, we will try to handle the exception so that the program finishes gracefully.

- In Java, there is a class hierarchy to handle exceptions..

java.lang.Object

java.lang.Throwable

java.lang.Exception

java.lang.RuntimeException
java.lang.NullPointerException

java.lang.ArithmeticException

java.lang.NumberFormatException
java.lang.ArrayIndexOutOfBoundException

java.io.IOException

java.sql.SQLException

java.lang.Error

Difference between Error and Exception

Error are irrecoverable exceptions such as VirtualMachineError, OutOfMemoryError, StackOVerflowError Exceptions are recoverable. We can continue with the application even after the exception has occurred by properly handling them.

What is the difference between checked and unchecked exception?

If the exception class extends the Exception class then it falls under the category of checked exception. When you use such a method for which the compiler forces you to provide either a try-catch block to handle the exception or add throws clause to delegate the exception.The checked exceptions are where the methods can fail because of external factors.

Ex1:

```
public static void main(String[] args) {
        String str="2022-05-15";
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        try {
```

```
                Date date=sdf.parse(str);
                System.out.println(date);
            } catch (ParseException e) {
                System.out.println(e.getMessage());
            }
}
```

Ex2:

```
public static void main(String[] args) {
            try {
                FileReader fr=new FileReader("D://myfile.txt");
            } catch (FileNotFoundException e) {
                System.out.println(e.getMessage());
            }
        }
```

- If the exception class extends the RuntimeException class then it falls under the category of unchecked exception.
- The compiler does not force you to add any such blocks.

Ex 3:

```
class Person{
        String name;
}
public class Main {
        public static Person fetch(int id) {
            return null;
        }
        public static void main(String[] args) {
            Person p=fetch(105);
            if(p!= null)
                System.out.println(p.name);
```

```
            else
                System.out.println("No record found");
        }
}
```

- ## Try, Catch and Finally Keywords :

a) handle the exception using try, catch and finally keywords

- try block will contain the statements which may raise an exception
- catch block contains statements to handle the exception if occurred
- finally block contains statements to close the resources irrespective of the fact that exception has occurred or not.

```
TRY [
    OPEN A TAP
    -WATER FLOWS
    -NO WATER
]CATCH[
    CHECK THE RESERVOIR SWITCH ON THE PUMP
]
FINALLY[
     CLOSE THE TAP
]
```

What is the difference between final and finally?

We use final keyword to make a class not inheritable or a method not overridden in the derived class or a variable constant in Java.We use finally block to close any resources whether the situation is normal or exceptional.

Points:

a) try block is always accompanied by catch(n), finally or both.

b) try-catch-finally can also be nested.

c) A catch block with Exception reference MUST BE THE LAST CATCH BLOCK.

Ex4:

```
import java.io.*;
public class Main {
	public static void main(String[] args) {
	BufferedReader br = null;
		try {
	FileReader fr = new FileReader("myfile.txt");
	br = new BufferedReader(fr);
		String s = br.readLine();
		while (s != null) {
		System.out.println("Read: " + s);                                    s = br.readLine();
		}
		} catch (IOException fnfe) {
			System.out.println(fnfe.getMessage());
		} catch (Exception fnfe) {
			System.out.println(fnfe.getMessage());
		} finally {
			try {
				if (br != null)
					br.close();
			} catch (IOException e) {
				System.out.println(e.getMessage());
			}
		}
	}
}
```

b) delegate the exception using throw and throws clause

When we actually raise the exception, we use throw

We create an object of that Exception class.

A method can declare mutiple exception classes separated by comma after throws clause

Ex5:

```
public class Main {
	static void readLoud(String fileName) throws IOException,ParseException {
		SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
		sdf.setLenient(false);
		BufferedReader br = null;
		FileReader fr = new FileReader(fileName);
		br = new BufferedReader(fr);
		String s = br.readLine();
		while (s != null) {
			Date dt=sdf.parse(s);
		System.out.println("Read: " + dt);
			s = br.readLine();
		}
			br.close();
	}
	public static void main(String[] args) {
		try {
			readLoud("myfile.txt");
		} catch (IOException fnfe) {
			System.out.println(fnfe.getMessage());
		} catch (Exception fnfe) {
			System.out.println(fnfe.getMessage());
		} finally {

		}
	}
}
```

StringBuilder class creates mutable references unlike String class. StringBuffer class also provides similar methods as StringBuilder but the methods are synchronized which means thread safe.

Ex 1:

```
public class Main2 {
	public static void main(String[] args) {
		StringBuilder sb = new StringBuilder();
		sb.append("Java ")
		.insert(0, "Welcome to ")
		.append("programming")
		.delete(sb.length()-3,sb.length()).reverse();
		System.out.println(sb);// toString()
	}
}
```

- **new StringBuilder(String str);**

  Constructs a string builder initialized to the contents of the specified string. The initial capacity of the string builder is 16 plus the length of the string argument.

Ex:

```
public class Main2 {
	public static void main(String[] args) {
	StringBuilder sb = new StringBuilder("java");
	System.out.println(sb.capacity());// 16 +length of string passed
	sb.append(" is a object oriented");
	System.out.println(sb);
	System.out.println(sb.capacity());//oldcap*2+2
	}
}
```

Garbage Collection

Java provides inbuilt mechanism to collect and deallocate the memory of unused objects from the heap of JVM.

We can pass a request to the JVM for executing the gc process in either of the two ways:

a) Runtime.getRuntime().gc()

b) System.gc()

Runs the garbage collector.

Calling the gc method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse.

Ex1:

```
class Person{}
public class Main2 {
	public static void main(String[] args) {
		Runtime runtime=Runtime.getRuntime();
System.out.println("A| "+runtime.freeMemory()/1024+" kb");
		for(int i=1;i<=100000;i++) {
			Person p=new Person();
		}

System.out.println("B| "+runtime.freeMemory()/1024+" kb");		System.gc();
System.out.println("C| "+runtime.freeMemory()/1024+" kb");
	}
}
```

Whenever the gc process runs, it implicitly calls the finalize() method. We get this method inherited from the Object class which we can also override. The utility of this method is do clean up operations.

Ex 3|

```
class Person{
	@Override
	protected void finalize() throws Throwable {
```

```
		System.out.println("finalize called");
		super.finalize();
	}
}
public class Main2 {
	public static void main(String[] args) {
		Runtime runtime=Runtime.getRuntime();
		Person p=null;
		for(int i=1;i<=3;i++) {
			p=new Person();
			System.out.println(p);
		}
		System.out.println(p);
		p=null;
		System.gc();
	}
}
```

- **Annotations**|
  These are instructions to the compiler or the virtual machine or both

Examples:

@Override | Indicates that a method declaration is intended to override a method declaration in a supertype.

@Deprecated | A program element annotated @Deprecated is one that programmers are discouraged from using, typically because it is dangerous,or because a better alternative exists. Compilers warn when a deprecated program element is used or overridden in non-deprecated code.

@SupressWarnings |Indicates that the named compiler warnings should be suppressed in the annotated element

@FunctionalInterface |An informative annotation type used to indicate that an interfacetype declaration is intended to be a functional interface

Ex 1:

```
class Person{
```

```
	String name;
	Person(String name){
		this.name=name;
	}
	@Deprecated
	Person(){
		this.name="guest";
	}
	@Override
	public String toString() {
		return (name+ " says Hello!!");
	}
}
public class Main2 {
	public static void main(String[] args) {
		Person p=new Person();
		System.out.println(p);//toString()
	}
}
```

Ex 2:

```
@FunctionalInterface
interface IFace{
	public void meth();
	public default void meth1() {}
}
public class Main2 {
	public static void main(String[] args) {
	@SuppressWarnings("rawtypes")
	List list=new ArrayList();
	//ArrayList is a raw type. References to generic type
```

```
        //ArrayList<E> should be parameterized


        }
}
```

- **Singleton Design Pattern**

According to this pattern, we will only be able to access a single instance of the class.

For example java.lang.Runtime class follows Singleton pattern.

Runtime runtime=Runtime.getRuntime();

1. Create a static reference of the class.
2. Make the no arg constructor private.
3. Create a static method to access the reference object of the class.

Example:

```
//singleton pattern
public class DBConn {
    private static DBConn instance=null;
    private DBConn() {        }
    public static DBConn getInstance() {
            if(instance==null) {
    System.out.println("object created");
                    instance=new DBConn();
            }else {
    System.out.println("object already present");;
            }
            return instance;
    }
}
```

- **Lambda Expressions**
  - Lamdba Expression is a way of implementing Functional Programming in Java
  - For writing Lambda Expression, we need a functional interface.
  - It is an interface which has one and only one abstract method defined in it.

Example:

```
interface IFace{
	int compute(int a ,int b);
}
class Addition implements IFace{
	public int compute(int x,int y) {
		return x+y;
	}
}
class Product implements IFace{
	public int compute(int x,int y) {
		return x*y;
	}
}
```

- **Example: Anonymous class** |
  - A nested class without a name.
  - Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name. Use them if you need to use a local class only once.

```
interface IFace{
	int compute(int a ,int b);
}
public class Main {
	public static void main(String[] args) {
		IFace iface=new IFace() {
			public int compute(int x,int y) {
				return x+y;
```

```
            }
        };
        System.out.println(iface.compute(5, 6));

        iface=new  IFace() {
            public int compute(int x,int y) {
                return x*y;
            }
        };
        System.out.println(iface.compute(5,6));
    }
}
```

Points:

- Example: Java 8 we will use Lambda Expressions
- Lambda expressions enable you to do this, to treat functionality as method argument, or code as data.

```
interface IFace{
    int compute(int a ,int b);
}
public class Main {
    public static void main(String[] args) {
        IFace iface=(int x,int y) -> {
            return x+y;
        };
        System.out.println(iface.compute(5, 6));

        iface=(x,y) ->(x*y);
        System.out.println(iface.compute(5,6));

        iface=(x,y) ->Math.abs(x-y);
```

```
        System.out.println(iface.compute(5,6));
    }
}
```

FYR: https://www.geeksforgeeks.org/anonymous-inner-class-java/

- **Another real time example of Lambda Expression**

java.util.Comparator interface: Example of functional interface. A comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order.

int compare(T o1, T o2);

```
Comparator<Book>
    int compare(Book b1, Book b2);
Comparator<Book> comp=(x,y)->(x.getTitle().compareTo(y.getTitle());
    Collections.sort(books,comp);
```

- All the Wrapper clases and String class have already overridden the compareTo() method from the java.lang.Comparable interface.

Example:

Assuming that you have the class Book with private data members

String title and Double price.

```
public class Main3 {
    private List<Book> books = new ArrayList<>();
    Main3() {
    books.add(new Book("Complete Java", 226.52));
    books.add(new Book("ABC of Java", 162.57));                    books.add(new Book("Learn Java", 284.02));
    books.add(new Book("How to code in Java", 339.0));
    books.add(new Book("Code in Java", 195.20));
```

```
	}
	public void printAll() {
	for (Book bk : books) {
System.out.println(bk.getTitle() + " | " + bk.getPrice());
	}
	}

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("1. Title Asc,2. Title Desc");
		System.out.println("3. Price Asc,4. Price Desc");
		String opt = sc.nextLine();
		Comparator<Book> comp=null;
		switch (opt) {
		case "1":
			comp=(b1,b2)-> b1.getTitle().compareTo(b2.getTitle());
			break;
		case "2":
			comp=(b1,b2)-> b2.getTitle().compareTo(b1.getTitle());
			break;
		case "3":
			comp=(b1,b2)-> b1.getPrice().compareTo(b2.getPrice());
			break;
		case "4":
			comp=(b1,b2)-> b2.getPrice().compareTo(b1.getPrice());
			break;
		}
		Main3 m3=new Main3();
		Collections.sort(m3.books,comp);
		m3.printAll();
		sc.close();
```

```
	}
}
```

```
class Book implements Comparable<Book>{
	public int compareTo(Book bk){
		return this.title.compareTo(bk.title);
	}
}
```

```
Comparator<Book>
	public int compare(Book b1,Book b2){
		return b1.getPrice().compareTo(b2.getPrice());
	}
```

- **Stream API**
  - **java.util.stream package**

    - It makes working with arrays and collections easier.

Way using Stream API

```
import java.util.*;
import java.util.stream.*;

public class Main4 {
	public static void main(String[] args) {
		String[] arr= {"p","a","b","c","a","d","c","e","t","k"};
		Stream<String> stream=Arrays.stream(arr);
		stream
		.distinct()
		.sorted(Comparator.reverseOrder())
```

```
                .forEach(p -> System.out.println(p));


                System.out.println(Arrays.toString(arr));
        }
}
```

Stream API :

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

There are two types of operations:

a) intermediate operation like .distinct(),.sorted(),.filter() because they return a new Stream reference for further operation.

b) terminal operation like .count(), .collect(), .forEach() because they do not return a new Stream but are either void or return some other data type.

A stream can have multiple intermediate operations but only one terminal operation.

Points:

a) Stream takes input from the Collections, Arrays

b) Streams don't change the original data structure, they only provide the result as per the pipelined methods.

c) Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

- **Java IntStream** class is an specialization of Stream interface for int primitive. It represents an stream of primitive int-valued elements.

Ex 1:

```
public class Main4 {
        public static void main(String[] args) {
                //array of primitives
                int[] intArray= {1,2,5,1,2,3,6};
                IntStream stream=Arrays.stream(intArray);
                stream
                .distinct()
                .filter(y ->y%2==0)
```

```
                .forEach((u)->System.out.println(u));
        }
}
```

Ex 2:

```
public class Main4 {
        public static void main(String[] args) {
                //List of Strings
                List<String>
myList=Arrays.asList("abcd","abcb","accb","babc","bbca","bcbc","dadc","ddca");
                Stream<String> stream=myList.stream();
                long cnt=stream
        .filter(h-> (h.startsWith("a") && h.endsWith("b")))
        .count();
                System.out.println(cnt);

                stream=myList.stream();
                stream
                .filter(h-> (h.startsWith("a") && h.endsWith("b")))
                .map((y)->y.toUpperCase())
                .forEach(k-> System.out.println(k));
        }
}
```

Ex 3:

```
public class Main5 {
        public static void main(String[] args) {
                Main3 m3=new Main3();
                Stream<Book> bookStream=
                                m3.getBooks().stream();
```

```
		List<Book> sortedList=bookStream
			.filter(b->b.getPrice()>=250.0)
			.sorted((b1,b2)->b1.getPrice().compareTo(b2.getPrice()))
			//.forEach((h)->System.out.println(h.getTitle()+" - Rs."+h.getPrice()));
			.collect(Collectors.toList());

	for(Book b:sortedList) {				System.out.println(b.getTitle()+"
:"+b.getPrice());
		}
	}
}
```

Ex 4:

```
public class Main5 {
	public static void main(String[] args) {
		Main3 m3=new Main3();
		Stream<Book> bookStream=
				m3.getBooks().stream();

		Optional<Book> book1=bookStream
			.sorted((b1,b2)->b1.getPrice().compareTo(b2.getPrice()))

			.findFirst();

		Book bk=book1.get();
		System.out.println(bk.getTitle()+" "+bk.getPrice());

	}
}
```

- **File Handling**

- java.io System.out and System.in are static reference variables defined in the java.lang.System class which refers to java.io.InputStream and java.io.PrintStream classes respectively. System.err which also refers java.io.PrintStream class.
- Stream implies a source or a sink from which/to which the data flows.
- In Java, we have 2 types of stream, byte stream and character stream.
  - Byte stream is represented by the abstract classes InputStream and OutputStream respectivly. They provide the abstract method read() and write() respectively.
  - A FileInputStream obtains input bytes from a file in a file system. FileInputStream is meant for reading streams of raw bytes such as image data.
  - FileOutputStream is meant for writing streams of raw bytes such as image data.

Example:

```
import java.io.*;


public class Main {
	public static void main(String[] args) throws IOException {
		FileOutputStream fos=new FileOutputStream("myfile.txt",true);
		fos.write(65);//'A'
		String str="java";
		fos.write(str.getBytes());
		fos.close();

		FileInputStream fis = new FileInputStream("myfile.txt");
		int k = 0;
		k = fis.read();
		while (k != -1) {
			System.out.println((char)k);
			k = fis.read();
		}
		fis.close();
	}
}
```

- **Character Stream** has two main abstract classes: Reader and Writer used for reading from/writing to character streams.
- **FileReader** is meant for reading streams of characters.
- **BufferedReader** : Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters.
- **FileWriter** is meant for writing streams of characters.

Example:

```
import java.io.*;
public class Main2 {
	public static void main(String[] args) throws IOException {
		FileWriter fw=new FileWriter("myfile.txt",true);
		String str="Java program on Files";
		fw.write(str);
		fw.write(System.lineSeparator());
		//On UNIX systems, it returns "\n";
		//on MicrosoftWindows systems it returns "\r\n".
		fw.close();

		FileReader fr=new FileReader("myfile.txt");
		BufferedReader br=new BufferedReader(fr);
		String s=br.readLine();
		while(s!= null) {
			System.out.println("Read: "+s);
			s=br.readLine();
		}
		  br.close();
	}
}
```

- **How to accept user input?**

a) Use java.util.Scanner class

b) Use java.io.InputStreamReader and java.io.BufferedReader combination.

- An **InputStreamReader** is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.

Example:

```
import java.io.*;
public class Main3 {

	public static void main(String[] args) throws IOException {
		InputStreamReader isr=new InputStreamReader(System.in);
		BufferedReader br=new BufferedReader(isr);
		String s=br.readLine();

		while(s!= null) {
			System.out.println("Read: "+s);
			s=br.readLine();
		}
	}
}
```

- **How to read a .properties file?****
- A properties file is used to write configuration information.
- We use java.util.Properties class to read the .properties file.
- Generally, these files are used to store static information in key and value pair separated by = sign. Things that you do not want to hard code in your Java code goes into properties files. The advantage of using properties file is we can configure things which are prone to change over a period of time without the need of changing anything in code. Properties file provide flexibility in terms of configuration.

Example:

```
import java.io.*;
import java.util.Properties;
public class Main4 {
	public static void main(String[] args) throws IOException {
		FileInputStream fis=new FileInputStream("database.properties");
```

```
        Properties props=new Properties();
        props.load(fis);
    System.out.println(props.getProperty("username"));
    System.out.println(props.getProperty("password"));
    System.out.println(props.getProperty("driver"));
    }
}
```

- **Thread basics**
  - A thread is a light weight process. A process comprise of multiple threads.
  - In Java, we can create custom threads either by

a) extending the java.lang.Thread class

b) implementing the java.lang.Runnable interface.

  - The option (b) is preferred When we create any Java program and try to execute it, the jvm creates the first thread known as main.
  - Thread.currentThread() | static method :: Returns a reference to the currently executing thread object.
  - The start() is called : Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.
  - The result is that two threads are running concurrently: thecurrent thread (which returns from the call to the start method) and the other thread (which executes its run method).
  - The start() method should not be called more than once using same thread reference. If tried, we get IllegalThreadStateException.

There are 4 main states of a thread

a) NEW

b) RUNNABLE

c) BLOCKED/WAITING

d) TERMINATED

Example:

```
public class Main5 {
    public static void main(String[] args) throws InterruptedException {
        System.out.println(
    Thread.currentThread().getName()+" begins....");
```

```
Runnable r=()->System.out.println(
        Thread.currentThread().getName());


//It will use the stack of the main thread
                //r.run();


//The custom thread will have its own stack
                Thread th=new Thread(r);
                //th.setPriority(Thread.MAX_PRIORITY);//10
                th.join();
//The custom thread will make its creator (main thread)
                //to wait for it to die.
                System.out.println("A| "+th.getState());
                th.start();


                System.out.println("B| "+th.getState());
                System.out.println(Thread.currentThread().getName()+" continues...");


                System.out.println("C| "+th.getState());
                System.out.println(Thread.currentThread().getName()+" finishes...");
        }
}
```

- The Thread.sleep(long milis) is a static method to make the thread go to blocked state for a given number of miliseconds.
- Once the time is over, the thread becomes eligible to run (moves into RUNNABLE state). During the BLOCKED state, it does not release any resources held.

- **JDBC (Java Database Connectivity)**

We will make our Java application to connect to an underlying database which can be mysql or Oracle ...

- Analogy
  - Smart Phone <--> Java Application
  - Put the simcard into the slot of the smart phone.
  - SimCard <--> Connector Jar file
  - Every database vendor has a separate connector jar file.
  - We will download the jar file from Maven central repository.
  - Add the jar file to your java application.
  - Right click your java application
  - Go to Build Path >> Configure Build Path
  - Go to Libraries [tab]
  - Click [Add External JArs] button
  - Locate the jar file and click [Finish] button
  - The jar file will get added to your project.
  - Activate the network <--> Establishing the Connection to some underlying database.

We will open mysql server workbench and figure out a table to connect.

Create a table "student" in any db of your choice...

```
create table student
(
roll int primary key auto_increment,
firstName varchar(50) not null,
lastName varchar(50) not null,
birthDate date not null,
gender char(6) not null
);
```

Points:

a)We require to import java.sql package

b)We need the class DriverManager whoch provides a method

called getConnection(). This method accepts 3 parameters

and returns the Connection object.

c) The Connection is an interface so we actually get an object

of the implementation class whose named is hidden from us (abstraction).

d) We will creata a .properties() file with the values for those 3 parameterts mentioned above. Right click your project

>> New >> File and give any name .properties extension.

Ex: db.properties

---

username=root

password=root

url=jdbc:mysql://localhost:3306/jf036

driver=com.mysql.cj.jdbc.Driver

e) Read the properties file and create the Connection object.

```
public static void main(String[] args) {
		Connection cn=null;
		try {
		//create a FileInputStream object
		FileInputStream fis=new FileInputStream("db.properties");
		Properties props=new Properties();
		props.load(fis);
		//load the Driver class
		Class.forName(props.getProperty("driver"));
		//get the Connection instance
		cn=DriverManager.getConnection(
				props.getProperty("url"),
				props.getProperty("username"),
				props.getProperty("password"));

		System.out.println(cn);

		}catch(Exception e) {
			System.out.println(e.getMessage());
```

```
        }
    }
```

Start a call <--> Write a query to insert a new record into the student table.

CreateReadUpdateDelete operations

We have 2 main interfaces:

a) java.sql.Statement

- used for simple static queries

```
select firstname,lastname from student
where firstname like 'A%';
```

b) java.sql.PreparedStatement

- used for parameterised queries which are dynamic in nature

```
insert into student(firstName,lastName,birthDate,gender)
values(?,?,?,?);
```

We will now try out a SELECT query ...

```
select firstname,lastname from student;
```

The executeQuery() method returns a ResultSet reference.

What is it? A ResultSet is an inmemory representation of the

database table. We get a ResultSet when we execute a SELECT query. This ResultSet is by default FORWARD ONLY and READ ONLY.

ice cream freezer vs a cup of icecream and a spoon

table vs result set (result set cursor which points outside the result set. When you call result set's next() method the cursor moves to the first row in the result set (you dip the spoon into the cup of ice cream). Now every call to the next() method moves it forward to the upcoming rows untill we finish traversing and next() method returns false.

Lets see another example of SELECT query...

```
select firstname,lastname,birthdate, gender from student
where year(birthdate)=?
```

- **JUnit testing**

17. What is unit testing? Why we will do it?

- We perform unit testing to test a smallest code block in isolation to figure out whether it is giving the proper output in all situations. The unit testing is done by the coder himself.
- Junit is the most popular testing framework for testing Java codes. There is another framework TestNG.
- We will test using Junit version 4.0
- We will require certain jar files for getting hold of JUnit.
- Eclipse IDE has inbuild support for JUnit.
- Create a new Java project..
- Right click your project >> Build Path >> Configure Build Path
- Select [[Libraries]] tab, click [Add Library] button
- Chose JUnit 4 and [Apply and Close]

SUT(System Under Test),CUT(Class Under Test)

Calculator

PersonService

The name of the test class is preferred to be:

CalculatorTest

PersonServiceTest

Expected Output || Actual Output

(If both are same or equal the test case passed else test case fails)

@Test annotation |

- The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.
- A public void method annotated with @Before or @After annotation will be called once before and after each test method starts and completes its execution respectively.
- Sometimes several tests need to share computationally expensive setup(like logging into a database). While this can compromise the independence oftests, sometimes it is a necessary optimization. Annotating a public static void no-arg method with @BeforeClass or @AfterClass causes it to be run once before any of the test methods in the class and once after all the test methods in the class.
- @Ignore annotation when placed with any @Test method will be ignored by the JUnit Runner class.

18. How to run multiple test files together in one single go?

```
import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

@SuiteClasses({CalculatorTest.class,PersonServiceTest.class})

public class MySuite {


}
```

Suite class|Using Suite as a runner allows you to manually build a suite containing tests from many classes.

How to test if an exception is being thrown by a method under test?

```
@Test(expected=CustomException.class)

public void testThree() throws CustomException {
        assertNull(service.fetch(111));

}
```

How to create category in Junit?

We will create some marker interfaces called Sample and Hidden

Annotate the existing test methods in the test class

as follows:

```
import org.junit.experimental.categories.Category;

        @Test

        @Category({Sample.class,Hidden.class})

        public void testValidateTwo() {

                assertTrue(calc.validateEmail("abc@cognizant.com"));

        }


@RunWith(Categories.class)
```

```
@SuiteClasses({CalculatorTest.class})

@IncludeCategory({Hidden.class})

public class CatSuite {


}
```

- **Mockito Framework :**

  - Create a new Java project
  - Add the 4 files to your src folder. Save the code and execute the Main class.

    Student

    StudentService

    StudentRepository | interface

    Main
  - The reason behind the exception is that the impl class is not available.

19. How will the coder perform junit testing on the StudentService class in absence of StudentRepositoryImpl class?

- I will arrange a mock implies that I will assume that StudentRepositoryImpl is there... Hence, we will use Mockito framework along with JUnit modules. Unlike JUnit jars, Mockito jar file is not already available with Eclipse IDE.
- So we will download the jar from Maven central repository. https://mvnrepository.com/artifact/org.mockito/mockito-all/1.9.5
- We will add the jar to the project in eclipse along with junit
- Right click your project
- Build Path >> Configure Build Path
- Select [Libraries] tab
- Click [Add Library] to add JUnit 4 jar file
- Click [Add External Jars] to add mockito-all jar file into the project
- We will add a new Java class for testing StudentService class

```
import static org.mockito.Mockito.when;

public class StudentServiceTest{
```

```
@Mock
StudentRepository repo;

@InjectMocks
StudentService service;

@Before
public void before(){
	//MockitoAnnotations.initMocks(this);
}

@Test
public void test1(){
 when(repo.getMarks(101)).thenReturn(82);
assertEquals("B",service.computeGrade(new Student(101,"Sam")));
	verify(repo).getMarks(101);
	//verify(repo,times(2)).getMarks(101);
}

@Test
public void test2(){
 when(repo.getMarks(101)).thenReturn(93);
assertEquals("A",service.computeGrade(new Student(101,"Sam")));
}

@Test
public void test3(){
	Student s=new Student(105,"Rai");
	doNothing().when(repo).saveRecord(s);
	assertTrue(service.saveRecord(s));
```

```
        verify(repo,times(1)).saveRecord(s);
    }

    @Test
    public void test4(){
        Student s=new Student(105,"R");
        doNothing().when(repo).saveRecord(s);
        assertFalse(service.saveRecord(s));
        verify(repo,times(0)).saveRecord(s);
    }

    @Test
    public void test5(){
        when(repo.findById(101)).thenReturn(new Student(101,"Arjun"));
        assertNotNull(service.getStudent(101));
    }

    @Test(expected=java.lang.NullPointerException.class)
    public void test6(){
        when(repo.findById(105)).thenThrow(new NullPointerException("error"));
        assertNull(service.getStudent(105));
    }

}
```

- Mocking is done when you invoke methods of a class that has external communication like database calls or rest calls.
- Use @Mock to create mocks which are needed to support testing of class to be tested.
- Use @InjectMocks to create class instances which needs to be tested in test class.

  MockitoAnnotations.initMocks(this);

- we can enable Mockito annotations programmatically by invoking MockitoAnnotations.initMocks().

20. How can I check that the repo.getMarks() this method is being called???

Mockito provides the method verify()

a) verifying that the getMarks() was called from service class only once.

```
verify(repo).getMarks(101);
```

b) verifying that the getMarks() was called from service class twice

```
verify(repo,times(2)).getMarks(101);
```

Methods from Mockito

```
//methods with non void return type
when(repo.getMarks(101)).thenReturn(92);

//methods with void return type
doNothing().when(repo).saveRecord(s);

//methods to throw an exception at runtime
when(repo.findById(105))
        .thenThrow(new NullPointerException("error"));
```

Spring Core

Maven project management tool

- Maven will make it easier to arrange the dependencies required for the project.
- Let us create our first maven project.
  - File >> New >> Maven
  - groupId (name of the organization who created the project)

    org.jf036
  - artifactId (project name)

    spring-maven-demo
  - Multiple projects can be part of the same groupId. version
  - We will select Simple project(skip archetype selection).

21. **What is meant by archetype?**
    Archetype is the template structure of the project.

The folder structure shows the following source folders:

- src/main/java

- all your java coding will be wriiten

- src/main/resources

- properties file will be kept here

- src/test/java

- The testing java files will be kept

- src/test/resources

- any support files for testing shall be kept

- The pom.xml(Project Object Model) is the file where we will do the configuration related steps.
- For getting information about the dependencies (jar files),we will visit Maven central repository
- From there, we will copy the dependency I need and put it in the pom.xml file.
- The file will be downloaded from the central repository to your local machine's repository.

- C:\users\<<associate_id>>\.m2\repository\

a) First time to download, it will take time.

Next time if you add the dependencies in a new project, it will get done quicker.

b) I just mentioned spring-context. The other dependencies (aka transitive dependencies) I don't have to explicit mention.

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.6.RELEASE</version>
    </dependency>
</dependencies>
```

Spring Framework version 5 :-

a) Spring Core & Spring JdbcTemplate

b) Spring MVC using Spring Boot

c) Spring REST Web Services

d) Spring Data JPA

e) Spring Cloud

f) Spring Security

...

22.Why shall I use Spring framework?

- " At its core, Spring Framework's Inversion of Control (IoC) and Dependency Injection (DI) features provide the foundation for a wide-ranging set of features and functionality. "
- When two classes are highly dependent on each other, it is called tight coupling.In tight coupling, an object (parent object) creates another object (child object) for its usage.

- Any OOP looks for loose coupling... To get loose coupling, use interfaces or abstract classes.
- Loose coupling in Java means that the classes are independent of each other. The only knowledge one class has about the other class is what the other class has exposed through its interfaces in loose coupling.

23. What is Dependency Injection (DI)?

- Some other class will create the dependent objects and inject the dependency into the required class.
- Here, Main class create the Car instance and injects into the Rider instance.
- Dependency injection (DI) is the concept in which objects get other required objects from outside. DI can be implemented in any programming language. The general concept behind dependency injection is called Inversion of Control.

What you do when you are at home and you feel hungry??

You are hungry, you need something to eat..

Mother goes and prepares something for you to eat..

DI aka IoC

How does Spring framework apply DI?

- The interface org.springframework.context.ApplicationContext represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the beans.

a) Annotation based approach

```
@Configuration
public class AppConfig {
	@Bean
	public Person one() {
		return new Person("Arghya",20);
	}
}
```

@Configuration annotation indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

@Autowired: Autowiring feature of spring framework enables you to inject the object dependency implicitly.

The **@Qualifier annotation** in Spring is used to differentiate a bean among the same type of bean objects. If we have more than one bean of the same type and want to wire only one of them then use the @Qualifier annotation along with @Autowired to specify which exact bean will be wired.

b) XML based approach

Add a xml file

To read the XML file, we need ClassPathXmlApplicationContext

```
ClassPathXmlApplicationContext ctx=
        new ClassPathXmlApplicationContext("spring.xml");
```

spring.xml (to be kept under src/main/resources folder)

```
  <bean class="com.cts.model.Person" id="one">
        <property value="Ritik" name="name"/>
        <property name="age" value="20"/>
        <property name="laptop" ref="lapp"/>
  </bean>

  <bean class="com.cts.model.Laptop" id="lapp">
        <constructor-arg index="0" value="Dell"/>
  </bean>
```

- **XML based approach has 2 types of DI**

a) constructor based

```
<constructor-arg index="0" value="Ritik"/>
<constructor-arg index="1" value="20"/>
OR
<constructor-arg type="java.lang.String" value="Ritik"/>
<constructor-arg type="int" value="20"/>
```

- constructor with proper number of parameters

b) setter based

```
<property value="Ritik" name="name"/>
<property name="age" value="20"/>
```

- setName() and setAge() method implicitly

Autowiring in XML approach...

```
<bean class="com.cts.model.Person" id="one" autowire="byName">
    <property value="Ritik" name="name"/>
    <property name="age" value="20"/>
</bean>
```

autowire="byType" | setter

autowire="byName"

autowire="constructor"

autowire="no"

24.When is the bean instance created by the Spring container?

- The Spring container initializes all the bean instances on startup.
- In annotation based approach, how to prevent creation of instances on startup? We use @Lazy annotation when declaring the bean in the AppConfig class
- In xml based format, we can achieve the same thing by adding lazy-init=true in the <bean> method.
- The Spring container creates each bean instance as a singleton bean.
- In annotation based approach, how to get different instances of the same bean? We use @Scope("prototype") annotation when declaring the bean in the AppConfig class
- In xml based format, we can achieve the same thing by adding scope="prototype" in the <bean> method.
- "By default, Spring creates all singleton beans eagerly at the startup/bootstrapping of the application context."

1. Work with collections like Array, List in xml based configuration

```
<bean class="com.cts.model.Laptop" id="laptop2">
    <constructor-arg index="0" value="Lenovo"/>
</bean>

<bean class="com.cts.model.SomeClass" id="someClass">
    <property name="list">
        <list>
            <value>orange</value>
            <value>violet</value>
            <value>white</value>
        </list>
    </property>
    <property name="laptops">
        <list>
            <ref bean="laptop1" />
            <ref bean="laptop2"/>
```

```
            </list>
        </property>
        <property name="map">
            <map>
                <entry key="101" value="mango"/>
                <entry key="102" value="litchi"/>
                <entry key="103" value="papaya"/>
            </map>
        </property>
    </bean>
```

"value-ref attribute"

```
        <bean id="publisher1" class="org.kodejava.spring.core.Publisher">
      <property name="name" value="EMI Studios" />
    </bean>
        <bean id="publisher2" class="org.kodejava.spring.core.Publisher">
      <property name="name" value="Marvel Studios" />
    </bean>

<bean>
         <property name="publisher">
        <map>
 <entry key="P101" value-ref="publisher1" />
 <entry key="P102" value-ref="publisher2" />
        </map>
      </property>
</bean>
```

FYR: https://kodejava.org/how-do-i-inject-collections-using-map-element-in-spring/

2. Use of @ComponentScan annotation and its xml equivalent

- @Component | Indicates that an annotated class is a "component".Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
- @ComponentScan(basePackages="com") is written along with @Configuration annotation| The Spring container will scan all the packages under the base package to look for classes annotated with @Component or equivalent annotation and thereby create an instance of the class.
- The @Service, @Controller, @Repository are all equivalent annotations to @Component.

```
@Service
public class Calculator {
    public int add(int a,int b) {
            return a+b;
    }
}
```

```
@Configuration
@ComponentScan(basePackages="com")
public class AppConfig {
    ...
}
```

- Configures component scanning directives for use with @Configuration classes.Provides support parallel with Spring XML's <context:component-scan> element.

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

3. Use of @PropertySource annotation and its xml equivalent

- It helps to read the .properties file kept under the src/main/resources folder.

```
@Configuration
@ComponentScan(basePackages="com")
@PropertySource("classpath:db.properties")
public class AppConfig {

    //org.springframework.core.env.Environment;
    @Autowired
    Environment env;

    @Bean
    public DbUtil util() {
        DbUtil db=new DbUtil();
        db.setDriver(env.getProperty("driver"));
        db.setUsername(env.getProperty("username"));
        db.setPassword(env.getProperty("password"));
        return db;
    }
...
}
```

- **XML based approach:**

```
<context:property-placeholder location="classpath:db.properties"/>

  <bean class="com.cts.model.DbUtil" id="dbOne">
        <property name="driver" value="${driver}"/>
     <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
```

```
</bean>
```

4. Special XML approach concepts

- inner bean

```
public class Point {
        private int xCoord;
        private int yCoord;
}
public class Line{
        private Point p1;
        private Point p2;
}
```

spring.xml

```
<bean class="com.cts.model.Point" id="start">
        <property name="xCoord" value="0"/>
        <property name="yCoord" value="5"/>
</bean>

<bean class="com.cts.model.Line">
        <property name="p1" ref="start"/>
        <property name="p2">
                <!--  inner bean -->
                <bean class="com.cts.model.Point">
                        <property name="xCoord" value="5"/>
                        <property name="yCoord" value="10"/>
                </bean>
        </property>
```

```
   </bean>
```

- **bean definition inheritance (it is not same as java inheritance)**

```
public class Account {
        private Long accNum;
        private String branchCode;
        private String accHldrName;
...
}
```

spring.xml

```
<!--  parent bean -->
<bean class="com.cts.model.Account" id="account"
        abstract="true">
        <property name="branchCode" value="SBI1515" />
</bean>

<bean id="user1" parent="account">
        <property name="accNum" value="115566" />
        <property name="accHldrName" value="Rajesh" />
</bean>

<bean id="user2" parent="account">
        <property name="accNum" value="115568" />
        <property name="accHldrName" value="Raushan" />
</bean>
```

In the main() method, add the lines

```
Account account=ctx.getBean("user1",Account.class);
        System.out.println(account.getBranchCode());
        System.out.println(account.getAccHldrName());
        System.out.println(account.getAccNum());
```

- **Spring JdbcTemplate**

For today's demo, we need 3 dependencies

a) spring-context

b) spring-jdbc

c) mysql-connector

Create the Book table in the schema of your choice:

```
CREATE TABLE `books` (
  `isbn` int NOT NULL ,
  `title` varchar(50) NOT NULL,
  `price` int NOT NULL,
  `edition` int NOT NULL,
  PRIMARY KEY (`isbn`)
) ;
```

Add the following packages to your project src/main/java

com.cts

com.cts.model

com.cts.service

com.cts.repository

com.cts.exception

Add the db.proerties file under the src/main/resources source folder

driver=com.mysql.cj.jdbc.Driver

url=jdbc:mysql://localhost:3306/<<db_name>>

user=root

password=<<password>>

JdbcTemplate

- |This is the central class in the JDBC core package. It simplifies the use of JDBC and helps to avoid common errors. It executes core JDBC workflow, leaving application code to provide SQL and extract results.

DataSource

- A DataSource is a factory for connections to the physical databases. It is an alternative to the DriverManager facility. A datasource uses a URL along with username/password credentials to establish the database connection.

Code goes here...

```
package com.cts;

@Configuration

@PropertySource("classpath:db.properties")

@ComponentScan(basePackages="com.cts")

public class AppConfig {


    @Autowired

    Environment env;


    private final String URL = "url";

    private final String USER = "user";

    private final String DRIVER = "driver";

    private final String PASSWORD = "password";
```

```
        @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource
                = new DriverManagerDataSource();
        //MySQL database we are using
        dataSource.setDriverClassName(env.getProperty(DRIVER));
        dataSource.setUrl(env.getProperty(URL));
        dataSource.setUsername(env.getProperty(USER));
        dataSource.setPassword(env.getProperty(PASSWORD));

        return dataSource;
        }

        @Bean
    public JdbcTemplate jdbcTemplate() {
        JdbcTemplate jdbcTemplate = new JdbcTemplate();
        jdbcTemplate.setDataSource(dataSource());
        return jdbcTemplate;
    }
}



public class MainApp {
        public static void main(String[] args) {
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext(AppConfig.class);
        BookService service =context.getBean(BookService.class);

System.out.println("Menu");
```

```
System.out.println("1.View\n2.Save\n3.View All");
Scanner sc=new Scanner(System.in);
String opt=sc.nextLine();
        switch(opt) {
        case "1":
        System.out.println("Book code");
        try {
        Book book=                              service.getBookDetails(
                Integer.parseInt(sc.nextLine()));
        if(book!= null)
                System.out.println(book);
        else
                System.out.println("No record found");
        }catch(Exception e) {
                System.out.println(e.getMessage());
        }
                                break;
        case "2":
Book newBook=new Book();                System.out.println("Book code");
newBook.setIsbn(Integer.parseInt(sc.nextLine()));
System.out.println("Book title");                                   newBook.setTitle(sc.nextLine());
System.out.println("Book price");
        newBook.setPrice(Double.parseDouble(sc.nextLine()));
System.out.println("Book edition");
newBook.setEdition(Integer.parseInt(sc.nextLine()));


try {
        if(service.saveBookDetails(newBook))
        System.out.println("Book record added");
```

```
        else
        System.out.println("Failure!!");
}catch(Exception e) {
        System.out.println(e.getMessage());
}
        break;
                        case "3":
        List<Book> books=service.viewAll();
        if(books.isEmpty())
        System.out.println("No records found");
        else {
        System.out.println("---Details---");
                for(Book book:books) {
        System.out.println(book);
                                        }
        }
                break;
                }

                sc.close();

                context.close();

        }
}
```

Add the class Book in the com.cts.model package

```
public class Book {
        private int isbn;
        private String title;
```

```
	private double price;
	private int edition;
...
	public String toString() {
	return "[ "+isbn+"| "+title+" | "+String.format("%.2f",price)+" | "+edition+" ]";
	}
}
```

CRUD operations |

- Create Read Update Delete

We will create an interface BookRepository with the service methods

```
package com.cts.repository;

public interface BookRepository {
	Book getBookDetails(int isbn) throws CustomException;
	boolean saveBookDetails(Book book)throws CustomException;
	List<Book> viewAll();
}
```

We will create a class BookRepositoryImpl which implements the BookRepository interface and overrides those methods.

On top of the implementation class "BookRepositoryImpl", we will add the annotation @Repository.

```
@Repository
public class BookRepositoryImpl implements BookRepository {

	@Autowired
	JdbcTemplate jdbcTemplate;
....
```

}

SOLID design principal :-

- **The Service Layer -**

➢ We will create the class BookService in the package com.cts.service and annotate it with the annotation @Service.

```
@Service
public class BookService {
    private BookRepository repo;
    @Autowired
    public void setRepo(BookRepository repo) {
            this.repo = repo;
    }

public Book getBookDetails(int isbn)
            throws CustomException {
            return repo.getBookDetails(isbn);
    }

public boolean saveBookDetails(Book book)throws CustomException {
            return repo.saveBookDetails(book);
    }

public List<Book> viewAll() {
            return repo.viewAll();
    }
}
```

Create a CustomException class in the com.cts.exception package. The class should extend Exception class.

In the Impl class

```
public boolean saveBookDetails(Book book) throws CustomException{
String insert_qry = "insert into books values(?,?,?,?)";
        int cnt = 0;
        try {
        //DML operation
        cnt=jdbcTemplate.update(insert_qry, book.getIsbn(),
book.getTitle(),book.getPrice(),book.getEdition());
        }catch(Exception e) {
                        throw new CustomException("Duplicate book code");
        }
                if (cnt > 0)
                        return true;
                return false;
        }
```

******

update(String sql, Object... args):: int

- Issue a single SQL update operation (such as an insert, update or delete statement) via a prepared statement, binding the given arguments.

In the Impl class...

```
public Book getBookDetails(int isbn) throws CustomException {
String search_id_qry = "select * from books where isbn=?";
        try {
        return jdbcTemplate.queryForObject(search_id_qry, new Object[] { isbn },
        BeanPropertyRowMapper.newInstance(Book.class));
        }catch(Exception e) {
```

```
            throw new CustomException("Book code not present");
            }
        }
```

queryForObject(String sql, RowMapper<T> rowMapper, Object... args)

- Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a result object via a RowMapper.

25. What is RowMapper?

- RowMapper is an interface
- BeanPropertyRowMapper is the impl class for the RowMapper interface. All its needs is that the bean class (Book class) whose object it will create MUST HAVE A NO ARG CONSTRUCTOR.

BeanPropertyRowMapper |

- RowMapper implementation that converts a row into a new instance of the specified mapped target class (Book). The mapped target class must be a top-level class and it must have a default or no-arg constructor.

In the Impl class

```
public List<Book> viewAll() {
String search_all_qry="select * from books";
List<Book> list=jdbcTemplate.query(search_all_qry, (rs,rowNum)->{
                Book book=new Book();
                book.setIsbn(rs.getInt(1));
                book.setTitle(rs.getString(2));
                book.setPrice(rs.getDouble(3));
                book.setEdition(rs.getInt(4));

                return book;
            });
            return list;}
```

query(String sql, RowMapper<T> rowMapper)

- Execute a query given static SQL, mapping each row to a result object via a RowMapper.

mapRow(ResultSet rs, int rowNum) : T

- Implementations must implement this method to map each row of data in the ResultSet.

```
(rs, rowNum) -> {
        //retrieve the record from ResultSet rs
        //create a T instance here it is Book
        // add the fields to the Book instance using setter methods
        //return the Book instance
};
```

- **Aspect Oriented Programming**

- Trutime is a "cross cutting concern"
- In an application, we have logging, authentication as cross cutting concerns.
  - ATM counter,

    a) Change PIN

    b) Withdraw

    c) Deposit

    ...
- Validate PIN is another example of cross cutting concern

Spring core + Spring aop

spring-context

The other dependencies requires are:

```
<dependency>                    <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.2.5.RELEASE</version>
</dependency>
<dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.8.9</version>
</dependency>
<dependency>                    <groupId>org.aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>1.8.9</version>
</dependency>
```

- **AOP terminology**

Aspect

- It is a class annotated with @Aspect annotation.
- This aspect will contain the cross cutting concern related coding.

- The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.
- In the AppConfig class where you have already @Configuration annotation present, also add the annotation @EnableAspectJAutoProxy Enables support for handling components marked with AspectJ's @Aspect annotation, similar to functionality found in Spring's <aop:aspectj-autoproxy> XML element. To be used on @Configuration classes
- The class with @Aspect annotation on top of it can generaten different types of advices like @Before advice, @After,@AfterReturning, @AfterThrowing, @Around.
- Advice is nothing but a method which will get executed at a particular join point.
- The join point is also a method(s) on execution of which the advices are supposed to trigger and execute.

Example of advice (before)

```
@Before
public boolean validatePin(){
        //code here
}
```

Example of advice (after)

```
@After
public String notifyUserThruSMS(){
        //code here
}
```

- **Example of join point(s)**

  ```
  public void withdraw(){}
  public void deposit(){}
  public void changePin(){}
  ```

Let us explore through an example

```
package com.cts.model;
public class Person {
        private String name;
        private int age;
//getter, setters, constructors

public boolean showDetails(){
System.out.println(getName()+ " is now "                    +getAge()+" years old.");
        if(age>=60) {
                        //return true;
```

```
                throw new ArithmeticException();
        }

                return false;
}
}

package com.cts;
@Aspect
public class MyAspect {

@Pointcut("execution(* com.cts.model.Person.show* (..))"
        public void k(){}

@Before("k()")
        public void beforeAdvice() {
        System.out.println("before advice being called...");
        }

@After("k()")
        public void afterAdvice() {
        System.out.println("after advice being called...");
        }

@AfterReturning("k()")
        public void afterAdvice2() {
                System.out.println("After normal execution...");
        }
```

```
	@AfterThrowing("k()")
	public void afterAdvice3() {
		System.out.println("After abnormal execution...");
	}

}


package com.cts;
@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages = "com")
public class AppConfig {

	@Bean
	public Person one(){
		Person p=new Person();
		p.setName("Meghna");
		p.setAge(16);//61
		return p;
	}

	@Bean
	public MyAspect aspectBean(){
		return new MyAspect();
	}
}

package com.cts;
```

```
public class Main{

public static void main(String[] args) {

                    //annotation based

AnnotationConfifApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);


Person per=ctx.getBean("one",Person.class);

//per.showDetails();

                    try {

                              per.showDetails();

                    }catch(Exception e) {}

}


}
```

Sample Output

---

before advice being called...

Meghna is now 16 years old.

after advice being called...

"execution(* com.cts.model.Person.show* (..))"

The above syntax is known pointcut; a predicate that matches join points.

Look for methods that start with

showDetails(String x)

showDetails(Integer x, String y)

The primary Spring PCD (pointcut designator) is execution, which matches method execution join points.

@Pointcut("execution(* com.baeldung.pointcutadvice.dao.FooDao.*(..))")

Here the first wildcard matches any return value, the second matches any method name, and the (..) pattern matches any number of parameters (zero or more).

@After advice will execute for all possibilities of the

join point.

Output (when age is 16)

@AfterReturning advice will execute only when join point follows normal execution

---

before advice being called...

Meghna is now 16 years old.

after advice being called...

After normal execution...

Output (when age is 61)

@AfterThrowing advice will execute only when join point throws an exception

---

before advice being called...

Meghna is now 61 years old.

after advice being called...

After abnormal execution...

Web MVC

- **Model View Controller design pattern**
  - View layer is the presentation layer which is the interface of your application to the users. The client/users see the interface on their respective web browser windows.
  - This layer is created using HTML, CSS and Javascript, jquery, Bootstrap, JSP etc...
  - We can also use Angular and React to create such web pages.
  - Controller is the intermediate layer which controls the flow from the view layer for being processed in the back end. The response coming down the back end is also sent back to the view layer via the controller. This role is played by a special Java class known as Servlet.
  - Model is the entire back end layer. All the java classes we create either for handling business logic (service layer), database operations (repository layer) and java bean classes are members of this layer.

- **Web Server** :
  - We will use Apache Tomcat Server as our web server.
  - A web server is an application used to host and publish web applications/web services and we can send request for such applications through the browser address bar in the form of url to it.
  - Web server handles the request gets the response from the application and passes back to the browser from which the reuqest had come.

  Catalina |

  - Used to create and maintain the life cycle of user defined servlet classes.

    b) Jasper | Used to create and maintain the life cycle of JSP pages

    c) Coyote | handles HTTP protocol

- Lets us create our **web mvc application**.

  We will first change the perspective to Java EE(default).

  - File >> New >> Dynamic Web Project (Dynamic Web Module version : 3.0)
  - Java Resources
    src (We will create all our Java source files)
  - web content(contain all the HTML, CSS, images, js, jsp,... pages)
  - WEB-INF [restricted area]

    lib (place to keep jar files like mysql-connector)

WAR (Web Archive)

- WAR is a compressed file which you can deploy on some remote web server. The META-INF folder has one important file known manifest file...

  http://localhost:9000/mera-web-app/home.html

- Add the following classes in the src folder

  com.cts.model.User

  com.cts.service.UserService

  com.cts.controller.HomeController (New >> Servlet)

  The user defined servlet (HomeController.java) is a class which inherits the javax.servlet.http.HttpServlet abstract class

  The HttpServlet class further inherits the javax.servlet.GenericServlet abstract class.

  The GenericServlet class implements the javax.servlet.Servlet interface.

  The Servlet interface has **three life cycle method** definitions.

  - init(), service() and destroy(). The service() is called for serving each HTTP request.

- **The GenericServlet** class implements only the init() and destroy() methods leaving the service() method as abstract.
- The HttpServlet overrides the service() method to further create two handy methods doGet() and doPost(). The doGet() method will handle HTTP GET requests and doPost() will handle HTTP POST requests. The class is abstract to prevent someone from creating an instance of this class.

What is this @WebServlet({"/login","/register"})?

- The @WebServlet annotation is used for url mapping.

URL patterns

```
@WebServlet({"/login","/register"})
	public class HomeController extends HttpServlet{
	...
	}
```

khushi and genius (url patterns) and Kanchan (physical file name) are all names of the same person

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	response.getWriter().append("Served at: ").append(request.getContextPath());
	}
```

- **HttpServletRequest and HttpServletResponse** are interfaces whose instances are created and managed by the servlet container (catalina). The so called concrete class which had implemented this interface but I don't get to know about it.
- **RequestDispatcher** is an interface used to dispatch the control to some other resource(.html page,.jsp page, another servlet url pattern)

- **RequestDispatcher dispatcher** =

  request.getRequestDispatcher("<<target>>");

  a) dispatcher.forward(request,response);

  - The response of the current page where the dispatcher is called will be ignored/skipped

  b) dispatcher.include(request,response);

  - The response of the current page where the dispatcher is called will be combined with the response from the target page.

c)response.setContentType("text/html");

- It is used to instruct what to do with the response.

home.html -> HomeController -> UserService

a) Not Okay (false) -> home.html

b) Okay (true) -> success.html

- **JSP & JSTL**
  - **Servlet :**
    - Servlet is a class which inherits the javax.servlet.http.HttpServlet class and provides the two main methods to handle incoming requests doGet() and doPost() . The HTTP GET requests are entertained by doGet() and HTTP POST requests are handled by doPost() method.

    - The Servlet has 3 lifecycle methods: init(), service() and destroy()

- **@WebServlet**({"/login","/register"}) is the annotation which provides the url patterns (nick name to find the servlet without disclosing the physical file name).
- **RequestDispatcher interface** which is used to dispatch the request to the next resource which can be another servlet, html or JSP page)

- **JSP(Java Server Pages):**
  - JSP page is like an HTML page in which we can embed java codes in special sections called scripting elements.

a) directive

<%@ page ... %> | pass instructions to the jasper component during translation.

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

To import java packages, we will add the import attribute to the page directive.

<%@ page import="java.util.Date,java.util.List" %>

b) scriplet | is used to add java coding in between html elements and codes. What we add within the scriplet is added to the _jspService() method after translation.

```
<% .... %>

        <%
            int k=5;
                for(int i=1;i<=5;i++){
            %>
            <%
                }
            %>
```

c) expression | is used to display something as response

```
<%=  ..... %>

    <%=new Date() %> on translation becomes
                        out.print(new Date());
```

DO NOT GIVE A SEMICOLON AT THE END

d) declaration | is used to add methods, static and instance

variables.

```
<%! .... %>

    <%!
        public int cube(int x){
            return x*x*x;
        }
    %>
```

e) comment

```
<%--  --%>
```

- A JSP page on execution first gets translated into the equivalent servlet file. Then the file is compiled as a .class file...

  index.jsp ---> index_jsp.java ---> index_jsp.class

- The JSP page is maintained by the Jasper component of the Apache Tomcat server.

26. Where shall I add the JSP page?

- We will add it in the webcontent folder [outside WEB-INF]
- Right click webcontent >> Add >> New >> JSP page

<<your workspace>>\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\<<project name>>\org\apache\jsp

Implicit objects in JSP

I. JspWriter out
II. HttpServletRequest request
III. HttpServletResponse response
IV. ServletConfig config
V. ServletContext application
VI. Throwable exception
VII. Object page
VIII. HttpSession session

<%@include %> directive is used to include another file (jsp page, html page) into this source jsp page.

```
<%@include file="header.html" %>
```

The content of the file will be copy pasted into your source code during translation.

We have to write scripting elements in between HTML elements which at times becomes clumsy and cumbersome, meaning hard to maintain.

```
<%
    for(int i=1;i<=5;i++){
%>
<tr>
```

```
<td><%=i %> * <%=k %> = </td><td><%= (i*k)%></td>
</tr>
<%
        }
%>
```

- JSP Standard Tag Library which provides a pre built set of tags. We get a chance to make our JSP page scripting free.
- First need to add the jstl-1.2 jar file in the WEB-INF\lib folder of our web project. Refresh your project to find if it has been added or not.
- Add the JSTL to our JSP page

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

We use <c:forEach> tag as an alternative

```
<c:set value="5" var="k"/>
<table>
<c:forEach var="i" begin="1" end="5">
<tr>
        <td>${i} * ${k} = </td><td>${i*k}</td>
</tr>
</c:forEach>
</table>
```

Example of <c:if> tag as an alternative to if-else

```
<c:set value="guest" var="role"/>
<c:if test="${role eq 'admin' }">
        <h2> Hello Admin</h2>
</c:if>
<c:if test="${role ne 'admin' }">
        <h2> Welcome user</h2>
</c:if>
```

Example of applying <c:forEach> on list of items

```
<%
List<String> depts=Arrays.asList("Human Resources","Finance","Accounts","Transport");
	application.setAttribute("depts",depts);
%>
	Department: <select>
	<c:forEach items="${depts}" var="d">
	<option value="${d}" ${d eq 'Finance'?'selected':'' }>${d}</option>
	</c:forEach>
	</select>
```

Complete Demo using Servlet and JSP

com.cts.model package

```
public class User {
	private String userName;
	private String gender;
	private LocalDate birthDate;
----
}
```

com.cts.service package

```
public class UserService {
	public boolean validateUser(User user) {
		if(user.getUserName().length()>=2)
			return true;

		return false;
	}
}
```

com.cts.controller package

Replace the doGet() method as given below:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
        PrintWriter pw = response.getWriter();
        if (request.getParameter("btnSubmit") != null) {

        String userName = request.getParameter("userName");
        String gender = request.getParameter("gender");
        String sDate = request.getParameter("birthDate");
        // convert String to LocalDate                  LocalDate birthDate =
LocalDate.parse(sDate);

        User user = new User(userName, gender, birthDate);
        RequestDispatcher dispatcher = null;
        response.setContentType("text/html");
        if (service.validateUser(user)) {
        Period p=Period.between(birthDate, LocalDate.now());
        request.setAttribute("user", user);                  request.setAttribute("age",
                                p.getYears()+" years, "+p.getMonths()+" months,
"+p.getDays()+" days.");
        dispatcher = request.getRequestDispatcher("success.jsp");
        dispatcher.forward(request, response);
        } else {
                pw.println("Access Denied");
        dispatcher = request.getRequestDispatcher("home.jsp");
        dispatcher.include(request, response);
                }
```

```
        } else
                pw.println("The page is under maintanance...");
}
```

home.jsp

```
<form action="register" method="post">
        Name: <input type="text" name="userName"/><br/>
        Gender:
        <input type="radio" name="gender" value="male">Male
        <input type="radio" name="gender" value="female">Female
        <br/>
        Birth Date:
        <input type="date" name="birthDate"/><br/>                    <input
type="submit" name="btnSubmit"/>
</form>
```

success.jsp

```
        <h2>Details</h2>
        User Name: ${user.userName }<br/>
        Gender :${user.gender }<br/>
        Age: ${age}<br/>
```

Add the <%@ taglib %> directive accordingly.

SPRING MVC

- We will implement MVC design pattern using Spring framework.

1. What is Spring Boot used for?

- Spring Boot allows us to create production grade, ready to use,stand alone applications with ease.

We will find most of the common configurations steps already completed. We will only need to override certain steps when the common ones are not woking specially for you.

CONVENTION OVER CONFIGURATION

2. What shall I get when designing Spring MVC application using Spring Boot?

- The embedded tomcat server.

- We will create a simple maven project (skip archetype selection).

  groupId org.jf036

  artifactId spring-mvc-demo1

Open the pom.xml once the above step is done.

We will add the parent dependency which will provide me the

basic configuration settings

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.6.RELEASE</version>
</parent>
<properties>
        <java.version>1.8</java.version>
        <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
</properties>
```

spring-boot-starter-parent

- The "spring-boot-starter-parent" is a special starter that provides useful Maven defaults i.e it adds all the required jars and other things automatically. It also provides a dependency-management section so that you can omit version tags for dependencies you are using in pom.xml.

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

- Spring Boot provides a number of starters that allow us to add jars in the classpath. Spring Boot built-in starters make development easier and rapid.

  - For Spring MVC | spring-boot-starter-web
  - For Spring Data JPA | spring-boot-starter-data-jpa
  - For testing |spring-boot-testing

- Starter of Spring web uses Spring MVC, REST and Tomcat as a default embedded server. The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development. It also reduces the build dependency count.

3. How to start the application?

- We will add the Main class.
- On top of the class we will write @SpringBootApplication annotation.
- Within the main() method, we will add SpringApplication.run(Main.class, args);

```
package com.cts;
@SpringBootApplication
public class MainApp {
    public static void main(String[] args) {
    ApplicationContext ctx=
        SpringApplication.run(MainApp.class,args);
```

```
            System.out.println(ctx);
        }
}
```

- Spring Boot tries to work with certain default configuration. The tomcat server should run on port number 8080. Suppose this port number is already in use.
- We need to change the port number.
- We will add the application.properties in the src/main/resources folder

  server.port=9000

4. What does @SpringBootApplication annotation do?

- This annotation is equivalent to three other annotations:

  a) @Configuration

  b) @ComponentScan

  c) @EnableAutoConfiguration

- It enables scanning for configuration classes, properties file and loads them into the Spring Application context.

5. What is the use of SpringApplication.run() method?

- Class that can be used to bootstrap and launch a Spring application from a Java main method. By default class will perform the following steps to bootstrap your application:

  a) Create an appropriate ApplicationContext instance (depending on your classpath)

  b) Refresh the application context, loading all singleton beans

  c) Start the embedded tomcat server after reading the default configuration

DispatcherServlet | If we add the spring-boot-starter-web dependency then automatically this DispatcherServlet (aka Front Controller servlet) will be made available.

We will now add the java class with @Controller annotation

```
package com.cts.controller;

@Controller
```

```
public class HomeController {
	@GetMapping("/")
	public String showPage() {
		return "hello";
	}
}
```

We will add the jsp page

First we added the src/main/webapp source folder

Within the source folder, add folder WEB-INF/views

We finally added the jsp page in the views folder.

Add the JSP and JSTL dependency in the pom.xml

```
<dependency>		<groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>		</dependency>
<dependency>
	<groupId>javax.servlet</groupId>		<artifactId>jstl</artifactId>
	</dependency>
```

Add certain entries in the application.properties file.

6. Why so complex???

- It is no longer mandatory to add JSP page in the view layer.
- Hence, there is no default configuration.
- **Ways to proceed**

  a) Use HTML with Thymeleaf view engine

  b) No need to add JSP; make it a web service (a web application without view layer). Access the web service using some other application built through React or Angular etc

c)We need to restart our Spring Boot application to feel any new change done. For that we either need to stop the already running server otherwise we get "port number in use" error.

Solution: We will a dependency called spring-boot-devtools

<dependency>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-devtools</artifactId>

</dependency>

spring-boot-devtools module includes an embedded LiveReload server that is used to trigger a browser refresh when a resource is changed.

@Controller | The @Controller annotation indicates that a particular class serves the role of a controller. Spring Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It's used to mark a class as a web request handler.

@GetMapping("/")

is the shortcut for @RequestMapping(value="/",method=RequestMethod.GET)

-- will listen to HTTP GET requests.

@PostMapping("/")

is the shortcut for @RequestMapping(value="/",method=RequestMethod.POST)

-- will listen to HTTP POST requests.

Spring MVC Execution Flow

http://localhost:9000/login

```
@RequestMapping(value="/login",method=RequestMethod.GET)
	public String showPage() {
		return "home";
	}
```

URL Pattern

The string "home" is returned to the DispatcherServlet

The ViewResolver component will search for a view page (jsp page) by consulting the application.properties file.

spring.mvc.view.prefix=WEB-INF/views/

spring.mvc.view.suffix=.jsp

Locate the jsp page | WEB-INF/views/home.jsp

The JSP page will be returned to the DispatcherServlet

and it will be rendered by the browser in response.

In Servlet programming, you wrote

String un=request.getParam("userName");

@RequestParam("userName") String userName;

we can use @RequestParam to extract query parameters, form parameters, and even files from the request.

Model |Java-5-specific interface that defines a holder for model attributes. Primarily designed for adding attributes to the model. Allows for accessing the overall model as a java.util.Map.

model.addAttribute("message","Access Denied");

model.addAttribute("list",new ArrayList<>());

- **Spring MVC form validations**

Case Study 1|

- We will have the <form> in the home.jsp page
- We will validate using the Controller class
- We will generate the response through success.jsp page

7. Why not using client side validation?

1. Client side scripting can be blocked

2. If the view layer is not present

Web service is a web application w/o the view layer.

The incoming data is in JSON format

Suppose you have few <form> input elements from which you want to capture date in your handler metod.

[

@RequestParam("fname") String firstName,

@RequestParam("lname") String lastName,

@RequestParam("gender") String gender

]

When the number of <form> input elements is quite big..

We can create a model class and map the view layer with the model class instance using @ModelAttribute annotation.

The <input> element's name attribute and the class private variable should have the same name.

private String firstName;

```
<input type="text" name="firstName"/>
```

@PostMapping("/save")

```
public String process(@ModelAttribute("person") Person person,BindingResult result) {


boolean error = false;

if (person.getFirstName().isEmpty()) {                    result.rejectValue("firstName",
"", "First Name not found");

                    error = true;

        }


        if(error)

                return "home";

        else

                 return "success";

}

}
```

"Once present in the model, the arguments fields should populate from all request parameters that have matching names."

https://www.baeldung.com/spring-mvc-and-the-modelattribute-annotation#:~:text=%40ModelAttribute%20is%20an%20annotation%20that,submitted%20from%20a%20company's%20employee.

```
void rejectValue(@Nullable

            String field,

            String errorCode,

            String defaultMessage)
```

field - the field name (may be null or empty String)

errorCode - error code, interpretable as a message key

defaultMessage - fallback default message

```
public class Person {

        private String firstName;

        private String lastName;
```

```
    private String email;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate birthDate;

...
}
```

Case Study 2

We will now modify the exisiting demo with spring mvc form tags instead of HTML form tags.

The value for attribute "path" is the private variable name

defined in the Person class.

The modelAttribute mentioned in the <frm:form> tag tells that this form is backed by instance of Person class.

<form:errors> tag helps to display the validation error message easily.

The Spring MVC form tags can be seen as data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it.

```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

 <form:form method="post" action="save" modelAttribute="person">
First Name: <form:input path="firstName" />
            <form:errors path="firstName" cssStyle="color:red;" /><br/>

            Last Name:<form:input path="lastName"/>
            <form:errors path="lastName" cssStyle="color:red;" /><br/>

            Email:<form:input type="email" path="email"/>
            <form:errors path="email" cssStyle="color:red;" /><br/>
```

```
		Birth Date:<form:input type="date" path="birthDate"/>
		<form:errors path="birthDate" cssStyle="color:red;" /><br/>

		<input type="submit"/>
</form:form>

@GetMapping("/")
	public String showHome(
			@ModelAttribute("person") Person person) {
		return "home";
	}
```

Case Study 3

Validations using annotations

Modify the private data members of Person class

to include the annotations:

[javax.validation.constraints]

@NotEmpty with String data type

@NotNull with non String data type

@Size(max=50, min=2, message="") //String

@Max(value="5",message="")//numeric

@PastOrPresent

@DateTimeFormat

.....

We MUST ADD @Valid before @ModelAttribute annotation to inform Spring Application context to validate using annotations.

The reference to the BindingResult interface MUST BE after the @ModelAttribute annotation to fetch the errors due to validation failure.

Add in the HomeController

```
@PostMapping("/saveOne")
public String processOne(@Valid @ModelAttribute("person") Person person, BindingResult result,Model model) {
		if (result.hasErrors()) {
			return "home";
		}
		return "success";
}
```

```
	@NotEmpty(message="FirstName is missing")
	private String firstName;
	@NotEmpty(message="LastName is missing")
	private String lastName;

	@NotEmpty(message="Email is missing")
	@Email(message="Email format is incorrect")
	private String email;

	@DateTimeFormat(pattern = "yyyy-MM-dd")
	@PastOrPresent
	@NotNull
	private LocalDate birthDate;
```

<form:form action="saveOne" >

…

</form:form>

FYR

https://web.archive.org/web/20161120115428/http://codetutr.com/2013/05/28/spring-mvc-form-validation/

Case Study 4 | Custom Validator example

Add the following handler method in the HomeController class

```
@Autowired
PersonValidator validator;

@PostMapping("/saveTwo")
public String processTwo(@ModelAttribute("person") Person person, BindingResult result) {
        validator.validate(person, result);
        if (result.hasErrors()) {
                        return "home";
        }
        return "success";
}
```

- **We will use our own validator instead of @Valid annotation.**

```
@Component
public class PersonValidator implements Validator {

        public boolean supports(Class clazz) {
                return Person.class.isAssignableFrom(clazz);
```

```
	}


public void validate(Object target, Errors errors)

	{

	Person person = (Person) target;

	/*Utility class offering convenient methods for invoking a Validator  and for rejecting empty fields. */

	ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "", "First name is required.");

	ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "", "Last name is required."); ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "", "Email is required.");

	ValidationUtils.rejectIfEmpty(errors,"birthDate","","Birth date is required.");


//custom validation rule

if(person != null && person.getBirthDate()!= null

	&& person.getBirthDate().isAfter(LocalDate.now())){

	errors.rejectValue("birthDate", "", "Birth date cannot be future date");

		}


if(person!=null && person.getFirstName()!= null && person.getFirstName().length()<2) {

	errors.rejectValue("firstName", "", "FirstName too short");

}


if(person!=null && person.getEmail()!= null && person.getEmail().endsWith("@gmail.com")==false) {

	errors.rejectValue("email", "", "Email should be a gmail account");

		}

	}

}
```

8. How to manage i18n in your Spring Application?

- We need to add messages.properties (default locale) file in the src/main/resources source folder.
- We will further add messages_XX.properties file corresponding to each locale.

  For eg. messages-de.properties

  German locale code= de, de-DE

  French locale code= fr

  English locale code= en-US, en-IN

- Each properties file will contain multiple key=value pairs.
- We will add the class AppConfig annotated with @Configuration annotation in the com package.
- This class will have two or more @Bean methods

```
@Configuration
public class AppConfig implements WebMvcConfigurer {

@Bean
public LocalValidatorFactoryBean validator(MessageSource messageSource) {
        LocalValidatorFactoryBean bean = new LocalValidatorFactoryBean();
        bean.setValidationMessageSource(messageSource);
                return bean;
}

        @Bean
        public MessageSource messageSource() {
                ReloadableResourceBundleMessageSource messageSource =
        new ReloadableResourceBundleMessageSource();
        messageSource.setBasename("classpath:messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
}
```

```
    @Bean
    public LocaleResolver localeResolver() {
      SessionLocaleResolver localeResolver = new SessionLocaleResolver();
      localeResolver.setDefaultLocale(Locale.US);
      return localeResolver;
    }

    @Bean
    public LocaleChangeInterceptor localeChangeInterceptor() {
      LocaleChangeInterceptor localeChangeInterceptor = new
LocaleChangeInterceptor();
      localeChangeInterceptor.setParamName("lang");
      return localeChangeInterceptor;
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
     registry.addInterceptor(localeChangeInterceptor());
    }
}
```

http://localhost:9006/login?lang=en

9. Why use WebMvcConfigurer interface?

- Interface WebMvcConfigurer. Defines callback methods to customize the Java-based configuration for Spring MVC enabled via @EnableWebMvc . @EnableWebMvc -annotated configuration classes may implement this interface to be called back and given a chance to customize the default configuration.

10.Why use LocaleResolver?

- DispatcherServlet enables you to automatically resolve messages using the client's locale. This is done with LocaleResolver objects.
- When a request comes in, the DispatcherServlet looks for a locale resolver and if it finds one it tries to use it to set the locale.

11.How shall I include them in our JSP pages?

```
<%@taglib uri="https://www.springframework.org/tags" prefix="spring"%>
<spring:message code="label.firstName" text="what-to-display-instead" />

<!-- label.firstName  is the key mentioned in the messages.properties file -->
```

Create a login.jsp and add the 3 taglib directives

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<body>
<a href="/login?lang=en">English</a> |
<a href="/login?lang=de">German</a>

<div style="text-align: center;">
<h1>
<s:message code="label.heading" text="Message" />
</h1>

<sf:form action="submitlogin" method="post"
        modelAttribute="login">

<s:message code="label.username" text="User Name" />: <sf:input path="userName" />
<sf:errors path="userName" cssStyle='color:red' />
```

```
<br />

<s:message code="label.password" text="Password" />: <sf:input type="password"
path="password" />
<sf:errors path="password" cssStyle='color:red' />
<br />

<s:message code="label.role" text="Role"/>:
<sf:select path="role">
        <sf:option value="" label="--- Select ---"/>
        <sf:options items="${roles}" />
</sf:select>
<sf:errors path="role" cssStyle='color:red' />
<br/>

<input type="submit" value="Login" />
<input type="reset" value="Cancel" />

</sf:form>
</div>

</body>
```

Create the class LoginBean in the com.bean package

```
import javax.validation.constraints.*;
public class LoginBean {
        @NotEmpty(message="{name.not.empty}")
        @Size(min=4,max=10,message="{name.size.incorrect}")
        private String userName;
```

```
	@NotEmpty(message="Password is mandatory")
	private String password;

	@NotEmpty(message="Select a role")
	private String role;
...
}
```

Create the class LoginController in the com.controller package. Annotate with @Controller annotation

```
@Controller
public class LoginController {
	@Autowired
	private LoginService service;

	@ModelAttribute("roles")
	public List<String> showAllRoles() {
		List<String> list = new ArrayList<String>();
		list.add("Member");
		list.add("Guest");
		list.add("Admin");

		return list;
	}

	@RequestMapping(value = "/login", method = RequestMethod.GET)
	public String showLoginpage(@ModelAttribute("login") LoginBean loginBean) {
		return "login";
	}
```

```
	@RequestMapping(value = "/submitlogin", method = RequestMethod.POST)
	public String checkLoginDetails(@Valid @ModelAttribute("login") LoginBean loginBean, BindingResult result, 	Model model) throws InvalidUserException {
		if (result.hasErrors()) {
			return "login";
		}
		if (service.validate(loginBean)) {
			return "success";
		}
	model.addAttribute("message", "Access denied");
		return "invalid";
	}
}
```

FYR: https://www.javadevjournal.com/spring-mvc/spring-mvc-model-attribute-annotation/

- **Spring MVC** makes a call to showAllRoles() method(annotated with @ModelAttribute)before the controller methods annotated with @RequestMapping are invoked. This sequence ensures that the model object is available before request processing starts in the Spring controller.

Create the class LoginService (annoated with @Service annotation) in the com.service package

```
@Service
public class LoginService {
	private List<String> names;

	public LoginService(){
		names=new ArrayList<>();
		names.add("Sanjay");
		names.add("Parth");
```

```
		names.add("Himani");
		names.add("Kajol");
	}
	public boolean validate(LoginBean bean) throws InvalidUserException	{
		if(bean.getUserName().equalsIgnoreCase(bean.getPassword())) {
		if(names.contains(bean.getUserName()))
			return true;
		else
	throw new InvalidUserException("Not a valid user");
		}
		return false;
	}
}
```

Post this add two new jsp pages: invalid.jsp and success.jsp

Add the 3 taglib directives on top

success.jsp

```
<div style="text-align: center;">
<h1>
	<s:message code="label.heading" text="Message" />
</h1>
<h2>Login Successful !!! Welcome ${login.userName}</h2>
		<br /> <a href="/login"> &lt;&lt;</a>
</div>
```

invalid.jsp

```
<div style="text-align: center;">
		<h2>${message}</h2>
</div>
```

We will now just add a class with @ControllerAdvice annotation which can contain multiple @ExceptionHandler methods..

```
@ControllerAdvice
public class GlobalControllerExceptionHandler {

@ExceptionHandler(com.exception.InvalidUserException.class)
public String showCustomErrorPage(Model model,Exception ex) {
	model.addAttribute("message",ex.getMessage());
	return "invalid";
}

}
```

Test Cases

1. i18n
2. validations
3. username and password same; not in list

"Not a valid user"

4. username and password not same; "Access Denied"
5. username and password same; also in list

"Welcome <<username>>

Model, ModelMap, ModelAndView

Both Model and ModelMap are interfaces to hold the attributes

model.addAttribute("key",value);

the model can supply attributes used for rendering views (all the JSP pages).

org.springframework.ui.Model, org.springframework.ui.ModelMap and org.springframework.web.servlet.ModelAndView provided by Spring MVC.

ModelAndView allows us to pass all the information required by Spring MVC in one return.

FYR: https://www.baeldung.com/spring-mvc-model-model-map-model-view

- **Code Quality**

a) Explore Lombok

b) Explore Logging

```
class Trainee{
        private String name;
        private int age;
}
```

You have to manually write/add several boiler plate coding

which increases the lines of code.

- Lombok jar when added and configured in your project will allow to not write /add this boiler plate coding explicitly.
- The Lombok annotations will take care of it thereby reducing the number of lines of code...

Create a simple maven project with skip archetype selection.

Give groupid and artifactid accordingly.

Open the pom.xml file and add the lobok dependency as given below:

```
<dependencies>
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
		<version>1.18.22</version>
		<scope>provided</scope>
	</dependency>
</dependencies>
```

Move to your .m2/respository/org/projectlombok/lombok/1.18.22

Issue the command

java -jar lombok-1.18.22.jar

Add the lombok to your respective Eclipse IDE

DO RESTART YOUR ECLIPSE IDE

```
package com.cts;
import lombok.*;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Trainee{
	private String name;
	private int age;
}
```

```
package com.cts;
public class Main{
	psvm(..){
		Trainee t=new Trainee("Kajol",20);
```

```
        Syso(t);//toString()
    }
}
```

- **Project Lombok** is a java library that automatically plugs into your editor and build tools, spicing up your java.
- Never write another getter or equals method again, with one annotation your class has a fully featured builder, Automate your logging variables, and much more.

FYR: https://projectlombok.org/

Window>> ShowView>> Outline

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

- **Logging** implies to note down all the activities that happened during the execution of an application.
- For logging we have several utilities or tools avaialble.

Log4j, java.util.Logging, Logback.

- **Log4j** is a simple and flexible logging framework.

a) Add the dependency in the pom.xml

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

b) Add the log4j.properties file with necessary instructions in the src/main/resources folder

[log4j.xml or log4j.properties]

```
#console window

log4j.rootLogger=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yy-MM-dd HH:mm:ss:SSS} %5p %t
%c{2}:%L - %m%n


#file

log4j.rootLogger=INFO,File

log4j.appender.File=org.apache.log4j.RollingFileAppender

log4j.appender.File.File=log/logfile.log

log4j.appender.File.layout=org.apache.log4j.PatternLayout

log4j.appender.File.layout.ConversionPattern=%d{yy-MM-dd HH:mm:ss:SSS} %5p %t
%c{2}:%L - %m%n
```

c) Add the following line in each class where you want to use the Logger reference:

```
Logger logger=Logger.getLogger(<<ClassName.class>>);
```

d) **Log4j has 6 levels of logging**

ALL TRACE DEBUG INFO WARN ERROR FATAL OFF

INFO (It will log all events >=INFO)

```
INFO WARN ERROR FATAL
```

12.What is SLF4J?

- Simple Logging Facade 4 Java
- It is not a logging framework.It provides a simple abstraction of all the logging frameworks in Java.
- As a coder I will use SLF4J, in the back end there can be any of the logging frameworks.

App SLF4J | Log4j

SLF4J |java.util.Logging

SLF4J | Logback

- It decouples the application and the logging framework.

13.How to involve SLF4J in your application?

- Add the slf4j dependency for your project with either log4j, logback, etc...

```
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>1.8.0-alpha2</version>
</dependency>
```

https://developpaper.com/use-of-slf4j-and-logback-in-maven-project/

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MainApp {
        private static Logger   logger=LoggerFactory.getLogger(MainApp.class);
...
}
```

```
<dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.2.3</version>
</dependency>
```

Add the logback.xml file in the src/main/resources folder

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <encoder>
         <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
      </encoder>
   </appender>

   <appender name="FILE" class="ch.qos.logback.core.FileAppender">
      <file>log/app.log</file>
      <append>true</append>
      <immediateFlush>true</immediateFlush>
      <encoder>
         <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
      </encoder>
   </appender>

   <logger name="com.cts" level="INFO"/>

   <root level="debug">
      <appender-ref ref="STDOUT"/>
          <appender-ref ref="FILE"/>
   </root>
</configuration>
```

SPRING DATA JPA

What is Spring Data JPA? Why shall I use it?

- JDBC|15 lines
- JdbcTemplate|10 lines
- Hibernate is a framework based on ORM,JPA|6 lines

- **Spring Data JPA** is a layer of abstraction over Hibernate

14.What is ORM?

- Object Relation Mapping

```
create table country(
co_code varchar(2) primary key,
co_name varchar(50)
);
```

We get a set of JPA annotations..

```
@Table(name="country")
@Entity
public class Country{
        @Id
        @Column(name="co_code")
        private String countryCode;

        @Column(name="co_name")
        private String countryName;
}
```

15.What is Hibernate?

- A Hibernate is a Java framework which is used to store the Java objects in the relational database system. It is an open-source, lightweight, ORM (Object Relational Mapping) tool.
- Hibernate is an implementation of JPA. So, it follows the common standards provided by the JPA.
- We can configure Hibernate either through xml approach or annotation based approach.

XML approach

```
<hibernate-mapping>
  <class name = "Employee" table = "EMPLOYEE">

    <meta attribute = "class-description">
      This class contains the employee detail.
    </meta>

    <id name = "id" type = "int" column = "id">
      <generator class="native"/>
    </id>

    <property name = "firstName" column = "first_name" type = "string"/>
    <property name = "lastName" column = "last_name" type = "string"/>
    <property name = "salary" column = "salary" type = "int"/>

  </class>
</hibernate-mapping>
```

employee.hbm.xml

The hibernate.cfg.xml is to be read...

url

username

password

dialect | The way Hibernate transform the commands into supporting SQL provider.

```
<hibernate-configuration>
 <session-factory>
   <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
   <property name="connection.url">jdbc:mysql://localhost:3306/database</property>
    <property name="connection.username">root</property>
   <property name="connection.password">password</property>
   <property name="connection.pool_size">3</property>
   <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
   <property name="current_session_context_class">thread</property>
   <property name="show_sql">true</property>
   <property name="format_sql">true</property>
   <property name="hbm2ddl.auto">update</property>
   <mapping class="com.cts.model.Employee" />
 </session-factory>
</hibernate-configuration>
```

- The dialect specifies the type of database used in hibernate so that hibernate generate appropriate type of SQL statements.

16.How to run a Hibernate application?

Create a singletom instance of SessionFactory

```
 private static SessionFactory factory;
factory = new Configuration().configure().buildSessionFactory();
```

```
//we can create multiple sessions as and when required.
Session session=factory.openSession();
Transaction tx=null;
    try {
     tx = session.beginTransaction();
     Employee employee = new Employee(fname, lname, salary);
     employeeID = (Integer) session.save(employee);
       tx.commit();
     } catch (HibernateException e) {
       if (tx!=null) tx.rollback();
       e.printStackTrace();
     } finally {
       session.close();
     }
```

- In annotation based approach, we will not require any such employee.hbm.xml file...

- **javax.persistence package**

1. We use the @Entity annotation to the Employee class, which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

2.The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.

3. Each entity bean will have a primary key, which you annotate on the class with the @Id annotation.

4. The @Column annotation is used to specify the details of the column to which a field or property will be mapped.

```
SessionFactory factory= new AnnotationConfiguration().
            configure().addPackage("com.cts").
```

addAnnotatedClass(Employee.class).

buildSessionFactory();

- **Spring Data JPA**

- Hibernate is a JPA provider and ORM that maps Java objects to relational database tables. Spring Data JPA is an abstraction that makes working with the JPA provider less verbose. Using Spring Data JPA you can eliminate a lot of the boilerplate code involved in managing a JPA provider like Hibernate.

Let us create a simple maven project

Open the pom.xml

parent| spring-boot-starter-parent

dependencies

spring-boot-starter-data-jpa

mysql-connector-java

lombok

spring-boot-devtools

Add the application.properties file in the src/main/resources folder

Add the following packages

com.cts.bean

com.cts.service

com.cts.repository

com.cts.exception

In the com.cts package, add the MainApp class with @SpringBootApplication and SpringApplication.run(...);

Add the following lines in the application.properties file...

#data source information

spring.datasource.url=jdbc:mysql://localhost:3306/jf036

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

#spring.jpa.generate-ddl=false

spring.jpa.show-sql=true

Execute the script in MySQL

```
create table country(
co_code varchar(2) primary key,
co_name varchar(50)
);
insert into country values ('IN', 'India');
insert into country values ('US', 'United States of America');
insert into country values ('ID', 'Indonesia');
insert into country values ('NO', 'Norway');
insert into country values ('AU', 'Australia');
```

Now, let create the classes

com.cts.bean

Create the class Country as below:

```
@Entity
@Table(name="country")
@Data
@AllArgsConstructor
@NoArgsConstructor
```

```
public class Country {
	@Id
	@Column(name="co_code")
	private String countryCode;

	@Column(name="co_name")
	private String countryName;
}
```

com.cts.service

Create the class CountryService as below:

```
@Service
public class CountryService {
	@Autowired
	CountryRepository repo;

	public List<Country> getAllCountries(){
		return repo.findAll();
	}

	public boolean addNewCountry(Country country) {
	country=repo.save(country);
		if(country != null)
			return true;
		return false;
	}

	public boolean removeCountry(String countryCode) {
		try {
```

```
                repo.deleteById(countryCode);
                return true;
            }catch(Exception e) {
                System.out.println(e.getMessage());
                return false;
            }
        }

        @Transactional
        public boolean removeCountryByName(String cName) {
            if(repo.deleteByCountryName(cName)>0)
                return true;

                return false;
        }

        public Optional<Country> searchCountry(String countryName){
        return repo.findByCountryName(countryName);
        }
        public List<Country> getAllCountriesSortedDesc() {
        return repo.findAllByOrderByCountryNameDesc();
        }

}

import java.util.Optional;
@Repository
public interface CountryRepository extends JpaRepository<Country,String>{
        public int deleteByCountryName(String countryName);
```

```
    List<Country> findAllByOrderByCountryNameDesc();
    public Optional<Country> findByCountryName(String name);
}
```

17. What is JpaRepository<Country, String>?

- JpaRepository<T,ID> is an interface which provides several service methods without being bothered about which class has implemented these methods...

```
@SpringBootApplication
public class MainApp implements CommandLineRunner {

    @Autowired
    CountryService service;

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

    @Override
    public void run(String... args) throws Exception {
    // fetch all records
    List<Country> countries = service.getAllCountries();
    System.out.println("List of all countries >>>");
  countries.stream().forEach((p) -> System.out.println(p));

    // add a new record into Country table
    Country country = new Country("DE", "Germany");
    if (service.addNewCountry(country))
        System.out.println("Record added");
```

```
        else
                System.out.println("Addition failed");
        }

        // remove a record
        String countryCode = "DE";
        if (service.removeCountry(countryCode))
        System.out.println("Record removed");
        else
                System.out.println("Removal failed");

        String cName="Australia";
        if(service.removeCountryByName(cName))
                System.out.println("Record removed");                    else
                System.out.println("Removal failed");

        // search by countryName
        String countryName = "India";
        Optional<Country> opt = service.searchCountry(countryName);
        if (opt.isPresent()) {
                System.out.println(opt.get());
        } else
                System.out.println("Record not found");

        //country names in asc order
        List<Country> cList = service.getAllCountriesSortedDesc();
        System.out.println("List of all countries >>>");
        cList.stream().forEach((p) -> System.out.println(p));

}
```

FYR: https://www.java4coding.com/contents/spring/springdatajpa/spring-data-jpa-method-naming-conventions

- **ORM**

Mappings

a) One-to-many

b) One-to-one

- XML based approach

- Annotation based approach

a)One to many relationship

A dept has many employees

```
create table dept(
        dpcode int primary key auto_increment,
        dpname varchar(100) not null,
        dploc varchar(100) not null
);

create table emp(
        empid int primary key auto_increment,
        fname varchar(100) not null,
        lname varchar(100) not null,
        dpid int not null,
        foreign key(dpid) references dept(dpcode)
);
```

An author has written many books

A shopping cart has multiple products

- **@Entity** | Is used over the class which is going to behave as an entity bean
- **@Table** | It is used to map an entity class with the table name
- **@Id** |It is used to denote the attribute as a PK column
- **@GeneratedValue** | Used to tell the way to generate the value for PK key column
- **@Column** |It is used to map a private data member to the field name in the table
- **@OneToMany** | Added on top of the collection reference of child type on the parent side to denote the parent side

  @OneToMany(mappedBy = "department")

This "department" is the name of the reference variable defined in the child bean.

- **@ManyToOne** | Added on top of the parent reference on the child side to denote the child side

```
@Entity
@Table(name = "dept")
@Data
@NoArgsConstructor
public class Department {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "dpcode", nullable = false, unique = true)
    private int id;

    @Column(name = "dpname", nullable = false, unique = true)
    private String name;

    @Column(name = "dploc", nullable = false, unique = false)
```

```
	private String location;

	@OneToMany(mappedBy = "department", cascade = CascadeType.ALL, fetch =
FetchType.EAGER)
	private List<Employee> empList;

	public Department(String name, String location) {
		super();
		this.name = name;
		this.location = location;
	}
}

@Entity
@Table(name = "emp")
@Data
@NoArgsConstructor
public class Employee {
	@Id
	@Column(name = "empid", nullable = false, unique = true)
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	@Column(name = "fname", nullable = false, unique = false)
	private String firstName;

	@Column(name = "lname", nullable = false, unique = false)
	private String lastName;

	@ManyToOne
```

```
	@JoinColumn(name = "dpid", nullable = false,
				unique = false)
	private Department department;

	@Override
	public String toString() {
		return "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" +
lastName + "]";
	}

	public Employee(String firstName, String lastName, Department department) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.department = department;
	}

}

@Repository
public interface DepartmentRepository extends JpaRepository<Department,Integer>{}

@Repository
public interface EmployeeRepository extends JpaRepository<Employee,Integer>{}
```

Points:

HikariDataSource

- **@OneToMany**

- **FetchType.EAGER**| EAGER fetching tells Hibernate to get the related entities with the initial query. This can be very efficient because all entities are fetched with only one query. But in most cases it just creates a huge overhead because you select entities you don't need in your use case. You can prevent this with FetchType.LAZY

- **CascadeType.ALL**| The meaning of CascadeType. ALL is that the persistence will propagate (cascade) all EntityManager operations (PERSIST, REMOVE, REFRESH, MERGE, DETACH) to the relating entities.

- **@GeneratedValue**| If we want to automatically generate the primary key value, we can add the @GeneratedValue annotation. This can use four generation types: AUTO, IDENTITY, SEQUENCE and TABLE.

- **@GeneratedValue (strategy = GenerationType.IDENTITY)**
  - identity column in the database which means they are auto-incremented

- **@GeneratedValue (strategy = GenerationType.SEQUENCE)**
  - Oracle has a explicit numeric value generator known as sequence. This generator uses sequences if our database supports them.

```
create sequence myseq

increment by 1

start with 100

max value 9999;
```

The TableGenerator uses an underlying database table that holds segments of identifier generation values.

@OneToOne mapping

A User instance must have a Profile instance

Add 2 interfaces: UsersRepository and ProfilesRepository

React

- React is a Javascript library used for creating the UI layer.
- React based applications can be used to consume Spring REST web services.

18. What is a web service?

- A web application without the view layer is the web service.It will talk to the db,query, collect the records into collections, perform business logic computations etc... but it will not display the data in the JSP or any equivalent page.

It will send the data as response object to another application in JSON format. JSON implies JavaScript Object Notation

```
[
        {
        "id":220466,
        "name":"Aditi"
        },
        {
        "id":220468,
        "name":"Raushan"
        },
        {
        "id":220485,
        "name":"Kanchan"
        }
]
```

- React will consume this incoming JSON object and display in desirable format. It will not have db layer of its own.

- **A JavaScript library for building user interfaces.**
- We need to first install node.js on your device.

- **Node.js** is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

- **V8** is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and in Node.js, among others.

Open the command prompt and try the following commands...

node -v

npm -v

❖ npm :

- **npm** (node package manager) is somewhat similar to maven.
- npm will read the package.json file and download the dependencies (packages/modules) accordingly.
- You have to install Node.js to get npm installed on your computer.[package manager for Node.js.]
- All npm packages are defined in files called package.json.
- npm can manage dependencies.
- npm can (in one command line) install all the dependencies of a project.
- Dependencies are also defined in package.json.

19.What do I need to install for making my React application?

- **create-react-app** | Create React App is a comfortable environment for learning React, and is the best way to start building a new single-page application in React. It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production.

Open the command prompt and type

npm install create-react-app

Let us create our first react application?

npx create-react-app first-app

[The name of your project should be in lower case]

npx is a package runner tool that comes with npm 5.2+.

cd first-app

npm start

It starts the node js server on the port number 3000

http://localhost:3000/

react-scripts start

This runs a predefined command specified in the "start" property of a package's "scripts" object. If the "scripts" object does not define a "start" property, npm will run node server. js .

VS code editor is the IDE to be sed to work on a React Application.

first-app

public

index.html

src

index.js

App.js

App.css

....

20. What is meant by a single-page application?

- It means that the application will not refresh and reload itself completely when some event is triggered. Only certain portions of the web page will be updated.

- A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages.

- Multi page application: The entire web page gets reloaded with each event triggered.

21. What is the Virtual DOM?

- The virtual DOM (VDOM) is a programming concept where an ideal, or "virtual", representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM. This process is called reconciliation.

```
<div id="root"></div>
```

- **Struture of a React Application**

- index.html
- index.js
- App.js

- **ES6 :-**

- ES6 (ECMASCript 2015)| ES6 stands for ECMAScript 6. ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

Things about JavaScript you should be comfortable with before learning React are: ES6 classes.

- The concepts we will explore in ES6 are

a) let,const keyword

b) default parameters

c) destructuring

d) map() method

e) classes in ES6

f) inheritance

g) modules

- Prior to ES6, we could only define a variable in JS was using var keyword.
- The variable declared using var keyword is having function scope. The same name variable can be defined multiple times.

```
//Ex of const keyword
const abc=[1,2,3,6,4];
console.log(abc);
abc[2]=5;
//abc=[5,7,8];
console.log(abc);

//Ex of let keyword
let abc=[1,2,3,6,4];
console.log(abc);
abc[2]=5;
//a variable defined using let keyword can
//work in block scope
{
        let abc=[5,7,8];
        console.log(abc);
}
console.log(abc);

//Ex of default parameters
function sayHello(name='JF036',num=3){
        console.log(name+" says 'hello' "
```

```
                                +num+" times");
//template literal (React | interpolation)
            console.log(`${name} says 'hello'
 ${num} times`);
}

sayHello('Amit');
sayHello();

//Ex of destructuring
let arr=[1,2,3,5];
//the line below
let [a,b,c,d]=arr;
console.log(`${a}
${b}
${c}
${d}`);

//when applying destructuring on an object, the variable names should match the property names
let person={name:'soma',city:'Kolkata'}
let {name,city}=person;
console.log(name);
console.log(city);

/* https://www.w3schools.com/react/react_es6_destructuring.asp
*/

Ex of map() method
let arr=[16,25,36,54];
```

```
let res=arr.map(Math.sqrt);
console.log(res);
console.log(arr);
res=arr.map((k) => (k/3).toFixed(2));//arrow function
console.log(res);

//array of objects
let users=[
        {id:101,
        fname:'Tanuja',
        lname:'Kanekar'
        },
        {id:102,
        fname:'Durga Shankar',
        lname:'Pandey'
        }
]
res=users.map((u) => `${u.fname} ${u.lname}`);
console.log(res);

/*https://www.w3schools.com/jsref/jsref_map.asp*/

Ex of class in ES6
class Book{

        print(){
console.log(`${this.name} has ${this.pages} pages. `);
        }
```

```
        constructor(name='Complete Javascript',pages=189){
                this.name=name;
                this.pages=pages;
        }
}

let book=new Book();
book.print();
book=new Book('HeadFirst jQuery',216);
book.print();

/* https://www.w3schools.com/js/js_classes.asp */
```

Ex of inheritance

Please try the example given in the below link

https://www.w3schools.com/js/js_class_inheritance.asp

- **Ex of modules**
  - A module is nothing more than a chunk of JavaScript code written in a file. We can export and import such modules

The **package.json** file is kind of a manifest for your project.

To create such a file, we will write the command in cmd

npm init

Open the json file created and add

```
        "type":"module"
```

```
//app.js
export class Book{
        print(){
```

```
console.log(`${this.name} has ${this.pages} pages. `);
        }

        constructor(name='Complete Javascript',pages=189){
                this.name=name;
                this.pages=pages;
        }
}

/* named export */
//app.js
export class Book{}

//myscript.js
import {Book} from './app.js'
let book=new Book();
book.print();

/* default export */
//app.js
export default class Book{}

//myscript.js
import myBook from './app.js'
let book=new myBook();
book.print();
book=new myBook('HeadFirst jQuery',216);
book.print();
```

FYR: https://medium.com/@etherealm/named-export-vs-default-export-in-es6-affb483a0910

- **Concepts of React**

---

npm install create-react-app

npx create-react-app first-app

cd first-app

npm start

- **Project structure**
  - **public/index.html**| is the main HTML file of our app that includes your React code and provides a context for React to render to..

  - **src/index.js** |This is the javascript file corresponding to index.html. This file has the following line of code which is very significant. ReactDOM.render(<App />, document.getElementById('root'));

```
//code snippet
import React from 'react';
import ReactDOM from 'react-dom';

//const elem=<><h2> Hello Everyone !!!</h2></>;
const elem=React.createElement("h2",{Style:'color:green'},"good evening");
ReactDOM.render(
  elem ,
  document.getElementById('my-root')
);
```

- The react package contains only the functionality necessary to define React components.
- The react-dom package serves as the entry point to the DOM and renders for React.

- **React.createElement()** | Create and return a new React element of the given type.

```
const elem=React.createElement("h2",{Style:'color:green'},"good evening");
```

Use JSX instead

```
const elem=<h2>Good Evening</h2>
```

```
const elem=<div>
  <h2>Good Evening</h2>
  <p>Have a great learning ahead</p>
</div>
```

22. **What is JSX?**
    - Extension of Javascript
    - It is recommended using it with React to describe what the UI should look like.
    - JSX is a JavaScript Extension Syntax used in React to easily write HTML and JavaScript together. This is simple JSX code in React. But the browser does not understand this JSX because it's not valid JavaScript code.

```
ReactDOM.render(
  elem ,
  document.getElementById('my-root')
)
```

- The **ReactDOM.render()** function takes two arguments, HTML code and an HTML element. The purpose of the function is to display the specified HTML code inside the specified HTML element.

//index.html

```
<div id="my-root"></div>
```

FYR: https://www.freecodecamp.org/news/quick-guide-to-understanding-and-creating-reactjs-apps-8457ee8f7123/

- **React** is based on Components.
- **Components** are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML. Components come in two types, Class components and Function components

Funtion component

//App.js

```
import './App.css';

function App(props) {
  return (
  <>
  <h2> We are learning {props.topic}. </h2>
  </>
  );
}

export default App;
```

//index.js

```
import {App} from './App';

ReactDOM.render(
  <App topic="React concepts"/> ,
```

```
  document.getElementById('my-root')
);
```

- **valid React component** because it accepts a single "props" (which stands for properties) object argument with data and returns a React element. We call such components "function components" because they are literally JavaScript functions.
- To add custom components in react, right click the src folder and add new folder and within the folder created add new file with .js extension.
- Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

- **Components**

a) function component

```
//MyComp.js
function MyComp(props){
   return(<div>
      <h2>Example of Function Component</h2>
      <h3>Hello {props.name}</h3>
      <ul>
         {props.nums.map((k)=><li>{k}</li>)}
      </ul>
   </div>);
}
export default MyComp

//index.js
import MyComp from './components/MyComp'

const arr=['abc','bca','cdc','bba'];
ReactDOM.render(
```

```
   <MyComp name="ADM21JF036" nums={arr}/> ,
 document.getElementById('my-root')
);
```

b) ES6 class component

```
//ClasComp.js
import React from 'react'
export default class ClassComp
            extends React.Component{
    render(){
       return(<div>
          <h1>This is class component</h1>
         <h2> Site for reference: {this.props.site}</h2>
          </div>)
    }
}
```

```
//MyComp.js
import ClassComp from "./ClassComp";
<ClassComp site="reactjs.org"/>
```

This is another approach to create a class component

```
import React,{Component} from 'react'
export default class ClassComp
            extends Component{

}
```

- "A React component should use props to store information that can be changed, but can only be changed by a different component. A React component should use state to store information that the component itself can change.
- The React philosophy is that props should be immutable and top-down. This means that a parent can send whatever prop values it likes to a child, but the child cannot modify its own props."

23. What is meant by State in a class component?

- State is similar to props, but it is private and fully controlled by the component.

```
constructor(props){
        super(props);//1
        this.state={
                site:this.props.site
        }
}
```

- Line 1: Class components should always call the base constructor with props.

- The **componentDidMount()** method runs after the component output has been rendered to the DOM.

Example:

```
//MyComp.js
<ClassComp fname="Samuel" lname="Gomes" scoreObtd="472"/>

//ClassComp.js
const findPercentage=(score) => ((score/5)).toFixed(2)+"%"

export default class ClassComp
        extends Component{
```

```
  constructor(props){
     super(props);
     this.state={
        fname:this.props.fname,
        lname:this.props.lname,
        score:this.props.scoreObtd
     }
  }

   render(){
      return(<div>
        <h2>{this.state.fname} {this.state.lname}</h2>
         <h3>You have scored: {this.state.score} </h3>
<h3>Percentage obtd: {findPercentage(this.state.score)}</h3>
     </div>)
   }
}
```

#HTML

```
<h3 style="color:red">
<div class="show"></div>
<img src="pic1.jpg"/>
<button onclick="f1">Click</button>
```

How to add inline CSS in React?

```
<h3 style={{color:'green'}}> //REACT
```

#styler.css

```
.show{
```

```
    color:red;
}
```

How to add external CSS in React?

```
<div className="show"></div>
```

24.How to add images in React?

We need to store the image within the src folder.

```
//ClassComp.js
import img1 from '../images/img.jpg'
<img src={img1} style={{width:'80px',height:'80px'}}
```

25.How to handle events in React?

```
<button onClick={this.f1}>Click</button>
```

In the class constructor,we add

```
this.f1=this.f1.bind(this);
this.state={
    message:'some message';
}
```

Add the method in the class Component

```
f1(){
    this.setState({message:'new message'})
}
```

26.How to accept dynamic input from user?

```
import React, {Component} from 'react'

class HomeComp extends Component{
```

```
    constructor(props){
        super(props);
        this.state={
            line1:'Write Something'
        }
        this.f1=this.f1.bind(this);
    }
    f1(e){
        if(e.target.value.length!=0)
            this.setState({line1:e.target.value})
        else
            this.setState({line1:'no wish mentioned'})
    }
    render(){
        return(<>
        <h3> Your Wish: {this.state.line1} </h3>
        Enter your wish: <input type="text" onChange={this.f1}/>

        </>)
    }
}

export default HomeComp
```

- **Components**

  - class component

  - function component

- **Props** is a way of passing data from one component to another component.

```
//function component
    Hello {props.name}
//class component
    Hello {this.props.name}
<ClassComp name="Preety"/>
```

- **State** is a private variable of a class component which can be altered/modified.

```
class ClassComp extends Component{
    constructor(props){
        super(props);
        this.state={
            name:this.props.name
        }
    }
    render(){
        return(<><h2> JSX </h2></>);
    }
}
```

- **Handle events**

```
class ClassComp extends Component{
    constructor(props){
        super(props);
        this.state={
```

```
                name:this.props.name,
                line1:'Write something'
        }
    this.handleClick=this.handleClick.bind(this);
    this.f1=this.f1.bind(this);
    }
    render(){
        return(<>
        <h2> JSX </h2>
    <button onClick={this.handleClick}>Click</button>
    Enter your wish:
    <input type="text" onChange={this.f1}/>
                </>);
    }

    handleClick(){
        this.setState({name:'Welcome to class'});
    }

    f1(e){
        this.setState({line1:e.target.value});
    }
}
```

- **React Lifecycle events**

- The **componentDidMount()** method runs after the component output has been rendered to the DOM.
- Mounting means putting elements into the DOM.

- React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

- The **render()** method is required and will always be called, the others are optional and will be called if you define them.

FYR: https://www.w3schools.com/react/react_lifecycle.asp

27.How to handle multiple inputs with the same event handler?

- Ensure the value for the name attribute of HTML input elements MUST BE IDENTICAL TO the state variable name.

```
class ClassComp2 extends Component{
   constructor(props){
      super(props);
      this.state={name:'constructor called',firstName:'',lastName:''}
      this.handleChange=this.handleChange.bind(this);
   }

   componentDidMount(){
      setTimeout(() =>{
         this.setState({name:'componentDidMount called'})
      },5000)
   }

/* single handler: multiple inputs */
```

```
  handleChange(e){
    this.setState({[e.target.name]:e.target.value})
  }

  render(){
    return(
      <>
        <p>{this.state.name}</p>
        <h2>Hello {this.state.firstName} {this.state.lastName}</h2>
         <div>
       First Name:<input type="text" name="firstName" onChange={this.handleChange}/>
       Last Name:<input type="text"  name="lastName" onChange={this.handleChange}/>
        </div>

      </>
    )
  }

}

export default ClassComp2
```

28. How to add conditional rendering in your React App?

- This can help you conditionally render a part of the component while the rest of the output doesn't change.

```
this.state={name:'constructor called',firstName:'',lastName:'',show:true}

//render()
<p>
```

```
   <button style={{color:'white',backgroundColor:'green'}}
 onClick={() =>this.setState({show: !this.state.show})}>Toggle Screen</button>
     {this.state.show?<ClassComp fname='Sachin' lname='Tendulkar' scoreObtd='455'/>
                :null}

 </p>
```

FYR: https://reactjs.org/docs/conditional-rendering.html

- **List and Keys in React**

```
//Cart.js
export default function Cart(props){
   const listItems=props.items.map(
      (item) =><li key={item.pcode}>{item.pname} {item.price}</li>
   )
   return(
      <>
         {listItems}
      </>
   )
}

//ClassComp2.js
 products:[
         {pcode:1,pname:'apple',price:50},
         {pcode:2,pname:'mango',price:65},
         {pcode:3,pname:'orange',price:45}
      ]
 <h2>Summer Fruits</h2>
```

```
<ul>
    <Cart items={this.state.products}/>
</ul>
```

- **React Forms**

- **How to add bootstrap in the React application?**

Copy the CDN links from the below site:

https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp

and paste the scripts appropriately in the <body> section just before closing </body> tag and the CSS link in the <head> section of the public/index.html page in your React Application

FYR:

https://blog.logrocket.com/how-to-use-bootstrap-with-react-a354715d1121/

```
<div className="container">
</div>
```

className="" instead of class="" unlike HTML

```
//UserForm.js
import React, {Component} from 'react'
export default class UserForm extends Component
{
  constructor(props) {
    super(props)
    this.state = {
      id:1,name:'',complaint:''
```

```
    }
    this.handleForm=this.handleForm.bind(this);
    this.handleChange=this.handleChange.bind(this);
}

handleChange(e){
    this.setState({[e.target.name]:e.target.value});
}

 render(){
                    return(<>
    <table className="table table-bordered table-stripped table-dark"><tbody>
        <tr>
          <td>Name:</td>
          <td>
            <input type="text" name="name" onChange={this.handleChange}></input>
          </td>
        </tr>
        <tr>
          <td>Complaint/Query:</td>
          <td>
            <input type="text" name="complaint" onChange={this.handleChange}></input>
          </td>
        </tr>
        <tr>
          <td></td>
          <td>
            <button onClick={this.handleForm}>Submit</button>
          </td>
```

```
        </tr>
        </tbody></table></>);
  }

handleForm(e)
  {
   if(this.state.name!=='' && this.state.complaint!==''){
        this.setState(prevState=>({
                id:prevState.id+1
         }));
        alert(`Thanks!! ${this.state.name}\r\nYour Complaint was Submitted \r\nTransaction ID is ${this.state.id}`);
      }else{
           alert('Fields missing');
      }
      e.preventDefault();
  }

}

//RegisterComp.js
import React, {Component} from 'react'
const validEmailRegex = new RegExp('[a-z0-9]+@[a-z]+.[a-z]{2,3}');

const validateForm = errors => {
  let valid = true;
    Object.values(errors)
       .forEach(val => val.length > 0 && (valid = false));
    return valid;
};
```

```
export default class RegisterComp extends Component
{
            constructor(props) {
             super(props);
       this.state = {
          fullName: null,
          email: null,
          password: null,
          errors: {
            fullName: '',
            email: '',
            password: ''
          },
          go:false
       }
       this.handleChange=this.handleChange.bind(this);
       this.handleSubmit=this.handleSubmit.bind(this);
   }

  handleChange(event){
       event.preventDefault();
           //object destructuring ES6
       const { name, value } = event.target;
       let errors = this.state.errors;

       switch (name) {
          case 'fullName':
            errors.fullName =
```

```
          value.length < 5
            ? 'Full Name must be at least 5 characters long!'
            : '';
        break;
      case 'email':
        errors.email =
          validEmailRegex.test(value)
            ? ''
            : 'Email is not valid!';
        break;
        case 'password':
        errors.password =
          value.length < 8
            ? 'Password must be at least 8 characters long!'
            : '';
        break;
      default:
        break;
    }

  this.setState({errors, [name]: value});
}

        render(){
        const {errors} = this.state;
  return(<>
  <form noValidate><table><tbody>
    <tr>
     <td>Name:</td>
```

```
          <td>
            <input type="text" name="fullName" onChange={this.handleChange} noValidate />
          </td>
          <td>{errors.fullName.length > 0 &&
            <span className='error'>{errors.fullName}</span>}
          </td>
        </tr>
        <tr>
          <td>Email:</td>
          <td>
            <input type="email" name="email"onChange={this.handleChange} noValidate />
          </td>
          <td> {errors.email.length > 0 &&
            <span className='error'>{errors.email}</span>}
          </td>
        </tr>
        <tr>
          <td>Password:</td>
          <td>
            <input type="password" name="password"onChange={this.handleChange}
noValidate />
          </td>
          <td>{errors.password.length > 0 &&
            <span className='error'>{errors.password}</span>}
          </td>
        </tr>
        <tr>
          <td></td>
          <td>
            <button onClick={this.handleSubmit}>Register</button>
```

```
            </td>
           </tr>
           </tbody>
          </table>
          </form>
          </>);


            }


 handleSubmit(event){
       event.preventDefault();
       if(this.state.fullName !== null && this.state.email !== null && this.state.password !==
null){
          this.setState({go:true})
       }
       if(validateForm(this.state.errors) && this.state.go===true) {
         console.info('Valid Form')
         alert('Form submitted');
       }else{
         console.error('Invalid Form')
         alert('Form Incomplete');
       }
    }

}



handleChange(e){
            e.preventDefault();
            //more code
```

}

Why do we write e.preventDefault();

FYR: https://reactjs.org/docs/handling-events.html

29. Why write "novalidate"?

- The novalidate attribute is a boolean attribute. When present, it specifies that the form-data (input) should not be validated when submitted.

FYR: https://www.educative.io/edpresso/react-form-validation

- **React Application :**

- React Application integrating with the Web service in the backend.
- React application will not have any database layer. It just exposes the UI.

30. How can react consume the json data from a web service?

- There are several ways like Fetch API, Axios
- The Fetch API through the fetch() method allows us to make an HTTP request to the backend. With this method, we can perform different types of operations using HTTP methods like the GET method to request data from an endpoint, POST to send data to an endpoint, and more.

- You should populate data with AJAX calls in the componentDidMount lifecycle method. This is so you can use setState to update your component when the data is retrieved.

Add a new component to your react project.

```
import React,{Component} from 'react'

export default class CompOne extends Component{

    constructor(props){
        super(props);
        this.state = {
            data: [],
```

```
                    name:'',
                    companyName:'' ,
                    website:'',
                    email:''
                }
          this.handleChange=this.handleChange.bind(this);
          this.save=this.save.bind(this);
        }

 handleChange(e){
    this.setState({[e.target.name]:e.target.value})
 }

componentDidMount() {
   fetch('https://jsonplaceholder.typicode.com/users').then(
      (response) => response.json()
   ).then(data => this.setState({data: data}))
}

        render(){
                return(<>
                <h3>Available API users data</h3>
                {this.state.data.map(d=>
                <details key={d.id}>
                <summary >{d.name}</summary>
        <span>Email: {d.email}</span> <br/>
        <span>Website: {d.website}</span>
                <span>Company Name:{d.company.name}</span>
                </details>
```

```
  )}


                    <h3>Add new API user data</h3>


 <span>{this.state.name} {this.state.companyName} {this.state.email}</span>
     <br/>
 Name: <input type="text" name="name" onChange={this.handleChange}/><br/>
 Company: <input type="text" name="companyName"
onChange={this.handleChange}/><br/>
 Email: <input type="text" name="email" onChange={this.handleChange}/><br/>
   <br/><button onClick={this.save}>Save</button>


                  </>);
         }


save(e){
      e.preventDefault();
      let userRef={
        id: 11,
        name:this.state.name,
        email:this.state.email,
        company: {
          name: this.state.companyName
        }
      }
    fetch('https://jsonplaceholder.typicode.com/users', {
      method: 'POST',
      body: JSON.stringify(userRef),
   headers: {
```

```
    'Content-type': 'application/json; charset=UTF-8',
    },
})
  .then((response) => response.json())
  .then((json) => console.log(json));
    }



}
```

App.js/index.js

<CompOne/>

HTTP VERBS

GET : fetch one or more rows

POST: create new resource (row)

PUT: update a resource

DELETE: remove a resource

FYR:https://jsonplaceholder.typicode.com/guide/

** React-Routing